
GeoNode Documentation

Release 2.8

GeoNode Development Team

Feb 26, 2020

Contents

1	First Steps	3
2	How To Use The Documentation	5
3	Table of contents	7
3.1	About GeoNode	7
3.1.1	About GeoNode	7
3.1.1.1	Online demo	7
3.1.1.2	Geospatial data storage	8
3.1.1.3	Data mixing, maps creation	8
3.1.1.4	GeoNode as a building block	9
3.1.1.5	Convinced! Where do I sign?	9
3.2	Tutorials	9
3.2.1	Tutorials	9
3.2.1.1	GeoNode Overview & Reference	9
3.2.1.1.1	Users' Features	9
3.2.1.1.2	Introduction	10
3.2.1.1.3	Reference Doc	23
3.2.1.2	Installation & Admin	31
3.2.1.2.1	Quick Installation Guide	32
3.2.1.2.2	Linux Admin Intro	38
3.2.1.2.3	Running Ansible scripts	47
3.2.1.2.4	GeoNode (v2.8) on Docker	52
3.2.1.2.5	VM Setup with VirtualBox	63
3.2.1.2.6	Running a VM with Vagrant	83
3.2.1.2.7	GeoNode (v2.8) installation on Ubuntu 16.04	85
3.2.1.2.8	GeoNode (v2.8) installation on CentOS 7	116
3.2.1.2.9	Network configuration issues	137
3.2.1.2.10	Windows Binary Installer	138
3.2.1.2.11	GeoNode (v2.8) update from older versions	162
3.2.1.3	Users Workshop	205
3.2.1.3.1	Accounts and users	205
3.2.1.3.2	Document Types	213
3.2.1.3.3	Searching	216
3.2.1.3.4	Managing layers	217
3.2.1.3.5	Edit Layer Style	234
3.2.1.3.6	Managing maps	235

3.2.1.3.7	Using GeoNode with other applications	255
3.2.1.4	Administrators Workshop	293
3.2.1.4.1	GeoNode and GeoServer Advanced Security	294
3.2.1.4.2	Usage of the GeoNode's Django Administration Panel	336
3.2.1.4.3	Management Commands for GeoNode	344
3.2.1.4.4	Configuring Alternate CSW Backends	348
3.2.1.4.5	LDAP configuration	350
3.2.1.4.6	Customize the look and feel	351
3.2.1.4.7	Debugging GeoNode Installations	366
3.2.1.4.8	Changing the Default Language	369
3.2.1.4.9	More on Security and Permissions	370
3.2.1.4.10	Loading Data into a GeoNode	379
3.2.1.4.11	Implementing S3 Bucket for Static and Media Files	395
3.2.1.4.12	Backup & Restore GeoNode - Data Migration	397
3.2.1.4.13	GeoNode Monitoring	411
3.2.1.4.14	Use datastore shards in GeoNode	430
3.2.1.4.15	Asynchronous signals handling	432
3.2.1.4.16	GeoNode Social Accounts	434
3.2.1.5	Developers Workshop	454
3.2.1.5.1	Introduction to GeoNode development	454
3.2.1.5.2	Django Overview	460
3.2.1.5.3	Development Prerequisites and Core Modules	471
3.2.1.5.4	Install GeoNode for Development	478
3.2.1.5.5	Start working on Geonode the next day after install	484
3.2.1.5.6	GeoNode debugging techniques	485
3.2.1.5.7	GeoNode APIs	493
3.2.1.5.8	Testing in GeoNode	564
3.2.1.5.9	Pavement.py and Paver	566
3.2.1.5.10	Introduction to GeoNode Projects	569
3.2.1.5.11	Make a GeoNode release	569
3.2.1.6	Advanced Workshop	572
3.2.1.6.1	Advanced Data Management and Processing	572
3.2.1.6.2	GeoNode Advanced Configuration	841
3.2.1.6.3	GeoNode on Production	860
3.3	Reference	946
3.3.1	Reference documentation	947
3.3.1.1	Security and Permissions	947
3.3.1.1.1	Permissions and GeoNode objects	948
3.3.1.1.2	Publishing and unpublishing objects	952
3.3.1.2	GeoNode ad-hoc API	952
3.3.1.2.1	API endpoints	953
3.3.1.2.2	API filtering	953
3.3.1.2.3	API limit and pagination	953
3.3.1.2.4	API settings	953
3.3.1.2.5	Searching with Haystack	953
3.3.1.3	Localization	954
3.3.1.4	Developers Reference	955
3.3.1.4.1	GeoNode Django Apps	955
3.3.1.4.2	JavaScript in GeoNode	962
3.3.1.4.3	Settings	963
3.3.1.4.4	GeoSites: GeoNode Multi-Tenancy	978
3.3.1.5	Supported Browsers	980
3.3.1.5.1	Internet Explorer	980
3.3.1.5.2	Testing on Internet Explorer	981

3.3.1.6	WorldMap	981
3.3.1.6.1	Installation	982
3.3.1.6.2	Hypermap Registry	985
3.3.1.6.3	Test the stack	985
3.3.1.6.4	Schedule Celery tasks	986
3.4	Organizational	987
3.4.1	Organizational	987
3.4.1.1	Project Information	987
3.4.1.2	Contributing	987
3.4.1.3	Table of Content	987
3.4.1.3.1	Roadmap Process	987
3.4.1.3.2	Community Resources	988
3.4.1.3.3	Community Bylaws	990
3.4.1.3.4	Contributing to GeoNode	992
3.4.1.3.5	GeoNode Patch Review Process	1003
3.4.1.3.6	Patch Review criteria	1004
3.4.1.3.7	How to contribute to GeoNode's translation	1006
3.4.1.3.8	How to contribute to GeoNode's Documentation	1010
3.4.1.3.9	How to write Documentation	1015
3.4.1.3.10	How to Translate the Documentation	1020
4	Need Help?	1021
	Python Module Index	1023
	Index	1025

Welcome to GeoNode's Documentation.

GeoNode is an Open Source, Content Management System (CMS) for geospatial data. It is a web-based application and platform for developing geospatial information systems (GIS) and for deploying spatial data infrastructures (SDI).

CHAPTER 1

First Steps

The following sections give an overview of GeoNode from different perspectives, they are targeted at a non-technical audience and the quick installation guide at people who just want to get it installed and will come back later to the complete documentation.

- *About GeoNode*
- *Quick Installation Guide*
- *GeoNode Users Quickstart Manual*

How To Use The Documentation

The documentation is geared toward three distinct types of users:

1. **Users:** Are people who log into a GeoNode website and use its functionality.
2. **Administrators:** Are people who install and deploy GeoNode websites in production for their Users.
3. **Developers:** Are people who write code to add functionality, integrate with other systems, fix bugs, and potentially help an Administrator setup a server and deploy a GeoNode instance for production.

The documentation is divided into three sections:

1. *Tutorials:* Step-by-step instructions in workshop format that help a user to accomplish a set of tasks.
2. *Reference:* Architecture, component information, API descriptions etc.
3. *Organizational:* About the project, how to contribute, links, resources, other info.

3.1 About GeoNode

3.1.1 About GeoNode



GeoNode is a geospatial content management system, a platform for the management and publication of geospatial data. It brings together mature and stable open-source software projects under a consistent and easy-to-use interface allowing non-specialized users to share data and create interactive maps.



Data management tools built into GeoNode allow for integrated creation of data, metadata, and map visualizations. Each dataset in the system can be shared publicly or restricted to allow access to only specific users. Social features like user profiles and commenting and rating systems allow for the development of communities around each platform to facilitate the use, management, and quality control of the data the GeoNode instance contains.

It is also designed to be a flexible platform that software developers can extend, modify or integrate against to meet requirements in their own applications.

3.1.1.1 Online demo


A live demo of the latest stable build is available at demo.geonode.org.

Anyone may sign up for a user account, upload and style data, create and share maps, and change permissions. Since it is a demo site, we don't make any guarantee that your data and maps will always be there. But it should hopefully allow you to easily preview the capabilities of GeoNode.

 Layers Maps Documents People Groups  Sign in

Welcome


GeoNode is an open source platform for sharing geospatial data and maps.
If you have any questions about the software or service, join our [mailing list](#).
Need help? [Getting Started](#)



33 Layers

Click to search for geospatial data published by other users, organizations and public sources.
Download data in standard formats.


[Explore layers »](#)



22 Maps

Data is available for browsing, aggregating and styling to generate maps which can be shared publicly or restricted to specific users only.

[Explore maps »](#)




10 Users

GeoNode allows registered users to easily upload geospatial data in several formats including shapefile and GeoTiff.

[See users »](#)

Powered by [GeoNode version 2.4.1](#) | [Developers](#) | [About](#)

English 

3.1.1.2 Geospatial data storage

GeoNode allows the user to upload vector data (currently only shapefiles) and raster data in their original projections using a web form. Vector data is uploaded in ESRI shapefile format and satellite imagery and other kinds of raster data are uploaded as GeoTIFFs.

Special importance is given to standard metadata formats like ISO 19139:2007. After the upload is finished, the user is presented with a form to fill in the metadata and it is made available using a [CSW](#) interface. Users may also upload a metadata XML document (in ISO, FGDC, or Dublin Core format) to fill in key GeoNode metadata elements automatically.

Similarly, GeoNode provides a web based styler, that lets the user change how the data looks and preview the changes in real time.

3.1.1.3 Data mixing, maps creation

Once the data has been uploaded, GeoNode lets the user search for it geographically or via keywords and create maps.

All the layers are automatically re-projected to web mercator for maps display, making it possible to use different popular base layers, like Open Street Map, Google Satellite or Bing layers.

Once maps are saved, it is possible to embed them in any web page or get a PDF version for printing.

3.1.1.4 GeoNode as a building block

A handful of other Open Source projects extend GeoNode's functionality by tapping into the re-usability of Django applications. Visit our gallery to see how the community uses GeoNode: [GeoNode Projects](#).

The development community is very supportive of new projects and contributes ideas and guidance for newcomers.

3.1.1.5 Convinced! Where do I sign?

The next steps are to follow the *Quick Installation Guide*, *read the tutorials* and subscribe to the [geonode-users](#) and/or [geonode-devel](#) mailing lists to join the community. Thanks for your interest!

3.2 Tutorials

The *Tutorials* section contains step-by-step workshops that are oriented around performing particular sets of tasks, like adding data or publishing maps, setting up and maintaining a server, or setting up a project to extend from GeoNode. These tutorials are written in a workshop like format and are broken into three groups: Users, Administrators and Developers.

3.2.1 Tutorials

The tutorials are based around performing tasks, like adding data or publishing maps. The tutorials are written in a workshop like format and are broken into three groups *Users*, *Administrators* and *Developers*.

3.2.1.1 GeoNode Overview & Reference

Welcome to the GeoNode Training *Overview & Reference* documentation v2.8.

This module guides the user to an overview of GeoNode and its main components.

At the end of this section you will have a clear view of what GeoNode is and can do. You will be able also to use the GeoNode main functionalities and understand some of the basic concepts of the system infrastructure.

3.2.1.1.1 Users' Features

Open Source Geospatial Content Management System

GeoNode is a web-based application and platform for developing geospatial information systems (GIS) and for deploying spatial data infrastructures (SDI).

What GeoNode can be used for... [GeoNode Demo](#) (admin/admin)

Spatial Data Discovery

GeoNode allows users to browse and search for geospatial data. By combining collaboration found in social networks with specialized geospatial tools, GeoNode makes it easy to explore, process, style, and share maps and geospatial data. Spatial datasets can be imported and shared, all through a non-technical user interface.

Features include:

- Powerful spatial search engine

- Federated OGC services
- Metadata catalogue

Import and Manage

GeoNode allows users to upload and share geospatial data, securely. GeoNode makes it easy to upload and manage geospatial data on the web. Any user can upload and make content available via standard OGC protocols such as Web Map Service (WMS) and Web Feature Service (WFS). Data is available for browsing, searching, styling, and processing to generate maps which can be shared publicly or restricted to specific users only. Supported upload formats include shapefile, GeoTIFF, KML and CSV. In addition, it is possible to connect to existing external spatial databases and services.

Features include:

- Publish raster, vector, and tabular data
- Manage metadata and associated documents
- Securely or publicly share data
- Versioned geospatial data editor

Interactive Mapping

GeoNode allows users to create and share interactive web maps. GeoNode comes with helpful cartography tools for styling and composing maps graphically. These tools make it easy for anyone to assemble a web-based mapping application with functionality traditionally found in desktop GIS applications. Users can gain enhanced interactivity with GIS-specific tools such as querying and measuring.

Features include:

- GeoExplorer GIS client
- Graphical style editor
- Create multi-layer interactive maps
- Share and embed maps in web pages

3.2.1.1.2 Introduction

This section introduces the GeoNode GUI and functionalities through a step-by-step workshop.

At the end of this module the users will be familiar with the GeoNode default GUI and objects.

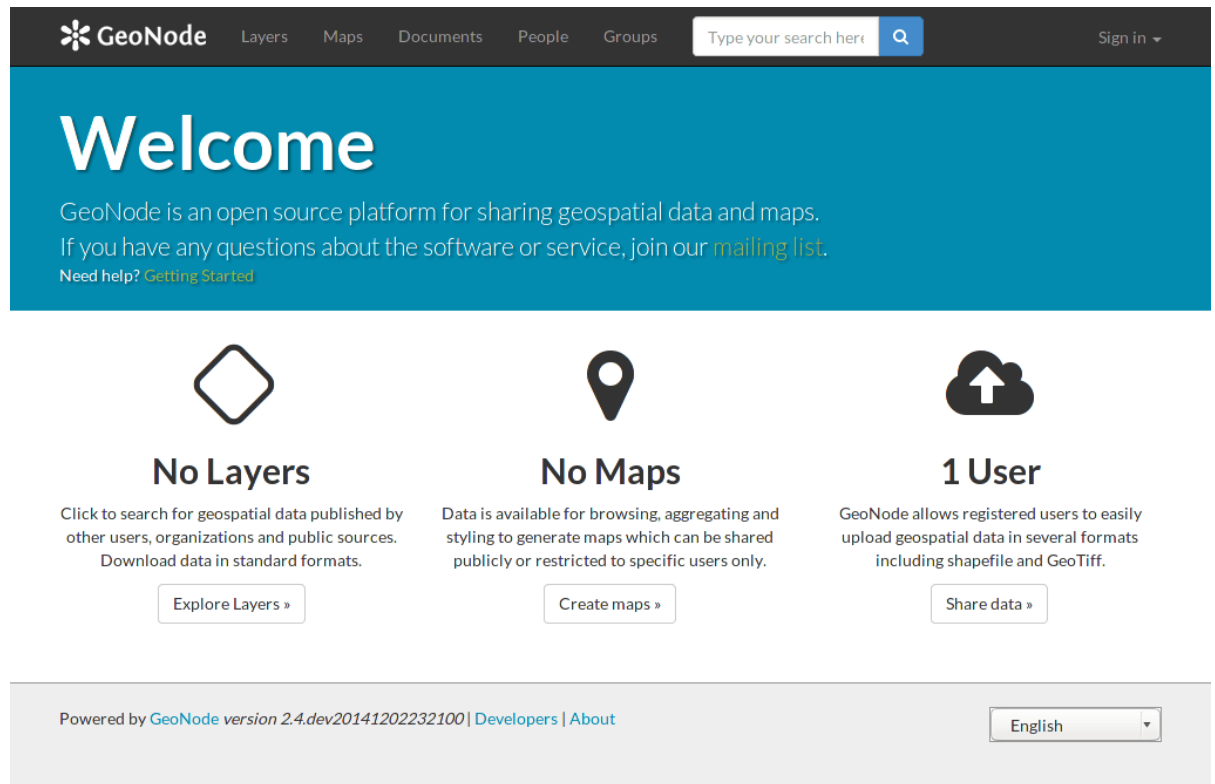
A Tour Of GeoNode

In order to get started, let's look at the GeoNode interface and get a feel for how to navigate around it.

The GeoNode web interface is the primary method of interacting with GeoNode as a user. From this interface, one can view and modify existing spatial layers and maps, as well as find information on other GeoNode users.

Without being logged in, you are limited to read-only access of public layers.

1. Navigate to your GeoNode instance (online demo available [here](#)):



The screenshot shows the GeoNode welcome page. At the top is a dark navigation bar with the GeoNode logo, links for Layers, Maps, Documents, People, and Groups, a search bar, and a Sign in link. Below this is a large blue banner with the word "Welcome" in large white text. Underneath, it says "GeoNode is an open source platform for sharing geospatial data and maps. If you have any questions about the software or service, join our [mailing list](#). Need help? [Getting Started](#)".

Below the banner are three white boxes with icons and text:

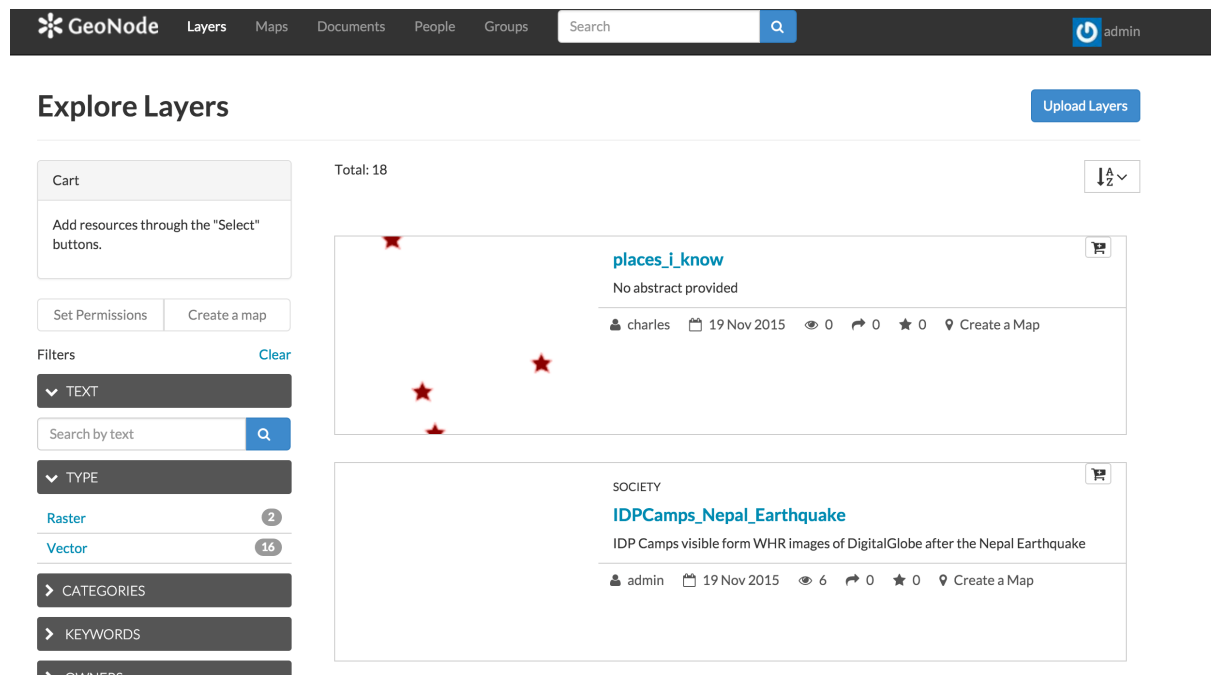
- No Layers**: Icon of a diamond. Text: "Click to search for geospatial data published by other users, organizations and public sources. Download data in standard formats." Button: "Explore Layers »".
- No Maps**: Icon of a location pin. Text: "Data is available for browsing, aggregating and styling to generate maps which can be shared publicly or restricted to specific users only." Button: "Create maps »".
- 1 User**: Icon of a cloud with an upload arrow. Text: "GeoNode allows registered users to easily upload geospatial data in several formats including shapefile and GeoTiff." Button: "Share data »".

At the bottom of the page, there is a footer bar with the text "Powered by GeoNode version 2.4.dev201411202232100 | [Developers](#) | [About](#)" and a language dropdown menu set to "English".

Welcome page

This page shows a variety of information about the current GeoNode instance. At the top of the page is a toolbar showing quick links to view [layers](#), [maps](#), documents (metadata), [people](#), and a search field. Below this is a listing of recently updated layers, including abstract, owner, rating, and download button (if available).

2. Click Explore button and choose Preview. Table data could be visualized as: Grid, Graph or Map.



The screenshot shows the "Explore Layers" page in GeoNode. The top navigation bar is the same as the welcome page. Below it, the page title "Explore Layers" is on the left, and an "Upload Layers" button is on the right. On the left side, there is a sidebar with a "Cart" section, "Set Permissions" and "Create a map" buttons, and a "Filters" section. The filters include "TEXT" (Search by text) and "TYPE" (Raster: 2, Vector: 16). There are also buttons for "CATEGORIES" and "KEYWORDS".

The main content area shows a map with red stars and a list of layers. The first layer is "places_i_know" by "charles", dated "19 Nov 2015", with 0 views, 0 likes, and 0 stars. The second layer is "IDPCamps_Nepal_Earthquake" by "admin", dated "19 Nov 2015", with 6 views, 0 likes, and 0 stars. Each layer entry includes a "Create a Map" button.

Explore Layers page

This page shows all layers known to GeoNode, available in either List or Grid viewing. Layers can be sorted by Most Recent, Most Popular, or Most Shared. Also available are a list of categories, with which layers can be connected with.

- Find a layer and click on its name.

Title	San Andres Y Providencia Coastline
Abstract	No abstract provided

Viewing a layer

- A layer viewing page will display, with the layer itself superimposed on a hosted base layer (in this case [MapQuest OpenStreetMap](#)). Explore this page, noting the various options available to you.
- Now click the Maps link in the tool bar to go to the Explore Maps page.

Explore Maps page

This page shows all maps known to GeoNode, available with similar viewing options as with the layers. Currently, there are no maps here, but we will create one later on in the workshop.

- Click the Search link in the toolbar to bring up the Search page.

Search page

This page contains a wealth of options for customizing a search for various information on this GeoNode instance. While a simple search box is available at the top of every page, this search form allows for much more fine-tuned searches.

Now that you are familiar with the basic interface, the next step is to create your own account so you manage some GeoNode resources of your own.

GeoNode Quickstart

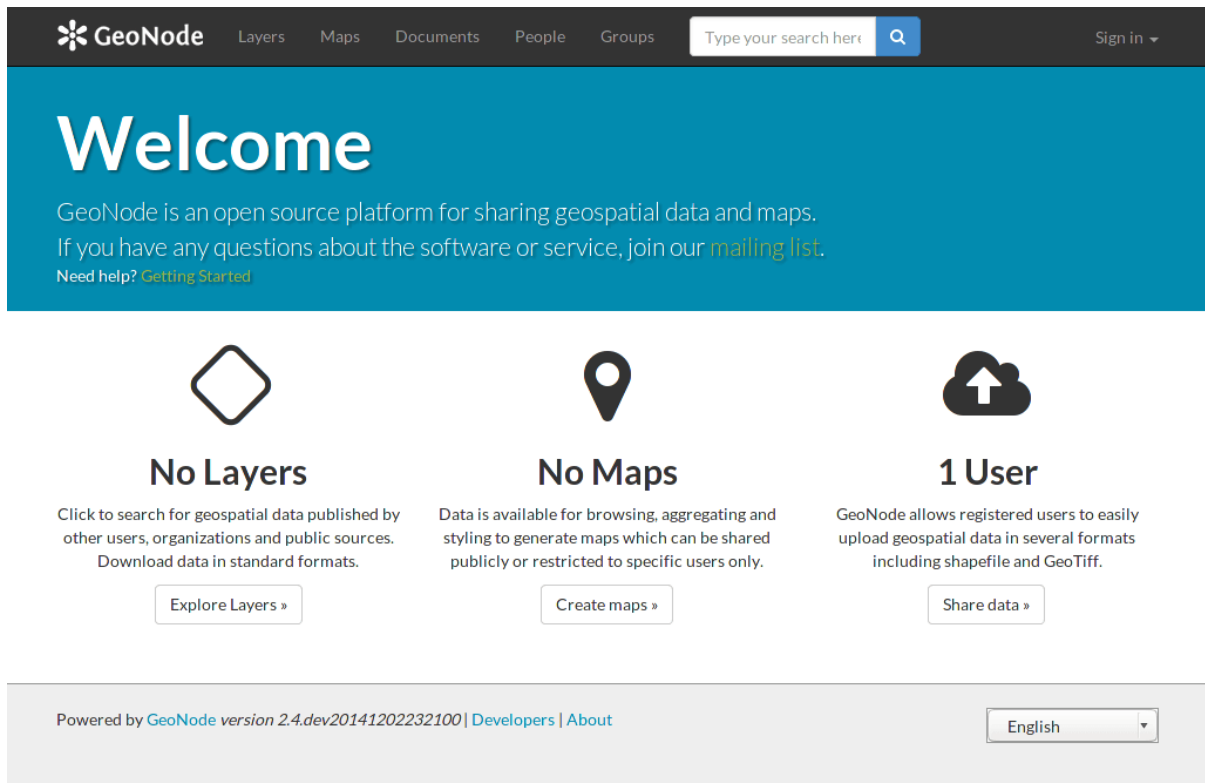
Open Source Geospatial Content Management System

GeoNode is a web-based application and platform for developing geospatial information systems (GIS) and for deploying spatial data infrastructures (SDI).

In this Quickstart guide you will learn the following:

1. How to register a new account to get started
2. How to add a new layer
3. How to create a map using your new layer
4. How to share your map with others

Start GeoNode on your Live DVD or local VM and redirect your browser at <http://localhost:8000/> (this is the default port). The page will look like shown in the image below.

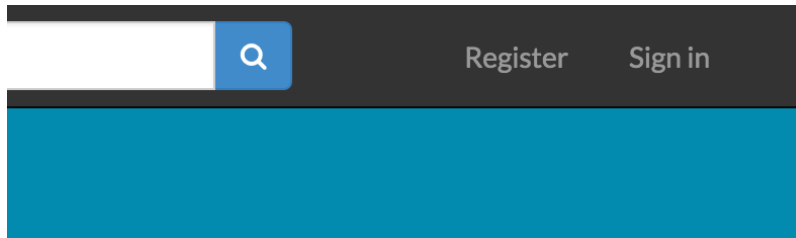


Welcome page

1. Register a new account

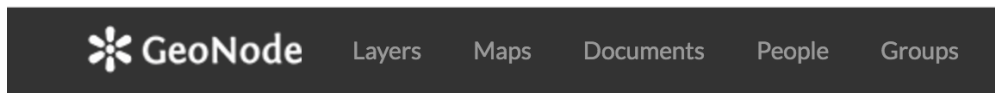
From the interface shown above, one can view and modify existing spatial layers and maps, as well as find information on other GeoNode users. But, without being logged in, you are limited to read-only access of public layers. In order to create a map and add layers to it, you have to have create an account first.

1. From any page in the web interface, you will see a *Sign in* link. Click that link, and in the dialog that displays, click the *Register now* link.



Sign in Form

2. On the next page, fill out the form. Enter a user name and password in the fields. Also, enter your email address for verification.



Sign up

Sign up

Username

Password

Password (again)

Email

Sign up

Register Form

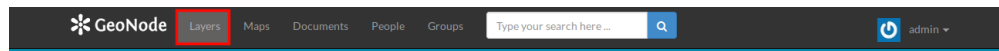
3. You will be returned to the welcome page. An email will be sent confirming that you have signed up. While you are now logged in, you will need to confirm your account. Navigate to the link that was sent in the email.
4. By clicking *Confirm* you will be returned to the homepage. Now you've registered an account, you are able to add layers to it as well as create maps and share those with other users.

Note: Depending on how the GeoNode you are using is configured, registration may not be available, or you may not be required to confirm your email address before logging in.

2. Add a new layer

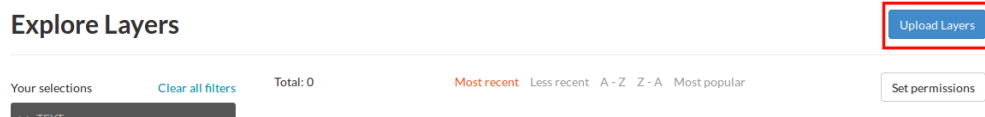
Layers are a published resource representing a raster or vector spatial data source. Layers also can be associated with metadata, ratings, and comments.

1. To add a layer to your account, navigate to the welcome page. There the following toolbar can be seen:



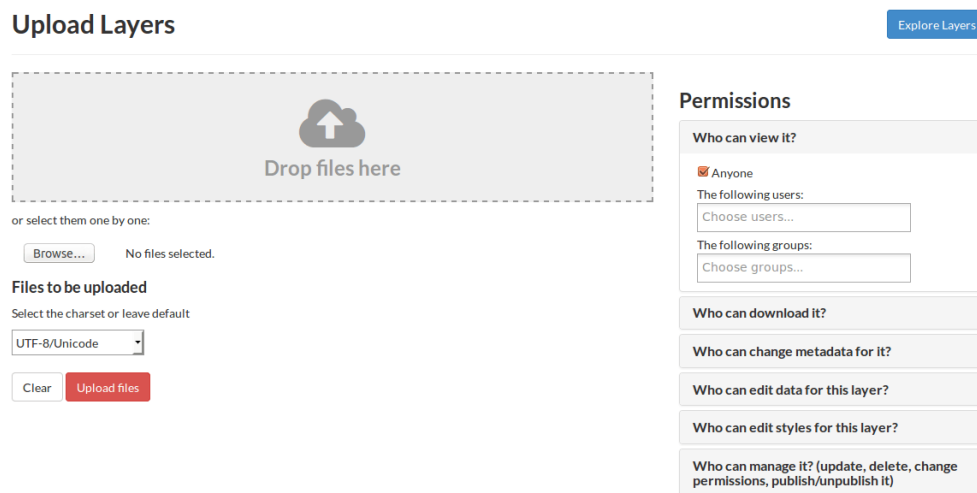
Toolbar

2. By clicking the *Layers* link you will be brought to the *Layers* menu where a new subtoolbar can be seen. This toolbar allows you to *Explore*, *Search* and *Upload* layers.



Upload Button

3. Now click *Upload Layers* and you'll see the upload form.



Upload Form

4. You have two possibilities to add your files. You can either do that by using *drag & drop* or you choose to *browse* them. Be aware that you have to upload a complete set of files, consisting of a **shp**, a **prj**, a **dbf** and a **shx** file. If one of them is missing, GeoNode will warn you before you upload them.
5. You should now be able to see all the files you want to upload.

streams

ESRI Shapefile

- streams.cst [Remove](#)
- streams.dbf [Remove](#)
- streams.prj [Remove](#)
- streams.shp [Remove](#)
- streams.shx [Remove](#)

Select the charset or leave default

UTF-8/Unicode

Clear

Upload files

Files to be Uploaded

6. GeoNode has the ability to restrict who can view, edit, and manage layers. On the right side of the page you can see the *Permission* section, where you can limit the access on your layer. Under **Who can view and download this data**, select **Any registered user**. This will ensure that *Anonymous* view access is disabled. In the same area, under **Who can edit this data**, select **your username**. This will ensure that *Only You* are able to edit the data in the layer.

Permissions

Who can view it?

☒ Anyone

The following users:

The following groups:

Who can download it?

Who can change metadata for it?

The following users:

The following groups:

Who can edit data for this layer?

Who can edit styles for this layer?

Who can manage it? (update, delete, change permissions, publish/unpublish it)

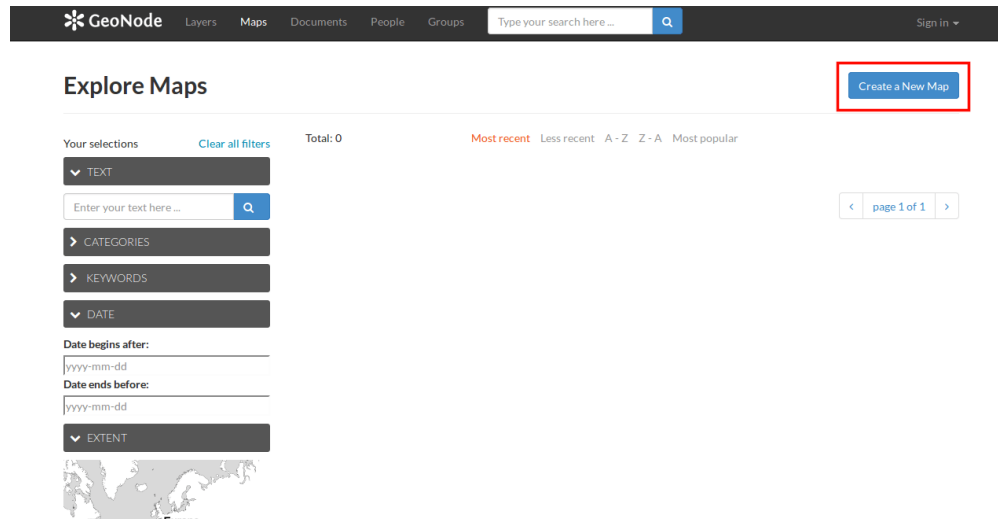
Permissions

7. To upload data, click the *Upload* button at the bottom.

3. Create a new map

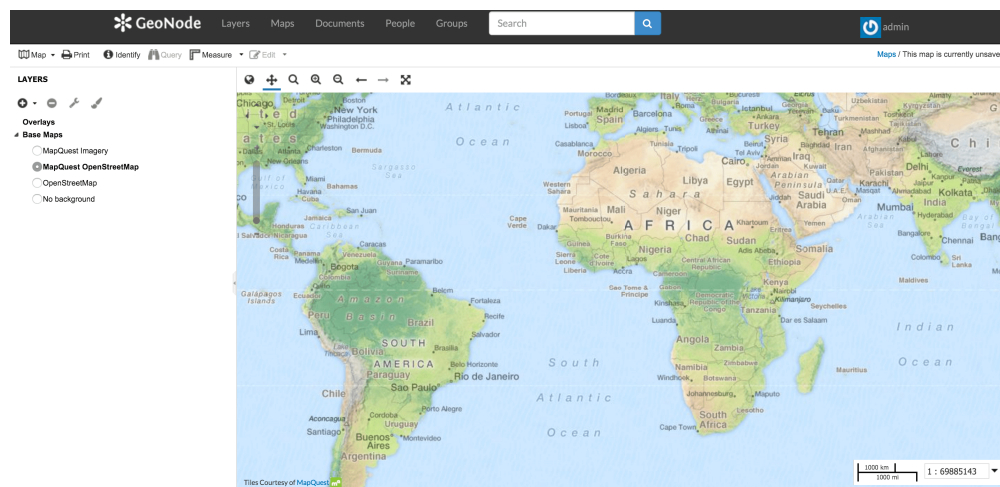
The next step for you is to create a map and add the newly created layers to this map.

1. Click the *Maps* link on the top toolbar. This will bring up the list of maps.



Create new Map Button

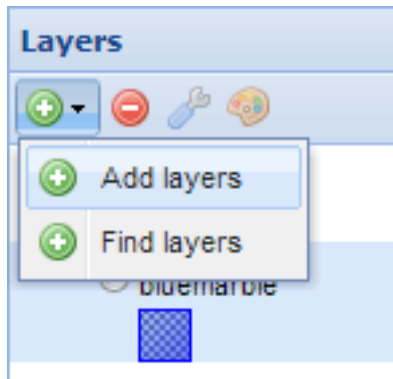
2. Currently, there aren't any maps here. To add one click the *Create a New Map* button and a map composition interface will display.



Maps Editor

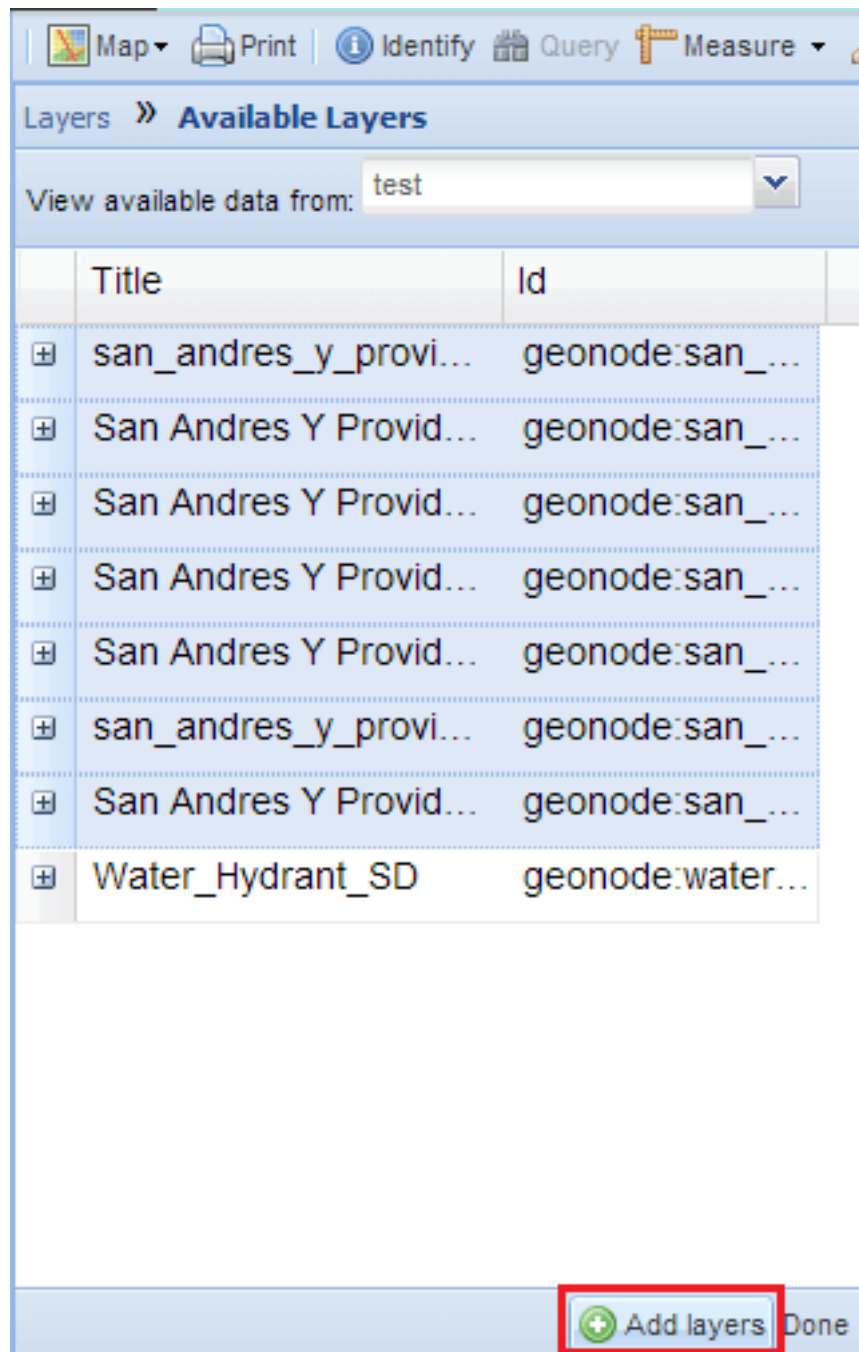
In this interface there is a toolbar, layer list, and map window. The map window contains the MapQuest OpenStreetMap layer by default. There are other service layers available here as well: Blue Marble, Bing Aerial With Labels, MapQuest, and OpenStreetMap.

3. Click on the *New Layers* button and select *Add Layers*.



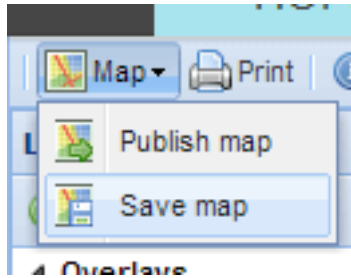
Add Layers

4. Now you should be able to see all the available layers. In your case, this should only be the ones you've added before (San Andreas?).
5. Select all of the layers by clicking the top entry and **Shift-clicking** the bottom one. Click *Add Layers* to add them all to the map.



Add Layers

6. The layers will be added to the map. Click *Done* (right next to *Add Layers* at the bottom) to return to the main layers list.
7. To save the map click on the Map button in the toolbar, and select *Save Map*.



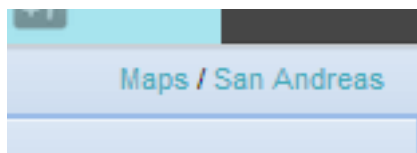
Save Map

8. Enter a title and abstract for your map.

A screenshot of a dialog box titled 'About this Map'. It has a close button (X) in the top right corner. The dialog contains two text input fields. The first field is labeled 'Title:' and contains the text 'San Andreas'. The second field is labeled 'Abstract:' and contains the text 'This map contains lots of good information about San Andreas. Highway, coastline, points of interest, you name it.' At the bottom of the dialog, there are three buttons: 'Save as Copy', 'Save', and 'Cancel'. The 'Save' button is highlighted.

Edit Map Metadata

9. Click **Save**. Notice that the link on the top right of the page changed to reflect the map's name.



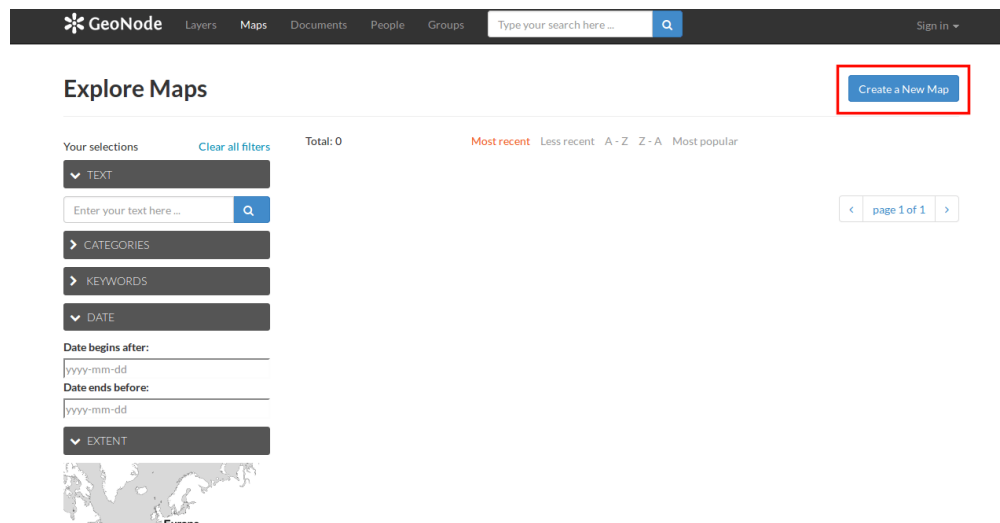
Save Map

This link contains a permalink to your map. If you open this link in a new window, your map will appear exactly as it was saved.

4. Share your map

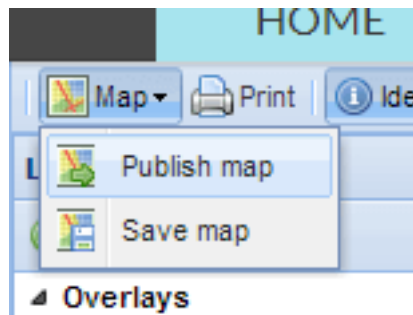
Now let's finish our map.

1. Check the box next to the **highway** layer to activate it. If it is not below the **POI** layer in the list, click and drag it down.



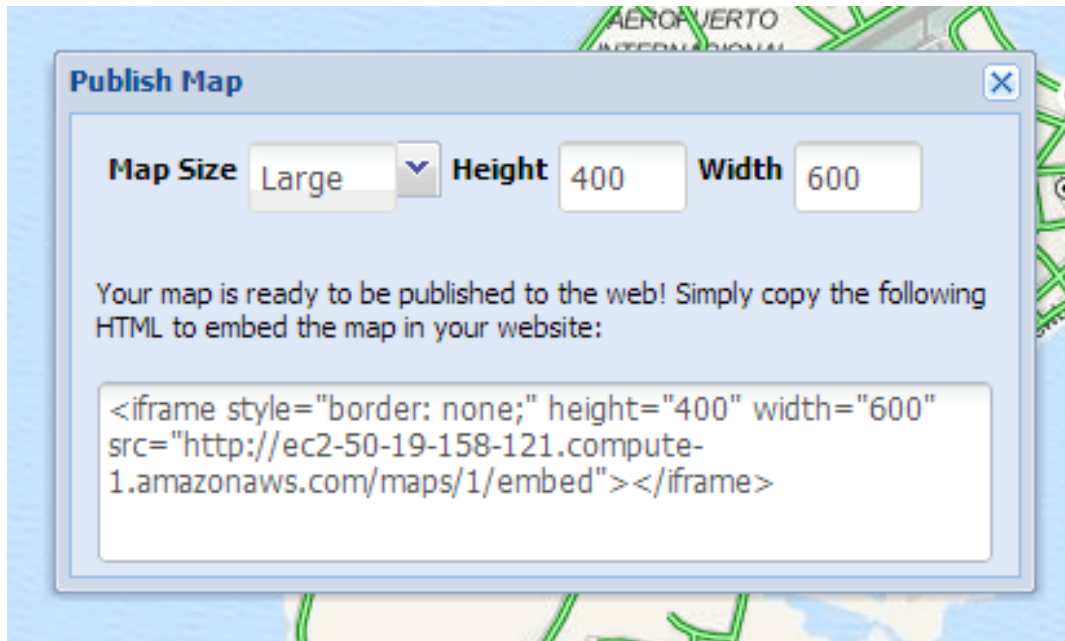
Activate Layers on Map

2. Make any final adjustments to the map composition as desired, including zoom and pan settings.
3. Click the *Map* button in the toolbar, and then click *Publish Map*.



Publish Map button

4. The title and abstract as previously created should still be there. Make any adjustments as necessary, and click *Save*.
5. A new dialog will appear with instructions on how to embed this map in a web page, including a code snippet. You can adjust the parameters as necessary.



Publish the Map

Your map can now be shared!

To be continued

Now you've gotten some quick insight in the possibilities of GeoNode. To learn more about GeoNode and its features, visit [the official GeoNode web page](#).

Stay in touch with the GeoNode community through the [Gitter.im chatroom](#) or by asking your question in our [mailing list](#)!

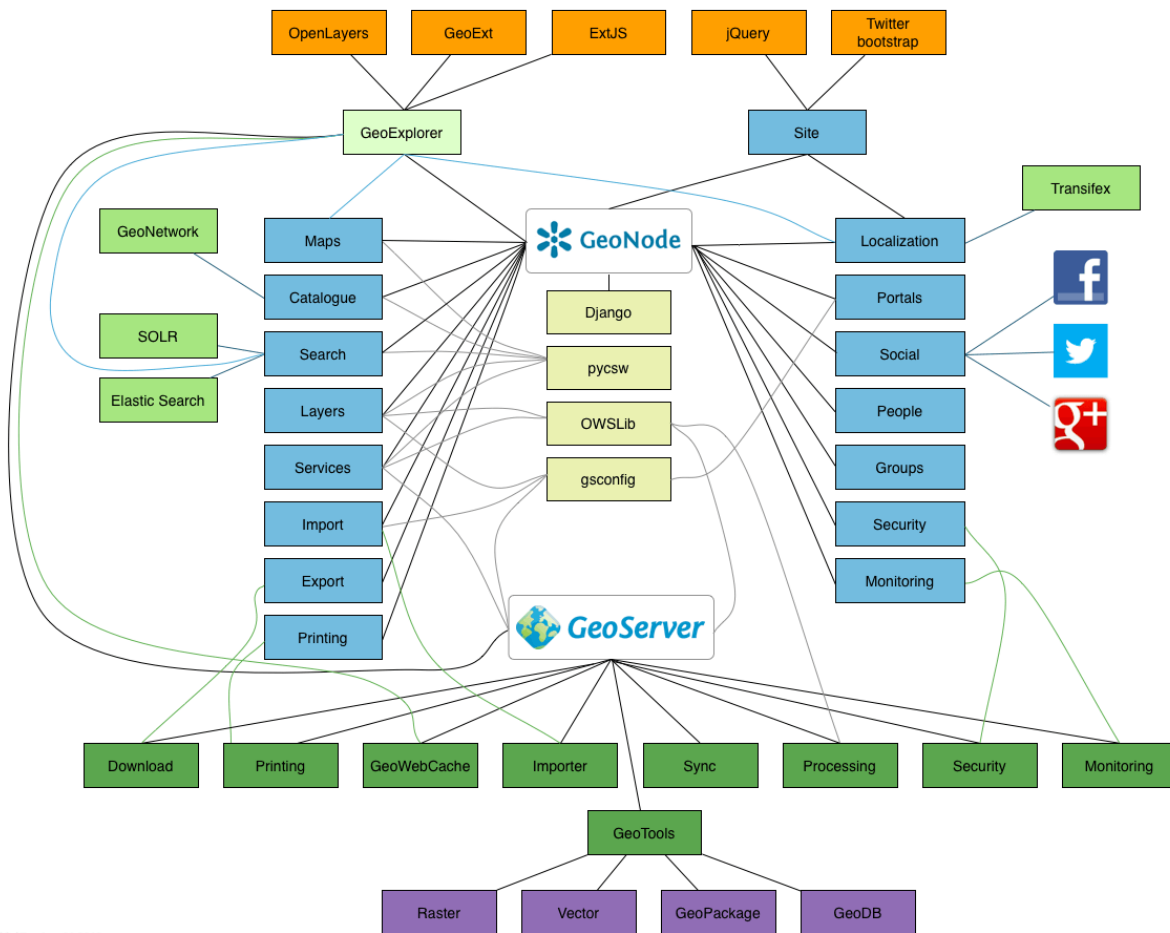
3.2.1.1.3 Reference Doc

In this section, you will find information about each and every component of GeoNode, for example *GeoServer*, *GeoNode Settings*, *Security*, etc.

The Big Picture

Architecture

GeoNode Component Architecture



SS / Tue Aug 21 2012

GeoNode Component Architecture

GeoNode core is based on Django web framework with few more dependencies necessary for the communication with the geospatial servers (GeoServer, pyCSW)

On the left side you can see the list of *Entities* defined in GeoNode and managed by the Django ORM framework. Those objects will be detailed in a future section.

On the right side the list of *Services* available allowing GeoNode to communicate with the *social* world.

The GeoNode catalog is strictly connected to the GeoServer one (see the bottom of the figure). The geospatial dataset and the OGC Services are implemented and managed by GeoServer. GeoNode acts as a broker for the geospatial layers, adding metadata information and tools that make easier the management, cataloging, mapping and searching of the datasets.

Thanks to the ORM framework and the auxiliary Python libraries, GeoNode is constantly aligned with the GeoServer catalog. An ad-hoc security module allows the two modules to strictly interact and share security and permissions rules.

In the advanced sections of this documentation we will go through GeoNode commands allowing administrators to re-sync the catalogs and keep them consistently aligned.

Django Architecture

GeoNode is based on [Django](#) which is a high level Python web development framework that encourages rapid development and clean pragmatic design. Django is based on the Model View Controller (MVC) architecture pattern, and as such, GeoNode models layers, maps and other modules with Django's [Model](#) module and these models are used via Django's [ORM](#) in views which contain the business logic of the GeoNode application and are used to drive HTML templates to display the web pages within the application.

Django explained with model/view/controller (MVC)

- Model represents application data and provides rich ORM functionality.
- Views are a rendering of a Model most often using the Django template engine.
- In Django, the controller part of this commonly discussed, layered architecture is a subject of discussion. According to the standard definition, the controller is the layer or component through which the user interacts and model changes occur.

MVP/MVC

MVP

Model, View, Presenter

In MVP, the *Presenter* contains the UI business logic for the *View*. All invocations from the *View* delegate directly to the *Presenter*. The *Presenter* is also decoupled directly from the *View* and talks to it through an interface. This is to allow mocking of the *View* in a unit test. One common attribute of MVP is that there has to be a lot of two-way dispatching. For example, when someone clicks the *Save* button, the event handler delegates to the *Presenter*'s *OnSave* method. Once the save is completed, the *Presenter* will then call back the *View* through its interface so that the *View* can display that the save has completed.

MVP tends to be a very natural pattern for achieving separated presentation in Web Forms.

Two primary variations (You can [find out more about both variants.](#))

Passive View: The View is as dumb as possible and contains almost zero logic. The Presenter is a middle man that talks to the View and the Model. The View and Model are completely shielded from one another. The Model may raise events, but the Presenter subscribes to them for updating the View. In Passive View there is no direct data binding, instead the View exposes setter properties which the Presenter uses to set the data. All state is managed in the Presenter and not the View.

- *Pro:* maximum testability surface; clean separation of the View and Model
- *Con:* more work (for example all the setter properties) as you are doing all the data binding yourself.

Supervising Controller: The Presenter handles user gestures. The View binds to the Model directly through data binding. In this case it's the Presenter's job to pass off the Model to the View so that it can bind to it. The Presenter will also contain logic for gestures like pressing a button, navigation, etc.

- *Pro:* by leveraging data binding the amount of code is reduced.
- *Con:* there's less testable surface (because of data binding), and there's less encapsulation in the View since it talks directly to the Model.

MVC

Model, View, Controller

In the MVC, the Controller is responsible for determining which View is displayed in response to any action including when the application loads.

This differs from MVP where actions route through the View to the Presenter. In MVC, every action in the View correlates with a call to a Controller along with an action. In the web each action involves a call to a URL on the other side of which there is a Controller who responds. Once that Controller has completed its processing, it will return the correct View. The sequence continues in that manner throughout the life of the application:

```
1 Action in the View
2     -> Call to Controller
3     -> Controller Logic
4     -> Controller returns the View.
```

One other big difference about MVC is that the View does not directly bind to the Model. The view simply renders, and is completely stateless. In implementations of MVC the View usually will not have any logic in the code behind. This is contrary to MVP where it is absolutely necessary as if the View does not delegate to the Presenter, it will never get called.

Presentation Model

One other pattern to look at is the Presentation Model pattern. In this pattern there is no Presenter. Instead the View binds directly to a **Presentation Model**. The Presentation Model is a Model crafted specifically for the View. This means this Model can expose properties that one would never put on a domain model as it would be a violation of separation-of-concerns. In this case, the Presentation Model binds to the domain model, and may subscribe to events coming from that Model. The View then subscribes to events coming from the Presentation Model and updates itself accordingly. The Presentation Model can expose commands which the view uses for invoking actions. The advantage of this approach is that you can essentially remove the code-behind altogether as the PM completely encapsulates all of the behaviour for the view.

This pattern is a very strong candidate for use in WPF applications and is also called **Model-View-ViewModel**.

More: <http://reinout.vanrees.org/weblog/2011/12/13/django-mvc-explanation.html>

WSGI

Web Server Gateway Interface (whis-gey)

- This is a python specification for supporting a common interface between all of the various web frameworks and an application (Apache, for example) that is ‘serving’.
- This allows any WSGI compliant framework to be hosted in any WSGI compliant server.
- For most GeoNode development, the details of this specification may be ignored.

More: <http://en.wikipedia.org/wiki/Wsgi>

GeoNode and GeoServer

GeoNode uses GeoServer for providing OGC services.

At its core, GeoNode provides a standards-based platform to enable integrated, programmatic access to your data via OGC Web Services, which are essential building blocks required to deploy an OGC-compliant spatial data infrastructure (SDI). These Web Services enable discovery, visualization and access your data, all without necessarily having to interact directly with your GeoNode website, look and feel/UI, etc.

- GeoNode configures GeoServer via the REST API
- GeoNode retrieves and caches spatial information from GeoServer. This includes relevant OGC service links, spatial metadata, and attribute information.

In summary, GeoServer contains the layer data, and GeoNode's layer model extends the metadata present in GeoServer with its own.

- GeoNode can discover existing layers published in a GeoServer via the WMS capabilities document.
- GeoServer delegates authentication and authorization to GeoNode.
- Data uploaded to GeoNode is first processed in GeoNode and finally published to GeoServer (or ingested into the spatial database).

OGC Web Services:

- operate over HTTP (GET, POST)
- provide a formalized, accepted API
- provide formalized, accepted formats

The OGC Web Services provided by GeoNode have a mature implementation base and provide an multi-application approach to integration. This means, as a developer, there are already numerous off-the-shelf GIS packages, tools and webapps (proprietary, free, open source) that natively support OGC Web Services.

There are numerous ways to leverage OGC Web Services from GeoNode:

- desktop GIS
- web-based application
- client libraries / toolkits
- custom development

Your GeoNode lists OGC Web Services and their URLs at <http://localhost:8000/developer>. You can use these APIs directly to interact with your GeoNode.

The following sections briefly describe the OGC Web Services provided by GeoNode.

Web Map Service (WMS)

WMS provides an API to retrieve map images (PNG, JPEG, etc.) of geospatial data. WMS is suitable for visualization and when access to raw data is not a requirement.

WFS

WFS provides an API to retrieve raw geospatial vector data directly. WFS is suitable for direct query and access to geographic features.

WCS

WCS provides an API to retrieve raw geospatial raster data directly. WCS is suitable for direct access to satellite imagery, DEMs, etc.

CSW

CSW provides an interface to publish and search metadata (data about data). CSW is suitable for cataloging geospatial data and making it discoverable to enable visualization and access.

WMTS

WMTS provides an API to retrieve pre-rendered map tiles of geospatial data.

WMC

WMC provides a format to save and load map views and application state via XML. This allows, for example, a user to save their web mapping application in WMC and share it with others, viewing the same content.

More: <http://geoserver.org>

GeoNode and PostgreSQL/PostGIS

In production, GeoNode is configured to use PostgreSQL/PostGIS for its persistent store. In development and testing mode, often an embedded SQLite database is used. The latter is not suggested for production.

- The database stores configuration and application information. This includes users, layers, maps, etc.
- It is recommended that GeoNode be configured to use PostgreSQL/PostGIS for storing vector data as well. While serving layers directly from shapefile allows for adequate performance in many cases, storing features in the database allows for better performance especially when using complex style rules based on attributes.

GeoNode and pycsw

GeoNode is built with pycsw embedded as the default CSW server component.

Publishing

Since pycsw is embedded in GeoNode, layers published within GeoNode are automatically published to pycsw and discoverable via CSW. No additional configuration or actions are required to publish layers, maps or documents to pycsw.

Discovery

GeoNode's CSW endpoint is deployed available at <http://localhost:8000/catalogue/csw> and is available for clients to use for standards-based discovery. See <http://docs.pycsw.org/en/latest/tools.html> for a list of CSW clients and tools.

Javascript in GeoNode

GeoNode provides a number of facilities for interactivity in the web browser built on top of several high-quality JavaScript frameworks:

- **Bootstrap** for GeoNode's front-end user interface and common user interaction.
- **Bower** for GeoNode's front-end package management.
- **ExtJS** for component-based UI construction and data access
- **OpenLayers** for interactive mapping and other geospatial operations
- **GeoExt** for integrating ExtJS with OpenLayers
- **Grunt** for front-end task automation.
- **GXP** for providing some higher-level application building facilities on top of GeoExt, as well as improving integration with GeoServer.
- **jQuery** to abstract Javascript manipulation, event handling, animation and Ajax.

GeoNode uses application-specific modules to handle pages and services that are unique to GeoNode. This framework includes:

- A **GeoNode mixin** class that provides GeoExplorer with the methods needed to properly function in GeoNode. The class is responsible for checking permissions, retrieving and submitting the **CSRF token**, and user authentication.
- A **search module** responsible for the GeoNode's site-wide search functionality.
- An **upload and status module** to support file uploads.
- **Template files** for generating commonly used html sections.
- A **front-end testing module** to test GeoNode Javascript.

The following concepts are particularly important for developing on top of the GeoNode's JavaScript framework.

- **Components** Ext components handle most interactive functionality in "regular" web pages. For example, the scrollable/sortable/filterable table on the default Search page is a Grid component. While GeoNode does use some custom components, familiarity with the idea of Components used by ExtJS is applicable in GeoNode development.
- **Viewers** Viewers display interactive maps in web pages, optionally decorated with Ext controls for toolbars, layer selection, etc. Viewers in GeoNode use the GeoExplorer base class, which builds on top of GXP's Viewer to provide some common functionality such as respecting site-wide settings for background layers. Viewers can be used as components embedded in pages, or they can be full-page JavaScript applications.
- **Controls** Controls are tools for use in OpenLayers maps (such as a freehand control for drawing new geometries onto a map, or an identify control for getting information about individual features on a map.) GeoExt provides tools for using these controls as ExtJS "Actions" - operations that can be invoked as buttons or menu options or associated with other events.

Components

architecture is based on a set of core tools and libraries that provide the building blocks on which the application is built. Having a basic understanding of each of these components is critical to your success as a developer working with GeoNode.

Lets look at each of these components and discuss how they are used within the GeoNode application.

Django

GeoNode is based on [Django](#) which is a high level Python web development framework that encourages rapid development and clean pragmatic design. Django is based on the Model View Controller (MVC) architecture pattern, and as such, GeoNode models layers, maps and other modules with Django's [Model](#) module and these models are used via Django's [ORM](#) in views which contain the business logic of the GeoNode application and are used to drive HTML templates to display the web pages within the application.

GeoServer

[GeoServer](#) is an open source software server written in Java that provides OGC compliant services which publish data from many spatial data sources. GeoServer is used as the core GIS component inside GeoNode and is used to render the layers in a GeoNode instance, create map tiles from the layers, provide for downloading those layers in various formats and to allow for transactional editing of those layers.

GeoExplorer

[GeoExplorer](#) is a web application, based on the [GeoExt](#) framework, for composing and publishing web maps with OGC and other web based GIS Services. GeoExplorer is used inside GeoNode to provide many of the GIS and cartography functions that are a core part of the application.

PostgreSQL and PostGIS

[PostgreSQL](#) and [PostGIS](#) are the database components that store and manage spatial data and information for GeoNode and the Django modules that it is composed of, pycsw and GeoServer. All of these tables and data are stored within a geonode database in PostgreSQL. GeoServer uses PostGIS to store and manage spatial vector data for each layer which are stored as a separate table in the database.

pycsw

[pycsw](#) is an OGC CSW server implementation written in Python. GeoNode uses pycsw to provide an OGC compliant standards-based CSW metadata and catalogue component of spatial data infrastructures, supporting popular geospatial metadata standards such as Dublin Core, ISO 19115, FGDC and DIF.

Geospatial Python Libraries

GeoNode leverages several geospatial python libraries including [gsconfig](#) and [OWSLib](#). gsconfig is used to communicate with GeoServer's REST Configuration API to configure GeoNode layers in GeoServer. OWSLib is used to communicate with GeoServer's OGC services and can be used to communicate with other OGC services.

Django Pluggables

GeoNode uses a set of Django plugins which are usually referred to as pluggables. Each of these pluggables provides a particular set of functionality inside the application from things like Registration and Profiles to interactivity with external sites. Being based on Django enables GeoNode to take advantage of the large ecosystem of these pluggables out there, and while a specific set is included in GeoNode itself, many more are available for use in applications based on GeoNode.

jQuery

jQuery is a feature-rich Javascript library that is used within GeoNode to provide an interactive and responsive user interface as part of the application. GeoNode uses several jQuery plugins to provide specific pieces of functionality, and the GeoNode development team often adds new features to the interface by adding additional plugins.

Bootstrap

Bootstrap is a front-end framework for laying out and styling the pages that make up the GeoNode application. It is designed to ensure that the pages render and look and behave the same across all browsers. GeoNode customizes Bootstrap's default style and its relatively easy for developers to customize their own GeoNode based site using existing Bootstrap themes or by customizing the styles directly.

Users' Features Open Source Geospatial Content Management System

GeoNode is a web-based application and platform for developing geospatial information systems (GIS) and for deploying spatial data infrastructures (SDI).

What GeoNode can be used for... [GeoNode Demo](#) (admin/admin)

Introduction This section introduces the GeoNode GUI and functionalities through a step-by-step workshop.

At the end of this module the users will be familiar with the GeoNode default GUI and objects.

Reference Doc In this section, you will find information about each and every component of GeoNode, for example *GeoServer*, *GeoNode Settings*, *Security*, etc.

3.2.1.2 Installation & Admin

Welcome to the GeoNode Training *Installation & Admin* documentation v2.8.

This module is more oriented to users having some System Administrator background.

At the end of this section you will be able to setup from scratch the whole GeoNode infrastructure and understand how to the different pieces are interconnected and which are their dependencies.

Prerequisites

Before proceeding with the reading, it is strongly recommended to be sure having clear the following concepts:

1. GeoNode and Django framework basic concepts
2. What is Python
3. What is a DBMS
4. What is a Java Virtual Machine and the JDK
5. Basic TCP/IP and networking concepts

6. Linux OS basic shell and maintenance commands
7. Apache HTTPD Server and WSGI Python bindings

3.2.1.2.1 Quick Installation Guide

The following is a quick guide to get GeoNode up and running in most common operating systems. This is meant to be run on a fresh machine with no previously installed packages or GeoNode versions.

Recommended Minimum System Requirements

For deployment of GeoNode on a single server, the following are the *bare minimum* system requirements:

- 6GB of RAM, including swap space.
- 2.2GHz processor. (Additional processing power may be required for multiple concurrent styling renderings)
- 1 GB software disk usage.
- Additional disk space for any data hosted with GeoNode and tiles cached with GeoWebCache. For spatial data, cached tiles, and “scratch space” useful for administration, a decent baseline size for GeoNode deployments is 100GB.
- 64-bit hardware recommended.

Linux

Ubuntu (standard deployment)

The easiest way to get the .deb is to install it using APT, the standard installation management tool for Ubuntu. The current release, 2.8 is available only for Ubuntu 16.04.

These instructions have been tested on a fresh install of Ubuntu 16.04 64 bit server edition. Please ensure the latest packages are installed with:

```
sudo apt-get update; sudo apt-get upgrade
```

Amongst other things, this will ensure that the `software-properties-common` package is installed, which is required to make the `add-apt-repository` command used below available.

The steps to install geonode and all dependencies in Ubuntu 16.04 are as follows:

1. Set up the GeoNode PPA repository (you only need to do this once; the repository will still be available for upgrades later):

```
sudo add-apt-repository ppa:geonode/stable
```

Note: If you want to try latest unstable version of GeoNode, add the repository:

```
sudo add-apt-repository ppa:geonode/testing
```

2. Install the geonode package and dependencies:

```
sudo apt-get update; sudo apt-get upgrade; sudo apt-get autoremove  
sudo apt-get install geonode
```

3. Set the correct IP address (NB: the first command below looks up the IP address automatically):

```
.. note:: If you want to run geonode on your local instance, just run::
```

```
sudo geonode-updateip -p localhost
IP_ADDRESS=$(ip route get 8.8.8.8 | awk '{print $NF; exit}') sudo geonode-updateip -p
$IP_ADDRESS
```

Note: If geoserver and geonode are not on the same machine then add your local geonode address:

```
sudo geonode-updateip -p $IP_ADDRESS -l yourlocaladdress
```

NB: The IP address must be set to enable access from another machine, e.g. the host machine if geonode is installed in a virtual machine.

4. Access geonode from your web browser, using the IP address from step 3 above. The geonode web application should be displayed and you can log in using the superuser details entered immediately above:

Note: If you want to access geonode on your local instance, just go to:

```
http://localhost/
```

```
http://{IP_ADDRESS}/
```

NB: With this installation method you use `sudo geonode xxxx` in place of any `python manage.py xxx` command referred to in documentation. For example, try:

```
sudo geonode help
```

For further information, read the Admin Docs at <http://docs.geonode.org/en/master/#for-administrators>.

Ubuntu (for development)

This option installs geonode in a virtual environment. This option is very useful in case you want to develop using Ubuntu (tested on Ubuntu 16.04 LTS):

```
# Install Ubuntu dependencies
sudo apt-get update
sudo apt-get install python-virtualenv python-dev libxml2 libxml2-dev libxslt1-dev
↳zlib1g-dev libjpeg-dev libpq-dev gdal libgdal-dev git default-jdk

# Install Java 8 (needed by latest GeoServer 2.9)
sudo apt-add-repository ppa:webupd8team/java
sudo apt-get update
sudo apt-get install oracle-java8-installer

# Create and activate the virtulenv
virtualenv --no-site-packages env
source env/bin/activate

# git clone geonode
git clone https://github.com/GeoNode/geonode
cd geonode
```

(continues on next page)

(continued from previous page)

```
# Install pip dependencies
pip install -r requirements.txt --upgrade
pip install -e . --upgrade --no-cache

sudo add-apt-repository ppa:ubuntugis/ppa && sudo apt-get update
sudo apt-get install gdal-bin

# install the correct PyGDAL version
gdal-config --version | cut -c 1-5 | xargs -I % pip install 'pygdal>=%0,<=%0.999'

# if the command cannot install a suitable version, be sure to install at least the
↪ closer major one e.g. 2.1.2 -> 2.1.2.3
# gdalinfo --version
# pip install pygdal==`gdal-config --version`
```

You can now setup and start GeoNode:

```
# Paver setup
paver setup
paver sync
paver start
```

In case you want to be involved in static files development:

```
# The following lines must be run only the first time
sudo apt-get install -y nodejs, npm
```

Rebuild libraries through:

```
cd geonode/static

# The following lines must be run only the first time
npm install --save-dev
npm install bower --save-dev
npm install grunt-cli --save-dev
npm install grunt-contrib-jshint --save-dev
npm install grunt-contrib-less --save-dev
npm install grunt-contrib-concat --save-dev
npm install grunt-contrib-copy --save-dev
npm install grunt-text-replace --save-dev
npm install grunt-contrib-uglify --save-dev
npm install grunt-contrib-cssmin --save-dev
npm install grunt-contrib-watch --save-dev
npm install -g grunt-cli

# Later you can just do the following
bower cache clean
bower update
grunt production
```

Warning: If you get an error like `/usr/bin/env: node: No such file or directory` while running bower, try to execute the following command:

```
sudo ln -s /usr/bin/nodejs /usr/bin/node
```

On a production environment, remember to refresh also the “static_root” folder:

```
python manage.py collectstatic --clear -i site-packages* -i .components* -i node_
↳modules* --noinput
```

Windows, OSX and others

Windows

It is now available an automatic installer for Windows systems which configures GeoNode with PostgreSQL DB and few sample layers on a few steps. See detailed instructions and steps of the GeoNode Windows Installer at section *Windows Binary Installer*.

If you have different needs or just want to configure the GeoNode environment manually, please continue reading the guide below.

To install in Windows it is assumed you’re familiar with python development and virtualenv on Windows and that you’re familiar with the windows command prompt. You will need the follow prerequisites installed:

- Java JDK
- Python 2.7.9 * Earlier versions of python require you to install distutils (easy_install) - <http://www.lfd.uci.edu/~gohlke/pythonlibs/#setuptools>
- ant (bin directory must be on system PATH)
- maven2 (bin directory must be on system PATH)
- git

Install and configure from the windows command prompt, if you don’t already have python virtualenv installed, then do it now:

```
easy_install virtualenv
```

Create virtualenv and activate it:

```
cd <Directory to install the virtualenv & geonode into>
virtualenv <name of virtualenv folder>
virtualenv <name of virtualenv folder>\scripts\activate
```

Clone GeoNode:

```
git clone https://github.com/GeoNode/geonode.git
cd geonode
```

Install Python native dependencies, this command will look for and install binary distributions (pip install will attempt to build and fail):

```
pip install paver
paver win_install_deps
```

Install GeoNode in the local virtualenv:

```
pip install -e . --upgrade --no-cache
```

You have two options to set up the GEOS and GDAL libraries. Either create an environment variable or add the variables to your local_settings.py file as below.

```
GEOS_LIBRARY_PATH="C:/path/to/geos_c.dll" GDAL_LIBRARY_PATH="C:/path/to/gdal111.dll"
```

The GEOS and GDAL libraries can be found in your <virtualenv directory>Libsite-packagesGDAL-1.11.0-py2.7-win32.eggosgeofolder.

Setup GeoServer:

```
paver setup
```

Start the servers. You have the option to set the JAVA_HOME environment variable or use the -java_path.:

```
paver start --java_path="C:\path\to\java\java.exe"
```

or if you set your JAVA_HOME environment variables (e.g. JAVA_HOME="C:Program FilesJavajdk1.7.0_75"):

```
paver start
```

Once the package is installed, please consult custom_install to learn how to create an account for the admin user and tweak the settings to get more performance.

OSX

The recommended install method in these platforms is to use a virtualization solution, like **Virtual Box**, install the latest **Ubuntu Linux** and then proceed with the steps mentioned above. Some GeoNode developers prefer to use **Vagrant** - a VirtualBox wrapper, the steps for this are detailed below. The Vagrant quickstart guide shows how to get a Linux VM configured in most operating systems. However, if you would like to develop natively on Mac OS X please follow the following instructions.

You may need brew install various dependencies:

```
mkdir -p ~/pyenv
virtualenv ~/pyenv/geonode
source ~/pyenv/geonode/bin/activate
git clone https://github.com/GeoNode/geonode
cd geonode
pip install lxml
pip install pyproj
pip install nose
pip install httpplib2
pip install shapely
pip install pillow
pip install paver
```

Node and tools required for static files development:

```
brew install node
npm install -g bower
npm install -g grunt-cli
```

Rebuild libraries through:

```
cd geonode/static
bower cache clean
bower update
grunt production
```

On a production environment, remember to refresh also the "static_root" folder:


```
python manage.py collectstatic --clear -i site-packages* -i .components* -i node_
↳modules* --noinput
```

Install pip dependencies:

```
pip install -e . --upgrade --no-cache
```

Paver handles dependencies for Geonode, first setup (this will download and update your python dependencies - ensure you're in a virtualenv):

```
paver setup
paver start
```

Optional: To generate document thumbnails for PDFs and other ghostscript file types, download ghostscript: <https://www.macupdate.com/app/mac/9980/gpl-ghostscript>:

```
sudo apt-get install imagemagick
brew install imagemagick
pip install Wand==0.3.5
```

Once fully started, you should see a message indicating the address of your geonode. The default username and password are admin and admin:

```
Development GeoNode is running at http://localhost:8000/
The GeoNode is an unstoppable machine
Press CTRL-C to shut down
```

Before starting GeoNode (paver start), you could test your installation by running tests:

```
paver test
paver test_integration
```

In case you want to build yourself the documentation, you need to install Sphinx and the run 'make html' from within the docs directory:

```
pip install Sphinx
cd docs
make html
```

You can eventually generate a pdf containing the whole documentation set. For this purpose, if using Ubuntu 12.4 you will need to install the texlive-full package:

```
sudo apt-get install texlive-full
make latexpdf
```

Note: When running `virtualenv venv the --system-site-packages` option is not required. If not enabled, the bootstrap script will sandbox your virtual environment from any packages that are installed in the system, useful if you have incompatible versions of libraries such as Django installed system-wide. On the other hand, most of the times it is useful to use a version of the Python Imaging Library provided by your operating system vendor, or packaged other than on PyPI. When in doubt, however, just leave this option out.

Vagrant

<https://www.vagrantup.com/docs/getting-started/index.html>

CentOS/RHEL and other *nix distros

We recommend you to download the latest release and modify the included `install.sh` and `support/config.sh`. GeoNode has been installed in CentOS/RHEL using this mechanism.

Once the package is installed, please consult the `custom_install` to learn how to create the admin user and tweak the settings to get more performance.

3.2.1.2.2 Linux Admin Intro

This part of the documentation contains basic instruction on how to setup and manages Virtual Machine.

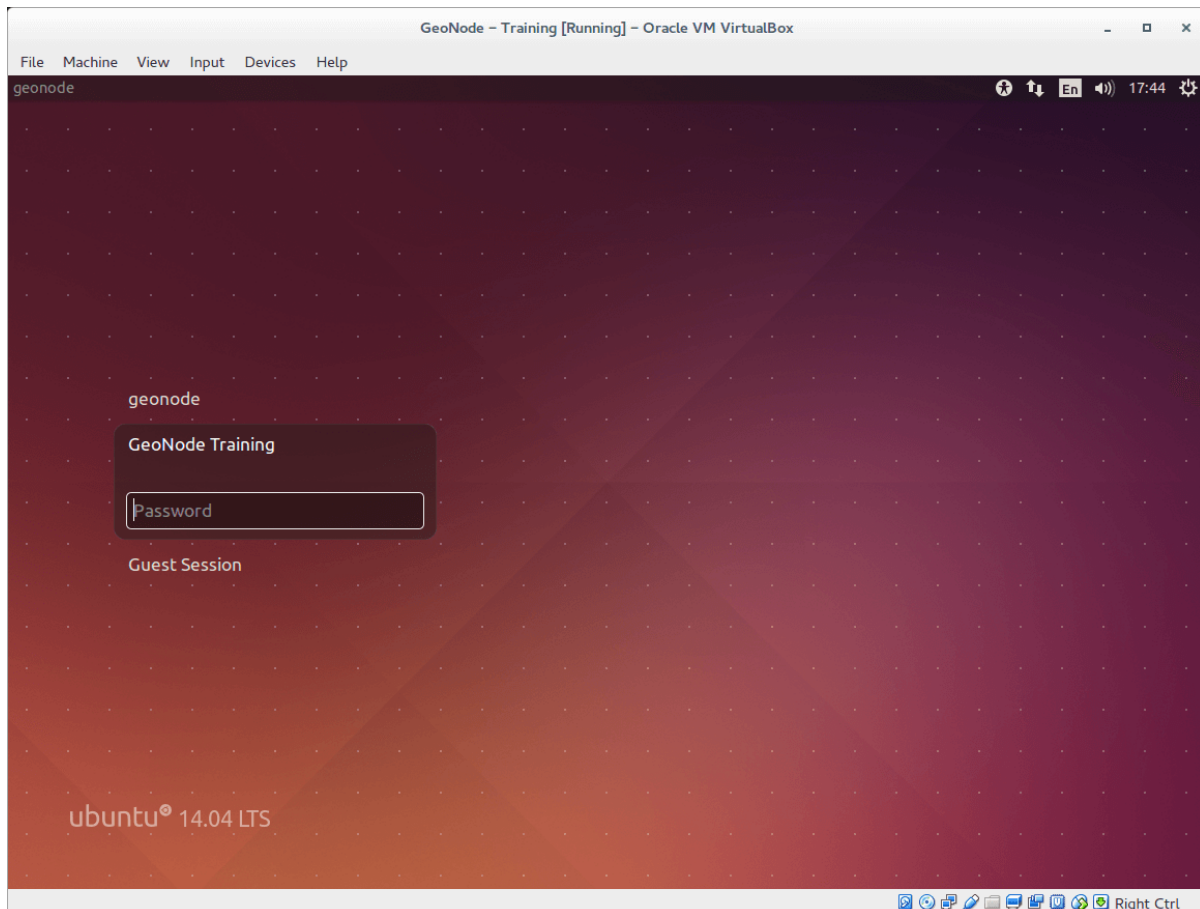
Ubuntu Basic Tutorial

Ubuntu is one of the most widespread Linux Distributions .

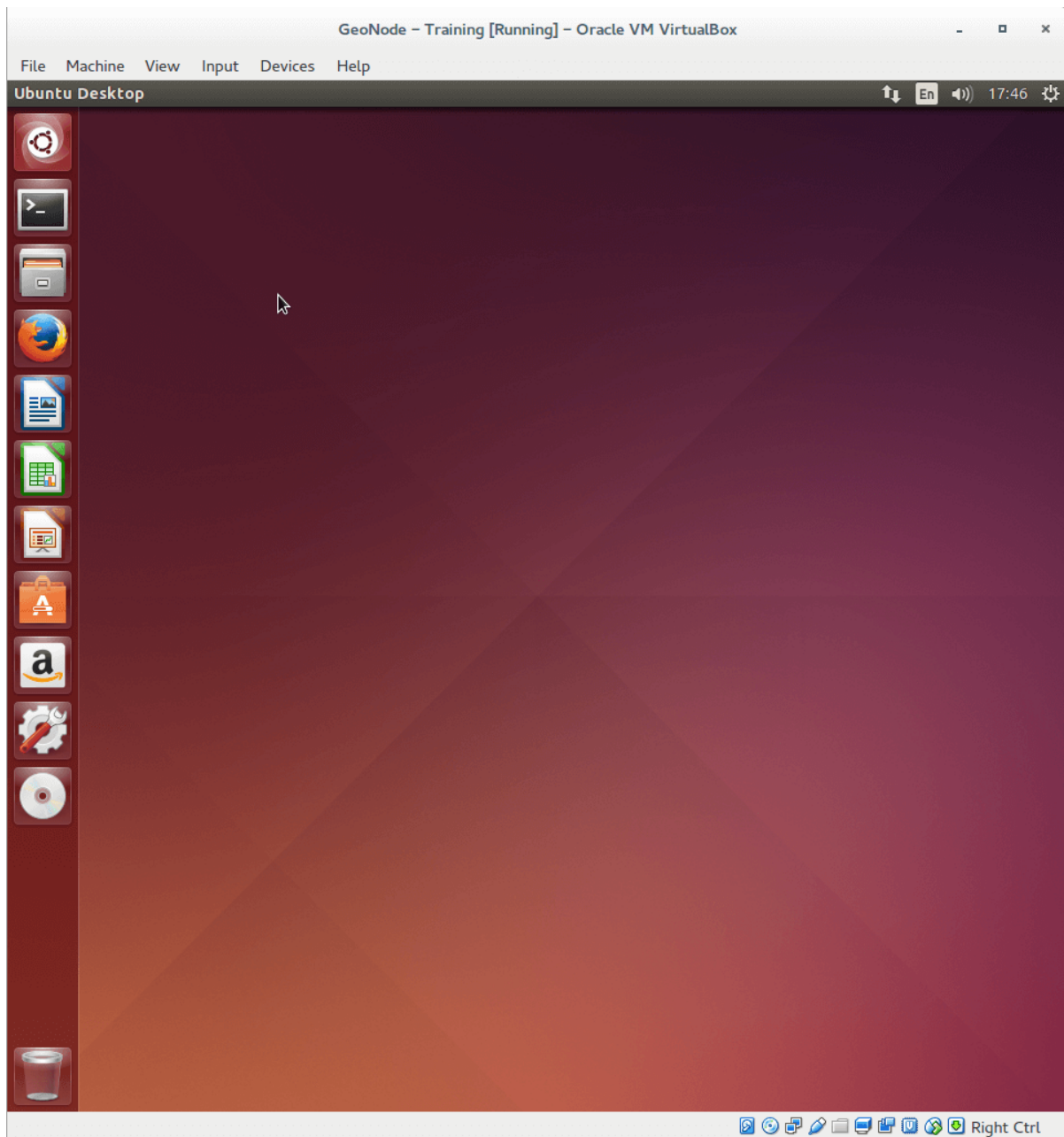
In this section of the documentation you will learn how to do basic operations in Ubuntu such as login and logout, launching applications and installing new software.

User Login

When you first start Ubuntu, at the end of the boot process you see the Ubuntu login screen



Select the user you want to login as, enter the password and press *Enter*. In a few second the user's desktop will appear.



User Interface Walkthrough

The panel on the left side of the screen contains shortcuts to frequently used application. From dark grey bar at the top you can reach network settings (the icon with two arrows pointing in opposite directions) system language (the icon with *En* written inside it), audio volume, system date and time and power menu (top right corner with an icon half way between a gear and power button).

From the power menu you can switch to a different user, logout, power off the system or access system settings.

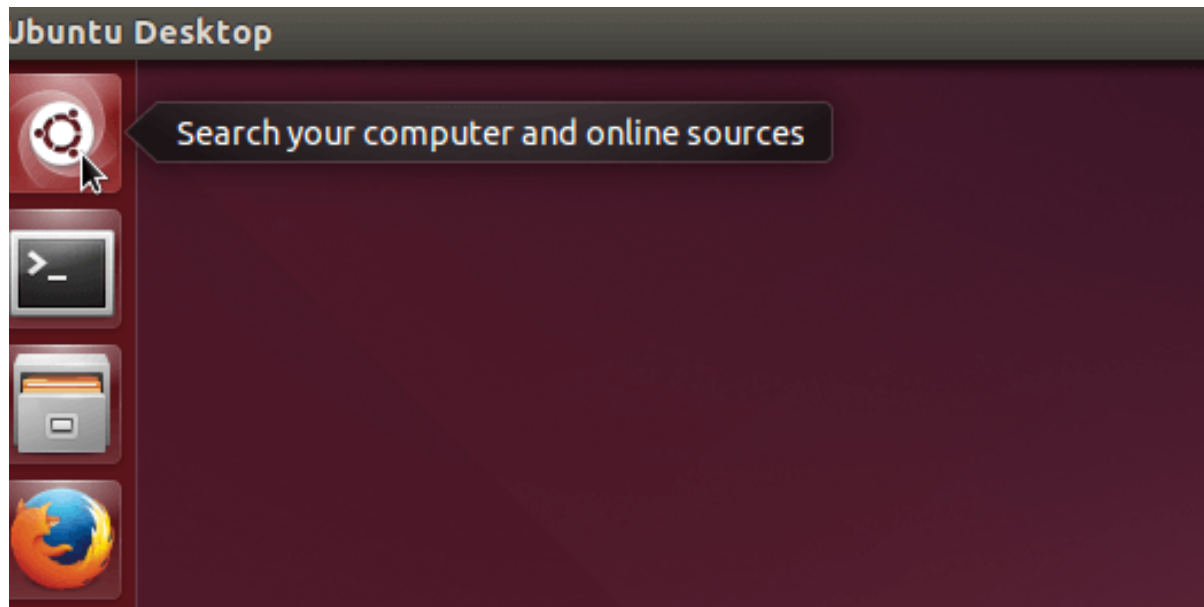
In 'system setting ' menu you can set several different parameters for the system



Launch an application

You can launch the applications listed in the *Favourites* panel simply by clicking on them.

If the application you want to launch is not in the favourites panel, use the Ubuntu Launcher. Click on the Ubuntu Launcher icon in the top left corner of the screen

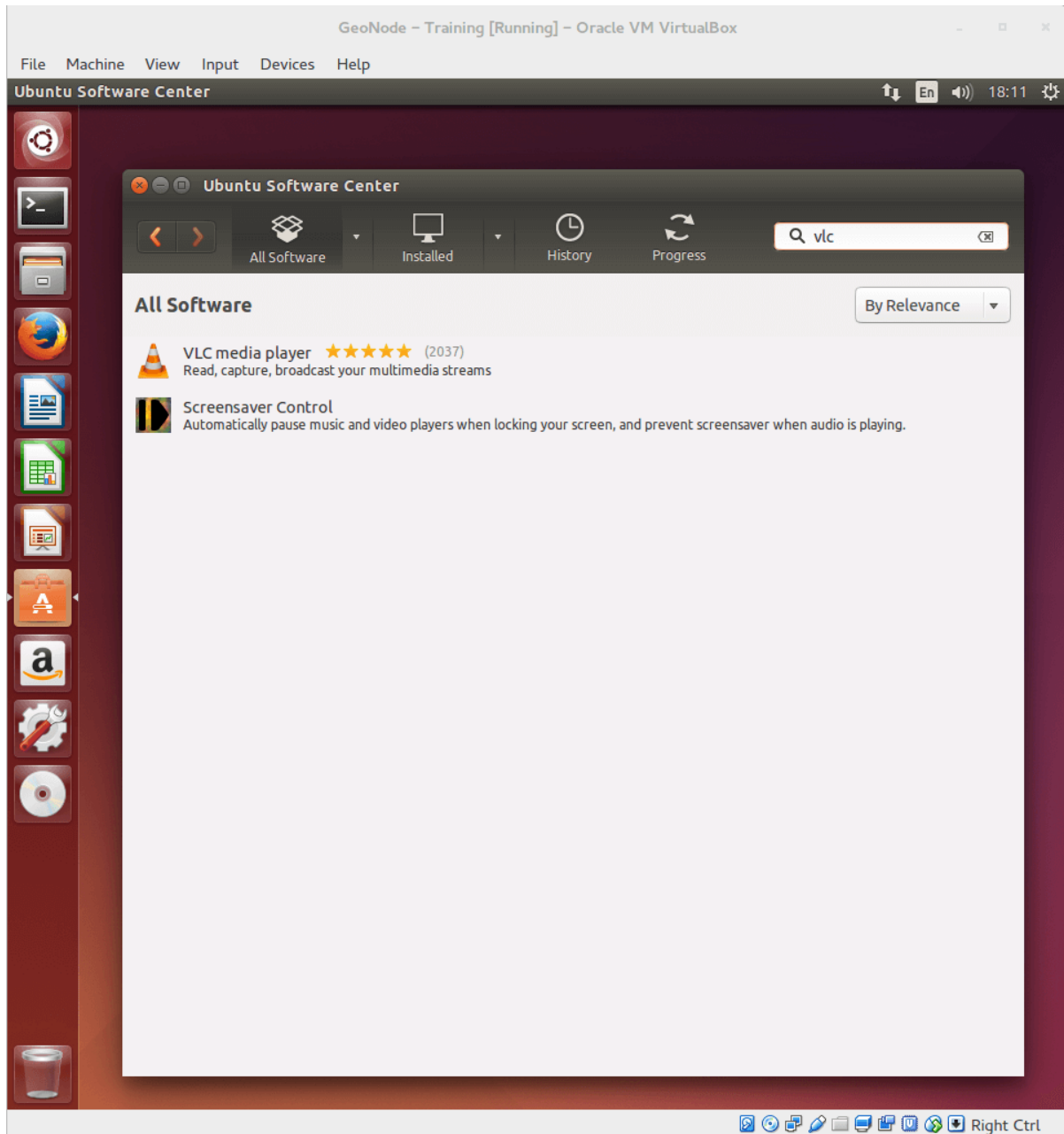


Write down the name of the application. A list of applications matching the name you are searching will show up, for and press *Enter* or click on the icon of the application.

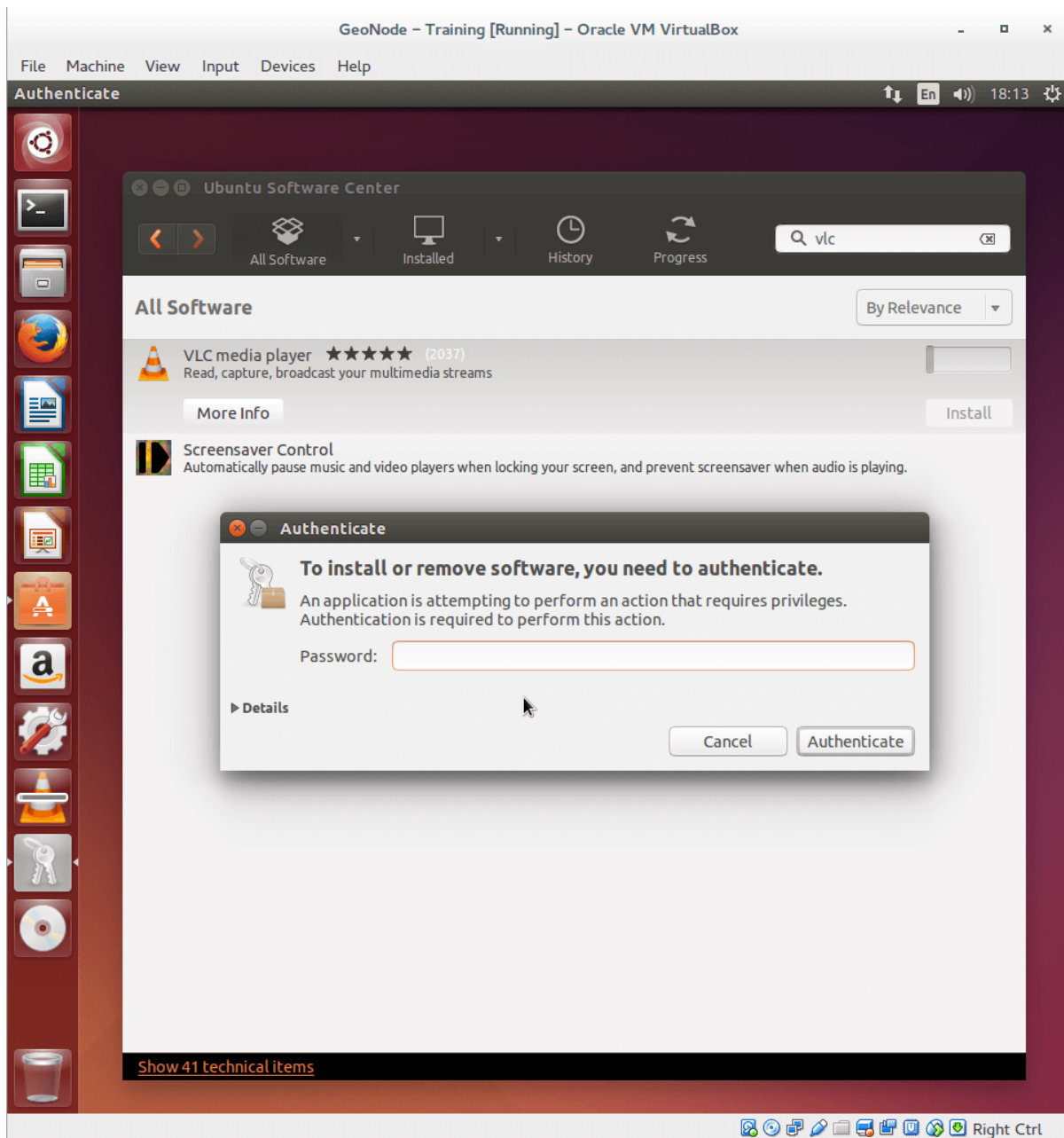
Install new software

To install new software, open the *Ubuntu software Center* (you will find it in the favourite applications panel).

Enter the name of the application you are looking for in the search bar



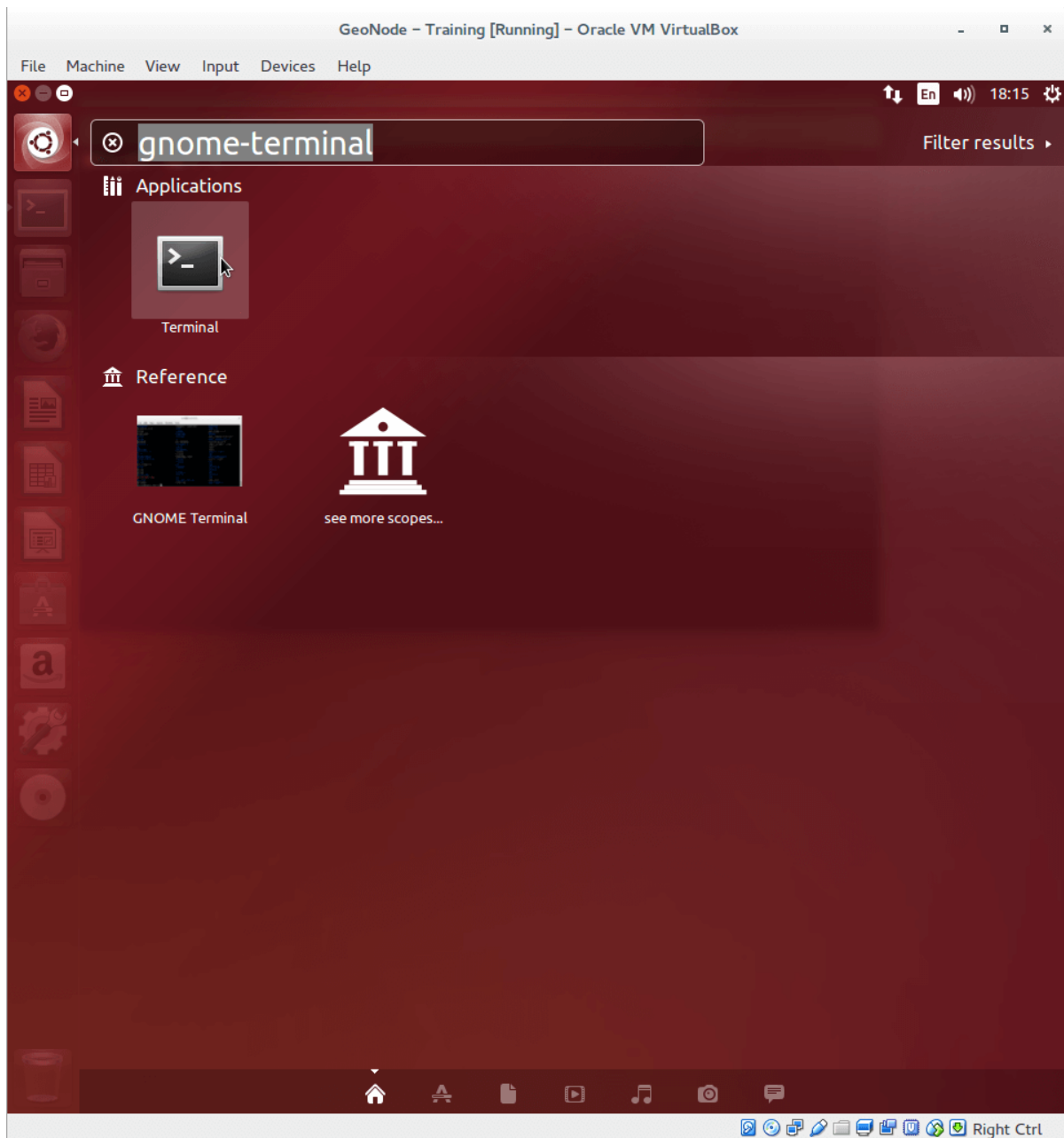
A list of candidate applications will appear. Click on the one you want to install, then click *install* to install it. You will be prompted for administrative password



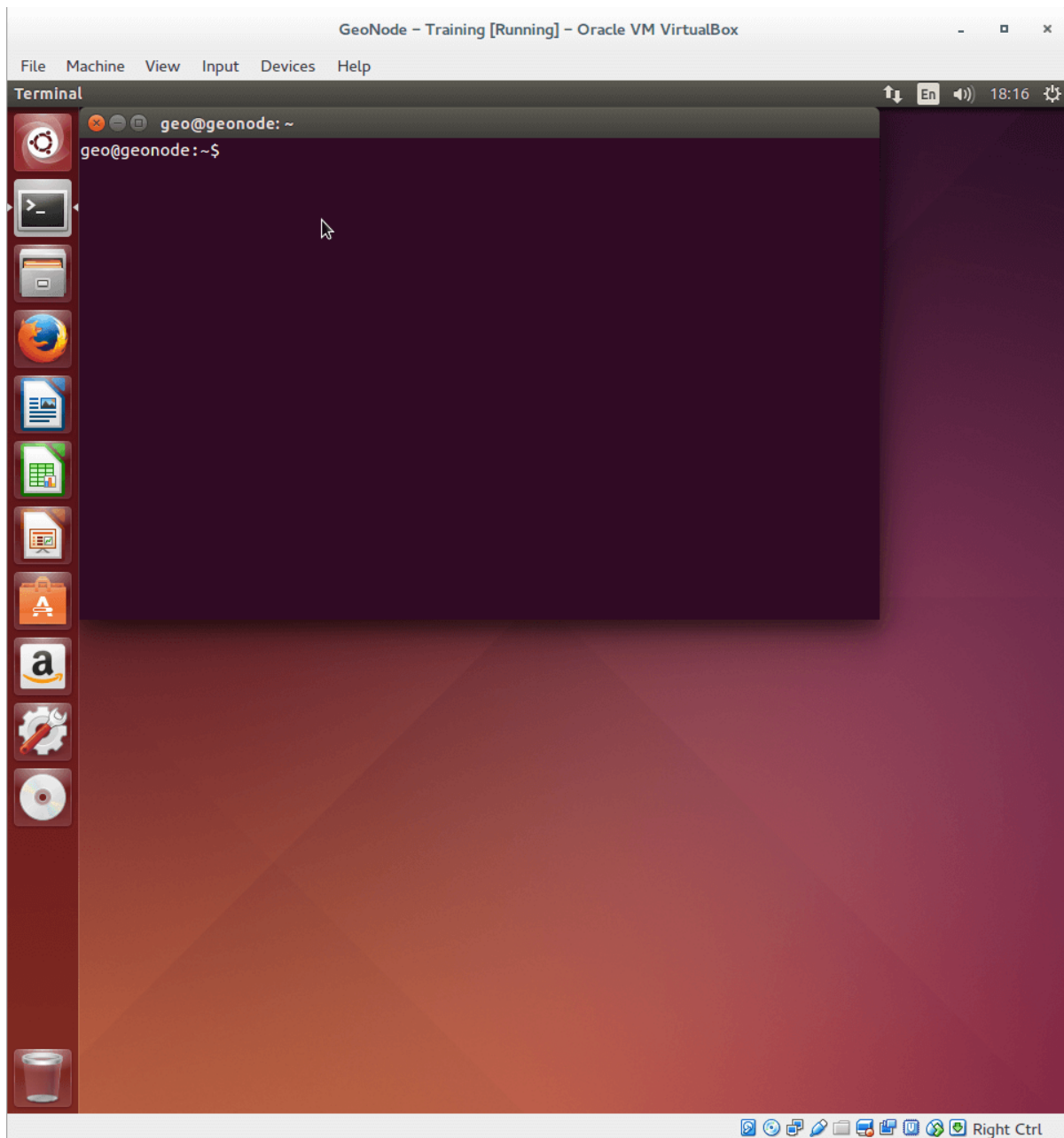
And your application will be installed in the system.

Launch the terminal emulator

Click on the *Ubuntu Launcher* icon in the top left corner of the screen, and type `gnome-terminal` in the search box



And launch the terminal emulator.



Terminal emulator will open and will be ready for your commands.

Basic commands

Current working directory

```
$ pwd
/home/geo
```

The `pwd` command will show you your *working directory*, that is the directory you are inside of and running your commands in.

Create a directory

```
$ mkdir test
```

To create a new directory inside your working directory use the *mkdir* command followed by the folder name argument

Delete a directory

```
$ rmdir test
```

To delete an empty directory type *rmdir* followed by the folder name argument

Create an empty file

```
$ touch testfile
```

To create an empty file in your current working directory use the *touch* command followed by the name of the file

Delete a file

```
$ rm filename
```

To delete a file use the *rm* command followed by the file name

Change working directory

```
$ cd /home
```

To change your current working directory use the *cd* command followed by the *path* (location) you want to change to

List content of a folder

```
$ ls
```

The *ls* command will list the content of your current working directory. You can optionally provide a path to a directory as argument, in that case *ls* will show you the content of that directory

```
$ ls /home  
geo  geonode
```

Home folder

A user's home folder is the folder where he or she will do most of the operations in. Inside your home folder you can freely create or delete file and folders.

To switch to your home folder you can use the tilde *~* character as a shortcut

```
$ cd ~  
$ pwd  
/home/geo
```

For more information on Ubuntu refer to the [Ubuntu user manual](#)

For more terminal commands read the [Using the terminal](#) guide

CentOS Basic Tutorial

This section is still under construction... please be patient!

3.2.1.2.3 Running Ansible scripts

[Ansible](#) is a free software platform used for configuring and managing computers. It is written in [Python](#) and allows users to manage *nodes* (computers) over SSH.

Configuration files are written in [YAML](#), a simple, human-readable, data serialization format.

Installing Ansible

Before you install *Ansible* make sure you have Python 2.6 or Python 2.7 on the controlling machine, you will also need an SSH client. Most Linux distributions come with an SSH client preinstalled.

If you encounter any problems during the installation, refer to the [official documentation](#).

Windows

Windows is not supported as a controlling machine.

Ubuntu

Fist configure Ansible [PPA](#):

```
sudo apt-get install software-properties-common  
sudo apt-add-repository ppa:ansible/ansible
```

Then update your available software index and install Ansible:

```
sudo apt-get update  
sudo apt-get install ansible
```

Running Ansible

To test your Ansible installation, run the following command

Note: you need a running SSH server on your machine for this to work:

```
ansible localhost -m ping
```

You should get the following output:

```
localhost | success >> {
  "changed": false,
  "ping": "pong"
}
```

Ansible Hosts file

Ansible keeps information about the managed nodes in the *inventory* or *hosts file*. Edit or create the hosts file:

```
vim /etc/ansible/hosts
```

This file contains a list of nodes for Ansible to manage. Nodes can be referred either with IP or host name. The syntax is the following:

```
192.168.1.50
aserver.example.org
bserver.example.org
```

You can also arrange hosts in groups:

```
mail.example.com

[webservers]
foo.example.com
bar.example.com

[dbservers]
one.example.com
two.example.com
three.example.com
```

Public Key access

To avoid having to type your user's password to connect to the nodes over and over, using SSH keys is recommended.

To setup Public Key SSH access to the nodes. First create a key pair:

```
ssh-keygen
```

And follow the instructions on the screen. A new key pair will be generated and placed inside the *.ssh* folder in your user's home directory.

All you need to do now is copy the public key (*id_rsa.pub*) into the *authorized_keys* file on the node you want to manage, inside the user's home directory. For example if you want to be able to connect to *training.geonode1.com* as user *geo* edit the */home/geo/.ssh/authorized_keys* file on the remote machine and add the content of your public key inside the file.

For more information on how to setup SSH keys in Ubuntu refer to [this](#) document.

Connect to managed nodes

Now that SSH access to the managed nodes is in place for all the nodes inside the Ansible *inventory* (hosts file), we can run our first command:

```
ansible all -m ping -u geo
```

Note: change *geo* with the username to use for SSH login

The output will be similar to this::

```
ansible all -m ping -u geo
84.33.2.70 | success >> {
  "changed": false,
  "ping": "pong"
}
```

We asked Ansible to connect to all the machine in our *Inventory* as user *geo* and run the *module* *ping* (modules are Ansible's units of work, more on that later...). As you can see by the output, Ansible successfully connected to the remote machine and executed the module *ping*.

Ad hoc commands

An ad-hoc command is something that you might type in to do something really quick, but don't want to save for later.

Later you are going to write so called *Playbooks* with the commands to run on the controlled node but for learning purposes ad-hoc commands can be used to do quick things.

One example of an ad-hoc command is the *ping* command we just ran. We typed in the command line and ran it interactively.

Another example:

```
ansible all -m shell -a "free" -u geo
84.33.2.70 | success | rc=0 >>
      total      used      free     shared    buffers     cached
Mem:    4049236   3915596   133640         0     650560     2487416
-/+ buffers/cache:    777620   3271616
Swap:    4194300   730268   3464032
```

In this example we ran the *free* command on the remote hosts to get memory usage stats. Note that we used the *shell* module (*-m* flag) with the command as the argument (*-a* flag).

File Transfer

Another use case for the Ansible command is to transfer files over SCP:

```
ansible 84.33.2.70 -m copy -a "src=/home/geo/test dest=~/" -u geo
84.33.2.70 | success >> {
  "changed": true,
  "dest": "/home/geo/test",
  "gid": 1000,
  "group": "geo",
```

(continues on next page)

(continued from previous page)

```
"md5sum": "d41d8cd98f00b204e9800998ecf8427e",
"mode": "0664",
"owner": "geo",
"size": 0,
"src": "/home/geo/.ansible/tmp/ansible-tmp-1444051174.15-189094870931130/source",
"state": "file",
"uid": 1000
```

We used the *ansible* command to transfer the local file `/home/geo/test` to the remote node in user's home directory (`~`).

Managing Packages

Another use case is installing or upgrading packages on the remote nodes. You can use the *apt* module to achieve this on Debian based systems or the *yum* module on Red Hat based systems:

```
ansible 84.33.2.70 -m apt -a "name=apache2 state=present"
```

For example the previous command will install the Apache web server on the remote system (if not present).

You can use the same module to make sure a package is at the latest version:

```
ansible 84.33.2.70 -m apt -a "name=apache2 state=latest"
```

Managing Services

Use the *service* module to ensure a given service is started on all web servers:

```
ansible webserver -m service -a "name=httpd state=started"
```

(where *webserver* is a group defined in Ansible Inventory)

Restart the service:

```
ansible webserver -m service -a "name=httpd state=restarted"
```

Or stop it:

```
ansible webserver -m service -a "name=httpd state=stopped"
```

For more information on ad-hoc command refer to the [official documentation](#).

These were just a few of the modules available for Ansible. See the [complete list](#) available at the Ansible web site.

Ansible Playbooks

Playbooks are Ansible's configuration, deployment and orchestration language.

Playbooks are a completely different way to use Ansible than in ad-hoc task execution mode, and are particularly powerful.

Playbooks can declare configurations, but they can also orchestrate steps of any manual ordered process.

While you might run the main `/usr/bin/ansible` program for ad-hoc tasks, playbooks are more likely to be kept in source control and used to push out your configuration or assure the configurations of your remote systems are in spec.

Playbooks language example

Playbooks are expressed in [YAML](#) format

Here is an example of a Playbook:

```
---
- hosts: webservers
  vars:
    http_port: 80
    max_clients: 200
  remote_user: root
  tasks:
    - name: ensure apache is at the latest version
      yum: pkg=httpd state=latest
    - name: write the apache config file
      template: src=/srv/httpd.j2 dest=/etc/httpd.conf
      notify:
        - restart apache
    - name: ensure apache is running (and enable it at boot)
      service: name=httpd state=started enabled=yes
  handlers:
    - name: restart apache
      service: name=httpd state=restarted
```

Every Playbook begins with three dashes at the very top of the file to indicate that this is a YAML file.

This example Playbook contains only one Play. The play is composed of three parts:

- hosts
- tasks
- handlers

The hosts part specifies to which hosts in the Inventory this playbook applies and how to connect to them.

The tasks part describes the desired state or actions to perform on the hosts.

The handlers part describes the handlers for this playbook (more on handlers later).

In the example above there are three tasks. Each task has a name, a module and zero or more arguments for the module.

The first task specifies that we want the latest version of Apache installed on the system. This is accomplished by the *yum* module.

The second task specifies a configuration file for Apache using a *template*. Template files are written in [Jinja2 template language](#).

The third task make sure the Apache web server is running using the *service* module.

When you run a Playbook using the `ansible-playbook` command, Ansible will connect to the hosts specified in the hosts section and run the tasks one by one, in order.

One or more tasks may have a *notify* section (just like the second task in our example). The *notify* actions are triggered at the end of each block of tasks in a playbook, and will only be triggered once even if notified by multiple different tasks. When triggered, the corresponding handler will be executed. In the example above the handler will restart Apache because we changed a config file.

Run a Playbook

Now that we have created a sample Playbook, save it on the file system and execute it:

```
ansible-playbook test.yml -u geo

PLAY [84.33.2.70] *****

GATHERING FACTS *****
ok: [84.33.2.70]

TASK: [test] *****
ok: [84.33.2.70]

PLAY RECAP *****
84.33.2.70          : ok=2    changed=0    unreachable=0    failed=0
```

This concludes our brief tutorial on Ansible. For a more thorough introduction refer the [official documentation](#).

Also, take a look at the [Ansible examples repository](#) or a set of Playbooks showing common techniques.

3.2.1.2.4 GeoNode (v2.8) on Docker

This part of the documentation describes the complete setup process for GeoNode on Docker.

Docker installation and setup

[Docker](#) is a free software platform used for packaging software into standardized units for development, shipment and deployment.

Note: credits to Docker

Introducing main concepts

A container image is a lightweight, stand-alone, executable package of a piece of software that includes everything needed to run it: code, runtime, system tools, system libraries, settings.

Docker containers running on a single machine share that machine's operating system kernel; they start instantly and use less compute and RAM.

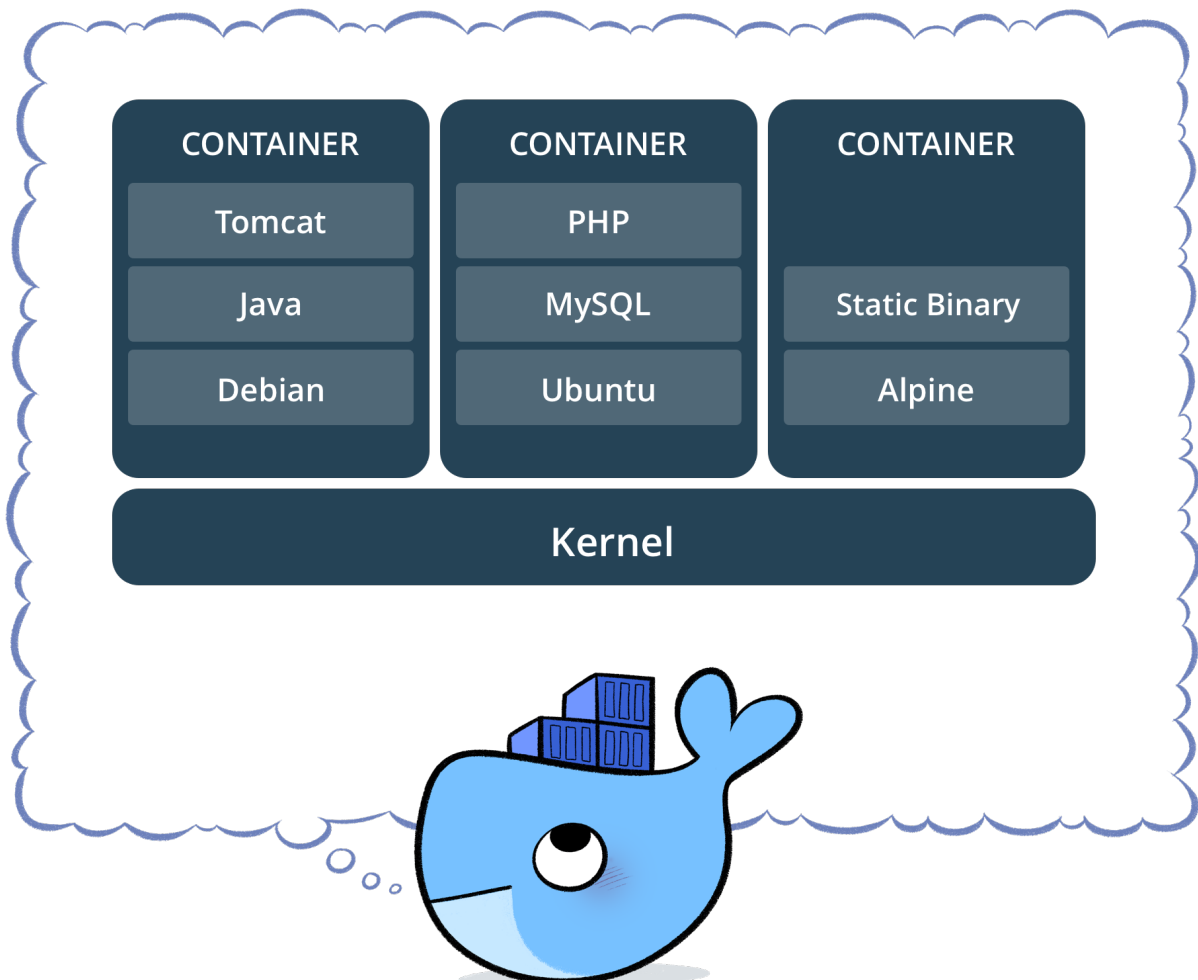
Containers can share a single kernel, and the only information that needs to be in a container image is the executable and its package dependencies, which never need to be installed on the host system.

Multiple containers can run on the same machine and share the OS kernel with other containers, each running as isolated processes in user space.

This tutorial will introduce the use of Docker community edition on Ubuntu 16.04. The same instructions can be applied to CentOS accordingly.

Install Docker CE on Ubuntu

Docker CE is supported on Ubuntu on x86_64 , armhf , s390x (IBM Z), and ppc64le (IBM Power) architectures.



Warning: Make sure to check the OS version as one among supported ones

Show your OS details running:

```
uname -a
```

Uninstall old docker versions

if old versions of Docker binary were installed then uninstall them:

```
sudo apt-get remove docker docker-engine docker.io
```

Install docker

The package of **Docker CE** is now called `docker-ce`. Before doing the installation steps please make sure that the `apt` package index has been updated:

```
sudo apt-get update
```

Add packages to allow the use of secure http channel:

```
sudo apt-get install \
    apt-transport-https \
    ca-certificates \
    curl \
    software-properties-common
```

Add the official GPG key from Docker:

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

Run the following command to setup the **stable** repository:

```
sudo add-apt-repository \
    "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
    $(lsb_release -cs) \
    stable"
```

Update the package index:

```
sudo apt-get update
```

Install the latest version of the binary or a specific version with the command:

```
sudo apt-get install docker-ce # latest
sudo apt-get install docker-ce=<VERSION> # specific
```

The docker daemon will start automatically.

Add your user to the `docker` group if you want to run docker command without `sudo` privileges:

```
sudo usermod -aG docker $USER
source $HOME/.bashrc
```

Verify the health of your installation by running the sample `hello-world` image:

```
docker run hello-world
```

The following message has to be displayed if everything is working properly:

```
Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
   (amd64)
3. The Docker daemon created a new container from that image which runs the
   executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it
   to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash
```

Orchestrate GeoNode stack with Docker Compose

Docker Compose is a tool for defining and running multi-container Docker applications automatically. This tool relies on **Docker** daemon so make sure you have completed successfully the section on `_setup_docker`.

Introducing main concepts

In a composed application, different pieces of the application are called **services**. A service runs just one image but it specifies how that image is executed, what ports have to be used, which containers are depending each other, which volumes and so on.

A **stack** is a group of interrelated services that share dependencies, and can be orchestrated and scaled together. A single stack is capable of defining and coordinating the functionality of an entire application like GeoNode.

Install Docker Compose on Ubuntu

Download the latest version of `docker-compose` binary:

```
sudo curl -L \
https://github.com/docker/compose/releases/download/1.19.0/docker-compose-`uname -s`-
↪`uname -m` \
-o /usr/local/bin/docker-compose
```

Adjust executable permissions to the binary:

```
sudo chmod +x /usr/local/bin/docker-compose
```

Verify the installation:

```
docker-compose --version
```

Running GeoNode stack on localhost

Clone the repository:

```
git clone https://github.com/GeoNode/geonode.git
```

Launch the stack with the build of GeoNode so any changes you did will be immediately available:

```
docker-compose -f docker-compose.yml -f docker-compose.override.localhost.yml up --  
↪build
```

Note: `docker-compose.override.localhost.yml` contains environment variables to allow you run the instances on *localhost*. You can use this file as template in order to run on other public addresses.

Note: For Windows users: In case you're using the native Docker for Windows (on Hyper-V) you will probably be affected by an error related to mounting the `/var/run/docker.sock` volume. It's due to a [problem with the current version of Docker Compose](#) for Windows. In this case you need to set the `COMPOSE_CONVERT_WINDOWS_PATHS` environmental variable:

```
set COMPOSE_CONVERT_WINDOWS_PATHS=1
```

before running `docker-compose up`

Running GeoNode stack on docker ip address

If you want to navigate the application from the address of the Docker socket than run simply:

```
docker-compose up --build
```

Note: For Windows users: In case you're using the native Docker for Windows (on Hyper-V) you will probably be affected by an error related to mounting the `/var/run/docker.sock` volume. It's due to a [problem with the current version of Docker Compose](#) for Windows. In this case you need to set the `COMPOSE_CONVERT_WINDOWS_PATHS` environmental variable:

```
set COMPOSE_CONVERT_WINDOWS_PATHS=1
```

before running `docker-compose up`

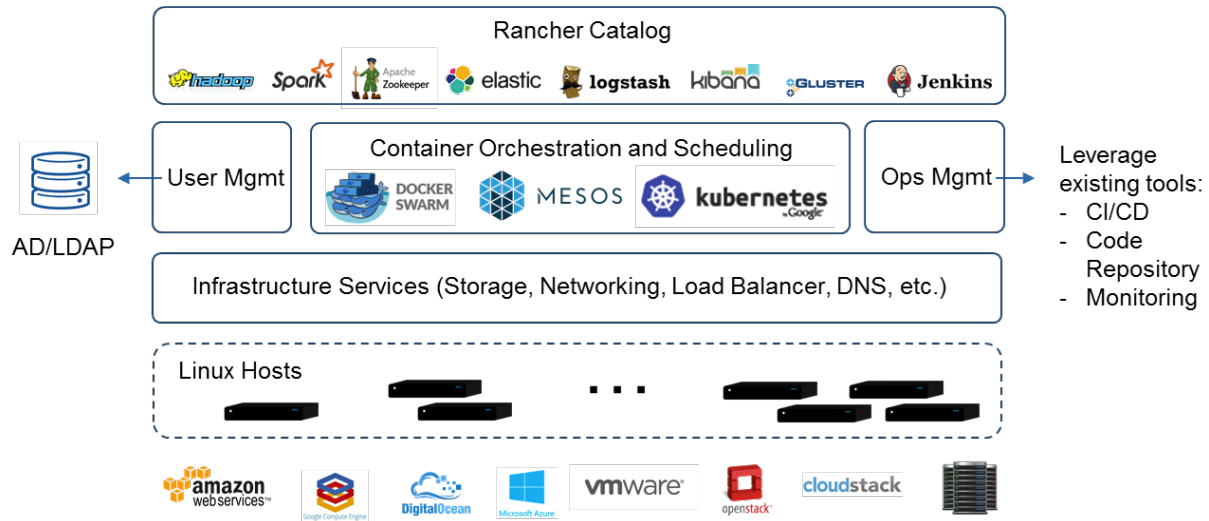
GeoNode will be available at the ip address of the `docker0` interface:

```
ifconfig -a
```

Scaling and deploy with Rancher

[Rancher](#) is an open source software platform that enables organizations to run and manage Docker and Kubernetes in production.

With Rancher, organizations no longer have to build a container services platform from scratch using a distinct set of open source technologies. Rancher supplies the entire software stack needed to manage containers in production.



Introducing main concepts

Rancher takes in raw computing resources from any public or private cloud in the form of Linux hosts. Each Linux host can be a virtual machine or physical machine.

Rancher includes a distribution of all popular container orchestration and scheduling frameworks today, including **Docker Swarm**, **Kubernetes**, and **Mesos**.

Rancher users can deploy an entire multi-container clustered application from the application catalog or docker compose files with few clicks.

Getting started with Rancher

Install Rancher

To install the latest stable version run the following docker command:

```
docker run -d --restart=unless-stopped -p 8080:8080 rancher/server:stable
```

Setup Rancher

Verify that Rancher server is running on the port defined above (default is 8080):

```
docker ps -a
```

After opening the browser at that port:

then the instance can be secured navigating the url [Local authentication](#) :

Choose a `username` and a `password` and then enable the access control by clicking the button at the bottom.

Default
STACKS
CATALOG
INFRASTRUCTURE
ADMIN
API

Before adding your first service or launching a container, you'll need to add a Linux host with a supported version of Docker. Add a host

User Stacks
Add Stack

Sort By:
State
Name

Adding your first Stack

A service is simply a group of containers created from the same Docker image but extends Docker's 'link' concept to leverage Rancher's lightweight distributed DNS service for service discovery. Services can be added individually or by deploying an item from the Catalog.

A service is also capable of leveraging other Rancher built-in services such as load balancers, health monitoring, upgrade support, and high availability. [Learn More](#)

Define a Service
Browse Catalog

v1.6.14
Help
Documentation
File an Issue
Forums
Slack
English
Download CLI

Access Control

Active Directory

Azure AD

GITHUB

LOCAL

OpenLDAP

SHIBBOLETH

Local Authentication is **not configured**

Rancher can be configured to restrict access to a set of accounts defined in the Rancher database. This is not currently set up, so anybody that reach this page (or the API) has full control over the system.

1. Setup an Admin user

This user will become the admin that has full control over Rancher.

Login Username*

e.g. jsmith

Full Name

e.g. John Smith

Password*

Confirm Password*

Add new host to the infrastructure

From the **Default** menu follow the link [add host](#) :

Hosts: Add Host

Host Registration URL

What base URL should hosts use to connect to the Rancher API?

☒ This site's address:

☐ Something else:

Don't include `/v1` or any other path, but if you are doing **SSL termination** in front of Rancher, be sure to use `https://`.

[Save](#)

v1.6.14 [Help](#) [Documentation](#) [File an Issue](#) [Forums](#) [Slack](#) [English](#) [Download CLI](#)

and click the button **Save** to confirm. The administrator will be landed to the following page:

Hosts: Add Host

Custom ☒ Amazon EC2 ☐ Azure ☐ DigitalOcean ☐ Packet ☐

Manage available machine drivers

- 1 Start up a Linux machine somewhere and install a supported version of Docker on it.
- 2 Make sure any security groups or firewalls allow traffic:
 - From and To all other hosts on **UDP** ports **500** and **4500** (for IPsec networking)
- 3 Optional: Add labels to be applied to the host.

[+ Add Label](#)
- 4 Specify the public IP that should be registered for this host. If left empty, Rancher will auto-detect the IP to use. This generally works for machines with unique public IPs, but will not work if the machine is behind a firewall/NAT or if it is the same machine that is running the **rancher/server** container.

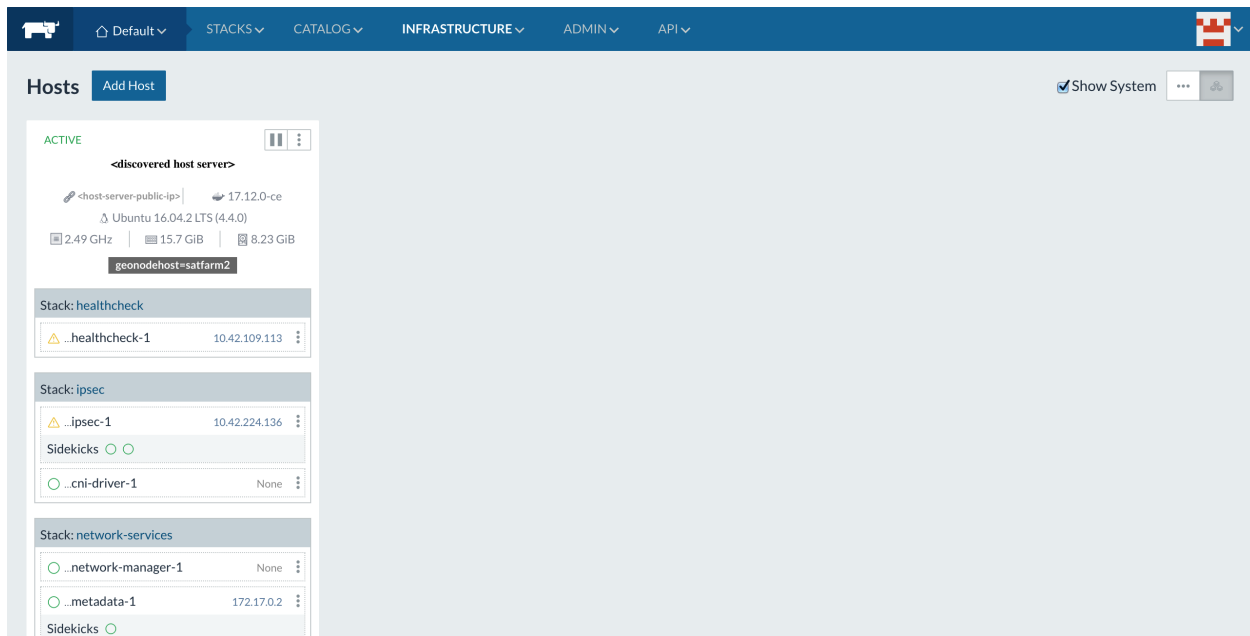
Several options will be shown to decide what kind of machine driver is going to be used. For a local bare metal or virtual machine choose the **Custom** option. After that follow the indicated steps to prepare the new host

1. Start up a Linux machine somewhere and install a [supported version](#) of Docker on it. Refer to the section on `_setup_docker`.
2. Allow IPsec networking between hosts on *UDP* ports *500* and *4500*.
3. Add labels to be applied to the host. This is optional but it is required for GeoNode if you want all the containers of a stack deployed on a same host. In that case add for example *Key:geonodehost* and *Value:<host label value>* which will be translated to the label *geonodehost=<host label value>*.
4. Specify the public IP *<host-server-public-ip>* that should be registered for the host. This is particularly import if the machine is behind a firewall/NAT.
5. Copy, paste, and run the command below to register the host with Rancher. The value of *registrationToken* is provided by the system.

```
sudo docker run -e CATTLE_AGENT_IP="<host-server-public-ip>" -e CATTLE_HOST_LABELS='geonodehost=<host label value>' --rm --privileged -v /var/run/docker.sock:/var/run/docker.sock -v /var/lib/rancher:/var/lib/rancher rancher/agent:v1.2.9 http://<rancher-server-ip>:8080/v1/scripts/<registrationToken> # sudo can be avoided if the user has been privileged
```

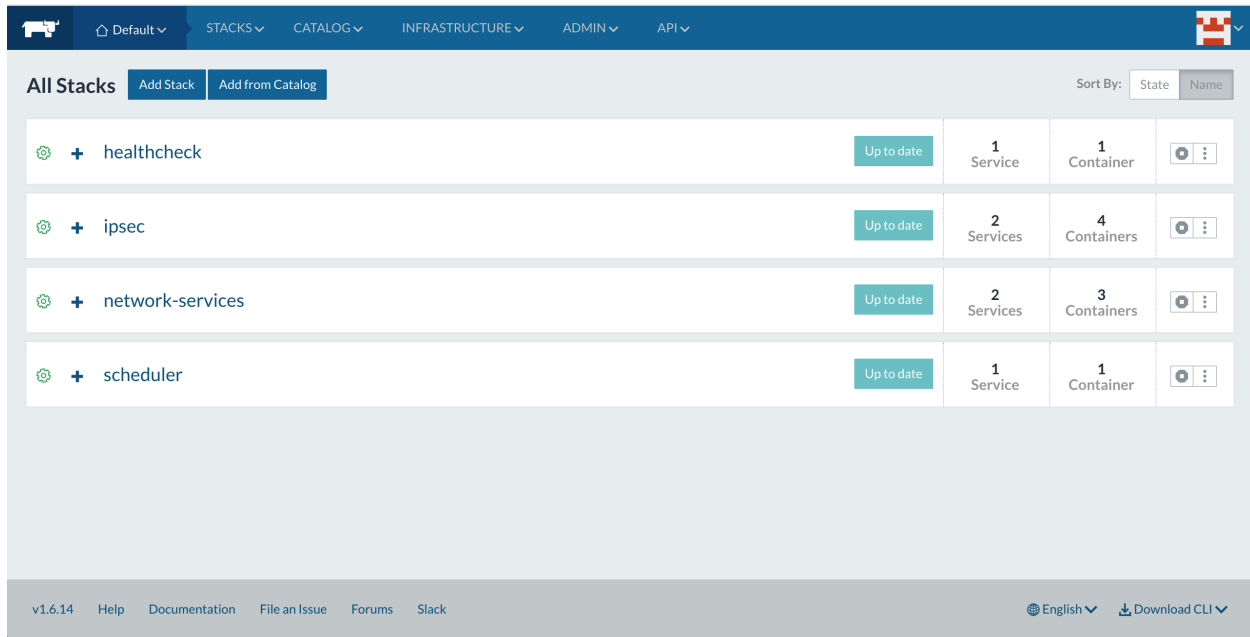
6. Click on the **Close** button.

After some minutes the new host should be registered and available in the active status under the menu **Infrastructure => Hosts**.



Since Rancher itself has containerized services, some of them which are managing the new host have to be available under the menu **Stacks => All**.

Deploy GeoNode as stack on Rancher



The screenshot shows the Rancher UI 'All Stacks' page. The top navigation bar includes links for Default, STACKS, CATALOG, INFRASTRUCTURE, ADMIN, and API. The 'All Stacks' section has buttons for 'Add Stack' and 'Add from Catalog'. A table lists the following stacks:

Stack Name	Status	Services	Containers	Actions
healthcheck	Up to date	1 Service	1 Container	Stop
ipsec	Up to date	2 Services	4 Containers	Stop
network-services	Up to date	2 Services	3 Containers	Stop
scheduler	Up to date	1 Service	1 Container	Stop

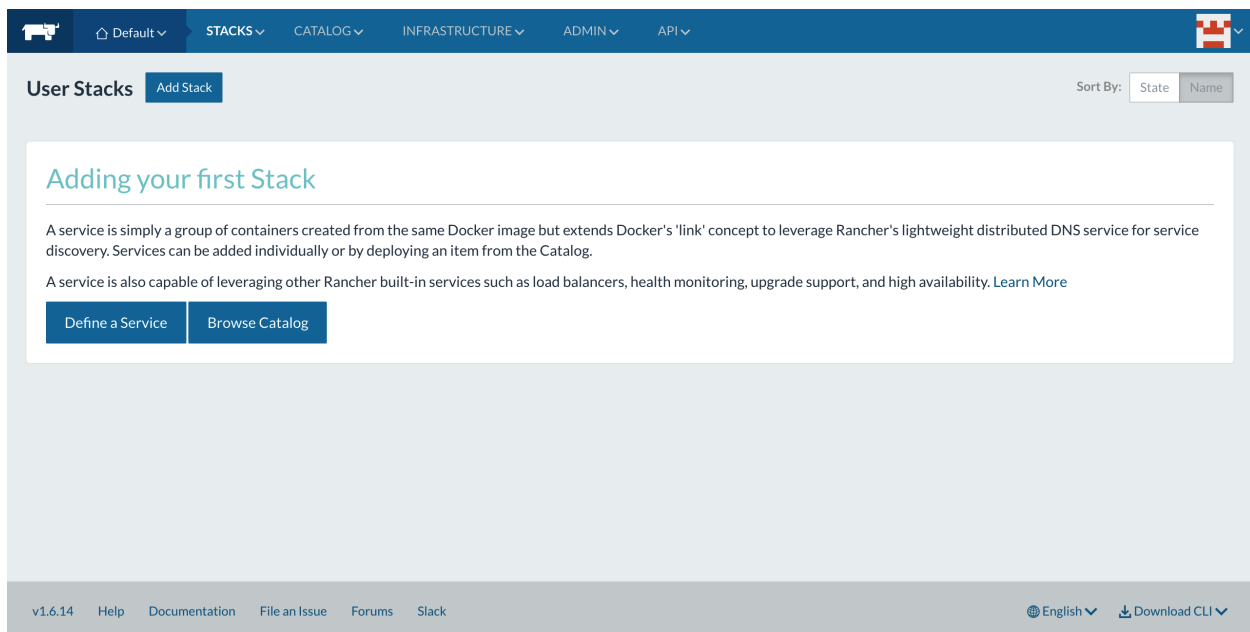
The footer includes version v1.6.14, links for Help, Documentation, File an Issue, Forums, and Slack, and options for English and Download CLI.

Create a stack from a docker-compose file

This is the first option to deploy GeoNode stack from a template like `this docker compose file`.

Before going to create a new stack by loading it from the Rancher console the values of the placeholders `<host label value>` and `<host-server-public-ip>` have to be edited accordingly to your previous setup.

Once the `docker-compose.yml` is ready follow the menu **Stacks => User**:



The screenshot shows the Rancher UI 'User Stacks' page. The top navigation bar is the same as the previous screenshot. The 'User Stacks' section has an 'Add Stack' button. Below this, there is a section titled 'Adding your first Stack' with the following text:

A service is simply a group of containers created from the same Docker image but extends Docker's 'link' concept to leverage Rancher's lightweight distributed DNS service for service discovery. Services can be added individually or by deploying an item from the Catalog.

A service is also capable of leveraging other Rancher built-in services such as load balancers, health monitoring, upgrade support, and high availability. [Learn More](#)

At the bottom of this section are two buttons: 'Define a Service' and 'Browse Catalog'.

The footer is the same as the previous screenshot.

Then click the **Add Stack** button:

Add Stack

Name:

Description:

OPTIONAL: IMPORT COMPOSE

Optional: docker-compose.yml

```
version: "2"
services:

  db:
    image: geonode/postgis:9.6
    restart: unless-stopped
    container_name: db4${COMPOSE_PROJECT_NAME}
    stdin_open: true
    tty: true
    labels:
```

Optional: rancher-compose.yml

Contents of rancher-compose.yml

ADVANCED OPTIONS ^

Create Cancel

From the property **docker-compose.yml** load the file previously edited, then click the **Create** button:

Warning: The first time the host has to provision all the images and can take a while

At the end of the provisioning job all the GeoNode services will be active and the stack available as deployed application:

Stack: MyGeoNode Add Service Active

Description: My first GeoNode stack

Active	celery	Image: geonode/geonode:2.6.x	Service	1 Container	
Started-Once	data-dir-conf	Image: geonode/geoserver_data:2.10.x	Service	1 Container	
Active	db	Image: geonode/postgis:9.6	Service	1 Container	
Active	django	Image: geonode/geonode:2.6.x	Service	1 Container	
Active	elasticsearch	Image: elasticsearch	Service	1 Container	
Active	geonode	Image: geonode/nginx:geoserver Ports: 80	Service	1 Container	
Active	geoserver	Image: geonode/geoserver:2.10.x	Service	1 Container	
Active	rabbitmq	Image: rabbitmq	Service	1 Container	

GeoNode is now available at the public ip address defined into the `GEONODE_LB_HOST_IP` variable of your docker-compose file.

As usual the default credentials for the master administrator is *admin/admin*.

3.2.1.2.5 VM Setup with VirtualBox

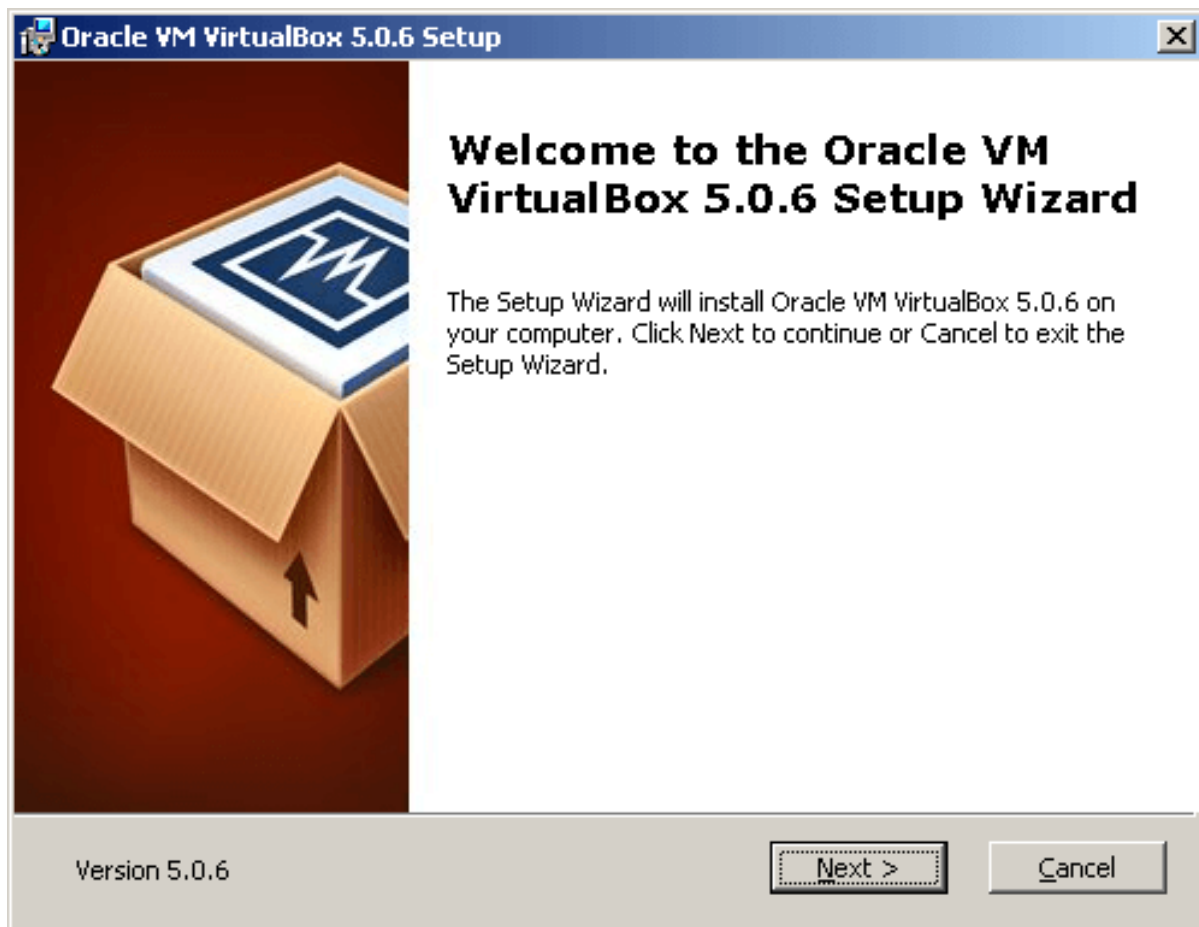
In this section you will find instructions on how to setup an Ubuntu 14.04 VM in [VirtualBox](#)

VirtualBox Setup

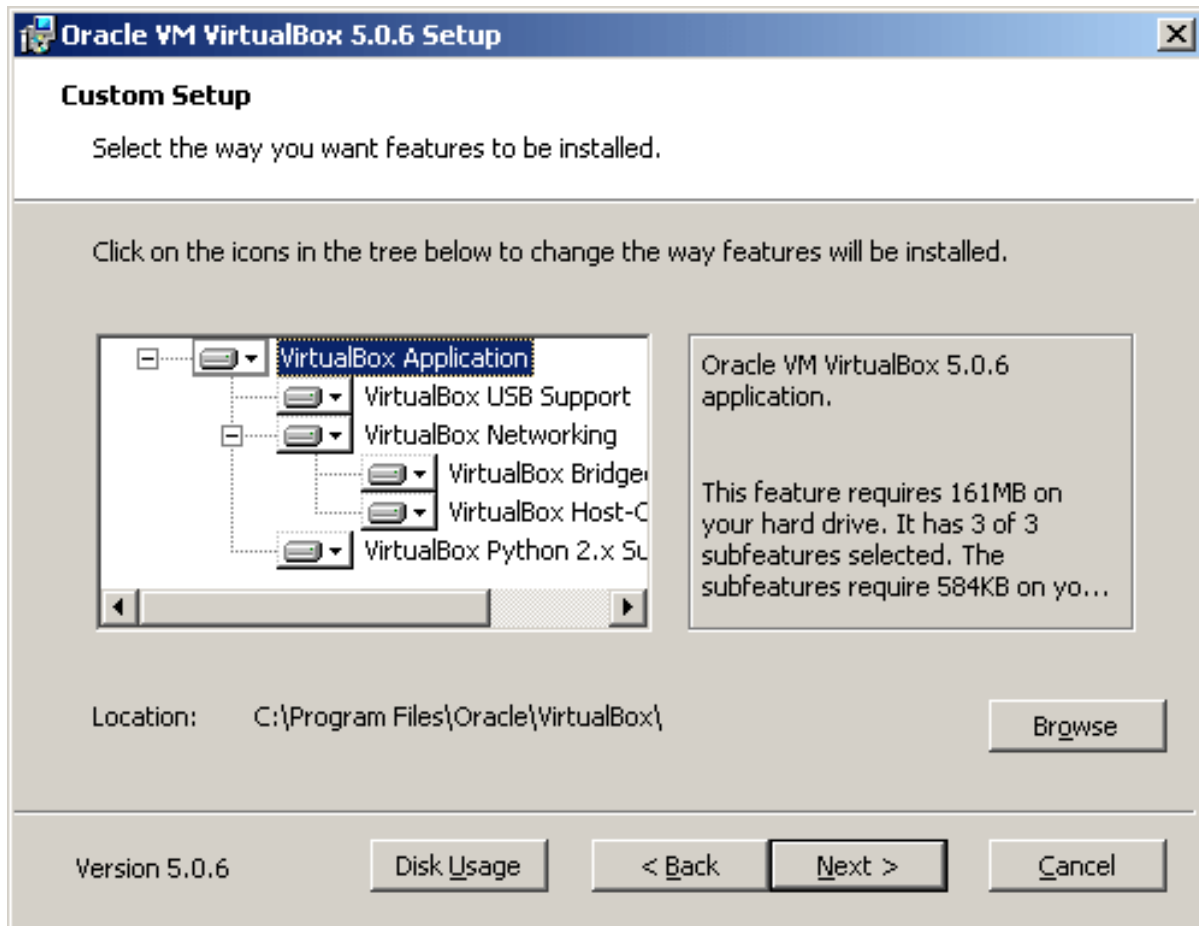
Download VirtualBox from [official](#) web site. Choose the installer matching your operating system and architecture. Installation process is straightforward, refer to VirtualBox [official documentation](#) if you encounter any problem.

Windows

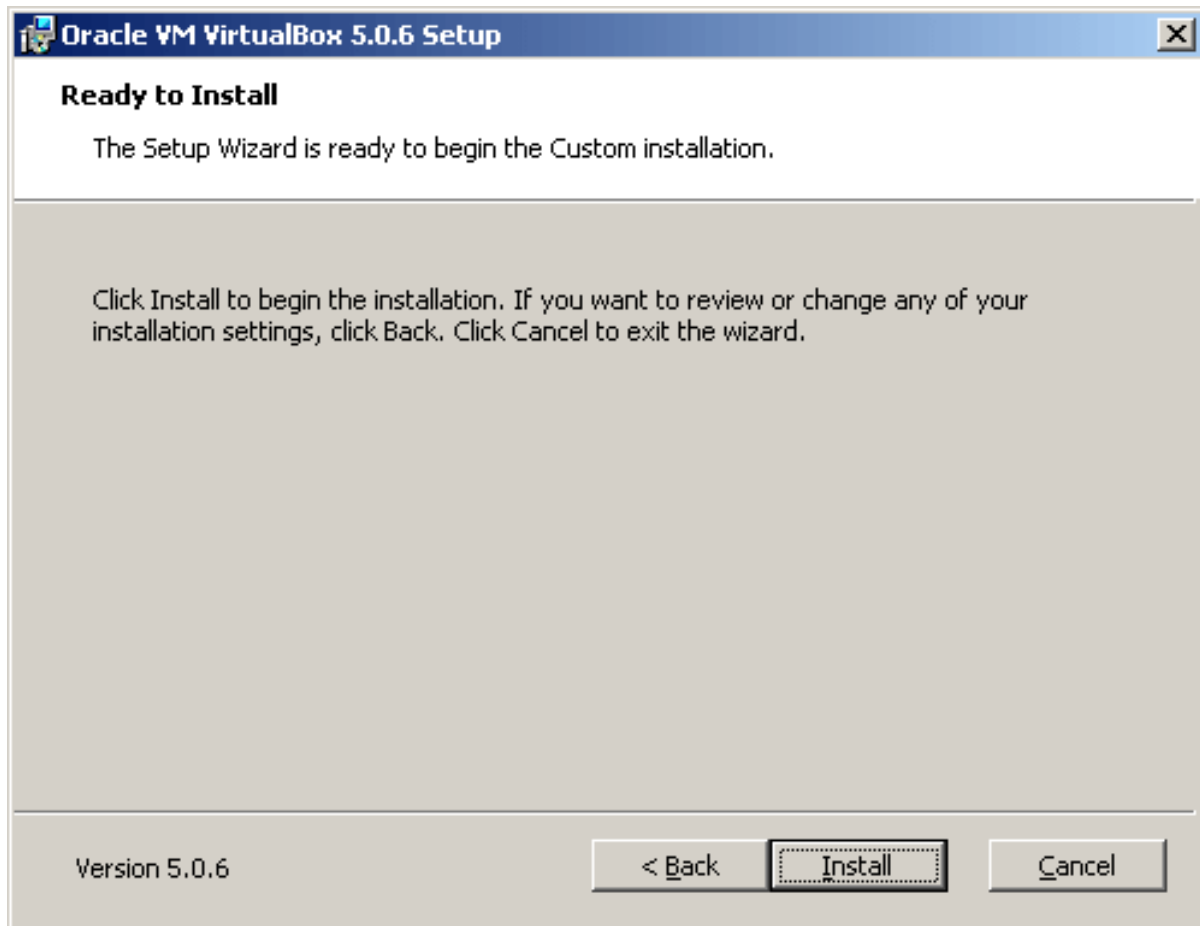
After you downloaded executable, double click on it to launch the installer.

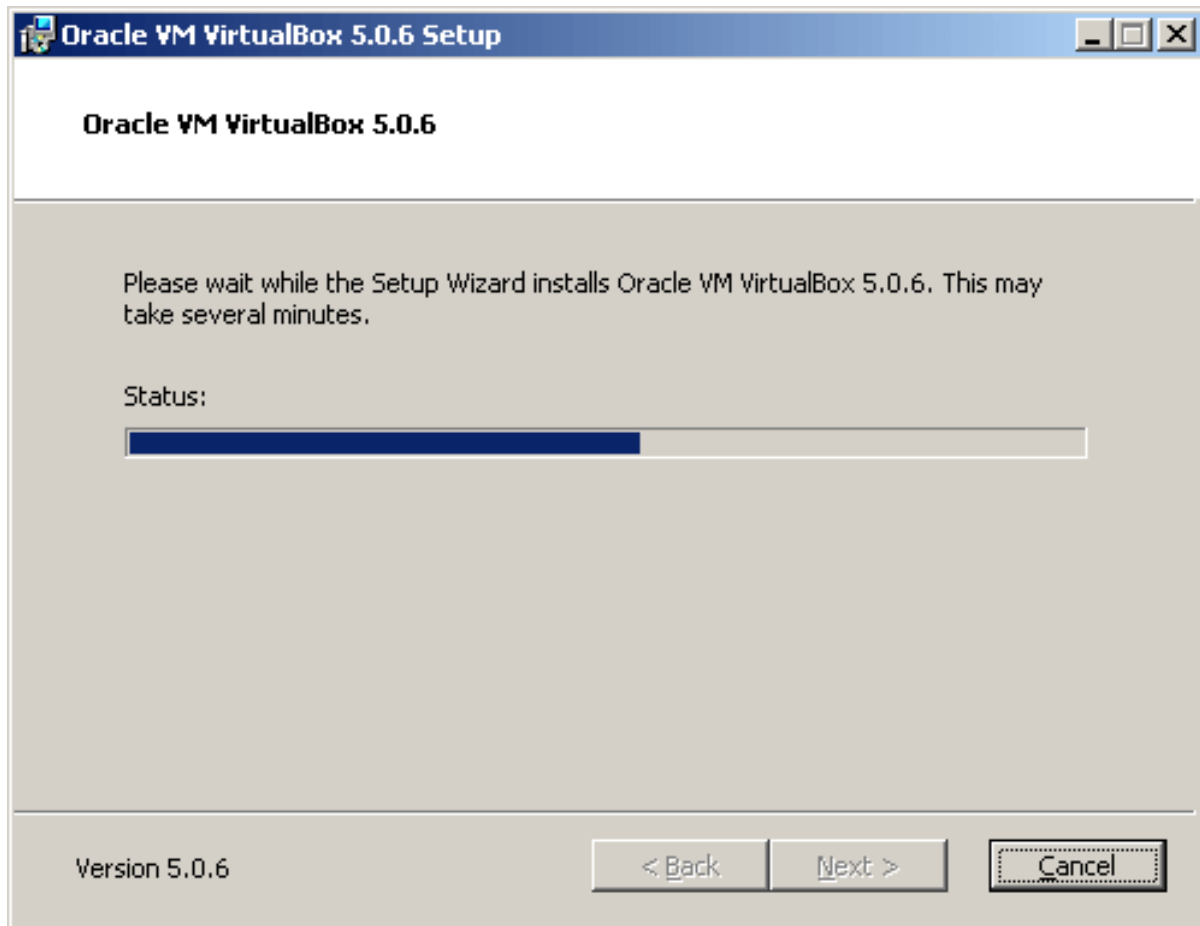


Customize VirtualBox features and paths if you need to or leave default ones



And start the installation process





Click on “Finish”



VirtualBox is now installed. And will automatically be launched

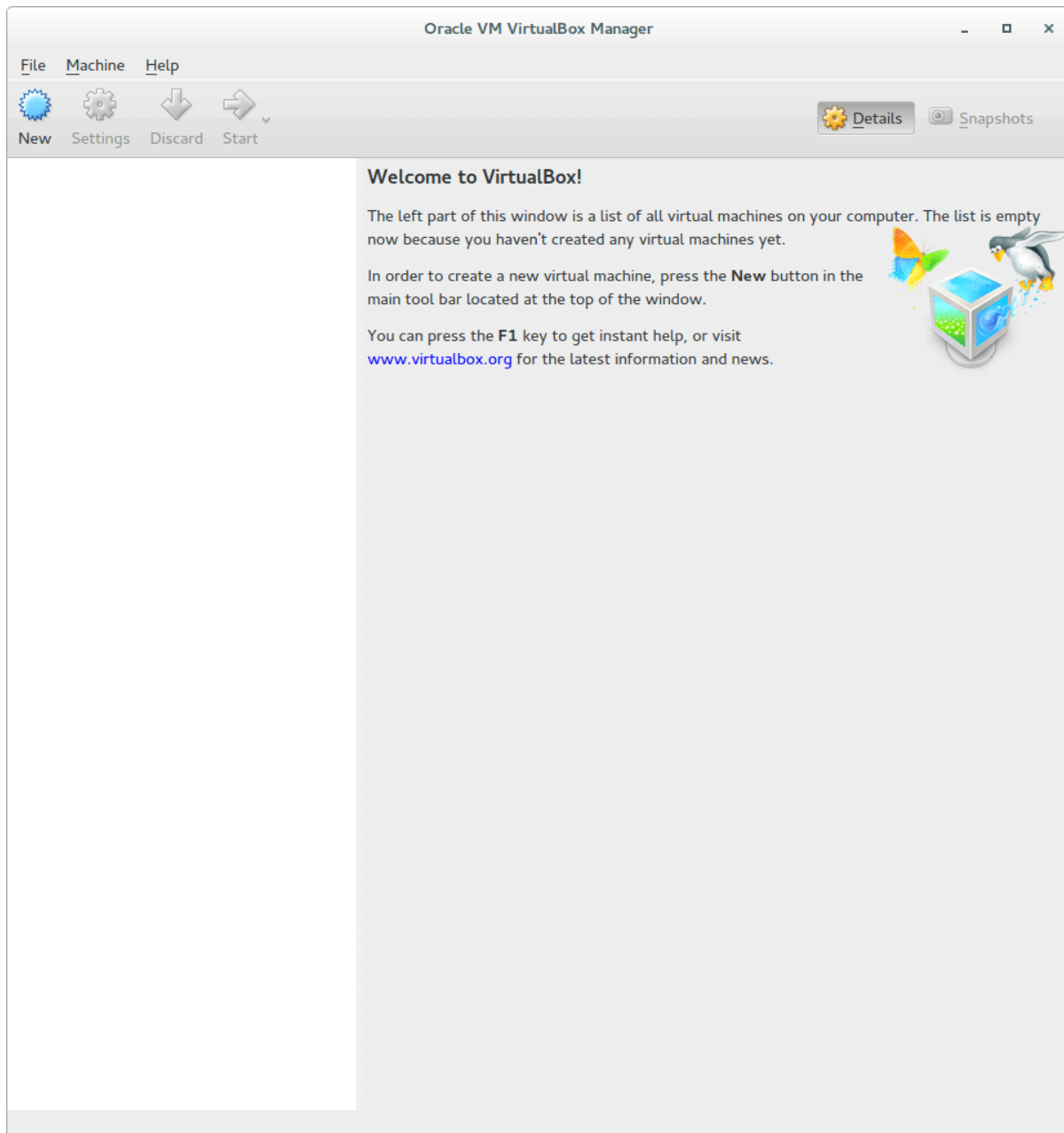
Ubuntu

After you downloaded the package, double click on it. The “Ubuntu Software Center” will pop up, click on “Install” to start the installation process



You will be prompted for administrator password.

At the end of the installation process, launch VirtualBox.



Download Ubuntu ISO

Navigate to [Ubuntu official Download page](#) and download the Ubuntu 14.04 *.iso* file for your architecture. The ISO image will be used to install Ubuntu 14.04 inside the Virtual Machine

Virtual Machine Setup

Now that VirtualBox is installed on the system it is time to setup our Ubuntu VM.

Click the light blue *New* button in VirtualBox user interface.

Choose a name for the Virtual Machine and select the appropriate VM type and version

Create Virtual Machine



Name and operating system

Please choose a descriptive name for the new virtual machine and select the type of operating system you intend to install on it. The name you choose will be used throughout VirtualBox to identify this machine.

Name:

Type: 

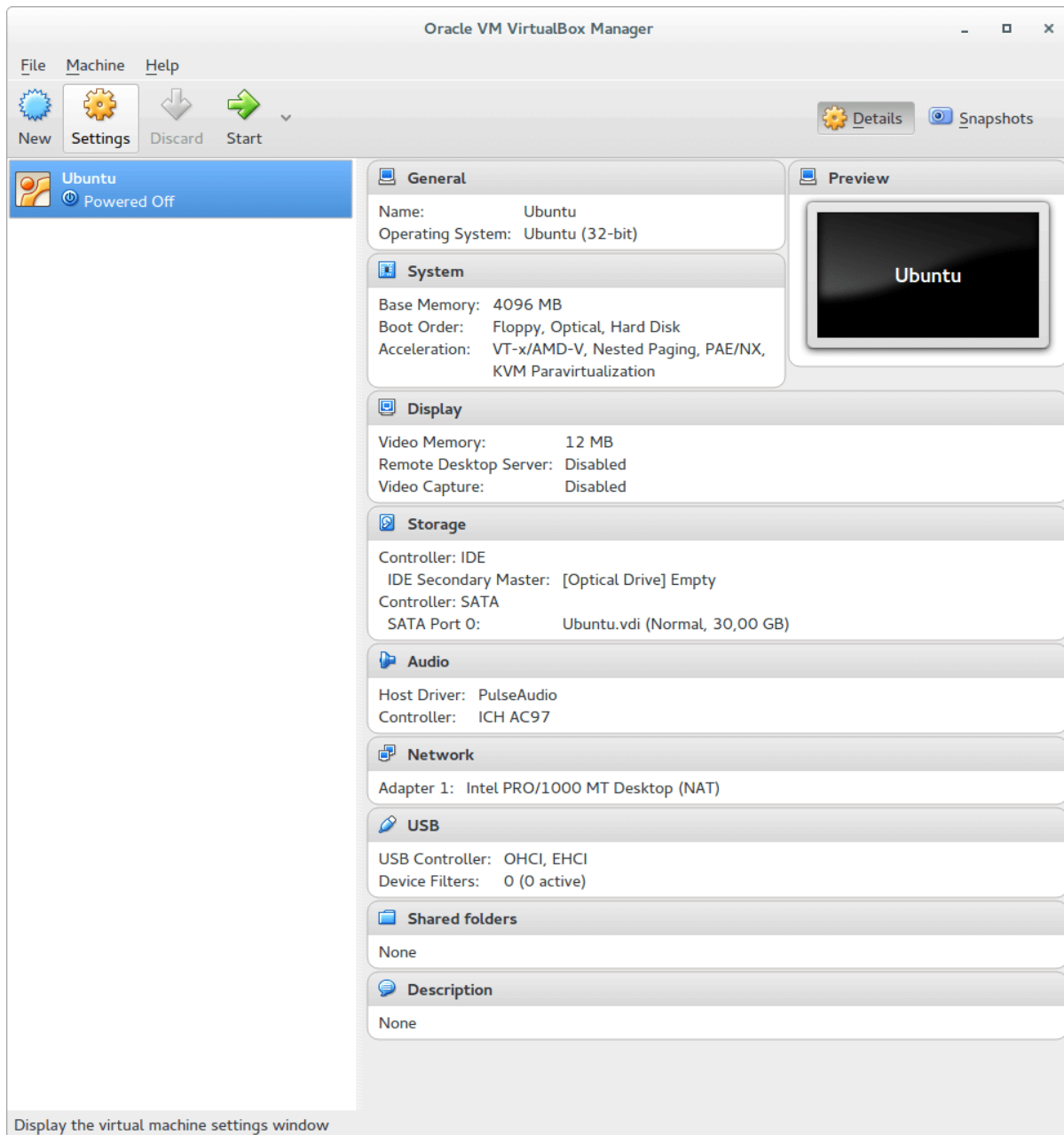
Version:

Then select the amount of memory you want to assign to the VM, [Ubuntu recommends](#) at least 512 MB of memory but we are going to need more than that to run GeoNode refer to *System Preparation & Prerequisites* sections for details.

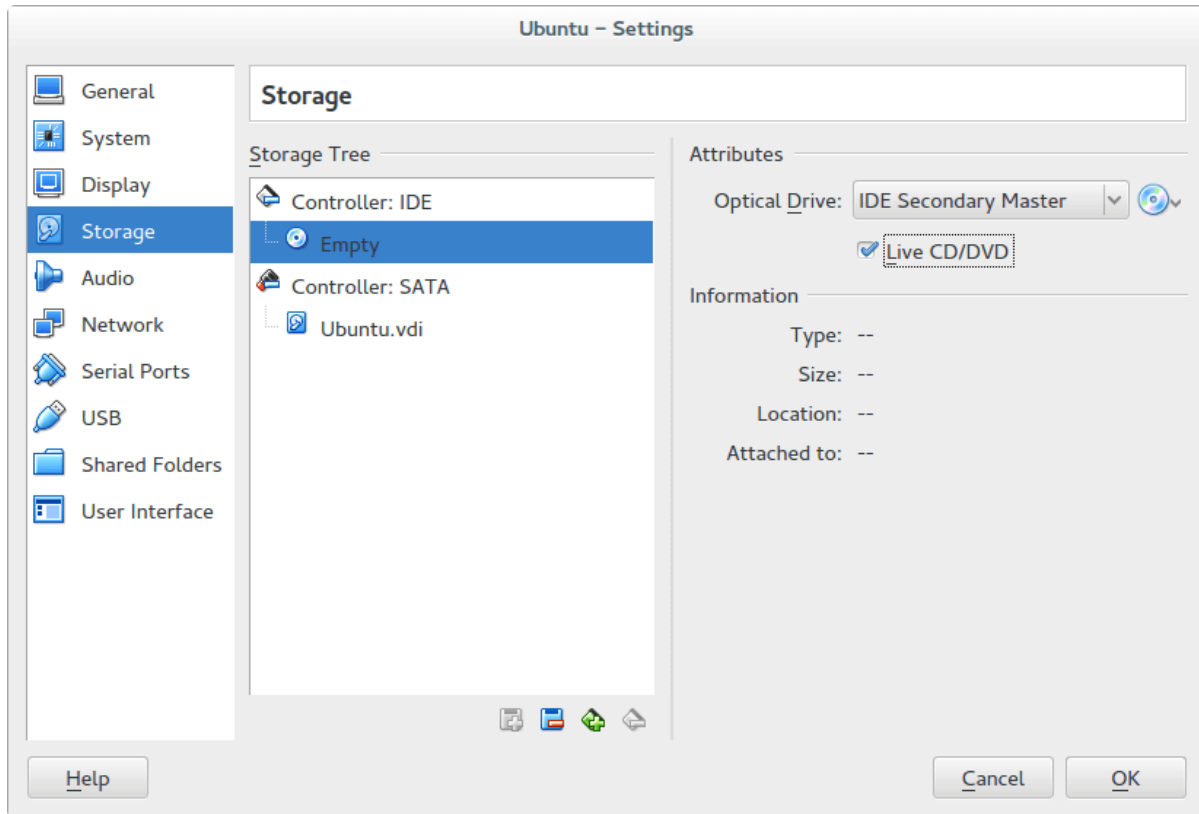


Create a new *virtual disk* for the VM. Again, refer to *System Preparation & Prerequisites* section for details about disk size, for testing purposes 30 GB will be enough.

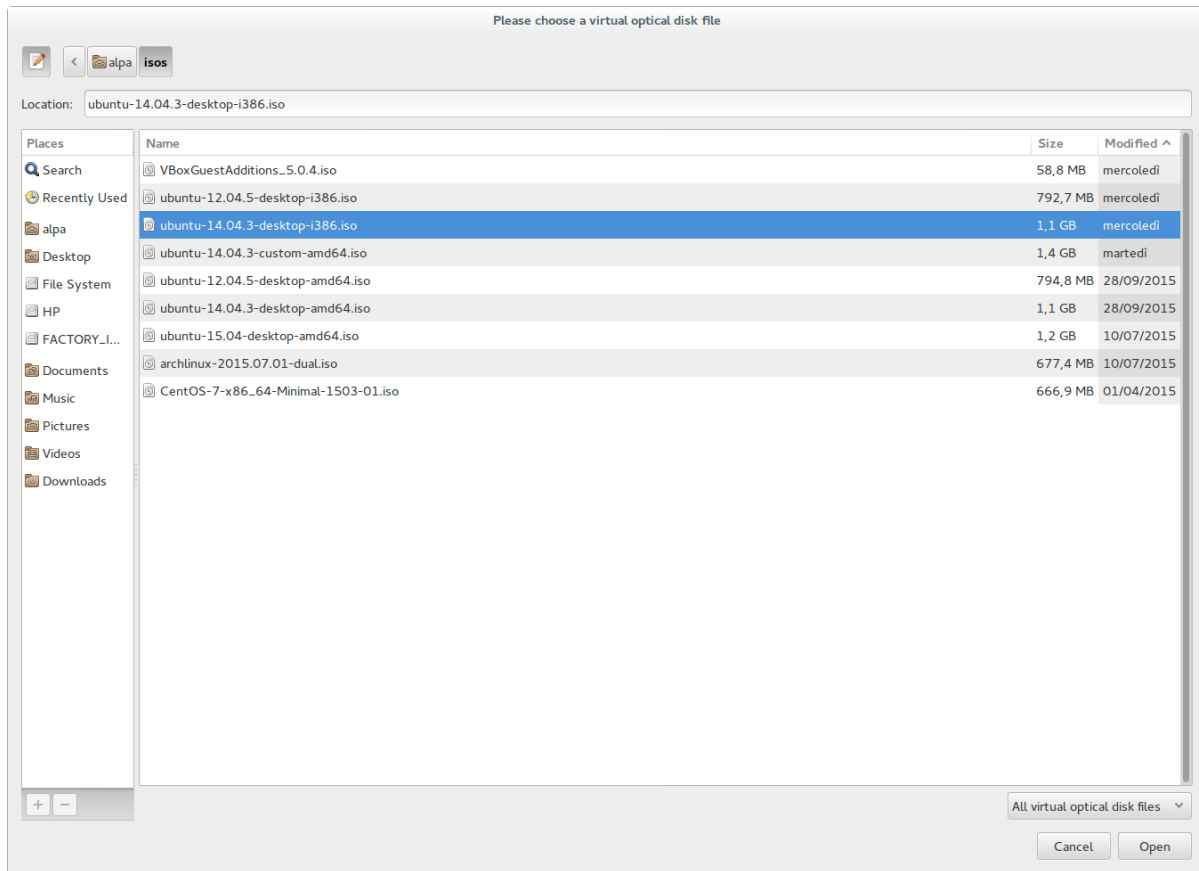
Now edit the Virtual Machine settings



Under “Storage” select the empty DVD drive, click on *Live CD/DVD* as shown below

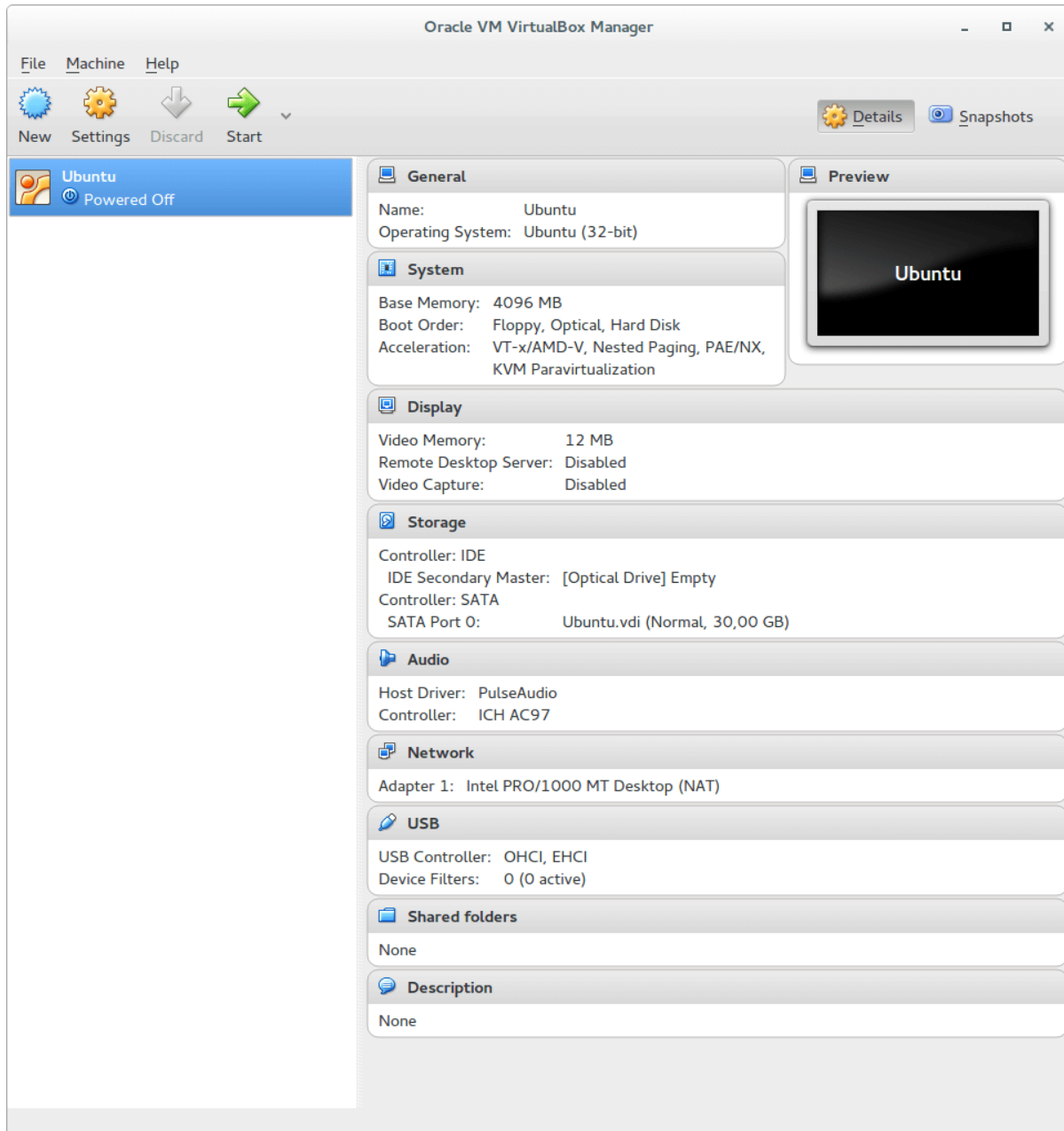


Click on the DVD icon next to the *Optical Drive* drop down menu and select the Ubuntu 14.04 *.iso* file that you downloaded before

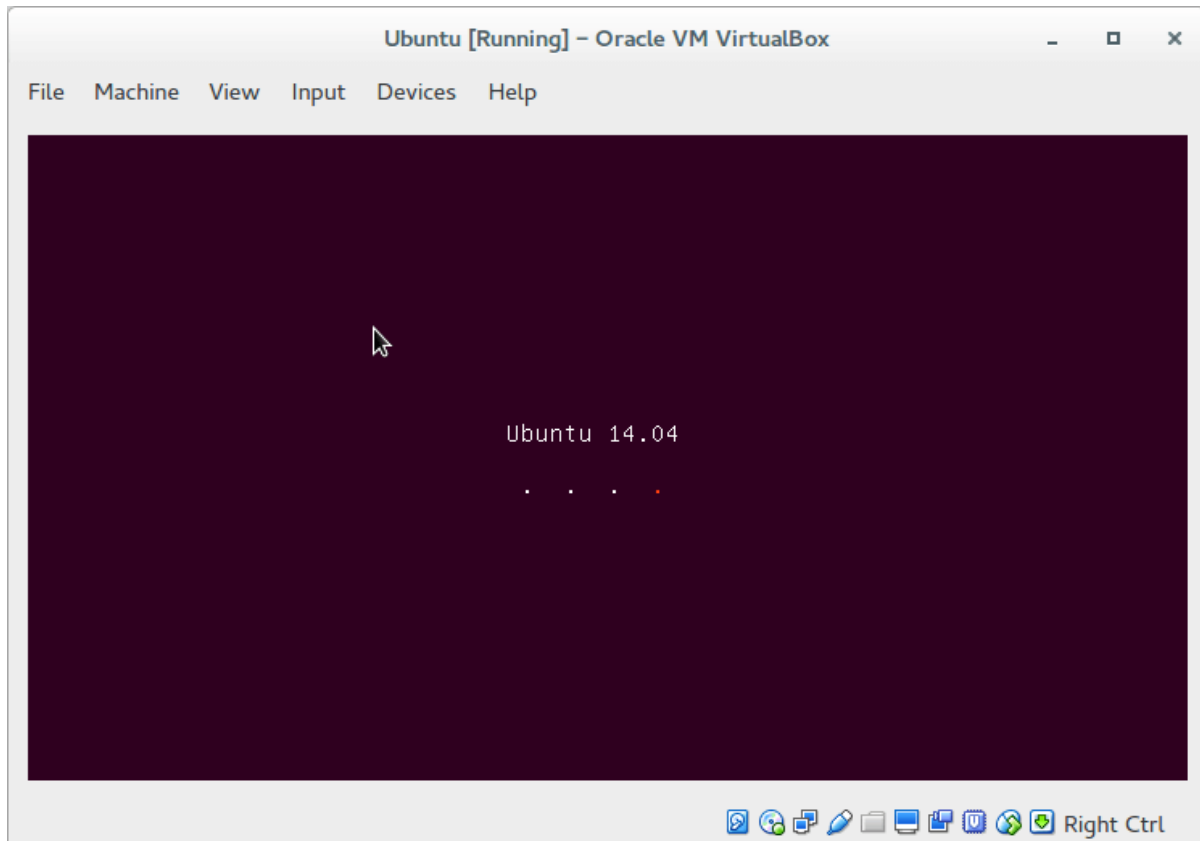


Edit other VM setting if you need to, then click *OK*.

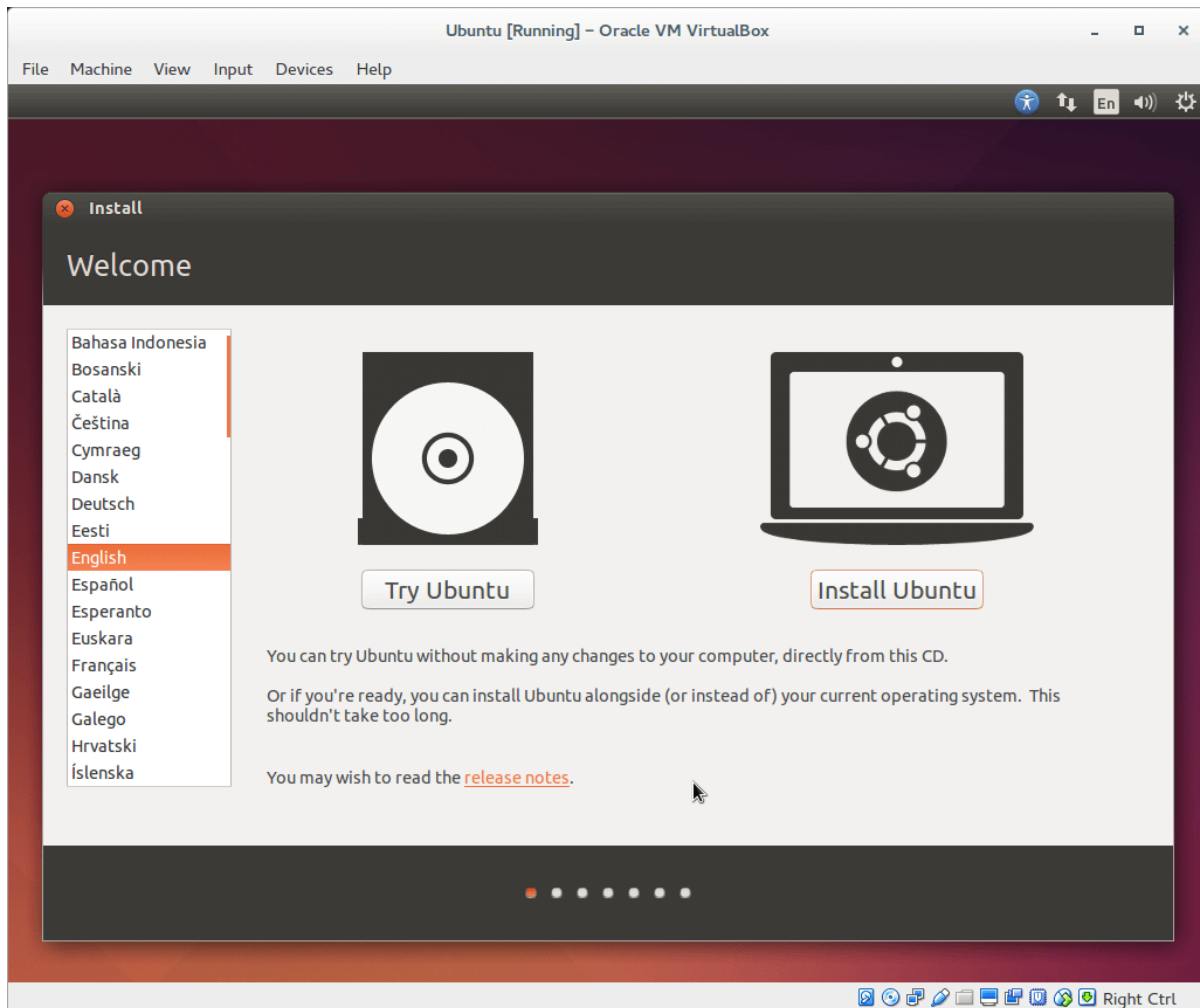
We are ready to start our Ubuntu VM for the first time. Select it from the main menu and click on *Start*



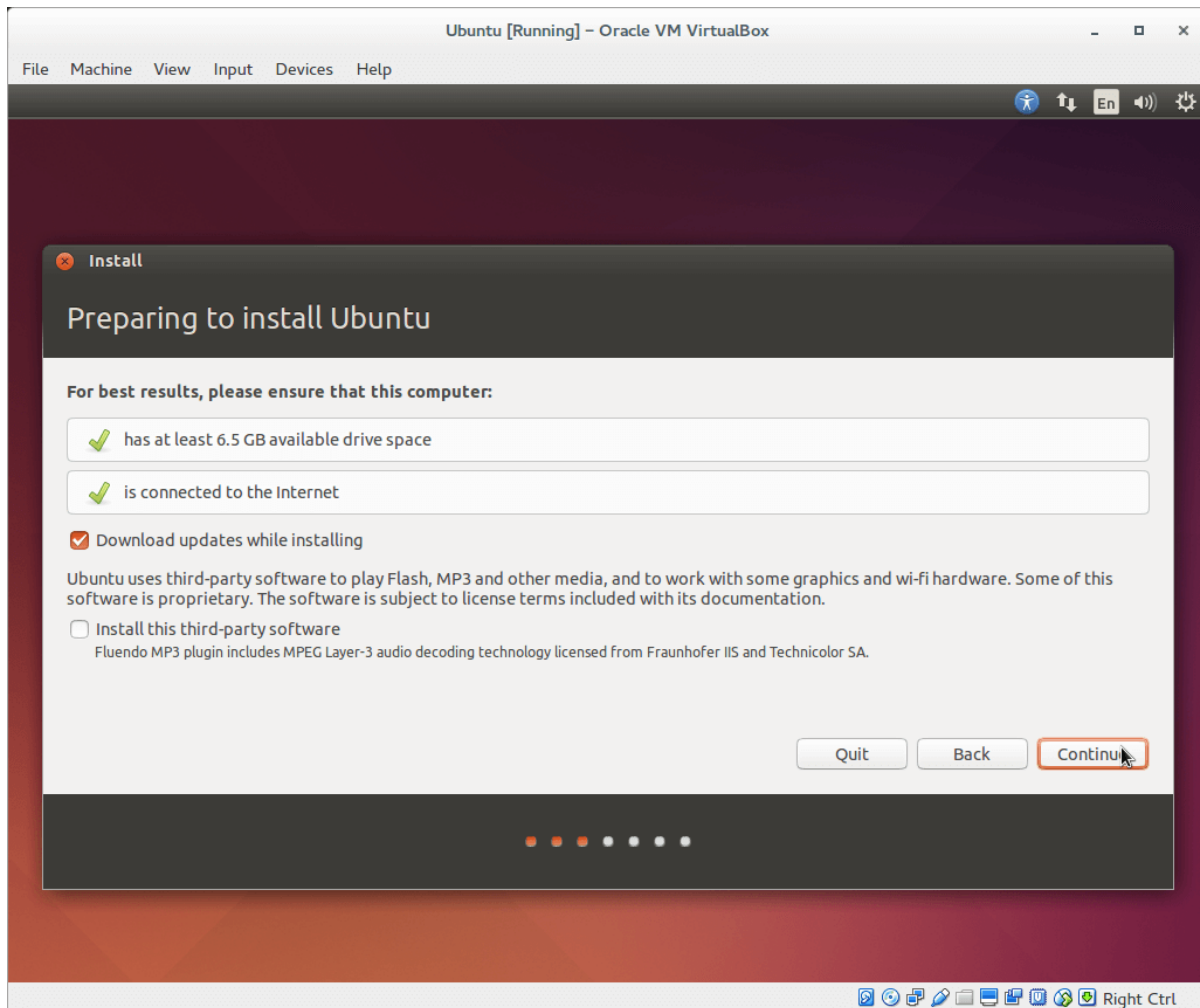
Ubuntu will start the boot process



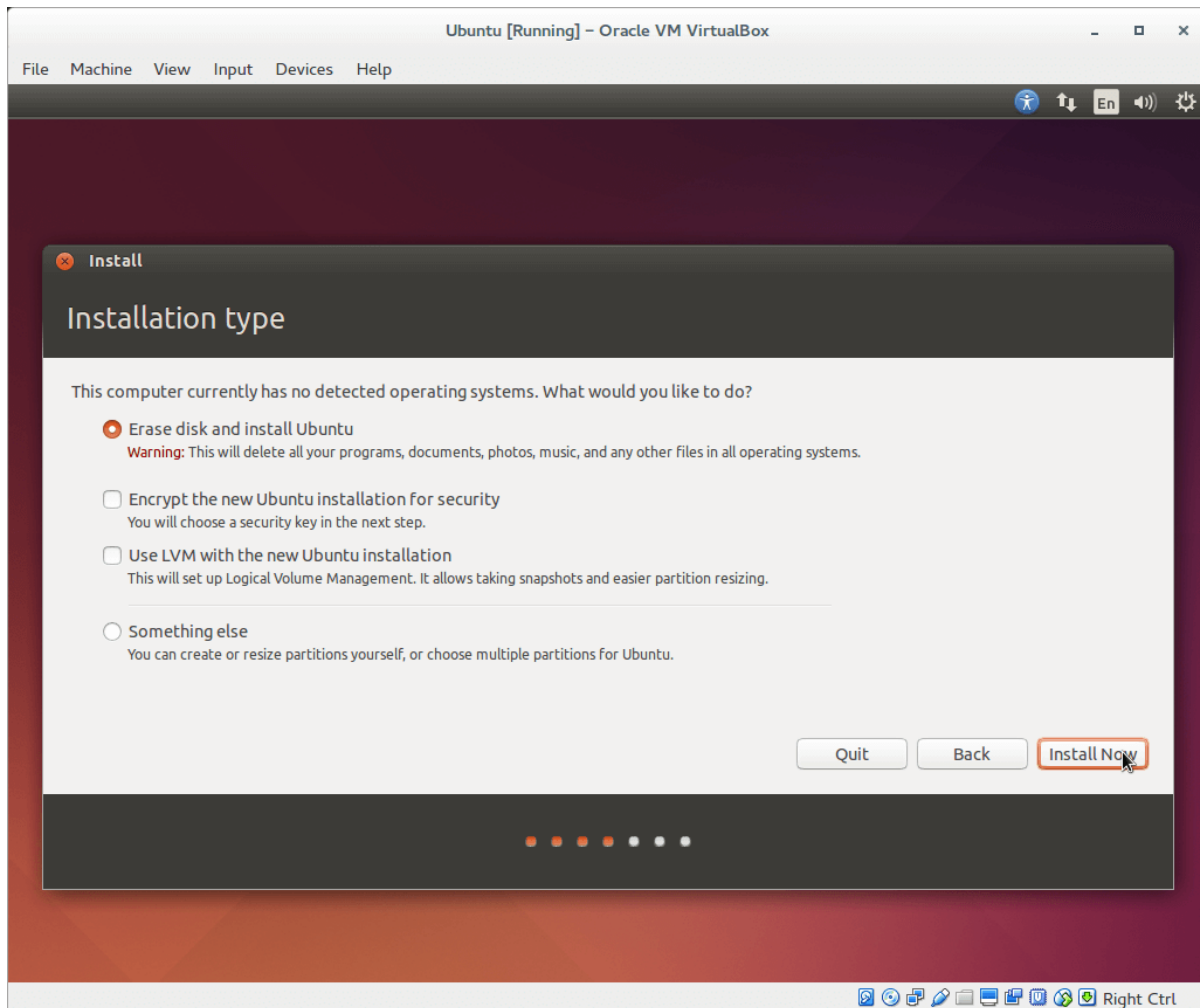
At the end of the boot process you will be asked if you want to *Try Ubuntu* or *Install Ubuntu*. Select the language in the left panel and click on *Install Ubuntu*



The installer will check your internet connection and available disk space. If you are connected to the internet check the *Download updates while installing* checkbox.

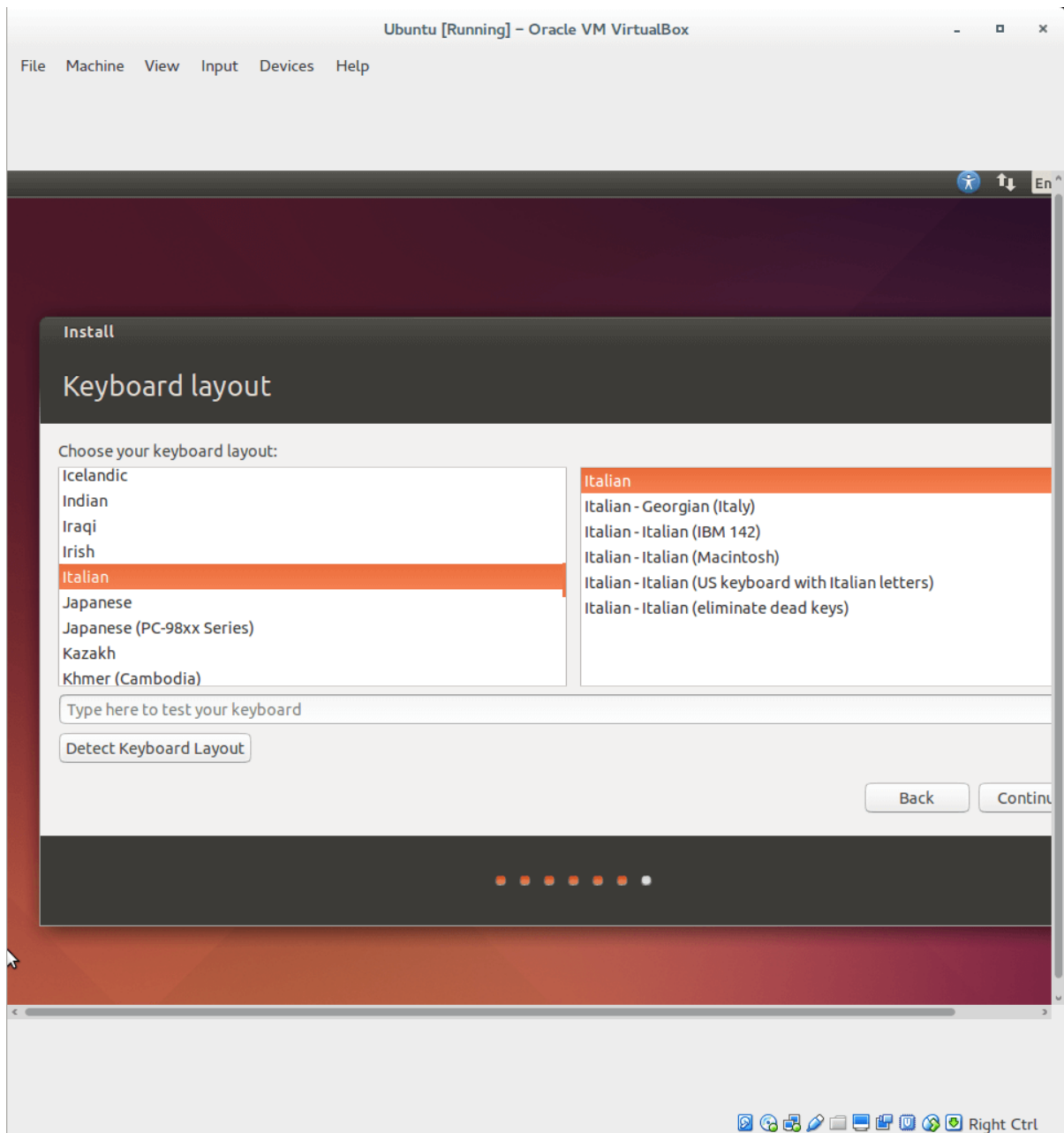


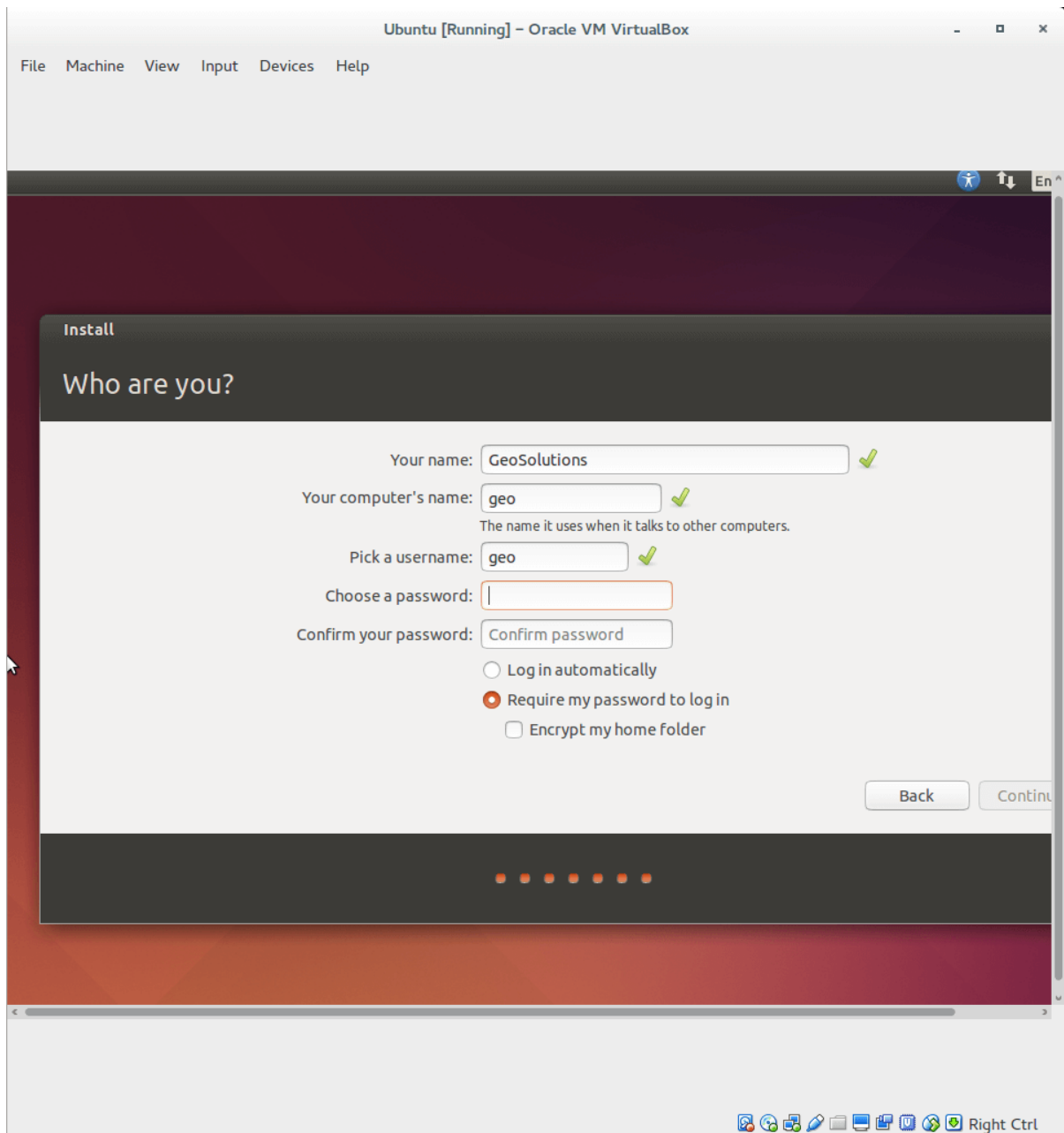
Click on *continue*. In the page you will configure the partitioning of the disks. If you recall we have created a new *virtual disk* during the VM configuration process for Ubuntu. We are going to assign the entire disk to it. Select *Erase disk and install Ubuntu*, then *Install Now*

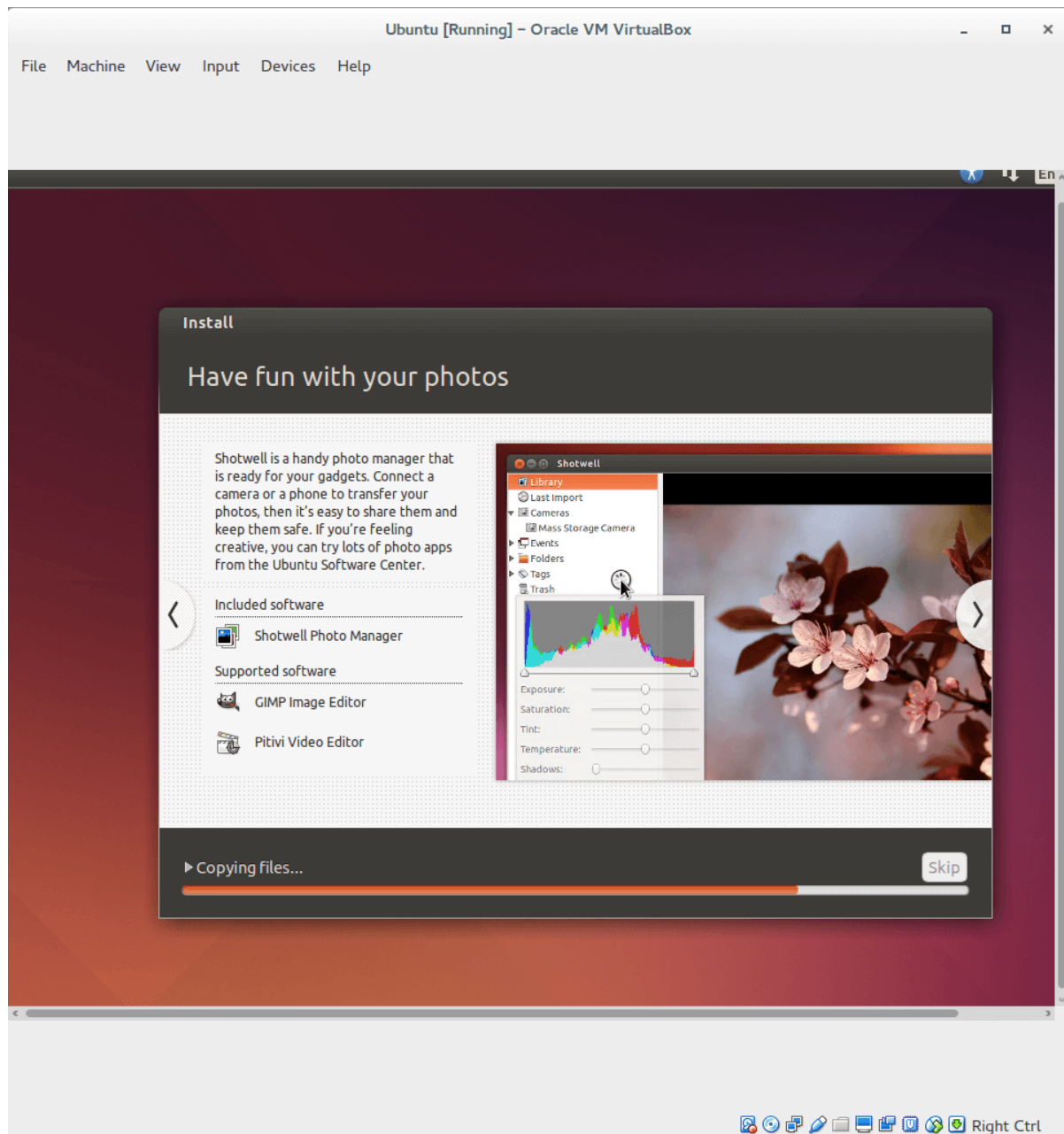


You will be prompted for confirmation.

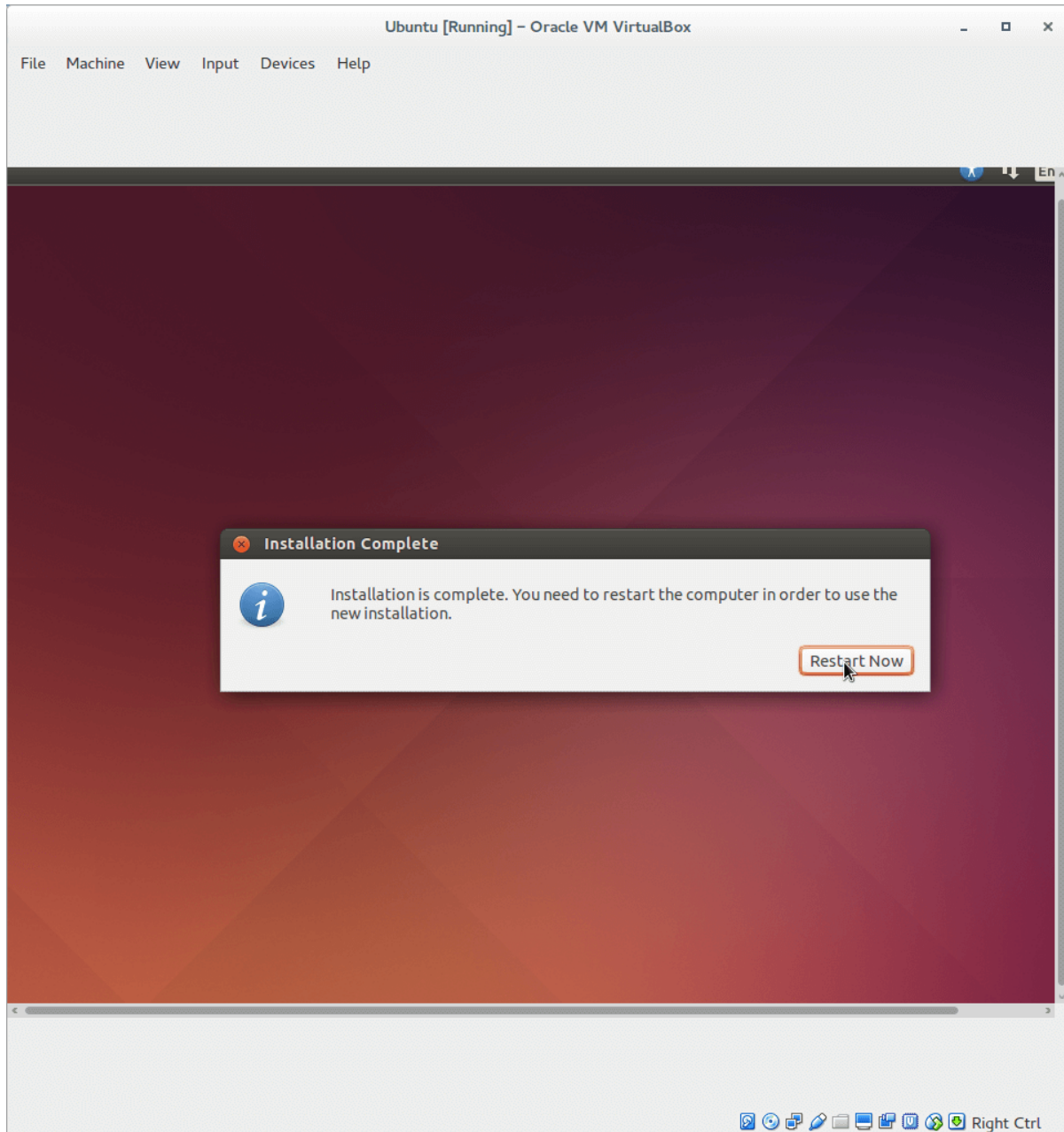
Now select the correct time zone for your location, then select the language for the VM and enter the details for the administrator user.







The installation will continue automatically. At the end of the installation process a pop up window will ask you to restart the system to start using Ubuntu. Click on *Restart Now*



3.2.1.2.6 Running a VM with Vagrant

In this section you will find instructions on how to setup an Ubuntu 16.04 VM using [Vagrant](#)

Vagrant Setup

Download Vagrant from the [official](#) web site. Choose the installer matching your operating system and architecture. Installation process is straightforward, refer to Vagrant [official documentation](#) if you encounter any problems. At the end of the installation process log out your system and log back in.

Vagrant is going to need a [provider](#) in order to setup the Virtual Machines. *VirtualBox* is supported out of the box. Just make sure you install [one of the supported versions of VirtualBox](#)

Open a terminal and type *vagrant version*. A message containing the installed version of Vagrant will be printed on the terminal

```
Installed Version: 2.0.2
Latest Version: 2.0.3
...
```

Virtual Machine Setup

Now that Vagrant is installed let's create our Ubuntu Virtual Machine. Open the terminal and create a new folder called *vagrant*. Within folder *vagrant* create a new file called *Vagrantfile* with following content.

```
Vagrant.configure(2) do |config|

  config.vm.box = "ubuntu/xenial64"

  # To automatically configure a private network uncomment following line.
  # config.vm.network "private_network", ip: "192.168.33.30"

  config.vm.provider "virtualbox" do |vb|
    # Customize the amount of memory on the VM:
    vb.memory = "4024"
  end

  # Port forwarding: If unneeded comment or remove following lines
  config.vm.network "forwarded_port", guest: 80, host: 8001
  config.vm.network "forwarded_port", guest: 8000, host: 8000
  config.vm.network "forwarded_port", guest: 8080, host: 8080

end
```

This *Vagrantfile* containing the settings for the virtual machine, notably the *config.vm.box* variable set to “ubuntu/xenial64” will tell Vagrant the specific VM we want to run (Ubuntu 16.04 “Xenial Xerus”, 64 bit version). Please take note of comments regarding private network setup, amount of memory and port forwarding within this file. Further visit official [vagrant documentation](#) for more explanations on fine tuning your VM.

To finally start the VM, run

```
vagrant up
```

The first time you run the command it is going to take some time since you do not have a locally available image of the Ubuntu 16.04 VM. Vagrant will download the VM from the [Vagrant Cloud](#) to your local system.

```
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Box 'ubuntu/xenial64' could not be found. Attempting to find and install.
...
default: Box Provider: virtualbox
default: Box Version: >= 0
==> default: Loading metadata for box 'ubuntu/xenial64'
default: URL: https://vagrantcloud.com/ubuntu/xenial64
==> default: Adding box 'ubuntu/xenial64' (v20180406.0.0) for provider: virtualbox
default: Downloading:
https://vagrantcloud.com/ubuntu/boxes/xenial64/versions/20180406.0.0/providers/
->virtualbox.box
```


At the end of the download process Vagrant will start the VM.

To access the Virtual machine, run

```
vagrant ssh
```

Note: You need an SSH client for the previous command to work. Most Linux distributions come with an SSH installed. If you are using Windows as the guest operating system install MinGW or Cygwin or Git to obtain a command line SSH client. More information available [here](#)

You will be connected to the guest Virtual Machine over SSH as user *vagrant*. Vagrant provides all files next to your Vagrantfile in a folder called */vagrant* at your VM.

3.2.1.2.7 GeoNode (v2.8) installation on Ubuntu 16.04

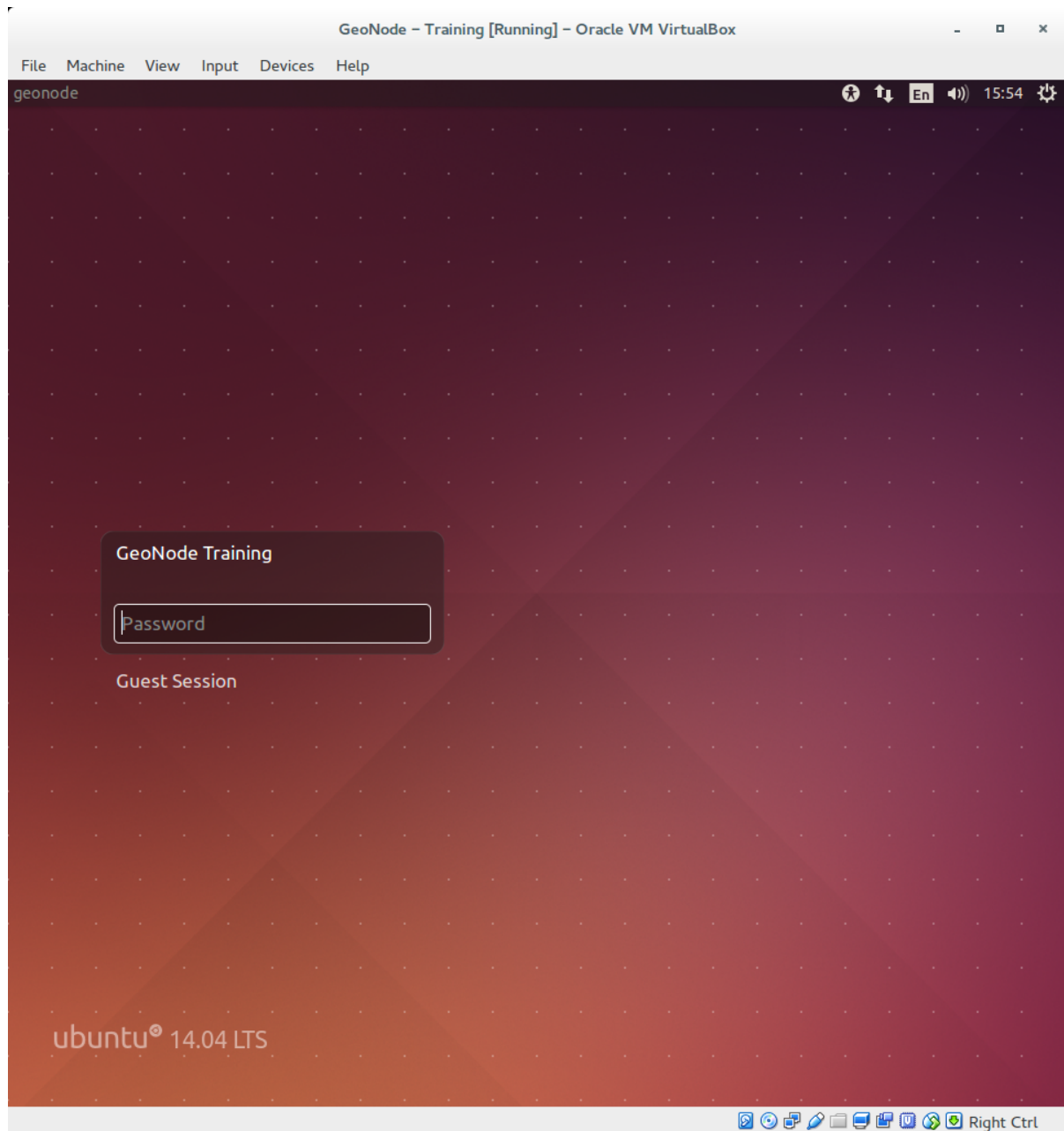
This part of the documentation describes the complete setup process for GeoNode on an Ubuntu 16.04 machine.

Install GeoNode Application

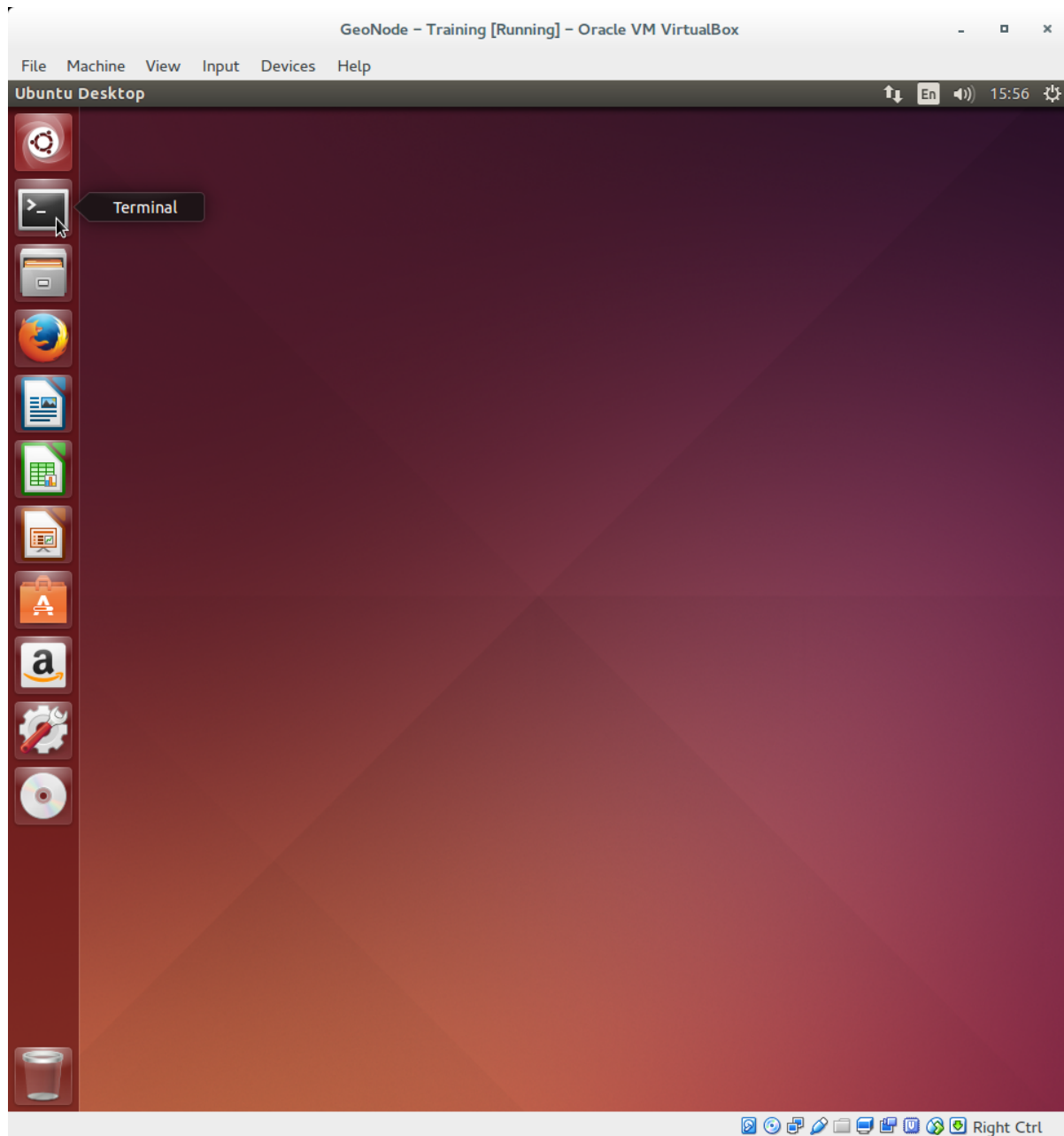
In this section you are going to install all the basic packages and tools needed for a complete GeoNode installation.

Login

When you first start the Virtual Machine at the end of the boot process you will be prompted for the user password to login. Enter *geo* as user password and press *Enter*.



You are now logged in as user 'geo'. On the left side of the screen there is a panel with shortcuts to common applications, launch a the terminal emulator.



Packages Installation

First we are going to install all the software packages we are going to need for the GeoNode setup. Among others *Tomcat 8*, *PostgreSQL*, *PostGIS*, *Apache HTTP server* and *Git*. Run the following command to install all the packages

```
$ sudo apt-get update

$ sudo apt-get install python-virtualenv python-dev libxml2 libxml2-dev libxslt1-dev
↳zlib1g-dev libjpeg-dev libpq-dev git default-jdk
$ sudo apt-get install build-essential openssh-server gettext nano vim unzip zip
↳patch git-core postfix
```

(continues on next page)

(continued from previous page)

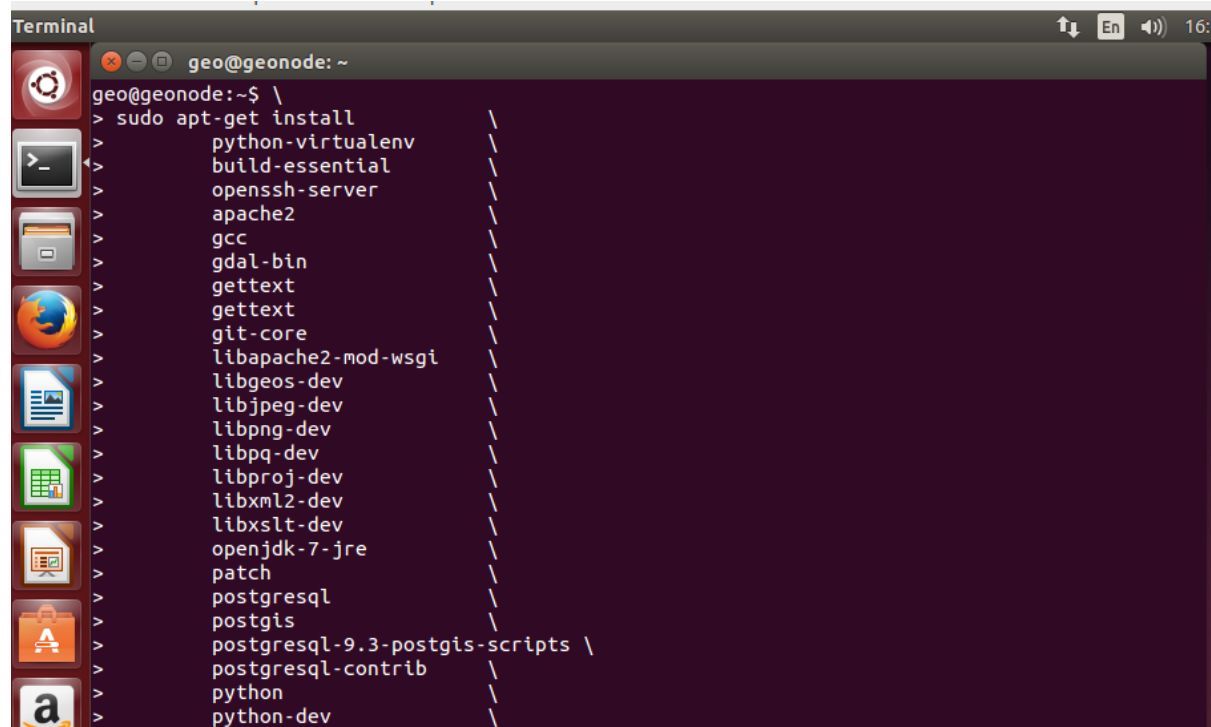
```

$ sudo apt-add-repository ppa:webupd8team/java
$ sudo apt-get update
$ sudo apt-get install oracle-java8-installer

$ sudo apt-add-repository ppa:ubuntugis && sudo apt-get update && sudo apt-get upgrade
$ sudo apt-add-repository ppa:ubuntugis/ppa && sudo apt-get update && sudo apt-get ↵
↵upgrade
$ sudo apt-get install gcc apache2 libapache2-mod-wsgi libgeos-dev libjpeg-dev libpng-
↵dev libpq-dev libproj-dev libxml2-dev libxslt-dev
$ sudo apt-add-repository ppa:ubuntugis/ubuntugis-testing && sudo apt-get update && ↵
↵sudo apt-get upgrade
$ sudo apt-get install gdal-bin libgdal20 libgdal-dev
$ sudo apt-get install python-gdal python-pycurl python-imaging python-pastescript ↵
↵python-psycpg2 python-urlgrabber
$ sudo apt-get install postgresql postgis postgresql-9.5-postgis-scripts postgresql-
↵contrib
$ sudo apt-get install tomcat8

$ sudo apt-get update && sudo apt-get upgrade && sudo apt-get autoremove && sudo apt-
↵get autoclean && sudo apt-get purge && sudo apt-get clean

```



```

Terminal
geo@geonode: ~
geo@geonode:~$ \
> sudo apt-get install
python-virtualenv
build-essential
openssh-server
apache2
gcc
gdal-bin
gettext
gettext
git-core
libapache2-mod-wsgi
libgeos-dev
libjpeg-dev
libpng-dev
libpq-dev
libproj-dev
libxml2-dev
libxslt-dev
openjdk-7-jre
patch
postgresql
postgis
postgresql-9.3-postgis-scripts \
postgresql-contrib
python
python-dev

```

Note: If you will be prompted for *geo* user's password (*geo*) and for confirmation twice

```

GeoNode - Training [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Terminal
geo@geonode: ~
python-html5lib-whl python-numpy python-openid python-paste
python-pastedeploy python-pastedeploy-tpl python-pastescript python-pip
python-pip-whl python-psycpg2 python-pycurl python-requests-whl python-scgi
python-setuptools python-setuptools-whl python-six-whl python-tempita
python-urlgrabber python-urllib3-whl python-virtualenv python-wheel
python2.7-dev ssh-import-id tomcat7 tomcat7-common tzdata-java zlib1g-dev
The following packages will be upgraded:
  libcomerr2 libexpat1 openssh-client tzdata
4 upgraded, 142 newly installed, 0 to remove and 95 not upgraded.
Need to get 0 B/105 MB of archives.
After this operation, 254 MB of additional disk space will be used.
Do you want to continue? [Y/n]
WARNING: The following packages cannot be authenticated!
  libcomerr2 libgssrpc4 libkadm5clnt-mit9 libkdb5-7 libkadm5srv-mit9 libapr1
  libexpat1 libaprutil1 ca-certificates-java tzdata tzdata-java java-common
  openjdk-7-jre-headless libgif4 openjdk-7-jre libatk-wrapper-java
  libatk-wrapper-java-jni libbonobo2-common libidl-common libidl0 liborbit-2-0
  liborbit2 libbonobo2-0 libck-connector0 libdap11 libdapclient3 libepsilon1
  libfreexl1 libgfortran3 libgnomevfs2-common libgnomevfs2-0 libgnome2-common
  libgnome2-bin libgnome2-0 libhdf5-7 mysql-common libmysqlclient18 libodbc1
  libexpat1-dev libpython2.7-dev libgeos-3.4.2 libgeos-c1 proj-data libproj0
  libspatialite5 libxerces-c3.1 odbcinst odbcinstdebian2 ncurses-term
  openssh-client libaprutil1-dbd-sqlite3 libaprutil1-ldap apache2-bin
  apache2-data apache2 authbind fonts-dejavu-extra libblas3 liblapack3
  libarpack2 libarmadillo4 libhdf4-0-alt liburiparser1 libkml0 libnetcdfc7
  libogdi3.2 libpq5 libgdal1h gdal-bin git-core comerr-dev krb5-multidev
  libapache2-mod-wsgi libcommons-collections3-java libcommons-pool-java
  libcommons-dbc-java libecj-java libgconf2-4 libgeos-dev
  libgeronimo-jta-1.1-spec-java libjpeg-turbo8-dev libjpeg8-dev libjpeg-dev
  liblwgeom-2.1.2 liboss-p-uuid16 zlib1g-dev libpng12-dev libssl-dev libpq-dev
  libproj-dev libpython-dev libservlet3.0-java libssl-doc libtomcat7-java
  libxml2-dev libxslt1-dev openssh-sftp-server openssh-server postgres
  postgresql-client-common postgresql-client-9.3 postgresql-common
  postgresql-9.3 postgresql postgresql-9.3-postgis-scripts
  postgresql-9.3-postgis-2.1 postgresql-contrib-9.3 postgresql-contrib
  proj-bin python-chardet-whl python-colorama python-colorama-whl
  python2.7-dev python-dev python-distlib python-distlib-whl python-dns
  python-egenix-mxtools python-egenix-mxdatetime python-formencode
  python-numpy python-gdal python-html5lib python-html5lib-whl python-openid
  python-tempita python-paste python-pastedeploy-tpl python-pastedeploy
  python-setuptools python-pastescript python-six-whl python-urllib3-whl
  python-requests-whl python-setuptools-whl python-pip-whl python-pip
  python-psycpg2 python-pycurl python-scgi python-urlgrabber
  python-virtualenv python-wheel tomcat7-common tomcat7 ssh-import-id
Install these packages without verification? [y/N] y

```

Warning: The installation process is going to take several minutes and it will need to download packages from Internet.

At this point we have all the packages we need on the system.

GeoNode Setup

First of all we need to prepare a new Python Virtual Environment:

```
$ sudo apt-get install python-pip
$ pip install --upgrade pip
$ pip install --user virtualenv
$ pip install --user virtualenvwrapper
# The commands above will install the Python Venv packages

$ export WORKON_HOME=~/.Envs
$ mkdir -p $WORKON_HOME
$ source $HOME/.local/bin/virtualenvwrapper.sh
$ printf '\n%s\n%s\n%s' '# virtualenv' 'export WORKON_HOME=~/.Envs' 'source $HOME/.
→local/bin/virtualenvwrapper.sh' >> ~/.bashrc
$ source ~/.bashrc
# We have now configured the user environment

$ mkvirtualenv --no-site-packages geonode
# Through this command we have created a brand new geonode Virtual Environment

$ sudo useradd -m geonode
$ sudo usermod -a -G geonode geo
$ sudo chmod -Rf 775 /home/geonode/
$ sudo su - geo
# The commands above are needed only if geo and geonode users have not been already_
→defined
```

Let's activate the new *geonode* Python Virtual Environment:

```
$ workon geonode
```

Move into the *geonode* home folder

```
$ cd /home/geonode
```

We are going to install GeoNode as a dependency of a **Customized Django Project**

Note: A custom project is a Django application with *ad hoc* configuration and folders, which allows you to extend the original **GeoNode** code without actually dealing or modifying the main source code.

This will allow you to easily customize your GeoNode instance, modify the theme, add new functionalities and so on, and also being able to keep updated with the GeoNode latest source code.

For more details please check <https://github.com/GeoNode/geonode-project/tree/master>

```
$ pip install Django==1.8.18
$ django-admin.py startproject --template=https://github.com/GeoNode/geonode-project/
→archive/2.8.0.zip -e py,rst,json,yml my_geonode
```

Let's install the GeoNode dependencies and packages into the Python Virtual Environment:

```
$ cd my_geonode

# Find the closest pygdal version.
# Example: 2.2.1 ... 2.2.1.3, ...
$ gdal-config --version && pip install pygdal==

$ vim requirements.txt
# Make sure requirements contains reference to geonode 2.8 branch
```

(continues on next page)

(continued from previous page)

```
# and correct gdal version (see above)
-e git://github.com/GeoNode/geonode.git@2.8.0#egg=geonode
pygdal==2.2.1.3

$ pip install -r requirements.txt --upgrade
$ pip install -e . --upgrade --no-cache
```

In the next section we are going to setup PostgreSQL Databases for GeoNode and finalize the setup

Create GeoNode DB & Finalize GeoNode Setup

In this section we are going to setup users and databases for GeoNode in PostgreSQL.

Warning: Be sure you have successfully completed the steps in the previous section.

Databases and Permissions

First create the *geonode* user. GeoNode is going to use this user to access the database

```
$ sudo -u postgres createuser -P geonode
```

You will be prompted asked to set a password for the user. Enter *geonode* as password

Create *geonode* database with owner *geonode*

```
$ sudo -u postgres createdb -O geonode geonode
```

And database *geonode_data* with owner *geonode*

```
$ sudo -u postgres createdb -O geonode geonode_data
```

Switch to user *postgres* and create PostGIS extension

```
$ sudo -u postgres psql -d geonode_data -c 'CREATE EXTENSION postgis;'
```

Then adjust permissions

```
$ sudo -u postgres psql -d geonode_data -c 'GRANT ALL ON geometry_columns TO PUBLIC;'
$ sudo -u postgres psql -d geonode_data -c 'GRANT ALL ON spatial_ref_sys TO PUBLIC;'
$ sudo -u postgres psql -d geonode_data -c 'GRANT ALL PRIVILEGES ON ALL TABLES IN_
↳ SCHEMA public TO geonode;'
```

Now we are going to change user access policy for local connections in file *pg_hba.conf*

```
$ sudo vim /etc/postgresql/9.5/main/pg_hba.conf
```

Scroll down to the bottom of the document. We only need to edit one line. Change

```
# "local" is for Unix domain socket connections only
local    all                                all                                peer
```

Into


```
# "local" is for Unix domain socket connections only
local    all                all                                trust
```

Note: If your PostgreSQL database resides on a separate machine, you have to allow remote access to the databases in the `pg_hba.conf` for the *geonode* user and tell PostgreSQL to accept non local connections in your *postgresql.conf* file

Then restart *PostgreSQL* to make the change effective

```
$ sudo service postgresql restart
```

PostgreSQL is now ready. To test the configuration try to connect to the *geonode* database as *geonode*

```
$ psql -U geonode geonode
```

Finalize GeoNode Setup

Once the DB has been correctly configured, we can finalize the *GeoNode* setup.

If not already active let's activate the new *geonode* Python Virtual Environment:

```
$ workon geonode
```

Move into the *geonode* home folder

```
$ cd /home/geonode
```

Move into the *my_geonode* custom project base folder

```
$ cd my_geonode
```

First of all we need to tweak a bit the *my_geonode* `local_settings`. In order to do that, rename the `my_geonode/local_settings.py.sample` file to `my_geonode/local_settings.py` and edit it:

```
$ cp my_geonode/local_settings.py.sample my_geonode/local_settings.py
$ vim my_geonode/local_settings.py
```

Update the following sections accordingly to your server configuration

```
...
SITE_HOST_NAME = os.getenv('SITE_HOST_NAME', "localhost")
SITE_HOST_PORT = os.getenv('SITE_HOST_PORT', "8000")
SITEURL = os.getenv('SITEURL', "http://%s:%s/" % (SITE_HOST_NAME, SITE_HOST_
↪PORT))

...

EMAIL_ENABLE = True

if EMAIL_ENABLE:
    EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
    EMAIL_HOST = 'localhost'
    EMAIL_PORT = 25
```

(continues on next page)

(continued from previous page)

```

EMAIL_HOST_USER = ''
EMAIL_HOST_PASSWORD = ''
EMAIL_USE_TLS = False
DEFAULT_FROM_EMAIL = '{{ project_name }} <no-reply@{{ project_name }}>'

...

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'geonode',
        'USER': 'geonode',
        'PASSWORD': 'geonode',
        'CONN_TOUT': 900,
    },
    # vector datastore for uploads
    'datastore': {
        'ENGINE': 'django.contrib.gis.db.backends.postgis',
        'ENGINE': '', # Empty ENGINE name disables
        'NAME': 'geonode_data',
        'USER': 'geonode',
        'PASSWORD': 'geonode',
        'HOST': 'localhost',
        'PORT': '5432',
        'CONN_TOUT': 900,
    }
}

...

OGC_SERVER_DEFAULT_USER = os.getenv(
    'GEOSERVER_ADMIN_USER', 'admin'
)

OGC_SERVER_DEFAULT_PASSWORD = os.getenv(
    'GEOSERVER_ADMIN_PASSWORD', 'geoserver'
)

...

```

You may also want to tweak some configuration on *my_geonode* settings. This file inherits *my_geonode* *local_settings* and set some GeoNode default settings:

```
$ vim my_geonode/settings.py
```

Update the following sections at the accordingly to your server configuration

```

...
# Make sure GeoNode recognizes your servers

ALLOWED_HOSTS = # Add here your hosts

...
# Modify time zone accordingly

TIME_ZONE = os.getenv('TIME_ZONE', "America/Chicago")

```

(continues on next page)

(continued from previous page)

```

...
# Tweak GeoNode behavior with the following settings
# (see GeoNode documentation for more details)

CLIENT_RESULTS_LIMIT = 20
API_LIMIT_PER_PAGE = 1000
FREETEXT_KEYWORDS_READONLY = False
RESOURCE_PUBLISHING = False
ADMIN_MODERATE_UPLOADS = False
GROUP_PRIVATE_RESOURCES = False
GROUP_MANDATORY_RESOURCES = True
MODIFY_TOPICCATEGORY = True
USER_MESSAGES_ALLOW_MULTIPLE_RECIPIENTS = True
DISPLAY_WMS_LINKS = True

# prevent signing up by default
ACCOUNT_OPEN_SIGNUP = True
ACCOUNT_EMAIL_REQUIRED = True
ACCOUNT_EMAIL_VERIFICATION = 'optional'
ACCOUNT_EMAIL_CONFIRMATION_EMAIL = True
ACCOUNT_EMAIL_CONFIRMATION_REQUIRED = True
ACCOUNT_CONFIRM_EMAIL_ON_GET = True
ACCOUNT_APPROVAL_REQUIRED = True

...
# Modify your maps and backgrounds

# default map projection
# Note: If set to EPSG:4326, then only EPSG:4326 basemaps will work.
DEFAULT_MAP_CRS = "EPSG:3857"

# Where should newly created maps be focused?
DEFAULT_MAP_CENTER = (0, 0)

# How tightly zoomed should newly created maps be?
# 0 = entire world;
# maximum zoom is between 12 and 15 (for Google Maps, coverage varies by
↪area)
DEFAULT_MAP_ZOOM = 0

ALT_OSM_BASEMAPS = os.environ.get('ALT_OSM_BASEMAPS', False)
CARTODB_BASEMAPS = os.environ.get('CARTODB_BASEMAPS', False)
STAMEN_BASEMAPS = os.environ.get('STAMEN_BASEMAPS', False)
THUNDERFOREST_BASEMAPS = os.environ.get('THUNDERFOREST_BASEMAPS', False)
MAPBOX_ACCESS_TOKEN = os.environ.get('MAPBOX_ACCESS_TOKEN', '')
BING_API_KEY = os.environ.get('BING_API_KEY', None)

MAP_BASELAYERS = [{
...

# Enable/Disable the notification system
# (see GeoNode documentation for more details)

NOTIFICATION_ENABLED = True

...

```

(continues on next page)

(continued from previous page)

```
# Enable/Disable the integrated monitoring system
# (see GeoNode documentation for more details)

MONITORING_ENABLED = False

# Tweak the logging options

LOGGING = {
    ...
    "loggers": {
        "django": {
            "handlers": ["console"], "level": "INFO", },
        "geonode": {
            "handlers": ["console"], "level": "INFO", },
        "gsconfig.catalog": {
            "handlers": ["console"], "level": "INFO", },
        "owslib": {
            "handlers": ["console"], "level": "INFO", },
        "pycsw": {
            "handlers": ["console"], "level": "INFO", },
        "{ project_name }": {
            "handlers": ["console"], "level": "DEBUG", },
    },
}
```

Finalize GeoNode Setup & Test

The following Python commands will finalize the setup, configure and create DB tables and download GeoServer.

Warning: Before running the next commands be sure that:

- You have completed all the steps from the beginning of this chapter
- You are located into the `my_geonode` custom project base folder
- The *geonode* Python Virtual Environment is enabled

Stop all the services

```
$ sudo service apache2 stop
$ sudo service tomcat8 stop
# Being sure other services are stopped
```

Cleanup old stuff

- Hard Reset

Warning: This will delete all data you created until now.

```
$ paver reset_hard
# Cleanup folders and old DB Tables
```

- Hard Reset

Note: This will restore only GeoServer.

```
$ rm -Rf geoserver
$ rm -Rf downloaded/*.*
```

Revert to default site settings

You need to revert some customizations of the *my_geonode* `local_settings`. In order to do that, edit the `my_geonode/local_settings.py` file:

```
$ vim my_geonode/local_settings.py
```

Comment the following pieces

```
...
# SITEURL = 'http://localhost'
...
#GEOSERVER_LOCATION = os.getenv(
#     'GEOSERVER_LOCATION', '{}'/geoserver/'.format(SITEURL)
#)

#GEOSERVER_PUBLIC_LOCATION = os.getenv(
#     'GEOSERVER_PUBLIC_LOCATION', '{}'/geoserver/'.format(SITEURL)
#)
...
```

Being sure folders permissions are correctly set

```
$ sudo chown -Rf geonode: my_geonode/uploaded/
$ sudo chown -Rf geonode: my_geonode/static*
```

Setup and start the system in DEV mode

```
$ paver setup
# This command downloads and extract the correct GeoServer version

$ paver sync
# This command prepares the DB tables and loads initial data

$ paver start
# This command allows you to start GeoNode in development mode
```

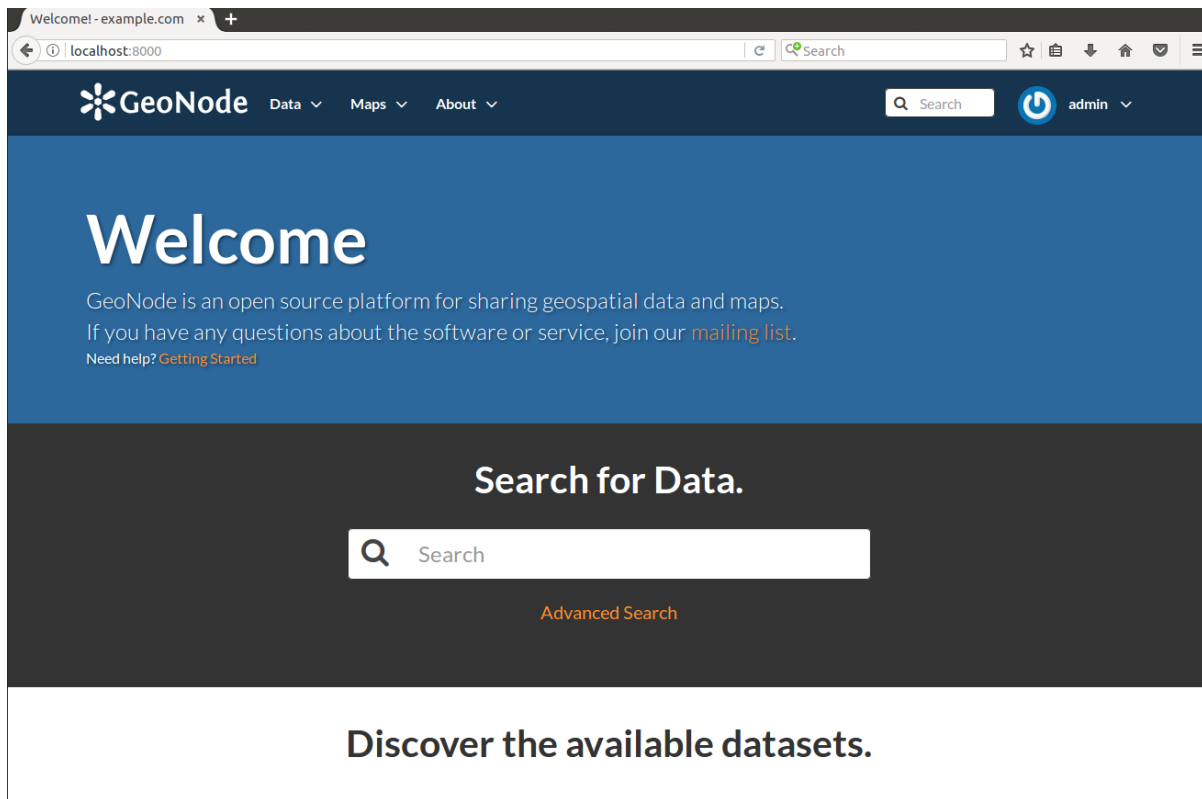
GeoNode and GeoServer in Development mode

The `paver start` command allows you to start the server in **development (DEV)** mode. That means that you will be able to directly do changes to your code and see the results on the browser.

You need to be careful to the different ports of the services. In *DEV* mode the services will run on:

- *GeoNode* port 8000 -> `http://localhost:8000/`
- *GeoServer* port 8080 -> `http://localhost:8080/geoserver`

In order to test it, move to `http://localhost:8000/`



Setup & Configure HTTPD

In this section we are going to setup Apache HTTP to serve GeoNode.

Preliminary Steps & Checks

1. Be sure **development (DEV)** mode has been stopped

If not already active let's activate the new *geonode* Python Virtual Environment:

```
$ workon geonode
```

Move into the *geonode* home folder

```
$ cd /home/geonode
```

Move into the *my_geonode* custom project base folder

```
$ cd my_geonode
```

If `paver start` command is running you need to stop it

```
$ paver stop
```

2. Restore site settings

You need to restore initial customizations of the *my_geonode* `local_settings`. In order to do that, edit the `my_geonode/local_settings.py` file:

```
$ vim my_geonode/local_settings.py
```

Un-comment the following pieces

```
...
SITEURL = 'http://localhost'
...
GEOSERVER_LOCATION = os.getenv(
    'GEOSERVER_LOCATION', '{}geoserver/'.format(SITEURL)
)

GEOSERVER_PUBLIC_LOCATION = os.getenv(
    'GEOSERVER_PUBLIC_LOCATION', '{}geoserver/'.format(SITEURL)
)
...
```

Apache Configuration

Navigate to Apache configurations folder

```
$ cd /etc/apache2/sites-available
```

And create a new configuration file for GeoNode:

```
$ sudo vim geonode.conf
```

Place the following content inside the file

```
WSGIDaemonProcess geonode python-path=/home/geonode/my_geonode:/home/geo/Envs/geonode/
↳lib/python2.7/site-packages user=www-data threads=15 processes=2

<VirtualHost *:80>
    ServerName http://localhost
    ServerAdmin webmaster@localhost
    DocumentRoot /home/geonode/my_geonode/my_geonode

    LimitRequestFieldSize 32760
    LimitRequestLine 32760

    ErrorLog /var/log/apache2/error.log
    LogLevel warn
    CustomLog /var/log/apache2/access.log combined

    WSGIProcessGroup geonode
    WSGIPassAuthorization On
    WSGIScriptAlias / /home/geonode/my_geonode/my_geonode/wsgi.py

    Alias /static/ /home/geonode/my_geonode/my_geonode/static_root/
    Alias /uploaded/ /home/geonode/my_geonode/my_geonode/uploaded/

    <Directory "/home/geonode/my_geonode/my_geonode/">
        <Files wsgi.py>
            Order deny,allow
            Allow from all
            Require all granted
```

(continues on next page)

(continued from previous page)

```

    </Files>

    Order allow,deny
    Options Indexes FollowSymLinks
    Allow from all
    IndexOptions FancyIndexing
</Directory>

<Directory "/home/geonode/my_geonode/my_geonode/static_root/">
    Order allow,deny
    Options Indexes FollowSymLinks
    Allow from all
    Require all granted
    IndexOptions FancyIndexing
</Directory>

<Directory "/home/geonode/my_geonode/my_geonode/uploaded/thumbs/">
    Order allow,deny
    Options Indexes FollowSymLinks
    Allow from all
    Require all granted
    IndexOptions FancyIndexing
</Directory>

<Directory "/home/geonode/my_geonode/my_geonode/uploaded/avatars/">
    Order allow,deny
    Options Indexes FollowSymLinks
    Allow from all
    Require all granted
    IndexOptions FancyIndexing
</Directory>

<Directory "/home/geonode/my_geonode/my_geonode/uploaded/people_group/">
    Order allow,deny
    Options Indexes FollowSymLinks
    Allow from all
    Require all granted
    IndexOptions FancyIndexing
</Directory>

<Directory "/home/geonode/my_geonode/my_geonode/uploaded/group/">
    Order allow,deny
    Options Indexes FollowSymLinks
    Allow from all
    Require all granted
    IndexOptions FancyIndexing
</Directory>

<Directory "/home/geonode/my_geonode/my_geonode/uploaded/documents/">
    Order allow,deny
    Options Indexes FollowSymLinks
    Deny from all
    Require all granted
    IndexOptions FancyIndexing
</Directory>

<Directory "/home/geonode/my_geonode/my_geonode/uploaded/layers/">

```

(continues on next page)

(continued from previous page)

```
    Order allow,deny
    Options Indexes FollowSymLinks
    Deny from all
    Require all granted
    IndexOptions FancyIndexing
</Directory>

<Proxy *>
    Order allow,deny
    Allow from all
</Proxy>

ProxyPreserveHost On
ProxyPass /geoserver http://127.0.0.1:8080/geoserver
ProxyPassReverse /geoserver http://127.0.0.1:8080/geoserver

</VirtualHost>
```

This sets up a VirtualHost in Apache HTTP server for GeoNode and a reverse proxy for GeoServer.

Note: In the case that GeoServer is running on a separate machine change the *ProxyPass* and *ProxyPassReverse* accordingly

Now load apache *proxy* module

```
$ sudo a2enmod proxy_http
```

And enable geonode configuration file

```
$ sudo a2ensite geonode
```

Postfix Configuration

Postfix is a service allowing the host to send e-mail and notifications to the users. In order to make GeoNode being able to send e-mails you will need to enable the service.

```
$ sudo ufw disable
# This will be switch-off the
```

Edit the postfix configuration in order to allow the service act as a web service

```
$ sudo vim /etc/postfix/main.cf
```

Check that at the end of the file the following properties are configured as follows

```
$ sudo vim /etc/postfix/main.cf

...
recipient_delimiter = +
inet_interfaces = all
inet_protocols = all
```

Finally restart the postfix service


```
$ sudo service postfix restart
```

Finalize GeoNode Setup

Once the Apache2 Virtual Host has been correctly configured, we can finalize the *GeoNode* setup.

If not already active let's activate the new *geonode* Python Virtual Environment:

```
$ workon geonode
```

Move into the *geonode* home folder

```
$ cd /home/geonode
```

Move into the *my_geonode* custom project base folder

```
$ cd my_geonode
```

First of all we need to tweak a bit the *my_geonode* *local_settings*. In order to do that, edit the *my_geonode/local_settings.py* file:

```
$ vim my_geonode/local_settings.py
```

Double check that existing properties match the following and add the missing ones

```
SITEURL = 'http://localhost'
...
# account registration settings
ACCOUNT_OPEN_SIGNUP = True
ACCOUNT_APPROVAL_REQUIRED = False
ACCOUNT_EMAIL_CONFIRMATION_EMAIL = False
ACCOUNT_EMAIL_CONFIRMATION_REQUIRED = False

# notification settings
NOTIFICATION_ENABLED = False
NOTIFICATION_LANGUAGE_MODULE = "account.Account"

# Queue non-blocking notifications.
NOTIFICATION_QUEUE_ALL = False

# pinax.notifications
# or notification
NOTIFICATIONS_MODULE = 'pinax.notifications'

if NOTIFICATION_ENABLED:
    INSTALLED_APPS += (NOTIFICATIONS_MODULE, )

#Define email service on GeoNode
EMAIL_ENABLE = False

if EMAIL_ENABLE:
    EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
    EMAIL_HOST = 'localhost'
    EMAIL_PORT = 25
    EMAIL_HOST_USER = ''
    EMAIL_HOST_PASSWORD = ''
```

(continues on next page)

(continued from previous page)

```

EMAIL_USE_TLS = False
DEFAULT_FROM_EMAIL = 'My GeoNode <no-reply@geonode.org>'

# set to true to have multiple recipients in /message/create/
USER_MESSAGES_ALLOW_MULTIPLE_RECIPIENTS = True

INSTALLED_APPS = INSTALLED_APPS + ('my_geonode',)
...
GEOSERVER_LOCATION = os.getenv(
    'GEOSERVER_LOCATION', '{}geoserver/'.format(SITEURL)
)

GEOSERVER_PUBLIC_LOCATION = os.getenv(
    'GEOSERVER_PUBLIC_LOCATION', '{}geoserver/'.format(SITEURL)
)
...
CATALOGUE = {
    'default': {
        # The underlying CSW implementation
        # default is pycsw in local mode (tied directly to GeoNode Django DB)
        'ENGINE': 'geonode.catalogue.backends.pycsw_local',
        # pycsw in non-local mode
        # 'ENGINE': 'geonode.catalogue.backends.pycsw_http',
        # GeoNetwork opensource
        # 'ENGINE': 'geonode.catalogue.backends.geonetwork',
        # deegree and others
        # 'ENGINE': 'geonode.catalogue.backends.generic',

        # The FULLY QUALIFIED base url to the CSW instance for this GeoNode
        'URL': '%s/catalogue/csw' % SITEURL,
        # 'URL': 'http://localhost:8080/geonetwork/srv/en/csw',
        # 'URL': 'http://localhost:8080/deegree-csw-demo-3.0.4/services',

        # login credentials (for GeoNetwork)
        'USER': 'admin',
        'PASSWORD': 'admin',
    }
}
...

```

In the end the `my_geonode/local_settings.py` should be something like this

```

# -*- coding: utf-8 -*-
#####
#
# Copyright (C) 2012 OpenPlans
#
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.
#

```

(continues on next page)

(continued from previous page)

```

# You should have received a copy of the GNU General Public License
# along with this program. If not, see <http://www.gnu.org/licenses/>.
#
#####

# Django settings for the GeoNode project.
import os
from geonode.settings import *
#
# General Django development settings
#

# SECRET_KEY = '*****'

SITEURL = 'http://localhost'
SITENAME = 'my_geonode'

# Defines the directory that contains the settings file as the LOCAL_ROOT
# It is used for relative settings elsewhere.
LOCAL_ROOT = os.path.abspath(os.path.dirname(__file__))

MEDIA_ROOT = os.getenv('MEDIA_ROOT', os.path.join(LOCAL_ROOT, "uploaded"))

STATIC_ROOT = os.getenv('STATIC_ROOT',
                        os.path.join(LOCAL_ROOT, "static_root")
                        )

WSGI_APPLICATION = "my_geonode.wsgi.application"

# Load more settings from a file called local_settings.py if it exists
try:
    from local_settings import *
except ImportError:
    pass

# Additional directories which hold static files
STATICFILES_DIRS.append(
    os.path.join(LOCAL_ROOT, "static"),
)

# Location of url mappings
ROOT_URLCONF = 'my_geonode.urls'

# Location of locale files
LOCALE_PATHS = (
    os.path.join(LOCAL_ROOT, 'locale'),
) + LOCALE_PATHS

# ##### #
# account registration settings
ACCOUNT_OPEN_SIGNUP = True
ACCOUNT_APPROVAL_REQUIRED = False
ACCOUNT_EMAIL_CONFIRMATION_EMAIL = False
ACCOUNT_EMAIL_CONFIRMATION_REQUIRED = False

# notification settings

```

(continues on next page)

(continued from previous page)

```

NOTIFICATION_ENABLED = False
NOTIFICATION_LANGUAGE_MODULE = "account.Account"

# Queue non-blocking notifications.
NOTIFICATION_QUEUE_ALL = False

# pinax.notifications
# or notification
NOTIFICATIONS_MODULE = 'pinax.notifications'

if NOTIFICATION_ENABLED:
    INSTALLED_APPS += (NOTIFICATIONS_MODULE, )

#Define email service on GeoNode
EMAIL_ENABLE = False

if EMAIL_ENABLE:
    EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
    EMAIL_HOST = 'localhost'
    EMAIL_PORT = 25
    EMAIL_HOST_USER = ''
    EMAIL_HOST_PASSWORD = ''
    EMAIL_USE_TLS = False
    DEFAULT_FROM_EMAIL = 'My GeoNode <no-reply@geonode.org>'

# set to true to have multiple recipients in /message/create/
USER_MESSAGES_ALLOW_MULTIPLE_RECIPIENTS = True
# #####

INSTALLED_APPS = INSTALLED_APPS + ('my_geonode',)

TEMPLATES[0]['DIRS'].insert(0, os.path.join(LOCAL_ROOT, "templates"))

# #####
↪#
ALLOWED_HOSTS = ['127.0.0.1', 'localhost', '::1']
PROXY_ALLOWED_HOSTS = ("127.0.0.1", 'localhost', '::1')

POSTGIS_VERSION = (2, 0, 7)

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'geonode',
        'USER': 'geonode',
        'PASSWORD': 'geonode',
    },
    # vector datastore for uploads
    'datastore': {
        'ENGINE': 'django.contrib.gis.db.backends.postgis',
        #'ENGINE': '', # Empty ENGINE name disables
        'NAME': 'geonode_data',
        'USER': 'geonode',
        'PASSWORD': 'geonode',
        'HOST': 'localhost',
        'PORT': '5432',
    }
}

```

(continues on next page)

(continued from previous page)

```

}

GEOSERVER_LOCATION = os.getenv(
    'GEOSERVER_LOCATION', '{}'/geoserver/'.format(SITEURL)
)

GEOSERVER_PUBLIC_LOCATION = os.getenv(
    'GEOSERVER_PUBLIC_LOCATION', '{}'/geoserver/'.format(SITEURL)
)

OGC_SERVER_DEFAULT_USER = os.getenv(
    'GEOSERVER_ADMIN_USER', 'admin'
)

OGC_SERVER_DEFAULT_PASSWORD = os.getenv(
    'GEOSERVER_ADMIN_PASSWORD', 'geoserver'
)

# OGC (WMS/WFS/WCS) Server Settings
OGC_SERVER = {
    'default': {
        'BACKEND': 'geonode.geoserver',
        'LOCATION': GEOSERVER_LOCATION,
        'LOGIN_ENDPOINT': 'j_spring_oauth2_geonode_login',
        'LOGOUT_ENDPOINT': 'j_spring_oauth2_geonode_logout',
        # PUBLIC_LOCATION needs to be kept like this because in dev mode
        # the proxy won't work and the integration tests will fail
        # the entire block has to be overridden in the local_settings
        'PUBLIC_LOCATION': GEOSERVER_PUBLIC_LOCATION,
        'USER' : OGC_SERVER_DEFAULT_USER,
        'PASSWORD' : OGC_SERVER_DEFAULT_PASSWORD,
        'MAPFISH_PRINT_ENABLED' : True,
        'PRINT_NG_ENABLED' : True,
        'GEONODE_SECURITY_ENABLED' : True,
        'GEOGIG_ENABLED' : False,
        'WMST_ENABLED' : False,
        'BACKEND_WRITE_ENABLED': True,
        'WPS_ENABLED' : False,
        'LOG_FILE': '%s/geoserver/data/logs/geoserver.log' % os.path.
→abspath(os.path.join(PROJECT_ROOT, os.pardir)),
        # Set to dictionary identifier of database containing spatial data_
→in DATABASES dictionary to enable
        'DATASTORE': 'datastore',
    }
}

CATALOGUE = {
    'default': {
        # The underlying CSW implementation
        # default is pycsw in local mode (tied directly to GeoNode Django DB)
        'ENGINE': 'geonode.catalogue.backends.pycsw_local',
        # pycsw in non-local mode
        # 'ENGINE': 'geonode.catalogue.backends.pycsw_http',
        # GeoNetwork opensource
        # 'ENGINE': 'geonode.catalogue.backends.geonetwork',
        # deegree and others
        # 'ENGINE': 'geonode.catalogue.backends.generic',

```

(continues on next page)

(continued from previous page)

```

    # The FULLY QUALIFIED base url to the CSW instance for this GeoNode
    'URL': '%s/catalogue/csw' % SITEURL,
    # 'URL': 'http://localhost:8080/geonetwork/srv/en/csw',
    # 'URL': 'http://localhost:8080/deegree-csw-demo-3.0.4/services',

    # login credentials (for GeoNetwork)
    'USER': 'admin',
    'PASSWORD': 'admin',
}
}

ALT_OSM_BASEMAPS = os.environ.get('ALT_OSM_BASEMAPS', False)
CARTODB_BASEMAPS = os.environ.get('CARTODB_BASEMAPS', False)
STAMEN_BASEMAPS = os.environ.get('STAMEN_BASEMAPS', False)
THUNDERFOREST_BASEMAPS = os.environ.get('THUNDERFOREST_BASEMAPS', False)
MAPBOX_ACCESS_TOKEN = os.environ.get('MAPBOX_ACCESS_TOKEN', None)
BING_API_KEY = os.environ.get('BING_API_KEY', None)

MAP_BASELAYERS = [{
    "source": {"ptype": "gxp_olsource"},
    "type": "OpenLayers.Layer",
    "args": ["No background"],
    "name": "background",
    "visibility": False,
    "fixed": True,
    "group": "background"
},
# {
#     "source": {"ptype": "gxp_olsource"},
#     "type": "OpenLayers.Layer.XYZ",
#     "title": "TEST TILE",
#     "args": ["TEST_TILE", "http://test_tiles/tiles/${z}/${x}/${y}.png"],
#     "name": "background",
#     "attribution": "&copy; TEST TILE",
#     "visibility": False,
#     "fixed": True,
#     "group": "background"
# },
{
    "source": {"ptype": "gxp_osmsource"},
    "type": "OpenLayers.Layer.OSM",
    "name": "mapnik",
    "visibility": True,
    "fixed": True,
    "group": "background"
}]

LOCAL_GEOSERVER = {
    "source": {
        "ptype": "gxp_wmcsources",
        "url": OGC_SERVER['default']['PUBLIC_LOCATION'] + "wms",
        "restUrl": "/gs/rest"
    }
}

baselayers = MAP_BASELAYERS
MAP_BASELAYERS = [LOCAL_GEOSERVER]

```

(continues on next page)

(continued from previous page)

```

MAP_BASELAYERS.extend(baselayers)

LOGGING = {
    'version': 1,
    'disable_existing_loggers': True,
    'formatters': {
        'verbose': {
            'format': '%(levelname)s %(asctime)s %(module)s %(process)d '
                      '%(thread)d %(message)s'
        },
        'simple': {
            'format': '%(message)s',
        },
    },
    'filters': {
        'require_debug_false': {
            '()': 'django.utils.log.RequireDebugFalse'
        }
    },
    'handlers': {
        'console': {
            'level': 'DEBUG',
            'class': 'logging.StreamHandler',
            'formatter': 'simple'
        },
        'mail_admins': {
            'level': 'ERROR', 'filters': ['require_debug_false'],
            'class': 'django.utils.log.AdminEmailHandler',
        }
    },
    "loggers": {
        "django": {
            "handlers": ["console"], "level": "ERROR", },
        "geonode": {
            "handlers": ["console"], "level": "DEBUG", },
        "gsconfig.catalog": {
            "handlers": ["console"], "level": "DEBUG", },
        "owslib": {
            "handlers": ["console"], "level": "DEBUG", },
        "pycsw": {
            "handlers": ["console"], "level": "ERROR", },
    },
}

# #####
#

```

Finalize HTTPD Setup

Warning: Those steps must be completed from folder `/home/geonode/my_geonode` and inside *geonode* Python Virtual Environment.

Download GeoNode data to be served by Apache. You will be prompted for confirmation

```
$ python manage.py migrate
$ python manage.py collectstatic
```

Add thumbs and layers folders

```
$ sudo mkdir -p /home/geonode/my_geonode/my_geonode/uploaded/thumbs
$ sudo mkdir -p /home/geonode/my_geonode/my_geonode/uploaded/layers
```

Change permissions on GeoNode files and folders to allow Apache to read and edit them

```
$ sudo chown -Rf geonode /home/geonode/my_geonode/
$ sudo chown -Rf geonode:www-data /home/geonode/my_geonode/my_geonode/static/
$ sudo chown -Rf geonode:www-data /home/geonode/my_geonode/my_geonode/uploaded/
$ chmod -Rf 777 /home/geonode/my_geonode/my_geonode/uploaded/thumbs
$ chmod -Rf 777 /home/geonode/my_geonode/my_geonode/uploaded/layers
$ sudo chown www-data:www-data /home/geonode/my_geonode/my_geonode/static_root/
```

Finally restart Apache to load the new configuration:

```
$ sudo service apache2 restart
```

Install GeoServer Application

In this section we are going to setup GeoServer for GeoNode. GeoServer will run inside *Tomcat* servlet container.

Setup GeoServer

1. You've already installed *Tomcat 8* in the system in the first section of the training. Before you deploy GeoServer stop the running Tomcat instance

```
$ sudo service tomcat8 stop
```

2. Now copy the downloaded GeoServer archive inside Tomcat's webapps folder

```
$ sudo cp -Rf /home/geonode/my_geonode/geoserver/geoserver/ /var/lib/
↪tomcat8/webapps/
```

3. Move GEOSERVER_DATA_DIR on an external location

```
$ sudo mkdir -p /data/geoserver-data
$ sudo mkdir -p /data/geoserver-logs
$ sudo mkdir -p /data/gwc_cache_dir
$ sudo cp -Rf /home/geonode/my_geonode/geoserver/data/* /data/geoserver-
↪data/
$ sudo chown -Rf tomcat8: /data/geoserver-data/
$ sudo chown -Rf tomcat8: /data/geoserver-logs/
$ sudo chown -Rf tomcat8: /data/gwc_cache_dir/
```

4. Set default *Java* settings

You need to edit the `/etc/default/tomcat8` file

```
$ sudo vim /etc/default/tomcat8
```

Make sure `JAVA_OPTS` are configured as follows


```
#JAVA_OPTS="-Djava.awt.headless=true -Xmx128m -XX:+UseConcMarkSweepGC"
GEOSERVER_DATA_DIR="/data/geoserver-data"
GEOSERVER_LOG_LOCATION="/data/geoserver-logs/geoserver.log"
GEOWEBCACHE_CACHE_DIR="/data/gwc_cache_dir"
GEOFENCE_DIR="$GEOSERVER_DATA_DIR/geofence"

JAVA_OPTS="-Djava.awt.headless=true -XX:MaxPermSize=512m -
↪XX:PermSize=128m -Xms512m -Xmx2048m -Duser.timezone=GMT -Dorg.geotools.
↪shapefile.datetime=true -XX:+UseConcMarkSweepGC -XX:+UseParNewGC -
↪XX:ParallelGCThreads=4 -Dfile.encoding=UTF8 -Duser.timezone=GMT -Djavax.
↪servlet.request.encoding=UTF-8 -Djavax.servlet.response.encoding=UTF-8 -
↪DGEOSERVER_DATA_DIR=$GEOSERVER_DATA_DIR -Dgeofence.dir=$GEOFENCE_DIR -
↪DGEOSERVER_LOG_LOCATION=$GEOSERVER_LOG_LOCATION -DGEOWEBCACHE_CACHE_DIR=
↪$GEOWEBCACHE_CACHE_DIR"
```

Warning: Double check memory options `-Xms512m -Xmx2048m` are compatible with your VM available RAM

5. Set default *Catalina* settings

You need to edit the `/var/lib/tomcat8/conf/catalina.properties` file

```
$ sudo vim /var/lib/tomcat8/conf/catalina.properties
```

Make sure `bcprov*.jar` is skipped at run-time

```
tomcat.util.scan.StandardJarScanFilter.jarsToSkip=\
...
xom-*.jar,\
bcprov*.jar
```

6. Restart *Tomcat 8* service

```
$ sudo service tomcat8 restart
```

You can follow the start-up logs by running the following shell command

```
$ sudo tail -F -n 300 /var/lib/tomcat8/logs/catalina.out
```

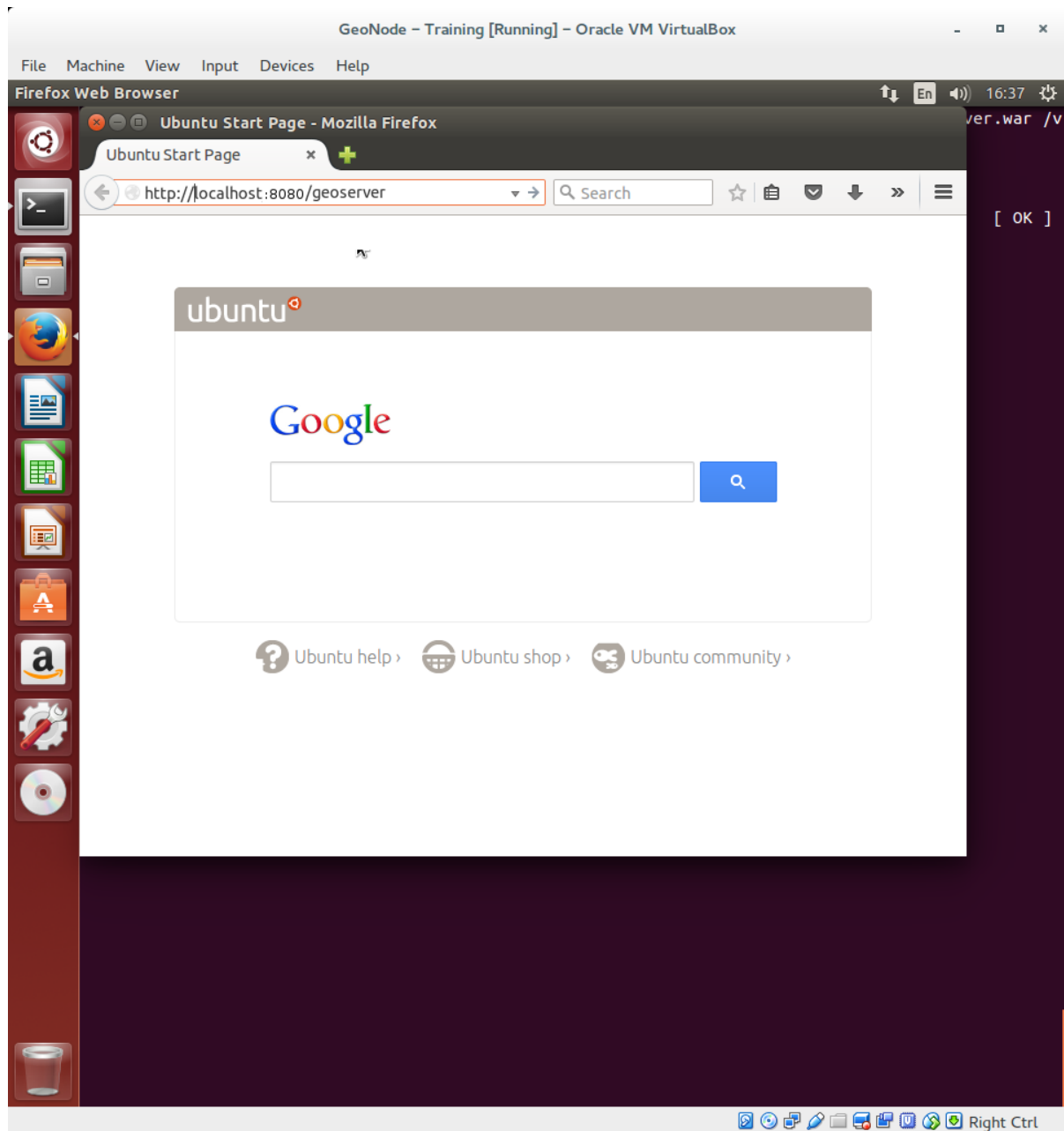
Test GeoServer

Now start Tomcat to deploy GeoServer:

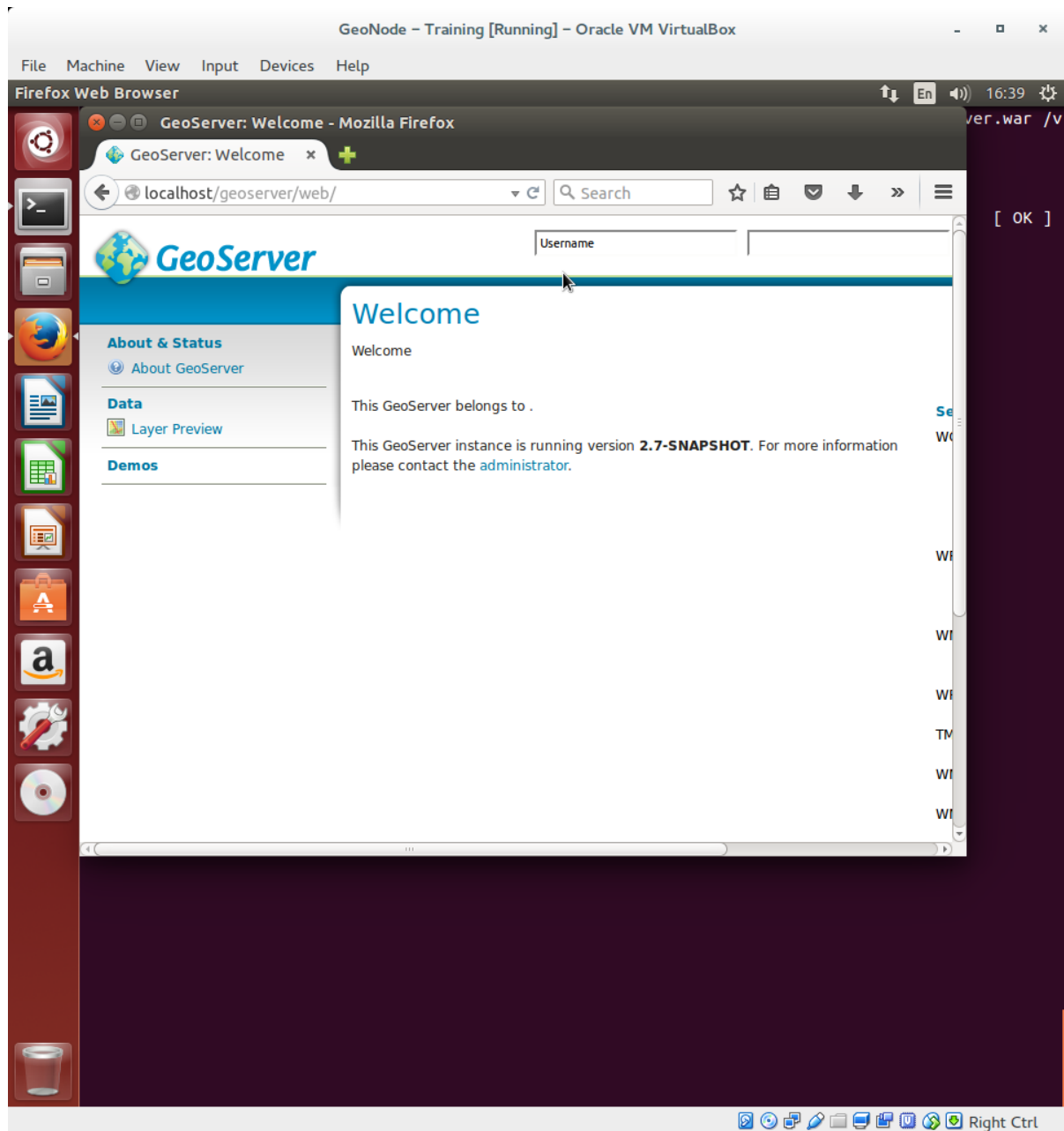
```
sudo service tomcat8 start
```

Tomcat will extract GeoServer web archive and start GeoServer. This may take some time

Open a web browser (in this example *Firefox*) and navigate to <http://localhost:8080/geoserver>



In a few seconds GeoServer web interface will show up:



GeoNode authentication integration

All we need to do now is to integrate GeoNode authentication so that GeoNode administrator will be able to access and administer GeoServer as well.

1. Stop GeoServer

```
$ sudo service tomcat8 stop
```

2. Edit `/data/geoserver-data/security/filter/geonode-oauth2/config.xml` with a text editor

```
$ sudo gedit /data/geoserver-data/security/filter/geonode-oauth2/config.xml
```

And make sure the following values are configured as follows:

```
<accessTokenUri>http://localhost/o/token/</accessTokenUri>
<userAuthorizationUri>http://localhost/o/authorize/</userAuthorizationUri>
<redirectUri>http://localhost/geoserver</redirectUri>
<checkTokenEndpointUrl>http://localhost/api/o/v4/tokeninfo/</
checkTokenEndpointUrl>
<logoutUri>http://localhost/account/logout/</logoutUri>
```

3. Edit /data/geoserver-data/security/role/geonode\ REST\ role\ service/config.xml with a text editor

```
$ sudo gedit /data/geoserver-data/security/role/geonode\ REST\ role\
service/config.xml
```

And make sure the following values are configured as follows:

```
<baseUri>http://localhost</baseUri>
```

4. Edit /data/geoserver-data/global.xml with a text editor

```
$ sudo gedit /data/geoserver-data/global.xml
```

And make sure the following values are configured as follows:

```
<proxyBaseUrl>http://localhost/geoserver</proxyBaseUrl>
```

5. Restart GeoServer to make the changes effective

```
$ sudo service tomcat8 restart
```

You can follow the start-up logs by running the following shell command

```
$ sudo tail -F -n 300 /var/lib/tomcat8/logs/catalina.out
```

Finish installation

In previous sections you've setup all the applications we need to run GeoNode.

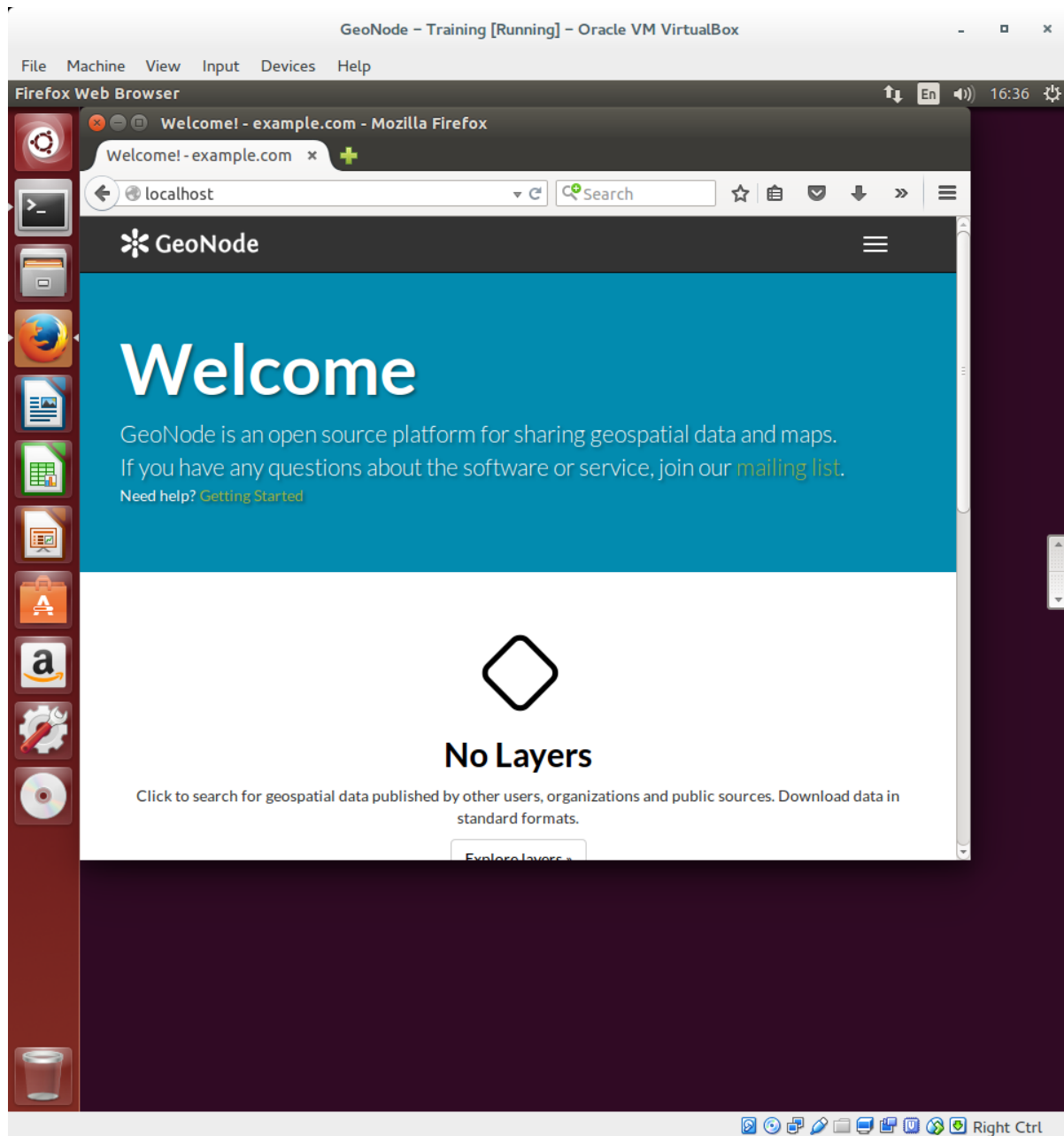
Test the installation

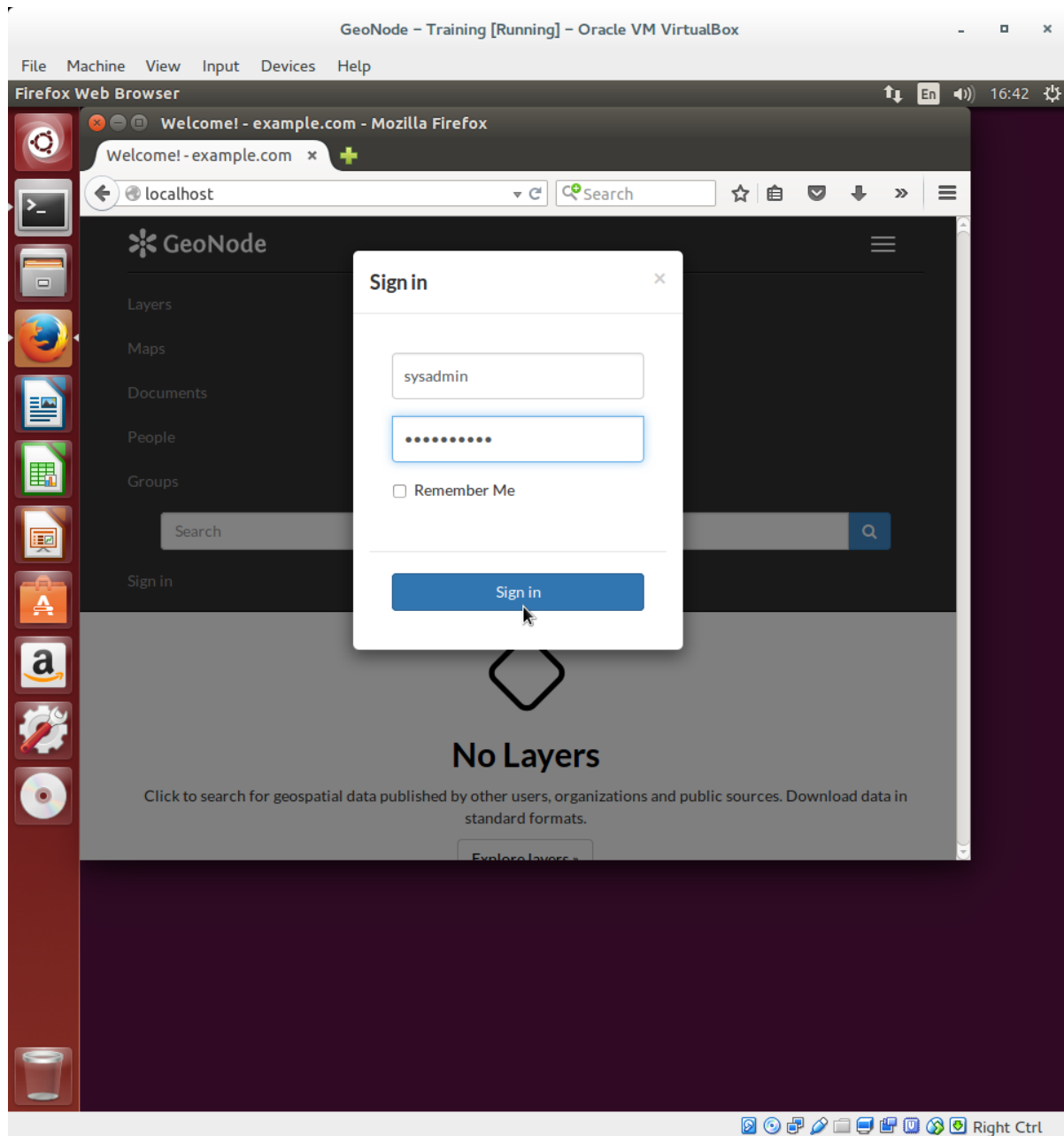
We are ready to restart GeoNode (Apache) and test the installation. Restart Apache

```
$ sudo service apache2 restart
```

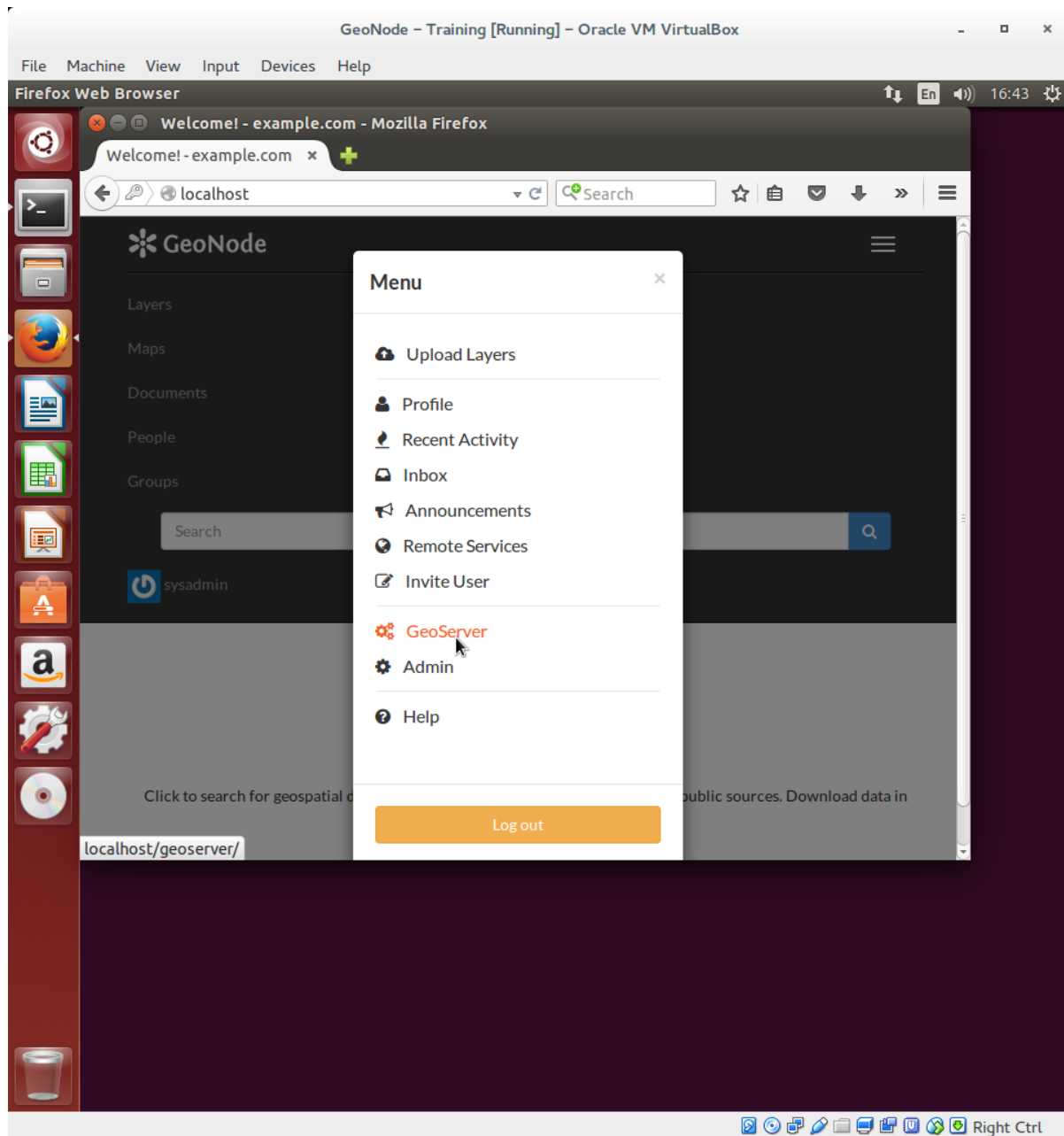
Open the browser and navigate to <http://localhost/>

GeoNode User interface will show up. Login with admin username and password you just set.

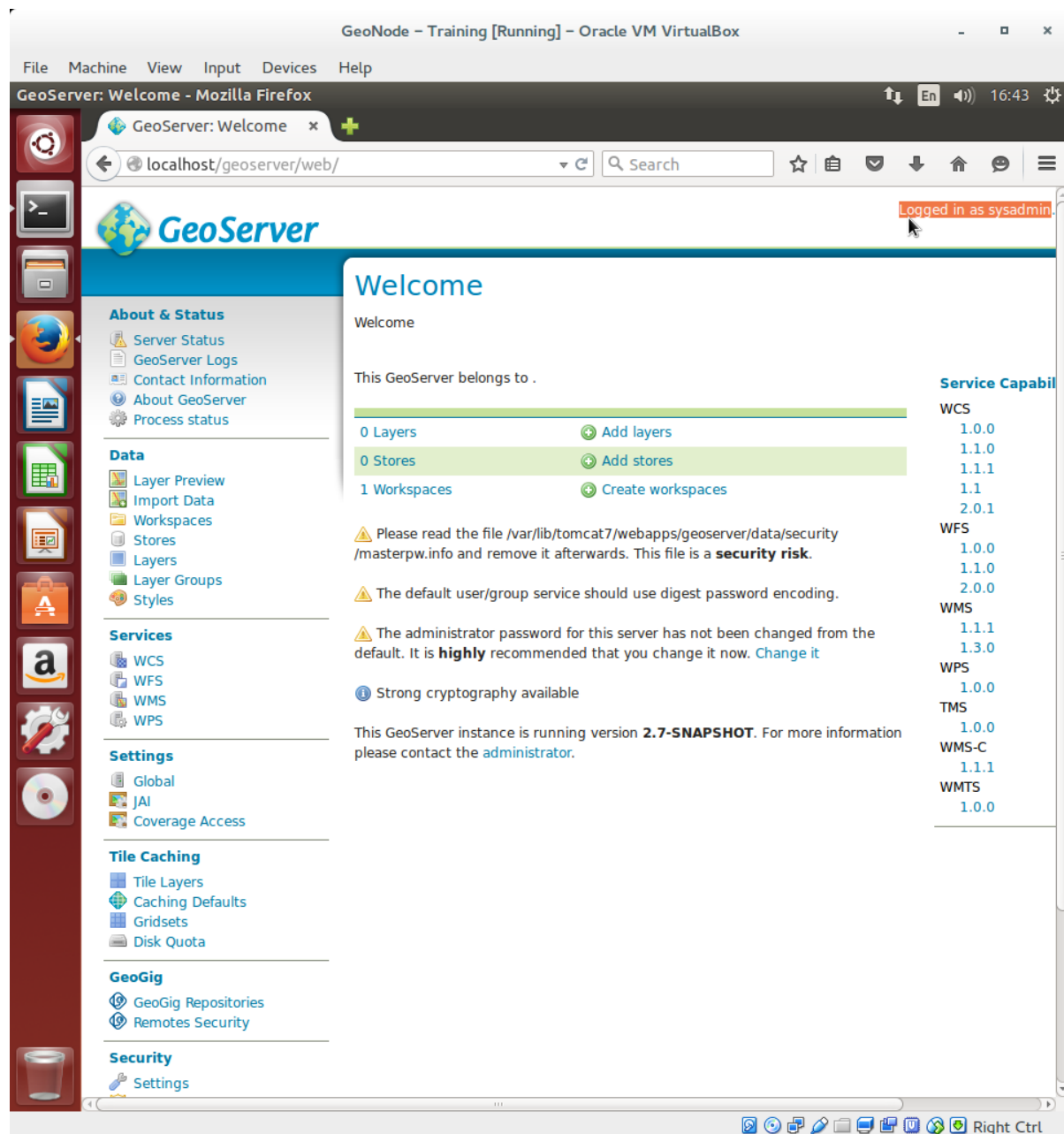




Now open the main menu and click on *GeoServer*



You will be redirected to GeoServer user interface. You will automatically be logged in as administrator in GeoServer.



3.2.1.2.8 GeoNode (v2.8) installation on CentOS 7

This part of the documentation describes the complete setup process for GeoNode on a CentOS 7 machine.

Installing the Operating System

CentOS Setup

We are re going to install a minimal CentOS 7 distribution. You can get a copy of the .iso the image used for the installation [here](#).

Boot up the installation DVD and start the *CentOS 7* Installation wizard.

- Under *Select Date & Time* an set appropriate Date and Time settings
- Under *Keyboard* and choose the keyboard layout
- Under *Installation Destination* select the hard disk where CentOS will be installed.

Create a custom partitioning scheme as follows:

Partition Label	Partition Type	Size	Mount Point
boot	ext3	700 MB	/boot
root	ext4	35 GB	/
swap	swap	4 GB	

- Under *Networking* configure your network interface according to your infrastructure you can either set it to *DHCP* to automatically get all the settings from a local DHCP server or configure it by hand.
- Enable the network interface, then go back to *Select Date & Time* and enable NTP synchronization periodically get date and time settings from CentOS servers
- Click on *Begin Installation*
- Now set the password for the *root* user. Also click on *User Creation* to create the *geo* user.
- Wait for the installation process to finish, then reboot your machine

Network configuration

The network configuration should already be set, since it was set during CentOS setup stage.

You may want to review the configuration files

```
/etc/sysconfig/network-scripts/ifcfg-DEVICE
```

You may also want to review the file `/etc/resolv.conf` to check the name servers.

Check that the connection is up by pinging and external server:

```
ping 8.8.8.8
```

Check that the DNS are properly configuring by pinging a host by its name:

```
ping google.com
```

Warning: Please note that in CentOS only ssh incoming connections are allowed; all other incoming connections are disabled by default.

In the paragraph related to the httpd service you can find details about how to enable incoming traffic.

Note that after configuring the network, you may continue installing the system setup using a ssh connection.

User access configuration

Login as `root` user and give the `geo` user administrative privileges by adding him to the `wheel` group:

```
usermod -aG wheel geo
```

SSH access

Allow SSH connections through the firewall

On CentOS 7 the firewall is enabled by default. To allow SSH clients to connect to the machine allow incoming connections on port 22:

```
firewall-cmd --zone=public --add-port=22/tcp --permanent
firewall-cmd --zone=public --add-service=ssh --permanent
firewall-cmd --reload
```

Disable SSH login for the *root* user

Warning: Before you disable root login make sure you are able to login via SSH with *geo* user account and you have the privileges to run *sudo su* to switch to the *root* user account.

Edit file */etc/ssh/sshd_config* to disable *root* login via SSH:

```
PermitRootLogin no
```

Public key authentication

Public key authentication is generally considered a safer way to authenticate users for SSH access. Let's set it up and disable password based authentication

First generate a public/private key pair using *ssh-keygen*:

```
ssh-keygen
```

Follow the procedure, you will end up with your newly generated key under *~/.ssh*. Now copy your *public* (by default it is called *id_rsa.pub*) key over the CentOS machine in */home/geo/.ssh/authorized_keys*. There are several ways to do it, we are going to use the *ssh-copy-id* tool:

```
ssh-copy-id -i ~/.ssh/id_rsa.pub geo@<server-ip-address>
```

You should now be able to login via SSH as *geo* without been asked for the password:

```
ssh geo@<server-ip-address>
```

You can now disable password based login over SSH

Warning: Before disabling password authentication make sure you've installed your public key on the server and you are able to login without password

Edit */etc/ssh/sshd_config* as follows:

```
...
RSAAuthentication yes
...
PubkeyAuthentication yes
...
PasswordAuthentication no
...
UsePAM no
...
```

Installing ntp

Install the program for NTP server synchronization:

```
sudo yum install ntp
```

Edit `/etc/ntp.conf` and add the following line before the first `server` directive:

```
server tempo.ien.it      # Galileo Ferraris
```

Replace `tempo.ien.it` with your nearest NTP server.

Sync with the server by issuing:

```
systemctl start ntpd
```

Set the time synchronization as an automatically started daemon:

```
systemctl enable ntpd
```

Installing base packages

Install common utility packages:

```
sudo yum install -y man vim openssh-clients zip unzip wget
```

Update the system with EPEL latest release:

```
sudo yum install -y epel-release
sudo yum update -y && sudo reboot
```

Install pip and necessary dependencies:

```
sudo yum install -y python-devel python-setuptools python-pip python-virtualenv
sudo pip install --upgrade pip
```

Installing the Operating System

Warning: In order to follow this section you must have [Vagrant](#) already installed into your machine. If not then you can follow the link and install properly in your computer.

CentOS Setup

Start a new base CentOS server from scratch is as easier as possible with this simple command:

```
vagrant init centos/7
```

A new configuration file called *Vagrantfile* is created in the same directory. That file will contain all the configuration of the box you are going to run.

Alternatively you can start a new specific version of a CentOS server from an ISO image like this *CentOS-7-x86_64-Minimal-1611.iso*. In order to have such a minimal box we can exploit [Packer](#) in doing that. Therefore follow the previous link and install the toolkit. [Ansible](#) is required too.

As an example for achieving this goal you might follow the instructions on [this](#) GitHub repository. Fork it and simply let's do:

```
packer build --only=virtualbox-iso centos7.json
```

The build artifact `virtualbox-centos7.box` will be in the directory `builds` and can be mounted as alternative Virtual Box image in the *Vagrantfile*.

If you, at any time, modify the configuration you can bring your changes by running:

```
vagrant reload
```

Network configuration

The configuration of your new CentOS machine can be customised at your need. You can choose to configure a private network which allows host-only access to the machine using a specific IP as well as a public one which generally matched to bridged network and the machine would appear as another physical device on your network. Initially you can opt also to have your internal port of GeoNode forwarded to a physical port on your host for navigating the application from the browser. The simplest configuration can be:

```
config.vm.network "forwarded_port", guest: 80, host: 8001
```

Please note that in development mode this configuration can be a little bit different:

```
config.vm.network "forwarded_port", guest: 8000, host: 8001 # GeoNode port
config.vm.network "forwarded_port", guest: 8080, host: 8080 # GeoServer port
```

You can even choose to mapping on 80 but be carefully you need administrative privileges on your host machine. So it is recommended to begin with a port higher like in the previous example.

Launch your CentOS box

After editing the Vagrant configurations you should end up with a *Vagrantfile* like this basic sample. From the same directory then run this command and launch your box:

```
vagrant up
```

Install GeoNode on CentOS 7 (dev mode)

Preparation

Make sure all the needed libraries are installed

```
sudo yum -y update
sudo yum groupinstall 'Development Tools'
sudo yum install -y libxml2-devel libxslt-devel libjpeg-turbo-devel
↪ postgresql postgresql-server postgresql-contrib postgresql-libs postgresql-
↪ devel postgis geos-python python python-tools python-devel python-pillow
↪ python-lxml openssh-clients zip unzip wget git gdal python-virtualenv gdal-
↪ python geos python-pip python-imaging python-devel gcc-c++ python-psycopg2
↪ libxml2 libxml2-devel libxml2-python libxslt libxslt-devel libxslt-python
```

Install pip

Installation of Python pip:

```
sudo rpm -iUvh http://dl.fedoraproject.org/pub/epel/7/x86_64/e/epel-release-
↪ 7-9.noarch.rpm
sudo yum -y update
sudo yum -y install python-pip
sudo pip install --upgrade pip
```

Warning: Make sure to align the version of EPEL to the latest for your current CentOS version

Install Python Virtual Environment

```
sudo pip install virtualenvwrapper
```

Set up PostgreSQL

Enable the PostgreSQL service

```
systemctl enable postgresql
```

Change the Unix password for the postgres user

```
sudo passwd -u postgres -f
sudo passwd postgres
```

Initialize the PostgreSQL database with the default service name *postgresql*

```
sudo /usr/bin/postgresql-setup initdb
```

Start the PostgreSQL service

```
systemctl start postgresql.service
```

Create the database for GeoNode

```
su - postgres
pg_ctl status
pg_ctl start # if not running
```

```
createdb geonode
```

Create the needed role and privileges

```
psql
postgres=#
postgres=# \password postgres
postgres=# CREATE USER geonode WITH PASSWORD 'geonode';
postgres=# GRANT ALL PRIVILEGES ON DATABASE "geonode" to geonode;
postgres=# \q
```

Install Java

First come back to the normal user, then check if Java is already installed

```
java -version
```

In case, install Java

```
sudo yum install -y java-1.8.0-openjdk-devel
```

Setup a virtual environment

Assuming your username is geonode, you need to edit your `.bashrc` file

```
nano /home/geonode/.bashrc
```

Add the following lines (please replace *geonode* with your actual user name):

```
# virtualenvwrapper
export VIRTUALENVWRAPPER_PYTHON=/usr/bin/python
export WORKON_HOME=home/geonode/.venvs
source /usr/bin/virtualenvwrapper.sh
export PIP_DOWNLOAD_CACHE=$HOME/.pip-downloads
```

Save and exit. Then,

```
source /home/geonode/.bashrc
```

Install GeoNode

Setup a virtual environment for GeoNode and enter into it

```
mkvirtualenv geonode --system-site-packages
workon geonode
```

Make sure to have the most updated Python pip version

```
pip install --upgrade pip
```

Clone the current stable branch 2.6.x

```
git clone -b 2.6.x https://github.com/GeoNode/geonode
```

Install GeoNode and its dependencies

```
pip install -e geonode && pip install -r geonode/requirements.txt
pip Paver
```

Verify the installation

```
pip freeze | grep geonode
```

In case of successful installation, this should return several (50+) lines.

Install GeoServer

The installation of GeoServer in development mode can be started with this `pavement` command script

```
paver setup
```

This should return

```
GeoNode development environment successfully set up.If you have not set
up an administrative account, please do so now. Use "paver start" to start
up the server.
```

Run the Django server

Before starting the GeoNode server you have to complete the installation with several import pre-running steps. Let's see them in their order.

Warning: Please note we have always to be in our virtual environment before running the following commands, while you don't need the `sudo` privileges anymore!

Migrate the database

```
python manage.py migrate
```

Create an administrative account as know as `superuser` in Django. Please make sure you will create at least one `superuser` named `admin` which is required in development mode.

```
python manage.py createsuperuser
```

Create the required initial data and OAuth2 configurations

```
python manage.py loaddata geonode/base/fixtures/initial_data.json
python manage.py loaddata geonode/base/fixtures/default_oauth_apps.json
```

Finally we are able to start all the components

```
paver start_geoserver # start GeoServer
python manage.py runserver 0.0.0.0:8000 # start GeoNode
```

Warning: Note that the default IP address, 127.0.0.1, is not accessible from other machines on your network. In case you are using `forwarded_port` on a Vagrantfile please make sure to start `runserver` with the option `:option:0.0.0.0:8000` which allows fundamentally to view your development server from other machines on the network, included your host machine in such a case. See more on [Installing the Operating System](#).

Installing PostgreSQL and PostGIS

Install PostgreSQL

Install the package for configuring the PGDG repository:

```
sudo yum install http://yum.postgresql.org/9.5/redhat/rhel-7-x86_64/pgdg-centos95-9.5-
↪3.noarch.rpm
```

Install PostgreSQL, PostGIS and related libs:

```
sudo yum update
sudo yum install -y postgis2_95 postgresql95 postgresql95-server postgresql95-libs_
↪postgresql95-contrib \
postgresql95-devel gdal gdal-python geos python-imaging gcc-c++ \
python-psycopg2 libxml2 libxml2-devel libxml2-python libxslt libxslt-devel libxslt-
↪python
```

Initialize the DB:

```
sudo /usr/pgsql-9.5/bin/postgresql95-setup initdb
```

Enable start on boot:

```
sudo systemctl enable postgresql-9.5
```

Start the PostgreSQL service manually:

```
sudo systemctl start postgresql-9.5
```

To restart or reload the instance, you can use the following commands:

```
sudo systemctl restart postgresql-9.5
sudo systemctl reload postgresql-9.5
```

Setting PostgreSQL access

Now we are going to change user access policy for local connections in file `pg_hba.conf`:

```
sudo vim /var/lib/pgsql/9.5/data/pg_hba.conf
```

Scroll down to the bottom of the document. We only need to edit one line. Change:


```
# "local" is for Unix domain socket connections only
local    all                all                                peer
# IPv4 local connections:
host     all                all                                127.0.0.1/32    ident
```

into:

```
# "local" is for Unix domain socket connections only
local    all                all                                trust
# IPv4 local connections:
host     all                all                                127.0.0.1/32    md5
```

Note: If your PostgreSQL database resides on a separate machine, you have to allow remote access to the databases in the `pg_hba.conf` for the `geonode` user and tell PostgreSQL to accept non local connections in your `postgresql.conf` configuration.

Once the configuration file has been edited, restart PostgreSQL to make these changes effective:

```
sudo systemctl restart postgresql-9.5
```

Create GeoNode Users Databases

Switch to the postgres user:

```
su - postgres
```

First create the geonode user. GeoNode is going to use this user to access the database:

```
createuser -P geonode
```

You will be prompted asked to set a password for the user. Enter `geonode` as password

Create geonode database with owner geonode:

```
createdb -O geonode geonode
```

And database geonode_data with owner geonode:

```
createdb -O geonode geonode_data
```

Create PostGIS extension on the database for spatial data:

```
psql -d geonode_data -c 'CREATE EXTENSION postgis;'
```

Then adjust permissions:

```
psql -d geonode_data -c 'GRANT ALL ON geometry_columns TO PUBLIC;'
psql -d geonode_data -c 'GRANT ALL ON spatial_ref_sys TO PUBLIC;'
```

We are setting the default encoding to UTF-8, which Django expects:

```
psql -d geonode -c 'ALTER ROLE geonode SET client_encoding TO 'utf8';'
psql -d geonode_data -c 'ALTER ROLE geonode SET client_encoding TO 'utf8';'
```

And exit *postgres* user:

```
exit
```

Tomcat Installation

Installing Java

We'll need a JDK to run GeoServer. In particular the latest GeoServer from GeoNode needs the JDK version 1.8

You may already have the OpenJDK package (`java-1.8.0-openjdk.x86_64`) installed. Check and see if Java 8 is already installed:

```
java -version
openjdk version "1.8.0_131"
OpenJDK Runtime Environment (build 1.8.0_131-b12)
OpenJDK 64-Bit Server VM (build 25.131-b12, mixed mode)
```

Otherwise install it by running:

```
sudo yum install java-1.8.0-openjdk.x86_64
```

Once done, the command `java -version` should return info about the installed version.

If java version does not match the one just installed, run the following command:

```
alternatives --set java /usr/lib/jvm/jre-1.8.0-openjdk/bin/java
```

Oracle JDK

Until recently, the Oracle JDK was a better performer than the OpenJDK, so it was the preferred choice. This is no longer true, anyway in the following paragraph you can find instruction about how to install the Oracle JDK.

You can download the Oracle JDK RPM from this page:

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

Programmatically you can then run:

```
curl -LO -H "Cookie: oraclelicense=accept-securebackup-cookie;" "http://download.
↪oracle.com/otn-pub/java/jdk/8u131-b11/d54c1d3a095b4ff2b6607d096fa80163/jdk-8u131-
↪linux-x64.rpm"
```

Once you have the `.rpm` file, you can install it by:

```
sudo rpm -ivh jdk-8u131-linux-x64.rpm
```

Once installed, you still see that the default `java` and `javac` commands are still the ones from OpenJDK. In order to switch JDK version you have to set the proper system alternatives.

Issue the command:

```
alternatives --install /usr/bin/java java /usr/java/latest/bin/java 200000 \
--slave /usr/lib/jvm/jre jre /usr/java/latest/jre \
--slave /usr/lib/jvm-exports/jre jre_exports /usr/java/latest/jre/lib \
```

(continues on next page)

(continued from previous page)

```
--slave /usr/bin/keytool keytool /usr/java/latest/jre/bin/keytool \
--slave /usr/bin/orbd orbd /usr/java/latest/jre/bin/orbd \
--slave /usr/bin/pack200 pack200 /usr/java/latest/jre/bin/pack200 \
--slave /usr/bin/rmid rmid /usr/java/latest/jre/bin/rmid \
--slave /usr/bin/rmiregistry rmiregistry /usr/java/latest/jre/bin/rmiregistry \
--slave /usr/bin/servertool servertool /usr/java/latest/jre/bin/servertool \
--slave /usr/bin/tnameserv tnameserv /usr/java/latest/jre/bin/tnameserv \
--slave /usr/bin/unpack200 unpack200 /usr/java/latest/jre/bin/unpack200 \
--slave /usr/share/man/man1/java.1 java.1 /usr/java/latest/man/man1/java.1 \
--slave /usr/share/man/man1/keytool.1 keytool.1 /usr/java/latest/man/man1/keytool.1 \
--slave /usr/share/man/man1/orbd.1 orbd.1 /usr/java/latest/man/man1/orbd.1 \
--slave /usr/share/man/man1/pack200.1 pack200.1 /usr/java/latest/man/man1/pack200.1 \
--slave /usr/share/man/man1/rmid.1.gz rmid.1 /usr/java/latest/man/man1/rmid.1 \
--slave /usr/share/man/man1/rmiregistry.1 rmiregistry.1 /usr/java/latest/man/man1/
↪rmiregistry.1 \
--slave /usr/share/man/man1/servertool.1 servertool.1 /usr/java/latest/man/man1/
↪servertool.1 \
--slave /usr/share/man/man1/tnameserv.1 tnameserv.1 /usr/java/latest/man/man1/
↪tnameserv.1 \
--slave /usr/share/man/man1/unpack200.1 unpack200.1 /usr/java/latest/man/man1/
↪unpack200.1
```

Then run:

```
alternatives --config java
```

and select the number related to /usr/java/latest/bin/java.

Now the default java version should be the Oracle one. Verify the proper installation on the JDK:

```
# java -version
java version "1.8.0_131"
Java(TM) SE Runtime Environment (build 1.8.0_131-b11)
Java HotSpot(TM) 64-Bit Server VM (build 25.131-b11, mixed mode)
```

Installing Tomcat

Tomcat

Let's install *Tomcat* first:

```
sudo yum install -y tomcat
```

Then prepare a clean instance called `base` to be used as a template for all tomcat instances:

```
sudo mkdir /var/lib/tomcats/base
sudo cp -a /usr/share/tomcat/* /var/lib/tomcats/base/
```

Then create GeoServer's instance directory structure:

```
sudo mkdir /var/lib/tomcats/geoserver
sudo cp -a /usr/share/tomcat/* /var/lib/tomcats/geoserver/
```

Instance manager script

Copy the existing management script:

```
sudo cp /usr/lib/systemd/system/tomcat.service \
/usr/lib/systemd/system/tomcat\@geoserver.service
```

Edit the EnvironmentFile variable in service management file as follows:

```
sudo vim /usr/lib/systemd/system/tomcat\@geoserver.service

# Systemd unit file for default tomcat
#
# To create clones of this service:
# DO NOTHING, use tomcat@.service instead.

[Unit]
Description=Apache Tomcat Web Application Container
After=syslog.target network.target

[Service]
Type=simple
EnvironmentFile=/etc/tomcat/tomcat.conf
Environment="NAME="
EnvironmentFile=-/etc/sysconfig/tomcat\@geoserver
ExecStart=/usr/libexec/tomcat/server start
ExecStop=/usr/libexec/tomcat/server stop
SuccessExitStatus=143
User=tomcat
Group=tomcat

[Install]
WantedBy=multi-user.target
```

Create the associated configuration file from template:

```
sudo cp /etc/sysconfig/tomcat /etc/sysconfig/tomcat\@geoserver
```

Edit the configuration file and customize the CATALINA_HOME and CATALINA_BASE variables:

```
...
CATALINA_BASE="/var/lib/tomcats/geoserver"
CATALINA_HOME="/usr/share/tomcat"
...
```

Now download and copy GeoServer web archive inside the webapps folder. Tomcat will extract the war file and run GeoServer:

```
curl -LO https://build.geo-solutions.it/geonode/geoserver/latest//geoserver-2.9.x-
-oauth2.war && mv geoserver-2.9.x-oauth2.war geoserver.war
sudo mkdir -p /var/lib/tomcats/geoserver/webapps/
sudo cp geoserver.war /var/lib/tomcats/geoserver/webapps/
```

And fix the permissions on the files:

```
sudo chown -R tomcat:tomcat /var/lib/tomcats*
```

Finally start GeoServer:

```
sudo systemctl start tomcat@geoserver
```

And enable it to automatically start at boot time:

```
sudo systemctl enable tomcat@geoserver
```

Install GeoNode application

Install required libs

Make sure all the needed libraries are installed:

```
sudo yum install -y git gdal gdal-python geos python-pip python-imaging \
python-virtualenv python-devel gcc-c++ python-psycopg2 libxml2 \
libxml2-devel libxml2-python libxslt libxslt-devel libxslt-python
```

Upgrade Python PIP version

Upgrade pip utility:

```
sudo pip install --upgrade pip
```

Create a geonode system user for the application

Create the group:

```
sudo groupadd --system geonode
```

Create the user:

```
sudo useradd --system --gid geonode --shell /bin/bash --home /home/geonode --create-
home geonode
```

Create project home somewhere:

```
sudo mkdir -p /opt/apps/geonode
```

Give the required permissions to the new user geonode:

```
sudo chown -R geonode:geonode /opt/apps/geonode/
```

Install GeoNode

As a user geonode install GeoNode sources from official repository:

```
curl -LO "https://github.com/GeoNode/geonode/archive/2.6.3.zip"
unzip 2.6.3.zip && mv geonode-2.6.3 geonode
```

Move the sources into project folder:

```
mv geonode /opt/apps/geonode/
```

Navigate to sources folder and install required packages:

```
cd /opt/apps/geonode/geonode
sudo pip install -e . --upgrade --no-cache
```

Edit settings

GeoNode Configuration

Now that all applications have been configured we are going to instruct GeoNode on how to connect to *PostgreSQL* and *GeoServer*. Also we are going to instruct GeoNode on who is allowed to connect to it.

First navigate to geonode configuration folder:

```
cd /opt/apps/geonode/geonode/geonode/
```

Copy the *local_settings.py* sample file called *local_settings.py.sample*:

```
cp local_settings.py.geoserver.sample local_settings.py
```

Then edit the configuration file:

```
vim local_settings.py
```

Add the *ALLOWED_HOSTS* and *PROXY_ALLOWED_HOSTS* variables at the top with the following values::

```
ALLOWED_HOSTS = ['127.0.0.1', 'localhost', ':::1']
PROXY_ALLOWED_HOSTS = ("127.0.0.1", 'localhost', ':::1', )
```

Add the *POSTGIS_VERSION* variable matching your PostGIS version::

```
POSTGIS_VERSION = (2, 3, 0)
```

This will instruct GeoNode to listen on connections from your local machine.

Change the value of the *SITEURL*:

```
SITEURL = "http://localhost/"
```

Now configure database access: Uncomment the *ENGINE*: *'django.contrib.gis.db.backends.postgis'* line and comment the one with empty *ENGINE* variable. Also set the *NAME* variable to *geonode_data*:

```
DATABASES = {
    'default': {
        ...
    },
    'datastore' : {
        'ENGINE': 'django.contrib.gis.db.backends.postgis',
        #'ENGINE': '', # Empty ENGINE name disables,
        'NAME': 'geonode_data',
        ...
    }
}
```

Then configure GeoServer location: Change the value of the *LOCATION* and *PUBLIC_LOCATION* variables as below:

```
GEOSERVER_LOCATION = os.getenv(
    'GEOSERVER_LOCATION', 'http://localhost/geoserver/'
)
GEOSERVER_PUBLIC_LOCATION = os.getenv(
    'GEOSERVER_PUBLIC_LOCATION', 'http://localhost/geoserver/'
)
```

Finally configure GeoServer datastore: Change the value of the *DATASTORE* variable to the dictionary identifier of database for spatial data as below:

```
OGC_SERVER = {
    'default': {
        ...
        'DATASTORE': 'datastore',
    }
}
```

The resulting configuration file should look like this::

```
import os

PROJECT_ROOT = os.path.abspath(os.path.dirname(__file__))

ALLOWED_HOSTS = ['127.0.0.1', 'localhost', '::1']
PROXY_ALLOWED_HOSTS = ('127.0.0.1', 'localhost', '::1', )

POSTGIS_VERSION = (2, 3, 0)

SITEURL = "http://localhost/"

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'geonode',
        'USER': 'geonode',
        'PASSWORD': 'geonode',
    },
    # vector datastore for uploads
    'datastore': {
        'ENGINE': 'django.contrib.gis.db.backends.postgis',
        #'ENGINE': '', # Empty ENGINE name disables
        'NAME': 'geonode_data',
        'USER': 'geonode',
        'PASSWORD': 'geonode',
        'HOST': 'localhost',
        'PORT': '5432',
    }
}

GEOSERVER_LOCATION = os.getenv(
    'GEOSERVER_LOCATION', 'http://localhost/geoserver/'
)
GEOSERVER_PUBLIC_LOCATION = os.getenv(
    'GEOSERVER_PUBLIC_LOCATION', 'http://localhost:8001/geoserver/'
)
```

(continues on next page)

(continued from previous page)

```

# OGC (WMS/WFS/WCS) Server Settings
OGC_SERVER = {
    'default': {
        'BACKEND': 'geonode.geoserver',
        'LOCATION': GEOSERVER_LOCATION,
        'LOGIN_ENDPOINT': 'j_spring_oauth2_geonode_login',
        'LOGOUT_ENDPOINT': 'j_spring_oauth2_geonode_logout',
        # PUBLIC_LOCATION needs to be kept like this because in dev mode
        # the proxy won't work and the integration tests will fail
        # the entire block has to be overridden in the local_settings
        'PUBLIC_LOCATION': GEOSERVER_PUBLIC_LOCATION,
        'USER' : 'admin',
        'PASSWORD' : 'geoserver',
        'MAPFISH_PRINT_ENABLED' : True,
        'PRINT_NG_ENABLED' : True,
        'GEONODE_SECURITY_ENABLED' : True,
        'GEOGIG_ENABLED' : False,
        'WMST_ENABLED' : False,
        'BACKEND_WRITE_ENABLED': True,
        'WPS_ENABLED' : False,
        'LOG_FILE': '%s/geoserver/data/logs/geoserver.log' % os.path.abspath(os.path.
→join(PROJECT_ROOT, os.pardir)),
        # Set to dictionary identifier of database containing spatial data in_
→DATABASES dictionary to enable
        'DATASTORE': 'datastore', #'datastore',
    }
}

# If you want to enable Mosaics use the following configuration
#UPLOADER = {
##     'BACKEND': 'geonode.rest',
#     'BACKEND': 'geonode.importer',
#     'OPTIONS': {
#         'TIME_ENABLED': True,
#         'MOSAIC_ENABLED': True,
#         'GEOGIG_ENABLED': False,
#     }
#}

CATALOGUE = {
    'default': {
        # The underlying CSW implementation
        # default is pycsw in local mode (tied directly to GeoNode Django DB)
        'ENGINE': 'geonode.catalogue.backends.pycsw_local',
        # pycsw in non-local mode
        # 'ENGINE': 'geonode.catalogue.backends.pycsw_http',
        # GeoNetwork opensource
        # 'ENGINE': 'geonode.catalogue.backends.geonetwork',
        # deegree and others
        # 'ENGINE': 'geonode.catalogue.backends.generic',

        # The FULLY QUALIFIED base url to the CSW instance for this GeoNode
        'URL': '%scatalogue/csw' % SITEURL,
        # 'URL': 'http://localhost:8080/geonetwork/srv/en/csw',
        # 'URL': 'http://localhost:8080/deegree-csw-demo-3.0.4/services',
    }
}

```

(continues on next page)

(continued from previous page)

```

    # login credentials (for GeoNetwork)
    'USER': 'admin',
    'PASSWORD': 'admin',
}
}

# Default preview library
# GEONODE_CLIENT_LAYER_PREVIEW_LIBRARY = 'geoext'

```

Initialize GeoNode

As user geonode, initialize the db, by creating the schema tables and populating the initial data::

```

cd /opt/apps/geonode/geonode/
python manage.py migrate
python manage.py loaddata geonode/base/fixtures/initial_data.json

```

Now create the admin user for GeoNode running the following:

```
python manage.py createsuperuser
```

You will be prompted for the username, email address and password for the user.

Warning: It is required to have at least one admin user called *admin*. If you want to break for security concerns this rule then you have to edit the default OAuth2 initial configuration `geonode/base/fixtures/default_oauth_apps.json` accordingly.

Finally create the default configuration for the OAuth2 application:

```
python manage.py loaddata geonode/base/fixtures/default_oauth_apps.json
```

In case your GeoNode will be exposed on a different port (i.e. 8001 in case of vagrant port forwarding) from the default 80 please make sure to do not forget to run the following commands:

Fix the site url from the `local_settings` file:

```
python manage.py fixsitename
```

Setting up OAuth2 configuration:

```

# Set oauth keys
export GEOSERVER_DATA_DIR=/var/lib/tomcat/webapps/geoserver/data
export NEWIP=localhost:8001 # Case of proxied geonode for example
oauth_keys=$(python manage.py fixoauthuri 2>&1)
oauth_keys_cleaned=$(echo $oauth_keys | cut -d " " -f 15)
client_id=`echo $oauth_keys_cleaned | cut -d \, -f 1`
client_secret=`echo $oauth_keys_cleaned | cut -d \, -f 2`
oauth_config="$GEOSERVER_DATA_DIR/security/filter/geonode-oauth2/config.xml"
sudo sed -i "s|<cliendId>.*</cliendId>|<cliendId>$client_id</cliendId>|g" $oauth_
↪config
sudo sed -i "s|<clientSecret>.*</clientSecret>|<clientSecret>$client_secret</
↪clientSecret>|g" $oauth_config
sudo sed -i "s|localhost:8000/localhost/g" $oauth_config

```

(continues on next page)

(continued from previous page)

```
sudo sed -i "s|<userAuthorizationUri>.*</userAuthorizationUri>|<userAuthorizationUri>  
↪http://$NEWIP/o/authorize/</userAuthorizationUri>|g" $oauth_config  
sudo sed -i "s|<logoutUri>.*</logoutUri>|<logoutUri>http://$NEWIP/account/logout/</  
↪logoutUri>|g" $oauth_config  
sudo sed -i "s|localhost:8080/geoserver|$NEWIP/geoserver/|g" $oauth_config  
sudo sed -i "s|localhost:8000|localhost|g" "$GEOSEVER_DATA_DIR/security/role/geonode_  
↪REST role service/config.xml"
```

Restart Tomcat to load new configuration settings:

```
sudo systemctl restart tomcat@geoserver
```

Run the command **updatelayers**:

```
python manage.py updatelayers
```

Run the command **updatemaplayerip**:

```
python manage.py updatemaplayerip
```

Download GeoNode data to be served by Apache. You will be prompted for confirmation:

```
python manage.py collectstatic
```

Create *uploaded* folder:

```
mkdir -p /opt/apps/geonode/geonode/geonode/uploaded/
```

Isolate GeoNode installation

In case you prefer to isolate the GeoNode installation from the rest of your OS the most used strategy is to rely on a virtual environment. You can also install all the dependencies with a totally separated version of the python binary as explained in the section `install_venv_httpd`.

Apache HTTP Server Installation

Install and setup base packages

Install Apache:

```
sudo yum install -y httpd
```

And additional modules:

```
sudo yum install -y mod_ssl mod_proxy_html mod_wsgi
```

Important: Please be aware that the WSGI module for Apache shipped by the CentOS repositories is pretty much old and obsolete. It is recommended to separately replace it with the latest binary from the official `mod_wsgi` project. Referer to `install_venv_httpd_wsgi`. Alternatively you can replace your HTTP web server with the combination of `Nginx` and `Gunicorn` for publishing GeoServer and the Django application as explained in `install_venv_nginx_gunicorn`.

Firewall configuration

Allow requests on port 80 through the firewall:

```
sudo firewall-cmd --zone=public --add-service=http --permanent
sudo firewall-cmd --reload
```

Security issues

There are a couple of security issues to fix when dealing with GeoNode.

GeoNode will run inside httpd through WSGI. This means that httpd will try to perform external connection toward the DB. This is usually blocked by default by strict security policies, so we need to relax them:

```
sudo setsebool -P httpd_can_network_connect_db 1
```

The other issue is about SELinux itself: it is not WSGI friendly, so we'll have to disable it. Edit the file `/etc/sysconfig/selinux` and change the line:

```
SELINUX=enforcing
```

into:

```
SELINUX=permissive
```

and reboot the machine.

httpd configuration

As root, create the file `/etc/httpd/conf.d/geonode.conf` and insert into it this content.

Add thumbs and layers folders:

```
mkdir -p /opt/apps/geonode/geonode/geonode/static_root
mkdir -p /opt/apps/geonode/geonode/geonode/uploaded/thumbs
mkdir -p /opt/apps/geonode/geonode/geonode/uploaded/layers
```

Change permissions on GeoNode files and folders to allow Apache to read and edit them::

```
chmod +x /opt/apps/geonode/
sudo chown -R geonode /opt/apps/geonode/geonode
sudo chown geonode:apache /opt/apps/geonode/geonode/geonode/static/
sudo chown geonode:apache /opt/apps/geonode/geonode/geonode/uploaded/
sudo chown geonode:apache /opt/apps/geonode/geonode/geonode/static_root/
chmod -Rf 777 /opt/apps/geonode/geonode/geonode/uploaded/thumbs
chmod -Rf 777 /opt/apps/geonode/geonode/geonode/uploaded/layers
```

SSL configuration

If a secure HTTP communication is needed then you have to add a virtual host listening on a secure port::

```
Listen 443
<VirtualHost *:443>
    ServerName https://localhost
    SSLEngine on
    SSLCertificateFile "/path/to/demo.geonode.org.cert"
    SSLCertificateKeyFile "/path/to/demo.geonode.org.key"
</VirtualHost>
```

In some cases even the proxy pass has to challenge with a secured GeoServer instance listening in HTTPS. In this situation Apache has to verify the remote server certificate with the certificate of its own Certification Authority (CA). For this purpose the concatenation of the various PEM-encoded certificate files can be used accordingly with this directive::

```
SSLProxyCACertificateFile /usr/local/apache2/conf/ssl.crt/ca-bundle-geoserver-remote-
↪server.crt
```

Alternatively the directive that sets the directory where you keep the certificates of Certification Authorities (CAs) whose remote servers you deal with can be used::

```
SSLProxyCACertificatePath /usr/local/apache2/conf/ssl.crt/
```

Note: If the verification of GeoServer certificate is not required then the SSL proxy has to be instructed with a directive which excludes the need of a valid certificate::

```
SSLProxyVerify none (instead of "require")
```

If a strong authentication with client certificates is needed then the secure virtual host has to contain at least these directives::

```
SSLVerifyClient require
SSLVerifyDepth 1
SSLCACertificateFile "conf/ssl.crt/geonode-ca.crt"
```

Note: This configuration above requires a client certificate which has to be directly signed by the GeoNode CA certificate in geonode-ca.crt. In certain cases you do not want the verification of GeoServer certificate as mandatory hence it is enough to apply the value *none*

Quick administration

If you change any directive then restart httpd to make it reload the new configurations::

```
sudo systemctl restart httpd
```

To automatically start Apache at boot, run::

```
sudo systemctl enable httpd
```

3.2.1.2.9 Network configuration issues

In this section you will find instructions how to understand any problem of connectivity when GeoNode is being exposed through a network different either from a local computer or a server.

GeoNode being proxied

A similar situations can be encountered in this scenarios:

- GeoNode behind a proxy server like [HAProxy](#) or [Squid](#).
- GeoNode in a Vagrant machine with NAT mode and port forwarding

Development mode

Note: Please note that this section is relevant **only** if your development machine is a Vagrant box and the GeoNode application is being accessed from a browser and an IP address of your host machine, usually your computer.

Assuming the following port forwarding configuration:

```
+-----+-----+-----+
```

Component | Host port | Guest port |

```
+=====+=====+=====+ | Django | 8001
| 8000 | +-----+-----+ | GeoServer | 8080 | 8080 |
+-----+-----+-----+
```

Important: In such a situation it is mandatory to start your development server on all IPv4 addresses of your guest machine in order to be reachable from the host.

```
python manage.py runserver 0.0.0.0:8000
```

or with Paver

```
paver start_django -b 0.0.0.0:8000
```

You have to review and make sure the following configurations are applied in GeoServer for correct communications:

```
- Configuration of GeoNode REST role service with proper `baseUrl` in the
↪:file:`config.xml` under the directory `$GEOSERVER_DATA_DIR/security/role/geonode\
↪REST\ role\ service/`
```

```
<baseUrl>http://localhost:8000/</baseUrl>
<!-- base url of geonode web server -->
```

- Configuration of GeoServer security for the oauth2 provider in the `config.xml` under the directory `$GEOSERVER_DATA_DIR/security/filter/geonode-oauth2/`

```
<!-- GeoNode accessTokenUri -->
<accessTokenUri>http://localhost:8000/o/token/</accessTokenUri>

<!-- GeoNode userAuthorizationUri -->
<userAuthorizationUri>http://localhost:8001/o/authorize/</
<userAuthorizationUri>

<!-- GeoServer Public URL -->
<redirectUri>http://localhost:8080/geoserver</redirectUri>

<!-- GeoNode checkTokenEndpointUrl -->
<checkTokenEndpointUrl>http://localhost:8000/api/o/v4/tokeninfo/</
<checkTokenEndpointUrl>

<!-- GeoNode logoutUri -->
<logoutUri>http://localhost:8001/account/logout/</logoutUri>

<proxyBaseUrl>http://localhost:80/geoserver</proxyBaseUrl>
<!-- proxy base url of geonode web server -->
```

GeoNode outbound connections

SELinux

Security-Enhanced Linux (**SELinux**) is a security mechanism implemented at kernel level. Generally when SELinux is *enabled* communication issues could arise. First of all let's see how to have a look at its status with this command:

```
.. code-block:: console

sestatus
```

The possible values of *SELinux status* can be *enabled* or *disabled* while if it is enabled the *Current mode* can vary between *enforcing* and *permissive*. If SELinux is enabled its policies will only allow services access to recognized ports associated with those services. For example if we wanted to allow Django server to listen on tcp port 800 then a new rule has to be added for such purpose. Simply by using the command *semanage* below:

```
.. code-block:: console

sudo semanage port -a -t http_port_t -p tcp 8000
```

Verify if the rule has been achieved by running:

```
.. code-block:: console

sudo semanage port -l
```

3.2.1.2.10 Windows Binary Installer

The GeoNode Windows Binary installer can be downloaded [here](#).

Current Versions

1. [GeoNode-2.4.x.exe](#); Old version of GeoNode, not fully maintained anymore.
2. [GeoNode-2.5.5.exe](#); Maintenance version of GeoNode, still based on old security system.
3. [GeoNode-2.6.x.exe](#); Stable version of GeoNode, based on new OAuth2 based security system.
4. [GeoNode-master.exe](#); Development version of GeoNode, not stable.

The Installer

The Windows binary installer is an executable file which automatically sets up your system for GeoNode. It will install everything on the target folder, chosen by the user as a first step.

In particular the installer configures the following components:

1. An instance of the Oracle JRE 32bits
2. An instance of Python 2.7.10 32bit with GDAL 1.11 extensions
3. PostgreSQL with PostGIS spatial extensions
4. Apache Tomcat with GeoServer for GeoNode webapp
5. Apache HTTPD Server 2.49 (WinLAMP version)

The source code of the installer, along with all the components needed to build a custom executable and instructions, can be found at <https://github.com/GeoNode/geonode-win-installer>

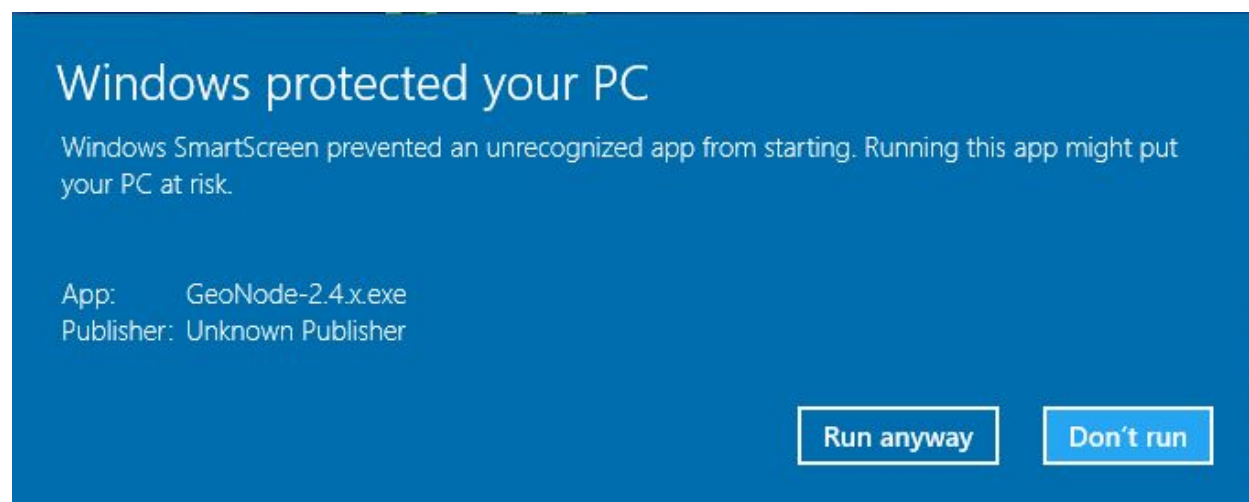
Warning: In order to install GeoNode using the binary installer, you will need Windows Administrator rights.

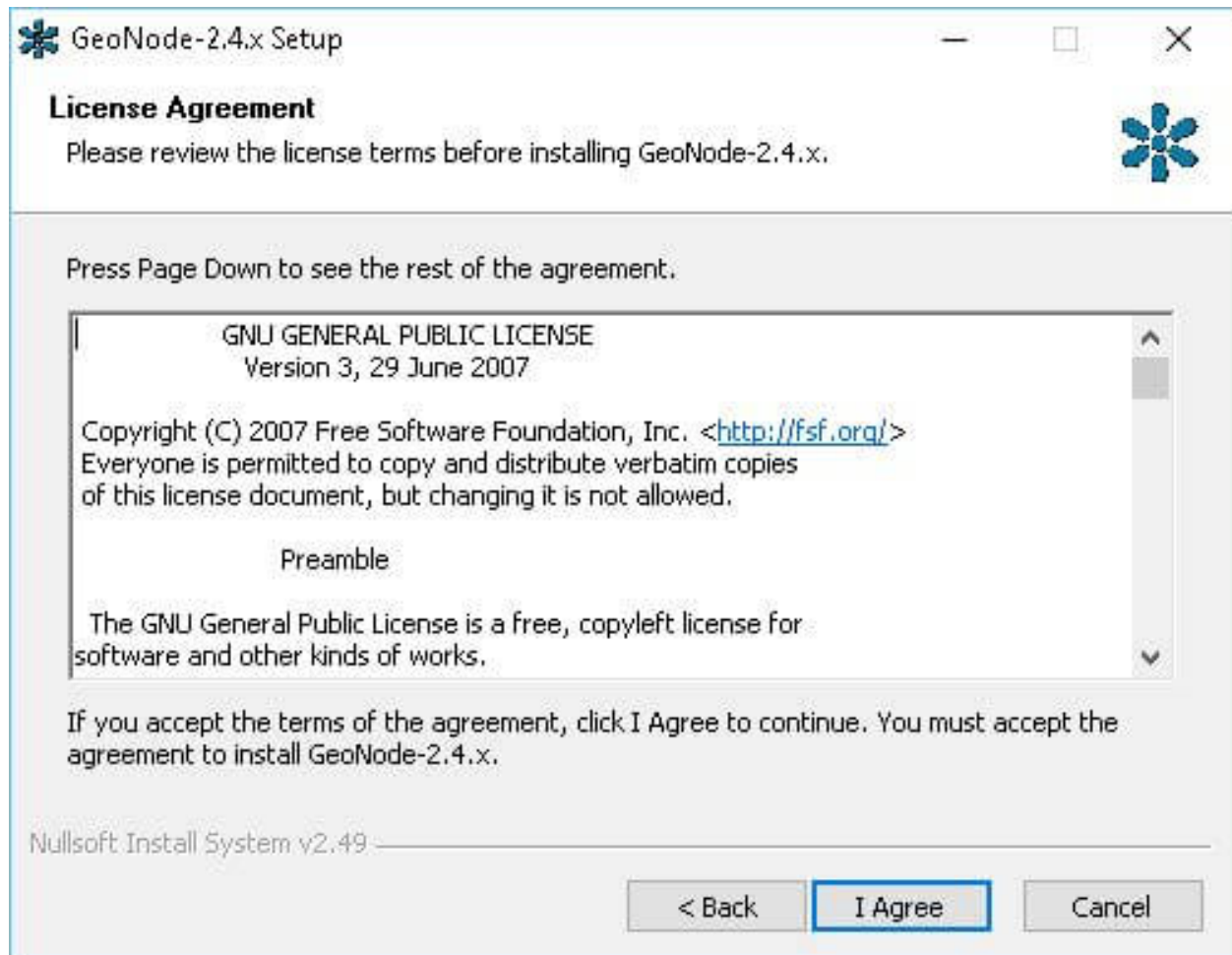
Step by step GeoNode Windows installation

1. Execute the binary [file](#) downloaded from GeoNode servers



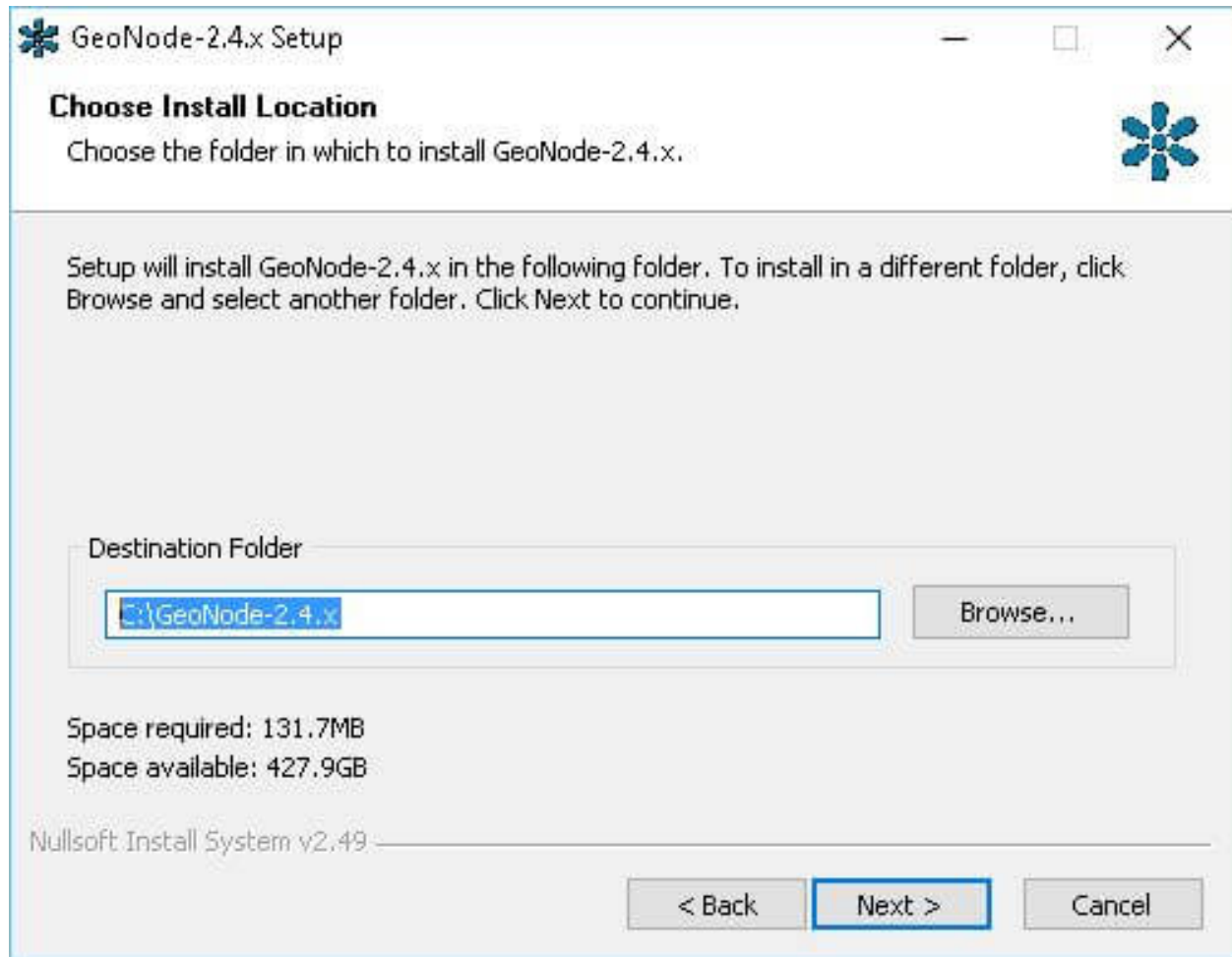
2. The first time you could get a Windows warning about the unknown publisher. Click on “Run Anyway” button in order to proceed
3. Click “Next” at the welcome screen
4. Read and accept the licence agreement on the next screen





5. Choose the install location. This is the folder where all the pieces of the GeoNode environment will be placed. You can also keep the default one.

Note: It is highly recommended to avoid long names with spaces and strange character. Also avoid positions which are system protected. Even if the installer runs with Administrator rights, you may encounter issues customizing GeoNode in the future.

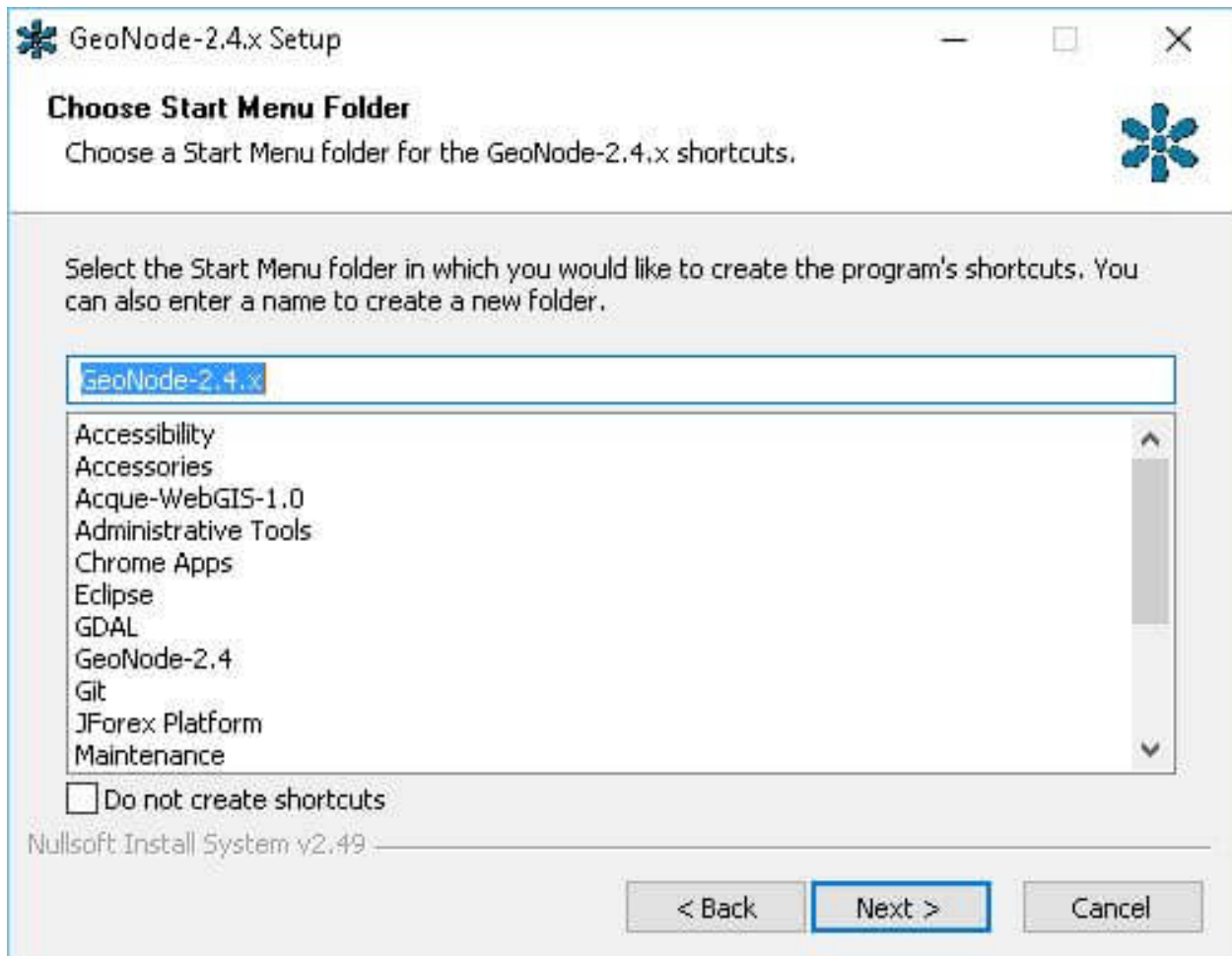


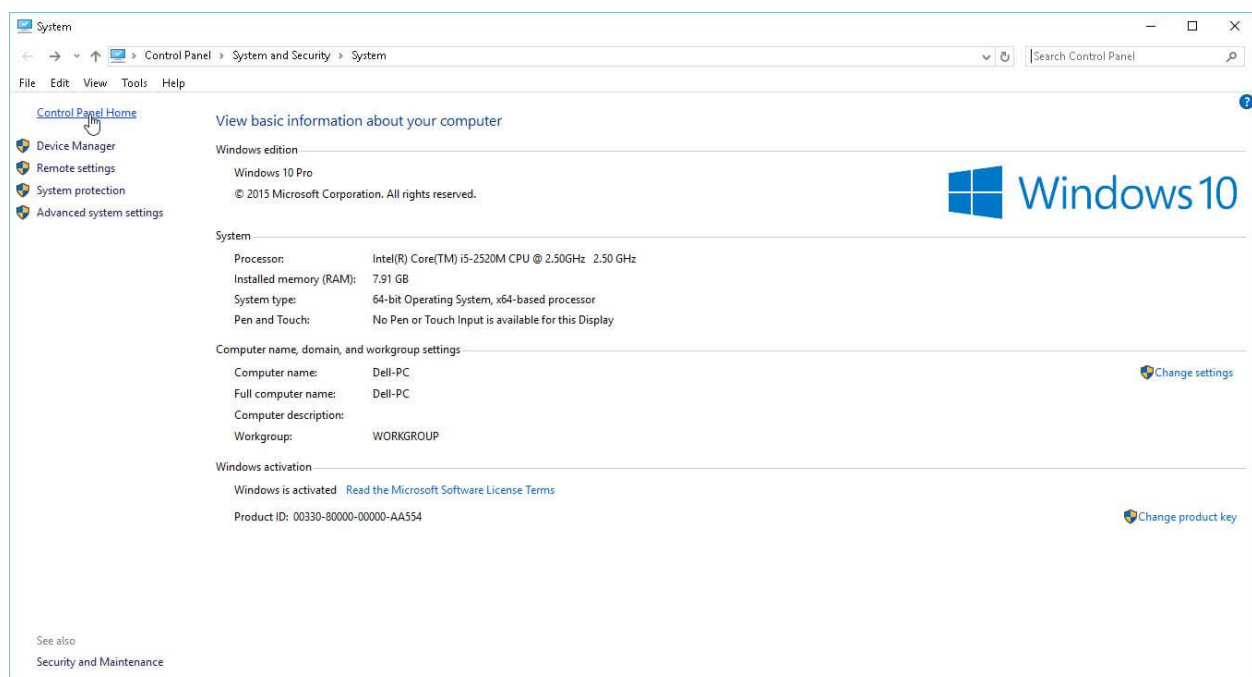
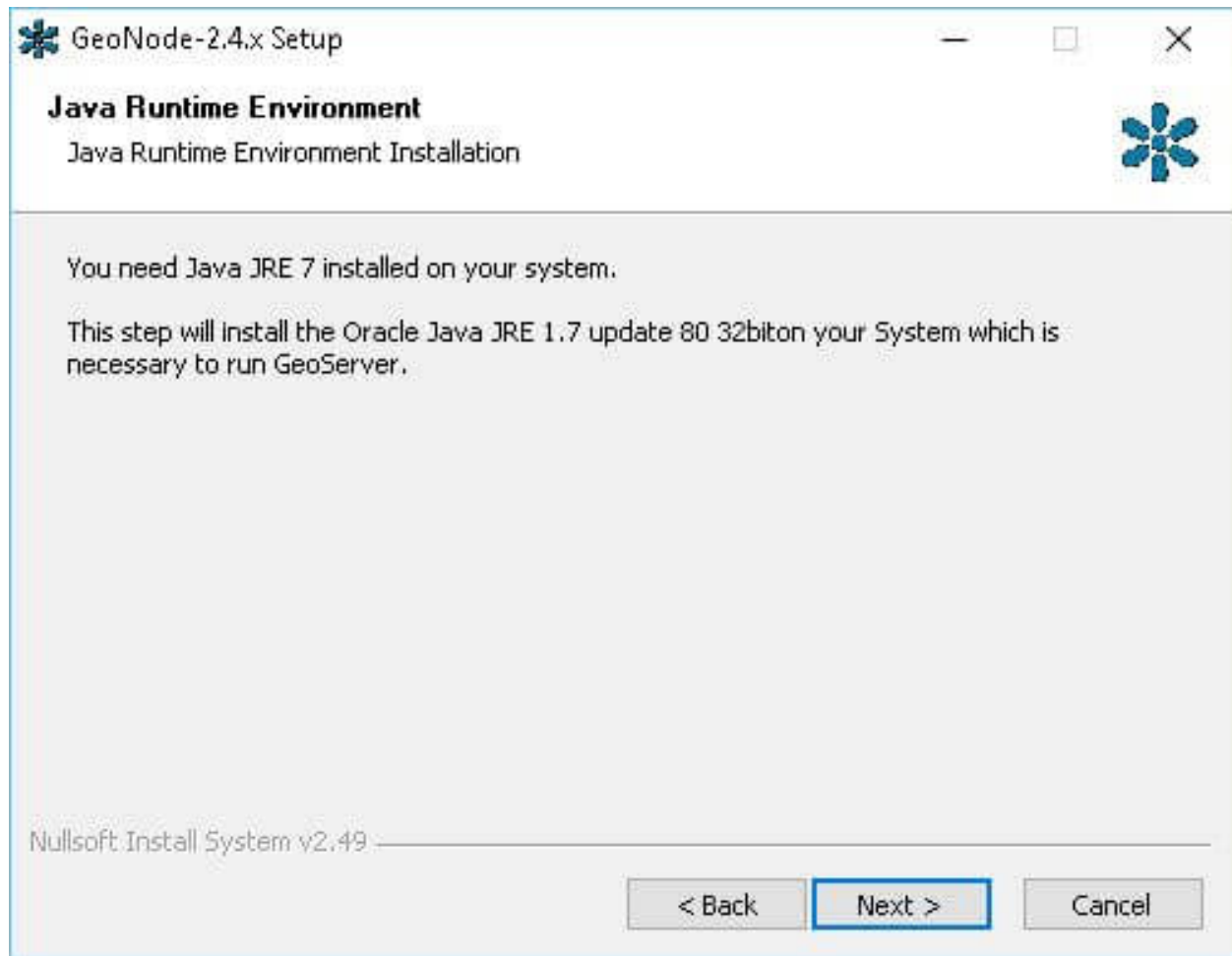
6. Choose the name of the Start Menu folder. This is the name of the folder which will be created into the “Windows App Toolbar”. You can also keep the default one.
7. Next step will install the JEW 1.7 32bit version on the installation folder.

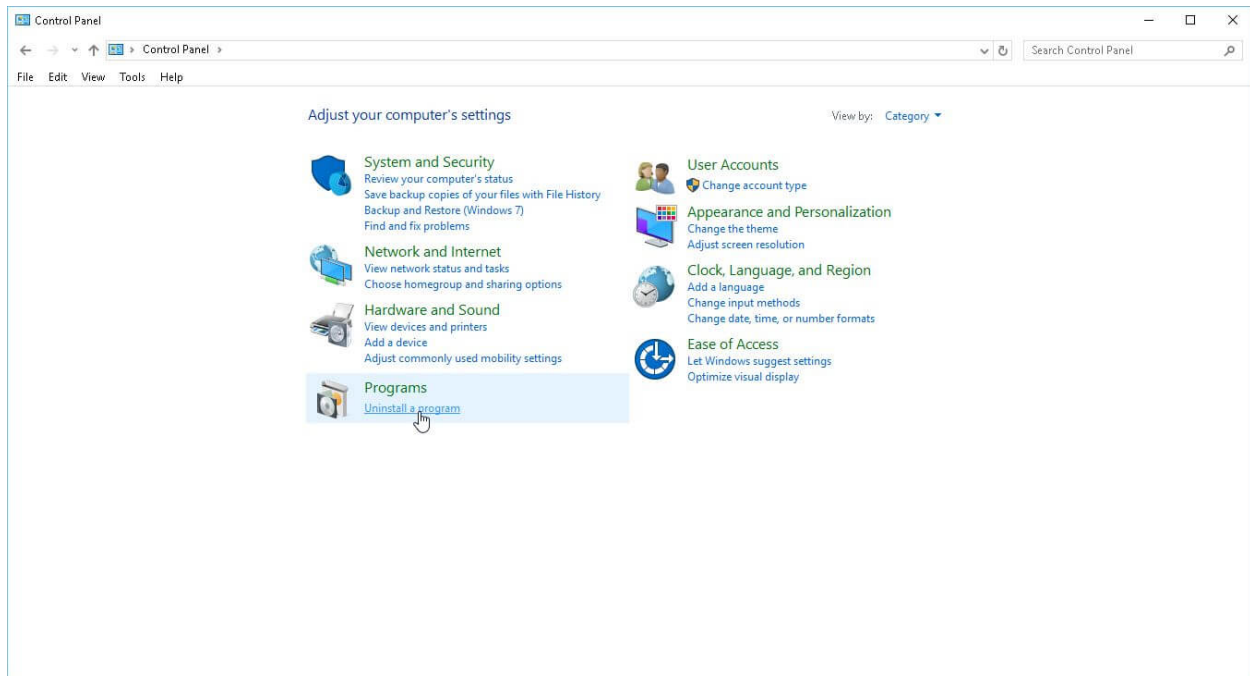
Warning: Before proceeding, please be sure you don’t have other JRE 1.7 32bit installation already present on your system. In order to do that please follow the instructions below:

Go to the “Control Panel”

Click on “Uninstall a Program”







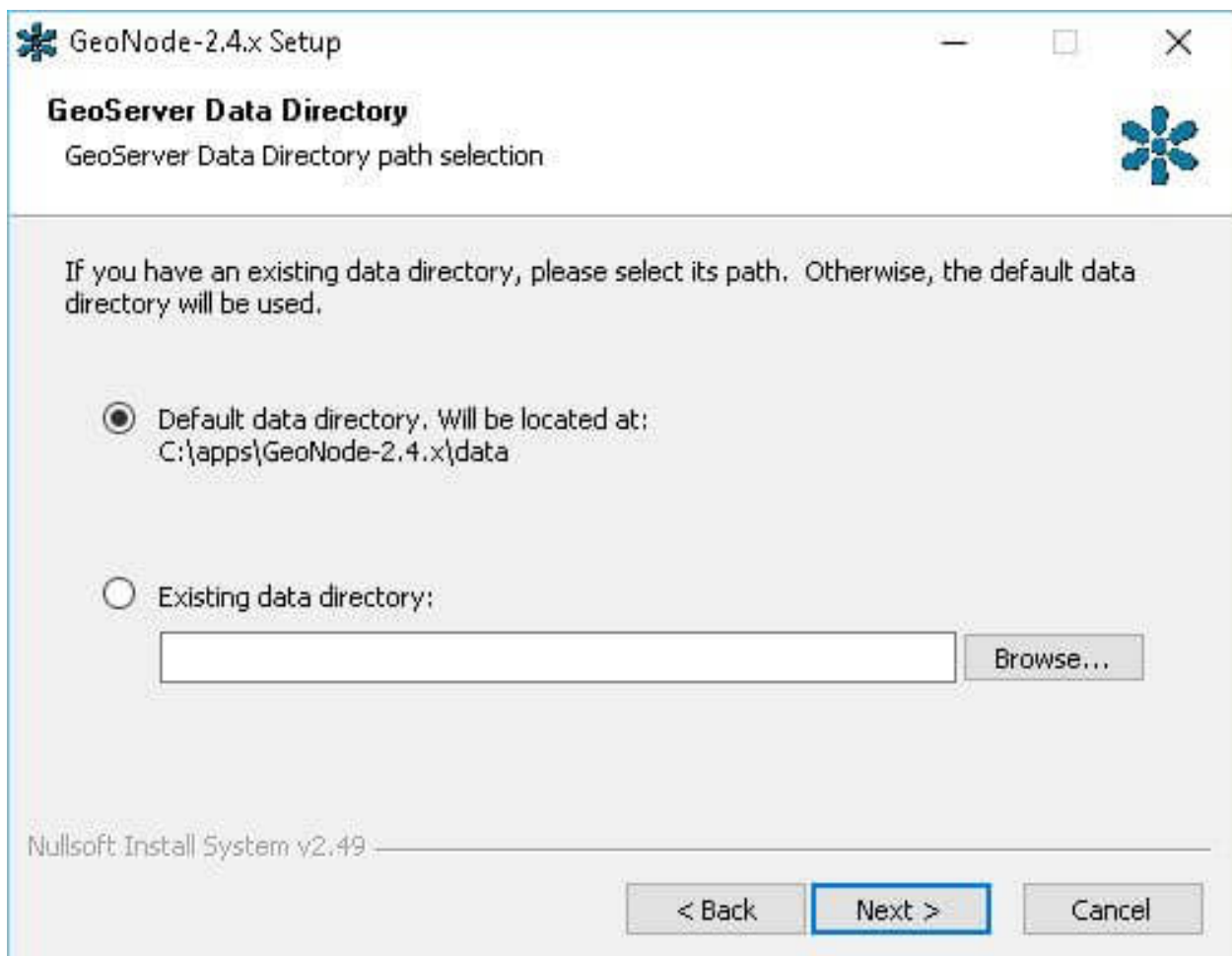
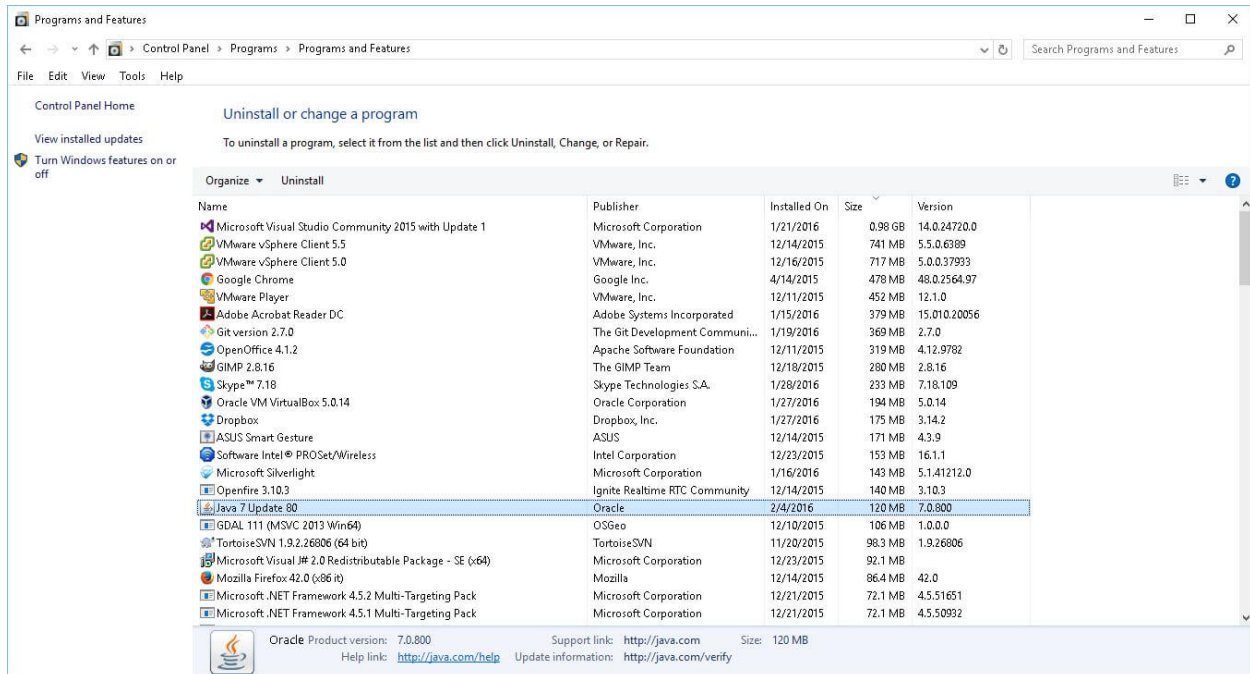
Check that the menu does not contain entries similar to the one shown in the following figure

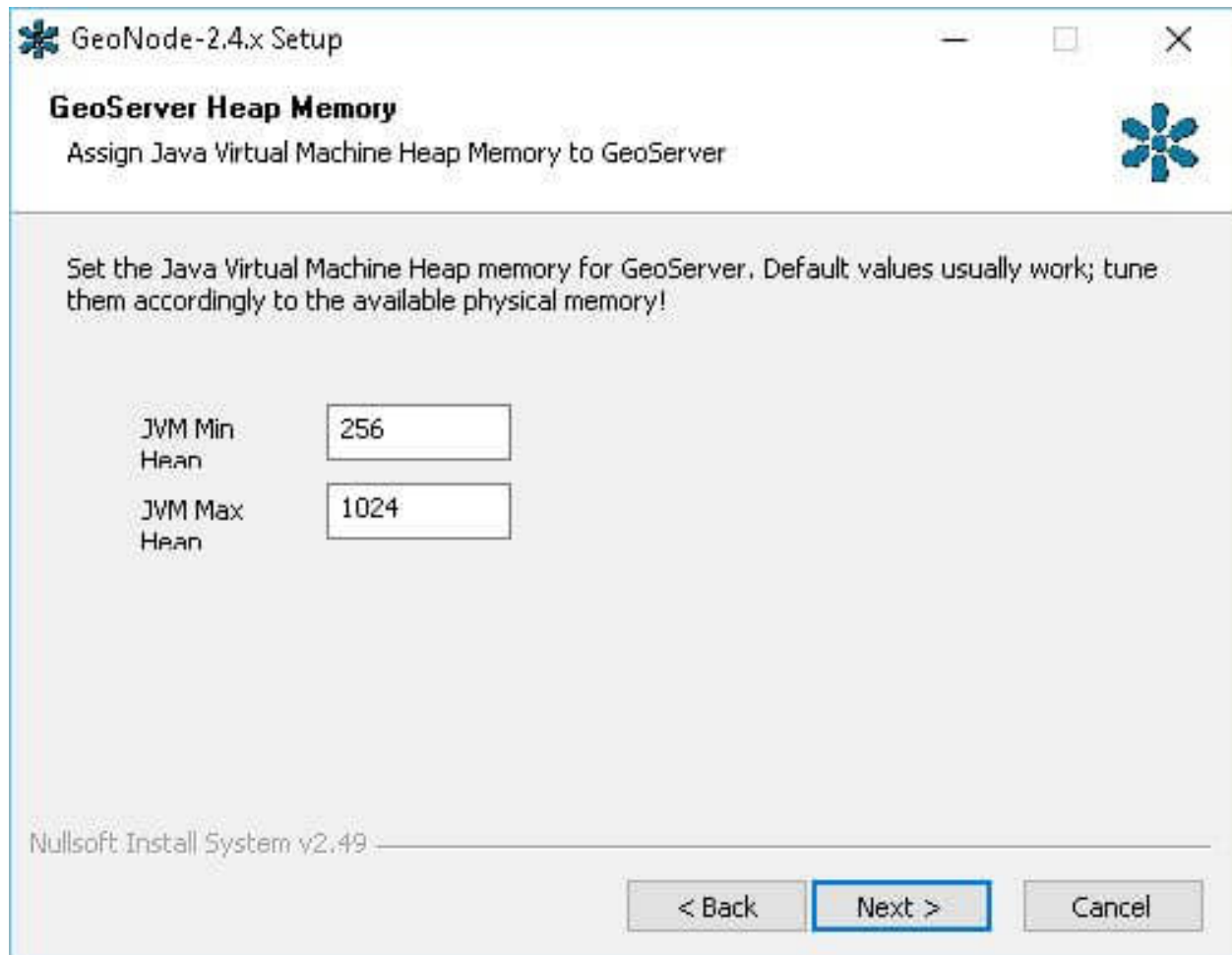
8. Choose the location of the external GeoServer Data directory (where the physical geospatial files will be placed). It is recommended to keep the default value.
9. Select the minimum and maximum values for the memory to assign to the GeoServer JVM. This depends mainly from the RAM available on the target PC. The default values should work on most cases. Increase the Maximum Heap Memory if possible.
10. Choose the user and password for the GeoServer Administrator **outside** GeoNode authentication. The form presents the default ones.
11. Choose the TCP/IP port to run Apache Tomcat. The default value (8080) is widely used from a lot of installation. If your system is not fully clean, it is kindly suggested to choose another value between 1024 and 65535.
12. Proceed with the installation of Python 2.7.10 32bit.
This will be placed into the target GeoNode installation folder along with GDAL 1.11 extensions
13. Proceed with the installation of PostgreSQL 8.4.22 and PostGIS 1.5.4.

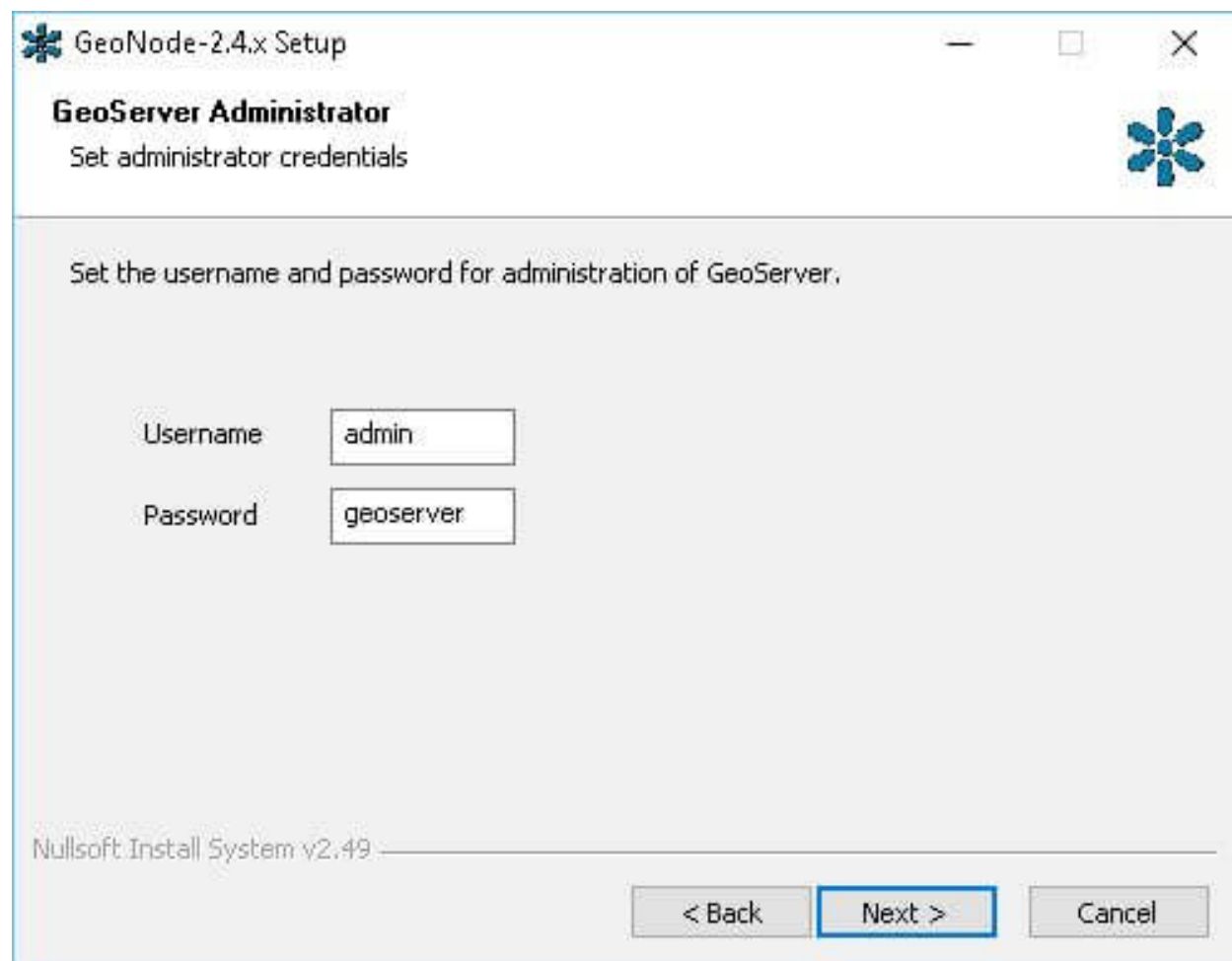
Note: The installation proceeds silently and using the default values:

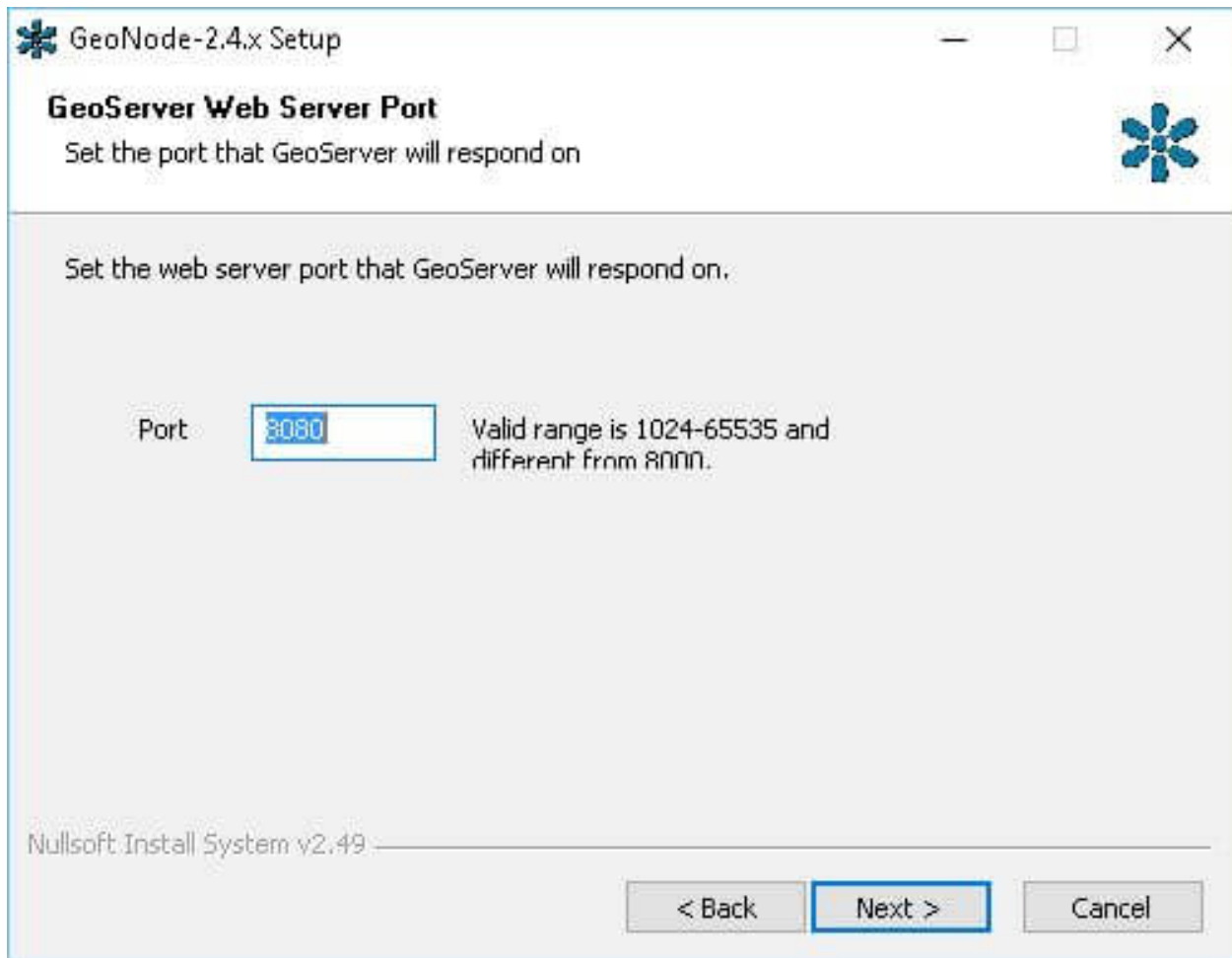
- PostgreSQL: TCP/IP Port 5454
 - User/password: postgres/g30Nod3-P0stgr3s
-

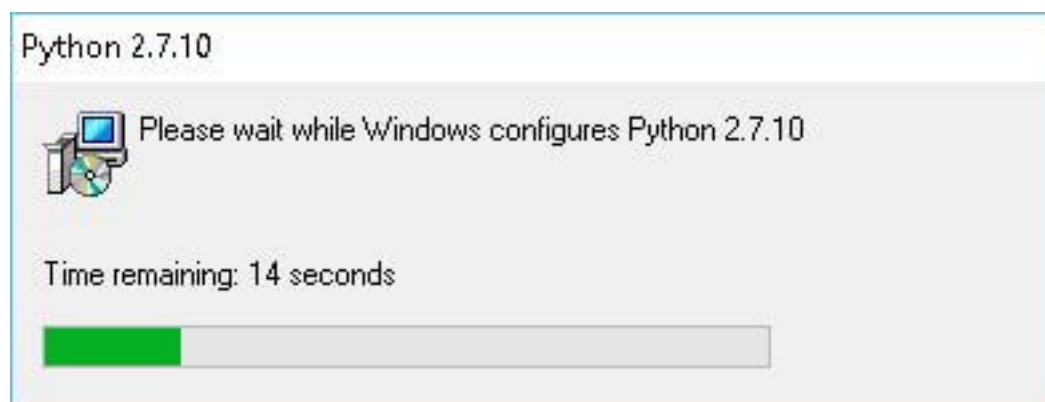
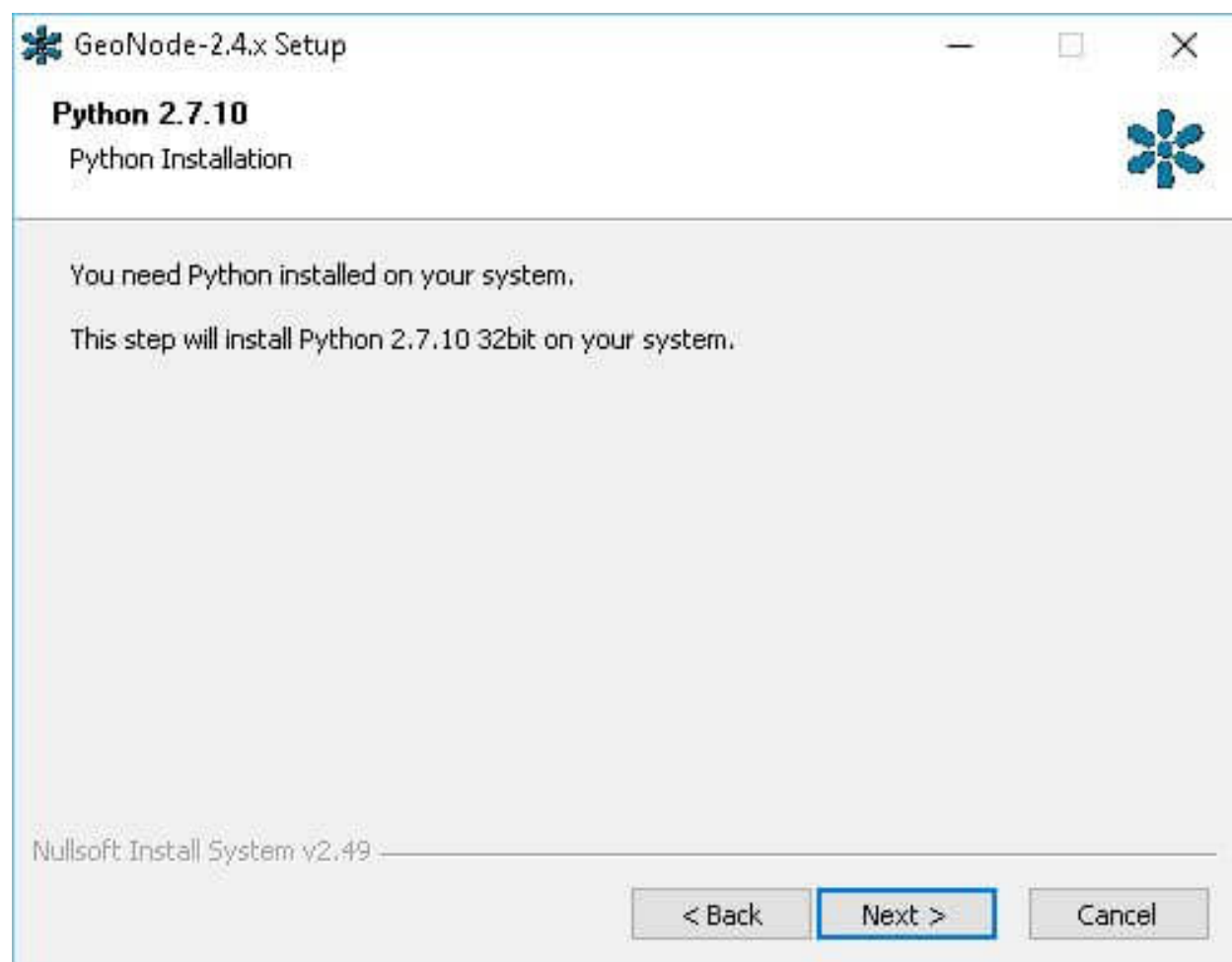
14. Ready to install. The page summarizes the user choices and asks to finish the installation process.

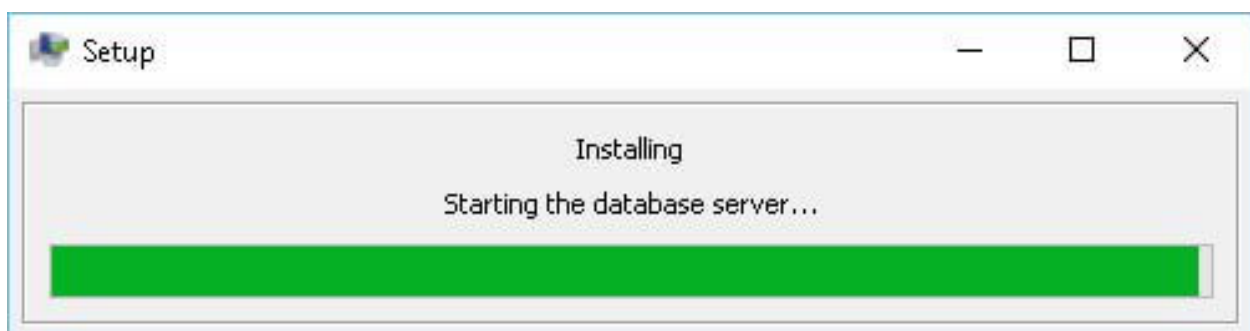
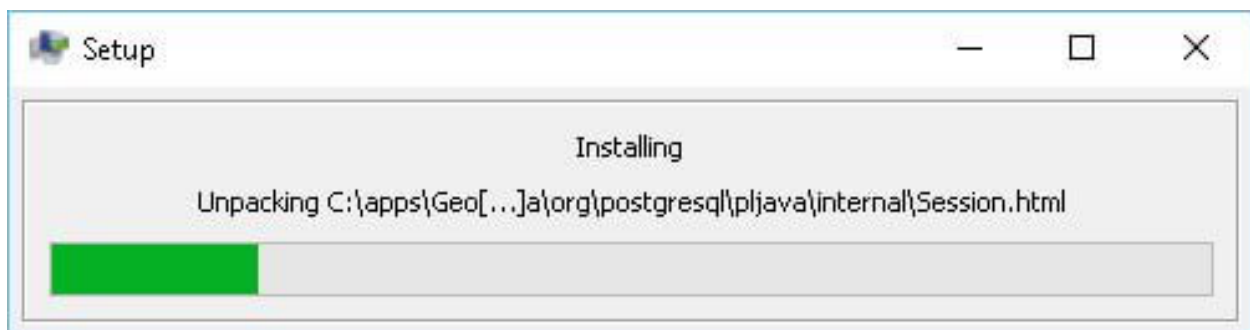
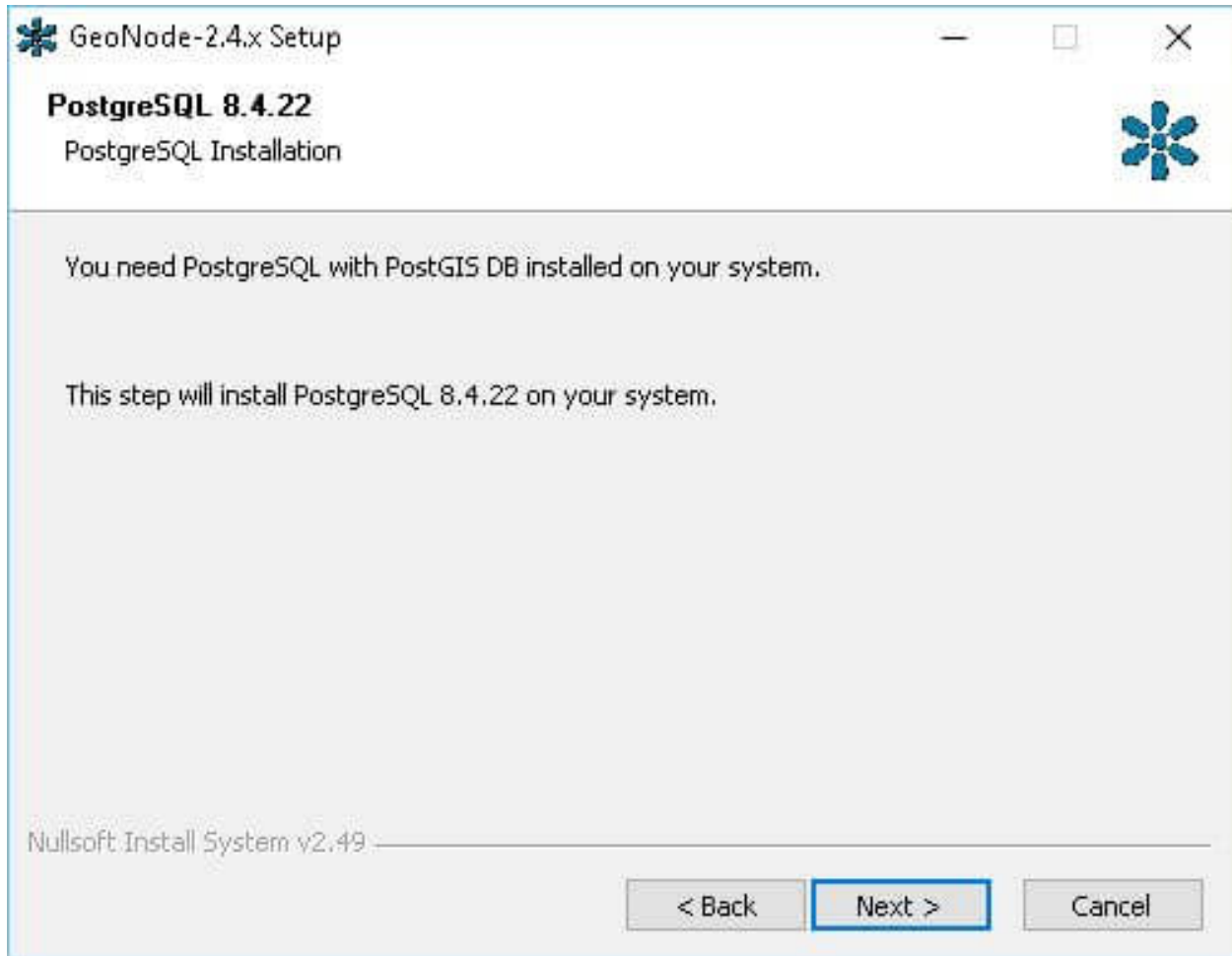


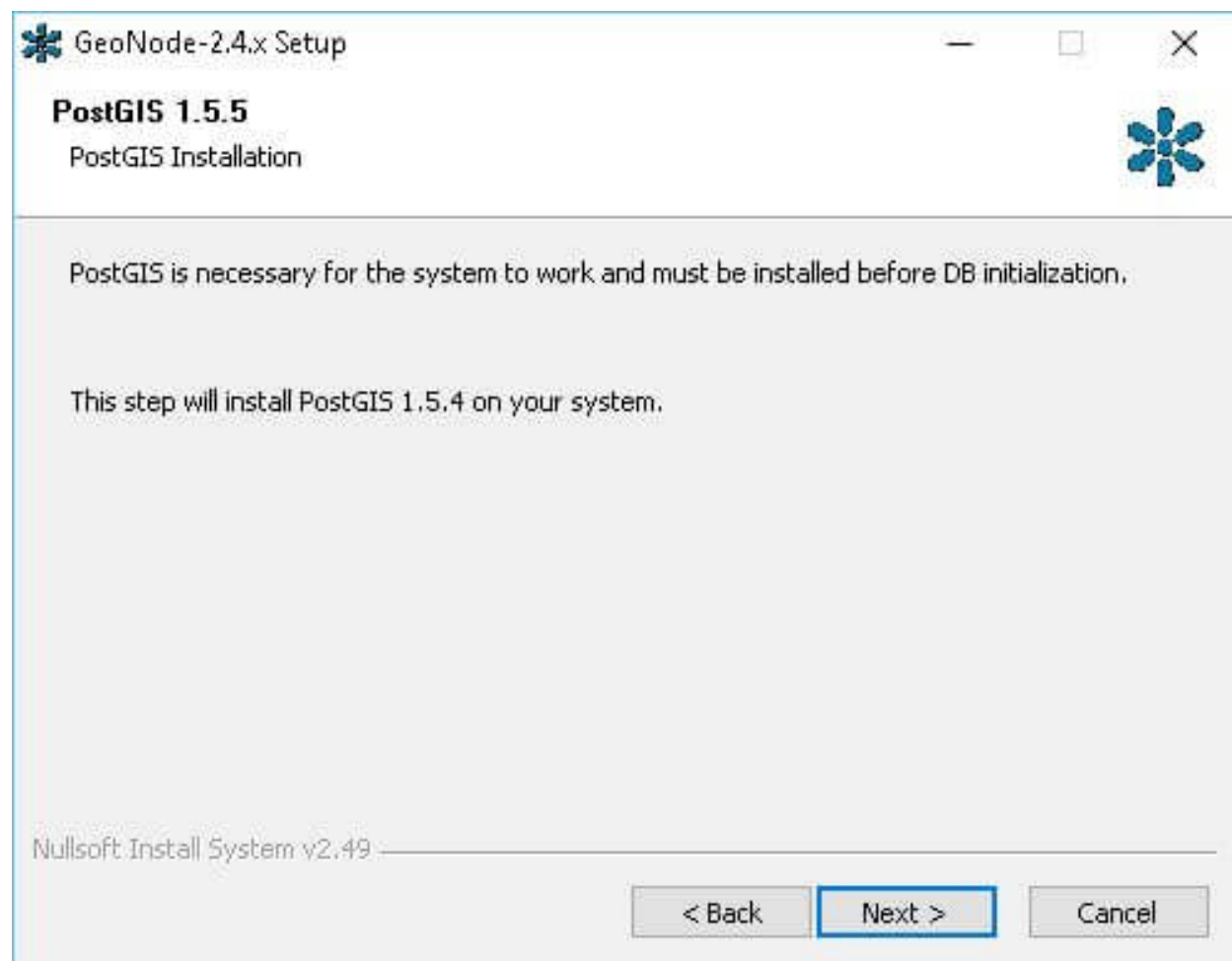


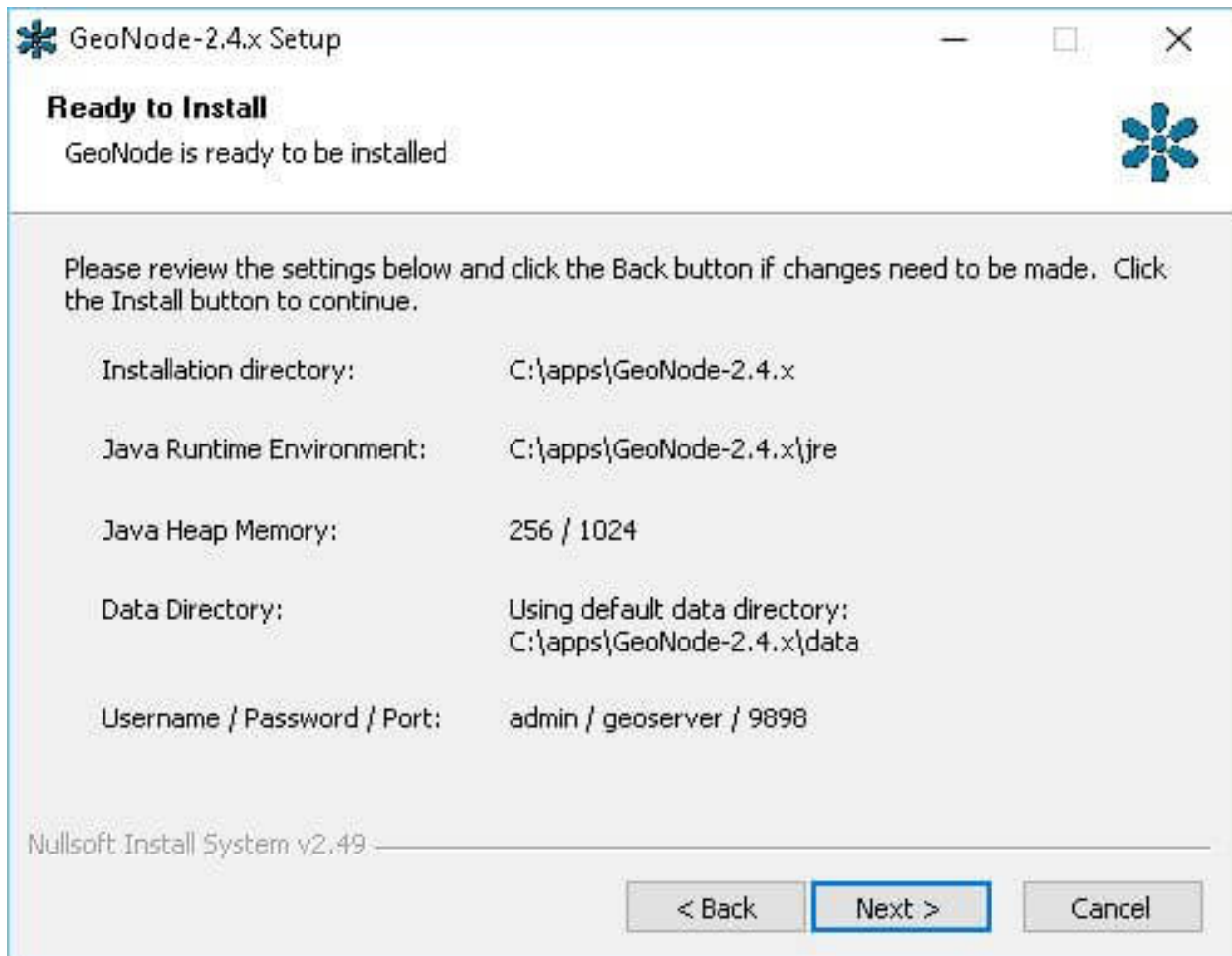




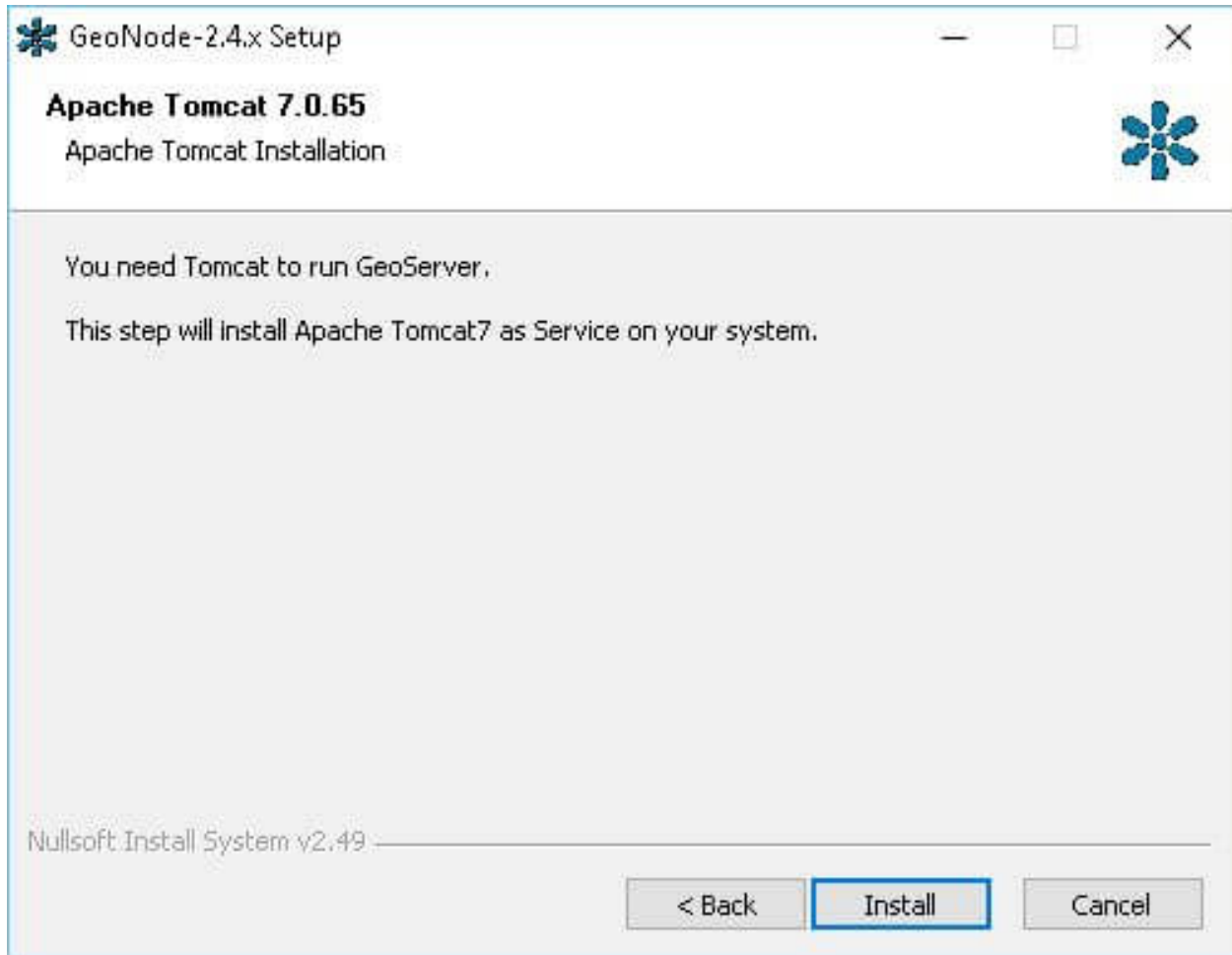




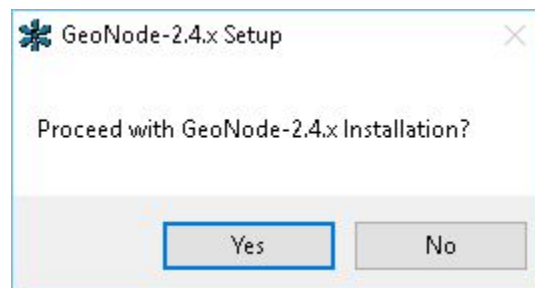




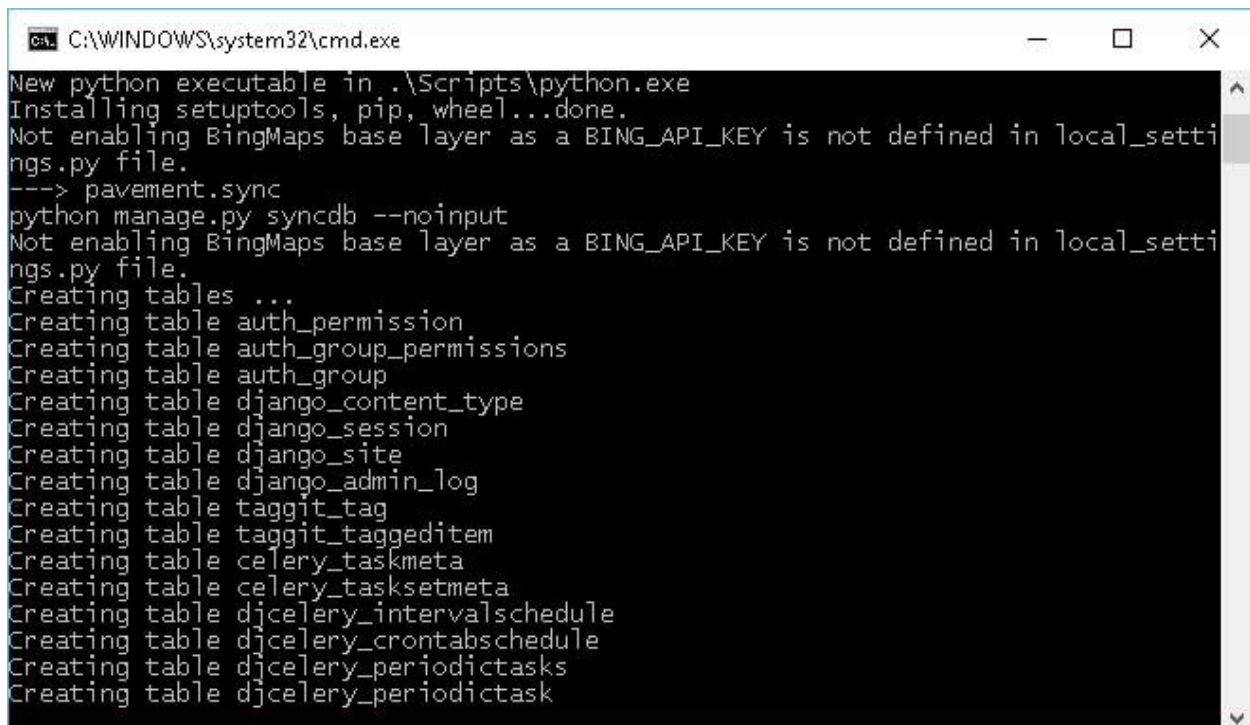
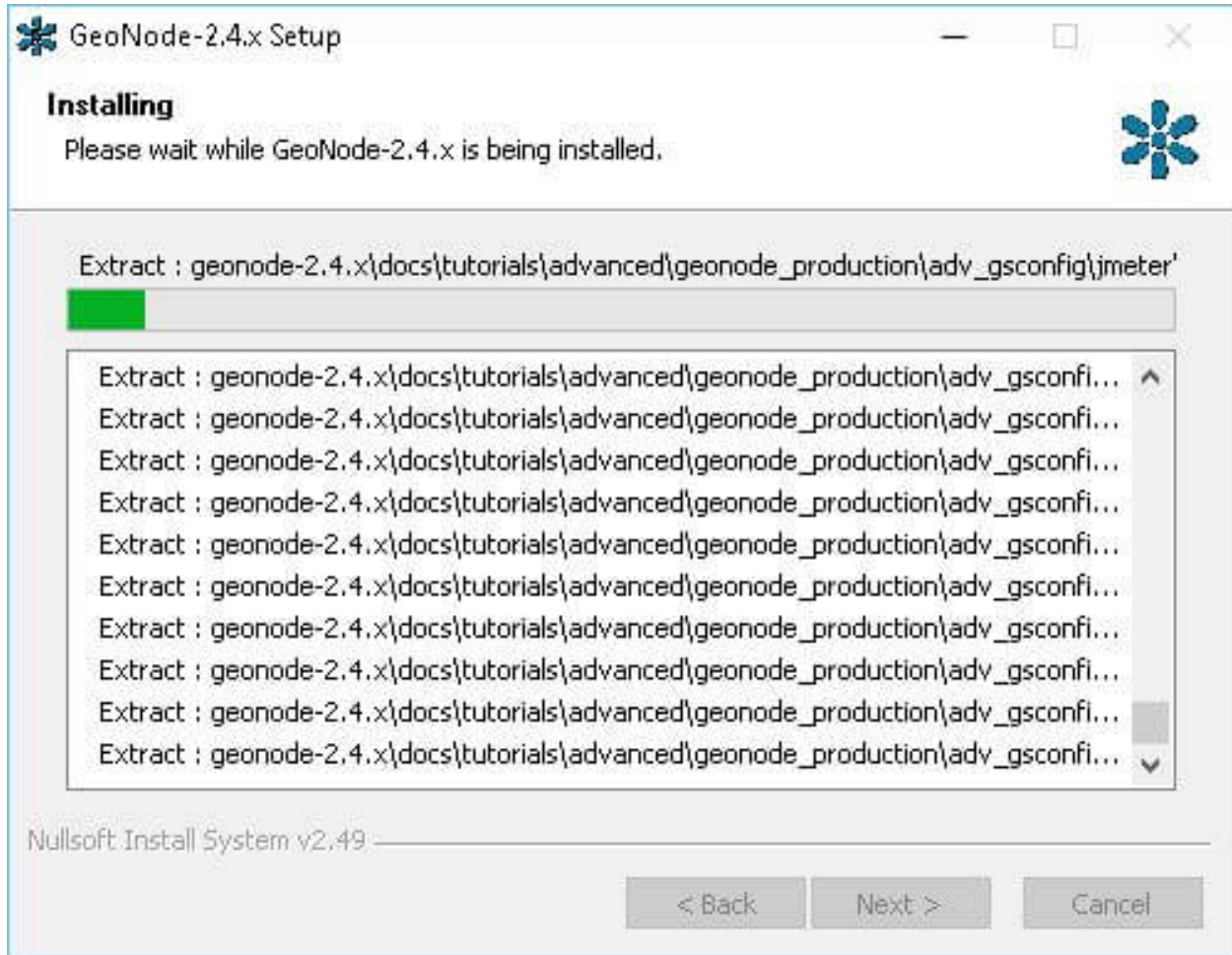
15. Click “Next” and proceed with the installation and automatic configuration of Apache Tomcat 7.0.6535



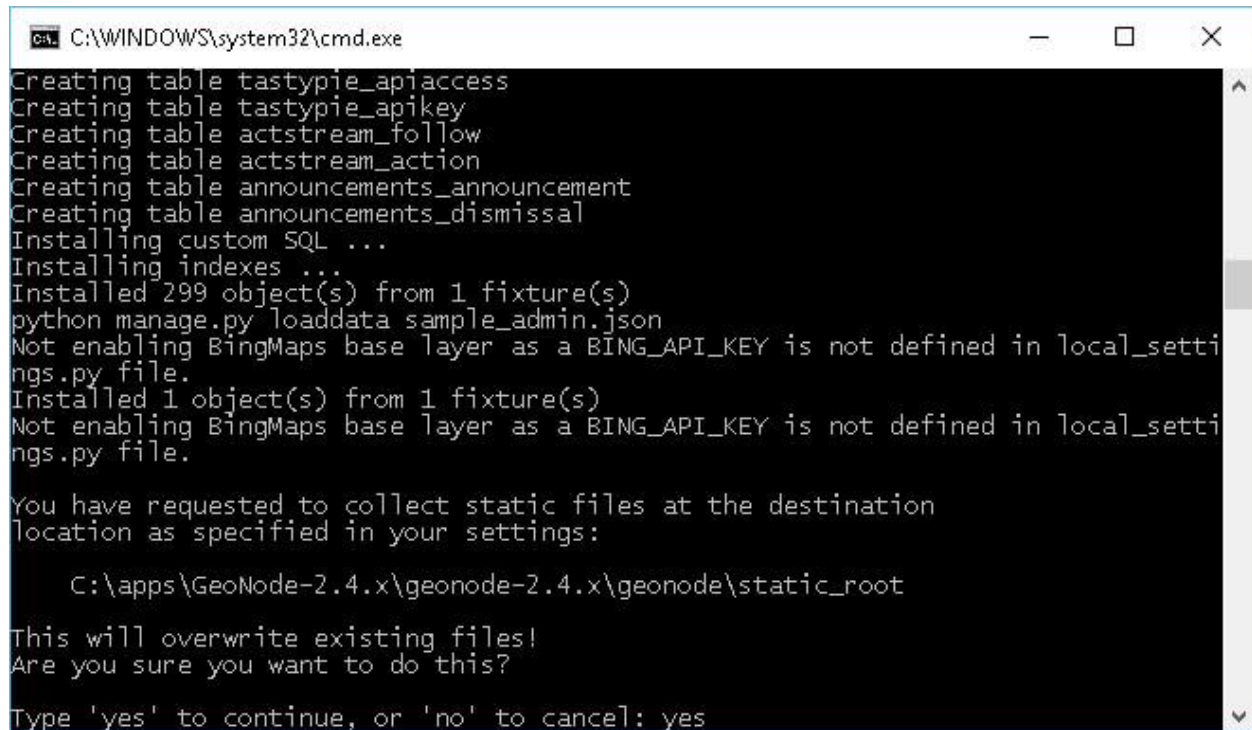
16. Confirm “Yes” in order to complete the GeoNode installation and configuration process.



17. Wait until the process finishes copying the files
18. At a certain point the process configures the database



and asks to copy the GeoNode static resources.



```
C:\WINDOWS\system32\cmd.exe
Creating table tastypie_apiaccess
Creating table tastypie_apikey
Creating table actstream_follow
Creating table actstream_action
Creating table announcements_announcement
Creating table announcements_dismissal
Installing custom SQL ...
Installing indexes ...
Installed 299 object(s) from 1 fixture(s)
python manage.py loaddata sample_admin.json
Not enabling BingMaps base layer as a BING_API_KEY is not defined in local_settings.py file.
Installed 1 object(s) from 1 fixture(s)
Not enabling BingMaps base layer as a BING_API_KEY is not defined in local_settings.py file.

You have requested to collect static files at the destination
location as specified in your settings:

    C:\apps\GeoNode-2.4.x\geonode-2.4.x\geonode\static_root

This will overwrite existing files!
Are you sure you want to do this?
Type 'yes' to continue, or 'no' to cancel: yes
```

Warning: Enter “yes” and click “Enter”. The system will wait the user input!

19. As a final step, the process automatically creates 8 sample layers on GeoNode
20. You need to “Reboot” the system in order to make everything working fine
21. Once the reboot has done, if everything has worked, you should be able to access the GeoNode Home directory on your local computer directly from the Windows App Toolbar

Congratulations! The GeoNode setup has successfully completed.

How your system has been updated after the GeoNode Setup

1. At the end of the installation, there will be **3** new Windows Services
Access the Windows local services from the “Control Panel”
The following services *should* be **up & running** at the end of the process and after a system reboot
2. At the end of the installation process, there will be few new Environment System Variables
Access the Environment variables from the System panel
Access the System Properties and click on “Environment Variables”
The new Environment Variables are:


```

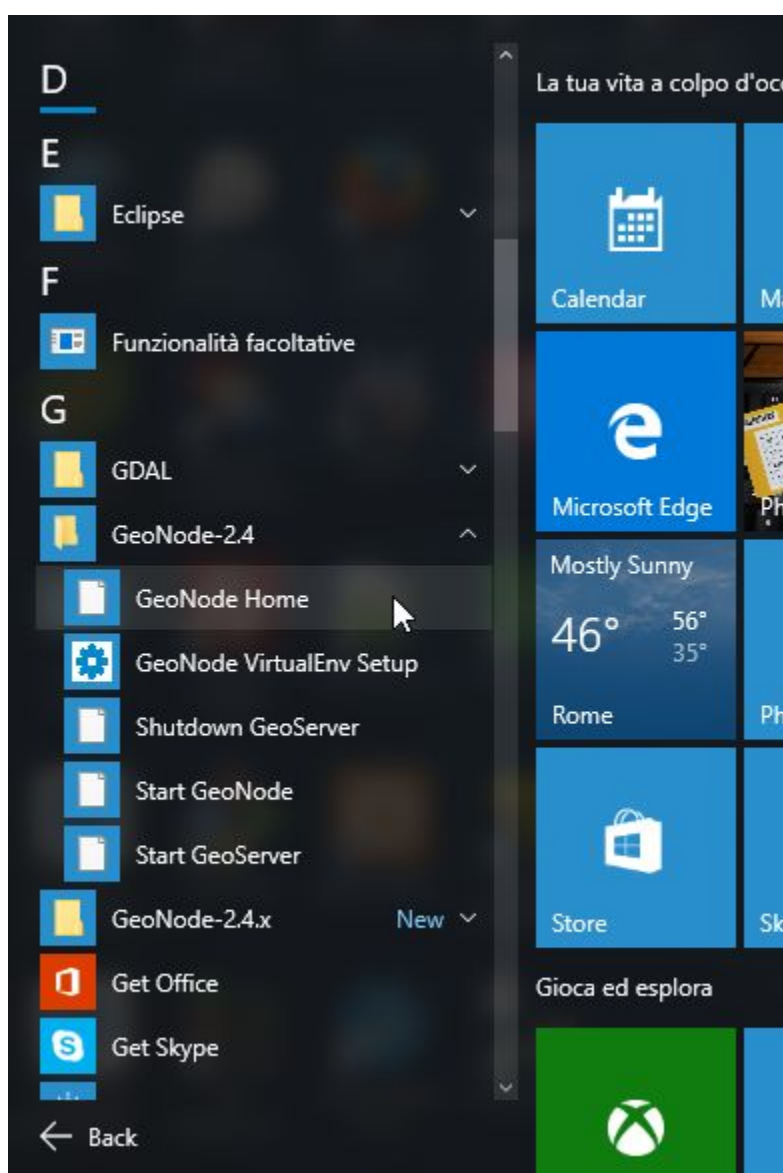
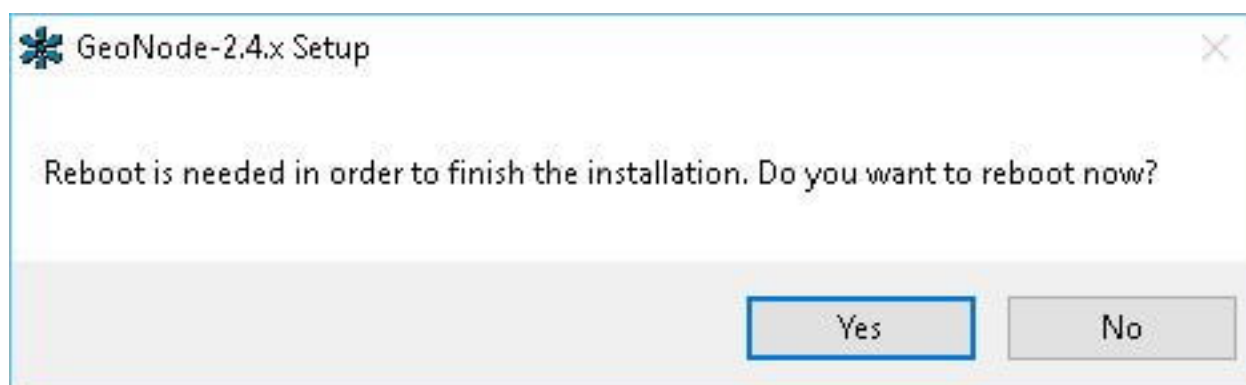
C:\WINDOWS\system32\cmd.exe
c:\geoserver\externals\ext\resources\images\gray\grid\col-move-bottom.gif'
Copying 'C:\apps\GeoNode-2.4.x\geonode-2.4.x\lib\site-packages\geoserver\stati
c\geoserver\externals\ext\resources\images\gray\grid\col-move-top.gif'
Copying 'C:\apps\GeoNode-2.4.x\geonode-2.4.x\lib\site-packages\geoserver\stati
c\geoserver\externals\ext\resources\images\gray\grid\grid3-hd-btn.gif'
Copying 'C:\apps\GeoNode-2.4.x\geonode-2.4.x\lib\site-packages\geoserver\stati
c\geoserver\externals\ext\resources\images\gray\grid\grid3-hrow-over.gif'
Copying 'C:\apps\GeoNode-2.4.x\geonode-2.4.x\lib\site-packages\geoserver\stati
c\geoserver\externals\ext\resources\images\gray\grid\grid3-hrow-over2.gif'
Copying 'C:\apps\GeoNode-2.4.x\geonode-2.4.x\lib\site-packages\geoserver\stati
c\geoserver\externals\ext\resources\images\gray\grid\grid3-hrow.gif'
Copying 'C:\apps\GeoNode-2.4.x\geonode-2.4.x\lib\site-packages\geoserver\stati
c\geoserver\externals\ext\resources\images\gray\grid\grid3-hrow2.gif'
Copying 'C:\apps\GeoNode-2.4.x\geonode-2.4.x\lib\site-packages\geoserver\stati
c\geoserver\externals\ext\resources\images\gray\grid\grid3-special-col-bg.gif'
Copying 'C:\apps\GeoNode-2.4.x\geonode-2.4.x\lib\site-packages\geoserver\stati
c\geoserver\externals\ext\resources\images\gray\grid\grid3-special-col-bg2.gif'
Copying 'C:\apps\GeoNode-2.4.x\geonode-2.4.x\lib\site-packages\geoserver\stati
c\geoserver\externals\ext\resources\images\gray\grid\grid3-special-col-sel-bg.
gif'
Copying 'C:\apps\GeoNode-2.4.x\geonode-2.4.x\lib\site-packages\geoserver\stati
c\geoserver\externals\ext\resources\images\gray\grid\group-collapse.gif'
Copying 'C:\apps\GeoNode-2.4.x\geonode-2.4.x\lib\site-packages\geoserver\stati

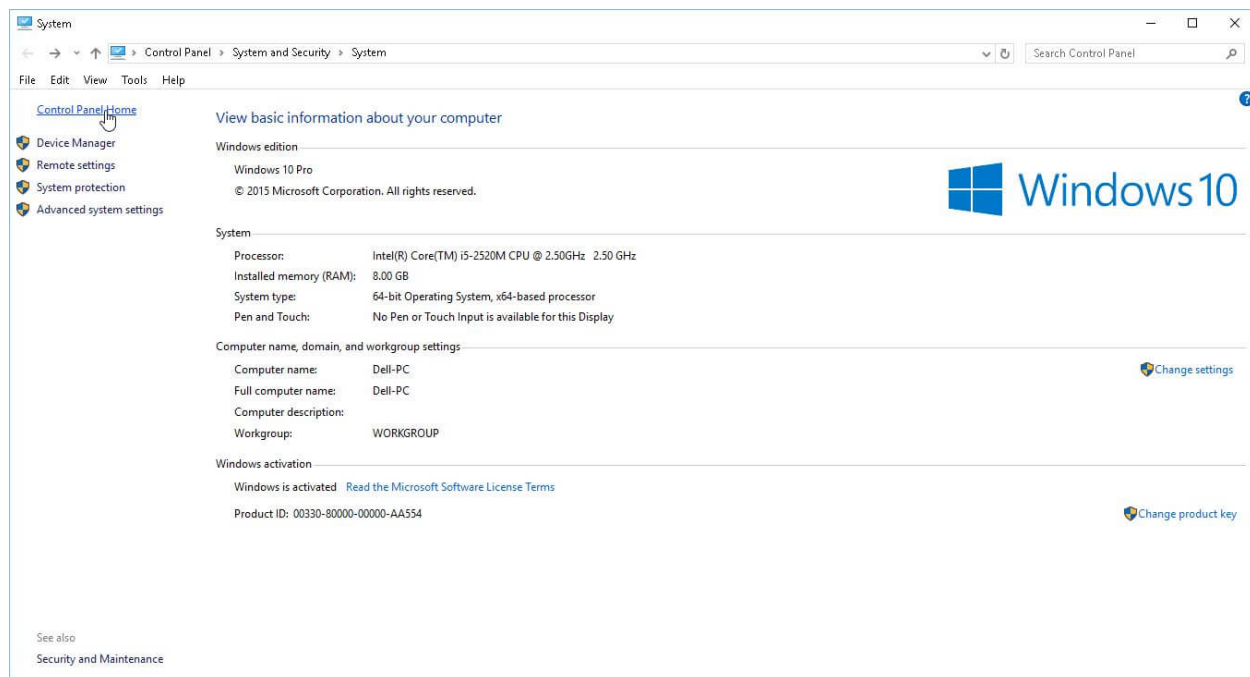
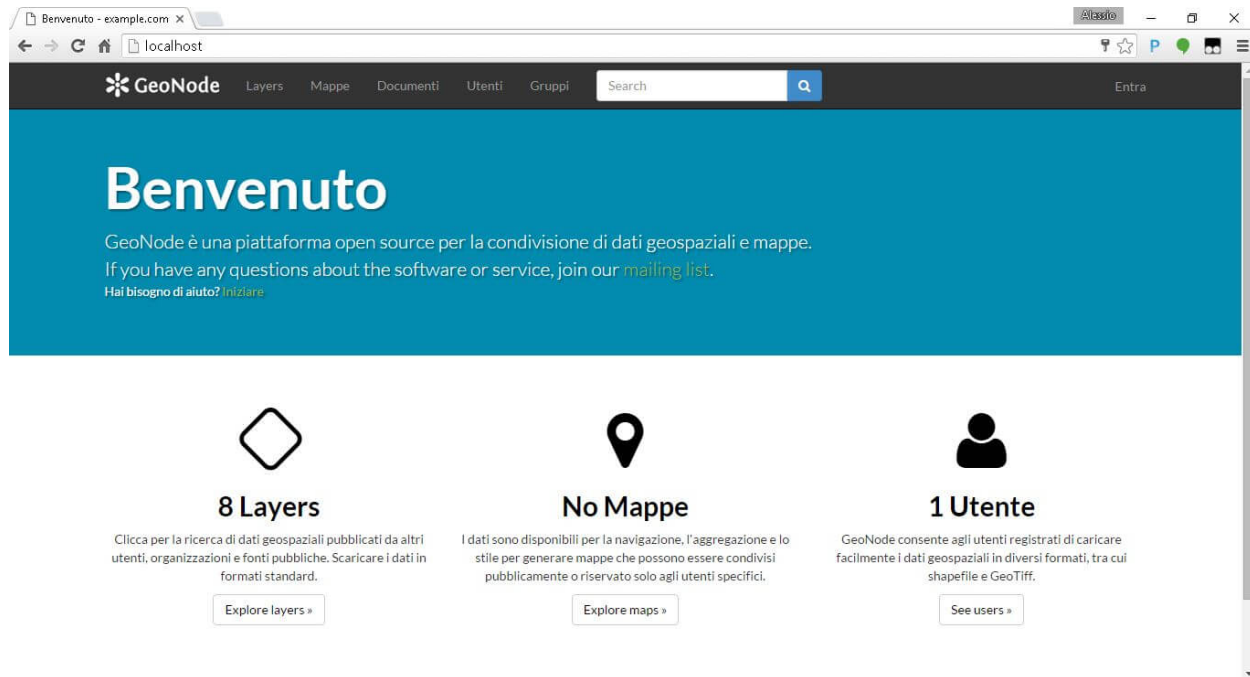
```

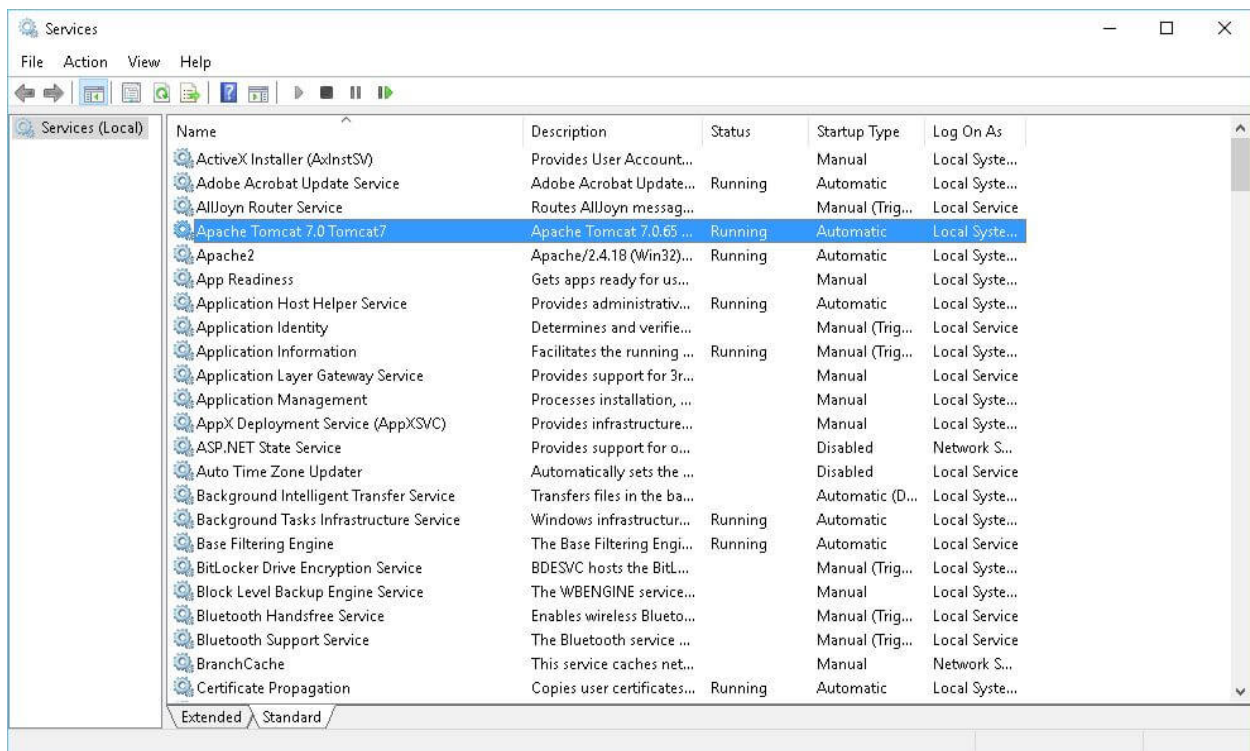
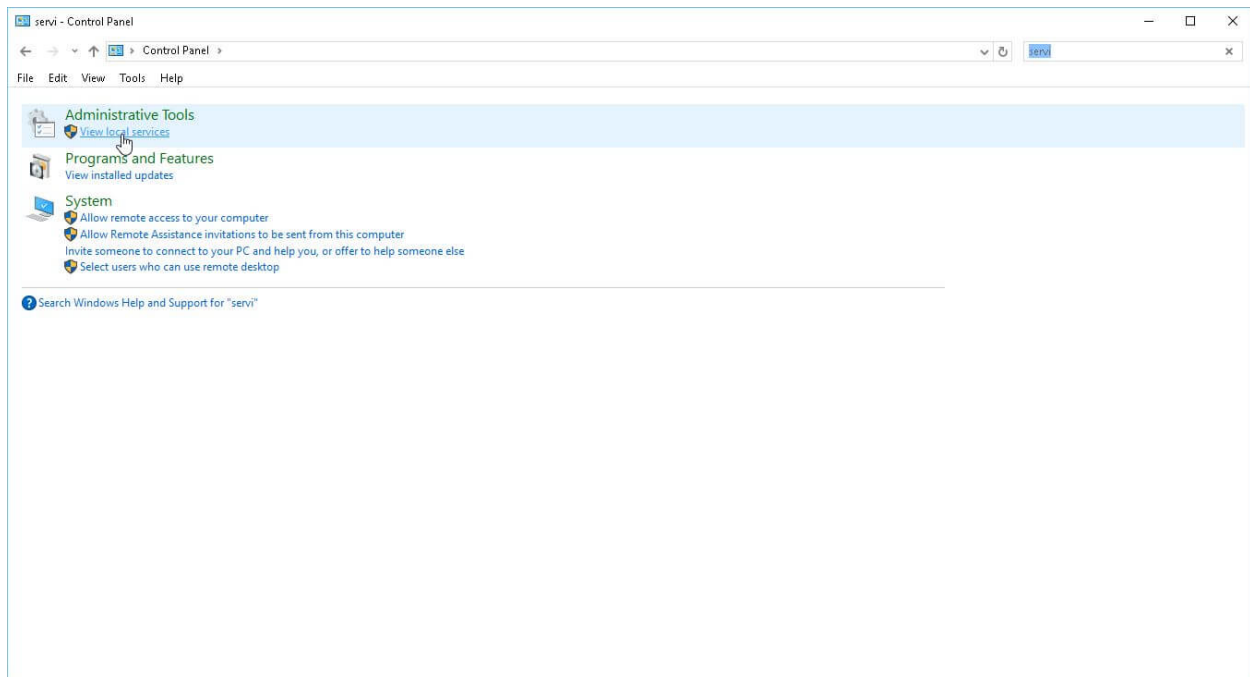
```

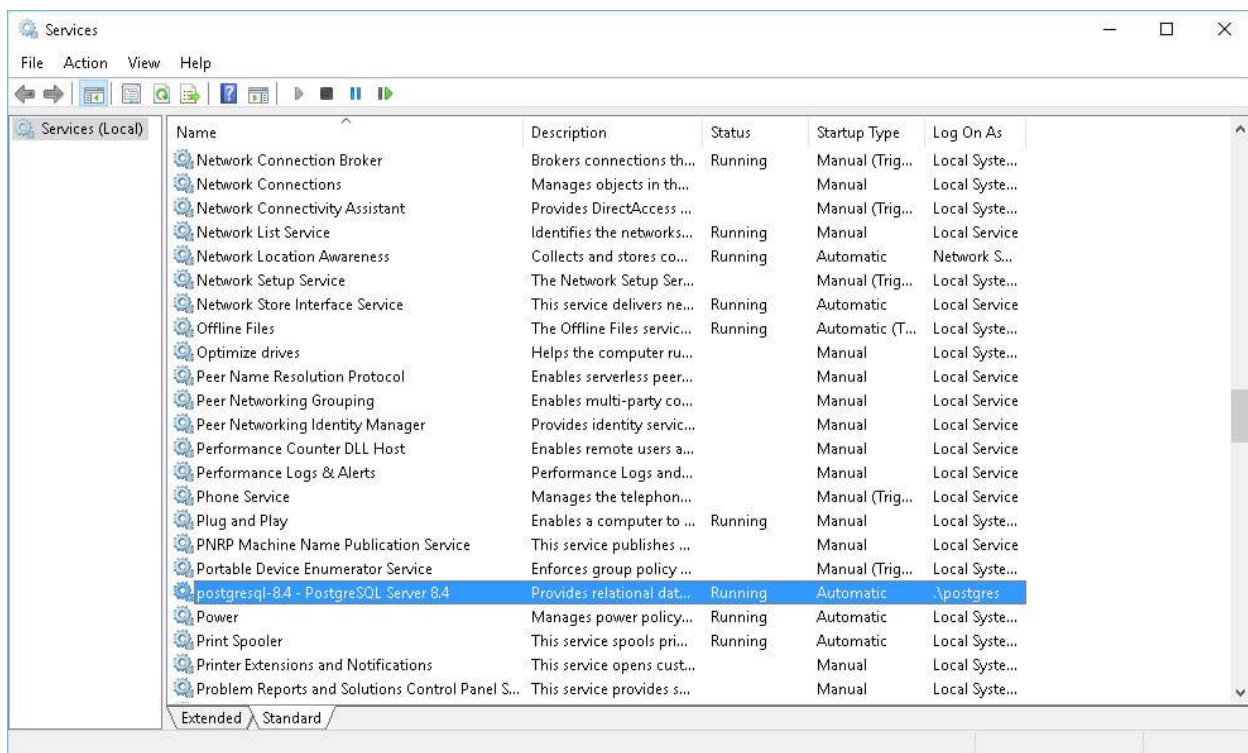
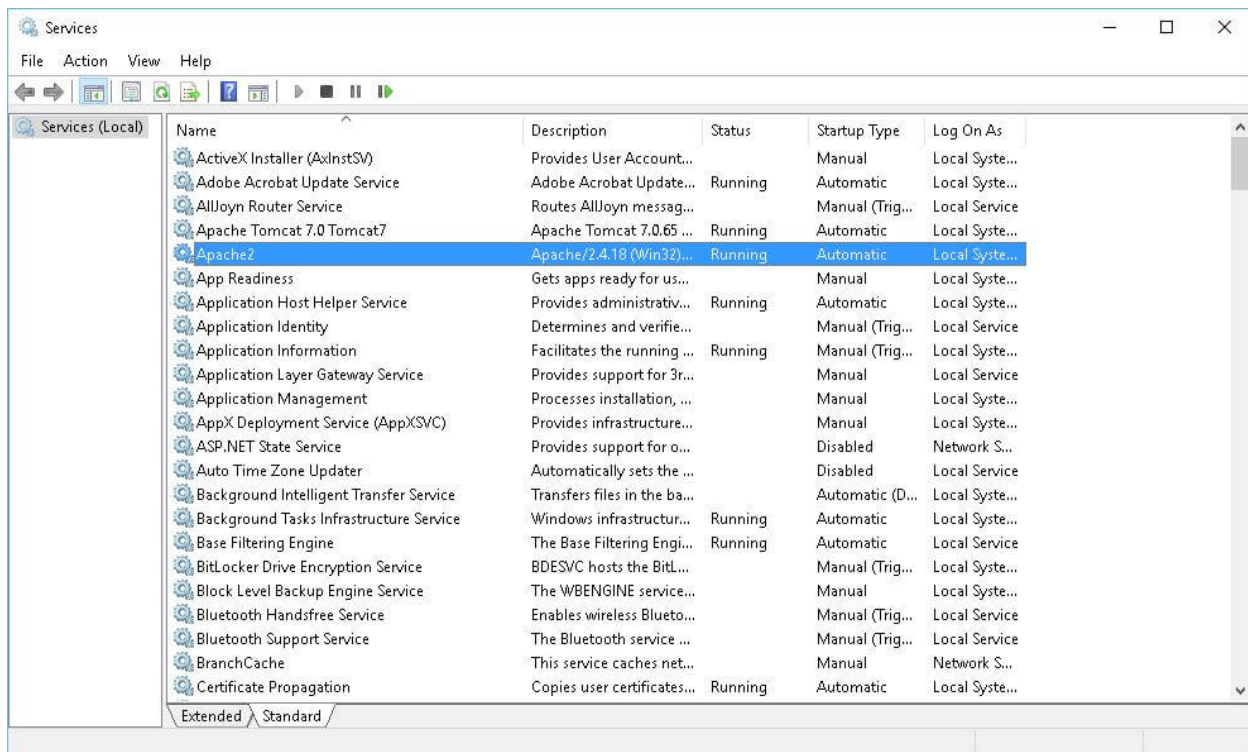
C:\WINDOWS\system32\cmd.exe
Not enabling BingMaps base layer as a BING_API_KEY is not defined in local_setti
ngs.py file.
Verifying that GeoNode is running ...
Found 8 potential layers.
[created] Layer for 'C:\apps\GeoNode-2.4.x\geonode-2.4.x\lib\site-packages\gisda
ta\data\good\vector\san_andres_y_providencia_administrative.shp' (1/8)
[created] Layer for 'C:\apps\GeoNode-2.4.x\geonode-2.4.x\lib\site-packages\gisda
ta\data\good\vector\san_andres_y_providencia_coastline.shp' (2/8)
[created] Layer for 'C:\apps\GeoNode-2.4.x\geonode-2.4.x\lib\site-packages\gisda
ta\data\good\vector\san_andres_y_providencia_highway.shp' (3/8)

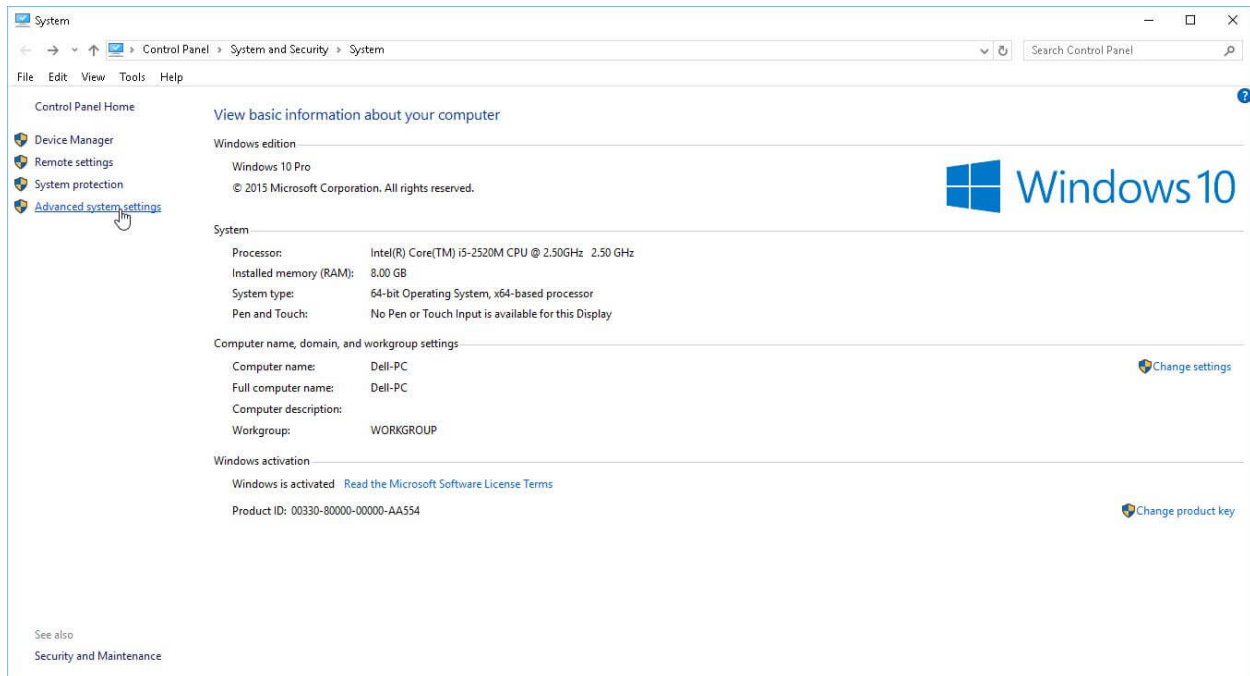
```











- `GDAL_HOME`; pointing to the folder of GDAL 1.11
- `GDAL_LIBRARY_PATH`; pointing to the **gdal111** dll
- `GEOS_LIBRARY_PATH`; pointing to the GEOS library dll
- `JRE_HOME`; pointing to the JRE 1.7 installed at the beginning of the process
- `GEONODE_PATHTEXT`; all the Python paths which will be added to the Windows default PATH

3.2.1.2.11 GeoNode (v2.8) update from older versions

This part of the documentation describes the complete setup process for GeoNode update from older versions.

Guidelines

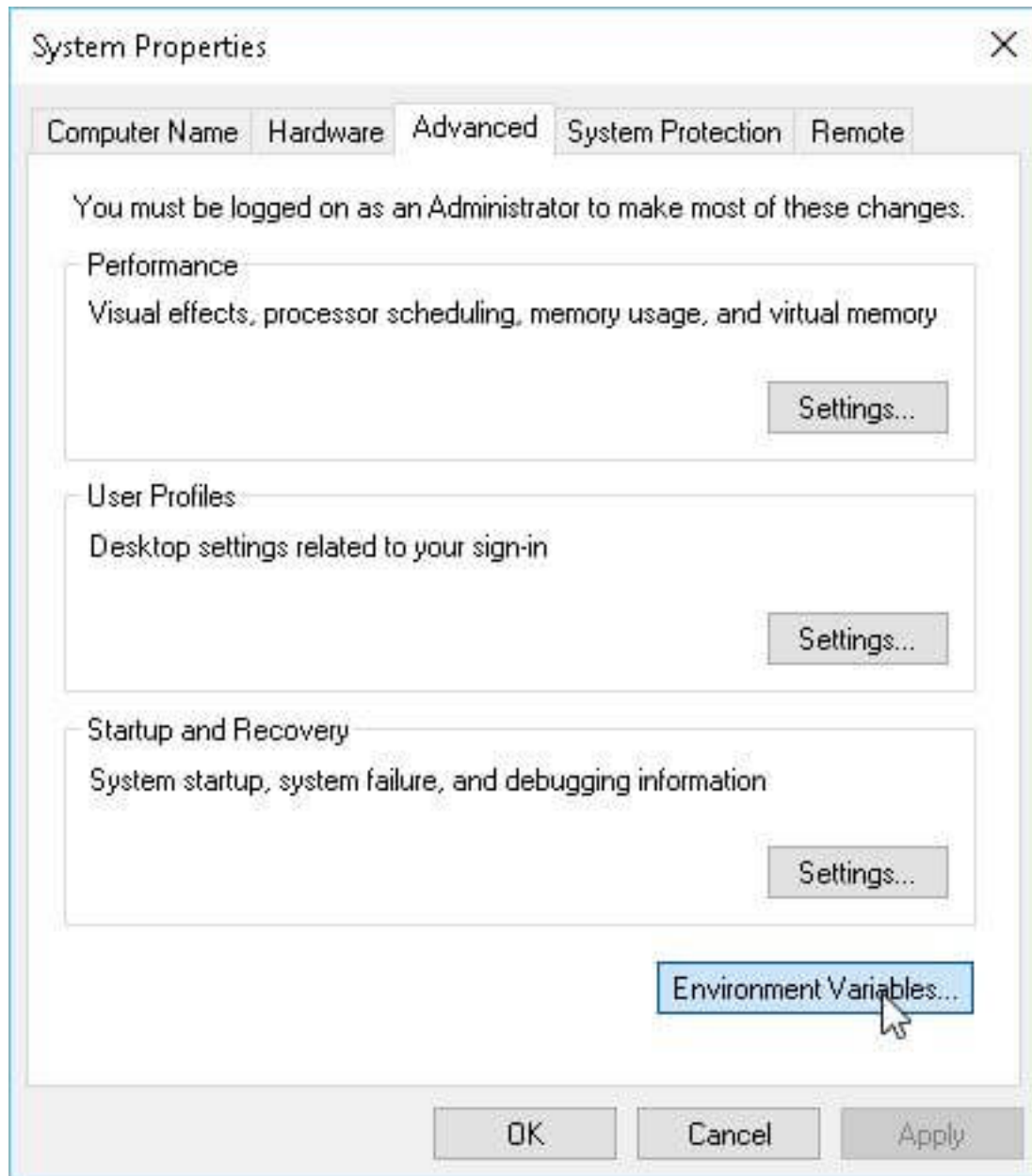
Update from GeoNode 2.6.3 to GeoNode 2.7+

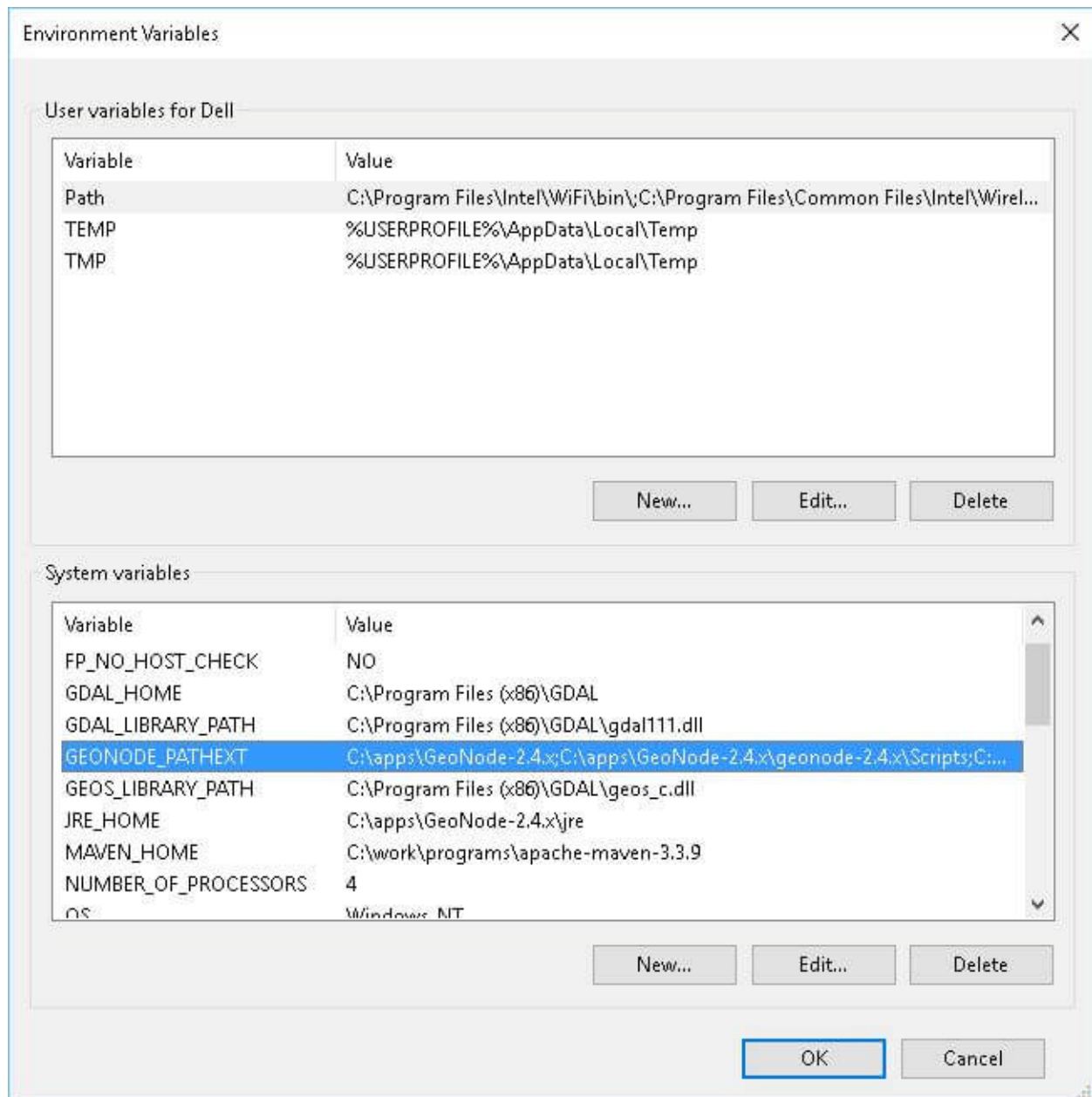
Warning: Before you proceed with the following steps, it is recommended that you perform a full backup of your current environment. At the end of this section there's a paragraph listing the steps required to perform a full / hard backup of the whole stack.

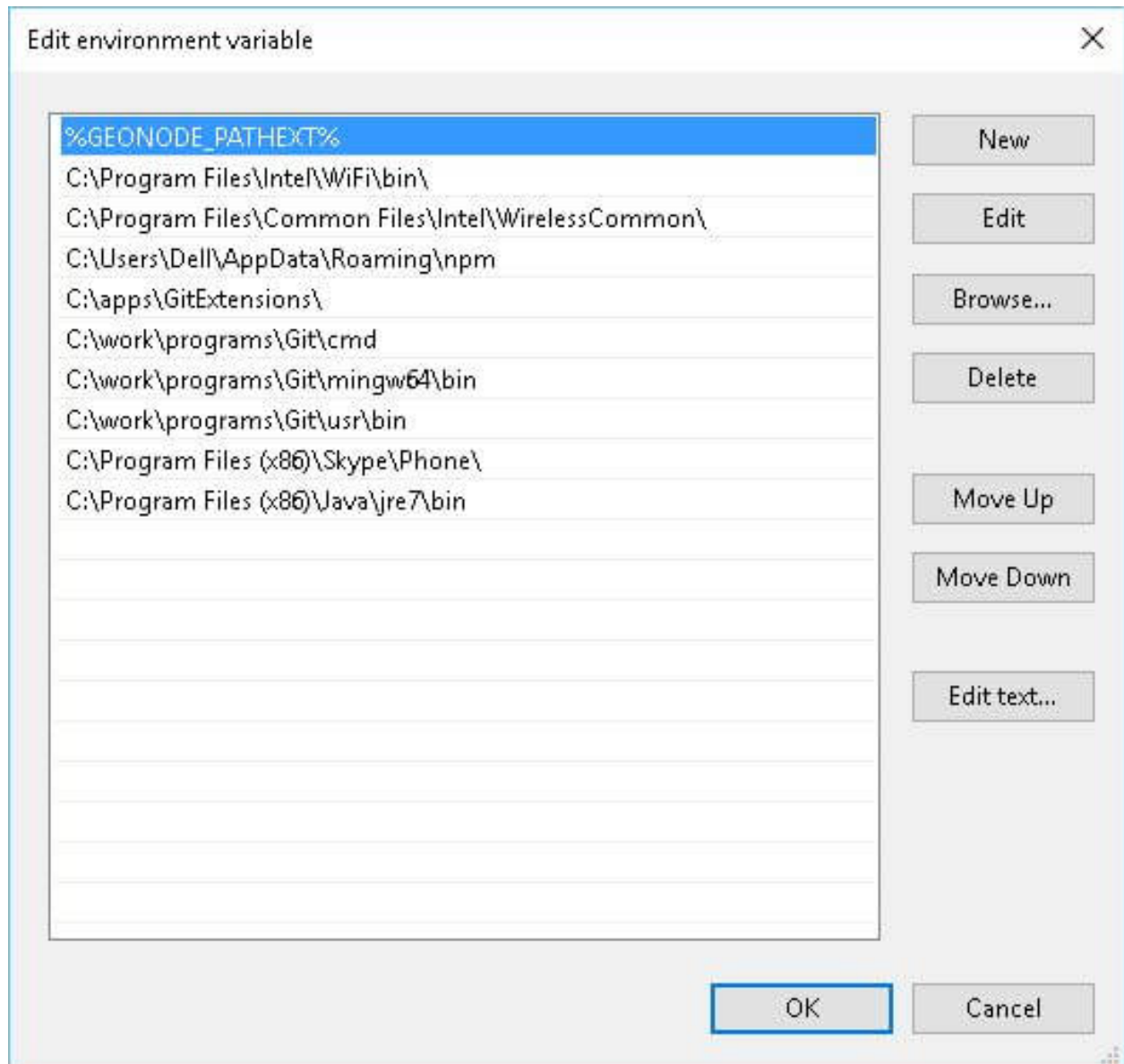
Upgrade Development Environment

Prerequisites

1. You did backup of the old Environment
2. You cloned GeoNode from GitHub (<https://github.com/GeoNode/geonode/tree/2.6.x>)







Steps

- From geonode git branch do

```
# to refresh all git repos and commits
git pull

# get the 2.7.x code: if you want to keep your local copy
# WARNING: you will need to fix conflicts manually
git pull origin 2.7.x

# if you want to switch to the new branch
git checkout 2.7.x
git pull
```

- Update the Python libraries
 - Exit from the current workspace

```
deactivate
```

- Create a new workspace

```
mkvirtualenv geonode-2.7.x
```

- Update the requirements

```
pip install pip --upgrade
pip install -r requirements.txt --upgrade
pip install -e . --upgrade --no-cache

# WARNING: your GDAL version might be different. Use the right one
↳ accordingly to gdal-config --version
pip install pygdal==2.2.1.3
```

Note: Starting from an old environment will require you to manually remove some old deps **before** running any kind of management command.

```
pip uninstall geonode-user-accounts -y
pip uninstall django-pagination -y
pip uninstall pinax-theme-bootstrap-account -y
```

- Update/tweak GeoNode local_settings (for GeoServer)

```
Left file: D:\work\code\python\geonode\geonode-2.7.
↳x\geonode\local_settings.py.geoserver.sample
Right file: D:\work\code\python\geonode\geonode-2.6.
↳x\geonode\local_settings.py.geoserver.sample
21 import os
↳
↳import os
-----
22 from geonode.settings import *
↳
-----
↳-----
```

(continues on next page)

(continued from previous page)

```

23                                     = 22
↳
24 PROJECT_ROOT = os.path.abspath(os.path.dirname(__file__))
↳
25                                     23
↳PROJECT_ROOT = os.path.abspath(os.path.dirname(__file__))
25                                     24
↳
-----
↳-----
26 MEDIA_ROOT = os.getenv('MEDIA_ROOT', os.path.join(PROJECT_
↳ROOT, "uploaded"))
↳-----
↳-----
27                                     =
↳
-----
↳-----
28 STATIC_ROOT = os.getenv('STATIC_ROOT',
↳
29                                     os.path.join(PROJECT_ROOT, "static_
↳root")
30                                     )
↳-----
↳-----
31                                     =
↳
-----
↳-----
32 # SECRET_KEY = '*****'
↳
↳-----
↳-----
33                                     =
↳
↳-----
↳-----
34 SITEURL = "http://localhost:8000/"
↳
↳-----
↳-----
35                                     = 26
↳
↳-----
↳-----
36 ALLOWED_HOSTS = ['localhost', 'geonode.example.com']
↳
↳-----
↳-----
37                                     =
↳
↳-----
↳-----
38 # TIME_ZONE = 'Europe/Paris'
↳
↳-----
↳-----

```

(continues on next page)

(continued from previous page)

```

39                                     =
40 DATABASES = {
41     'default': {
42         'ENGINE': 'django.db.backends.postgresql_psycopg2',
43         'ENGINE': 'django.db.backends.postgresql_psycopg2',
44         'NAME': 'geonode',
45         'NAME': 'geonode',
46         'USER': 'geonode',
47         'USER': 'geonode',
48         'PASSWORD': 'geonode',
49         'PASSWORD': 'geonode',
50         'HOST' : 'localhost',
51         'PORT' : '5432',
52     },
53     # vector datastore for uploads
54     # vector datastore for uploads
55     'datastore': {
56         'datastore' : {
57             # 'ENGINE': 'django.contrib.gis.db.backends.postgis',
58             # 'ENGINE': 'django.contrib.gis.db.backends.postgis',
59             'ENGINE': '', # Empty ENGINE name disables
60             'ENGINE': '', # Empty ENGINE name disables
61         }
62     },
63     'NAME': 'geonode_data',
64     'NAME': 'geonode',
65     'USER' : 'geonode',
66     'USER' : 'geonode',
67     'PASSWORD': 'geonode',
68     'PASSWORD' : 'geonode',
69     'HOST': 'localhost',
70     'HOST' : 'localhost',

```

(continues on next page)

(continued from previous page)

```

57         'PORT': '5432',
↳                                     42  ⌋
↳         'PORT' : '5432',
58     }
↳                                     43  ⌋
↳     }
59 }
↳                                     44  ⌋
↳                                     45
60
↳                                     46
61 GEOSERVER_LOCATION = os.getenv(
↳                                     47
↳ GEOSERVER_LOCATION = os.getenv(
62     'GEOSERVER_LOCATION', 'http://localhost:8080/geoserver/'
↳                                     48  ⌋
↳     'GEOSERVER_LOCATION', 'http://localhost:8080/geoserver/'
63 )
↳                                     49  ⌋
↳                                     50
64
↳                                     51
65 GEOSERVER_PUBLIC_LOCATION = os.getenv(
↳                                     52
↳ GEOSERVER_PUBLIC_LOCATION = os.getenv(
↳ -----
↳ -----
66 #     'GEOSERVER_PUBLIC_LOCATION', '{}geoserver/'.
↳ format(SITEURL)
↳     <>
67     'GEOSERVER_LOCATION', 'http://localhost:8080/geoserver/'
↳                                     50  ⌋
↳     'GEOSERVER_PUBLIC_LOCATION', 'http://localhost:8080/geoserver/'
↳     '
68 )
↳ -----
↳ -----
69
↳                                     =
↳ -----
↳ -----
70 OGC_SERVER_DEFAULT_USER = os.getenv(
↳                                     +-
71     'GEOSERVER_ADMIN_USER', 'admin'
72 )
↳ -----
↳ -----
73
↳                                     =
↳ -----
↳ -----
74 OGC_SERVER_DEFAULT_PASSWORD = os.getenv(
↳                                     +-
75     'GEOSERVER_ADMIN_PASSWORD', 'geoserver'
↳ -----
↳ -----
76 )
↳                                     = 51  ⌋
77
↳                                     52

```

(continues on next page)

(continued from previous page)

```

78 # OGC (WMS/WFS/WCS) Server Settings
↳
53 #
↳OGC (WMS/WFS/WCS) Server Settings
79 OGC_SERVER = {
↳
54
↳OGC_SERVER = {
80     'default': {
↳
55
↳     'default': {
81         'BACKEND': 'geonode.geoserver',
↳
56
↳         'BACKEND': 'geonode.geoserver',
↳-----
↳-----
↳-----
83         'LOGIN_ENDPOINT': 'j_spring_oauth2_geonode_login',
↳
58
↳         'LOGIN_ENDPOINT': 'j_spring_oauth2_geonode_login',
84         'LOGOUT_ENDPOINT': 'j_spring_oauth2_geonode_logout',
↳
59
↳         'LOGOUT_ENDPOINT': 'j_spring_oauth2_geonode_logout',
85         # PUBLIC_LOCATION needs to be kept like this because
↳
60
↳         # PUBLIC_LOCATION needs to be kept like this because in
↳dev mode
86         # the proxy won't work and the integration tests will
↳
61
↳         # the proxy won't work and the integration tests will fail
87         # the entire block has to be overridden in the local_
↳settings
62
↳         # the entire block has to be overridden in the local_
↳settings
88         'PUBLIC_LOCATION': GEOSERVER_PUBLIC_LOCATION,
↳
63
↳         'PUBLIC_LOCATION': GEOSERVER_PUBLIC_LOCATION,
↳-----
↳-----
89         'USER' : OGC_SERVER_DEFAULT_USER,
↳
64
↳         'USER' : 'admin',
90         'PASSWORD' : OGC_SERVER_DEFAULT_PASSWORD,
↳
65
↳         'PASSWORD' : 'geoserver',
↳-----
↳-----
91         'MAPFISH_PRINT_ENABLED' : True,
↳
66
↳         'MAPFISH_PRINT_ENABLED' : True,
92         'PRINT_NG_ENABLED' : True,
↳
67
↳         'PRINT_NG_ENABLED' : True,
93         'GEONODE_SECURITY_ENABLED' : True,
↳
68
↳         'GEONODE_SECURITY_ENABLED' : True,
↳-----
↳-----

```

(continues on next page)

(continued from previous page)

```

94      'GEOFENCE_SECURITY_ENABLED' : True,          +-
↪
-----
↪-----
95      'GEOGIG_ENABLED' : False,                    = 69
↪
↪      'GEOGIG_ENABLED' : False,
96      'WMST_ENABLED' : False,                      70
↪
↪      'WMST_ENABLED' : False,
97      'BACKEND_WRITE_ENABLED': True,               71
↪
↪      'BACKEND_WRITE_ENABLED': True,
98      'WPS_ENABLED': False,                        72
↪
↪      'WPS_ENABLED' : False,
99      'LOG_FILE': '%s/geoserver/data/logs/geoserver.log' %
↪os.path.abspath(os.path.join(PROJECT_ROOT, os.pardir)), 73
↪      'LOG_FILE': '%s/geoserver/data/logs/geoserver.log' % os.
↪path.abspath(os.path.join(PROJECT_ROOT, os.pardir)),
100     # Set to dictionary identifier of database containing
↪spatial data in DATABASES dictionary to enable          74
↪     # Set to dictionary identifier of database containing
↪spatial data in DATABASES dictionary to enable
101     'DATASTORE': '', # 'datastore',                75
↪
↪     'DATASTORE': '', # 'datastore',
-----
↪-----
102     'PG_GEOGIG': False,                            +-
↪
103     'TIMEOUT': 10 # number of seconds to allow for HTTP
↪requests
-----
↪-----
104     }
↪
↪                                     = 76
↪     }
105 }
↪
↪                                     77 }
106
↪                                     78
107 # If you want to enable Mosaics use the following
↪configuration
↪     79 # If you want to enable Mosaics use the following
↪configuration
-----
↪-----
108 UPLOADER = {
↪
↪                                     <> 80
↪#UPLOADER = {
109     # 'BACKEND': 'geonode.rest',
↪
↪                                     81 #
↪#     'BACKEND': 'geonode.rest',
110     'BACKEND': 'geonode.importer',
↪
↪                                     82 #
↪     'BACKEND': 'geonode.importer',

```

(continues on next page)

(continued from previous page)

```

111     'OPTIONS': {
↳                                     83 #
↳     'OPTIONS': {
112         'TIME_ENABLED': True,
↳                                     84 #
↳         'TIME_ENABLED': True,
113         'MOSAIC_ENABLED': False,
↳                                     85 #
↳         'MOSAIC_ENABLED': True,
114         'GEOGIG_ENABLED': False,
↳                                     86 #
↳         'GEOGIG_ENABLED': False,
115     },
↳                                     87 #
↳ }
116     'SUPPORTED_CRS': [
117         'EPSG:4326',
118         'EPSG:3785',
119         'EPSG:3857',
120         'EPSG:900913',
121         'EPSG:32647',
122         'EPSG:32736'
123     ],
124     'SUPPORTED_EXT': [
125         '.shp',
126         '.csv',
127         '.kml',
128         '.kmz',
129         '.json',
130         '.geojson',
131         '.tif',
132         '.tiff',
133         '.geotiff',
134         '.gml',
135         '.xml'
136     ]
↳                                     88 #}
137 }
↳                                     89
-----
↳ -----
138
↳                                     = 90
139 CATALOGUE = {
↳                                     91
↳ CATALOGUE = {
140     'default': {
↳                                     92
↳     'default': {
141         # The underlying CSW implementation
↳                                     93
↳         # The underlying CSW implementation
142         # default is pycsw in local mode (tied directly to
↳ GeoNode Django DB) 94
↳         # default is pycsw in local mode (tied directly to
↳ GeoNode Django DB)
143         'ENGINE': 'geonode.catalogue.backends.pycsw_local',
↳                                     (continues on next page)
↳
↳     'ENGINE': 'geonode.catalogue.backends.pycsw_local',

```


(continued from previous page)

```

-----
->-----
146      # GeoNetwork opensource
->                                     = 98
->      # GeoNetwork opensource
147      # 'ENGINE': 'geonode.catalogue.backends.geonetwork',
->                                     99
->      # 'ENGINE': 'geonode.catalogue.backends.geonetwork',
148      # deegree and others
->                                     100
->      # deegree and others
149      # 'ENGINE': 'geonode.catalogue.backends.generic',
->                                     101
->      # 'ENGINE': 'geonode.catalogue.backends.generic',
150
->                                     102
151      # The FULLY QUALIFIED base url to the CSW instance
->for this GeoNode
->103      # The FULLY QUALIFIED base url to the CSW instance
->for this GeoNode
-----
->-----
152      'URL': '%scatalogue/csw' % SITEURL,
->                                     <> 104
->      'URL': '%scatalogue/csw' % SITEURL,
-----
->-----
153      # 'URL': 'http://localhost:8080/geonetwork/srv/en/csw
->      ',
->                                     = 105
->      # 'URL': 'http://localhost:8080/geonetwork/srv/en/csw',
154      # 'URL': 'http://localhost:8080/deegree-csw-demo-3.0.
->4/services',
->                                     106
->      # 'URL': 'http://localhost:8080/deegree-csw-demo-3.0.4/
->services',
155
->                                     107
156      # login credentials (for GeoNetwork)
->                                     108
->      # login credentials (for GeoNetwork)
157      'USER': 'admin',
->                                     109
->      'USER': 'admin',
158      'PASSWORD': 'admin',
->                                     110
->      'PASSWORD': 'admin',
159      }
->                                     111
->      }
160 }
->                                     112 }
161
->                                     113
-----
->-----
162 # pycsw settings

```

(continues on next page)

(continued from previous page)

```

163 PYCSW = {
164     # pycsw configuration
165     'CONFIGURATION': {
166         # uncomment / adjust to override server config system
167         ↪defaults
168         # 'server': {
169         #     'maxrecords': '10',
170         #     'pretty_print': 'true',
171         #     'federatedcatalogues': 'http://catalog.data.gov/
172         ↪csw'
173         # },
174         'metadata:main': {
175             'identification_title': 'GeoNode Catalogue',
176             'identification_abstract': 'GeoNode is an open
177             ↪source platform' \
178             ' that facilitates the creation, sharing, and
179             ↪collaborative use' \
180             ' of geospatial data',
181             'identification_keywords': 'sdi, catalogue,
182             ↪discovery, metadata,' \
183             ' GeoNode',
184             'identification_keywords_type': 'theme',
185             'identification_fees': 'None',
186             'identification_accessconstraints': 'None',
187             'provider_name': 'Organization Name',
188             'provider_url': SITEURL,
189             'contact_name': 'Lastname, Firstname',
190             'contact_position': 'Position Title',
191             'contact_address': 'Mailing Address',
192             'contact_city': 'City',
193             'contact_stateorprovince': 'Administrative Area',
194             'contact_postalcode': 'Zip or Postal Code',
195             'contact_country': 'Country',
196             'contact_phone': '+xx-xxx-xxx-xxxx',
197             'contact_fax': '+xx-xxx-xxx-xxxx',
198             'contact_email': 'Email Address',
199             'contact_url': 'Contact URL',
200             'contact_hours': 'Hours of Service',
201             'contact_instructions': 'During hours of service.
202             ↪Off on ' \
203             'weekends.',
204             'contact_role': 'pointOfContact',
205             },
206             'metadata:inspire': {
207                 'enabled': 'true',
208                 'languages_supported': 'eng,gre',
209                 'default_language': 'eng',
210                 'date': 'YYYY-MM-DD',
211                 'gemet_keywords': 'Utility and governmental
212             ↪services',
213             'conformity_service': 'notEvaluated',
214             'contact_name': 'Organization Name',
215             'contact_email': 'Email Address',
216             'temp_extent': 'YYYY-MM-DD/YYYY-MM-DD',
217             }
218         }
219     }
220 }

```

(continues on next page)

(continued from previous page)

```

-----
↪-----
213                                     =                               ↪
↪-----
214 # GeoNode javascript client configuration                               ↪
↪                                     +-
-----
↪-----
215                                     =                               ↪
↪-----
216 # default map projection                                               ↪
↪                                     +-
217 # Note: If set to EPSG:4326, then only EPSG:4326 basemaps_
↪will work.
218 DEFAULT_MAP_CRSS = "EPSG:3857"
-----
↪-----
219                                     =                               ↪
↪-----
220 # Where should newly created maps be focused?                         ↪
↪                                     +-
221 DEFAULT_MAP_CENTER = (0, 0)
-----
↪-----
222                                     =                               ↪
↪-----
223 # How tightly zoomed should newly created maps be?                   ↪
↪                                     +-
224 # 0 = entire world;
225 # maximum zoom is between 12 and 15 (for Google Maps,
↪coverage varies by area)
226 DEFAULT_MAP_ZOOM = 0
-----
↪-----
227                                     =                               ↪
↪-----
228 # Default preview library                                              114 # ↪
↪Default preview library
-----
↪-----
229 LAYER_PREVIEW_LIBRARY = 'geoext'                                     <> 115 ↪
↪#LAYER_PREVIEW_LIBRARY = 'geoext'
230 #LAYER_PREVIEW_LIBRARY = 'leaflet'
231 #LEAFLET_CONFIG = {
232 #     'TILES': [
233 #         # Find tiles at:
234 #         # http://leaflet-extras.github.io/leaflet-providers/
↪preview/

```

(continues on next page)

(continued from previous page)

```

235 #
236 #     # Map Quest
237 #     ('Map Quest',
238 #      'http://otile4.mqcdn.com/tiles/1.0.0/osm/{z}/{x}/{y}
↳.png',
239 #      'Tiles Courtesy of <a href="http://www.mapquest.com/
↳">MapQuest</a> ',
240 #      '&mdash; Map data &copy; ',
241 #      '<a href="http://www.openstreetmap.org/copyright">
↳OpenStreetMap</a>'),
242 #     # Stamen toner lite.
243 #     # ('Watercolor',
244 #      # 'http://{s}.tile.stamen.com/watercolor/{z}/{x}/{y}
↳.png',
245 #      # 'Map tiles by <a href="http://stamen.com">Stamen_
↳Design</a>, \
246 #      # <a href="http://creativecommons.org/licenses/by/3.
↳0">CC BY 3.0</a> &mdash; Map data &copy; \
247 #      # <a href="http://openstreetmap.org">OpenStreetMap</
↳a> contributors, \
248 #      # <a href="http://creativecommons.org/licenses/by-
↳sa/2.0/">CC-BY-SA</a>'),
249 #     # ('Toner Lite',
250 #      # 'http://{s}.tile.stamen.com/toner-lite/{z}/{x}/{y}
↳.png',
251 #      # 'Map tiles by <a href="http://stamen.com">Stamen_
↳Design</a>, \
252 #      # <a href="http://creativecommons.org/licenses/by/3.
↳0">CC BY 3.0</a> &mdash; Map data &copy; \
253 #      # <a href="http://openstreetmap.org">OpenStreetMap</
↳a> contributors, \
254 #      # <a href="http://creativecommons.org/licenses/by-
↳sa/2.0/">CC-BY-SA</a>'),
255 #     ],
256 #     'PLUGINS': {
257 #         'esri-leaflet': {
258 #             'js': 'lib/js/esri-leaflet.js',
259 #             'auto-include': True,
260 #         },
261 #         'leaflet-fullscreen': {
262 #             'css': 'lib/css/leaflet.fullscreen.css',
263 #             'js': 'lib/js/Leaflet.fullscreen.min.js',
264 #             'auto-include': True,
265 #         },
266 #     },
267 #     'SRID': 3857,
268 #     'RESET_VIEW': False
269 # }

↳-----
270
↳-----
271 ALT_OSM_BASEMAPS = os.environ.get('ALT_OSM_BASEMAPS', False)
↳-----
272 CARTODB_BASEMAPS = os.environ.get('CARTODB_BASEMAPS', False)

```

(continues on next page)

(continued from previous page)

```

273 STAMEN_BASEMAPS = os.environ.get('STAMEN_BASEMAPS', False)
274 THUNDERFOREST_BASEMAPS = os.environ.get('THUNDERFOREST_
↳BASEMAPS', False)
275 MAPBOX_ACCESS_TOKEN = os.environ.get('MAPBOX_ACCESS_TOKEN',
↳None)
276 BING_API_KEY = os.environ.get('BING_API_KEY', None)

-----
↳-----
277                                     =
↳-----
↳-----
278 MAP_BASELAYERS = [{
↳                                     +-
279     "source": {"ptype": "gxp_olsource"},
280     "type": "OpenLayers.Layer",
281     "args": ["No background"],
282     "name": "background",
283     "visibility": False,
284     "fixed": True,
285     "group": "background"
286 },
287 # {
288 #     "source": {"ptype": "gxp_olsource"},
289 #     "type": "OpenLayers.Layer.XYZ",
290 #     "title": "TEST TILE",
291 #     "args": ["TEST_TILE", "http://test_tiles/tiles/${z}/${x}
↳/${y}.png"],
292 #     "name": "background",
293 #     "attribution": "&copy; TEST TILE",
294 #     "visibility": False,
295 #     "fixed": True,
296 #     "group": "background"
297 # },
298 {
299     "source": {"ptype": "gxp_osmsource"},
300     "type": "OpenLayers.Layer.OSM",
301     "name": "mapnik",
302     "visibility": True,
303     "fixed": True,
304     "group": "background"
305 }]

-----
↳-----
306                                     =
↳-----
↳-----
307 if 'geonode.geoserver' in INSTALLED_APPS:
↳                                     +-
308     LOCAL_GEOSERVER = {
309         "source": {
310             "ptype": "gxp_wmcs",
311             "url": OGC_SERVER['default']['PUBLIC_LOCATION'] +
↳"wms",
312             "restUrl": "/gs/rest"
313         }

```

(continues on next page)

(continued from previous page)

```

314     }
315     baselayers = MAP_BASELAYERS
316     MAP_BASELAYERS = [LOCAL_GEOSERVER]
317     MAP_BASELAYERS.extend(baselayers)
-----
↪-----
318                                     =
↪-----
-----
↪-----
319 # Use kombu broker by default                                     +-
↪-----
320 # REDIS_URL = 'redis://localhost:6379/1'
321 # BROKER_URL = REDIS_URL
322 # CELERY_RESULT_BACKEND = REDIS_URL
323 CELERYD_HIJACK_ROOT_LOGGER = True
324 CELERYD_CONCURENCY = 1
325 # Set this to False to run real async tasks
326 CELERY_ALWAYS_EAGER = True
327 CELERYD_LOG_FILE = None
328 CELERY_REDIRECT_STDOUTS = True
329 CELERYD_LOG_LEVEL = 1
-----
↪-----
330                                     =
↪-----
-----
↪-----
331 # Haystack Search Backend Configuration. To enable,               +-
↪-----
332 # first install the following:
333 # - pip install django-haystack
334 # - pip install elasticsearch==2.4.0
335 # - pip install woosh
336 # - pip install pyelasticsearch
337 # Set HAYSTACK_SEARCH to True
338 # Run "python manage.py rebuild_index"
339 # HAYSTACK_SEARCH = False
340 # Avoid permissions prefiltering
341 SKIP_PERMS_FILTER = False
342 # Update facet counts from Haystack
343 HAYSTACK_FACET_COUNTS = True
344 HAYSTACK_CONNECTIONS = {
345     'default': {
346         'ENGINE': 'haystack.backends.elasticsearch2_backend.
↪Elasticsearch2SearchEngine',
347         'URL': 'http://127.0.0.1:9200/',
348         'INDEX_NAME': 'haystack',
349     },
350     'db': {
351         'ENGINE': 'haystack.backends.simple_backend.
↪SimpleEngine',
352         'EXCLUDED_INDEXES': ['thirdpartyapp.search_indexes.
↪BarIndex'],
353     }
354 }
355 HAYSTACK_SIGNAL_PROCESSOR = 'haystack.signals.
↪RealtimeSignalProcessor'

```

(continues on next page)

(continued from previous page)

```

356 # HAYSTACK_SEARCH_RESULTS_PER_PAGE = 20
-----
357                                     =
-----
358 LOGGING = {
359     'version': 1,
360     'disable_existing_loggers': True,
361     'formatters': {
362         'verbose': {
363             'format': '%(levelname)s %(asctime)s %(module)s
364             %(process)d %(thread)d %(message)s'
365         },
366         'simple': {
367             'format': '%(message)s',
368         },
369     },
370     'filters': {
371         'require_debug_false': {
372             '()': 'django.utils.log.RequireDebugFalse'
373         }
374     },
375     'handlers': {
376         'null': {
377             'level': 'ERROR',
378             'class': 'django.utils.log.NullHandler',
379         },
380         'console': {
381             'level': 'DEBUG',
382             'class': 'logging.StreamHandler',
383             'formatter': 'simple'
384         },
385         'mail_admins': {
386             'level': 'ERROR', 'filters': ['require_debug_false']
387             'class': 'django.utils.log.AdminEmailHandler',
388         }
389     },
390     "loggers": {
391         "django": {
392             "handlers": ["console"], "level": "ERROR", },
393         "geonode": {
394             "handlers": ["console"], "level": "DEBUG", },
395         "gsconfig.catalog": {
396             "handlers": ["console"], "level": "DEBUG", },
397         "owslib": {
398             "handlers": ["console"], "level": "DEBUG", },
399         "pycsw": {
400             "handlers": ["console"], "level": "ERROR", },
401     },
402 }
-----

```

(continues on next page)

(continued from previous page)

```

403
↪                                     =
-----
↪ -----
404 CORS_ORIGIN_ALLOW_ALL = True
↪                                     +-
-----
↪ -----
405
↪                                     =
-----
↪ -----
406 GEOIP_PATH = "/usr/local/share/GeoIP"
↪                                     +-
-----
↪ -----
407
↪                                     =
-----
↪ -----
408 MONITORING_ENABLED = True
↪                                     +-
409 # add following lines to your local settings to enable
↪ monitoring
410 if MONITORING_ENABLED:
411     INSTALLED_APPS += ('geonode.contrib.monitoring',)
412     MIDDLEWARE_CLASSES += ('geonode.contrib.monitoring.
↪ middleware.MonitoringMiddleware',)
413     MONITORING_CONFIG = None
414     MONITORING_SERVICE_NAME = 'local-geonode'
-----
↪ -----

```

- Update the DB

```
DJANGO_SETTINGS_MODULE=geonode.local_settings paver sync
```

- Download the latest GeoServer WAR (<https://build.geo-solutions.it/geonode/geoserver/latest/geoserver-2.12.2.war>)

Index of /geoserver/latest

[ICO]	Name	Last modified	Size	Description
[PARENTDIR]	Parent Directory	-	-	-
[]	data-2.9.x-oauth2.zip	2017-06-09 09:34	120K	
[]	data-2.9.x.zip	2017-09-28 15:23	101K	
[]	data-2.10.x.zip	2017-10-02 10:18	118K	
[]	data-2.12.x.zip	2018-03-21 15:03	154K	
[]	data.zip	2017-06-09 09:34	101K	
[]	geonode-geoserver-ext-2.7.4-geoserver-plugin.zip	2017-06-09 09:34	166K	
[]	geonode-geoserver-ext-2.9.x-geoserver-plugin.zip	2017-09-28 15:23	164K	
[]	geonode-geoserver-ext-2.9.x-oauth2-geoserver-plugin.zip	2017-06-09 09:34	165K	
[]	geonode-geoserver-ext-2.10.x-geoserver-plugin.zip	2017-10-02 10:18	165K	
[]	geonode-geoserver-ext-2.12.x-geoserver-plugin.zip	2018-03-21 15:03	165K	
[]	geoserver-2.9.x-oauth2.war	2017-06-09 09:34	133M	
[]	geoserver-2.9.x.war	2017-09-28 15:23	123M	
[]	geoserver-2.10.x.war	2017-10-02 10:18	135M	
[]	geoserver-2.12.x.war	2018-03-21 15:03	148M	
[]	geoserver.war	2017-06-09 09:34	103M	

Apache/2.4.7 (Ubuntu) Server at build.geonode.org Port 80

- Stop GeoServer
- Extract the WAR content and/or substitute the old one

```
# move the old one
mv geoserver/ geoserver_old/

# create an empty folder for the new one
mkdir geoserver
cd geoserver

# unzip the new GeoServer to the new folder
unzip /home/geosolutions/Downloads/geoserver-2.12.2.war
```

- Delete the notifier configuration by deleting the content of the `GEOSERVER_DATA_DIR/notifier` folder
- Delete the printing configuration by deleting the content of the `GEOSERVER_DATA_DIR/printing` folder
- Insert the content of the 2.12.2 data dir (<https://build.geo-solutions.it/geonode/geoserver/latest/data-2.12.2.zip>), specifically:
 - The content of `data/notifier`, into the `GEOSERVER_DATA_DIR/notifier` folder.
 - The content of `data/monitoring`, into the `GEOSERVER_DATA_DIR/monitoring` folder.
 - The content of `data/styles`, into the `GEOSERVER_DATA_DIR/styles` folder.
 - The content of `data/user_projections`, into the `GEOSERVER_DATA_DIR/user_projections` folder.
- Update the `GEOSERVER_DATA_DIR/geofence/geofence-server.properties` as follows

```
Left file: D:\tmp\data-2.12.2\data\geofence\geofence-server.properties
Right file: D:\tmp\data-2.9.x-oauth2\data\geofence\geofence-server.
properties
13 useRolesToFilter=false
   = 13 useRolesToFilter=false
14 acceptedRoles=
   14 acceptedRoles=
15
   15
16
   16
17 ### Cache configuration
   17 ### Cache configuration
18
   18
-----
19 cacheSize=500000
   <> 19 cacheSize=50000
20 cacheRefresh=6000000
   20 cacheRefresh=600000
21 cacheExpire=6000000
   21 cacheExpire=600000
-----
```

(continues on next page)

(continued from previous page)

```

22
↪      =
-----
23 gwc.context.suffix=gwc
↪      +-
24 org.geoserver.rest.DefaultUserGroupServiceName=geonode REST role_
↪service
-----

```

- Create/modify GEOSERVER_DATA_DIR/gwc/geowebcache-diskquota.xml as follows

```

File: D:\tmp\data-2.12.2\data\gwc\geowebcache-diskquota.xml
1  <gwcQuotaConfiguration>                                     +-
2    <enabled>false</enabled>
3    <cacheCleanUpFrequency>10</cacheCleanUpFrequency>
4    <cacheCleanUpUnits>SECONDS</cacheCleanUpUnits>
5    <maxConcurrentCleanUps>2</maxConcurrentCleanUps>
6    <globalExpirationPolicyName>LRU</globalExpirationPolicyName>
7    <globalQuota>
8      <value>500</value>
9      <units>MiB</units>
10   </globalQuota>
11   <quotaStore>H2</quotaStore>
12 </gwcQuotaConfiguration>
-----

```

- Create/modify GEOSERVER_DATA_DIR/logs/gwc-gs.xml as follows

```

Left file: D:\tmp\data-2.12.2\data\gwc-gs.xml
Right file: D:\tmp\data-2.9.x-oauth2\data\gwc-gs.xml
2    <version>1.1.0</version>
↪                                     = 2    <version>1.1.0</
↪version>
3    <directWMSIntegrationEnabled>true</directWMSIntegrationEnabled>
↪                                     3
↪<directWMSIntegrationEnabled>true</directWMSIntegrationEnabled>
4    <WMSEnabled>true</WMSEnabled>
↪                                     4    <WMSEnabled>true
↪</WMSEnabled>
5    <TMSEnabled>true</TMSEnabled>
↪                                     5    <TMSEnabled>true</
↪TMSEnabled>
6    <securityEnabled>false</securityEnabled>
↪                                     6    <securityEnabled>
↪false</securityEnabled>
7    <innerCachingEnabled>false</innerCachingEnabled>
↪                                     7
↪<innerCachingEnabled>false</innerCachingEnabled>
-----
8    <persistenceEnabled>true</persistenceEnabled>
↪                                     <> 8
↪<persistenceEnabled>false</persistenceEnabled>
-----
9    <cacheProviderClass>class org.geowebcache.storage.blobstore.memory.
↪guava.GuavaCacheProvider</cacheProviderClass> = 9
↪<cacheProviderClass>class org.geowebcache.storage.blobstore.memory.
↪guava.GuavaCacheProvider</cacheProviderClass>

```

(continues on next page)

(continued from previous page)

```

10    <cacheConfigurations>
↳                                     10
↳<cacheConfigurations>
11    <entry>
↳                                     11    <entry>
12        <string>class org.geowebcache.storage.blobstore.memory.guava.
↳GuavaCacheProvider</string>                                     12    <string>
↳class org.geowebcache.storage.blobstore.memory.guava.GuavaCacheProvider
↳</string>
13        <InnerCacheConfiguration>
↳                                     13
↳<InnerCacheConfiguration>
14        <hardMemoryLimit>16</hardMemoryLimit>
↳                                     14
↳<hardMemoryLimit>16</hardMemoryLimit>
-----
26    <defaultCachingGridSetIds>
↳                                     = 26
↳<defaultCachingGridSetIds>
27        <string>EPSG:4326</string>
↳                                     27    <string>
↳EPSG:4326</string>
28        <string>EPSG:900913</string>
↳                                     28    <string>
↳EPSG:900913</string>
29    </defaultCachingGridSetIds>
↳                                     29    </
↳defaultCachingGridSetIds>
30    <defaultCoverageCacheFormats>
↳                                     30
↳<defaultCoverageCacheFormats>
31        <string>image/png</string>
↳                                     31    <string>image/
↳png</string>
-----
32        <string>image/vnd.jpeg-png</string>
↳                                     +-
-----
33        <string>image/jpeg</string>
↳                                     = 32    <string>image/
↳jpeg</string>
34        <string>image/gif</string>
↳                                     33    <string>image/
↳gif</string>
35        <string>image/png8</string>
↳                                     34    <string>image/
↳png8</string>
36    </defaultCoverageCacheFormats>
↳                                     35    </
↳defaultCoverageCacheFormats>
37    <defaultVectorCacheFormats>
↳                                     36
↳<defaultVectorCacheFormats>
-----
38        <string>application/json;type=utfgrid</string>
↳                                     +-

```

(continues on next page)

(continued from previous page)

```

39      <string>image/png</string>
↪                                     = 37      <string>image/
↪png</string>
-----
40      <string>image/vnd.jpeg-png</string>
↪                                     +-
-----
41      <string>image/jpeg</string>
↪                                     = 38      <string>image/
↪jpeg</string>
42      <string>image/gif</string>
↪                                     39      <string>image/
↪gif</string>
43      <string>image/png8</string>
↪                                     40      <string>image/
↪png8</string>
44  </defaultVectorCacheFormats>
↪                                     41  </
↪defaultVectorCacheFormats>
45  <defaultOtherCacheFormats>
↪                                     42
↪<defaultOtherCacheFormats>
46      <string>image/png</string>
↪                                     43      <string>image/
↪png</string>
-----
↪                                     +- 44      <string>image/
↪jpeg</string>
↪                                     45      <string>image/
↪gif</string>
↪                                     46      <string>image/
↪png8</string>
-----
47  </defaultOtherCacheFormats>
↪                                     = 47  </
↪defaultOtherCacheFormats>
48 </GeoServerGWCCConfig>
↪                                     48 </
↪GeoServerGWCCConfig>
-----

```

- Create/modify `GEOSERVER_DATA_DIR/logs/QUIET_LOGGING.properties` as follows

```

## This log4j configuration file needs to stay here, and is used as the
↪default logging setup
## during data_dir upgrades and in case the chosen logging config isn't
↪available.
##
## As GeoTools uses java.util.logging logging instead of log4j,
↪GeoServer makes
## the following mappings to adjust the log4j levels specified in this
↪file to
## the GeoTools logging system:

```

(continues on next page)

(continued from previous page)

```
##
## Log4J Level          java.util.logging Level
## -----
## ALL                  FINEST
## TRACE                FINER
## DEBUG                FINE (includes CONFIG)
## INFO                 INFO
## ERROR/ERROR          ERRORING
## ERROR                SEVERE
## OFF                  OFF

log4j.rootLogger=OFF, stdout

log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{dd MMM HH:mm:ss} %p [%c
↳{2}] - %m%n
```

- Create/modify GEOSERVER_DATA_DIR/logs/TEST_LOGGING.properties as follows

```
## This log4j configuration file needs to stay here, and is used as the
↳default logging setup
## during data_dir upgrades and in case the chosen logging config isn't
↳available.
##
## As GeoTools uses java.util.logging logging instead of log4j,
↳GeoServer makes
## the following mappings to adjust the log4j levels specified in this
↳file to
## the GeoTools logging system:
##
## Log4J Level          java.util.logging Level
## -----
## ALL                  FINEST
## TRACE                FINER
## DEBUG                FINE (includes CONFIG)
## INFO                 INFO
## ERROR/ERROR          ERRORING
## ERROR                SEVERE
## OFF                  OFF

log4j.rootLogger=ERROR, stdout

log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{dd MMM HH:mm:ss} %p [%c
↳{2}] - %m%n

GEOTOOLS_DEVELOPER_LOGGING.properties

log4j.category.org.geotools=ERROR
log4j.category.org.geotools.factory=ERROR
log4j.category.org.geoserver=ERROR
log4j.category.org.vfny.geoserver=ERROR

log4j.category.org.springframework=ERROR
```

(continues on next page)

(continued from previous page)

```
# wicket tester
log4j.category.org.apache.wicket.util.tester=INFO
```

- Delete old security configuration files, in particular delete the following folders:

- GEOSERVER_DATA_DIR/security/auth/geonodeAuthProvider
- GEOSERVER_DATA_DIR/security/filter/geonodeAnonymousFilter
- GEOSERVER_DATA_DIR/security/filter/geonodeCookieFilter

- Update/modify the `GEOSERVER_DATA_DIR/security` as follows

```
- ./filter/geonode-oauth2/config.xml
```

```
Left file: D:\tmp\data-2.12.2\data\security\filter\geonode-
↳oauth2\config.xml
Right file: D:\tmp\data-2.9.x-
↳oauth2\data\security\filter\geonode-oauth2\config.xml
17  <!-- GeoServer Public URL -->
↳
↳
17  <!--
↳GeoServer Public URL -->
-----
↳-----
18  <redirectUri>http://localhost:8080/geoserver/index.html</
↳redirectUri>
↳
<> 18  <redirectUri>
↳http://localhost:8080/geoserver</redirectUri>
-----
↳-----
```

```
- ./role/geonode REST role service/config.xml
```

```

Left file: D:\tmp\data-2.12.2\data\security\role\geonode REST
↪role service\config.xml
Right file: D:\tmp\data-2.9.x-
↪oauth2\data\security\role\geonode REST role service\config.
↪xml
12 <adminRoleJSONPath>$.adminRole</adminRoleJSONPath>
↪
↪ 13 <adminRoleJSONPath>$.
↪adminRole</adminRoleJSONPath>
-----
↪-----
13 <usersJSONPath>$.users[?(@.username==&apos;${username}&
↪apos;)].groups</usersJSONPath> <> 14 <usersJSONPath>$.
↪users[0].groups</usersJSONPath>
14 <cacheConcurrencyLevel>4</cacheConcurrencyLevel>
15 <cacheMaximumSize>60000</cacheMaximumSize>
16 <cacheExpirationTime>60000</cacheExpirationTime>
-----
↪-----
17 </authKeyRESTRoleService>
↪
↪ = 15 </authKeyRESTRoleService>
-----
↪-----

```

- ./config.xml

```

Left file: D:\tmp\data-2.12.2\data\security\config.xml
Right file: D:\tmp\data-2.9.x-oauth2\data\security\config.xml

↪
↪
↪
↪          +- 2    <roleServiceName>geonode_
↪REST role service</roleServiceName>
-----
↪-----
↪-----
27    <filters name="gwc" class="org.geoserver.security.
↪ServiceLoginFilterChain" interceptorName="restInterceptor"
↪exceptionTranslationName="exception" path="/gwc/**" disabled=
↪"false" allowSessionCreation="false" ssl="false"
↪matchHTTPMethod="false">                                <> 28
↪<filters name="gwc" class="org.geoserver.security.
↪ServiceLoginFilterChain" interceptorName="restInterceptor"
↪exceptionTranslationName="exception" path="/gwc/rest/**"
↪disabled="false" allowSessionCreation="false" ssl="false"
↪matchHTTPMethod="false">
-----
↪-----
-----
↪-----
30    <filter>anonymous</filter>
↪
↪
↪
↪          +-
31    </filters>
32    <filters name="geofence-rest" class="org.geoserver.
↪security.ServiceLoginFilterChain" interceptorName=
↪"restInterceptor" exceptionTranslationName="exception" path=
↪"/geofence/rest/**" disabled="false" allowSessionCreation=
↪"false" ssl="false" matchHTTPMethod="false">
33    <filter>basic</filter>
34    <filter>geonode-oauth2</filter>
35    <filter>anonymous</filter>
36    </filters>
37    <filters name="geofence" class="org.geoserver.security.
↪ServiceLoginFilterChain" interceptorName="interceptor"
↪exceptionTranslationName="exception" path="/geofence/**"
↪disabled="false" allowSessionCreation="false" ssl="false"
↪matchHTTPMethod="false">
38    <filter>basic</filter>
39    <filter>geonode-oauth2</filter>
40    <filter>anonymous</filter>
-----
↪-----
-----
↪-----
52    <bruteForcePrevention>
↪
↪
↪
↪          +-

```

(continues on next page)

(continued from previous page)

```

53     <enabled>true</enabled>
54     <minDelaySeconds>1</minDelaySeconds>
55     <maxDelaySeconds>5</maxDelaySeconds>
56     <maxBlockedThreads>100</maxBlockedThreads>
57     <whitelistedMasks>
58         <string>127.0.0.1</string>
59     </whitelistedMasks>
60 </bruteForcePrevention>
-----
↪-----

```

– ./rest.properties

```

Left file: D:\tmp\data-2.12.2\data\security\rest.properties
Right file: D:\tmp\data-2.9.x-oauth2\data\security\rest.
↪properties
-----
↪-----

```

```

18 /rest/monitor/*;GET=ROLE_ADMINISTRATOR
↪                                     +-
19 /rest/security/*;GET,POST,DELETE,PUT=ROLE_ADMINISTRATOR
20 /rest/br/*;GET,POST,DELETE,PUT=ROLE_ADMINISTRATOR
21 /geofence/rest/*;GET,POST,DELETE,PUT=ROLE_ADMINISTRATOR
-----
↪-----

```

```

23 /**;POST,DELETE,PUT=ROLE_AUTHENTICATED
↪                                     <> 4 /**;POST,DELETE,
↪PUT=ROLE_ADMINISTRATOR
-----
↪-----

```

Note: In case of doubts you can always try to do a “diff” between your old GEOSERVER_DATA_DIR and <https://build.geo-solutions.it/geonode/geoserver/latest/data-2.12.2.zip>

• Update/tweak GeoNode settings.py

– Add LOGIN_REDIRECT_URL

```
LOGIN_REDIRECT_URL = '/'
```

– Modify INSTALLED_APPS as follows

```

Left file: D:\work\code\python\geonode\geonode-2.7.
↪x\geonode\settings.py
Right file: D:\work\code\python\geonode\geonode-2.6.
↪x\geonode\settings.py
281     # GeoServer Apps                                     = ↪
↪269     # GeoServer Apps
282     # Geoserver needs to come last because
↪270     # Geoserver needs to come last because
283     # it's signals may rely on other apps' signals.
↪271     # it's signals may rely on other apps' signals.
284     'geonode.geoserver',
↪272     'geonode.geoserver',

```

(continues on next page)

(continued from previous page)

```

285     'geonode.upload',
↪273     'geonode.upload',
286     'geonode.tasks',
↪274     'geonode.tasks',
-----
↪-----
287     'geonode.messaging',
-----
↪-----
288                                     =
↪275
289 )
↪276 )
290
↪277
291 GEONODE_CONTRIB_APPS = (
↪278 GEONODE_CONTRIB_APPS = (
292     # GeoNode Contrib Apps
↪279     # GeoNode Contrib Apps
-----
↪-----
293     # 'geonode.contrib.dynamic',
↪280     'geonode.contrib.dynamic',
294     # 'geonode.contrib.exif',
↪281     'geonode.contrib.exif',
295     # 'geonode.contrib.favorite',
↪282     'geonode.contrib.favorite',
296     # 'geonode.contrib.geogig',
↪283     'geonode.contrib.geogig',
297     # 'geonode.contrib.geosites',
↪284     'geonode.contrib.geosites',
298     # 'geonode.contrib.nlp',
↪285     'geonode.contrib.nlp',
299     # 'geonode.contrib.slack',
↪286     'geonode.contrib.slack',
-----
↪-----
300     # 'geonode.contrib.createlayer',
301     # 'geonode.contrib.datastore_shards',
-----
↪-----
302     'geonode.contrib.metadataxsl',
↪287     'geonode.contrib.metadataxsl'
303     'geonode.contrib.api_basemaps',
304     'geonode.contrib.ows_api',
-----
↪-----
305 )
↪288 )
306
↪289
307 # Uncomment the following line to enable contrib apps
↪290 # Uncomment the following line to enable contrib apps
-----
↪-----
308 GEONODE_APPS = GEONODE_CONTRIB_APPS + GEONODE_APPS
↪291 # GEONODE_APPS = GEONODE_APPS + GEONODE_CONTRIB_APPS

```

(continues on next page)

(continued from previous page)

```

-----
↪-----
309                                     = ↪
↪292
310 INSTALLED_APPS = (                                     ↪
↪293 INSTALLED_APPS = (                                     ↪
311                                     ↪
↪294
312     'modeltranslation',                                     ↪
↪295     'modeltranslation',                                     ↪
313                                     ↪
↪296
314     # Bootstrap admin theme                               ↪
↪297     # Bootstrap admin theme
-----
↪-----
↪-----
334     'taggit',                                     = ↪
↪317     'taggit',                                     ↪
335     'treebeard',                                     ↪
↪318     'treebeard',                                     ↪
336     'friendlytagloader',                             ↪
↪319     'friendlytagloader',                             ↪
337     'geoexplorer',                                   ↪
↪320     'geoexplorer',                                   ↪
338     'leaflet',                                       ↪
↪321     'leaflet',                                       ↪
339     'django_extensions',                             ↪
↪322     'django_extensions',
-----
↪-----
340     'django_basic_auth',                             <> ↪
↪323     #'geonode-client',
-----
↪-----
341     # 'haystack',                                     = ↪
↪324     # 'haystack',                                     ↪
342     'autocomplete_light',                             ↪
↪325     'autocomplete_light',                             ↪
343     'mptt',                                           ↪
↪326     'mptt',                                           ↪
344     # 'modeltranslation',                             ↪
↪327     # 'modeltranslation',                             ↪
345     # 'djkomu',                                         ↪
↪328     # 'djkomu',
-----
↪-----
346     # 'djcelery',                                     <> ↪
↪329     'djcelery',
-----
↪-----
347     # 'kombu.transport.django',                       = ↪
↪330     # 'kombu.transport.django',                       ↪
348
349     'storages',                                       ↪
↪331     'storages',

```

(continues on next page)

(continued from previous page)

```

-----
↪-----
350     'floppyforms',                                +-
-----
↪-----
351                                           =  ↪
↪332
352     # Theme                                     ↪
↪333     # Theme
-----
↪-----
↪334     "pinax_theme_bootstrap_account",          -+ ↪
-----
↪-----
353     "pinax_theme_bootstrap",                    =  ↪
↪335     "pinax_theme_bootstrap",
354     'django_forms_bootstrap',                    ↪
↪336     'django_forms_bootstrap',
355                                           ↪
↪337
356     # Social                                    ↪
↪338     # Social
357     'account',                                  ↪
↪339     'account',
358     'avatar',                                  ↪
↪340     'avatar',
-----
↪-----
↪-----
364     'actstream',                                =  ↪
↪345     'actstream',
365     'user_messages',                            ↪
↪346     'user_messages',
366     'tastypie',                                  ↪
↪347     'tastypie',
367     'polymorphic',                              ↪
↪348     'polymorphic',
368     'guardian',                                  ↪
↪349     'guardian',
369     'oauth2_provider',                            ↪
↪350     'oauth2_provider',
-----
↪-----
370     'corsheaders',                                +-
-----
↪-----
371                                           =  ↪
↪351
-----
↪-----
372     'invitations',                                +-
-----
↪-----
373 ) + GEONODE_APPS                                =  ↪
↪352 ) + GEONODE_APPS

```

(continues on next page)

(continued from previous page)

```
-----
↪-----
```

- Add MONITORING flags as follows

```
MONITORING_ENABLED = False

# how long monitoring data should be stored
MONITORING_DATA_TTL = timedelta(days=7)

# this will disable csrf check for notification config views,
# use with caution - for dev purpose only
MONITORING_DISABLE_CSRF = False
```

- Update LOGGING handlers as follows

```
Left file: D:\work\code\python\geonode\geonode-2.7.
↪x\geonode\settings.py
Right file: D:\work\code\python\geonode\geonode-2.6.
↪x\geonode\settings.py
396     'filters': {
↪= 366         'filters': {
397         'require_debug_false': {
↪ 367             'require_debug_false': {
398                 '()': 'django.utils.log.RequireDebugFalse'
↪ 368                 '()': 'django.utils.log.RequireDebugFalse'
399             }
↪ 369         },
400     },
↪ 370 },
401     'handlers': {
↪ 371         'handlers': {
-----
↪-----
↪+ 372         'null': {
↪ 373             'level': 'ERROR',
↪ 374             'class': 'django.utils.log.NullHandler',
↪ 375         },
-----
↪-----
402         'console': {
↪= 376             'console': {
403                 'level': 'ERROR',
↪ 377                 'level': 'ERROR',
404                 'class': 'logging.StreamHandler',
↪ 378                 'class': 'logging.StreamHandler',
405                 'formatter': 'simple'
↪ 379                 'formatter': 'simple'
406             },
↪ 380         },
407         'mail_admins': {
↪ 381             'mail_admins': {
-----
↪-----
```

(continues on next page)

(continued from previous page)

```

-----
↪-----
410         }
↪= 384         }
411     },
↪ 385     },
412     "loggers": {
↪ 386         "loggers": {
413             "django": {
↪ 387                 "django": {
414                     "handlers": ["console"], "level": "ERROR", },
↪ 388                     "handlers": ["console"], "level": "ERROR",
↪ },
415         "geonode": {
↪ 389             "geonode": {
-----
↪-----
416             "handlers": ["console"], "level": "ERROR", },
↪+-
417         "geonode.qgis_server": {
-----
↪-----
418             "handlers": ["console"], "level": "ERROR", },
↪= 390             "handlers": ["console"], "level": "ERROR",
↪ },
419         "gsconfig.catalog": {
↪ 391             "gsconfig.catalog": {
420                 "handlers": ["console"], "level": "ERROR", },
↪ 392                 "handlers": ["console"], "level": "ERROR",
↪ },
421         "owslib": {
↪ 393             "owslib": {
422                 "handlers": ["console"], "level": "ERROR", },
↪ 394                 "handlers": ["console"], "level": "ERROR",
↪ },
423         "pycsw": {
↪ 395             "pycsw": {
424                 "handlers": ["console"], "level": "ERROR", },
↪ 396                 "handlers": ["console"], "level": "ERROR",
↪ },
425     },
↪ 397 },
426 }
↪ 398 }
-----
↪-----

```

- Update MIDDLEWARE and SECURITY flags as follows

```

Left file: D:\work\code\python\geonode\geonode-2.7.
↪x\geonode\settings.py
Right file: D:\work\code\python\geonode\geonode-2.6.
↪x\geonode\settings.py
458 MIDDLEWARE_CLASSES = (
↪
-----
↪-----

```

(continues on next page)

(continued from previous page)

```

459     'corsheaders.middleware.CorsMiddleware',
↳                                     +-
-----
↳ -----
460     'django.middleware.common.CommonMiddleware',
↳                                     = 431     'django.middleware.common.
↳ CommonMiddleware',
461     'django.contrib.sessions.middleware.SessionMiddleware',
↳                                     432     'django.contrib.sessions.
↳ middleware.SessionMiddleware',
462     'django.contrib.messages.middleware.MessageMiddleware',
↳                                     433     'django.contrib.messages.
↳ middleware.MessageMiddleware',
463
↳                                     434
464     # The setting below makes it possible to serve
↳ different languages per 435     # The setting below
↳ makes it possible to serve different languages per
465     # user depending on things like headers in HTTP
↳ requests. 436     # user depending on
↳ things like headers in HTTP requests.
-----
↳ -----
↳ -----
467     'pagination.middleware.PaginationMiddleware',
↳                                     = 438     'pagination.middleware.
↳ PaginationMiddleware',
468     'django.middleware.csrf.CsrfViewMiddleware',
↳                                     439     'django.middleware.csrf.
↳ CsrfViewMiddleware',
469     'django.contrib.auth.middleware.
↳ AuthenticationMiddleware', 440
↳ 'django.contrib.auth.middleware.AuthenticationMiddleware',
470     'django.middleware.clickjacking.XFrameOptionsMiddleware'
↳ , 441     'django.middleware.
↳ clickjacking.XFrameOptionsMiddleware',
471
↳                                     442
472     # Security settings
-----
↳ -----
473     'django.middleware.security.SecurityMiddleware',
↳                                     +-
-----
↳ -----
474
↳                                     =
475     # This middleware allows to print private layers for
↳ the users that have 443     # This middleware allows
↳ to print private layers for the users that have
476     # the permissions to view them.
↳ 444     # the permissions to view
↳ them.
477     # It sets temporary the involved layers as public
↳ before restoring the 445     # It sets temporary the
↳ involved layers as public before restoring the

```

(continues on next page)

(continued from previous page)

```

478     # permissions.
↳                                     446     # permissions.
479     # Beware that for few seconds the involved layers are
↳public there could be      447     # Beware that for few
↳seconds the involved layers are public there could be
-----
↳-----
↳-----
485     # django-oauth-toolkit.
↳                                     =
486     'django.contrib.auth.middleware.
↳SessionAuthenticationMiddleware',      453
↳'django.contrib.auth.middleware.
↳SessionAuthenticationMiddleware',
487     'oauth2_provider.middleware.OAuth2TokenMiddleware',
↳                                     454     'oauth2_provider.middleware.
↳OAuth2TokenMiddleware',
488 )
↳                                     455 )
489
↳                                     456
490 # Security stuff
-----
↳-----
491 MIDDLEWARE_CLASSES += ('django.middleware.security.
↳SecurityMiddleware',) +-
492 SESSION_COOKIE_SECURE = False
493 CSRF_COOKIE_SECURE = False
494 CSRF_COOKIE_HTTPONLY = False
495 X_FRAME_OPTIONS = 'DENY'
496 SECURE_CONTENT_TYPE_NOSNIFF = True
497 SECURE_BROWSER_XSS_FILTER = True
498 SECURE_SSL_REDIRECT = False
499 SECURE_HSTS_SECONDS = 3600
500 SECURE_HSTS_INCLUDE_SUBDOMAINS = True
-----
↳-----
501
↳                                     = 457
502 # Replacement of default authentication backend in order
↳to support      458 # Replacement of default
↳authentication backend in order to support
503 # permissions per object.
↳                                     459 # permissions per object.
504 AUTHENTICATION_BACKENDS = (
↳                                     460 AUTHENTICATION_BACKENDS = (
505     'oauth2_provider.backends.OAuth2Backend',
↳                                     461     'oauth2_provider.backends.
↳OAuth2Backend',
506     'django.contrib.auth.backends.ModelBackend',
↳                                     462     'django.contrib.auth.
↳backends.ModelBackend',
-----
↳-----
↳-----

```

(continues on next page)

(continued from previous page)

```

529 # Whether the uploaded resources should be public and
↳downloadable by default = 485 # Whether the uploaded
↳resources should be public and downloadable by default
530 # or not
↳
↳486 # or not
531 DEFAULT_ANONYMOUS_VIEW_PERMISSION = strtobool(
↳487 DEFAULT_ANONYMOUS_VIEW_
↳PERMISSION = strtobool(
532 os.getenv('DEFAULT_ANONYMOUS_VIEW_PERMISSION', 'True')
↳488 os.getenv('DEFAULT_ANONYMOUS_
↳VIEW_PERMISSION', 'True')
533 )
↳489 )
534 DEFAULT_ANONYMOUS_DOWNLOAD_PERMISSION = strtobool(
↳490 DEFAULT_ANONYMOUS_DOWNLOAD_
↳PERMISSION = strtobool(
-----
↳-----
535 os.getenv('DEFAULT_ANONYMOUS_DOWNLOAD_PERMISSION',
↳'True') <> 491 os.getenv('DEFAULT_
↳ANONYMOUS_VIEW_PERMISSION', 'True')
-----
↳-----
536 )
↳492 )
537
↳493
538 #
↳494 #
539 # Settings for default search size
↳495 # Settings for default search_
↳size
540 #
↳496 #
541 DEFAULT_SEARCH_SIZE = int(os.getenv('DEFAULT_SEARCH_SIZE',
↳'10')) 497 DEFAULT_SEARCH_SIZE = int(os.
↳getenv('DEFAULT_SEARCH_SIZE', '10'))
-----
↳-----
↳-----
565 'USE_JSONFIELD': True,
↳521 'USE_JSONFIELD': True,
566 'GFK_FETCH_DEPTH': 1,
↳522 'GFK_FETCH_DEPTH': 1,
567 }
↳523 }
568
↳524
569
570 # prevent signing up by default
↳525 # Settings for Social Apps
-----
↳-----
571 ACCOUNT_OPEN_SIGNUP = True
↳526 REGISTRATION_OPEN = strtobool(os.
↳getenv('REGISTRATION_OPEN', 'False'))

```

(continues on next page)

(continued from previous page)

```

-----
↪-----
572                                     =                                     ↪
↪                                     ↪
573 ACCOUNT_EMAIL_CONFIRMATION_EMAIL = strtobool(                                     ↪
↪                                     527 ACCOUNT_EMAIL_CONFIRMATION_EMAIL ↪
↪= strtobool(
574     os.getenv('ACCOUNT_EMAIL_CONFIRMATION_EMAIL', 'False') ↪
↪                                     528     os.getenv('ACCOUNT_EMAIL_
↪CONFIRMATION_EMAIL', 'False')
575 )                                     ↪
↪                                     529 )
576 ACCOUNT_EMAIL_CONFIRMATION_REQUIRED = strtobool(                                     ↪
↪                                     530 ACCOUNT_EMAIL_CONFIRMATION_
↪REQUIRED = strtobool(
577     os.getenv('ACCOUNT_EMAIL_CONFIRMATION_REQUIRED', 'False
↪')                                     531     os.getenv('ACCOUNT_EMAIL_
↪CONFIRMATION_REQUIRED', 'False')
578 )                                     ↪
↪                                     532 )
579 ACCOUNT_APPROVAL_REQUIRED = strtobool(                                     ↪
↪                                     533 ACCOUNT_APPROVAL_REQUIRED = ↪
↪strtobool(
580     os.getenv('ACCOUNT_APPROVAL_REQUIRED', 'False') ↪
↪                                     534     os.getenv('ACCOUNT_APPROVAL_
↪REQUIRED', 'False')
581 )                                     ↪
↪                                     535 )
-----
↪-----

```

– Update the Uploader Settings as follows

```

UPLOADER = {
    'BACKEND': 'geonode.rest',
    'OPTIONS': {
        'TIME_ENABLED': False,
        'MOSAIC_ENABLED': False,
        'GEOGIG_ENABLED': False,
    },
    'SUPPORTED_CRIS': [
        'EPSG:4326',
        'EPSG:3785',
        'EPSG:3857',
        'EPSG:900913',
        'EPSG:32647',
        'EPSG:32736'
    ],
    'SUPPORTED_EXT': [
        '.shp',
        '.csv',
        '.kml',
        '.kmz',
        '.json',
        '.geojson',
        '.tif',
        '.tiff',

```

(continues on next page)

(continued from previous page)

```

        '.geotiff',
        '.gml',
        '.xml'
    ]
}

```

– Update/modify NOTIFICATIONS settings as follows

```

Left file: D:\work\code\python\geonode\geonode-2.7.
↳x\geonode\settings.py
Right file: D:\work\code\python\geonode\geonode-2.6.
↳x\geonode\settings.py
1099 # notification settings
↳      =
-----
↳-----
1100 NOTIFICATION_ENABLED = True or TEST
↳      +-
1101 PINAX_NOTIFICATIONS_LANGUAGE_MODEL = "account.Account"
-----
↳-----
1102
↳      =
1103 # notifications backends
-----
↳-----
1104 _EMAIL_BACKEND = "pinax.notifications.backends.email.
↳EmailBackend" +-
1105 PINAX_NOTIFICATIONS_BACKENDS = [
1106     ("email", _EMAIL_BACKEND),
1107 ]
-----
↳-----
1108
↳      =
1109 # Queue non-blocking notifications.
↳      969 # Queue non-blocking notifications.
-----
↳-----
1110 PINAX_NOTIFICATIONS_QUEUE_ALL = False
↳      <> 970 NOTIFICATION_QUEUE_ALL = False
1111 PINAX_NOTIFICATIONS_LOCK_WAIT_TIMEOUT = -1
-----
↳-----
1112
↳      = 971
1113 # explicitly define NOTIFICATION_LOCK_LOCATION
1114 # NOTIFICATION_LOCK_LOCATION = <path>
1115
1116 # pinax.notifications
1117 # or notification
↳      972 # notification settings
-----
↳-----
1118 NOTIFICATIONS_MODULE = 'pinax.notifications'
↳      <> 973 NOTIFICATION_LANGUAGE_MODULE = "account.
↳Account"

```

(continues on next page)

(continued from previous page)

```

-----
1119                                     =
1120 # set to true to have multiple recipients in /message/
1121 create/
-----
1121 USER_MESSAGES_ALLOW_MULTIPLE_RECIPIENTS = False
1122                                     +-
-----
1122                                     =
-----
1123 if NOTIFICATION_ENABLED:
1124     +-
1124     if NOTIFICATIONS_MODULE not in INSTALLED_APPS:
1125         INSTALLED_APPS += (NOTIFICATIONS_MODULE, )
-----

```

– Update/modify CELERY settings as follows

```

Left file: D:\work\code\python\geonode\geonode-2.7.
1127 x\geonode\settings.py
Right file: D:\work\code\python\geonode\geonode-2.6.
1128 x\geonode\settings.py
1127 # async signals can be the same as broker url
1128                                     =
1128 # but they should have separate setting anyway
1129 # use amqp:// for local rabbitmq server
-----
1130 ASYNC_SIGNALS_BROKER_URL = 'memory://'
1131                                     +-
-----
1131                                     =
-----
1132 CELERY_BROKER_URL = os.getenv('BROKER_URL', "amqp://")
1133                                     <> 974 BROKER_URL = os.getenv('BROKER_URL',
1134                                     "django://")
-----
1135                                     975 CELERY_ALWAYS_EAGER = True
1136                                     976 CELERY_EAGER_PROPAGATES_EXCEPTIONS =
1137 True
1138                                     977 CELERY_IGNORE_RESULT = True
1139                                     978 CELERY_SEND_EVENTS = False
-----

```

(continues on next page)

(continued from previous page)

```

1133 CELERY_RESULT_BACKEND = None
↳          = 979 CELERY_RESULT_BACKEND = None
-----
↳-----
1134 CELERY_TASK_ALWAYS_EAGER = True # set this to False in_
↳order to run async +-
1135 CELERY_TASK_IGNORE_RESULT = True
1136 CELERY_TASK_DEFAULT_QUEUE = "default"
1137 CELERY_TASK_DEFAULT_EXCHANGE = "default"
1138 CELERY_TASK_DEFAULT_EXCHANGE_TYPE = "direct"
1139 CELERY_TASK_DEFAULT_ROUTING_KEY = "default"
1140 CELERY_TASK_CREATE_MISSING_QUEUES = True
-----
↳-----
1141 CELERY_TASK_RESULT_EXPIRES = 1
↳          = 980 CELERY_TASK_RESULT_EXPIRES = 1
-----
↳-----
1142 CELERY_WORKER_DISABLE_RATE_LIMITS = True
↳          <> 981 CELERY_DISABLE_RATE_LIMITS = True
↳
↳          982 CELERY_DEFAULT_QUEUE = "default"
↳
↳          983 CELERY_DEFAULT_EXCHANGE = "default"
↳
↳          984 CELERY_DEFAULT_EXCHANGE_TYPE =
↳"direct"
1143 CELERY_WORKER_SEND_TASK_EVENTS = False
↳          985 CELERY_DEFAULT_ROUTING_KEY = "default"
↳"
1144
↳          986 CELERY_CREATE_MISSING_QUEUES = True
1145 CELERY_QUEUES = [
↳          987 CELERY_IMPORTS = (
1146     Queue('default', routing_key='default'),
↳          988     'geonode.tasks.deletion',
1147     Queue('cleanup', routing_key='cleanup'),
1148     Queue('update', routing_key='update'),
↳          989     'geonode.tasks.update',
1149     Queue('email', routing_key='email'),
↳          990     'geonode.tasks.email'
1150 ]
↳          991 )
-----
↳-----
1177          = 1018
1178          1019
-----
1179 # djcelery.setup_loader() <> 1020 djcelery.setup_loader()
-----
1180          = 1021
-----

```

– Additional/new Geonode behavior settings

```

DISPLAY_SOCIAL = strtobool(os.getenv('DISPLAY_SOCIAL', 'True'))
DISPLAY_COMMENTS = strtobool(os.getenv('DISPLAY_COMMENTS',
↳ 'True'))
DISPLAY_RATINGS = strtobool(os.getenv('DISPLAY_RATINGS', 'True
↳ '))
DISPLAY_WMS_LINKS = strtobool(os.getenv('DISPLAY_WMS_LINKS',
↳ 'True'))

# Number of results per page listed in the GeoNode search pages
CLIENT_RESULTS_LIMIT = int(os.getenv('CLIENT_RESULTS_LIMIT',
↳ '20'))

# Number of items returned by the apis 0 equals no limit
API_LIMIT_PER_PAGE = int(os.getenv('API_LIMIT_PER_PAGE', '200
↳ '))
API_INCLUDE_REGIONS_COUNT = strtobool(
    os.getenv('API_INCLUDE_REGIONS_COUNT', 'False'))
# Make Free-Text Keywords writable from users or read-only
# - if True only admins can edit free-text kwds from admin_
↳ dashboard
FREETEXT_KEYWORDS_READONLY = False

# Each uploaded Layer must be approved by an Admin before_
↳ becoming visible
ADMIN_MODERATE_UPLOADS = False

# add following lines to your local settings to enable_
↳ monitoring
if MONITORING_ENABLED:
    if 'geonode.contrib.monitoring' not in INSTALLED_APPS:
        INSTALLED_APPS += ('geonode.contrib.monitoring',)
    if 'geonode.contrib.monitoring.middleware.
↳ MonitoringMiddleware' not in MIDDLEWARE_CLASSES:
        MIDDLEWARE_CLASSES += \
            ('geonode.contrib.monitoring.middleware.
↳ MonitoringMiddleware',)

GEOIP_PATH = os.path.join(PROJECT_ROOT, 'GeoIPCities.dat')
# If this option is enabled, Resources belonging to a Group won
↳ 't be
# visible by others
GROUP_PRIVATE_RESOURCES = False

# If this option is enabled, Groups will become strictly_
↳ Mandatory on
# Metadata Wizard
GROUP_MANDATORY_RESOURCES = False

# A boolean which specifies wether to display the email in user
↳ 's profile
SHOW_PROFILE_EMAIL = False

# Enables cross origin requests for geonode-client
MAP_CLIENT_USE_CROSS_ORIGIN_CREDENTIALS = strtobool(os.getenv(
    'MAP_CLIENT_USE_CROSS_ORIGIN_CREDENTIALS',
    'False'
))

```

- Update/modify THUMBNAIL GENERATOR

```
Left file: D:\work\code\python\geonode\geonode-2.7.
↳x\geonode\settings.py
Right file: D:\work\code\python\geonode\geonode-2.6.
↳x\geonode\settings.py
1284
↳
=
1285 # Choose thumbnail generator -- this is the default_
↳generator 1077 # Choose thumbnail generator --_
↳this is the default generator
-----
↳-----
1286 THUMBNAIL_GENERATOR = "geonode.layers.utils.create_gs_
↳thumbnail_geonode" <> 1078 THUMBNAIL_GENERATOR = "geonode.
↳geoserver.helpers.create_gs_thumbnail_geonode"
-----
↳-----
```

Final Steps

1. Run paver setup in order to download the latest Jetty Runner

Warning: Don't do this if your GEOSERVER_DATA_DIR is located under geonode/geoserver/data; it will be wiped out!! In this case download Jetty Runner manually from <http://repo2.maven.org/maven2/org/eclipse/jetty/jetty-runner/9.4.7.v20170914/jetty-runner-9.4.7.v20170914.jar> And put it under geonode/downloaded folder

```
DJANGO_SETTINGS_MODULE=geonode.local_settings paver setup
```

2. Start the server

```
DJANGO_SETTINGS_MODULE=geonode.local_settings paver start
```

3. Re-sync GeoFence Security Rules

```
DJANGO_SETTINGS_MODULE=geonode.local_settings python manage.py sync_
↳geofence
```

Appendix A

Warning: Before you proceed with the following steps, it is recommended that you perform a full backup of your current environment. In the next section there's a listing of the steps required to perform a full / hard backup of the whole stack.

Backup of the old Environment

Backup of the DataBase

1. From "local_settings" or "settings" (vim geonode/local_settings.py) retrieve all the DB connection parameters

```

geosolutions@geonode-2: ~/geonode
# You should have received a copy of the GNU General Public License
# along with this program. If not, see <http://www.gnu.org/licenses/>.
#
#####

import os

PROJECT_ROOT = os.path.abspath(os.path.dirname(__file__))

SITEURL = "http://localhost:8000/"

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'geonode',
        'USER': 'geonode',
        'PASSWORD': 'geonode',
    },
    # vector datastore for uploads
    'datastore' : {
        'ENGINE': 'django.contrib.gis.db.backends.postgis',
        # 'ENGINE': '', # Empty ENGINE name disables
        'NAME': 'geonode_data',
        'USER' : 'geonode',
        'PASSWORD' : 'geonode',
        'HOST' : 'localhost',
        'PORT' : '5432',
    }
}

```

2. Dump all the DBs

```

sudo su - postgres
pg_dump -d geonode -U geonode -f /tmp/geonode.dump
pg_dump -d geonode_data -U geonode -f /tmp/geonode_data.dump

```

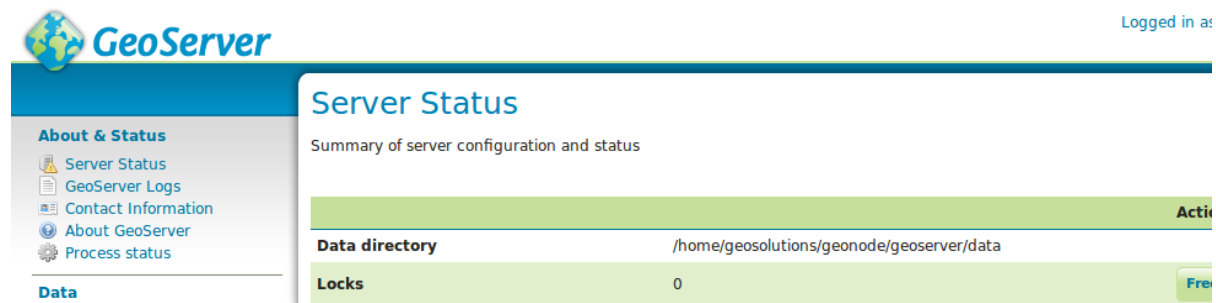
Backup of GeoServer

1. Backup the old GeoServer binaries

```
tar czvf /tmp/geoserver.tar.gz geoserver/
```

2. Backup of the GeoServer Data Dir

- As an admin login into GeoServer gui (<http://localhost:8080/geoserver/>)
- Click on “Server Status” and note the “Data Directory” path



Server Status
Summary of server configuration and status

Data directory	/home/geosolutions/geonode/geoserver/data	Active
Locks	0	Free

```
tar czvf /tmp/geoserver_data.tar.gz /home/geosolutions/geonode/geoserver/data/
```

3. Backup of Uploaded/Media and Static files

```
DJANGO_SETTINGS_MODULE=geonode.local_settings python manage.py print_settings | grep MEDIA_ROOT
```

```
MEDIA_ROOT = '/home/geosolutions/geonode/geonode/uploaded'
explicit_settings = set(['CASCADE_WORKSPACE', 'AWS_S3_BUCKET_DOMAIN', 'UPLOADER', 'LANGUAGE_CODE', 'LICENSES', 'MANAGERS', 'DOWNLOAD_FORMATS_METADATA', 'BRALIZER', 'DEFAULT_MAP_CENTER', 'REGISTRATION_OPEN', 'ALLOWED_HOSTS', 'OGC_SERVER', 'DOWNLNDERS', 'NOTIFICATION_QUEUE_ALL', 'S3_MEDIA_ENABLED', 'CELERY_IMPORTS', 'SOCIAL_BUTTONS', 'ME', 'DATABASES', '_DEFAULT_STATICFILES_DIRS', 'ACCOUNT_APPROVAL_REQUIRED', 'DEBUG_STATICGUAGES', 'MEDIA_ROOT', 'STAMEN_BASEMAPS', 'GEOSERVER_LOCATION', 'CACHES', 'CKAN_ORIGINS', '_OSM_BASEMAPS', 'ROOT_URLCONF', 'CATALOGUE', 'LOCKDOWN_GEONODE', 'MISSING_THUMBNAI', 'MILATION_FALLBACK_LANGUAGES', 'DEFAULT_MAP_CRS', 'THESAURI', '_DEFAULT_LANGUAGES', 'TASTYPILT_ANONYMOUS_DOWNLOAD_PERMISSION', 'CELERY_DEFAULT_EXCHANGE', 'OAUTH2_PROVIDER', 'CELERY_EAGER', 'MODIFY_TOPICCATEGORY', 'NLP_ENABLED', 'GUARDIAN_GET_INIT_ANONYMOUS_USER', 'PYCSHE TIME', 'SITEURL', 'STATIC_ROOT', 'GEONODE_APPS', 'LEAFLET_CONFIG', 'DOWNLOAD_FORMATS_V_ACCESS_KEY', 'ACCOUNT_EMAIL_CONFIRMATION_EMAIL', 'AWS_QUERYSTRING_AUTH', 'LOGIN_URL', 'SGATES_EXCEPTIONS', 'CELERY_IGNORE_RESULT', 'AUTH_USER_MODEL', 'HAYSTACK_SEARCH', 'AUTHENTLT_SECRET_KEY', 'SERVICE_UPDATE_INTERVAL', 'CELERY_RESULT_BACKEND', 'API_INCLUDE_REGIONST', 'LOGGING', 'CELERY_SEND_EVENTS', 'EXIF_ENABLED', 'NOSE_ARGS', 'NLP_MODEL_PATH', 'INTEEMAPS', 'SOCIAL_ORIGINS', 'MAP_BASELAYERS', 'USE_L10N', 'GEONODE_CONTRIB_APPS', 'NLP_LOCAPASSWORD_CHANGE', 'DEFAULT_ANONYMOUS_VIEW_PERMISSION', 'AGON_RATINGS_CATEGORY_CHOICES', 'MENT_TYPES', 'NOTIFICATION_LANGUAGE_MODULE', 'RESOURCE_PUBLISHING', 'PROXY_URL', 'SITE_ID
```

```
tar czvf /tmp/geonode_media.tar.gz /home/geosolutions/geonode/geonode/uploaded
```

- do the same for STATIC_ROOT, TEMPLATES (all folders listed), LOCALE (all folders listed)

4. Backup of the original source code

- Make sure you have everything committed and pushed for your local Git branches
- In case you are working locally, make sure you saved everything before proceeding with the update

Quick Installation Guide This section is a quick guide to get GeoNode up and running in most common operating systems. This is meant to be run on a fresh machine with no previously installed packages or GeoNode versions.

Linux Admin Intro This section describes how to setup a Virtual Machine running Ubuntu.

GeoNode (v2.8) on Docker This section describes how to setup GeoNode on Docker

GeoNode (v2.8) installation on Ubuntu 16.04 This section will guide the user through the steps necessary to install GeoNode on Ubuntu.

GeoNode (v2.8) installation on CentOS 7 This section will guide the user through the steps necessary to install GeoNode on CentOS.

Packaging for automatic installation are provided for Ubuntu, so, the only option for installing GeoNode on a CentOS platform is installing it from source.

Windows Binary Installer Install GeoNode on Windows through the binary installer.

Network configuration issues This section will guide the user through the steps necessary to understand and possibly solve any of the most common configuration issues on communication between Django and GeoServer.

GeoNode (v2.8) update from older versions This section will guide the user through the steps necessary to update GeoNode from old versions.

3.2.1.3 Users Workshop

Welcome to the GeoNode Training *Users Workshop* documentation v2.8.

This workshop will teach how to use the **GeoNode** going in depth into what we can do with software application. At the end of this section you will master all the GeoNode sections and entities from a user perspective.

You will know how to:

1. Manage users accounts and how to modify them.
2. Use and manage the different GeoNode basic resources.
3. Use the GeoNode searching tools to find your resources.
4. Manage Layers and Maps, update the styles and publish them.
5. Load datasets into GeoNode and keep them synchronized with GeoServer.

Prerequisites

Before proceeding with the reading, it is strongly recommended to be sure having clear the following concepts:

1. GeoNode and Django framework basic concepts
2. What is Python
3. What is a geospatial server and a basic knowledge of the geospatial web services.
4. What is a metadata and a catalog.
5. What is a map and a legend.

3.2.1.3.1 Accounts and users

GeoNode is primarily a *social* platform, and thus a primary component of any GeoNode instance is the user account. This section will guide you through account registration, updating your account information, and viewing other user accounts.

Creating a new account

Before you can save or edit any layers on a GeoNode instance, you need to create an account.

1. From any page in the web interface, you will see a *Register* link. Click that link, and the register form will appear

Note: The registrations in GeoNode must be open, in case you don't see the register link then it's not possible to register unless the administrator of the site does that for you.

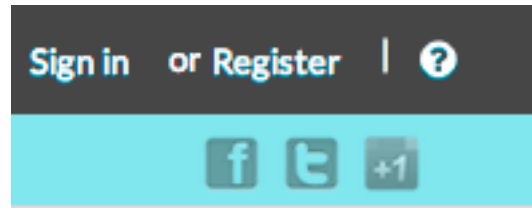


Fig. 1: Sign in screen

2. On the next page, fill out the form. Enter a user name and password in the fields. Also, enter your email address for verification.

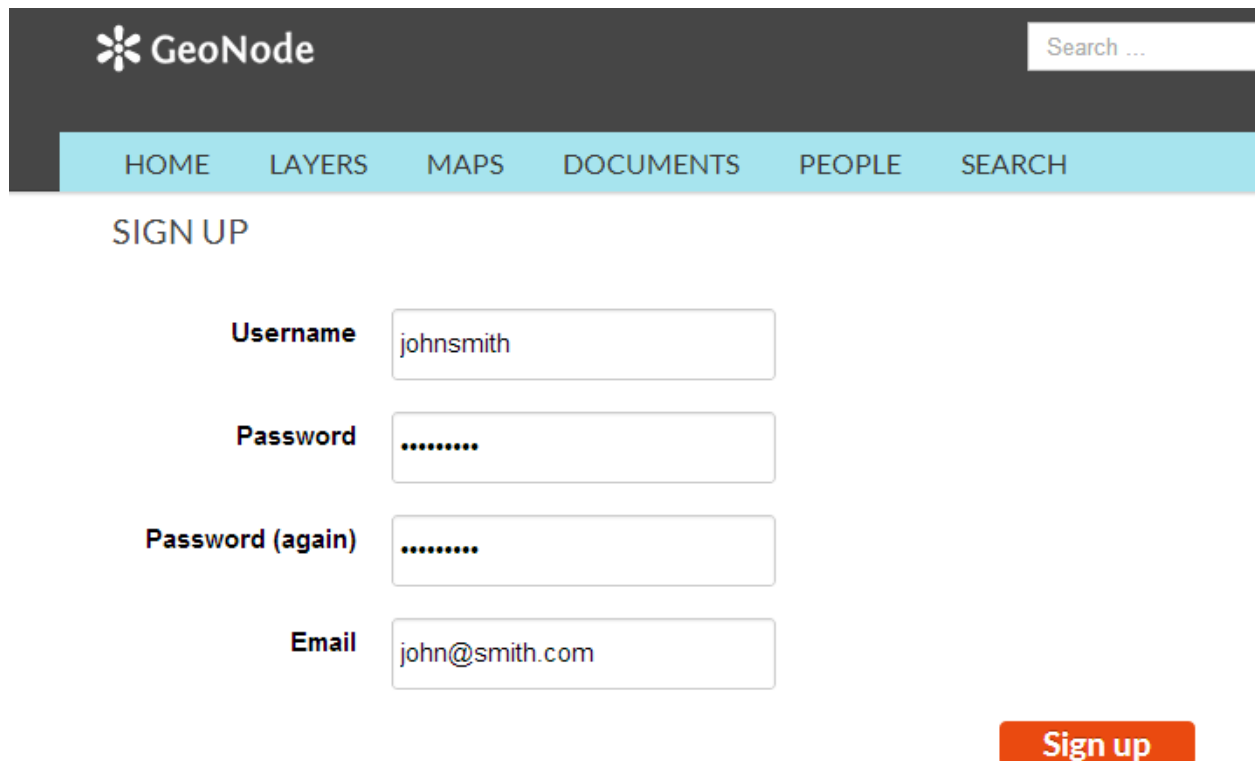
The registration form is on a dark grey background. At the top left is the GeoNode logo (a white asterisk-like shape) and the text "GeoNode". To the right is a search bar with the placeholder text "Search ...". Below the header is a light blue navigation bar with the links: HOME, LAYERS, MAPS, DOCUMENTS, PEOPLE, and SEARCH. The main content area is titled "SIGN UP" in large, bold, white letters. Below this are four form fields, each with a label to its left: "Username" with the value "johnsmith", "Password" with masked characters ".....", "Password (again)" with masked characters ".....", and "Email" with the value "john@smith.com". At the bottom right of the form is an orange button with the text "Sign up" in white.

Fig. 2: Registering for a new account

3. You will be returned to the welcome page. An email will be sent confirming that you have signed up. While you are now logged in, you will need to confirm your account. Navigate to the link that was sent in the email.
4. Click *Confirm*. You will be returned to the homepage.

Managing your profile

Your profile contains personal information.

1. Click on your user name in the top right of the screen. A drop-down list will show. Click on *Profile* to enter the Profile settings page.
2. The next page shows your profile, which is currently empty.

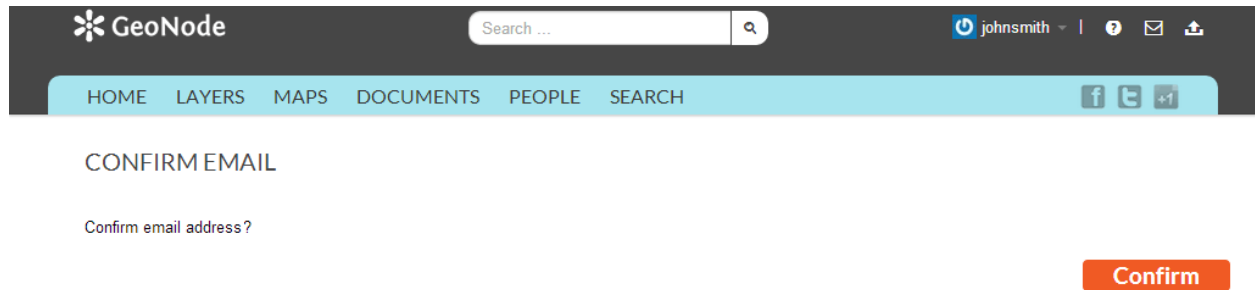


Fig. 3: Confirming your email address

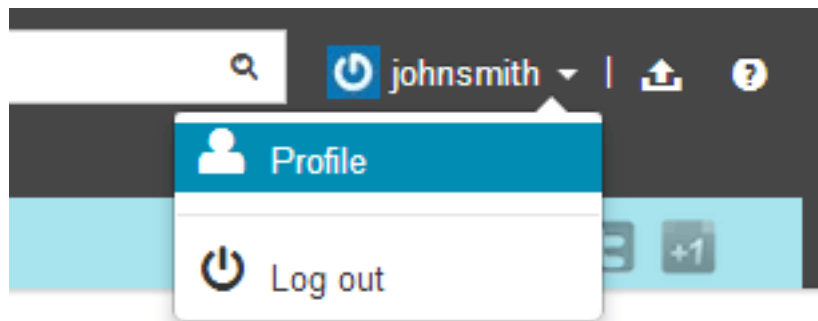


Fig. 4: Link to your profile

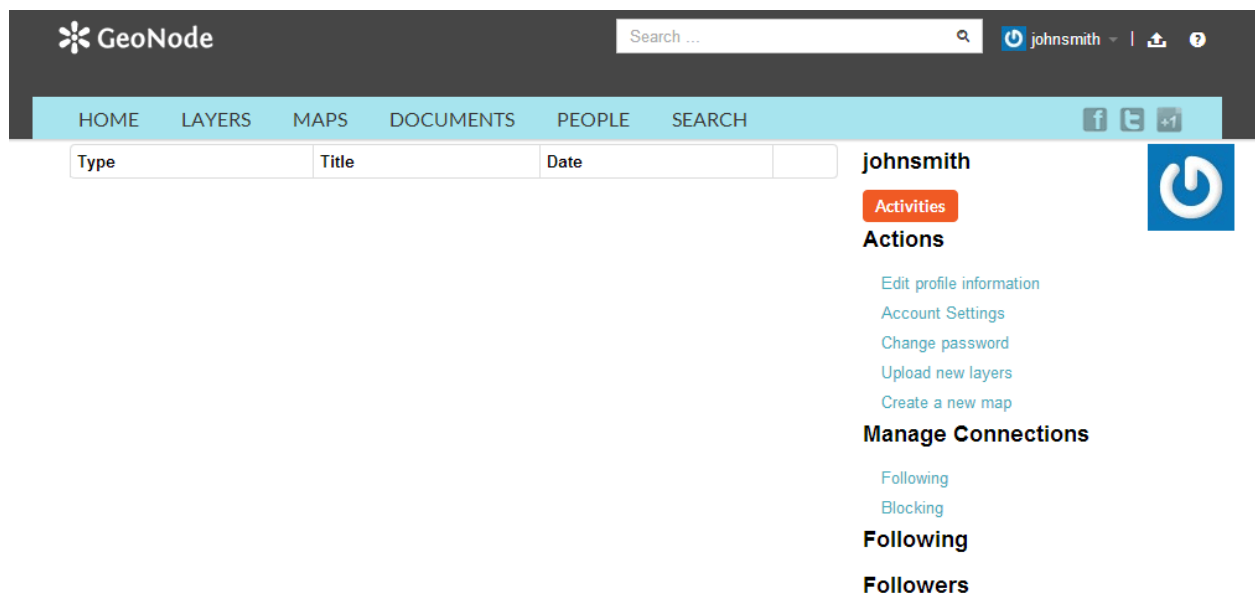


Fig. 5: Profile page

3. Click the *Edit profile information* link.

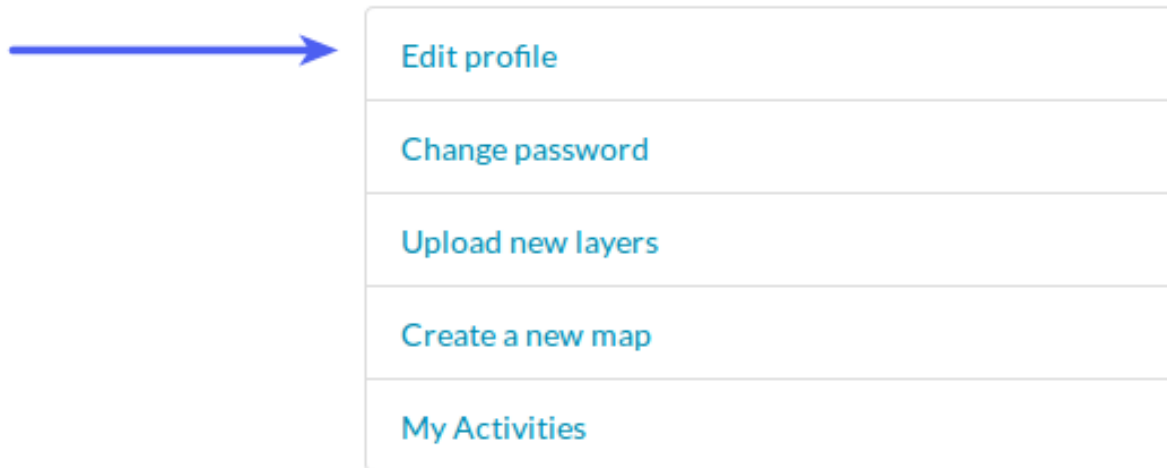


Fig. 6: Link to edit your profile

4. On this page, your personal information can be set, including your avatar. Enter some details in the *Profile* box as well as your city and country info.
5. When finished, click *Update profile*.
6. You will be returned to the main profile page. Now click *Account settings*.
7. On this page you can change your email address, time zone, and language. Your email should be populated already, but set the timezone to your current location.
8. When finished, click *Save*.

Setting notification preferences

By default GeoNode sends notifications to the users for events that the users could be subscribed such as a new layer uploaded or a new rate added to a map.

1. You can adjust your notification settings by clicking on your user name in the top right of the screen. A drop-down list will show. Click on *Notifications* to enter the Notifications Settings page.
2. Make sure to have a verified email address to which notices can be sent. If not, click on the proposed link to add one
3. Now check/uncheck the notification types you wish to receive or not receive. It is possible to be notified for the following events:
 - Layer Created
 - Layer Updated
 - Layer Deleted
 - Rating for Layer
 - Comment for Layer
 - Map Created
 - Map Updated

Edit Your Profile

First name
John

Last name
Smith

Email address
mail@example.com

Organization Name
Smith & Co.
name of the responsible organization

Profile
Manager of geospatial data for Smith & Co.
introduce yourself

Position Name

[Update profile](#)

Fig. 7: Editing your profile

address of the electronic mailbox of the responsible organization or individual

Keywords

A space or comma-separated list of keywords

[Update profile](#)

Fig. 8: Link to save your profile updates

Actions

[Edit profile information](#)

[Account Settings](#)

[Change password](#)

[Upload new layers](#)

[Create a new map](#)

Fig. 9: Link to edit your account settings

ACCOUNT

Email

Timezone

Language

Fig. 10: *Editing your account*

Notification Settings

Note: You do not have a verified email address to which notices can be sent. [Add one](#) now.

Notification Type	Email
User following you another user has started following you	<input checked="" type="checkbox"/>
Request to download a resource A request for downloading a resource was sent	<input checked="" type="checkbox"/>
Layer Created A Layer was created	<input checked="" type="checkbox"/>
Layer Updated A Layer was updated	<input checked="" type="checkbox"/>
Layer Deleted A Layer was deleted	<input checked="" type="checkbox"/>
Comment on Layer A layer was commented on	<input checked="" type="checkbox"/>
Rating for Layer A rating was given to a layer	<input checked="" type="checkbox"/>

- Map Deleted
- Rating for Map
- Comment for Map
- Document Created
- Document Updated
- Document Deleted
- Rating for Document
- Comment for Document
- User following you
- Request to download a resource

Viewing other user accounts

Now that your account is created, you can view other accounts on the system. Note that on the main profile page there are options for following (and blocking) other users.

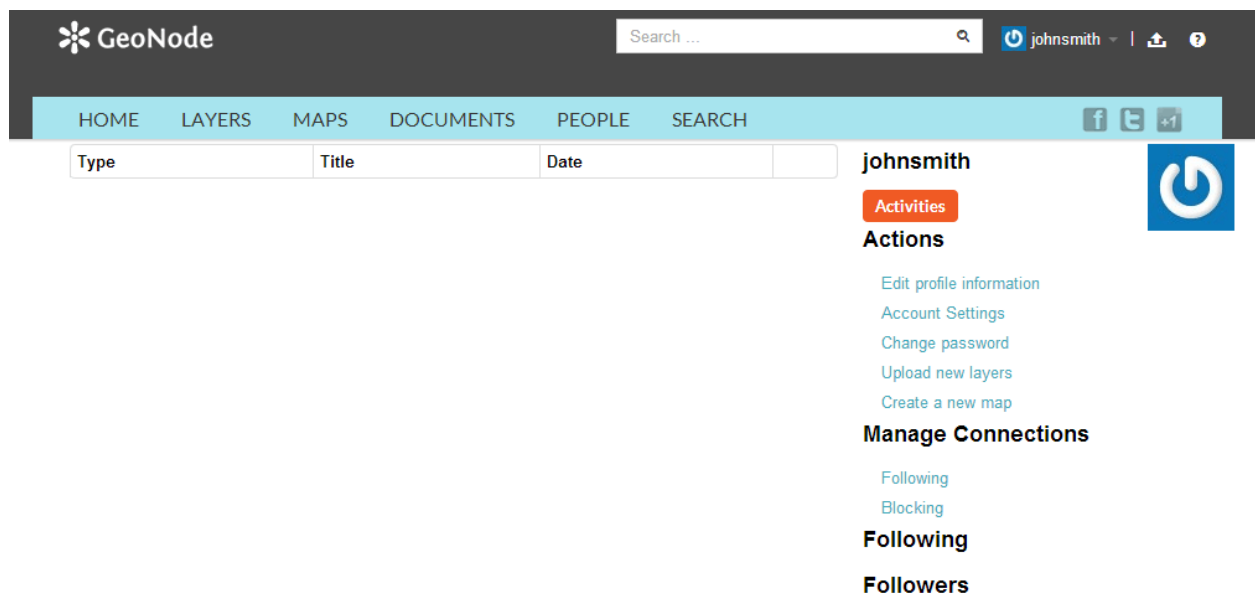


Fig. 11: *Profile page*

1. To see information about other users on the system, click the *People* link on the top toolbar. You will see a list of users registered on this system.
2. Click on the user name for a particular user. You will see the layers owned by this user.

1. You can also click *Activities* to see the activity feed.

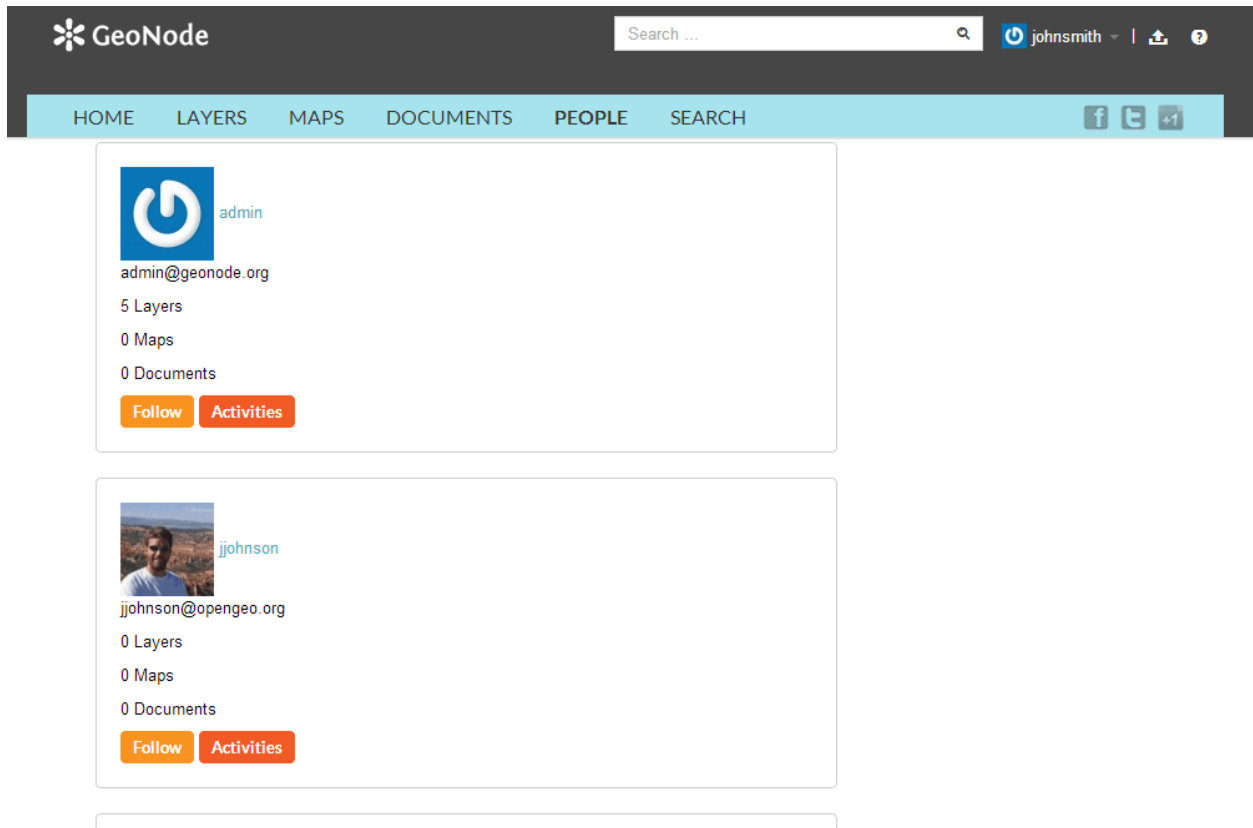


Fig. 12: *List of users*

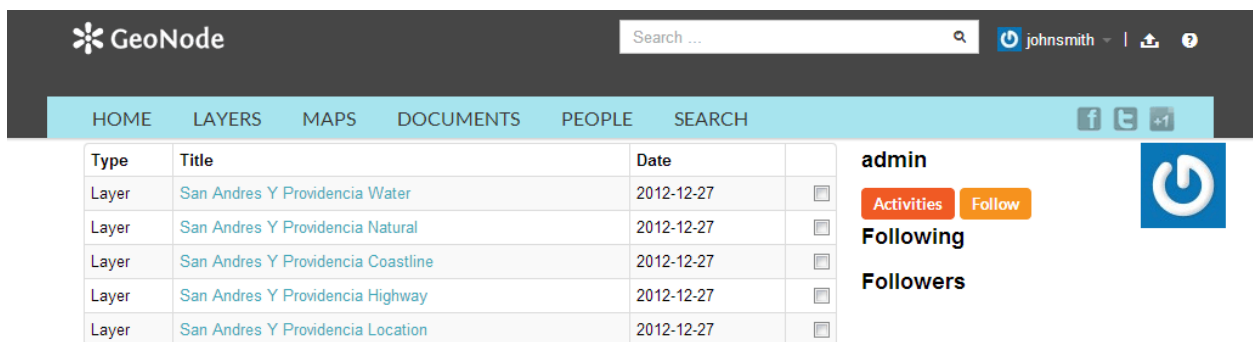


Fig. 13: *List of layers owned by a user*

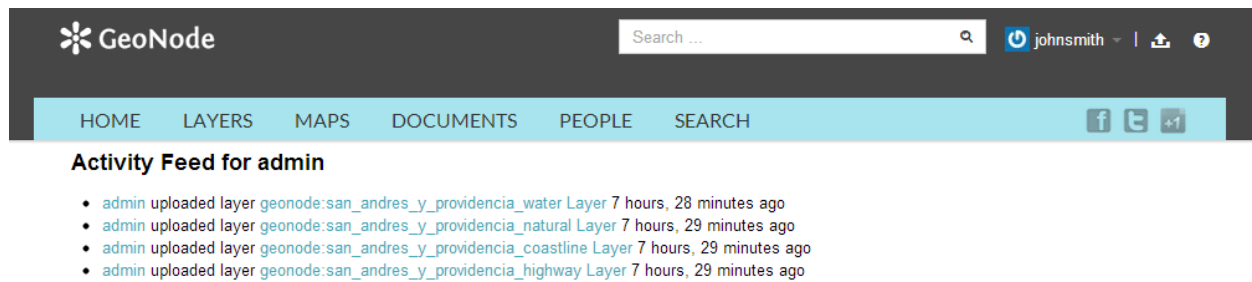


Fig. 14: List of users

2. If you are interested in keeping track of what this user does, go back to the previous page and click the *Follow* button.
3. A confirmation page will display. Click *Confirm*.

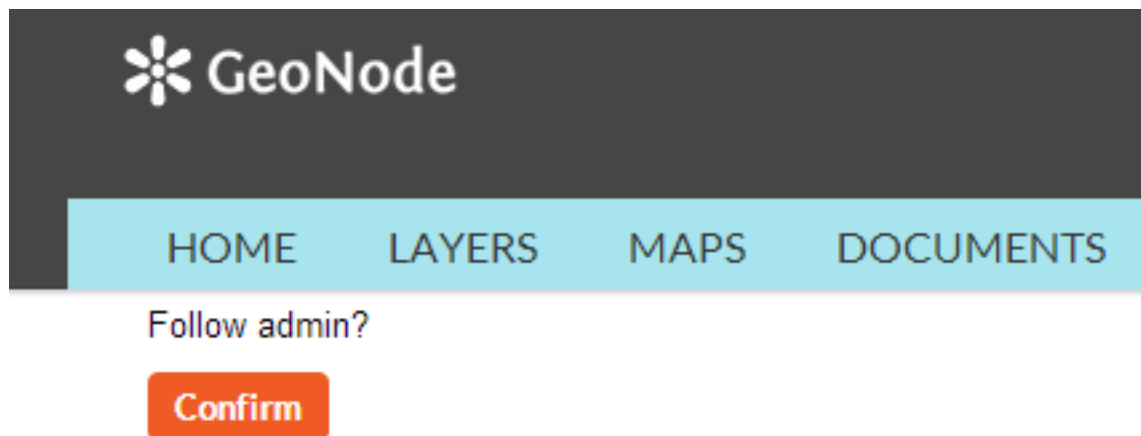


Fig. 15: Confirming following a user

4. You will now be following this user, and your profile page will note this.

3.2.1.3.2 Document Types

GeoNode welcome page shows a variety of information about the current GeoNode instance. At the top of the page is a toolbar showing quick links to document types: layers, maps and documents.

Data management tools built into GeoNode allow for integrated creation of data, documents, link to external documents, and map visualizations. Each dataset in the system can be shared publicly or restricted to allow access to only specific users. Social features like user profiles and commenting and rating systems allow for the development of communities around each platform to facilitate the use, management, and quality control of the data the GeoNode instance contains.

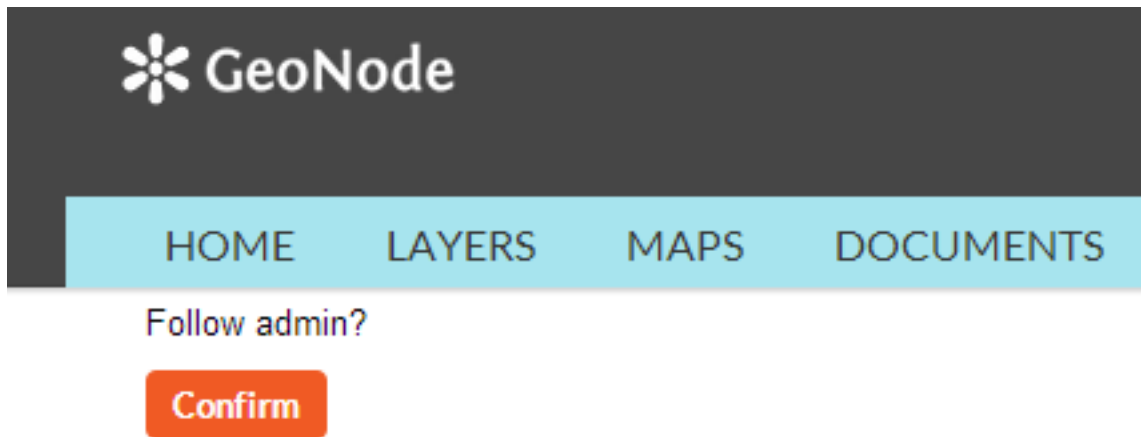


Fig. 16: *Success following a user*

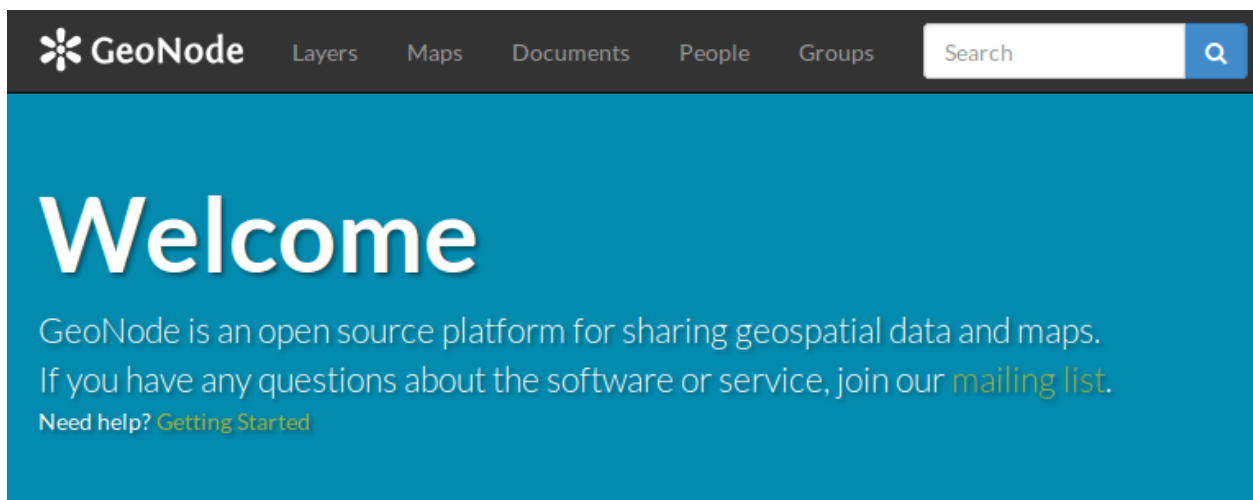


Fig. 17: *Document types in GeoNode welcome page*

Layers

Layers are a primary component of GeoNode.

Layers are publishable resources representing a raster or vector spatial data source. Layers also can be associated with metadata, ratings, and comments.

By clicking the Layers link you will get a list of all published layers. If logged in as an administrator, you will also see the unpublished layers in the same list.

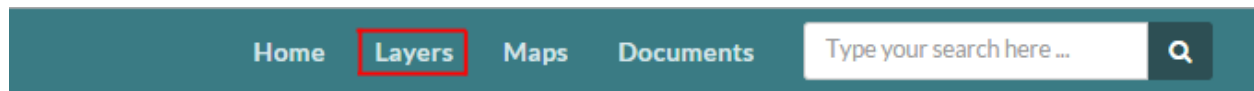


Fig. 18: *Layers in GeoNode toolbar*

GeoNode allows the user to upload vector (currently only Shapefiles) and raster data in their original projections using a web form.

Vector data is uploaded in ESRI Shapefile format and satellite imagery and other kinds of raster data are uploaded as GeoTIFFs.

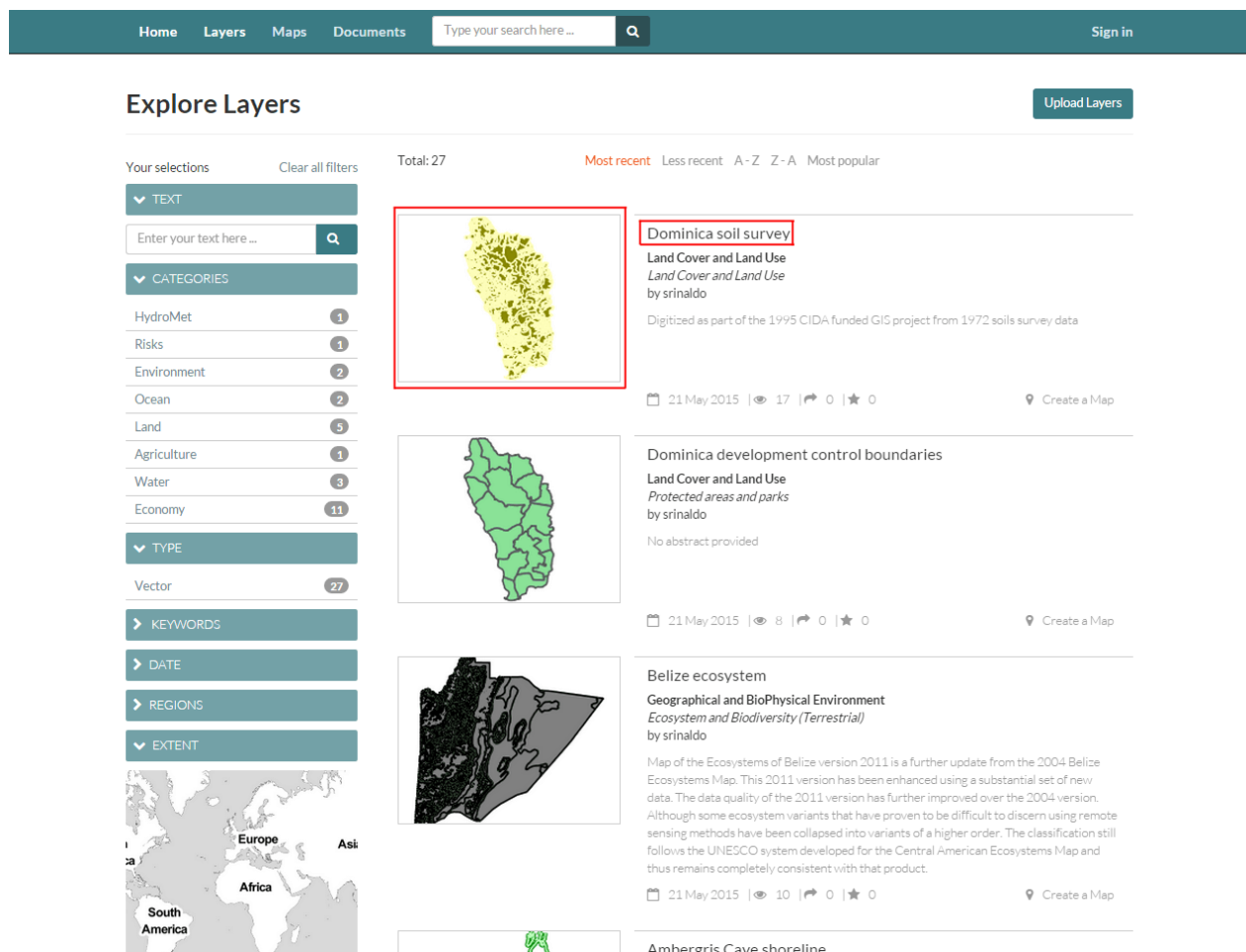


Fig. 19: *Layers list in GeoNode*

Maps

Maps are a primary component of GeoNode.

Maps are comprised of various layers and their styles. Layers can be both local layers in GeoNode as well as remote layers either served from other WMS servers or by web service layers such as Google or MapQuest.

GeoNode maps also contain other information such as map zoom and extent, layer ordering, and style.

By clicking the Map link you will get a list of all published maps.

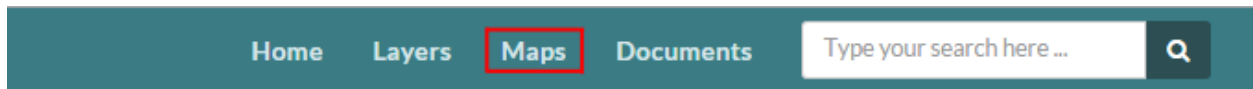


Fig. 20: Maps in GeoNode toolbar

This toolbar allows you create a map based on the uploaded layers combine them with some existing layers and a remote web service layer, and then share the resulting map for public viewing. Once the data has been uploaded, GeoNode lets the user search for it geographically or via keywords and create maps. All the layers are automatically reprojected to web mercator for maps display, making it possible to use different popular base layers, like Open Street Map, Google Satellite or Bing layers.

Documents

As for the layers and maps GeoNode allows to publish tabular and text data manage metadata and associated documents.

By clicking the Documents link you will be brought to the Documents menu where a new subtoolbar can be seen.

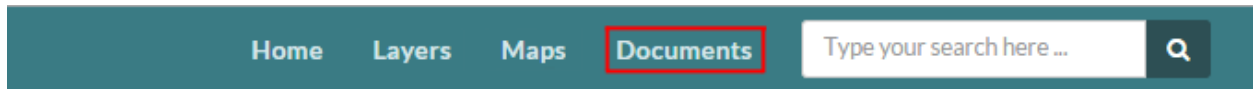


Fig. 21: Documents in GeoNode toolbar

Through the document detailed page is possible to view, download and manage a document.

3.2.1.3.3 Searching

In GeoNode welcome page, click the Search button to bring up the Search page.

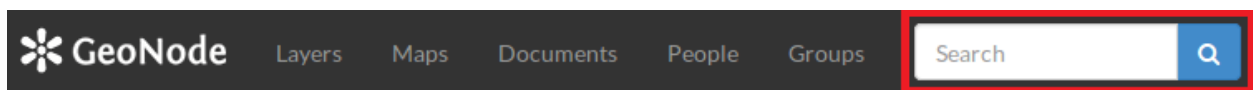


Fig. 22: Search tool in GeoNode welcome page

This page contains a wealth of options for customizing a search for various information on GeoNode. This search form allows for much more fine-tuned searches than the simple search box is available at the top of every page.

It is possible to search data by Text, Categories, Type, Keywords, Date, Regions or Extent.

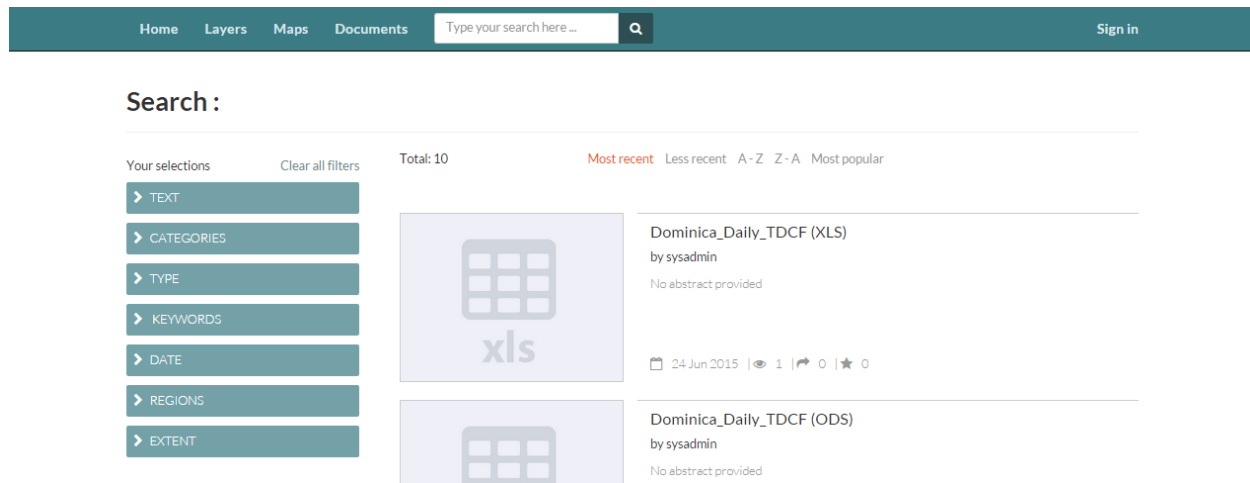


Fig. 23: Search page

3.2.1.3.4 Managing layers

After user accounts, the next primary component of GeoNode is the **layer**. Layers are a published resource representing a raster or vector spatial data source. Layers also can be associated with metadata, ratings, and comments.

In this section, you will learn how to create a new layer by uploading a local data set, add layer info, change the style of the layer, and share the results.

Uploading a layer

Now that we have taken a tour of GeoNode and viewed existing layers, the next step is to upload our own.

In your data pack is a directory called `data`. Inside that directory is a shapefile called `san_andres_y_providencia_administrative.shp`. This is a data set containing administrative boundaries for the San Andres Province. This will be the first layer that we will upload to GeoNode.

1. Navigate to the GeoNode welcome page.
2. Click the *Layers* link on the top toolbar. This will bring up the Layers menu.

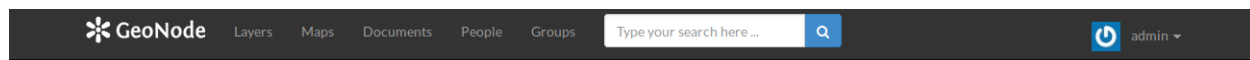


Fig. 24: Main toolbar for GeoNode

3. Click *Upload Layers* in the Layers toolbar. This will bring up the upload form
4. Fill out the form.

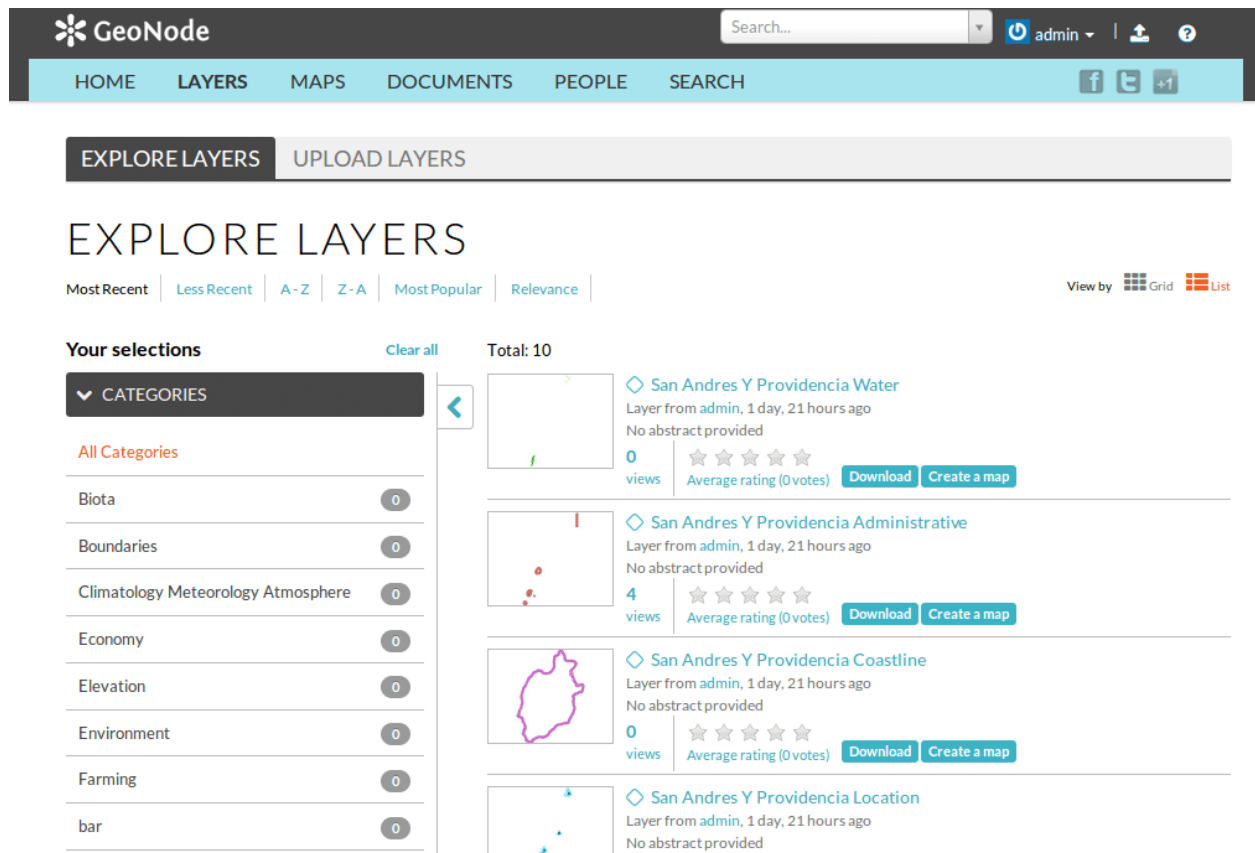


Fig. 25: Layers menu



Fig. 26: Layers toolbar

Fig. 27: Upload Layers form

- Click on the *Browse...* button. This will bring up a local file dialog. Navigate to your data folder and select all of the four files composing the shapefile (`san_andres_y_providencia_administrative.shp`, `san_andres_y_providencia_administrative.dbf`, `san_andres_y_providencia_administrative.shx`, `san_andres_y_providencia_administrative.prj`). Alternatively you could drag and drop the four files in the *Drop files here* area.
 - The upload form should appear like this now:
5. GeoNode has the ability to restrict who can view, edit, and manage layers. On the right side of the page, under *Who can view and download this data?*, select *Any registered user*. This will ensure that anonymous view access is disabled.
 6. In the same area, under *Who can edit this data?*, select the *Only the following users or groups* option and type your username. This will ensure that only you are able to edit the data in the layer.
 7. Click *Upload* to upload the data and create a layer. A dialog will display showing the progress of the upload.
 8. Your layer has been uploaded to GeoNode. Now you will be able to access to the its info page (clicking on the *Layer Info* button), access to its metadata edit form (clicking on the *Edit Metadata* button) or to manage the styles for it (clicking on the *Manage Styles* button).

Layer information

After upload, another form will displaying, containing metadata about the layer. Change any information as desired, and then click *Update* at the very bottom of the form.

After the update, the layer will display in a preview window.

This page contains lots of options for managing this layer. Let's look at a few of them:

UPLOAD LAYERS

Drop files here

or select them one by one:

Choose Files No file chosen

FILES TO BE UPLOADED

SAN_ANDRES_Y_PROVIDENCIA_ADMINISTRATIVE ESRI SHAPEFILE

- san_andres_y_providencia_administrative.dbf [Remove](#)
- san_andres_y_providencia_administrative.prj [Remove](#)
- san_andres_y_providencia_administrative.shp [Remove](#)
- san_andres_y_providencia_administrative.shx [Remove](#)

Select the charset or leave default

UTF-8/Unicode ▼

[Clear](#) [Upload files](#)

PERMISSIONS

Who can view and download this data?

- ☒ Anyone ☐ Any registered user
☐ Only users who can edit

Who can edit this data?

- ☒ Any registered user
☐ Only the following users or groups:

[Choose one or more users...](#)

Who can manage and edit this data?

[Choose one or more users...](#)

Fig. 28: Files ready for upload

PERMISSIONS

Who can view and download this data?

- ☐ Anyone ☒ Any registered user
☐ Only users who can edit

Who can edit this data?

- ☐ Any registered user
☒ Only the following users or groups:

× admin

Who can manage and edit this data?

Choose one or more users...

Fig. 29: Permissions for new layer

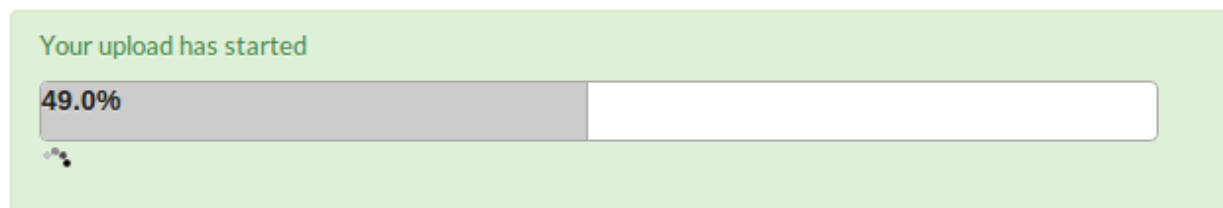
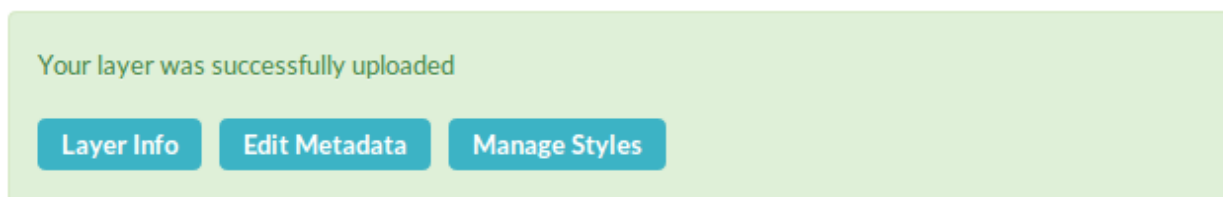
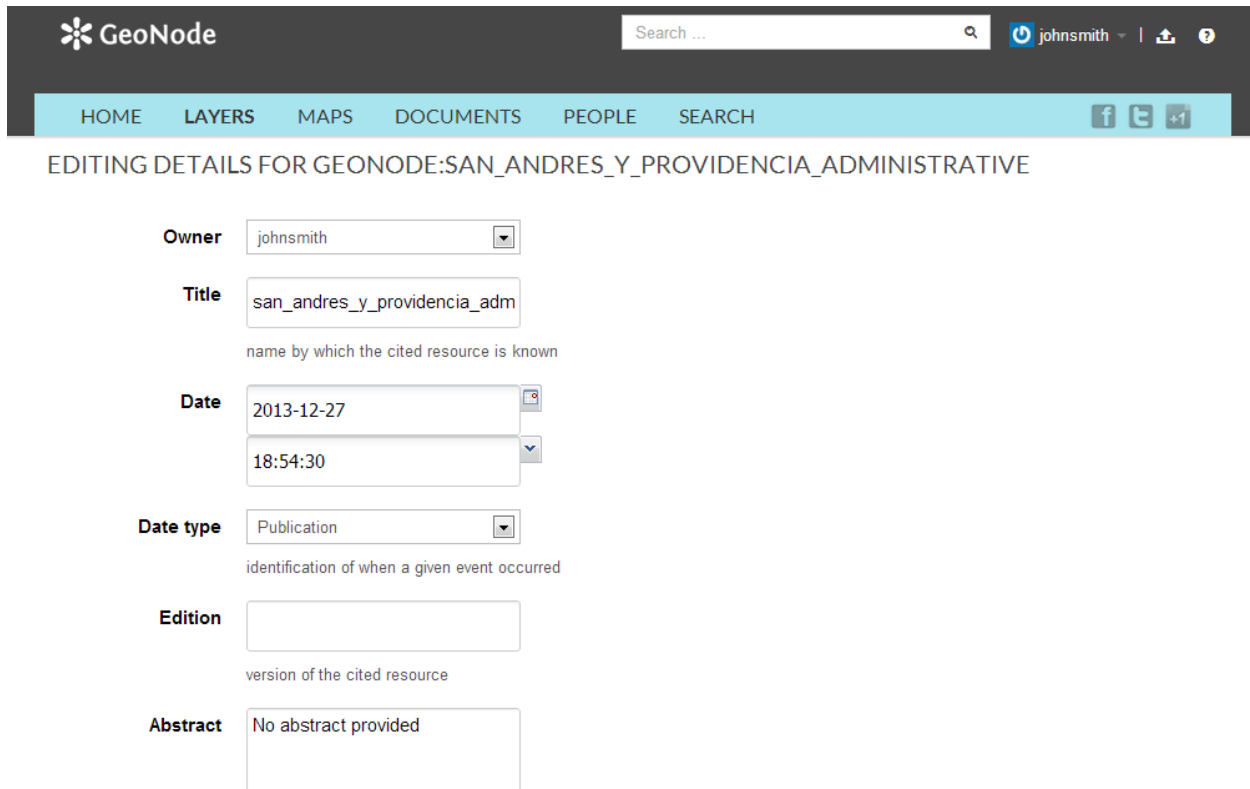


Fig. 30: Upload in progress





The screenshot shows the 'EDITING DETAILS' page for a GeoNode layer. The header includes the GeoNode logo, a search bar, and user information for 'johnsmith'. The navigation bar contains links for HOME, LAYERS, MAPS, DOCUMENTS, PEOPLE, and SEARCH. The main content area is titled 'EDITING DETAILS FOR GEONODE:SAN_ANDRES_Y_PROVIDENCIA_ADMINISTRATIVE'. It contains several form fields: 'Owner' (johnsmith), 'Title' (san_andres_y_providencia_adm) with a subtitle 'name by which the cited resource is known', 'Date' (2013-12-27 18:54:30), 'Date type' (Publication) with a subtitle 'identification of when a given event occurred', 'Edition' (empty) with a subtitle 'version of the cited resource', and 'Abstract' (No abstract provided).

Fig. 31: *Layer metadata*

Downloads

At the top of the page there are two buttons titled *Download Layer* and *Download Metadata*. These buttons provide access to the ability to extract geospatial data and metadata from within GeoNode. In this way, GeoNode allows for two way data and metadata access; one can import as well as export data.

Data

1. Click the *Download Layer* button. You will see a list of options of the supported export formats.
1. Click the option for *Zipped Shapefile*.
2. GeoNode will process the request and bring up a Save As dialog. Save this file to your computer, and note how it is the same content as was uploaded.

Metadata

1. Click the *Download Metadata* button. You will see a list of options of the supported export formats.
1. Click the option for *DUBLIN CORE*.
2. GeoNode will process the request and display XML metadata. Try clicking various metadata formats, and note how it is the same metadata content in various formats compatible with metadata and GIS packages.

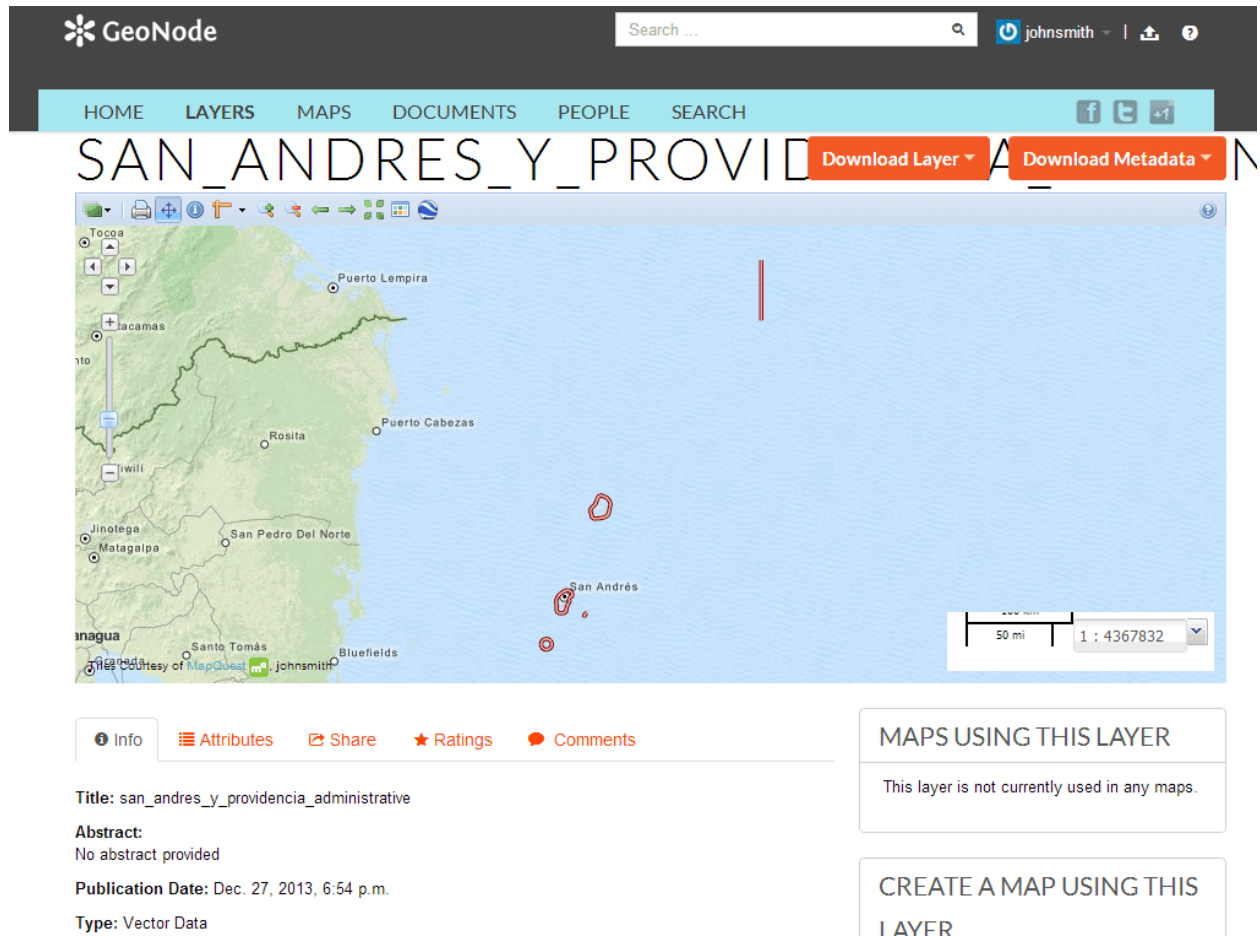


Fig. 32: Layer preview

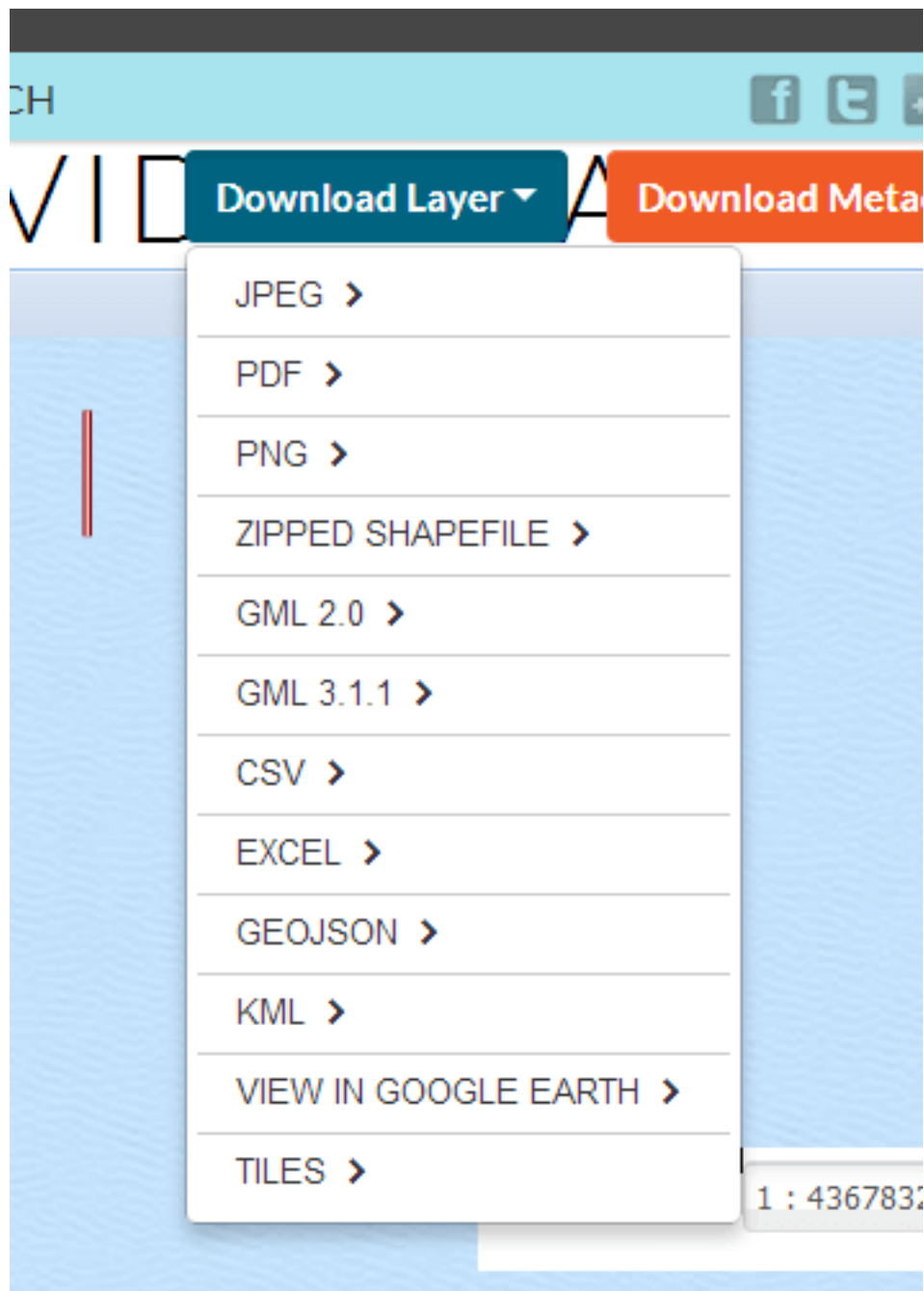


Fig. 33: Available export formats

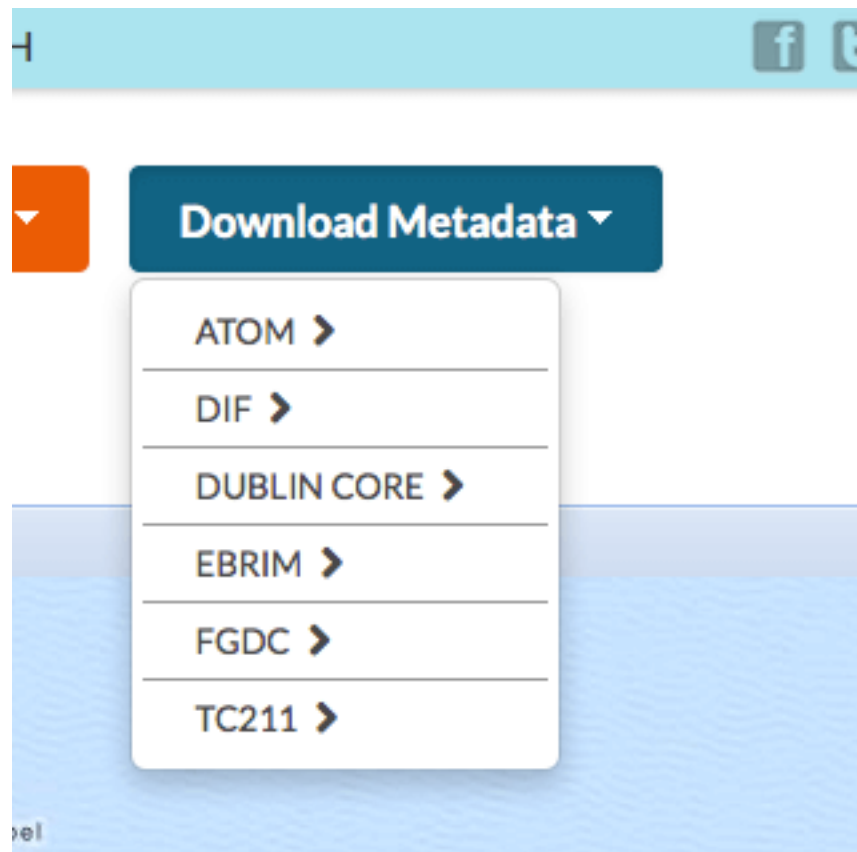


Fig. 34: Available metadata export formats

Layer Detail Tabs

1. Scroll down the page toward the bottom. Five tabs are available: *Info*, *Attributes*, *Share*, *Ratings*, and *Comments*. The info tab is already highlighted, and presents basic information about the layer, of the kind that was seen on the layer list page.

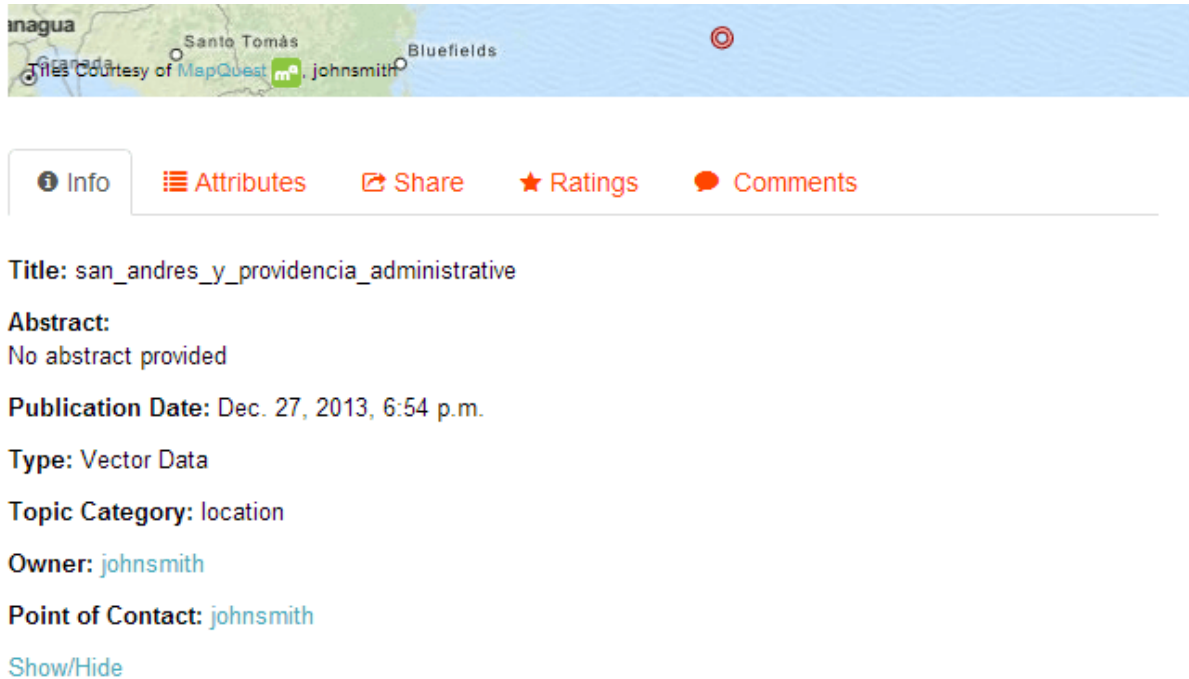
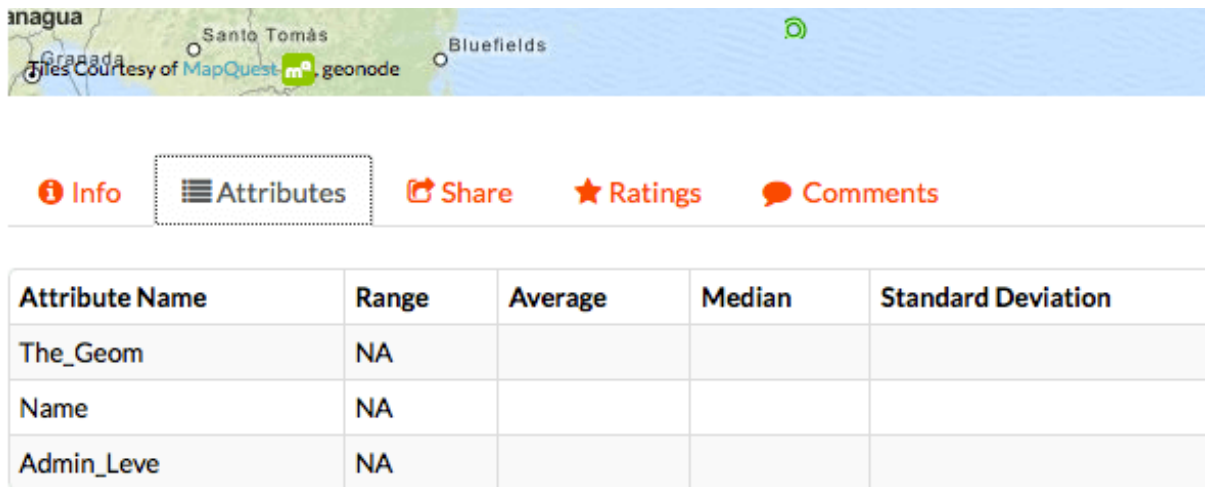


Fig. 35: *Layer Info tab*

2. Click the *Attributes* tab. This lists the attributes of the layer, including statistics (range, average, median and standard deviation). Layer attribute statistics are made available only for numeric attributes. As we can see, this layer's attributes are not numeric, so no statistics are calculated.
3. Click the *Ratings* tab. This tab allows you (and others viewing this page) to rate this layer. Ratings can be based on quality, accuracy, or any other metric. Click on the appropriate star to rate this layer.
4. Click the *Comments* tab. This tab allows you to leave a comment for other viewing this layer.
5. Click the *Add Comment* button and enter a comment.
6. When finished, click *Submit Comments*

Sharing layers

GeoNode has the ability to restrict or allow other users to access a layer and share on social media.

Fig. 36: *Attributes tab*Fig. 37: *Layer Ratings tab*

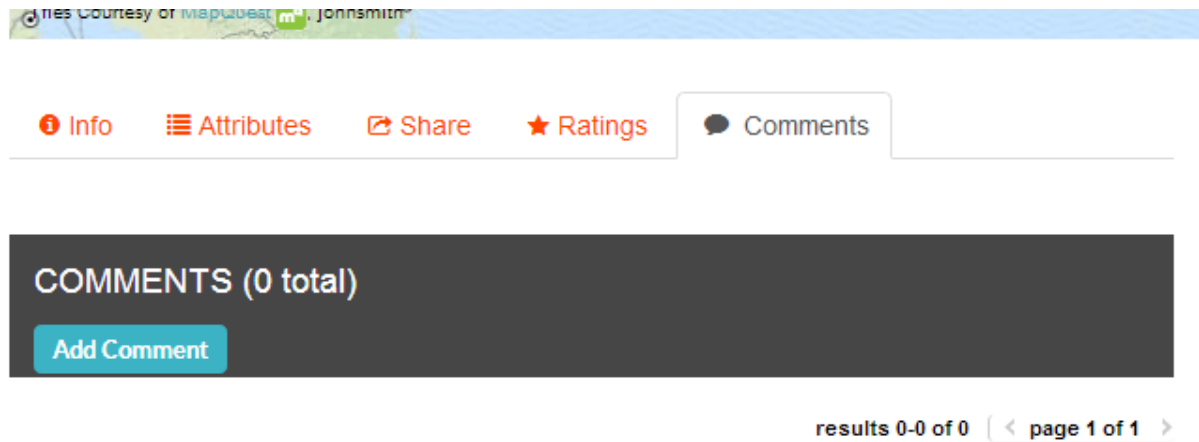


Fig. 38: *Layer Comments tab*

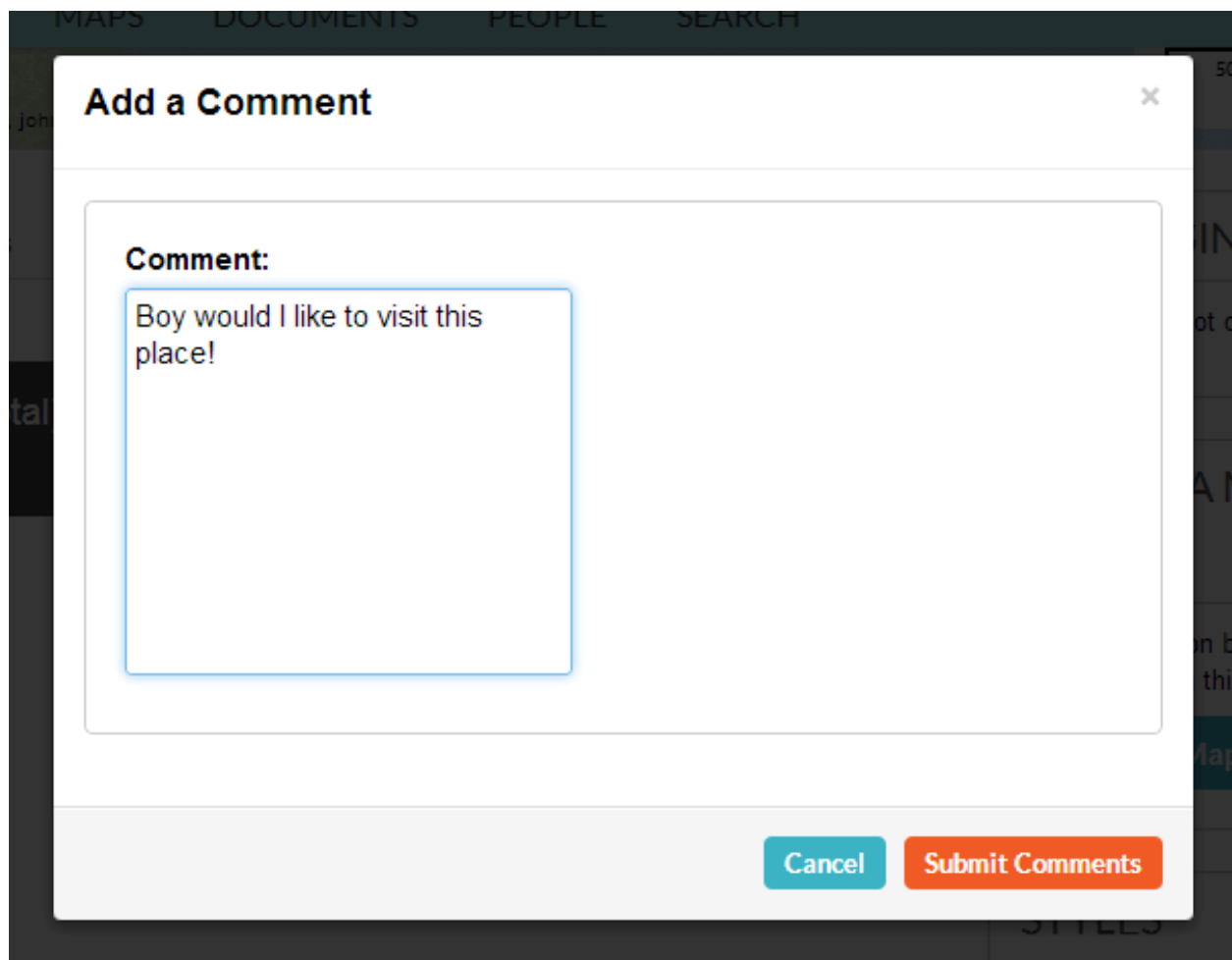


Fig. 39: *Adding a new comment*

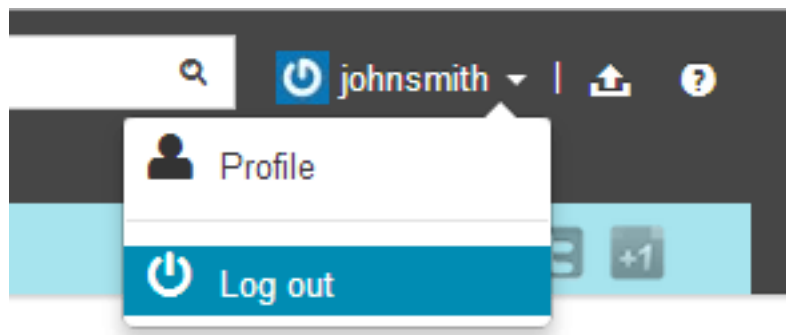
Fig. 40: *New comment posted*

Anonymous access

1. Go to the layer preview of the first layer uploaded, and copy the URL to that preview page.

Note: The URL should be something like: `http://GEONODE/layers/geonode:san_andres_y_providencia_administrative`

2. Now log out of GeoNode by clicking on your profile name and selecting *Log out*.

Fig. 41: *Log out*

3. When asked for confirmation, click the *Log out* button.

LOG OUT

Are you sure you want to log out?

Log out

Fig. 42: *Confirming log out*

4. Now paste the URL copied about into your browser address bar and navigate to that location.

5. You will be redirected to the Log In form. This is because when this layer was first uploaded, we set the view properties to be any registered user. Once logged out, we are no longer a registered user and so are not able to see or interact with the layer, unless we log in GeoNode again.

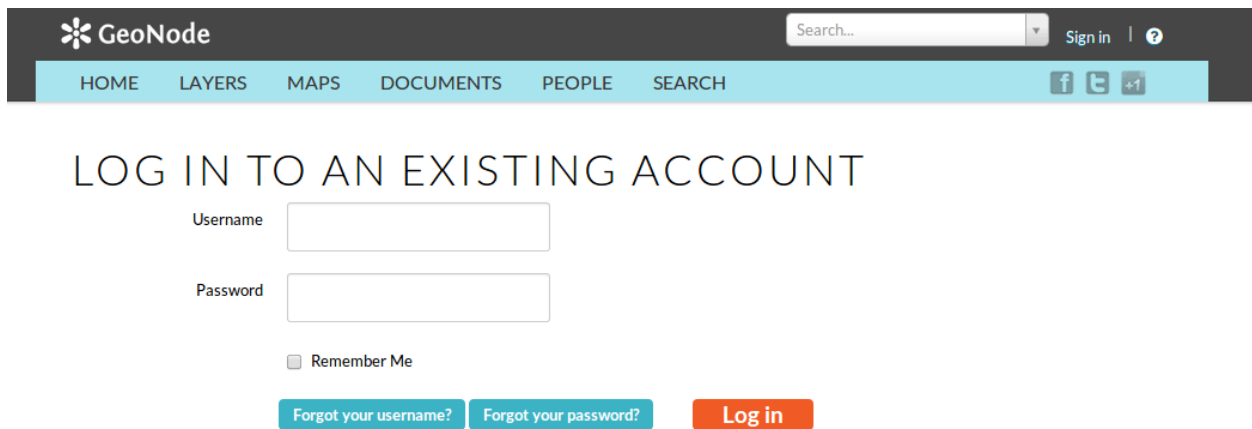


Fig. 43: *Unable to view this protected layer*

6. To stop this process from happening, you need to ensure that your permissions are set so *anyone* can view the layer for others to see it on social networks.
1. This is done by selecting *anyone* in the layer permissions tab, be aware this now means your layer is public!

Sharing with social media

1. On the taskbar below your username and profile picture there are three links to social media services, Twitter, Google Plus and Facebook.
2. Upon clicking on these icons you will be taken through the application's process for posting to the social network. Ensure the permissions are set so *anyone* can view the layer if you want unauthenticated to be able to access it.

Adding more layers

We've uploaded one layer so far. There is one more layer in the data directory associated with this workshop called `san_andres_y_providencia_poi.shp`.

1. Upload this layer, referring to the directions on [uploading a layer](#). As a difference, leave the permissions set to their default values.

Creating empty layers

In GeoNode it is possible to create empty layers, that can be populated with features at a later stage using the mapping client editing tools.

This is possible using the *createlayer* application, which can be enabled if GeoNode is installed with PostGIS.

Once the application is enabled in GeoNode you will be able to create an empty layer by browsing to "Data > Create Layer". You will see a form like this:

Permissions

Who can view and download this data?

☒ Anyone ☐ Any registered user ☐ Only users who can edit

Who can edit this data?

☐ Any registered user

☒ Only the following users or groups:

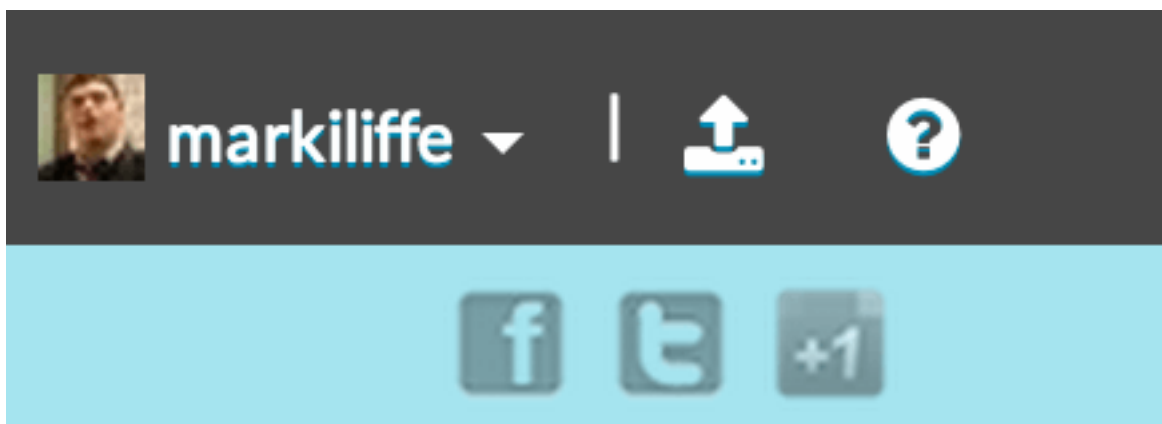
✕ barbara

Who can manage and edit this data?

✕ barbara

Cancel

Apply Changes



EXPLORE LAYERS

UPLOAD LAYERS

UPLOAD LAYERS

Drop files here

or select them one by one:

Choose Files No file chosen

FILES TO BE UPLOADED

SAN_ANDRES_Y_PROVIDENCIA_POI

ESRI SHAPEFILE

- san_andres_y_providencia_poi.dbf [Remove](#)
- san_andres_y_providencia_poi.prj [Remove](#)
- san_andres_y_providencia_poi.shp [Remove](#)
- san_andres_y_providencia_poi.shx [Remove](#)

Select the charset or leave default
UTF-8/Unicode

PERMISSIONS

Who can view and download this data?

☒ Anyone ☐ Any registered user
☐ Only users who can edit

Who can edit this data?

☒ Any registered user
☐ Only the following users or groups:

Choose one or more users...

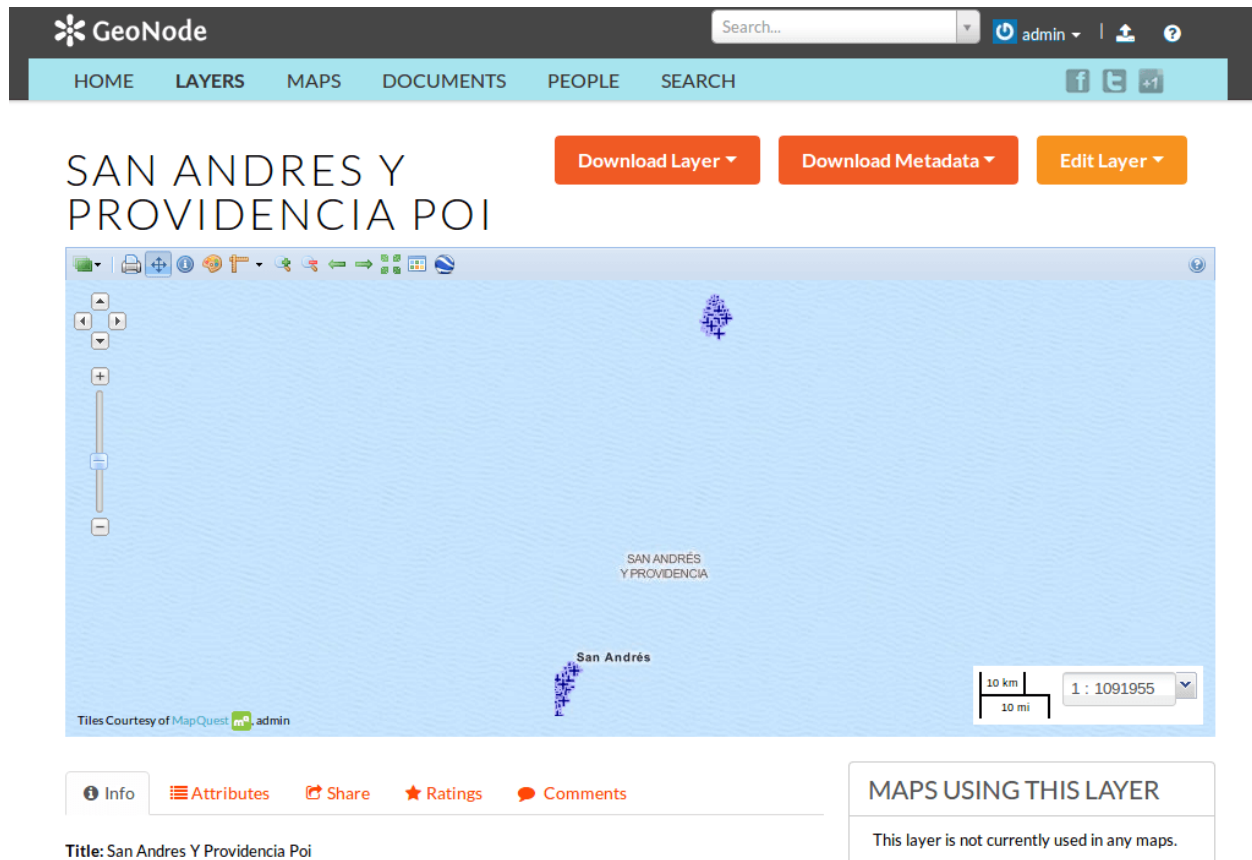
Who can manage and edit this data?

Choose one or more users...

Fig. 44: *Uploading the layer*

232

Chapter 3. Table of contents

Fig. 45: *Finished upload*

The screenshot shows the 'Create an empty layer' form in GeoNode. The form has the following fields and sections:

- Layer name:** A text input field containing 'empty-layer'.
- Layer title:** A text input field containing 'My empty layer'.
- Geometry type:** A dropdown menu set to 'Polygons'.
- Add Attribute:** A button to add a new attribute.
- Attribute:** A text input field containing 'a-string-field'.
- String:** A dropdown menu set to 'String'.
- Remove:** A button to remove the attribute.
- Create:** A red button to create the layer.
- Permissions:** A section with several options:
 - Who can view it?** A dropdown menu set to 'Anyone'.
 - The following users:** A text input field containing 'Choose users...'.
 - The following groups:** A text input field containing 'Choose groups...'.
 - Who can download it?** A dropdown menu.
 - Who can change metadata for it?** A dropdown menu.
 - Who can edit data for this layer?** A dropdown menu.
 - Who can edit styles for this layer?** A dropdown menu.
 - Who can manage it? (update, delete, change permissions, publish/unpublish it)** A dropdown menu.

Fill the form as needed:

- give a layer name
- give a layer title
- assign a geometry for the layer (Points, Lines, Polygons)
- add as many attributes as needed. For each attribute provide a name and a type. Type can be string, integer, float and date
- assign permissions as needed

Now by clicking the “Create” button your new empty layer should be created.

3.2.1.3.5 Edit Layer Style

Editing a style can only be performed by users with the correct permissions.

1. In the Explore Layer page, choose a Layer that you want to edit by clicking on the name of the layer or in the preview window.
2. In the Edit Layers page, click the Edit Layer button.
3. In the Edit Layer window, click the Edit button under Style icon. In this interface is it possible to change the style of layers. GeoNode allows editing of layer styles graphically, without the need to resort to programming or requiring a technical background.

In the following example, the layer has one style and one rule in that style. Click Edit in Styles menu, then change the Title and Abstract of the selected Style.

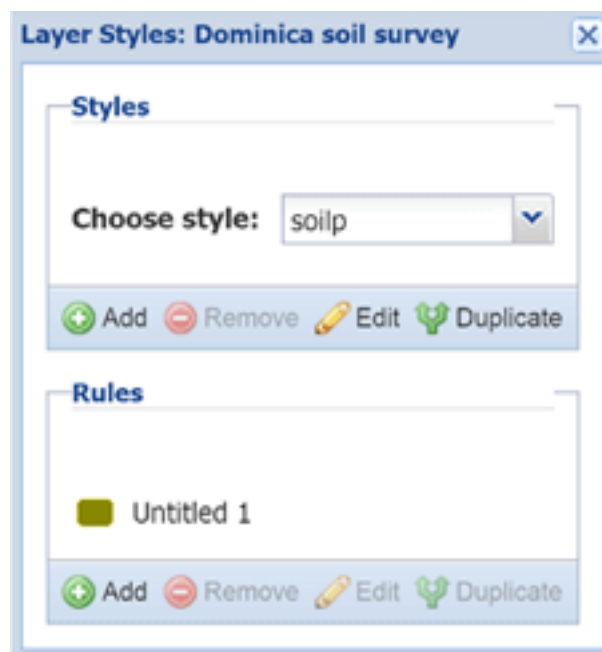


Fig. 46: *Layer Styles window*

Click the Rule (Untitled 1) to select it, and then click on *Edit* below it. Edit the style by choosing the *Basic* tab to edit symbology of layers, the *Labels* tab to add and manage labels, and the *Advanced*

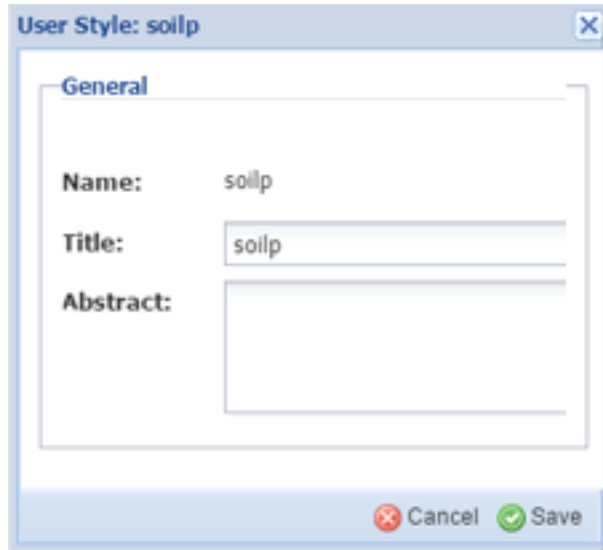


Fig. 47: User Styles window

tab to manage styles by scale and condition. When done, click *Save*, then click on the word *Layers* to return to the layer list.

4. In the Edit Layer window, click the *Manage* button under the Style icon. The Manage Styles function allows assigning an available style to selected layers.
5. If you import a Style created by other GIS software (or edited directly in GeoServer), make sure that your .sld file already has the <Title></Title> and <Abstract></Abstract> nodes under the <UserStyle> node. Otherwise the style will be shown as “None” in GeoNode.

3.2.1.3.6 Managing maps

The next primary component of GeoNode is the **map**. Maps are comprised of various layers and their styles. Layers can be both local layers in GeoNode as well as remote layers either served from other WMS servers or by web service layers such as Google or MapQuest.

GeoNode maps also contain other information such as map zoom and extent, layer ordering, and style.

In this section, we’ll create a map based on the layers uploaded in the previous section, combine them with some existing layers and a remote web service layer, and then share the resulting map for public viewing.

Creating a map

Adding layers

1. Click the *Maps* link on the top toolbar. This will bring up the list of maps.
2. Currently, there aren’t any maps here, so let’s add one. Click the *Create a New Map* button.
3. A map composition interface will display.

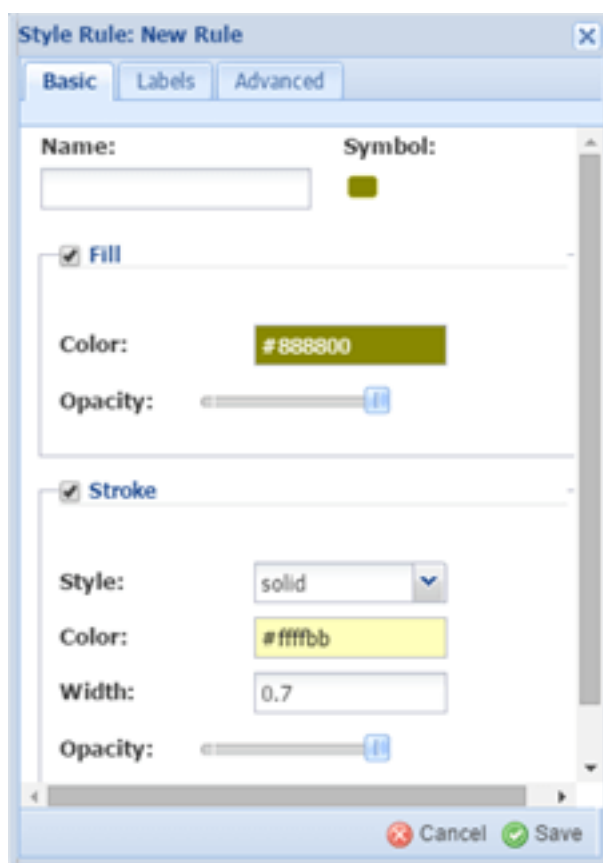


Fig. 48: *Basic Style Rule window*

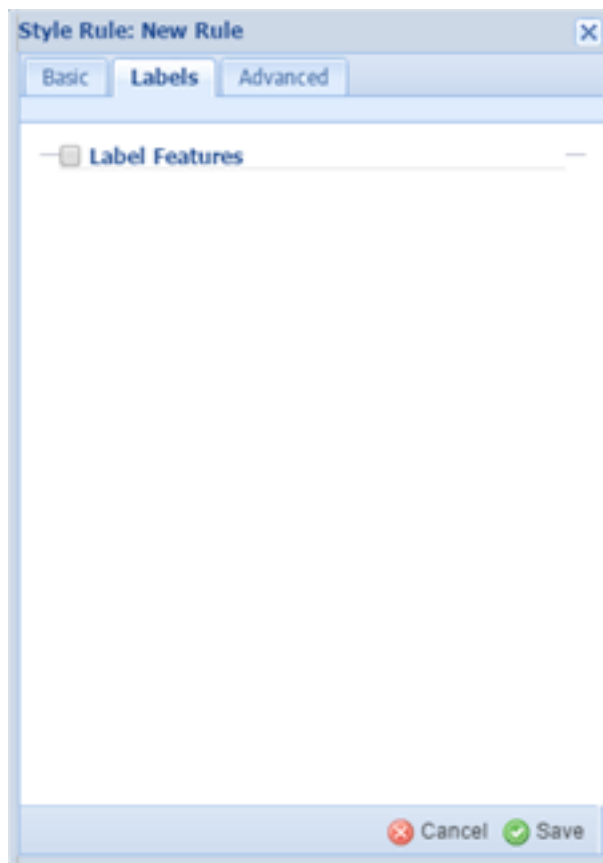


Fig. 49: *Labels Style Rule* window

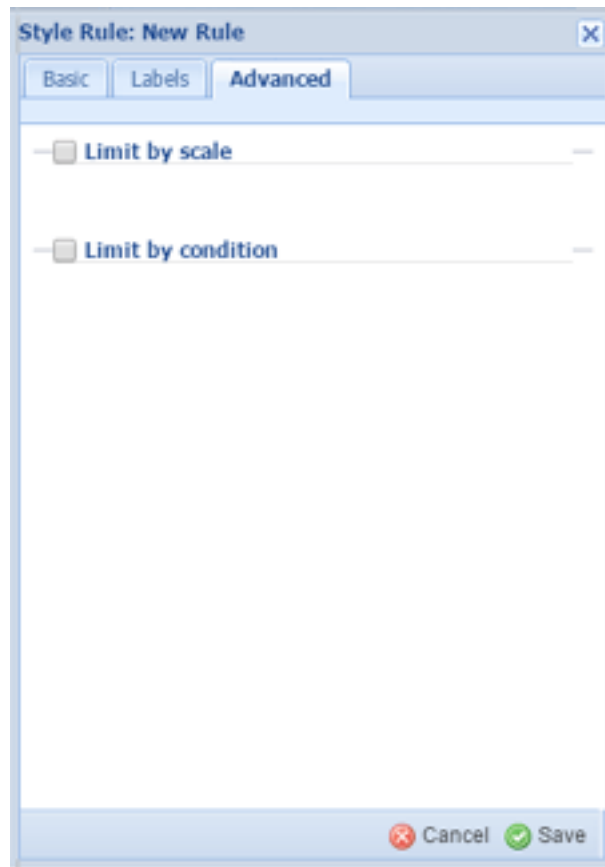


Fig. 50: Advanced Style Rule window

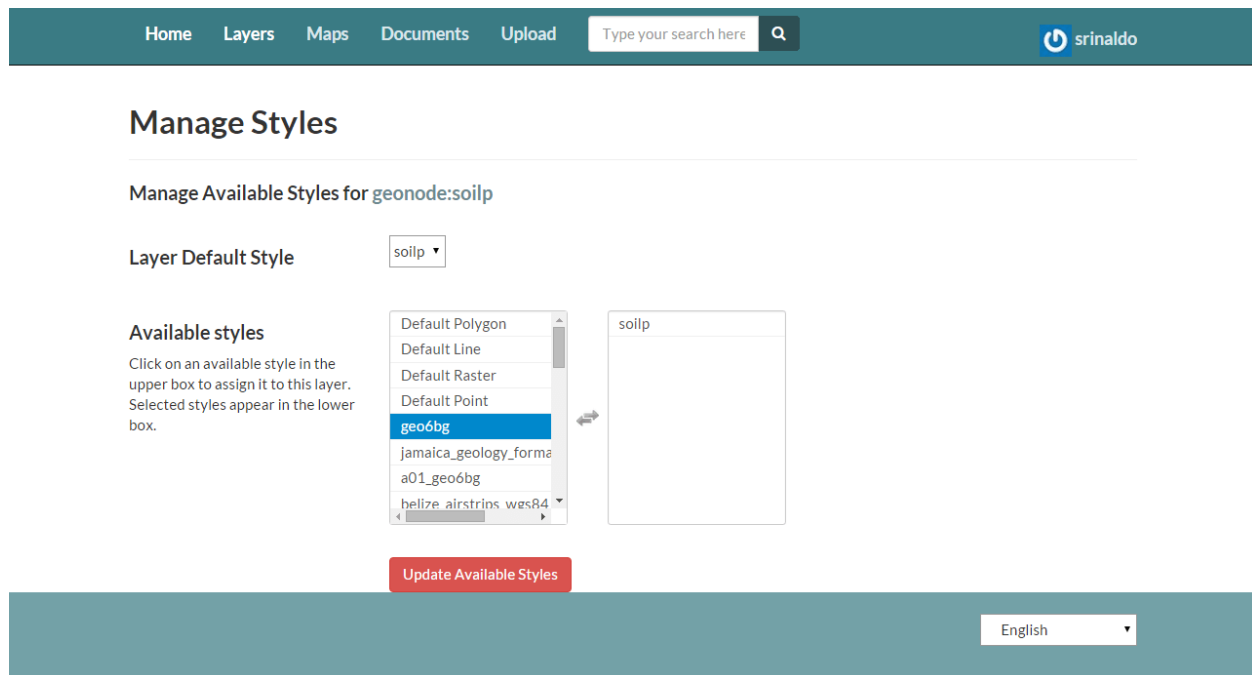
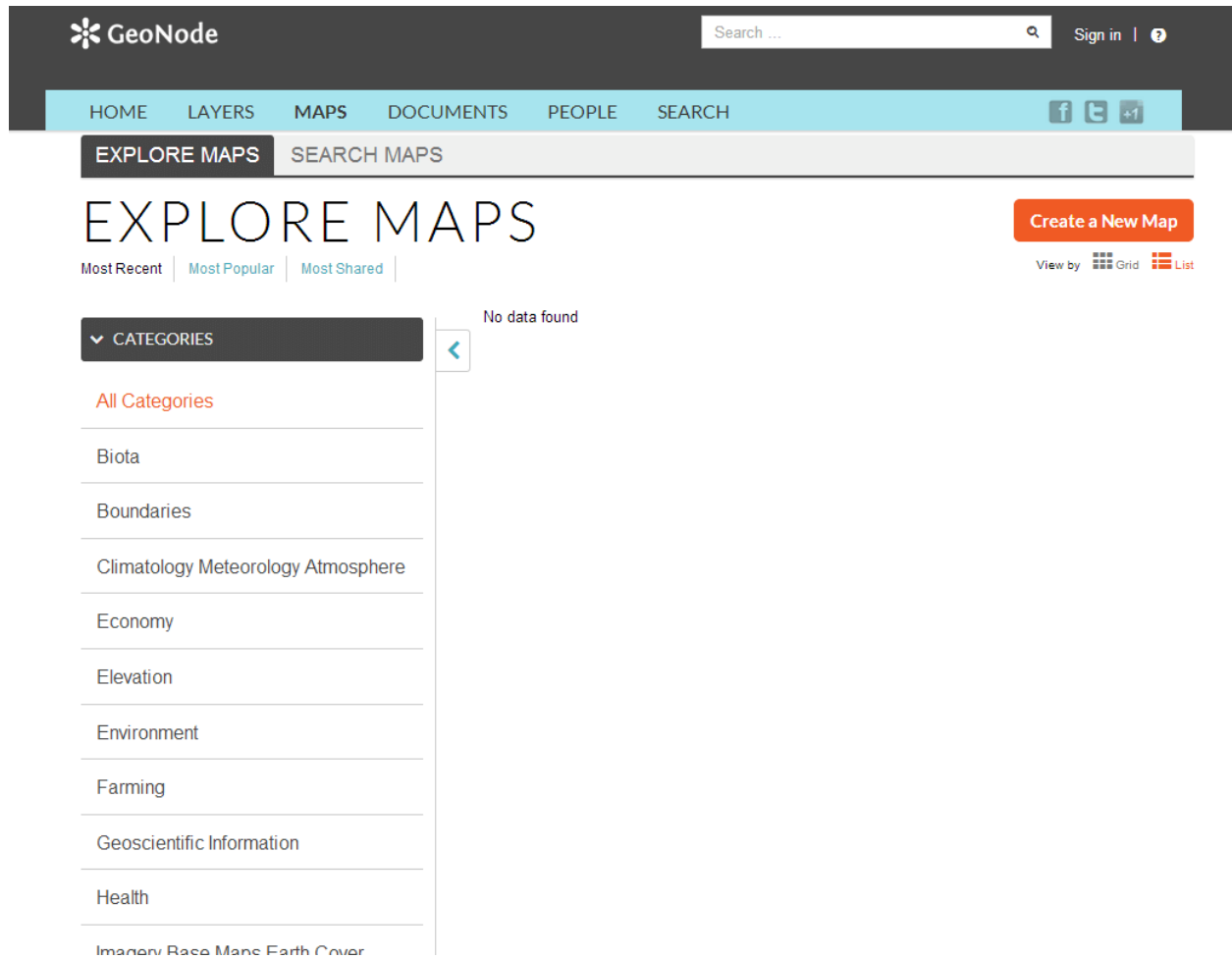


Fig. 51: Manage Layer Styles

Fig. 52: *Maps page*

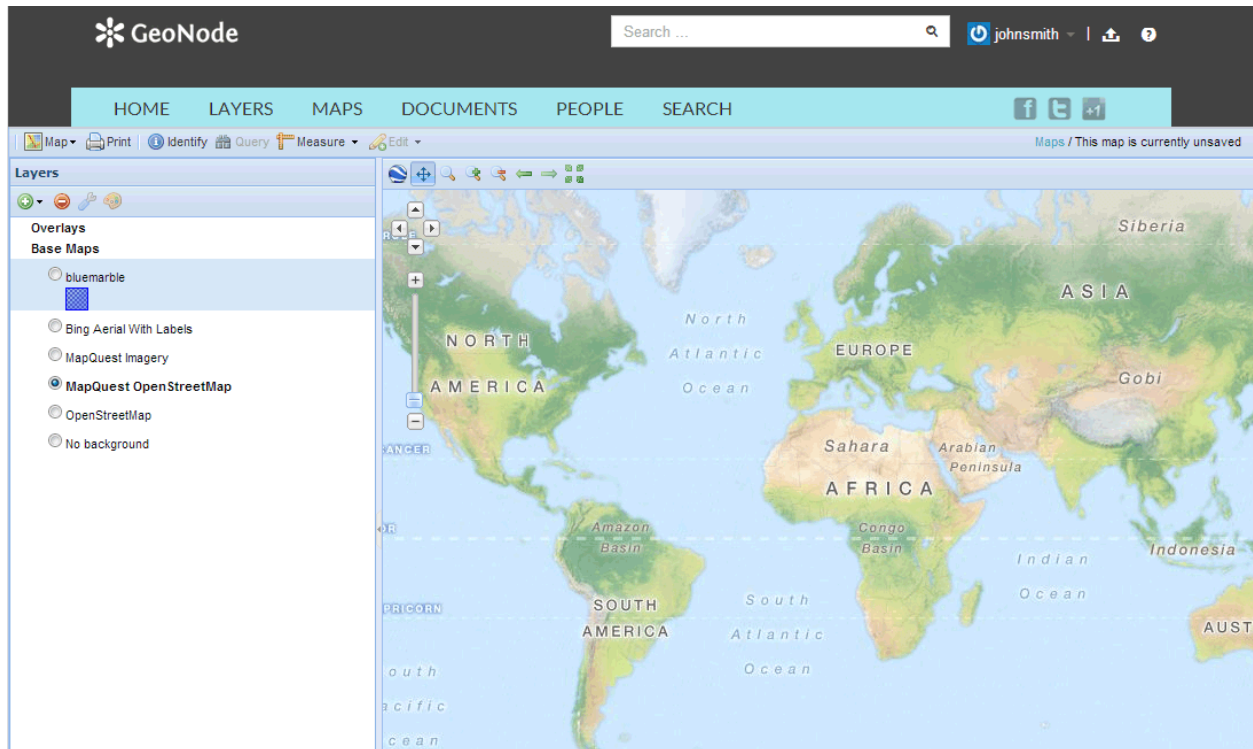


Fig. 53: Create maps interface

In this interface there is a toolbar, layer list, and map window. The map window contains the MapQuest OpenStreetMap layer by default. There are other service layers available here as well: Blue Marble, Bing Aerial With Labels, MapQuest, and OpenStreetMap.

4. Click on the New Layers button and select Add Layers.

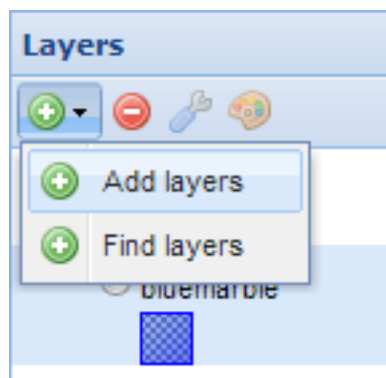


Fig. 54: Add layers link

5. Select all of the San Andreas layers by clicking the top entry and Shift-clicking the bottom one. Click *Add Layers* to add them all to the map.

Note: This selection includes not only the two layers uploaded in the previous section, but also the layers that were already hosted on GeoNode at the beginning of the workshop.

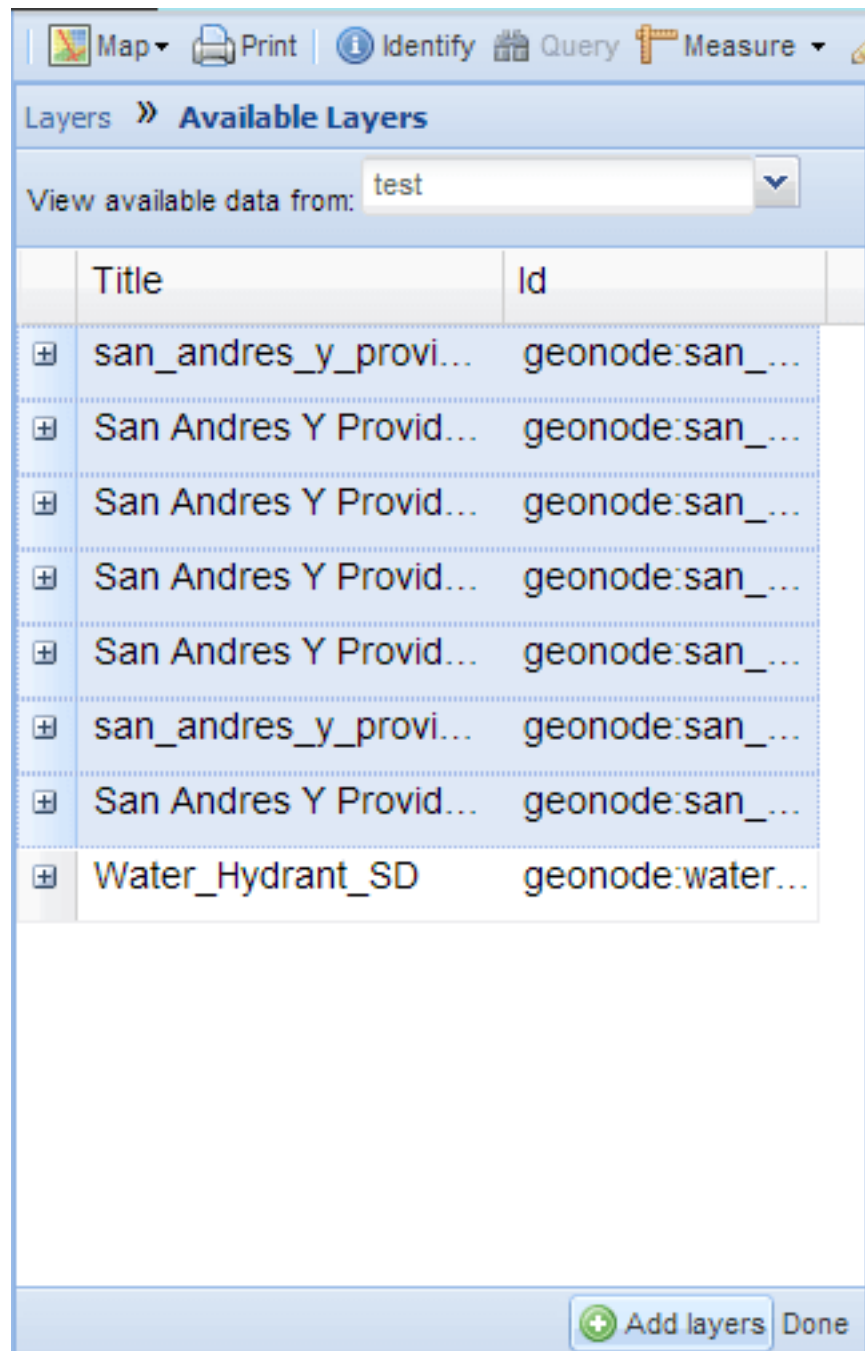


Fig. 55: Selecting layers

- The layers will be added to the map. Click *Done* (right next to *Add Layers* at the bottom) to return to the main layers list.

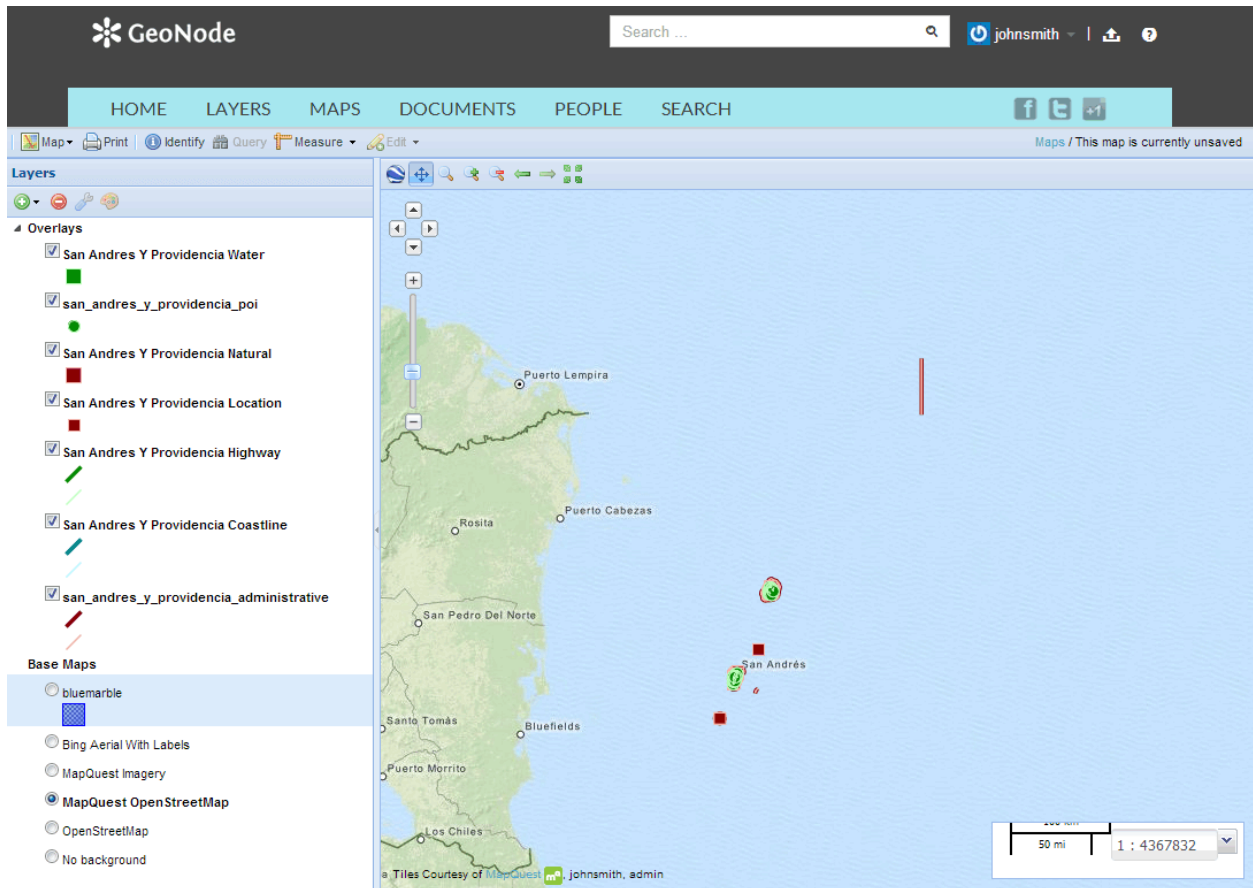


Fig. 56: Layers added to the map

Adding external layers

- Once again, click on the New Layers button and select Add Layers.
- From the top dropdown list, select *Add a New Server...*
- Enter the URL of the server, and select the correct type of server from the dropdown (WMS, TMS, or ArcGIS). For example, enter *http://e-atlas.org.au/geoserver/wms* for the URL and select *Web Map Service* as the type. Then click the *Add Server* button.
- Note - for security purposes, the URL you enter must be on a list of pre-approved external services set up by the GeoNode administrator. Otherwise you will receive a 403 error when trying to add the server.
- A list of layers available from that server should appear momentarily. The layers must be available in the Web Mercator projection or they will not show up in the list. Select the layers you want to add to the map. Click *Add Layers* to add them all to the map.
- The layers will be added to the map. Click *Done* (right next to *Add Layers* at the bottom) to return to the main layers list.

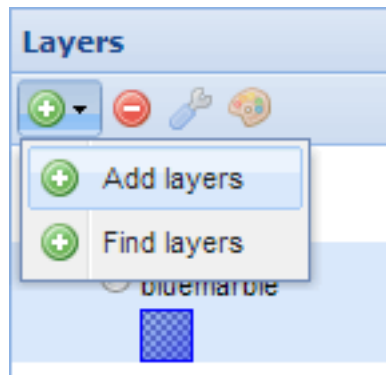


Fig. 57: Add layers link

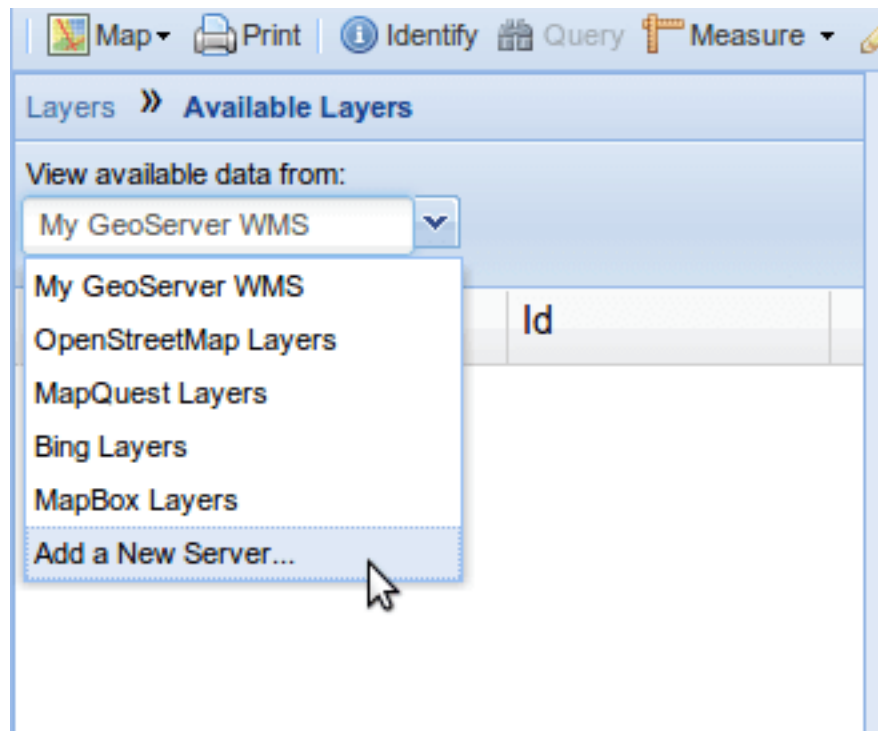


Fig. 58: Add a New Server

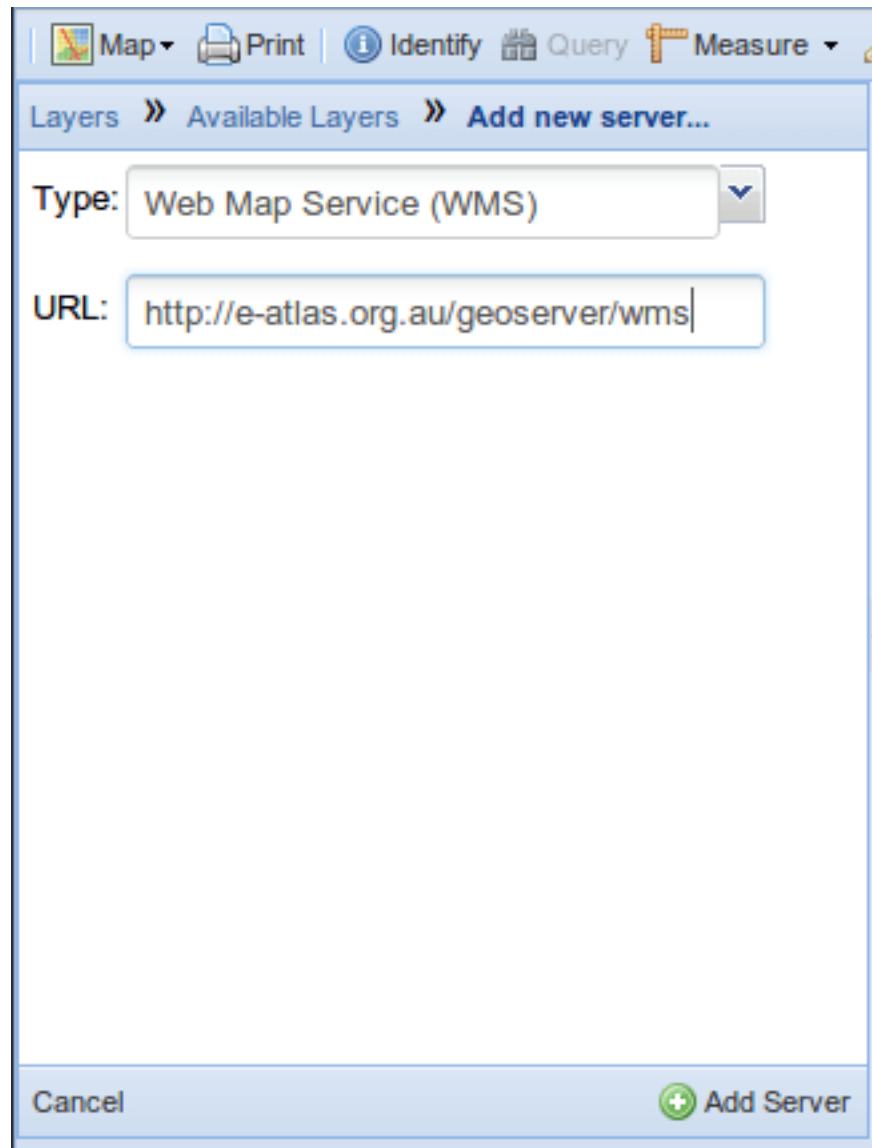


Fig. 59: *New Server URL and Type*

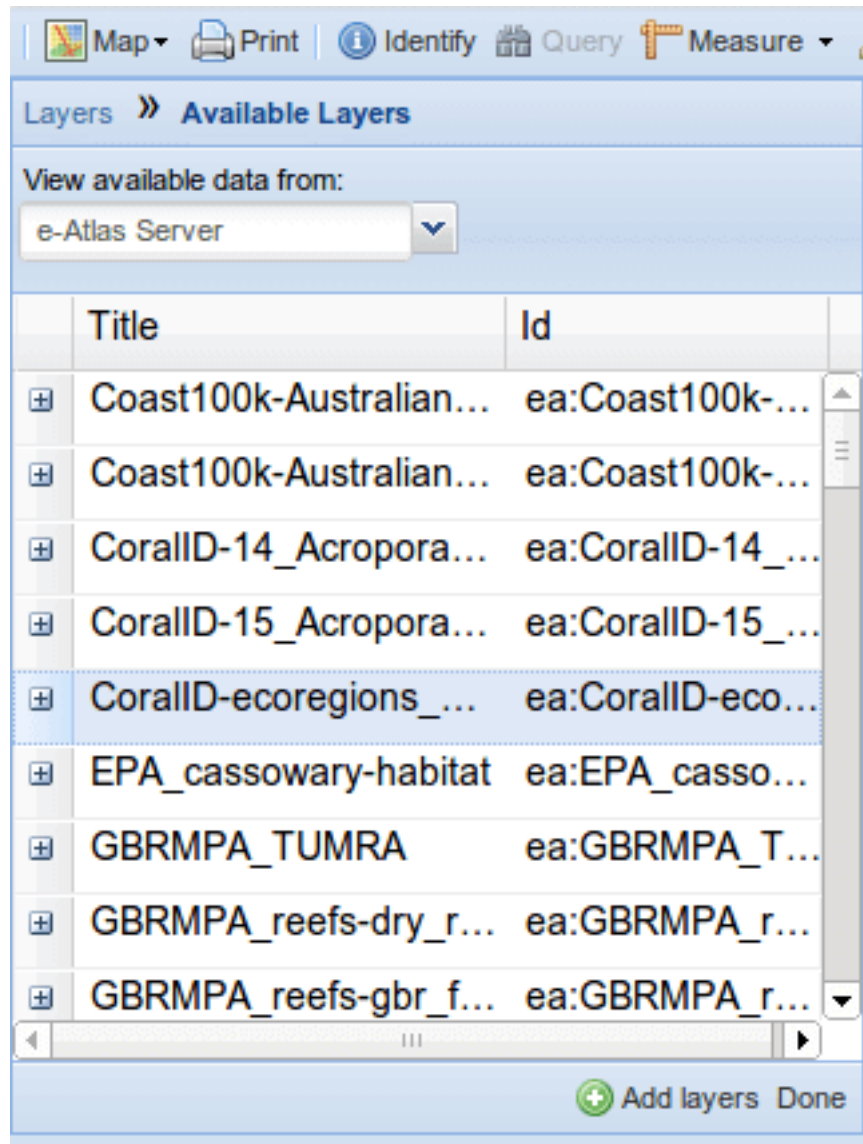


Fig. 60: Add layers

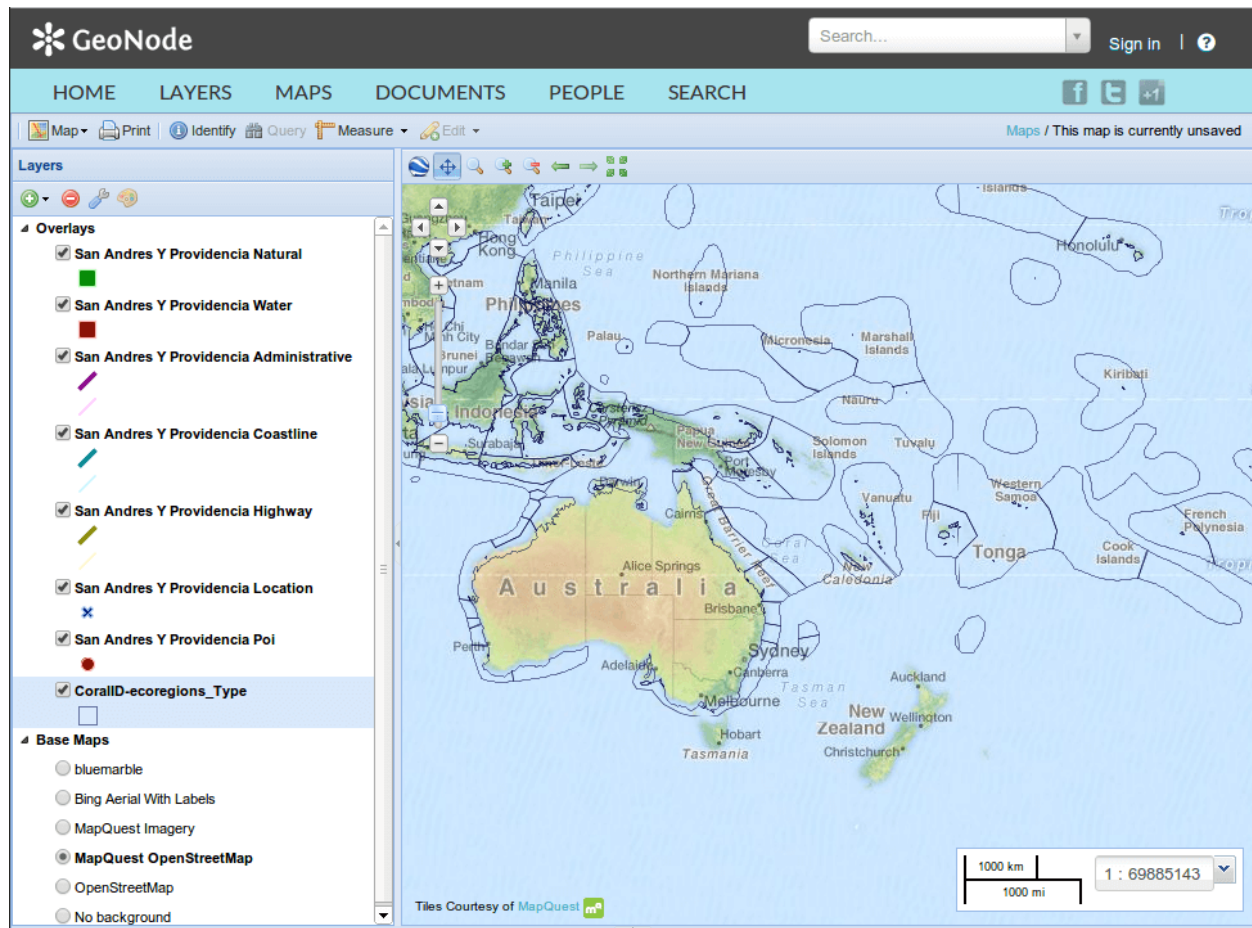


Fig. 61: Layers added to the map

Saving the map

1. While we still have some work to do on our map, let's save it so that we can come back to it later. Click on the *Map* button in the toolbar, and select *Save Map*.

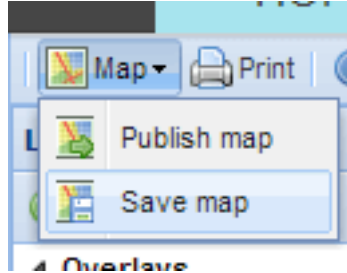


Fig. 62: Save map link

2. Enter a title and abstract for your map.
3. Click *Save*. Notice that the link on the top right of the page changed to reflect the map's name.

This link contains a permalink to your map. If you open this link in a new window, your map will appear exactly as it was saved.

Styling layers

In this interface, we can pause in our map creation and change the style of one of our uploaded layers. GeoNode allows you to edit layer styles graphically, without the need to resort to programming or requiring a technical background.

We'll be editing the `san_andres_y_providencia_poi` layer.

1. In the layer list, uncheck all of the layers except the above, so that only this one is visible (not including the base layer).
2. Zoom in closer using the toolbar or the mouse.
3. In the layer list, click to select the remaining layer and then click the palette icon (*Layer Styles*). This will bring up the style manager.
4. This layer has one style (named the same as the layer) and one rule in that style. Click the rule (*Untitled 1*) to select it, and then click on *Edit* below it.
5. Edit the style. You can choose from simple shapes, add labels, and even adjust the look of the points based on attribute values and scale.
6. When done, click *Save*, then click on the word *Layers* to return to the layer list.

Share your map

Now let's finish our map.

1. Check the box next to the *highway* layer to activate it. If it is not below the *POI* layer in the list, click and drag it down.
2. Make any final adjustments to the map composition as desired, including zoom and pan settings.
3. Click the *Map* button in the toolbar, and then click *Publish Map*.

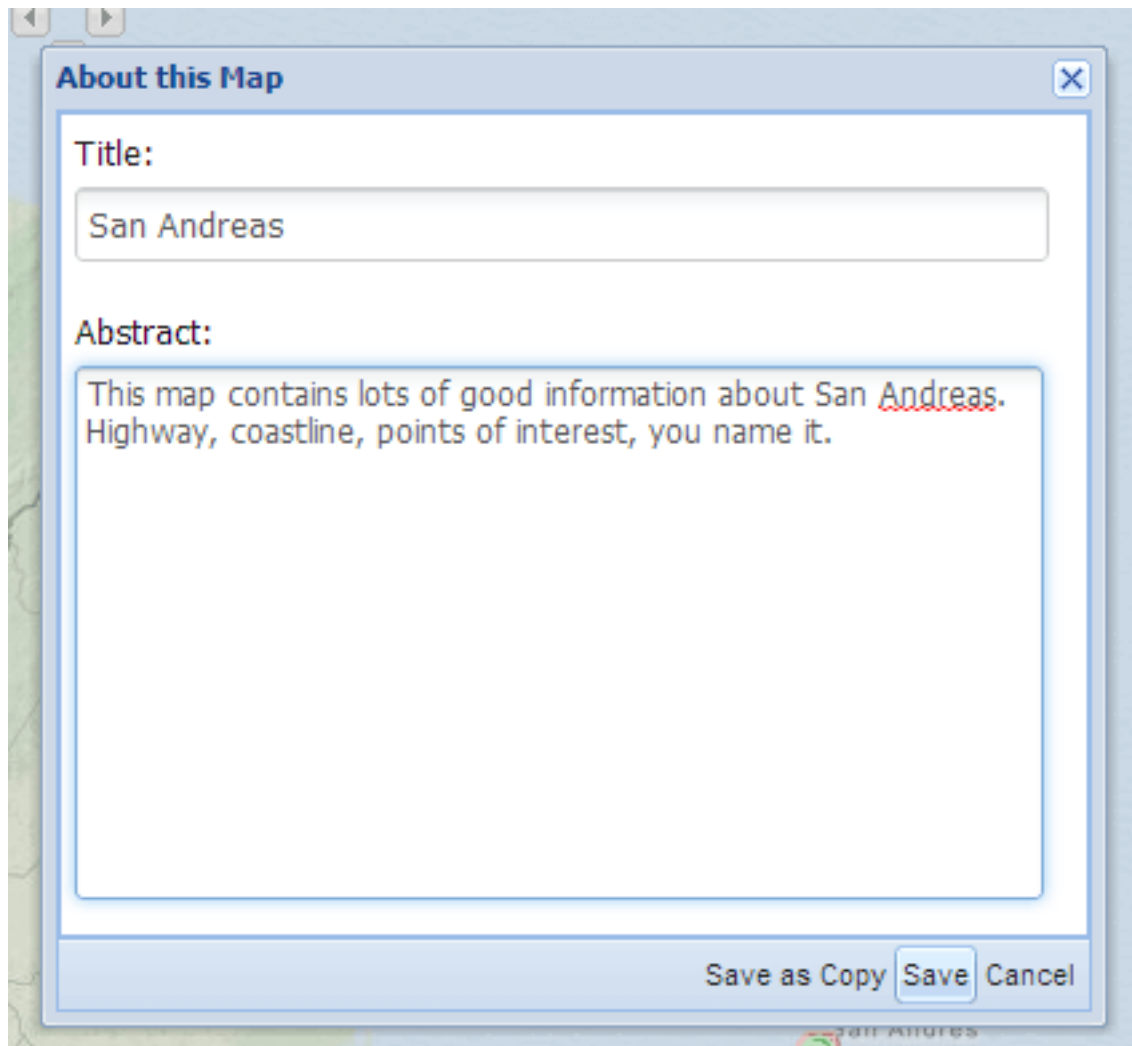


Fig. 63: *Save map dialog*

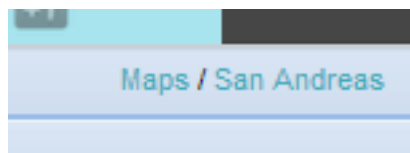


Fig. 64: *Saved map name*

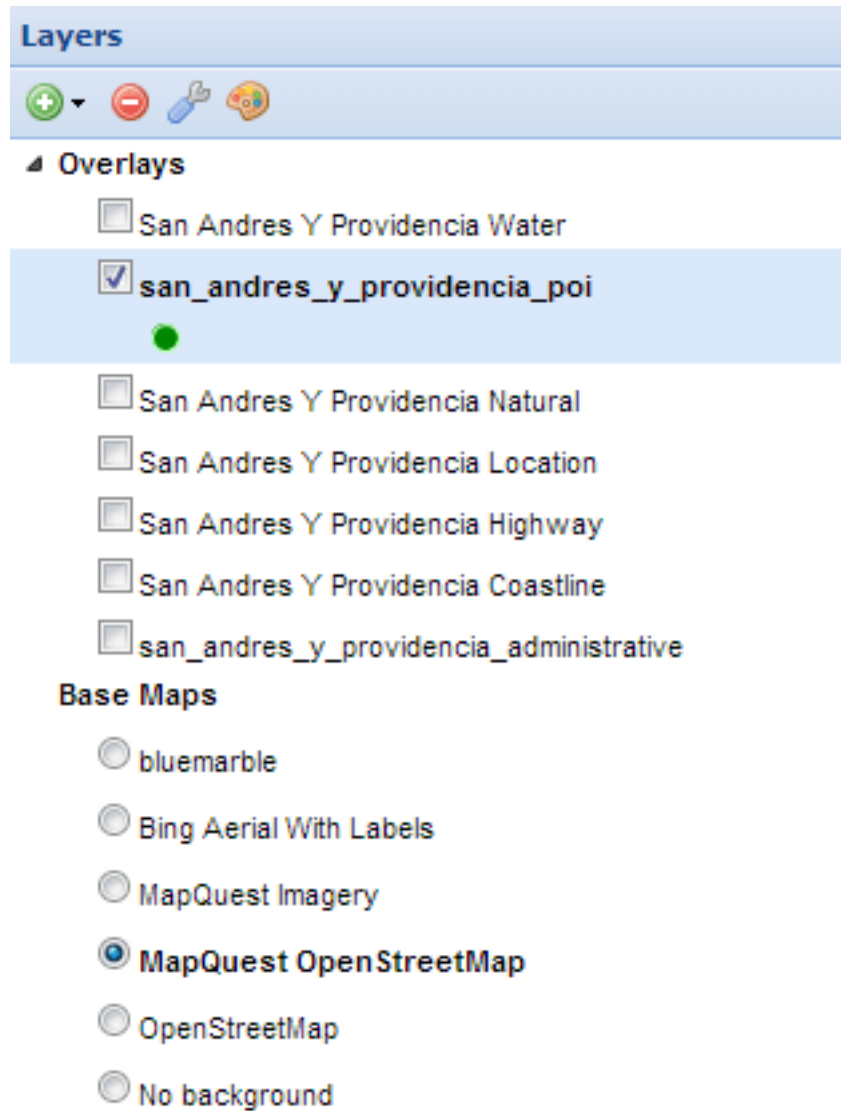


Fig. 65: Only one layer visible

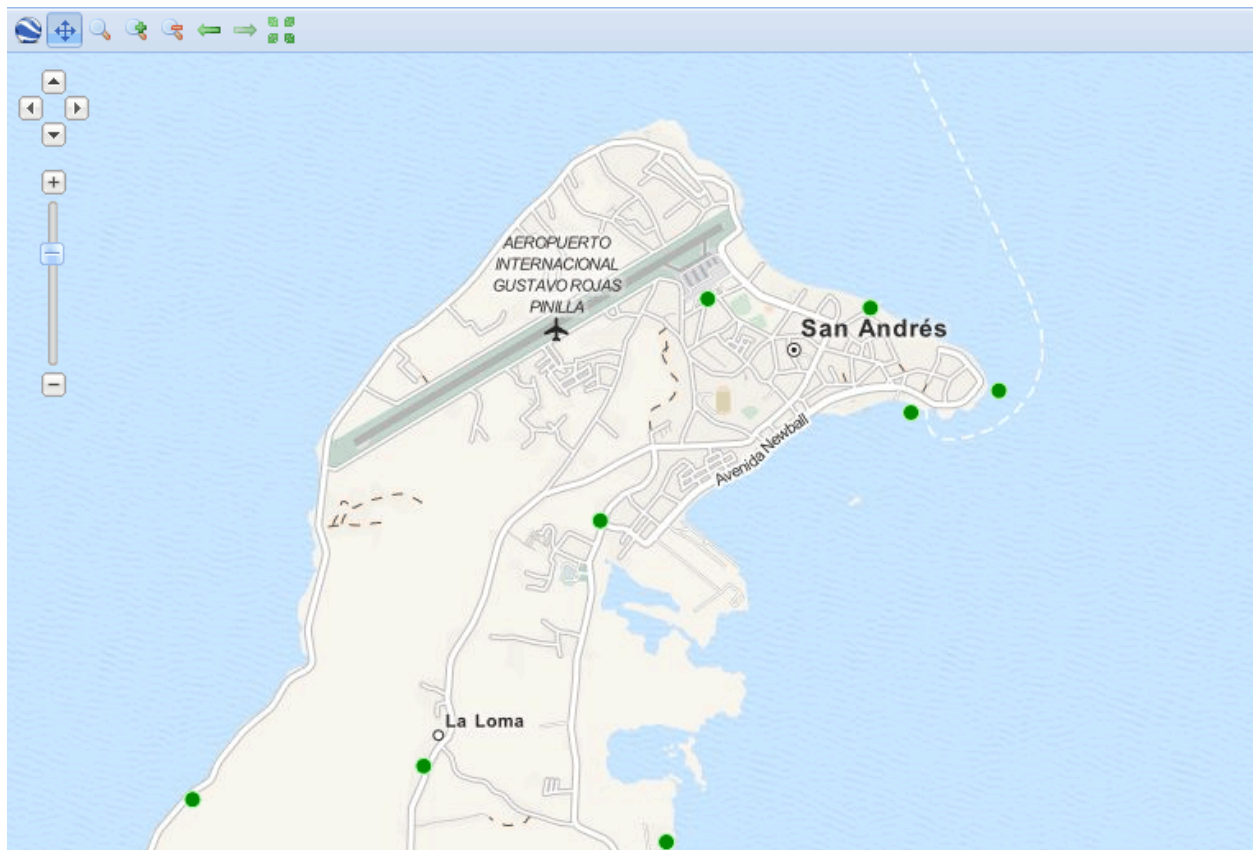
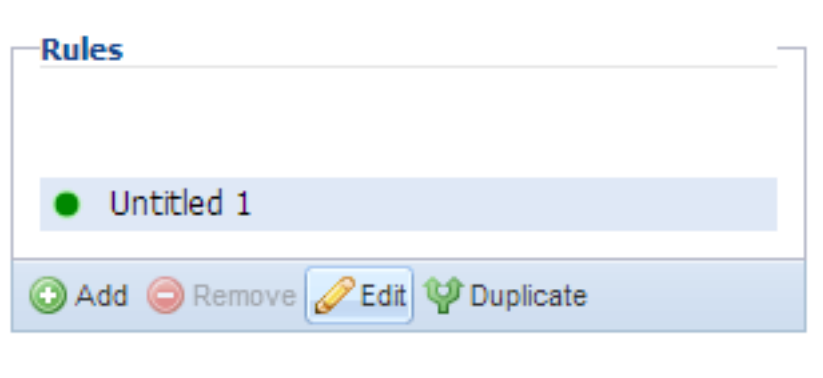


Fig. 66: Zoomed in to see the layer better

Fig. 67: *Styles manager*Fig. 68: *Edit style rule link*

The image shows a web-based style editor interface with three tabs: 'Basic', 'Labels', and 'Advanced'. The 'Basic' tab is active. It contains the following controls:

- Name:** A text input field containing 'Points of Interest'.
- Symbol:** A small green circle icon.
- Symbol:** A dropdown menu with 'circle' selected.
- Size:** A text input field containing '10'.
- Rotation:** An empty text input field.
- Fill:** A checked checkbox followed by the word 'Fill'. Below it is a color selection area with a teal square labeled '#006689' and an opacity slider set to approximately 50%.
- Stroke:** A checked checkbox followed by the word 'Stroke'. Below it is a style selection area with 'solid' selected, a color selection area with a light green square labeled '#bbffbb', a width text input field containing '1', and an opacity slider set to approximately 50%.

Fig. 69: *Editing basic style rules*

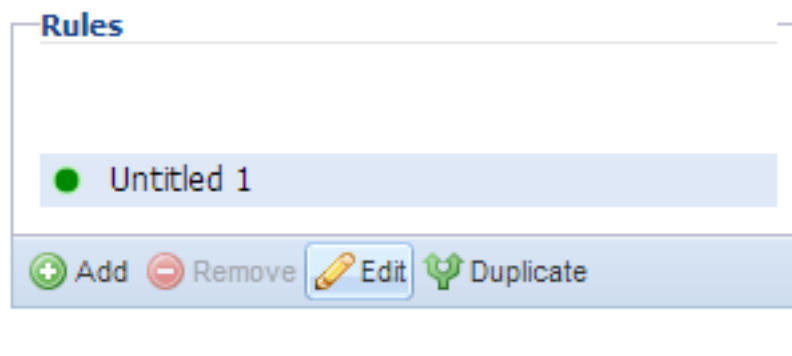


Fig. 70: Editing style labels

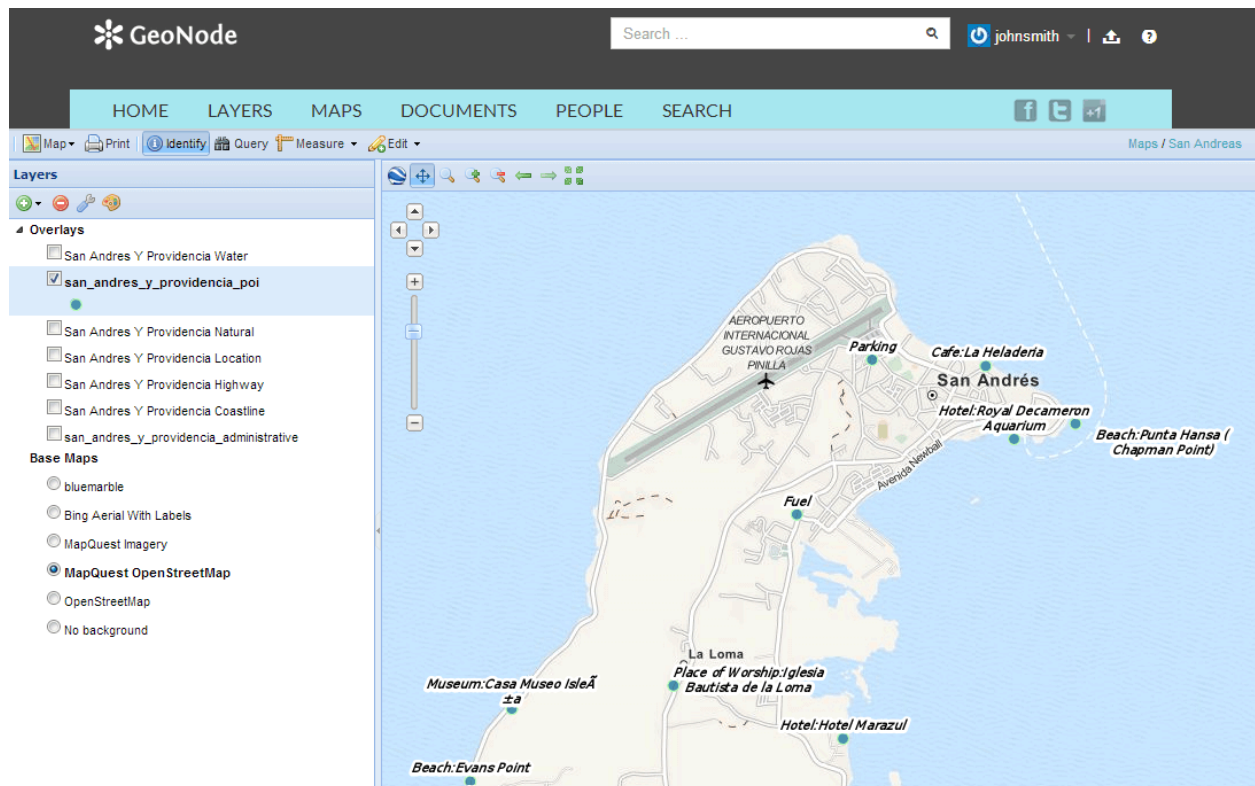


Fig. 71: Styled layer

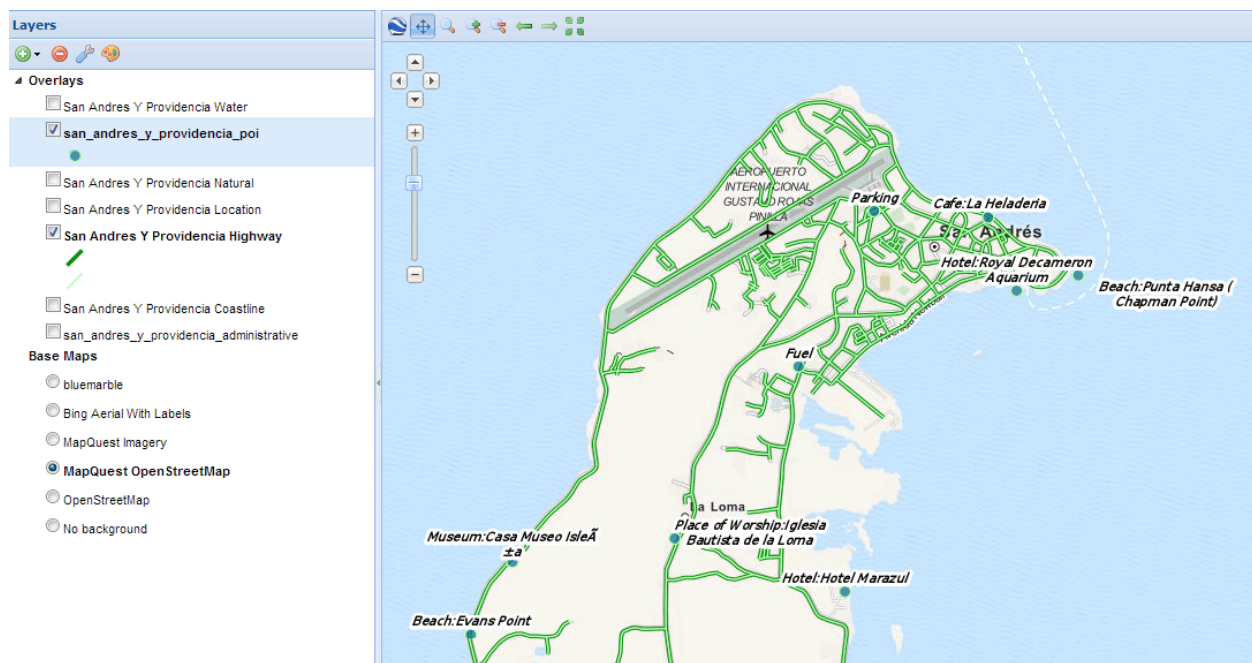


Fig. 72: Adjusting map composition

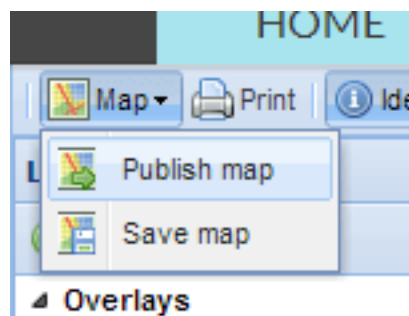


Fig. 73: Publish map link

4. The title and abstract as previously created should still be there. Make any adjustments as necessary, and click *Save*.
5. A new dialog will appear with instructions on how to embed this map in a webpage, including a code snippet. You can adjust the parameters as necessary.



Fig. 74: Map publishing options

Your map can now be shared.

3.2.1.3.7 Using GeoNode with other applications

Your GeoNode project is based on core components which are interoperable and as such, it is straightforward for you to integrate with external applications and services. This section will walk you through how to connect to your GeoNode instance from other applications and how to integrate other services into your GeoNode project. When complete, you should have a good idea about the possibilities for integration, and have basic knowledge about how to accomplish it. You may find it necessary to dive deeper into how to do more complex integration in order to accomplish your goals, but you should feel comfortable with the basics, and feel confident reaching out to the wider GeoNode community for help.

OGC services

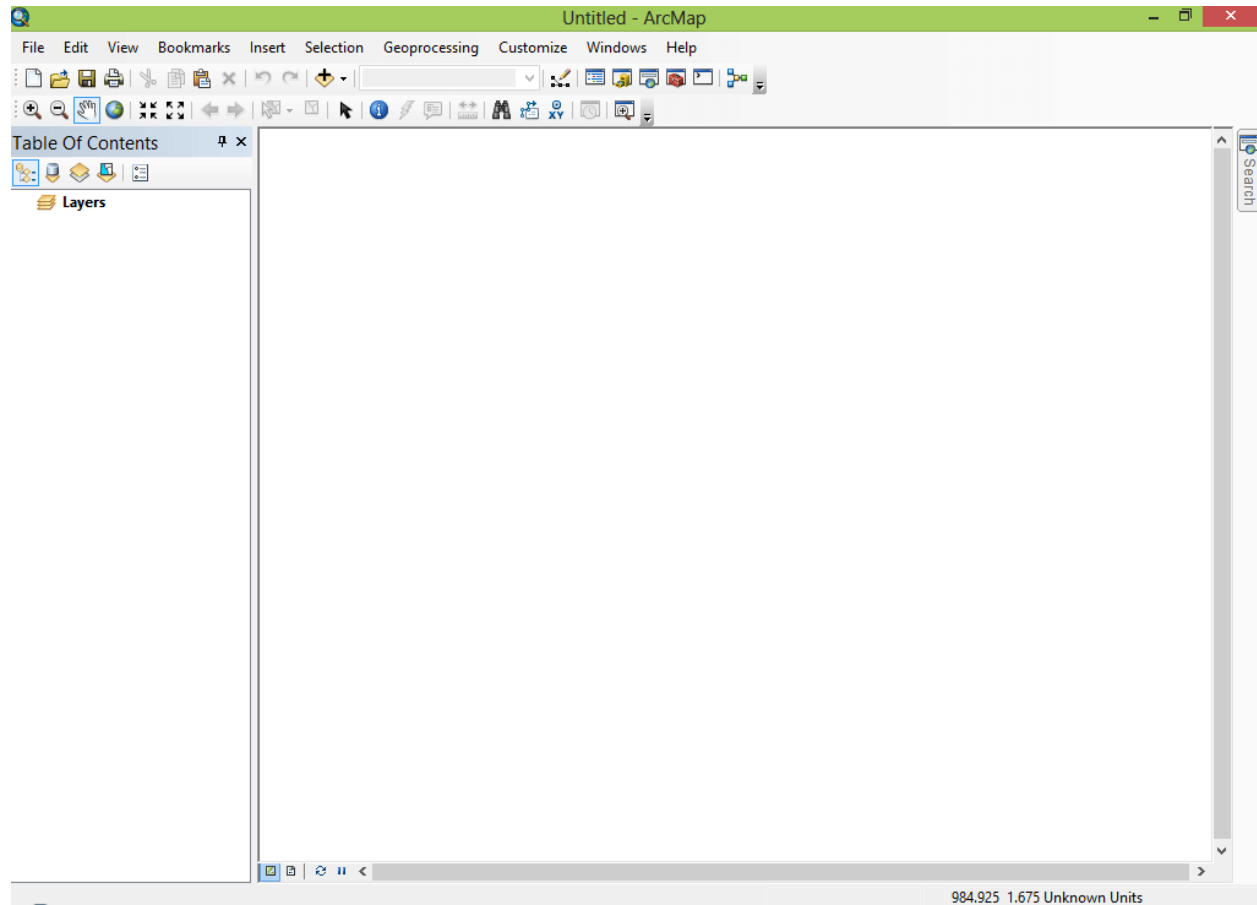
Since GeoNode is built on GeoServer which is heavily based on OGC services, the main path for integration with external services is via OGC Standards. A large number of systems, applications and services support adding WMS layers to them, but only a few key ones are covered below. WFS and WCS are also supported in a wide variety of clients and platforms and give you access to the actual data for use in geoprocessing or to manipulate it to meet your requirements. GeoServer also bundles GeoWebCache which produces map tiles that can be added as layers in many popular web mapping tools including Google Maps, Leaflet, OpenLayers and others. You should review the reference material included in the first chapter to learn more about OGC Services and when evaluating external systems make sure that they are also OGC Compliant in order to integrate as seamlessly as possible.

Use GeoNode with...

ArcGIS

ArcGIS Desktop (ArcMap) supports adding WMS layers to your map project. The following set of steps will walk you through how to configure a WMS Layer from your GeoNode within ArcMap.

First, you can start with a new empty project or add these layers to your existing project.



Next click the ArcCatalog button on the toolbar to bring up its interface.

From there, double click the “Add WMS Server” item in the tree to bring up the dialog that lets you enter the details for your WMS.

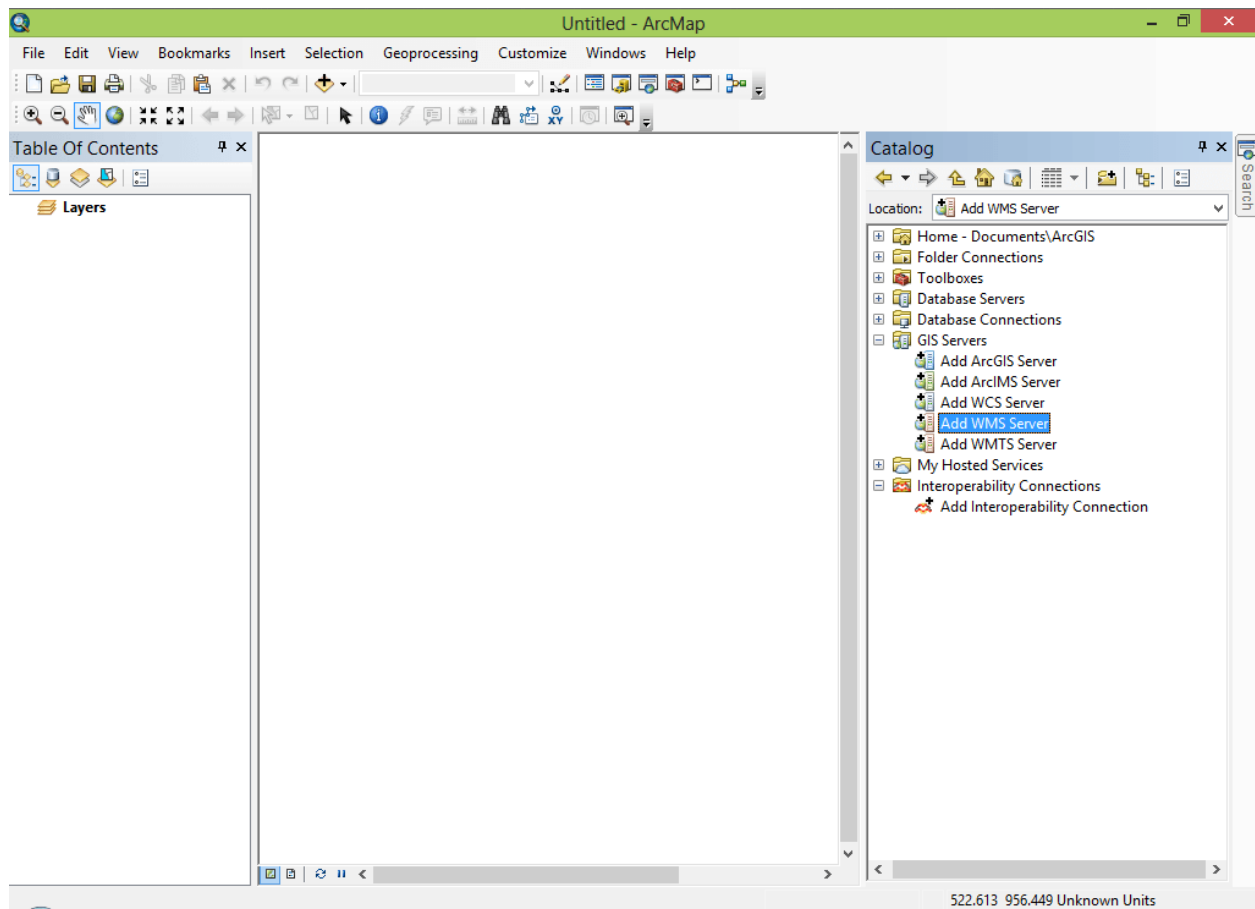
Next, enter the URL for your GeoNode’s WMS endpoint which is the base URL with /geoserver/wms appended to the end of the URL. You can also enter your credentials into the optional Account section of this dialog to gain access to non-public layers that your user may have access to.

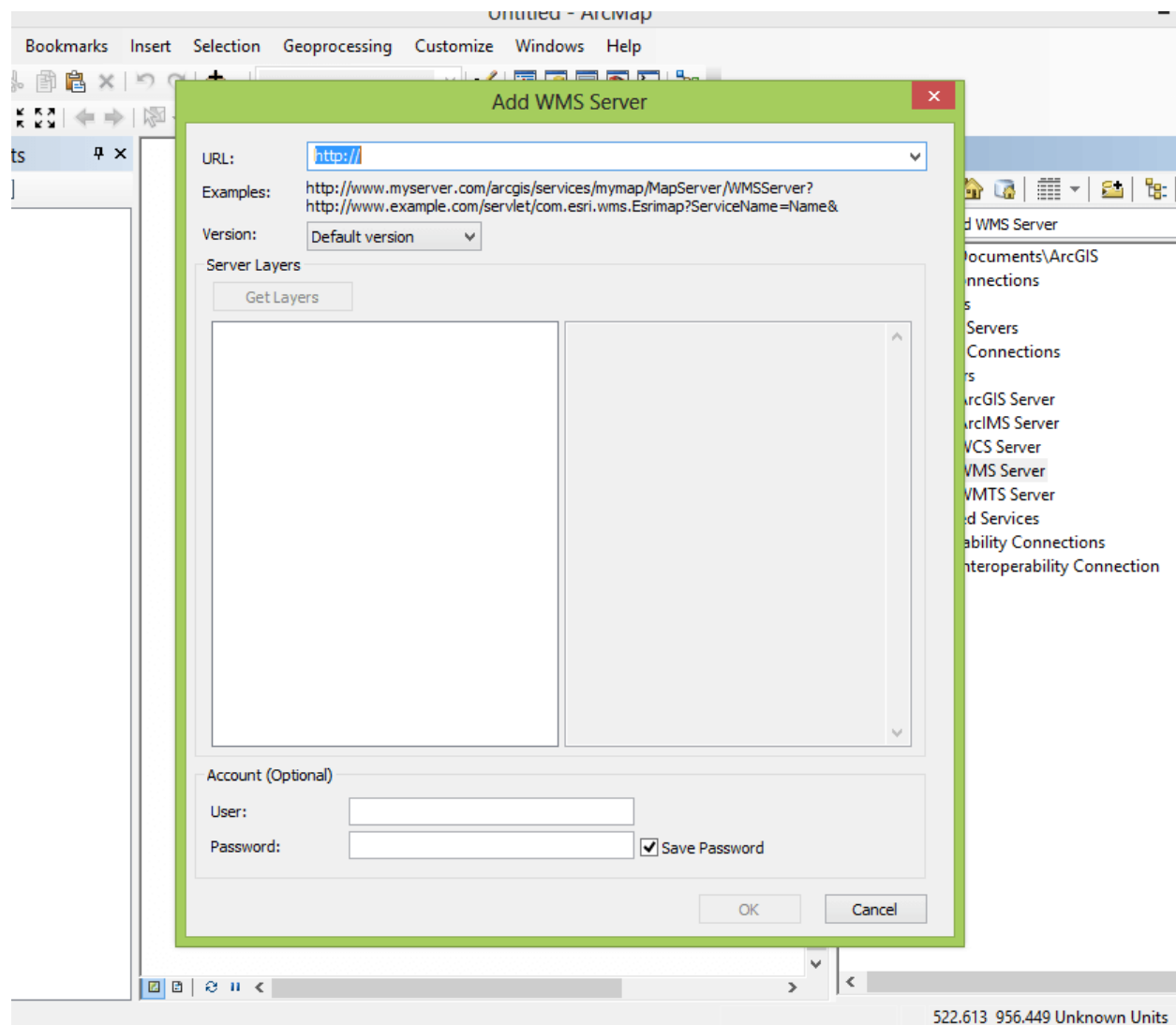
Click the “Get Layers” button to ask ArcMap to query your WMS’s GetCapabilities document to get the list of available layers.

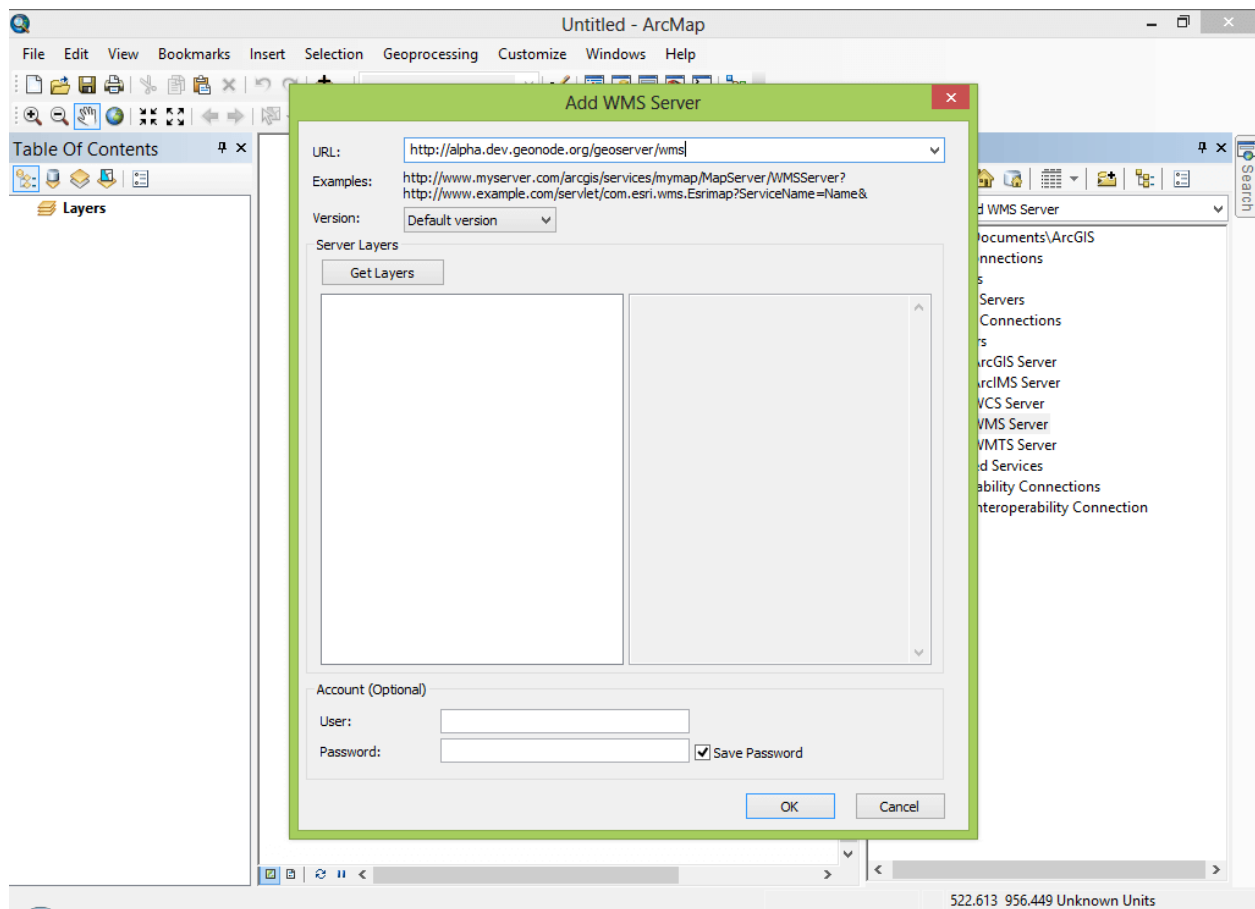
After you click the OK button, your GeoNode layers will appear in the ArcCatalog Interface.

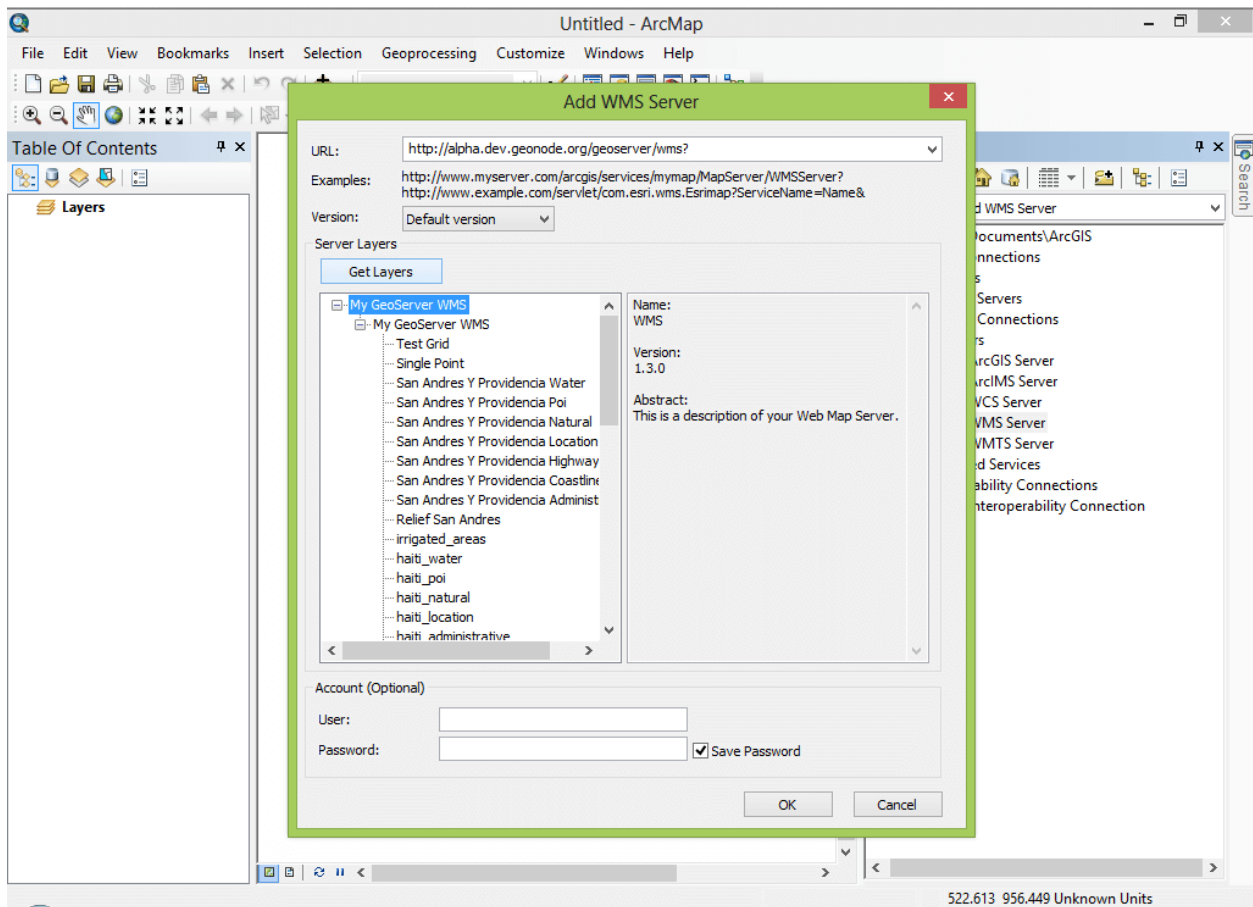
Once your server is configured in ArcMap, you can right click on one of the layers and investigate its properties.

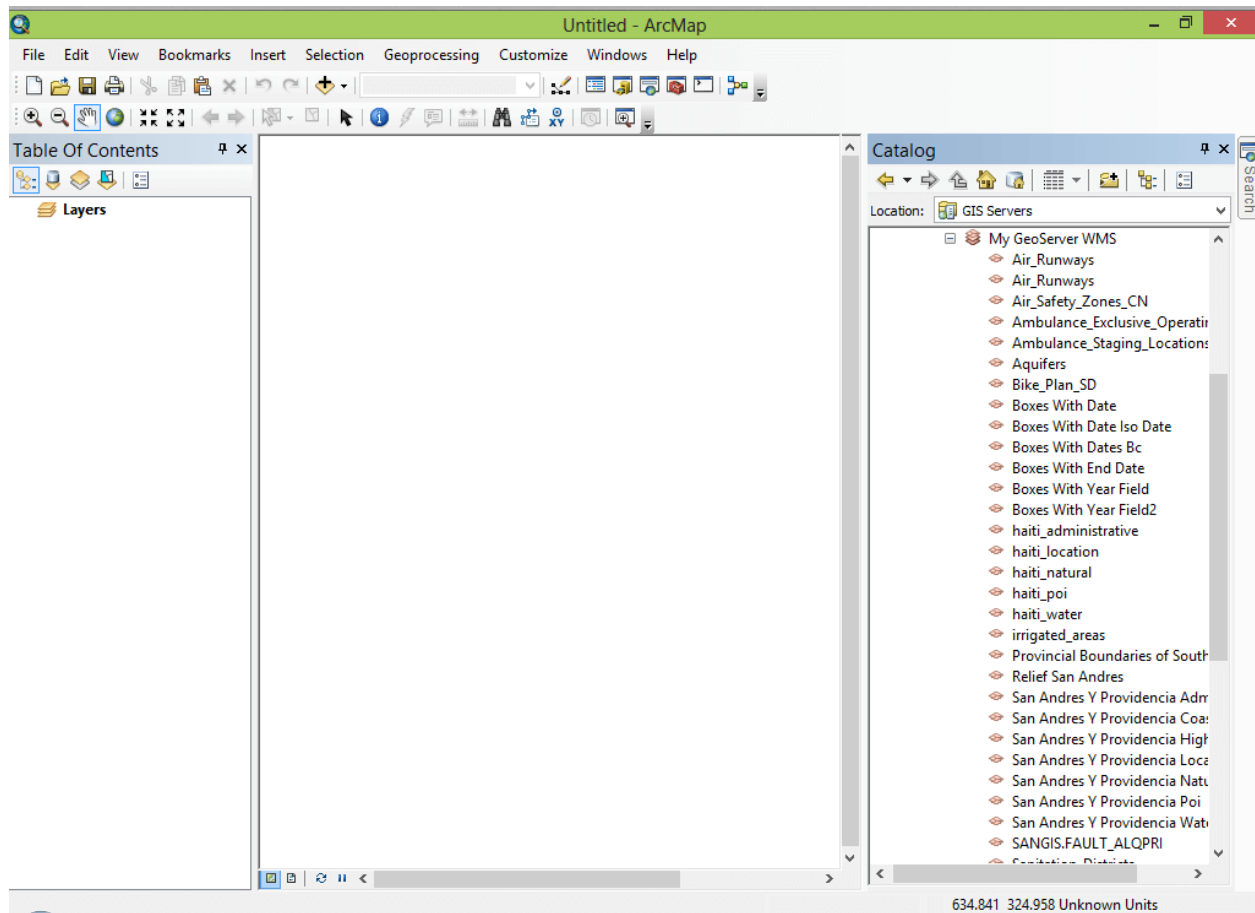
In order to actually add the layer to your project, you can drag and drop it into the Table of Contents, or right click and select “Create Layer”. Your Layer will now be displayed in the map panel of your project.

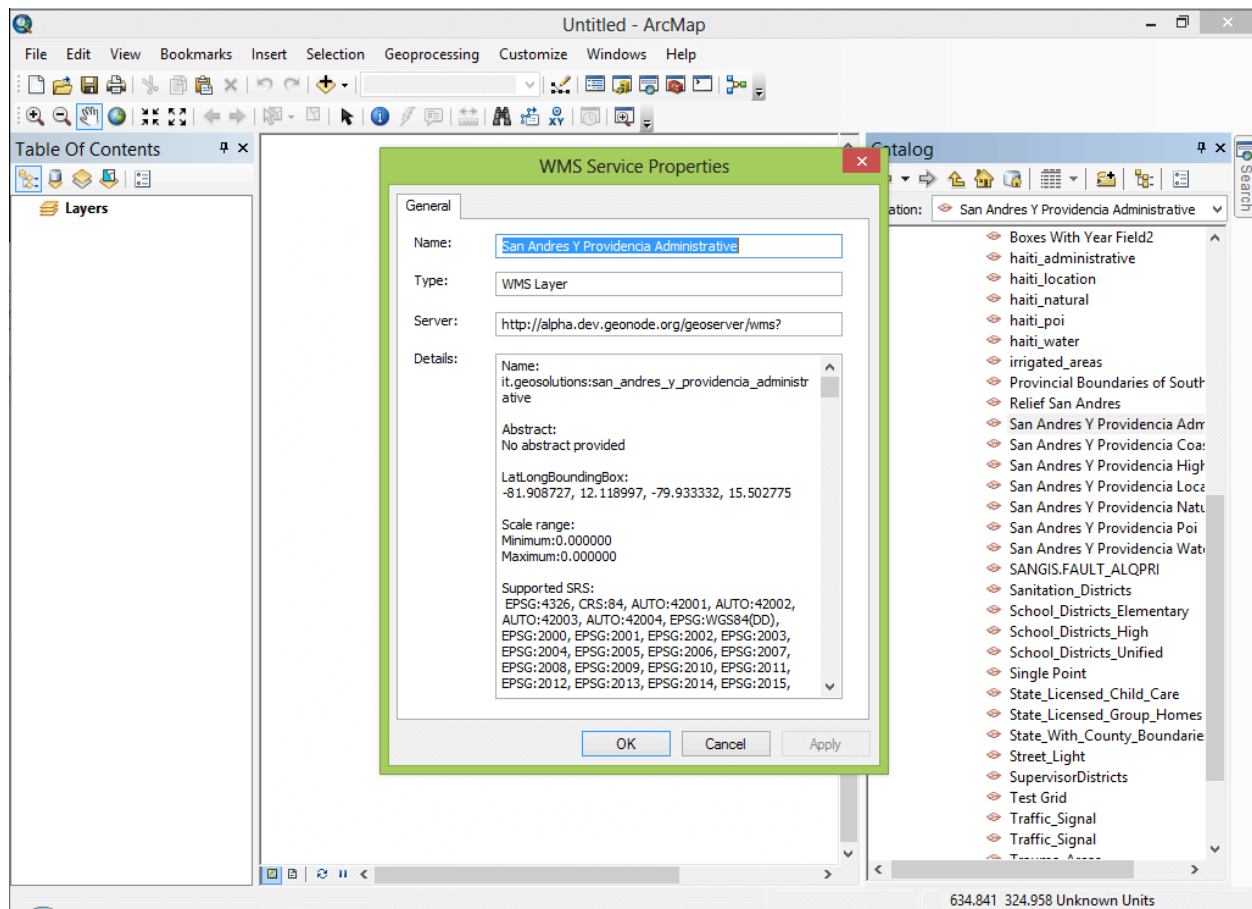


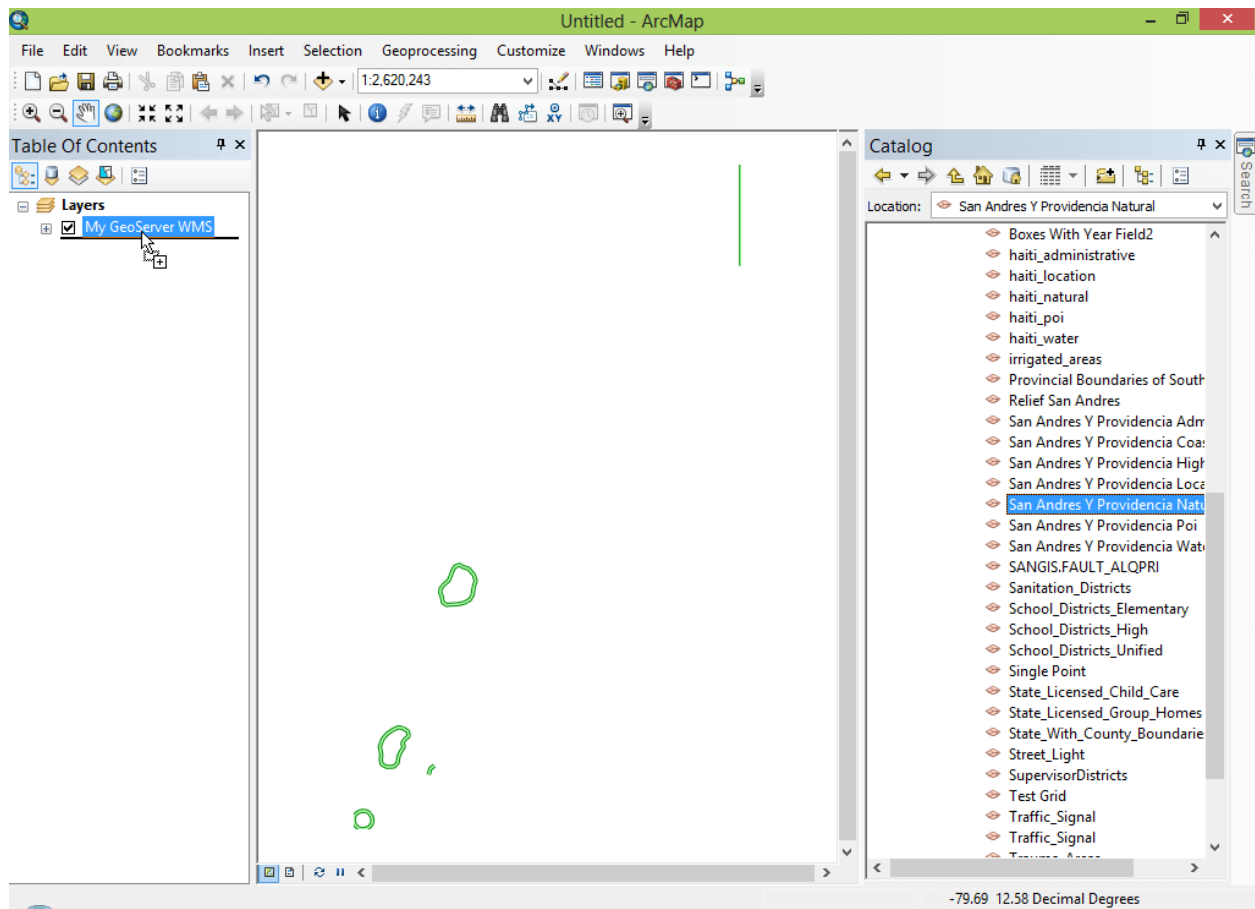


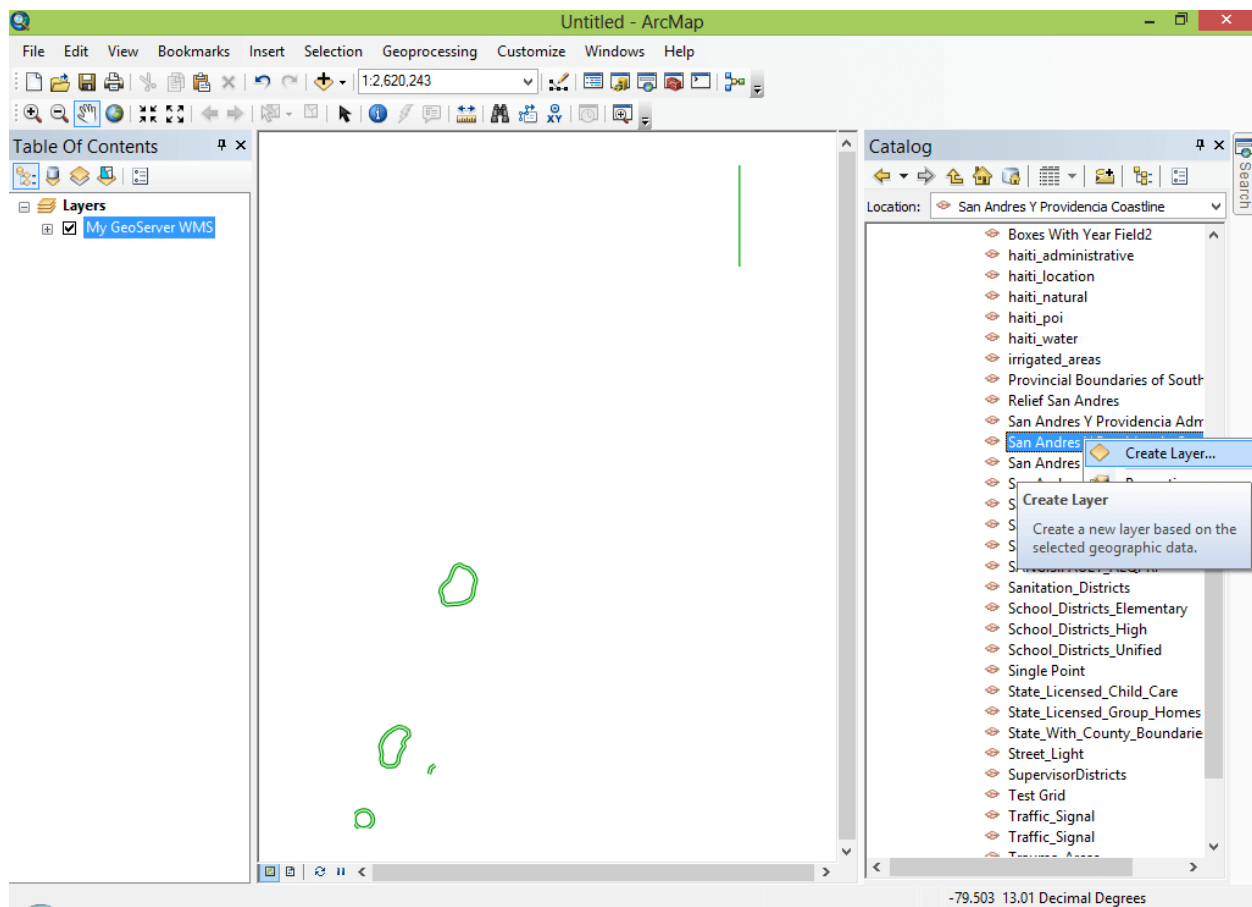




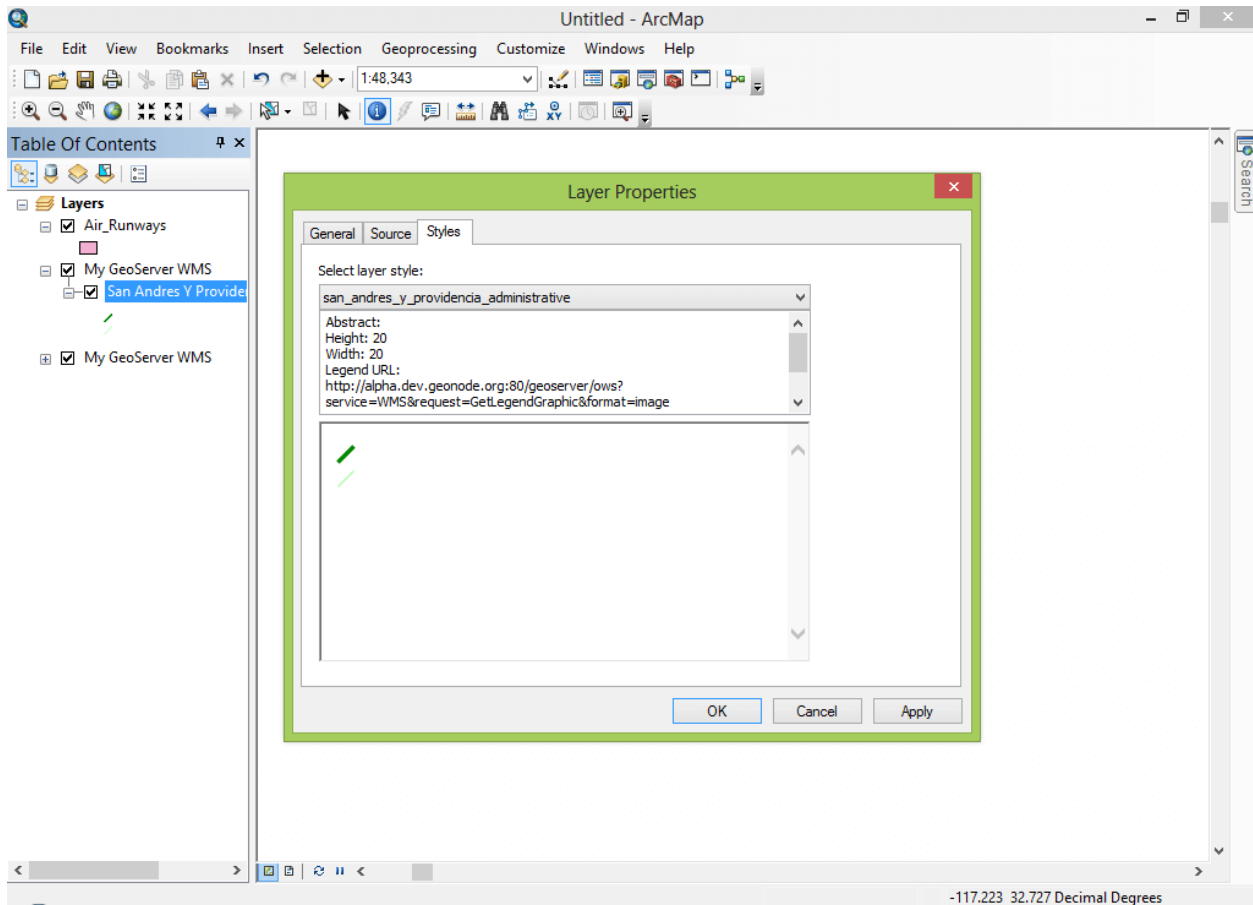








Once the layer is in your projects Table of Contents, you can right click on it and select the Layer Properties option and select the Styles Tab to choose from the available styles for that layer.



Now that we have seen how to add a WMS layer to our ArcMap project, let's walk through how to add the same layers as a WFS which retrieves the actual feature data from your GeoNode rather than a rendered map as you get with WMS. Adding layers as a WFS gives you more control over how the layers are styled within ArcMap and makes them available for you to use with other ArcGIS tools like the Geoprocessing toolbox.

Note: Adding WFS layers to ArcMap requires that you have the Data Interoperability Extension installed. This extension is not included in ArcMap by default and is licensed and installed separately.

Start by opening up the ArcCatalog Interface within ArcMap and make sure that you have the “Interoperability Connections” option listed in the list.

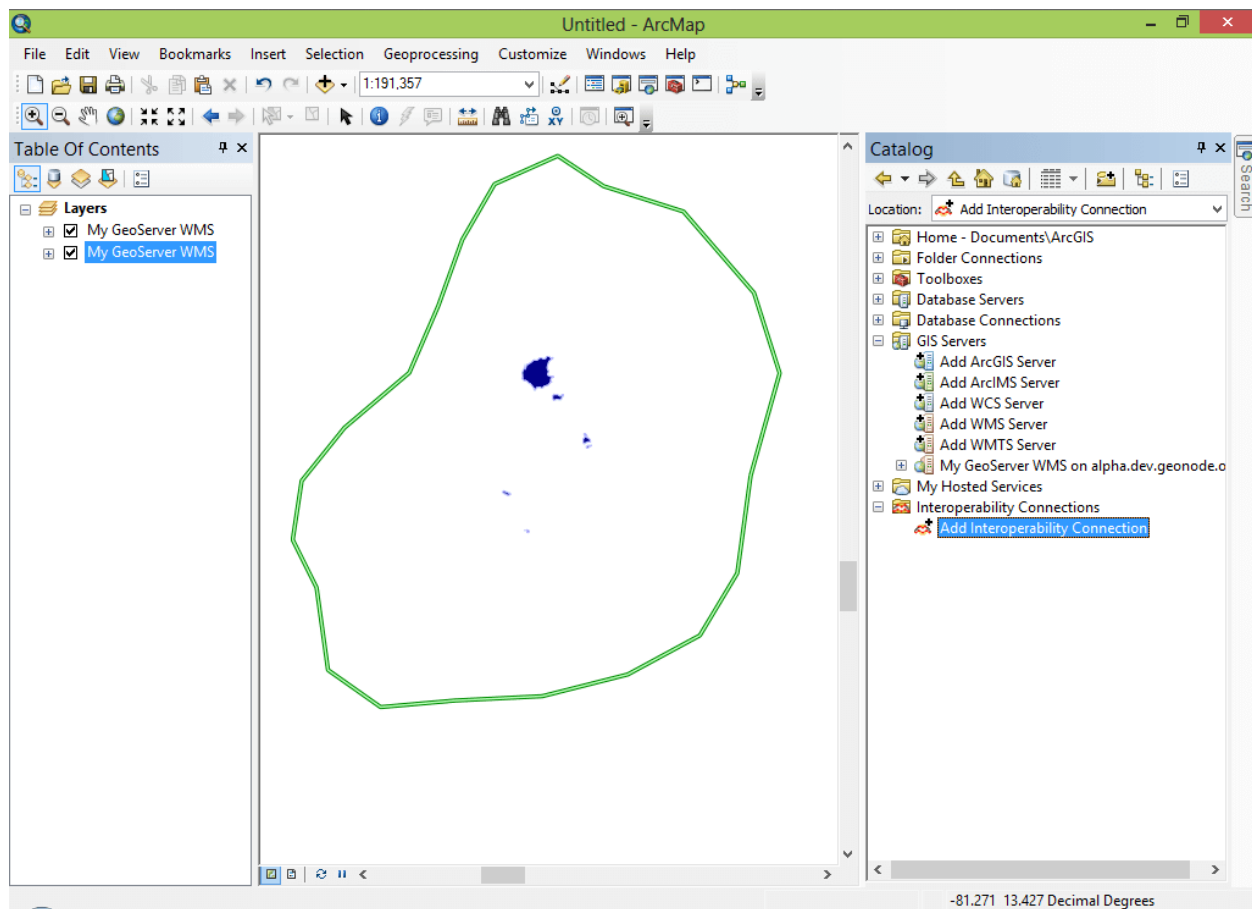
Next select “Add Interoperability Connection” to bring up the dialog that lets you add the WFS endpoint from your GeoNode.

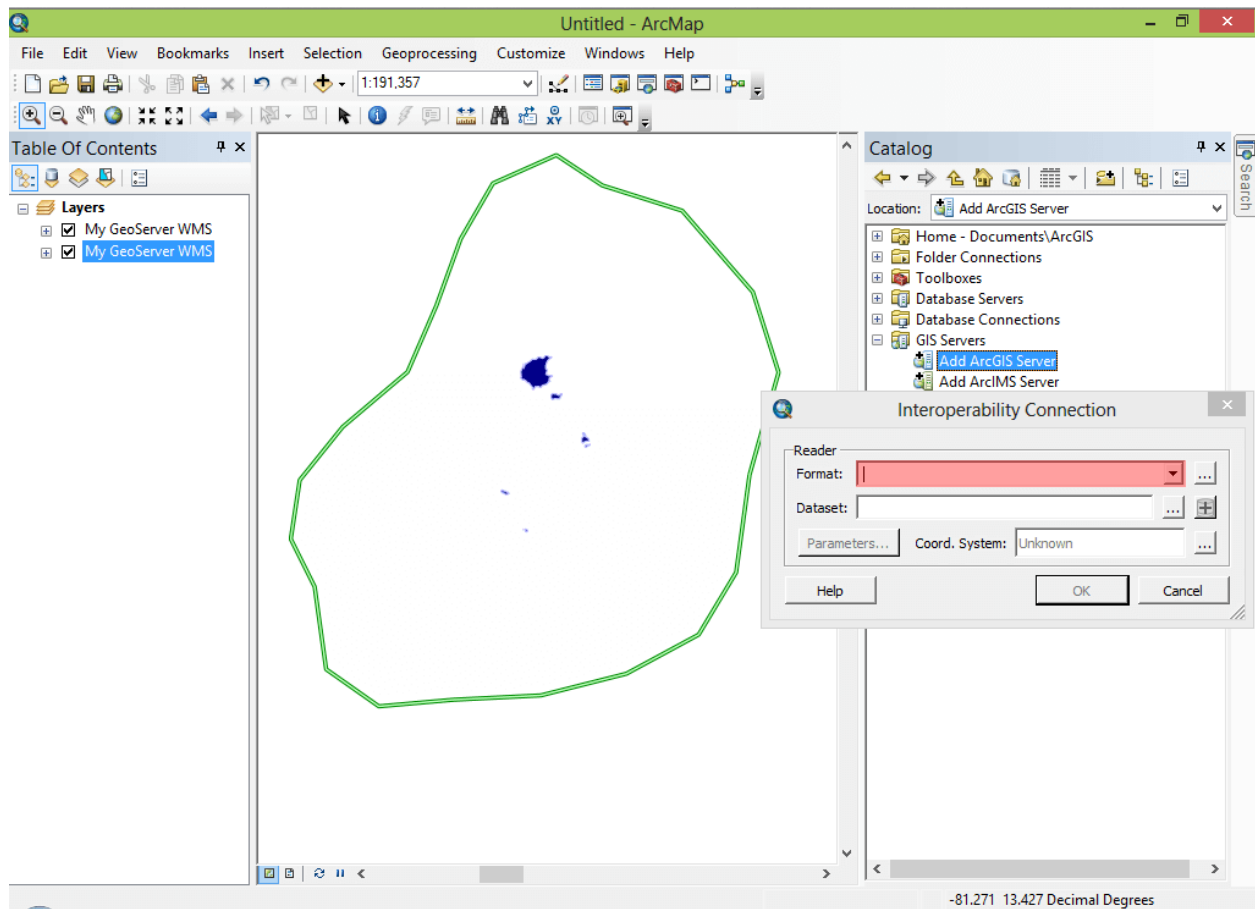
Select “WFS (Web Feature Service)” in the Format dropdown and enter the URL to the WFS endpoint for your GeoNode in the Dataset field. The WFS endpoint is your base URL + /geoserver/wfs

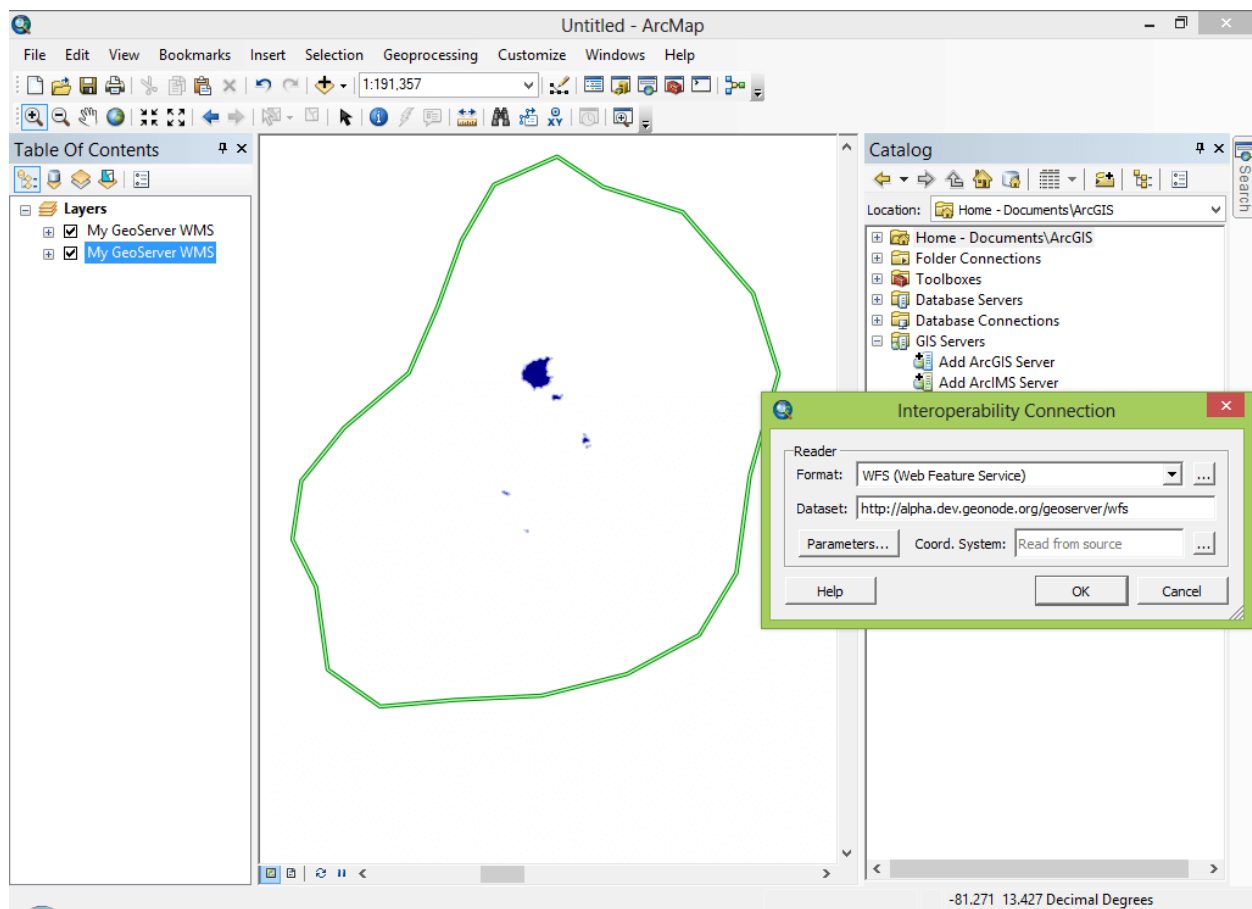
You will need to click the “Parameters” button to supply more connection information including your credentials which will give you the ability to use private layers that you have access to.

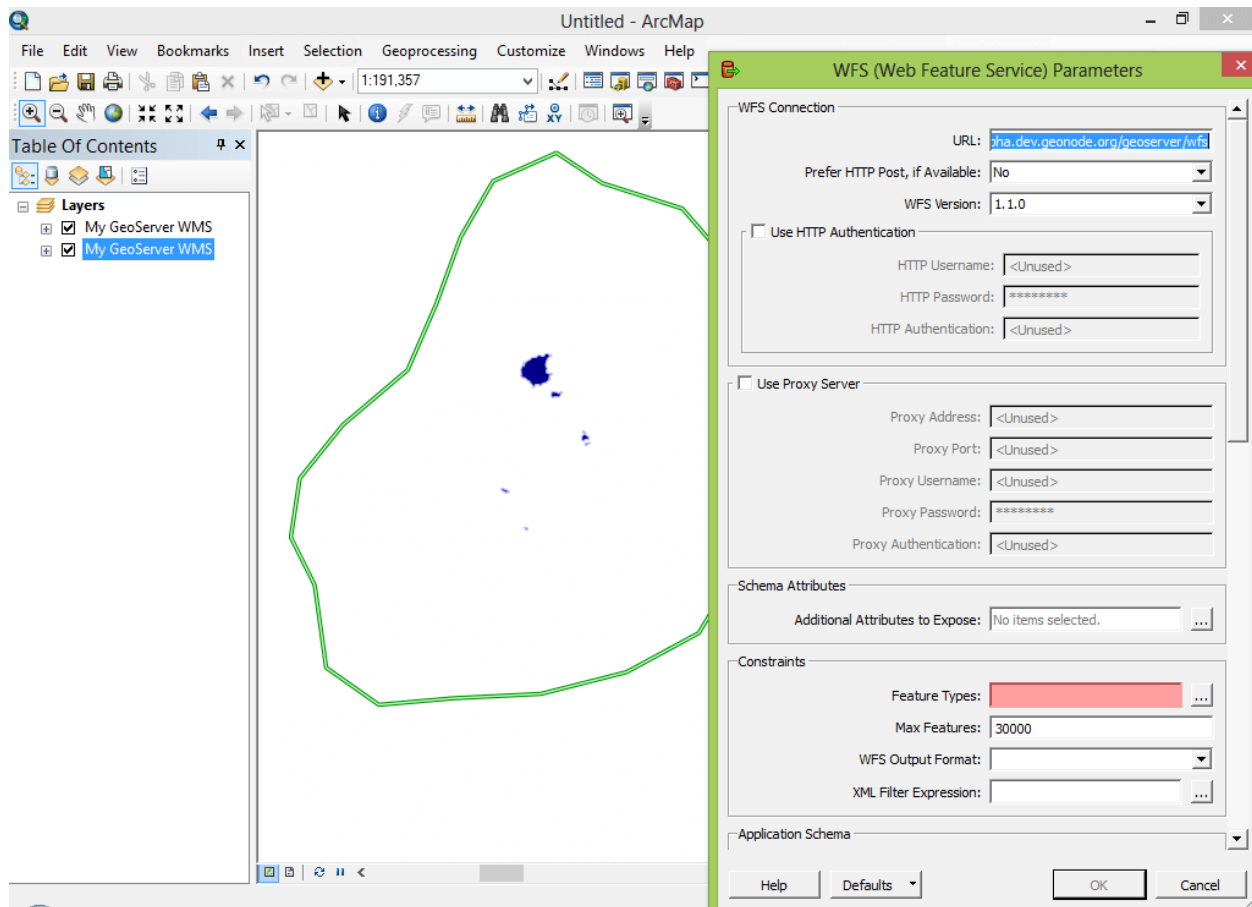
Select the Feature Types button to have ArcMap get a list of layers from the WFS Service of your GeoNode.

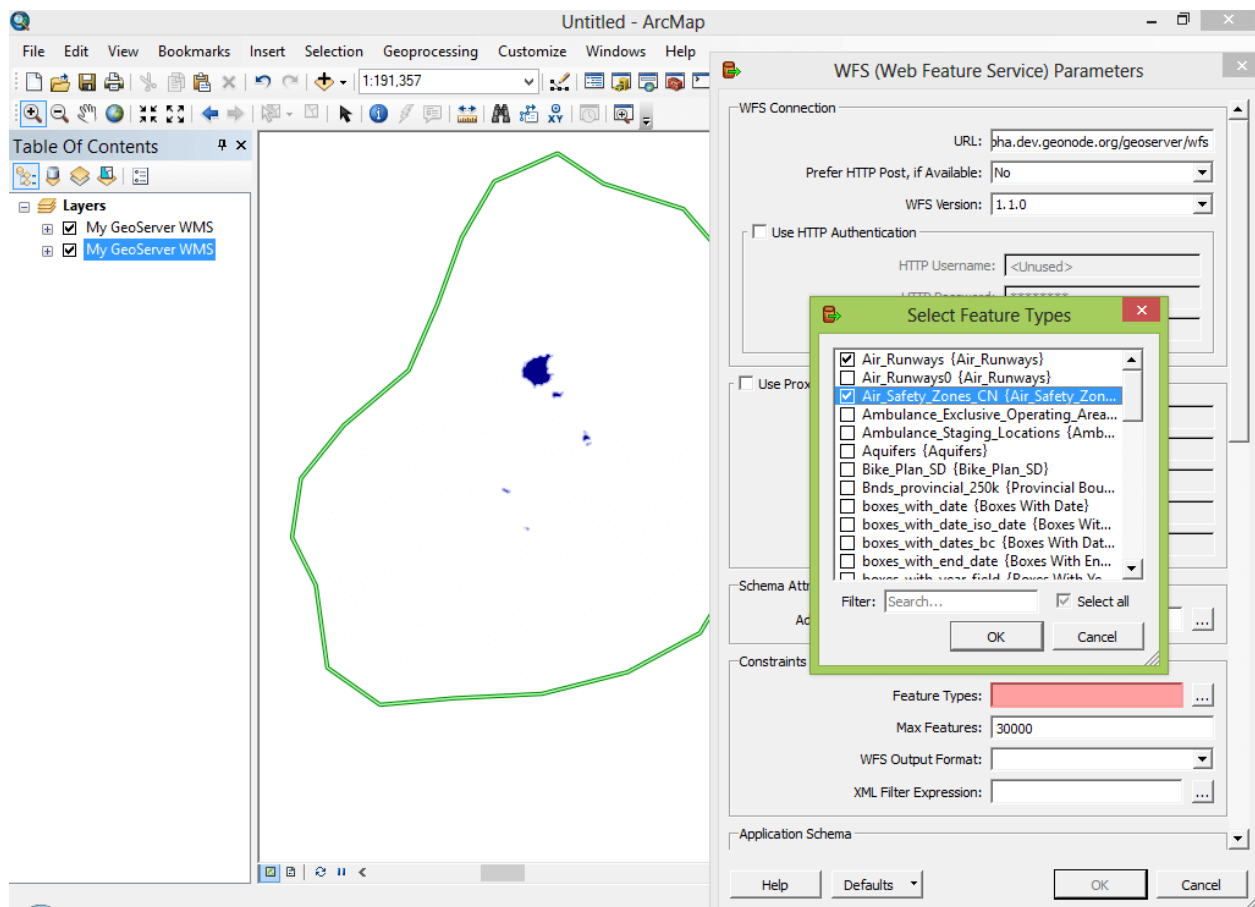
Select the layers that you want to add and click OK and ArcMap will import the features from your GeoNode into the



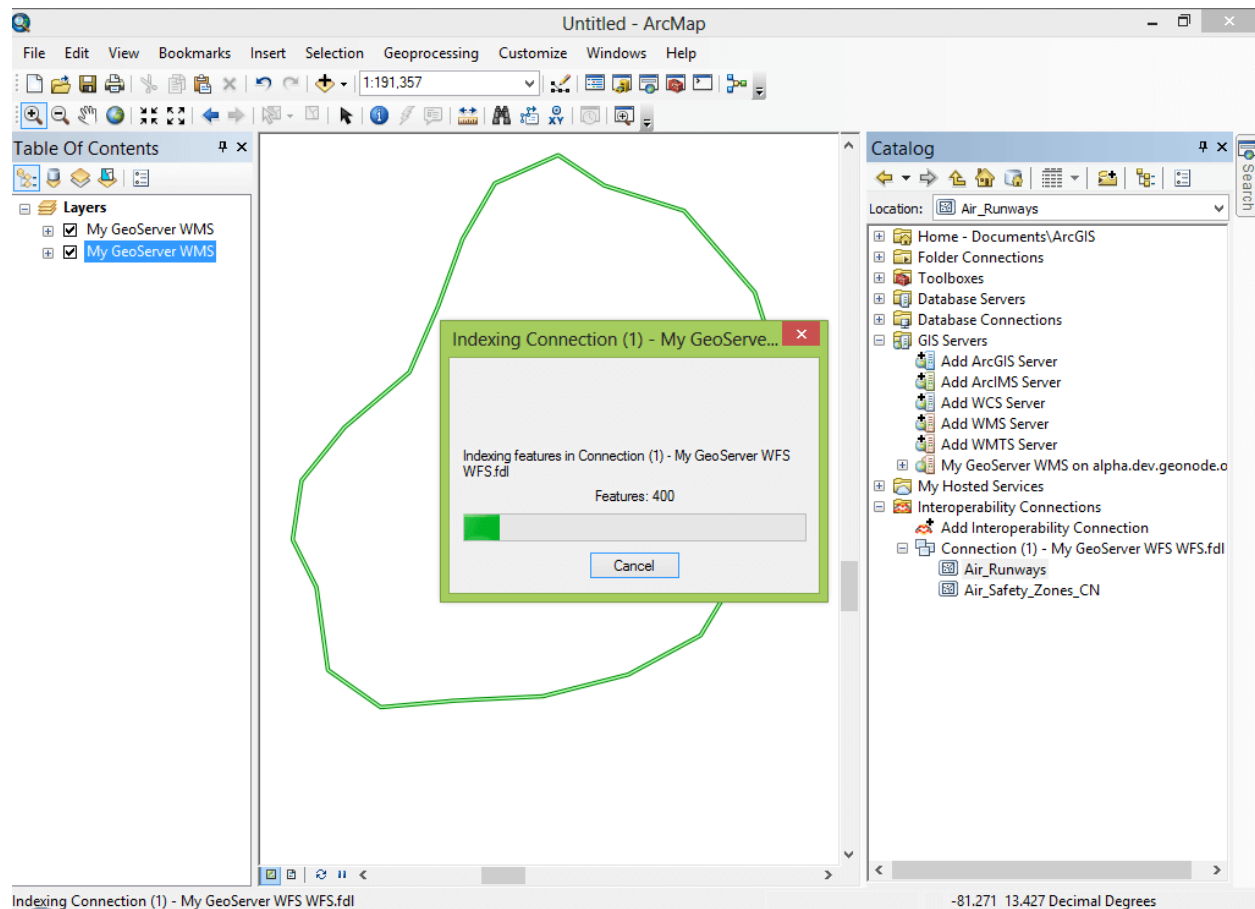








system.



Depending on the projection of your data, you may receive a warning about Alignment and Accuracy of data transformations. You can specify the transformation manually or simply hit close to ignore this dialog. If you don't want to be warned again, use the checkboxes in this dialog to hide these warnings temporarily or permanently.

Your WFS Layer will be added to your map and you can view it in the Map Panel. If you need to, use the "Zoom to Layer Extent" or other zoom tools to zoom to the bounds of your layer.

You can now use the identify tool to inspect a feature in your layer, or perform any other function that you can normally use to work with Vector Layers in ArcMap.

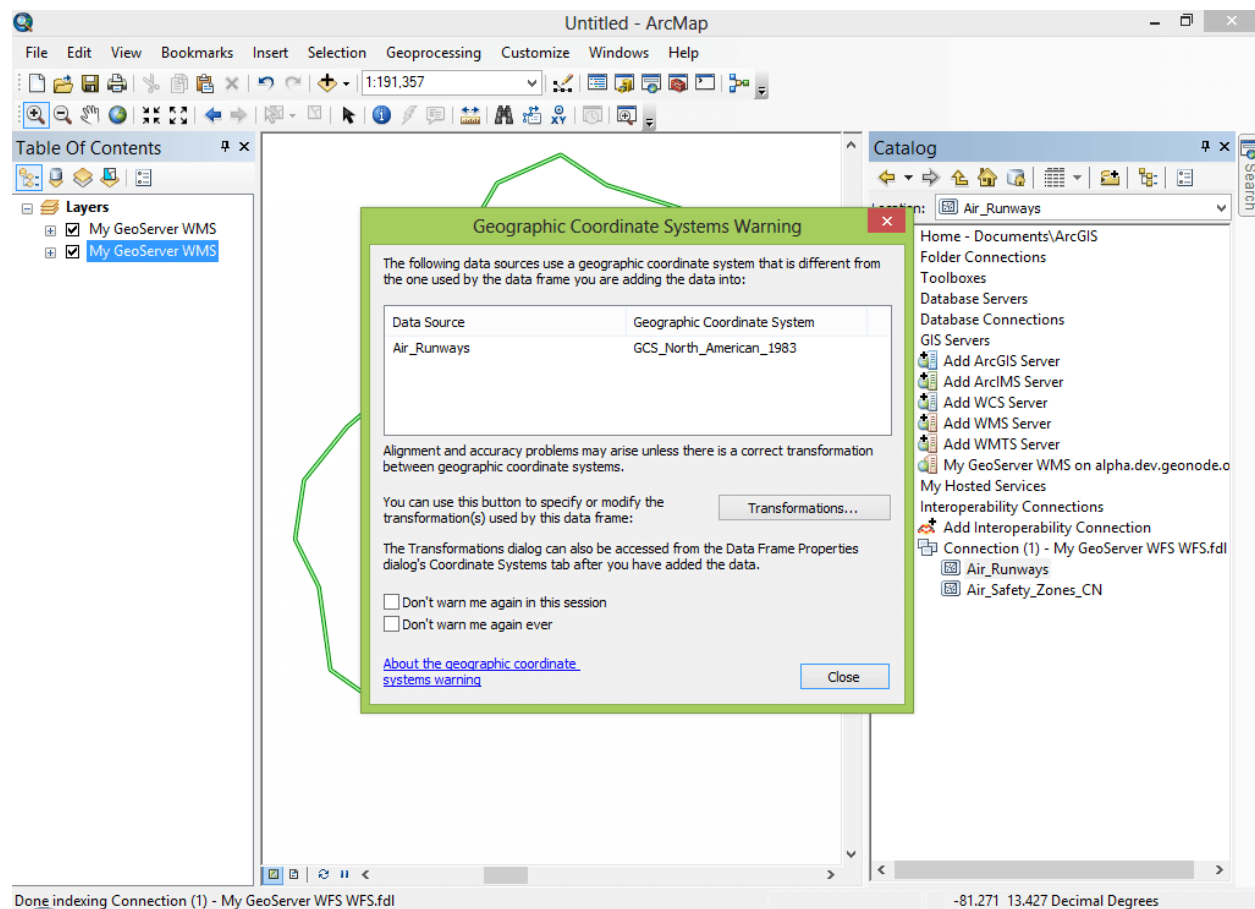
Since your layer was imported as actual vector features, you can use normal ArcMap styling tools to style the layer to match how you want it to be displayed.

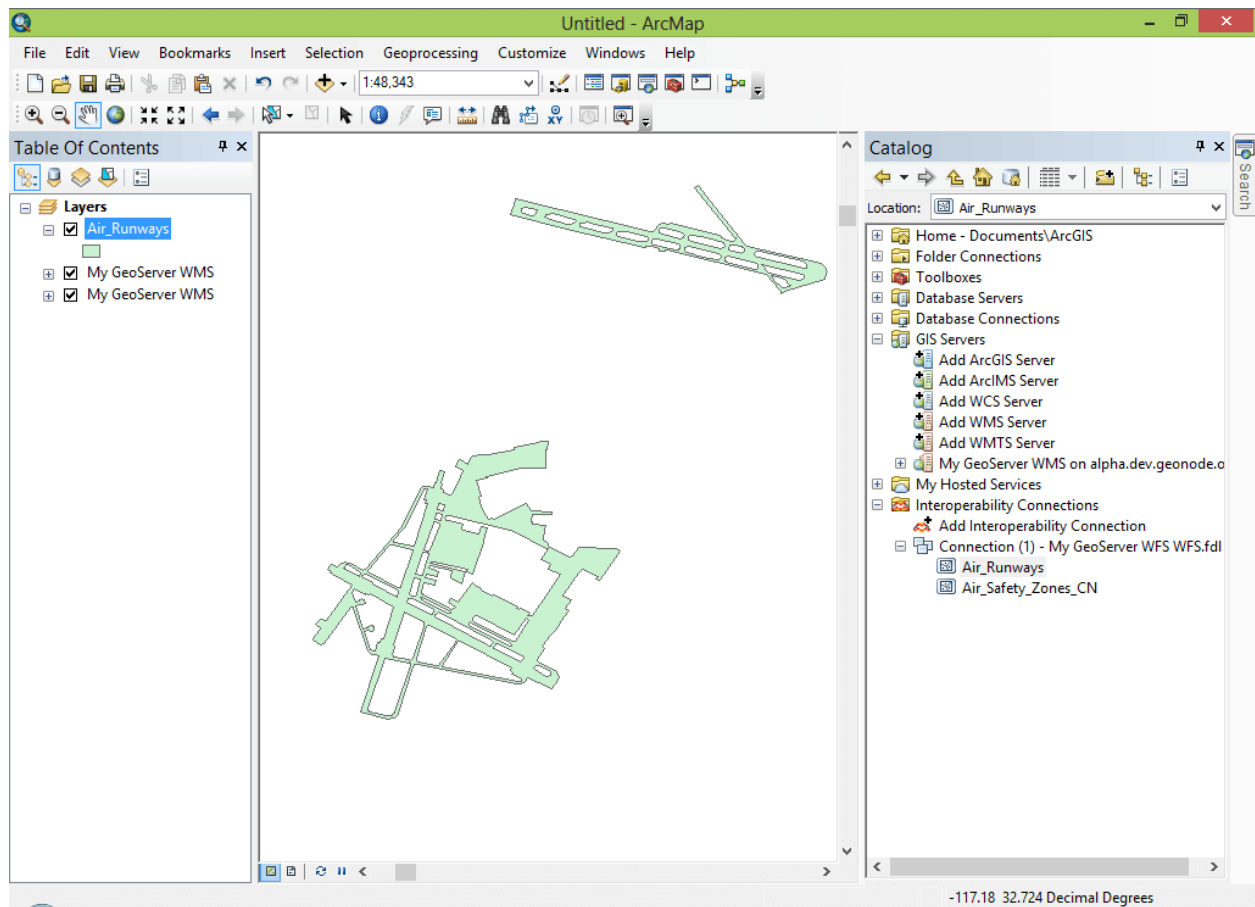
Now that you have added layers from your GeoNode as both WMS and WFS, you can explore the other options available to you with these layers within ArcMap.

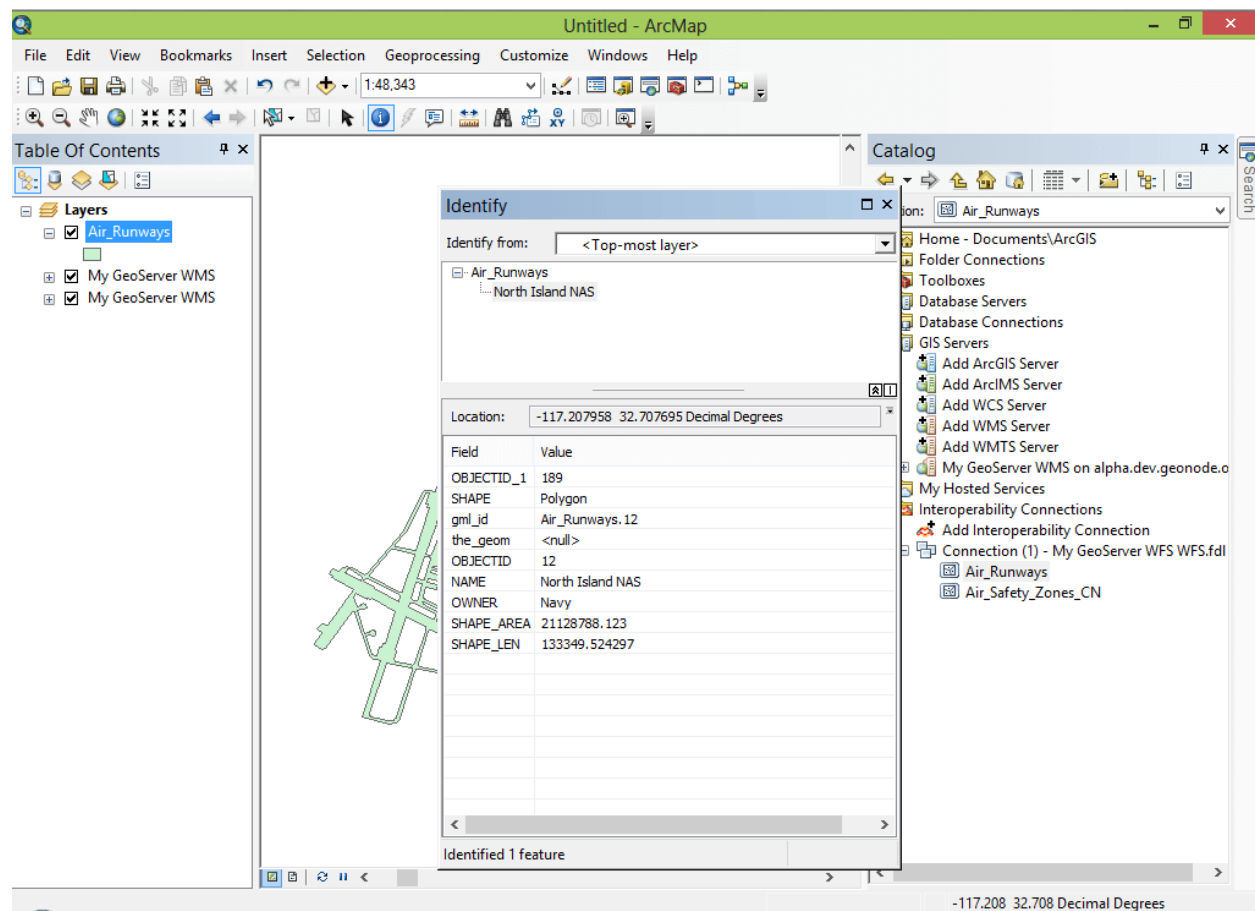
QGIS

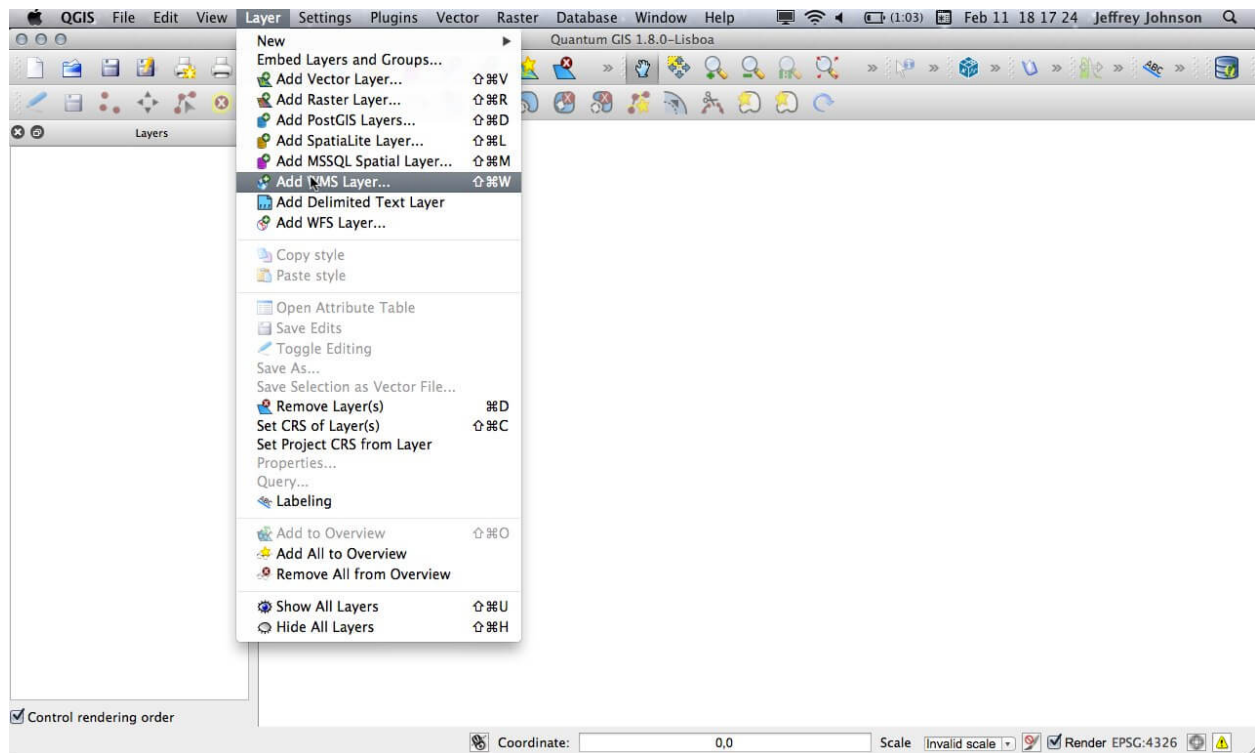
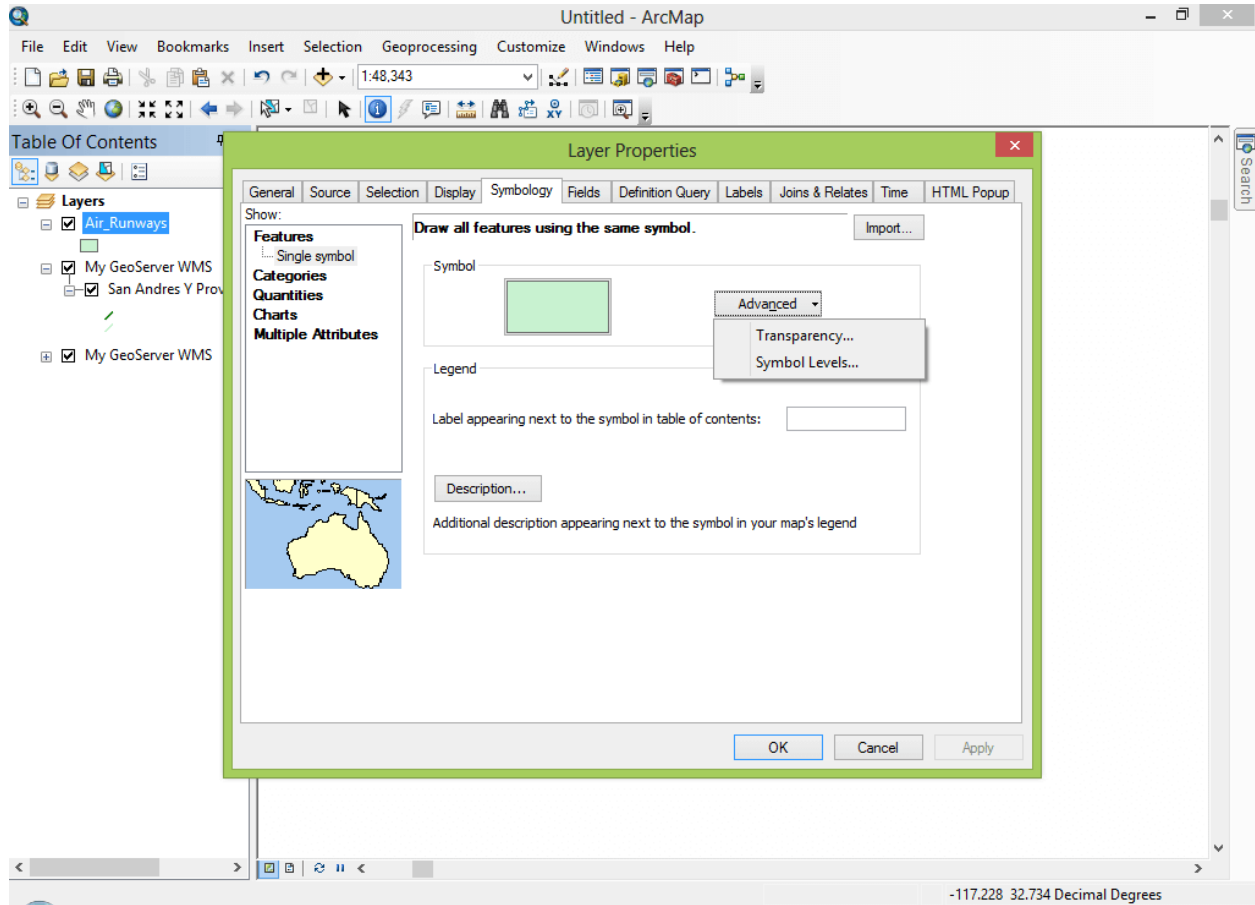
QGIS is an open source, cross platform desktop GIS app. It can also be used to add layers from your GeoNode instance as WMS or WFS. The process is very similar to how we add these same layers to ArcMap, and we will walk through the steps necessary in the following section.

First, select "Add WMS Layer" from the Layer menu.

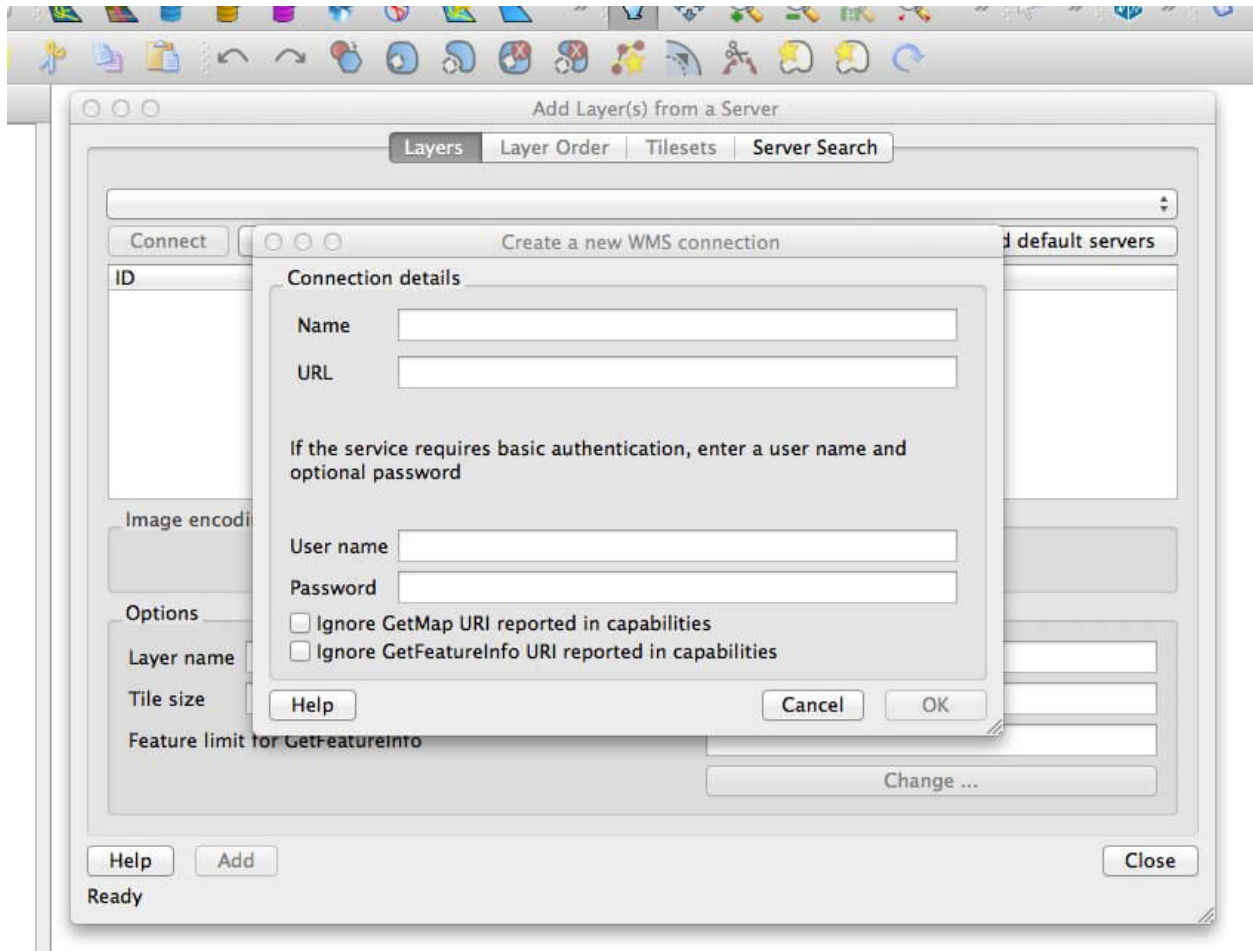








The Add WMS Layer Dialog will be displayed where you are able to specify the parameters to connect to your WMS server.



Next, you need to fill in the parameters to connect to your GeoNode instance. The URL for your GeoNode's WMS is the base URL + /geoserver/wms

After clicking the OK button, your server will show up in the list of servers. Make sure its selected, then, click the connect button to have QGIS retrieve the list of layers from your GeoNode.

Select the layers you want to add to your QGIS project and click "Add".

Your layer will be displayed in the map panel.

You can then zoom into your features in the Map.

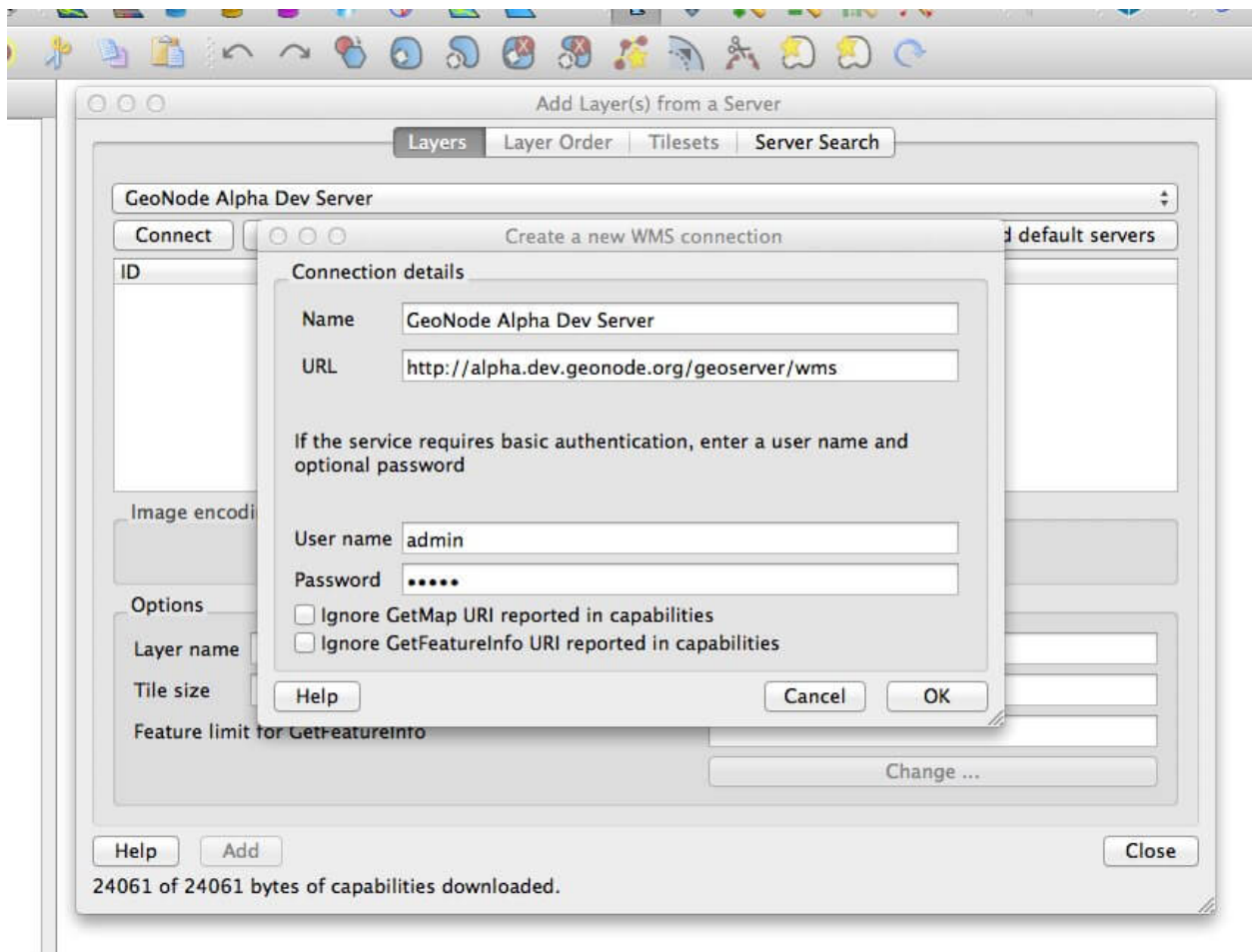
From there, you can use the identify tool to inspect the attributes of one of the features on the map.

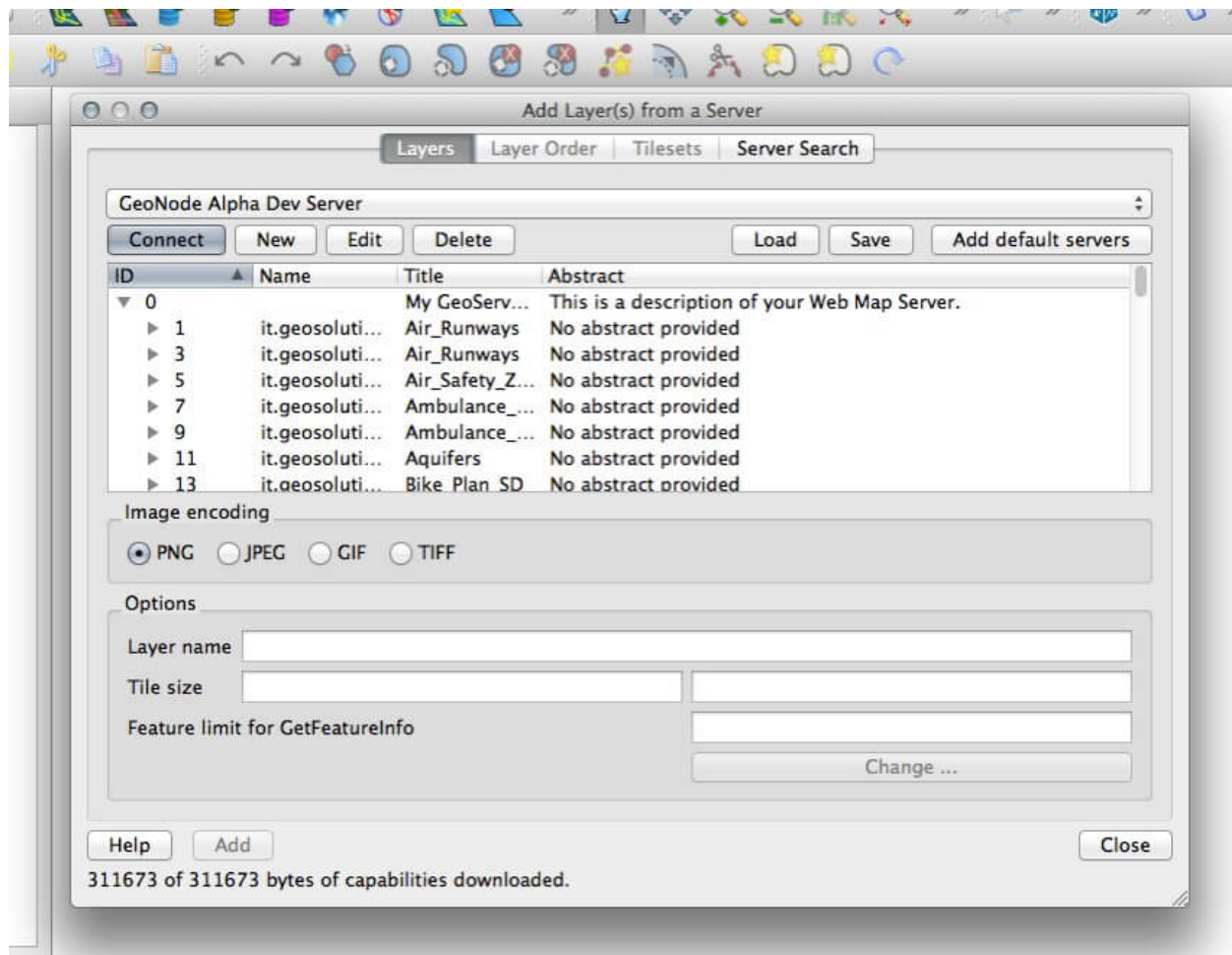
Or, you can look at the layer metadata by right clicking on the layer and selecting Layer Properties and selecting the metadata tab.

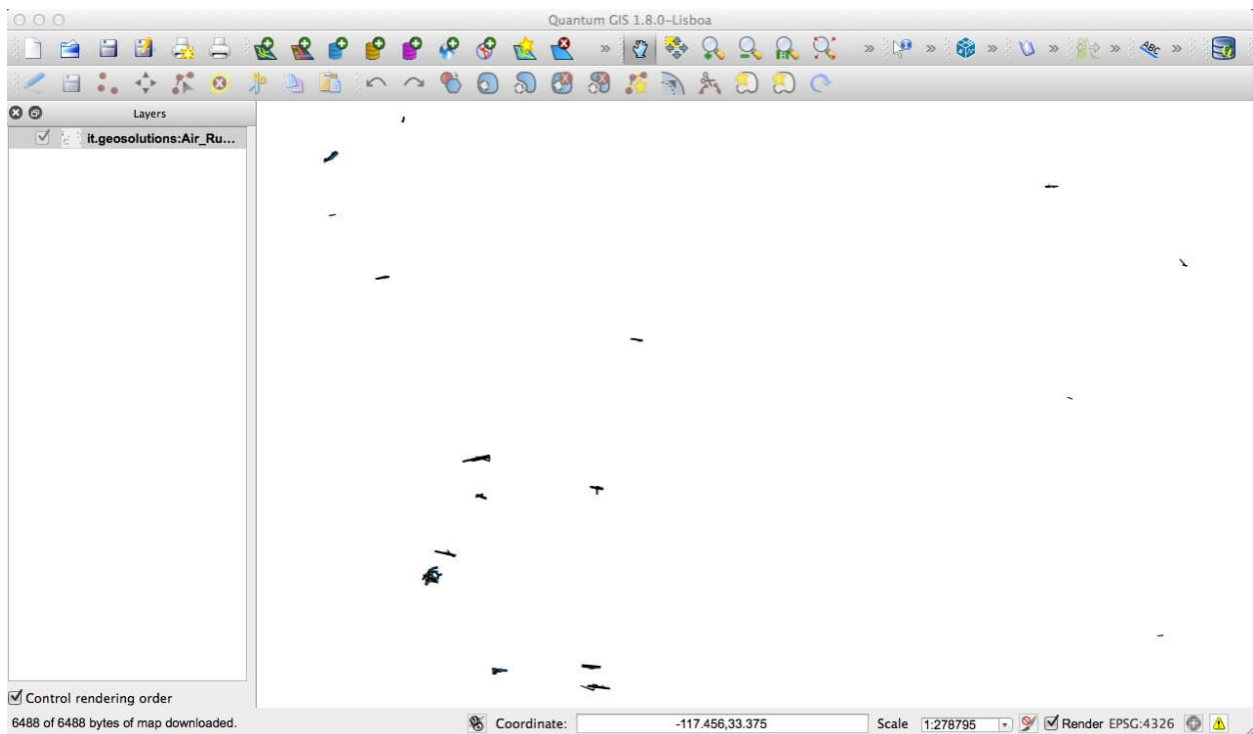
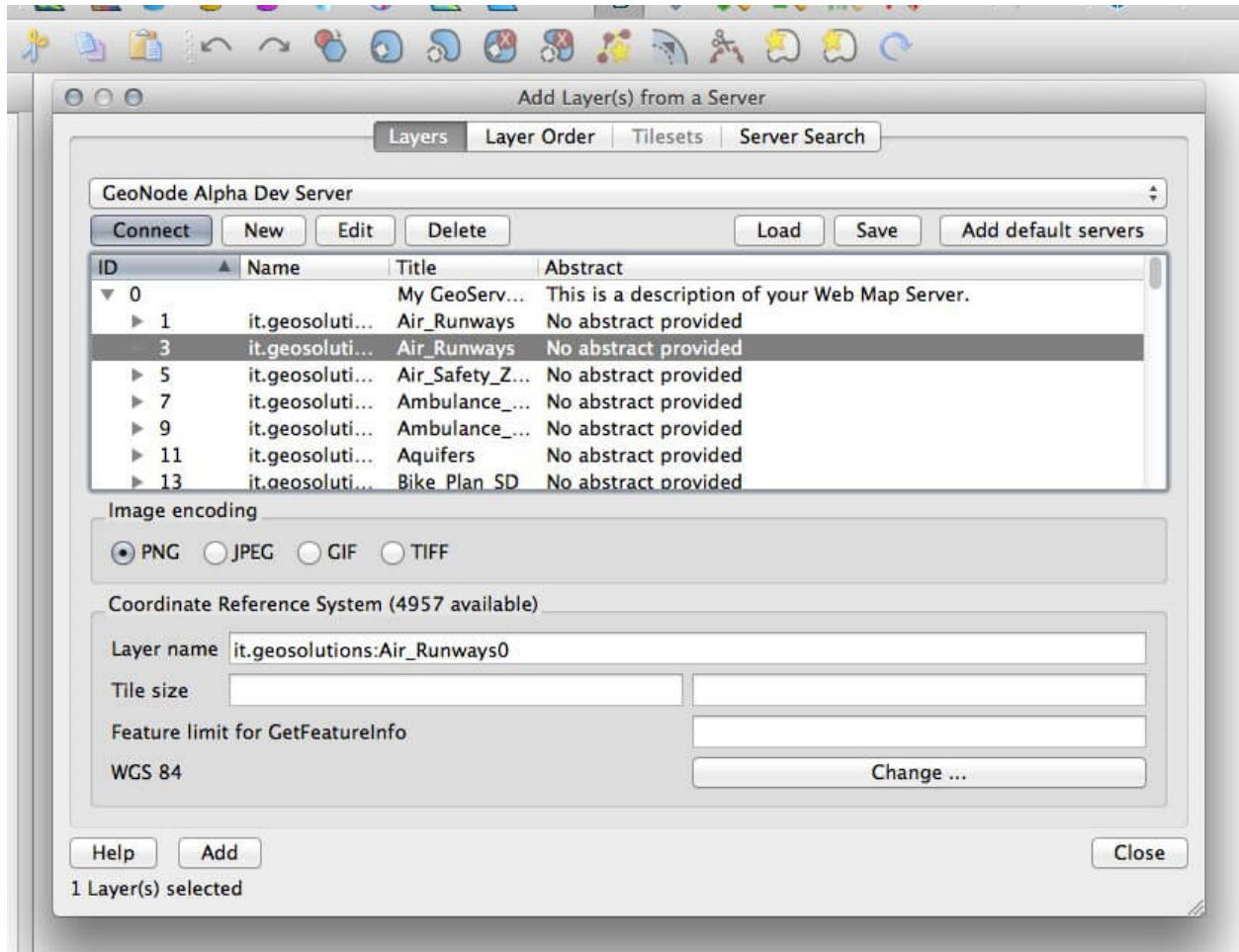
Adding WFS servers and layers to your QGIS project is very similar to adding WMS. Depending on your version of QGIS, you may need to add the WFS plugin. You can use the Plugin manager to add it.

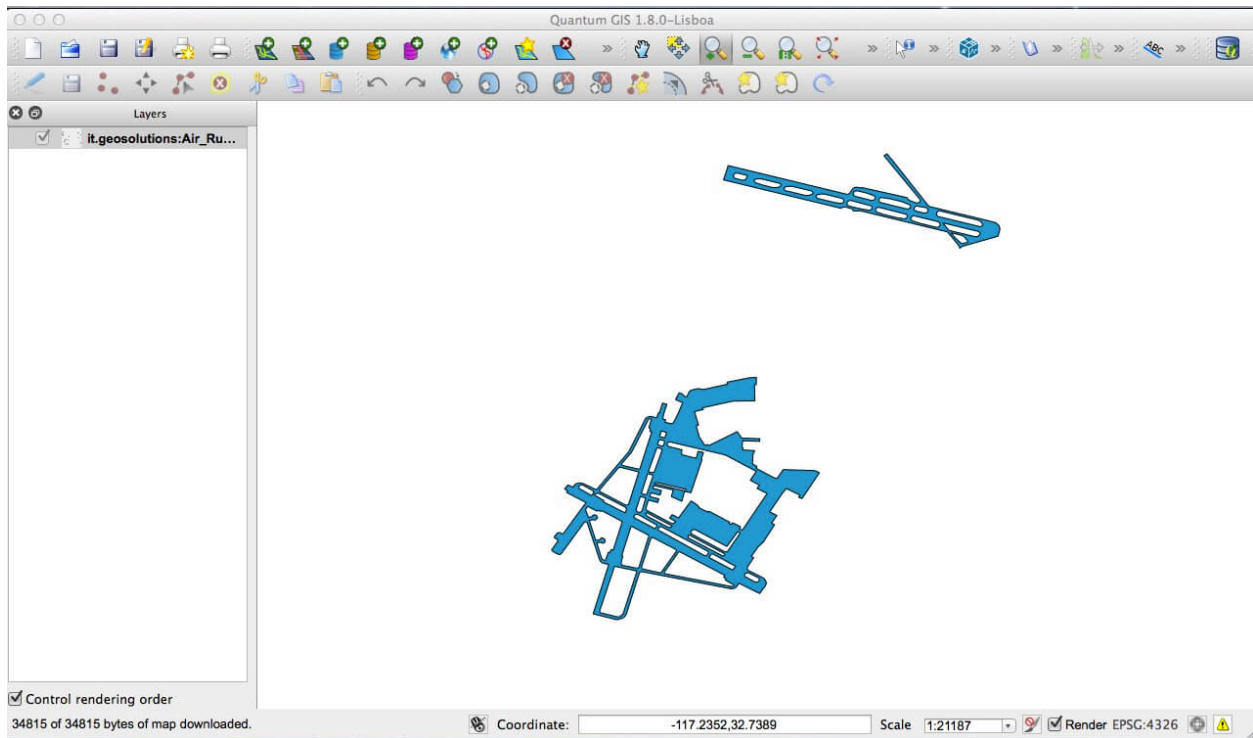
Once the plugin is installed, you can select the "Add WFS Layer" option from the Layer menu.

Step through the same process you did for WMS to create a new WFS connection. First specify server parameters and click OK.









Then click Connect to retrieve the list of layers on the server and select the layers you want to add and click Apply.

The layer(s) you selected will be displayed in the map panel.

You can use the same identify tool to inspect features in the map panel.

To look at more information about your layer, right click the layer in the Table of Contents and select Layer Properties. You can look at the list of fields.

... or set a style to match how you want your data to be displayed.

You now know how to add layers from your GeoNode instance to a QGIS project. You can explore all of the other options available to you in QGIS by consulting its documentation.

Google Earth

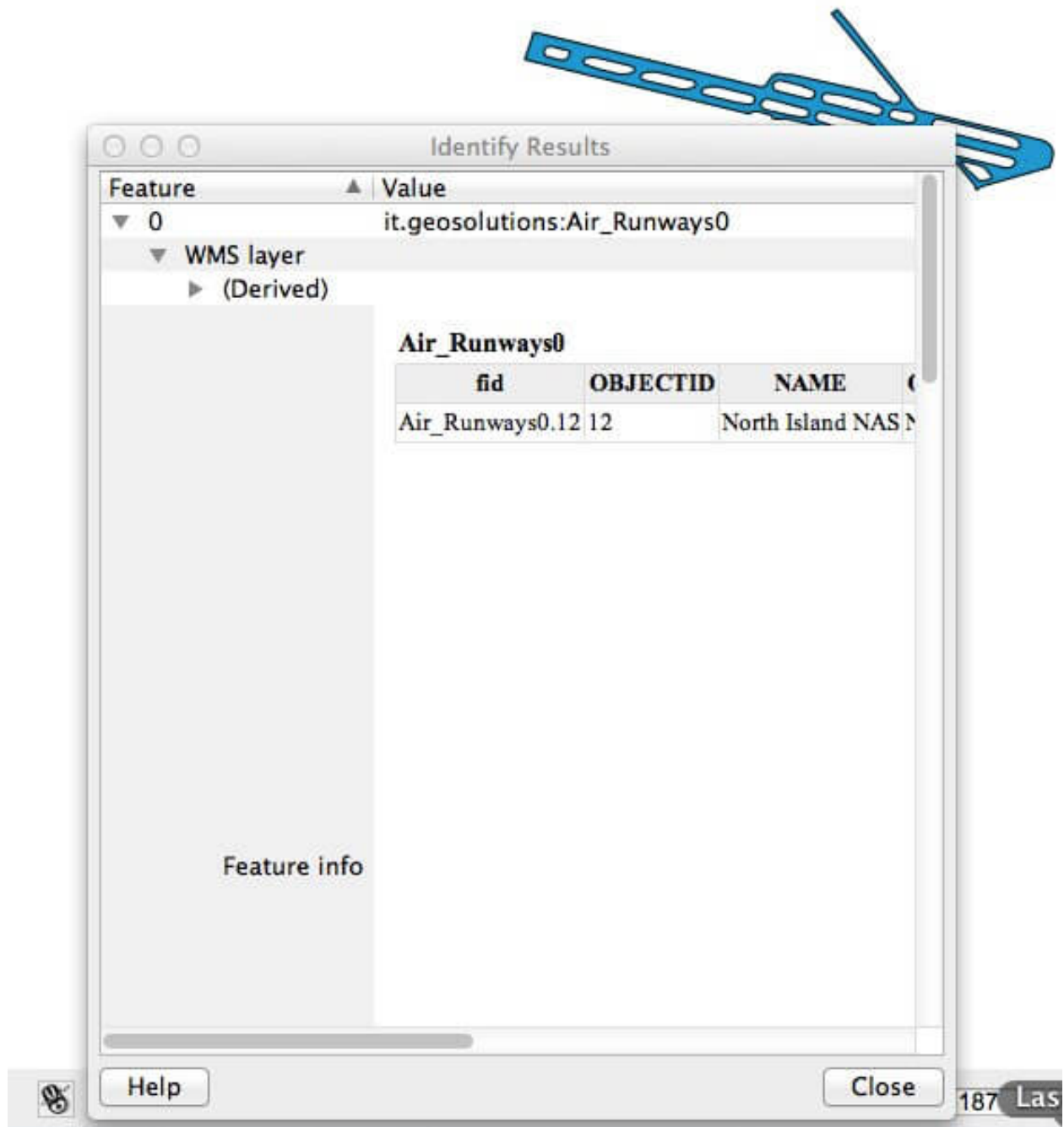
GeoNode's built in map interface lets you look at your layers and maps in the Google Earth plugin directly in your browser. You can switch to this 3D viewer directly in GeoNode by clicking the google earth icon in the map panel.

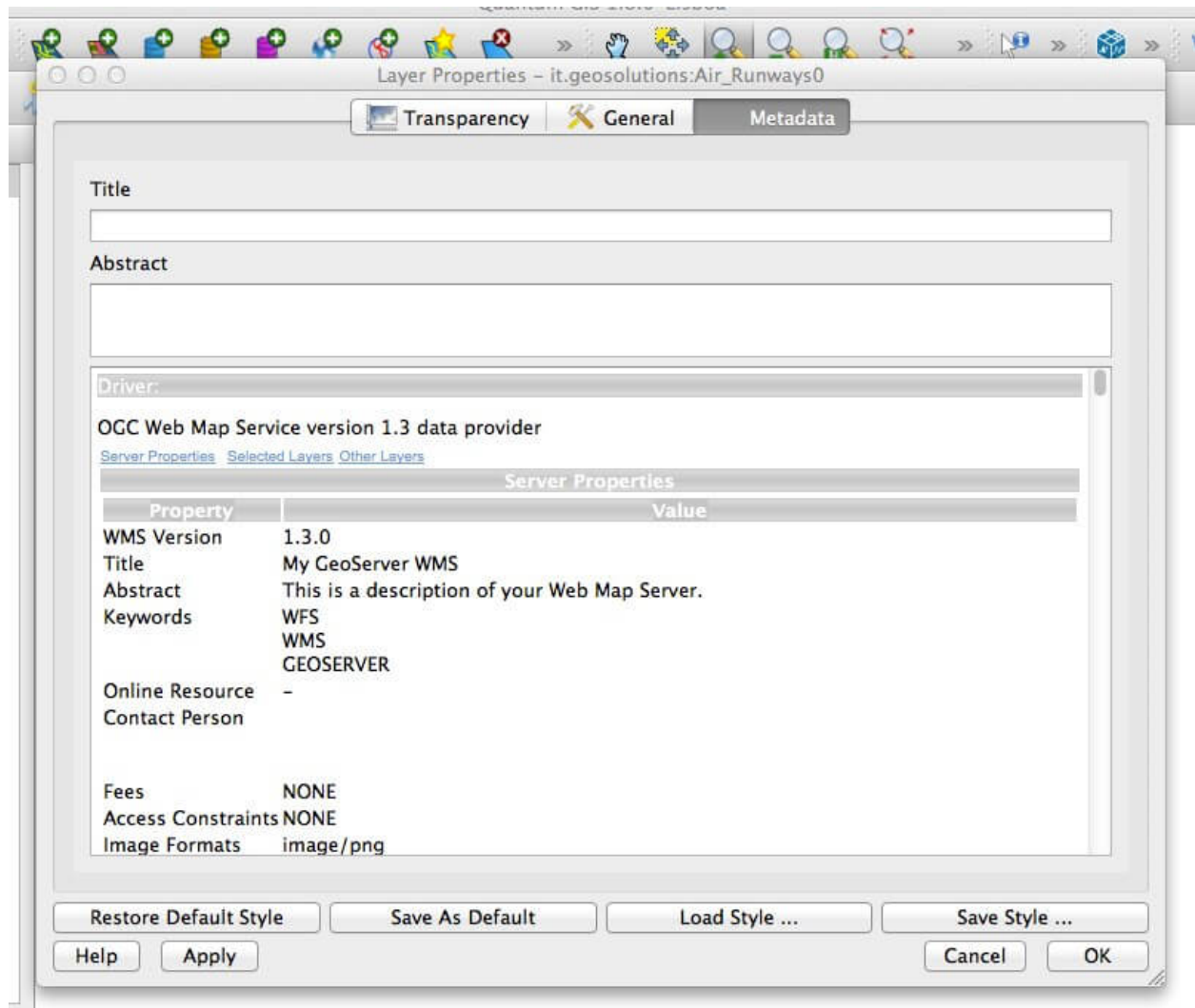
GeoServer will render your layer as an image until you are zoomed in sufficiently, and then it will switch to rendering it as a vector overlay that you can click on to view the attributes for the feature you clicked on.

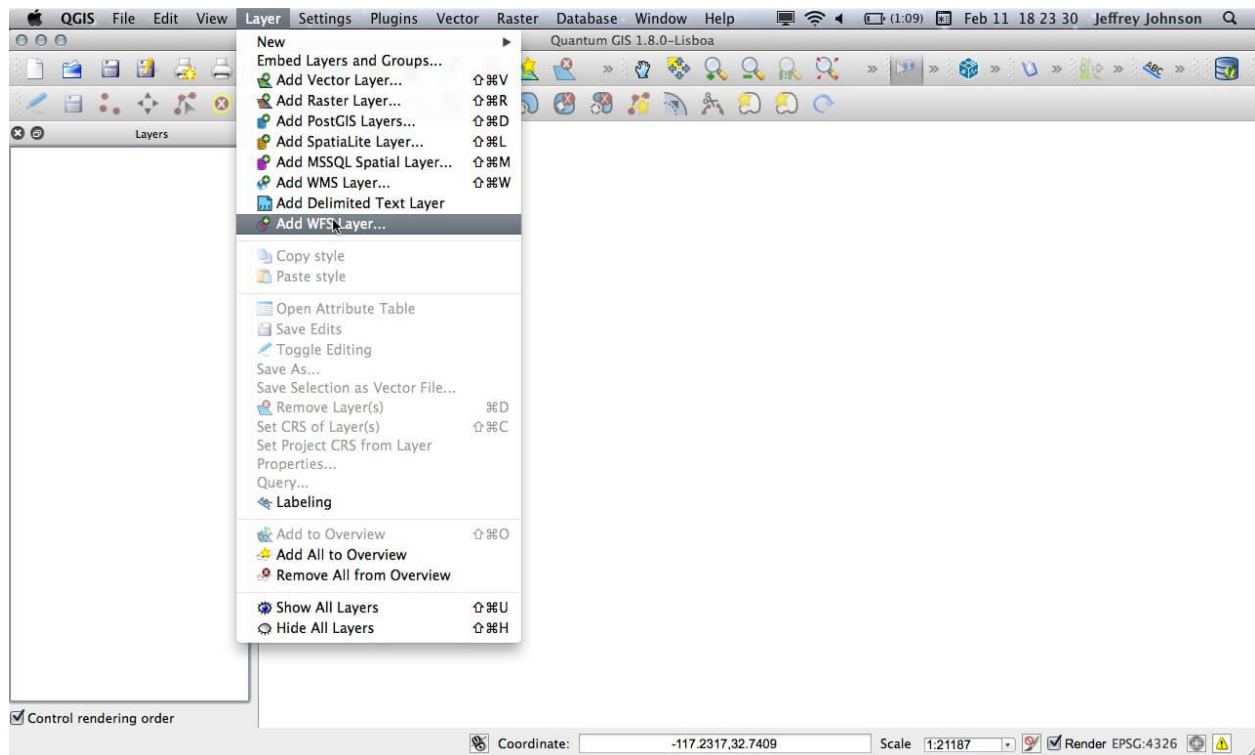
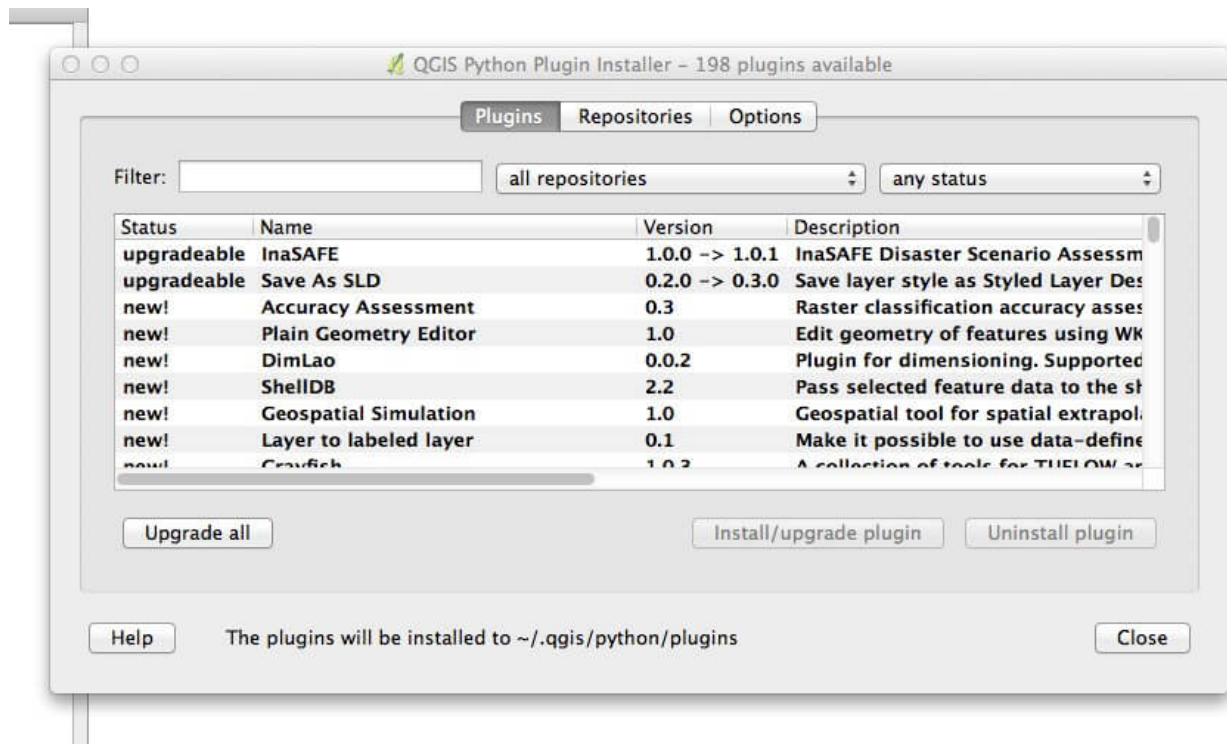
You can also use this option in the GeoExplorer client by clicking the same button.

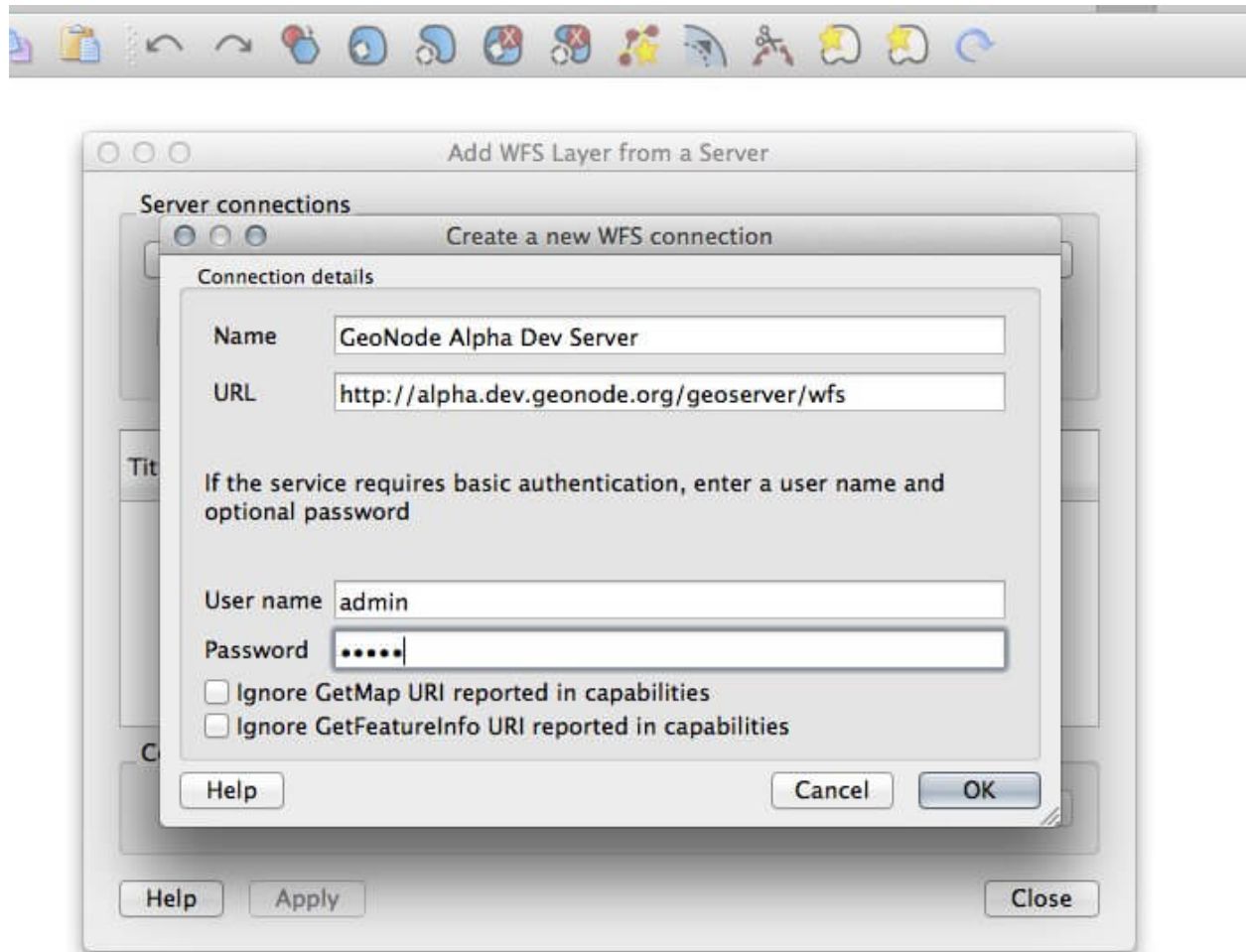
Note: Some of the GeoExplorer options will not be available to you when you are in this mode, they will be grayed out or inaccessible.

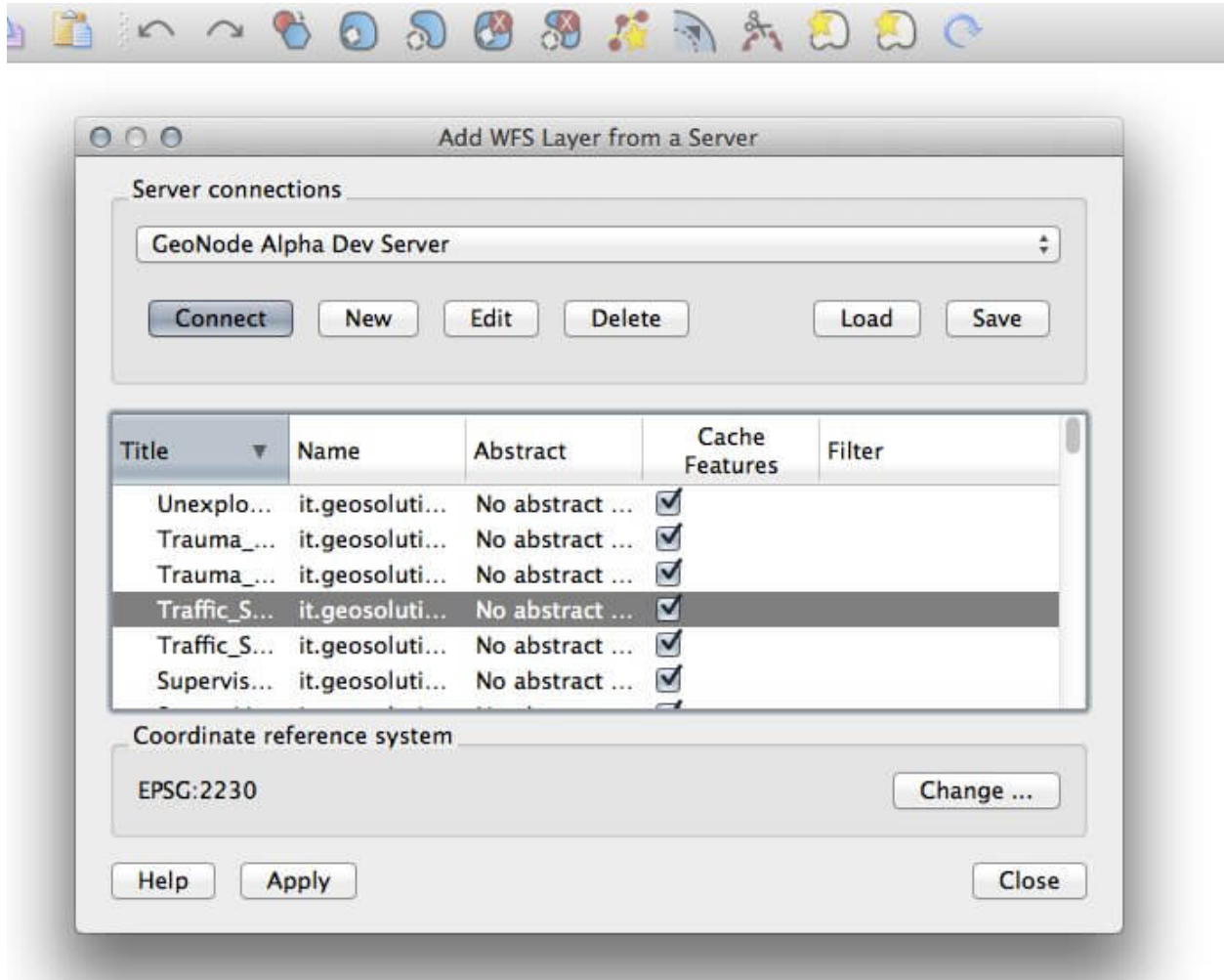
If instead you want to use layers from your GeoNode in the Google Earth client itself, you have a few options available to you.

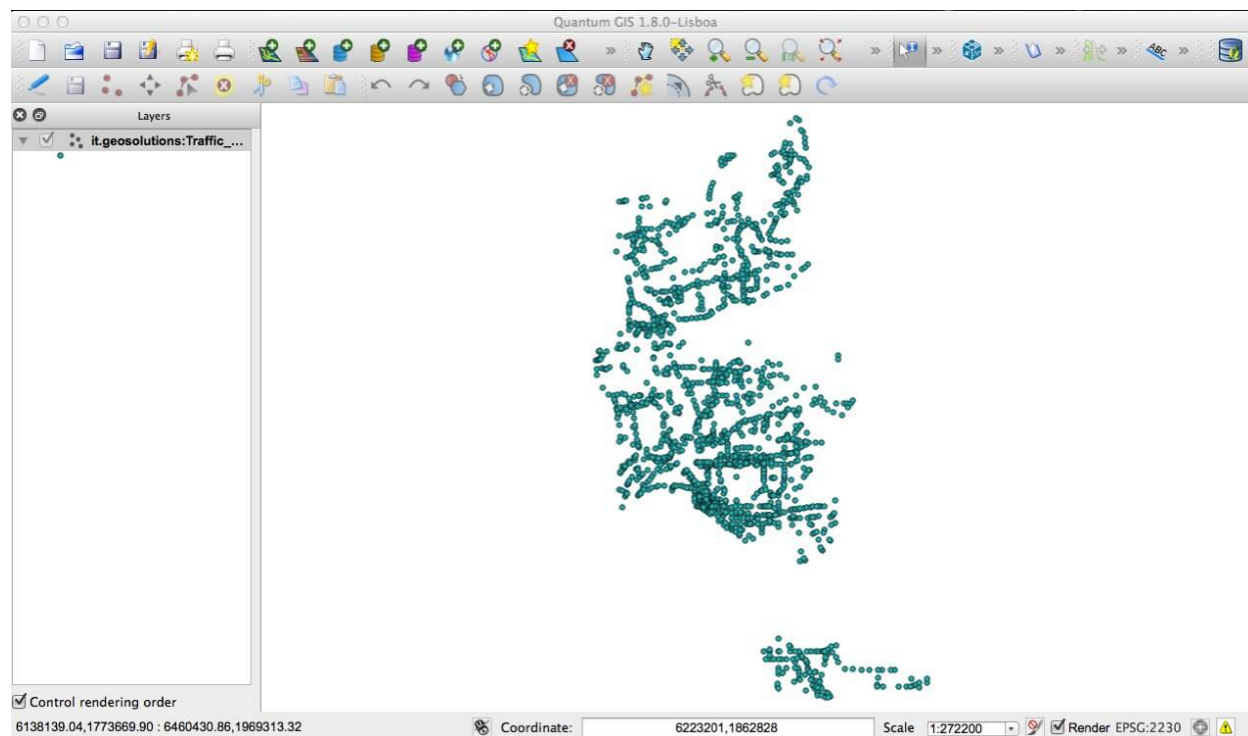












First, you can select the KML option from the Download Layer menu to download the entire layer in a single KML file. Depending on the size of the layer, your GeoNode could take several seconds or longer to generate this KML and return it to you.

When the layer is generated, it will be downloaded to your desktop machine and you can simply double click it to open it in Google Earth.

Alternatively, you can use the “View in Google Earth” option in the Layer Download menu to view the layer in Google Earth using the same methodology described above depending on the zoom level.

This will download a small KMZ to your desktop that contains a reference to the layers on the server and you can double click it to open it in Google Earth.

Note: The basic difference between these two options is that the first downloads *all* of the data to your desktop at once and as such, the downloaded file can be used offline while the second is simply a Network Link to the layer on the server. Choose whichever method is best for your own needs and purposes.

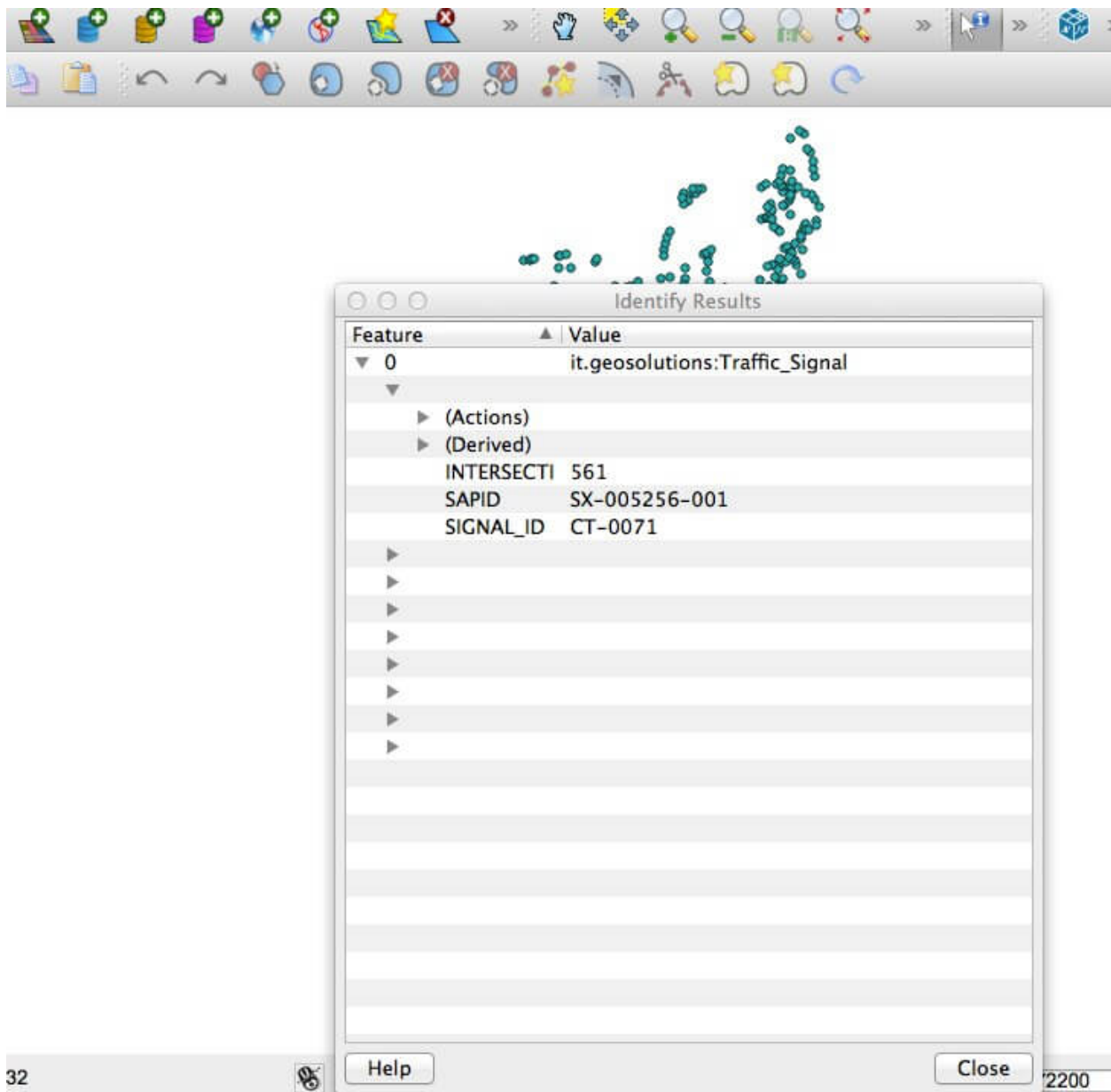
Once you have added your layers to the Places panel in Google Earth, you can move them from the Temporary Places section into My Places if you wish to use them after your current Google Earth session is complete. You can arrange them in folders and use Google Earth functionality to save your project to disk. Consult Google Earths documentation for more information about how to do this.

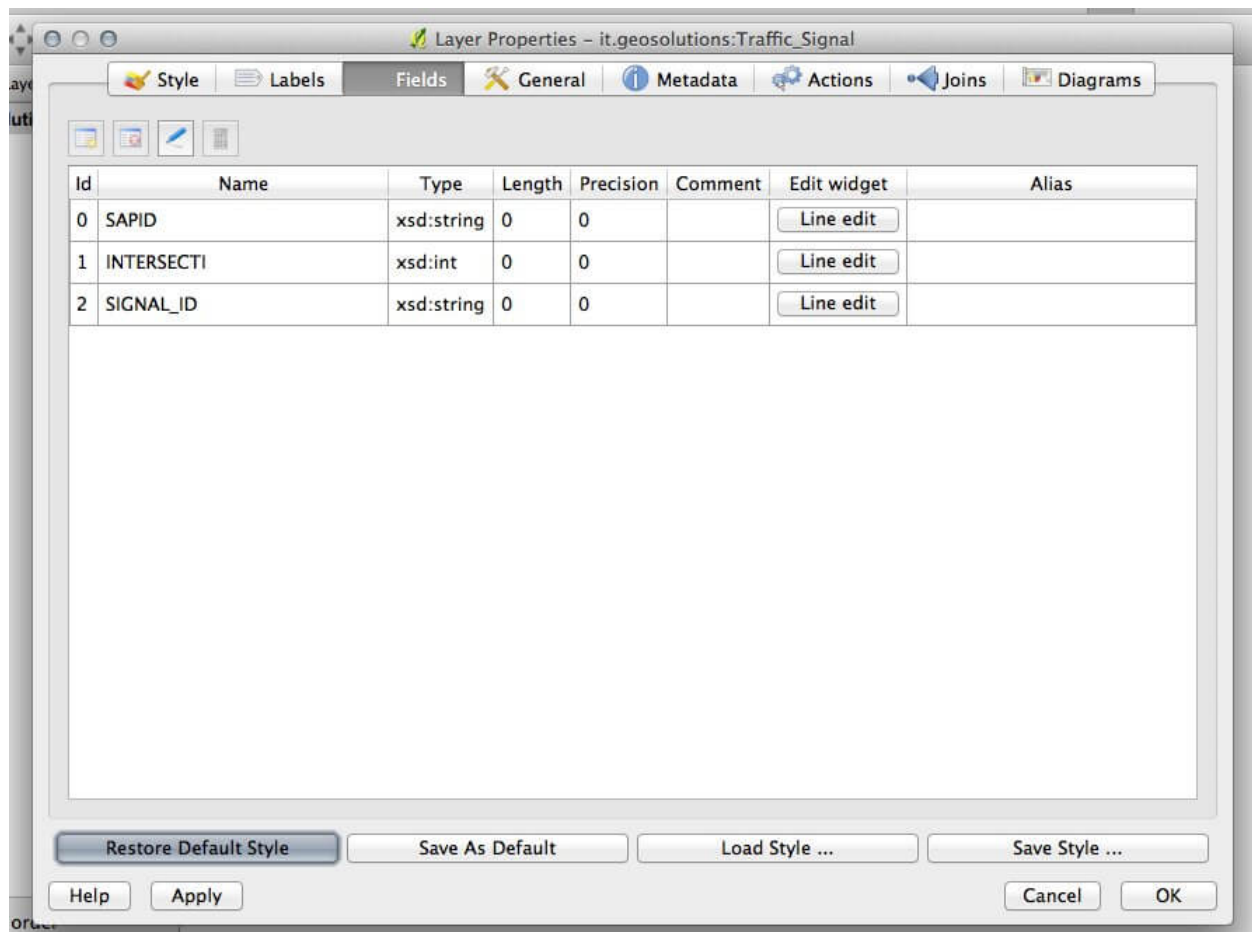
Accounts and users GeoNode is primarily a *social* platform, and thus a primary component of any GeoNode instance is the user account. This section will guide you through account registration, updating your account information, and viewing other user accounts.

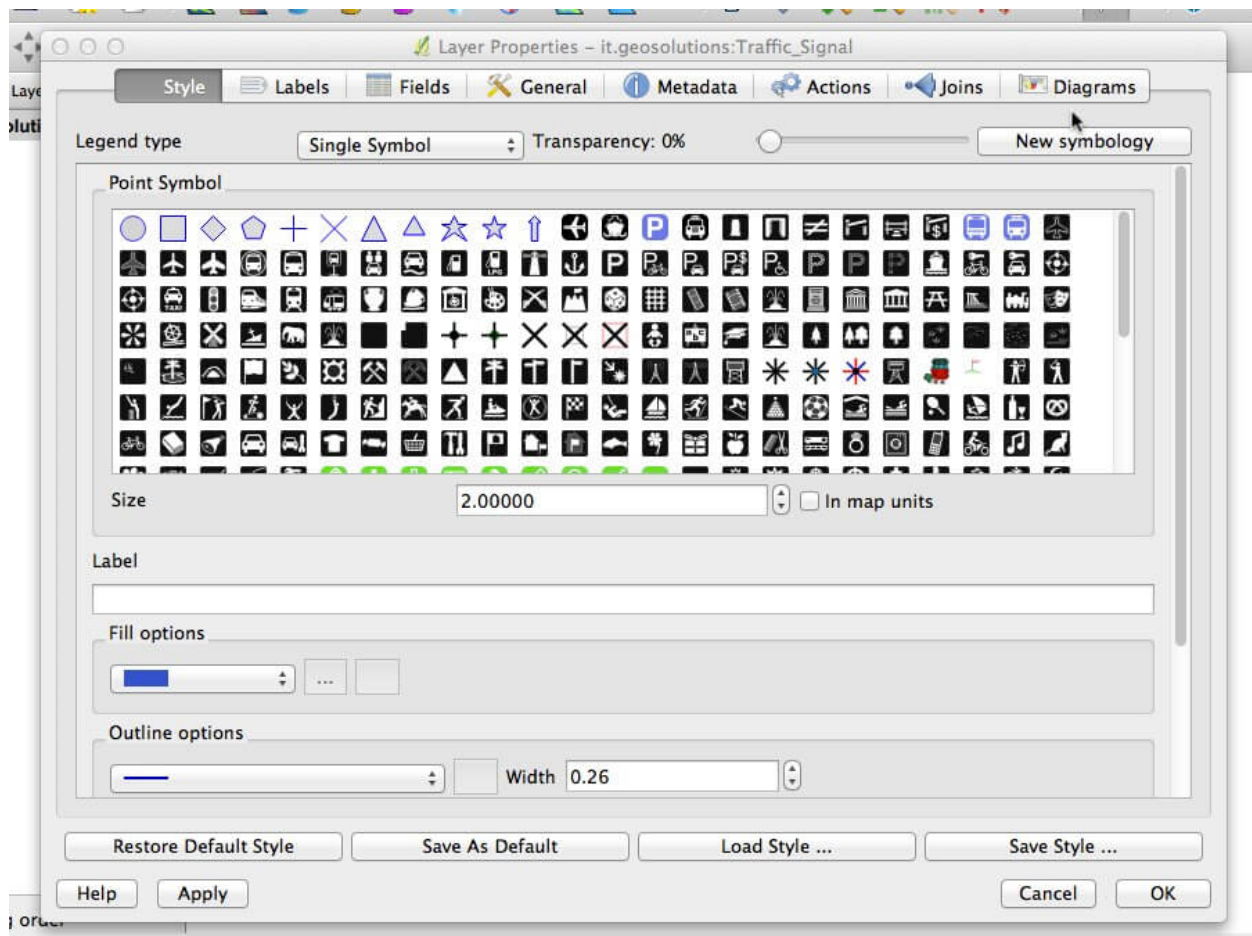
Document Types GeoNode welcome page shows a variety of information about the current GeoNode instance. At the top of the page is a toolbar showing quick links to document types: layers, maps and documents.

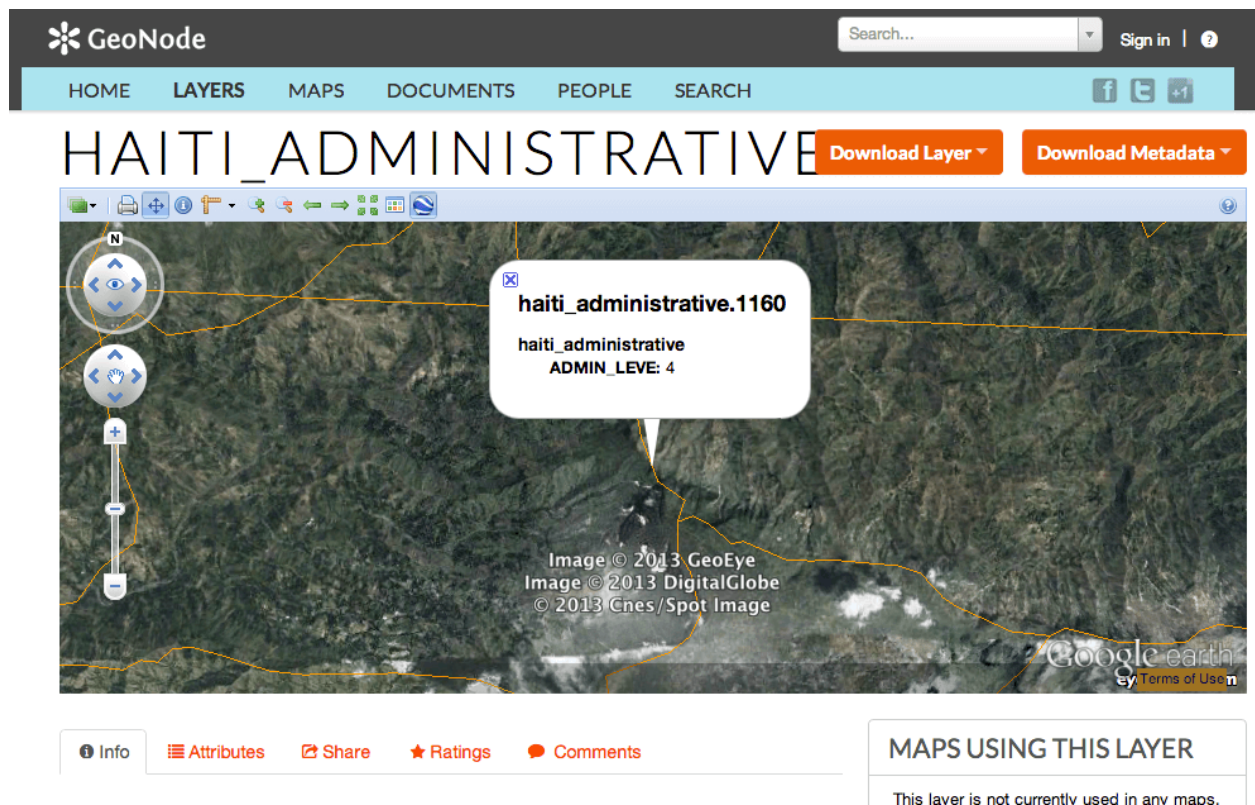
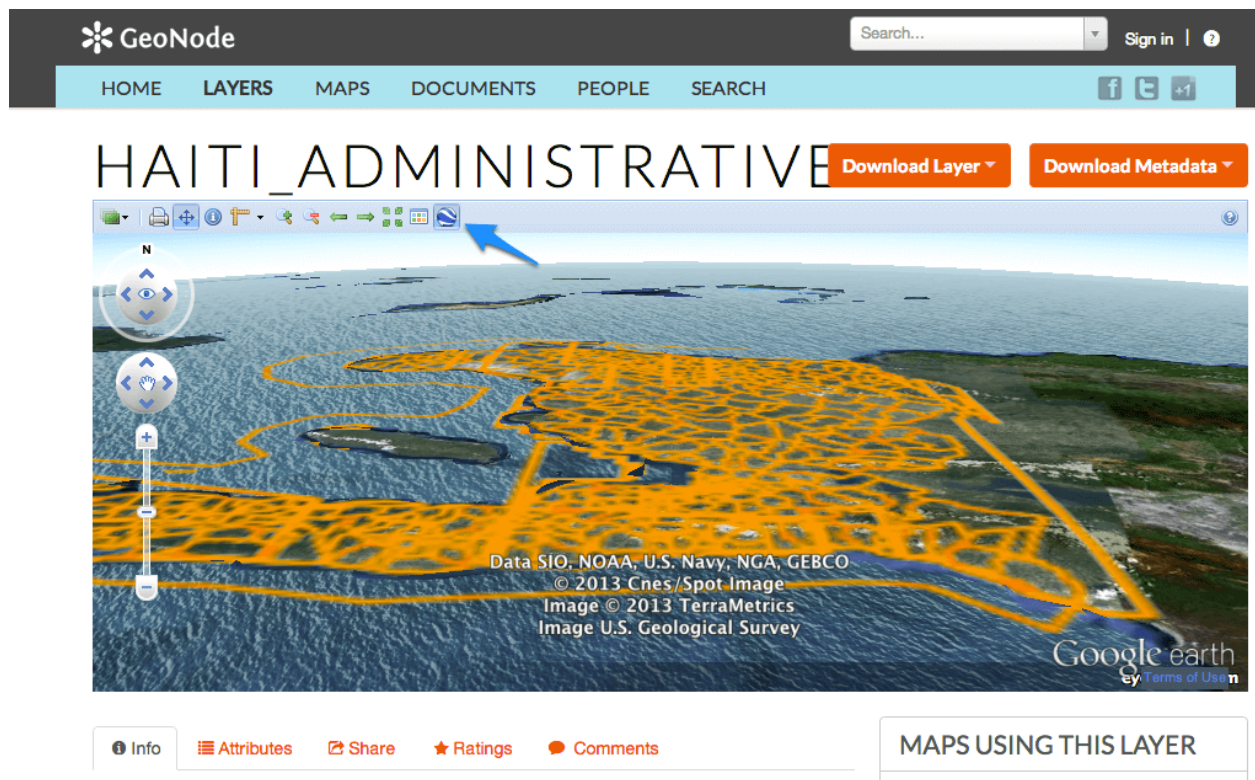
Searching GeoNode advanced Search tool.

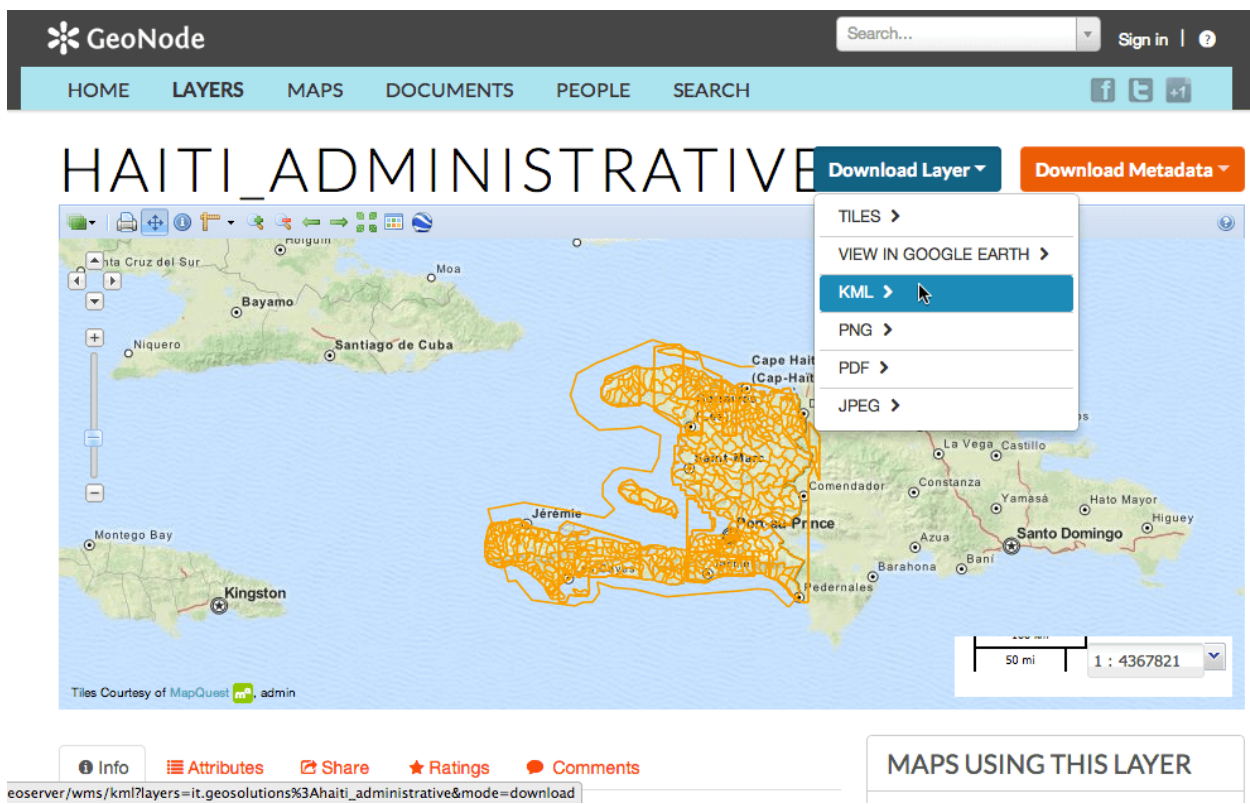
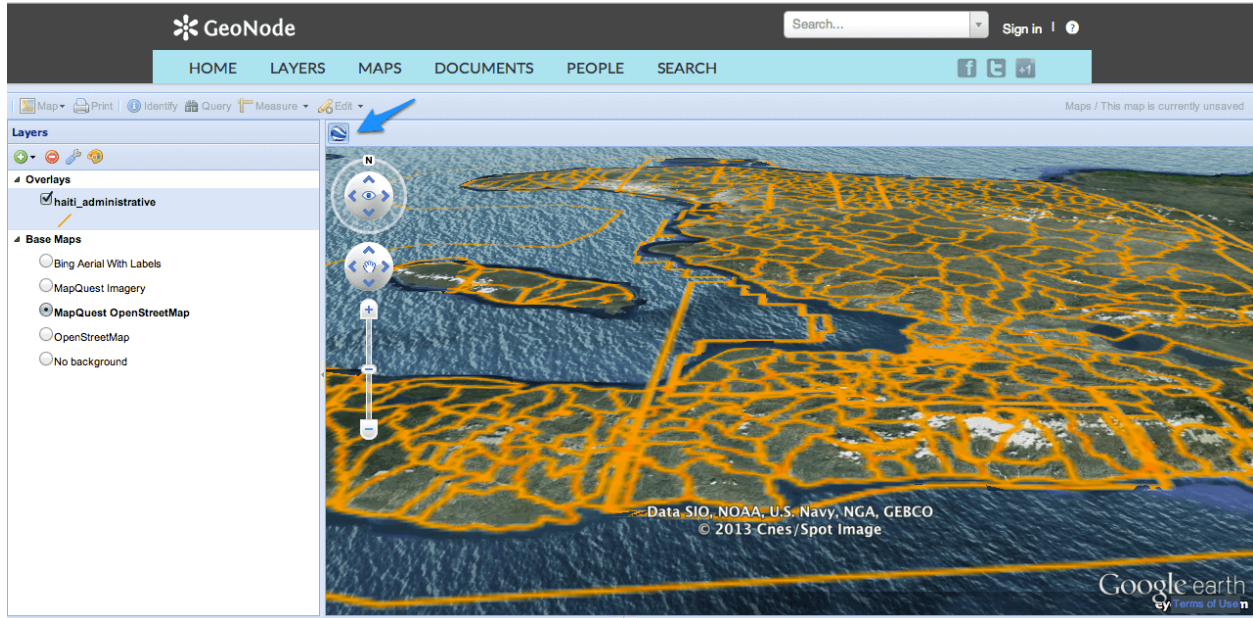
Managing layers Create, delete, manage and share Layers on GeoNode.

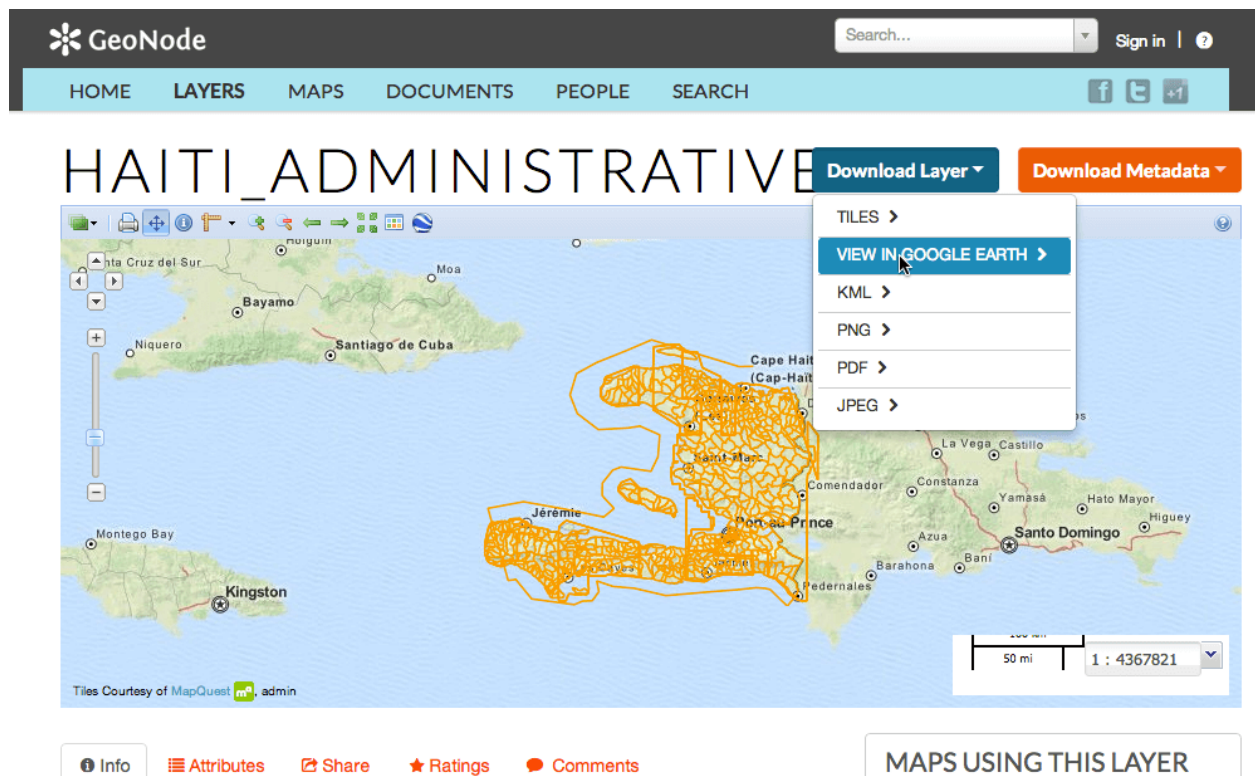
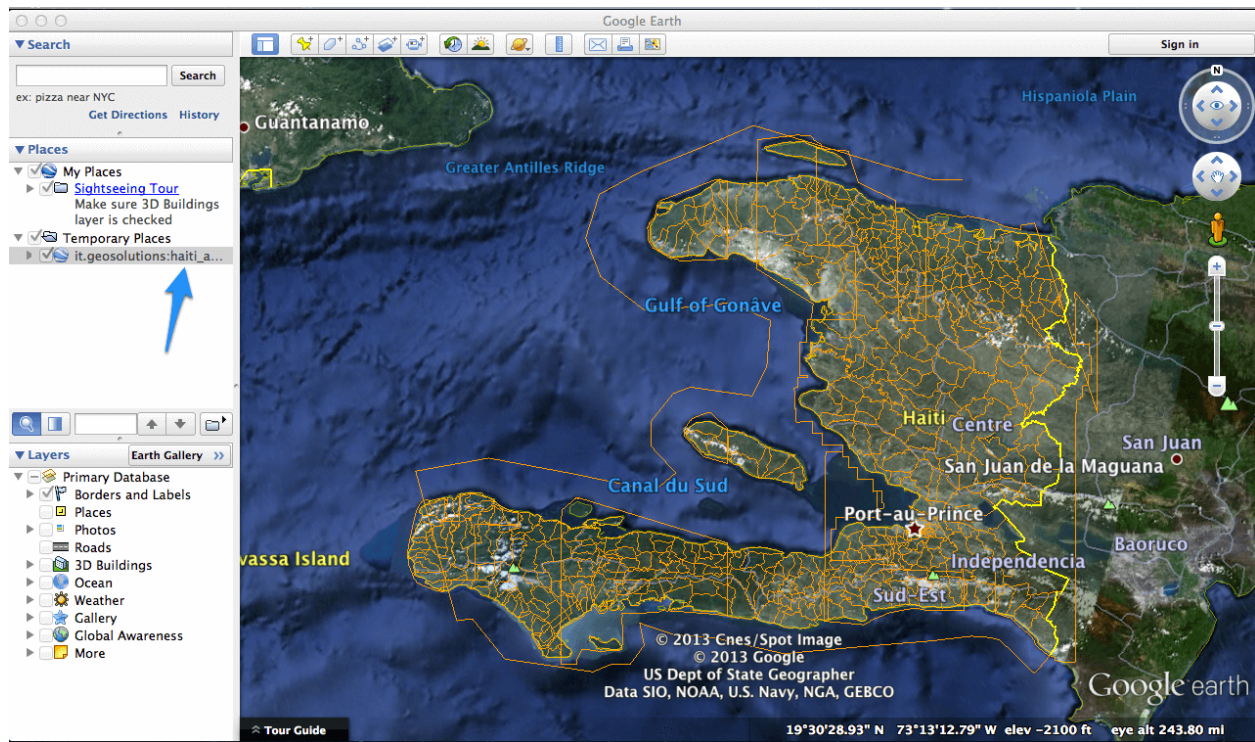


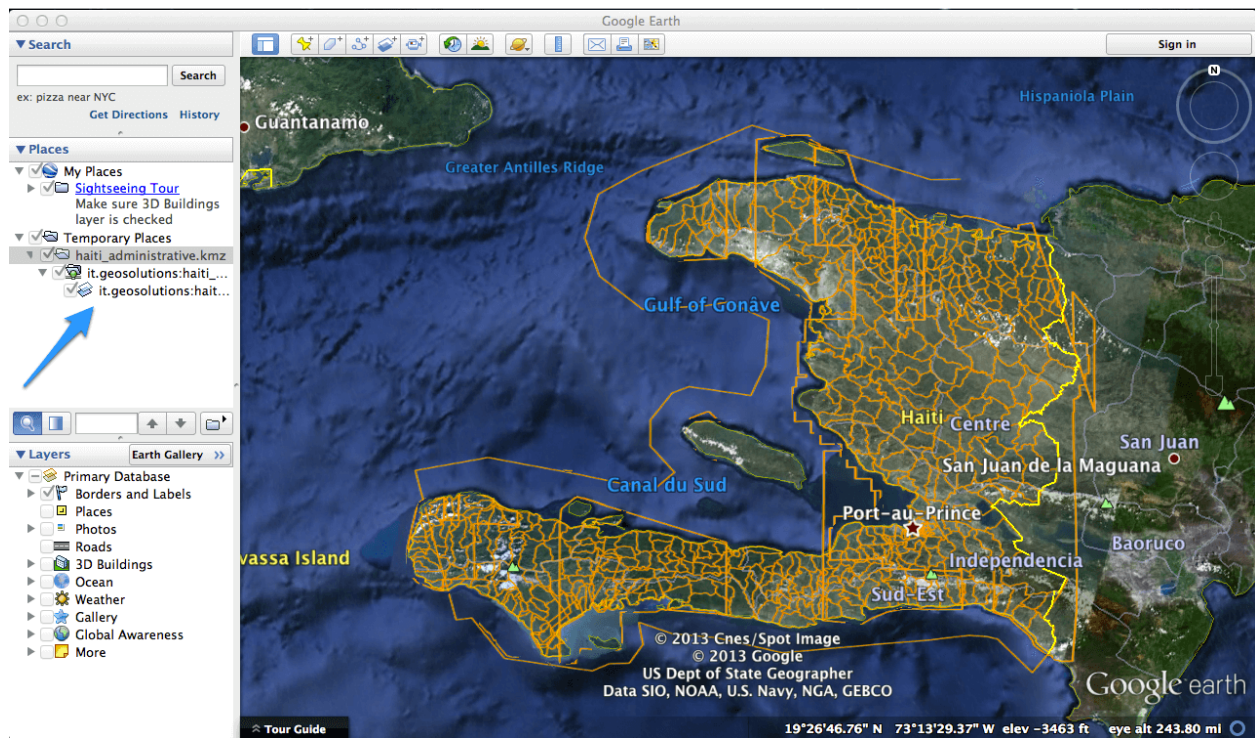












Edit Layer Style Beautify the Layer using the GeoNode Style editor.

Managing maps Create, delete, manage and share Maps on GeoNode.

Using GeoNode with other applications Your GeoNode project is based on core components which are interoperable and as such, it is straightforward for you to integrate with external applications and services. This section will walk you through how to connect to your GeoNode instance from other applications and how to integrate other services into your GeoNode project. When complete, you should have a good idea about the possibilities for integration, and have basic knowledge about how to accomplish it. You may find it necessary to dive deeper into how to do more complex integration in order to accomplish your goals, but you should feel comfortable with the basics, and feel confident reaching out to the wider GeoNode community for help.

3.2.1.4 Administrators Workshop

Welcome to the GeoNode Training *Administrators Workshop* documentation v2.8.

This workshop will teach how to install and manage a deployment of the [GeoNode](#) software application. At the end of this section you will master all the GeoNode sections and entities from an administrator perspective.

You will know how to:

1. Use the GeoNode's Django Administration Panel.
2. Use the console Management Commands for GeoNode.
3. Configure and customize your GeoNode installation.

Prerequisites

Before proceeding with the reading, it is strongly recommended to be sure having clear the following concepts:

1. GeoNode and Django framework concepts

2. Good knowledge of Python
3. Good knowledge of what is a geospatial server and geospatial web services.
4. Good knowledge of what is metadata and catalog.
5. Good knowledge of HTML and CSS.

3.2.1.4.1 GeoNode and GeoServer Advanced Security

GeoNode interacts with GeoServer through an advanced security mechanism based on OAuth2 Protocol and GeoFence. This section is a walk through of the configuration and setup of GeoNode and GeoServer Advanced Security.

What we will see in this section is:

- **Introduction**
- **GeoNode (Security Backend):**
 1. Django Authentication
 2. Django OAuth Toolkit Setup and Configuration
 3. Details on `settings.py` Security Settings
- **GeoServer (Security Backend):**
 1. GeoServer Security Subsystem
 2. Introduction to the GeoServer OAuth2 Security Plugin
 3. Configuration of the `GeoNode REST Role Service`
 4. Configuration of the `GeoNode OAuth2 Authentication Filter`
 5. The GeoServer Authentication Filter Chains
 6. Introduction to GeoFence Plugin, the Advanced Security Framework for GeoServer
- **Troubleshooting and Advanced Features:**
 1. Common Issues and Fixes
 2. How to setup HTTPS secured endpoints
 3. GeoFence Advanced Features

Introduction

GeoServer, i.e. the geospatial backend server of GeoNode, is a spatial server which needs authenticated users in order to access protected resources or administration functions.

GeoServer supports several kind of Authentication and Authorization mechanisms. Those systems are pluggable and GeoServer can use them at the same time by the use of a `Filter Chain`. Briefly this mechanism allows GeoServer to check for different A&A protocols one by one. The first one matching is used by GeoServer to authorize the users.

GeoNode Authentication is based by default on Django Security Subsystem. Django authentication allows GeoNode to manage its internal users, groups, roles and sessions.

GeoNode has some external components, like GeoServer or QGIS Server, which are pluggable and stand-alone services, devoted to the management of geospatial data. Those external services have their own authentication and authorization mechanisms which must be synchronized somehow with the GeoNode one. Also, those external services maintain, in most of the cases and unless specific configuration does not disable this, alternative security access

which for instance allow GeoNode to modify the geospatial catalog under the hood, or a system administrator to have independent and privileged access to the servers.

Before going deeply on how GeoServer/GeoNode A&A works and how it can be configured in order to work correctly with GeoNode, let's quickly clarify the difference between the `Authentication` and `Authorization` concepts.

Authentication

Authentication is the process of verifying the identity of someone through the use of some sort of credentials and an handshake protocol. If the credentials are valid, the authorization process starts. Authentication process always proceeds to Authorization process (although they may often seem to be combined). The two terms are often used synonymously but they are two different processes.

For more details and explanation about the authentication concepts, take a look [here](#).

Authorization

Authorization is the process of allowing authenticated users to access protected resources by checking its roles and rights against some sort of security rules mechanism or protocol. In other words it allows to control access rights by granting or denying specific permissions to specific authorized users.

GeoNode Security Backend

Django Authentication

The Django authentication system handles both authentication and authorization.

The auth system consists of:

1. Users
2. Permissions: Binary (yes/no) flags designating whether a user may perform a certain task.
3. Groups: A generic way of applying labels and permissions to more than one user.
4. A configurable password hashing system
5. Forms and view tools for logging in users, or restricting content
6. A pluggable backend system

The authentication system in Django aims to be very generic and doesn't provide some features commonly found in web authentication systems. Solutions for some of these common problems have been implemented in third-party packages:

1. Password strength checking
2. Throttling of login attempts
3. Authentication against third-parties (OAuth, for example)

Note: For more details on installation and configuration of Django authentication system, please refer to the official guide <https://docs.djangoproject.com/en/1.10/topics/auth/>.

GeoNode communicates with GeoServer through Basic Authentication under the hood, in order to configure the data and the GeoServer catalog.

In order to do this, you must be sure that GeoNode knows the **internal** admin user and password of GeoServer.

Warning: This must be an internal GeoServer user with admin rights, not a GeoNode one.

Make sure the credentials are correctly configured into the file `settings.py`

OGC_SERVER

Ensure that the OGC_SERVER settings are correctly configured.

Notice that the two properties `LOGIN_ENDPOINT` and `LOGOUT_ENDPOINT` must specify the GeoServer OAuth2 Endpoints (see details below). The default values `'j_spring_oauth2_geonode_login'` and `'j_spring_oauth2_geonode_logout'` work in most of the cases, unless you need some specific endpoints different from the latters. In any case those values **must** be coherent with the GeoServer OAuth2 Plugin configuration.

If in doubt, please use the default values here below.

Default values are:

```
...
# OGC (WMS/WFS/WCS) Server Settings
# OGC (WMS/WFS/WCS) Server Settings
OGC_SERVER = {
    'default': {
        'BACKEND': 'geonode.geoserver',
        'LOCATION': GEOSERVER_LOCATION,
        'LOGIN_ENDPOINT': 'j_spring_oauth2_geonode_login',
        'LOGOUT_ENDPOINT': 'j_spring_oauth2_geonode_logout',
        # PUBLIC_LOCATION needs to be kept like this because in dev mode
        # the proxy won't work and the integration tests will fail
        # the entire block has to be overridden in the local_settings
        'PUBLIC_LOCATION': GEOSERVER_PUBLIC_LOCATION,
        'USER': 'admin',
        'PASSWORD': 'geoserver',
        'MAPFISH_PRINT_ENABLED': True,
        'PRINT_NG_ENABLED': True,
        'GEONODE_SECURITY_ENABLED': True,
        'GEOGIG_ENABLED': False,
        'WMST_ENABLED': False,
        'BACKEND_WRITE_ENABLED': True,
        'WPS_ENABLED': False,
        'LOG_FILE': '%s/geoserver/data/logs/geoserver.log' % os.path.abspath(os.path.
↪join(PROJECT_ROOT, os.pardir)),
        # Set to name of database in DATABASES dictionary to enable
        'DATASTORE': '', # 'datastore',
        'PG_GEOGIG': False,
        'TIMEOUT': 10 # number of seconds to allow for HTTP requests
    }
}
...
```

GeoNode and GeoServer A&A Interaction

The GeoServer instance used by GeoNode, has a particular setup that allows the two frameworks to correctly interact and exchange informations on users credentials and permissions.

In particular GeoServer is configured with a `Filter Chain` for Authorization that makes use of the two following protocols:

1. **Basic Authentication; this is the default GeoServer Authentication mechanism. This makes use of [rfc2617](#) - Basic and Digest Authentication**

In other words, GeoServer takes a `username` and a `password` encoded [Base64](#) on the HTTP Request Headers and compare them against its internal database (which by default is an encrypted XML file on the GeoServer Data Dir). If the user's credentials match, then GeoServer checks for Authorization through its `Role Services` (we will see those services in details on the *GeoServer (Security Backend)* section below).

Note: GeoServer ships by default with `admin` and `geoserver` as the default administrator user name and password. Before putting the GeoServer on-line it is imperative to change at least the administrator password.

2. **OAuth2 Authentication;** this module allows GeoServer to authenticate against the [OAuth2 Protocol](#). If the Basic Authentication fails, GeoServer falls back to this by using GeoNode as OAuth2 Provider by default.

Note: Further details can be found directly on the official GeoServer documentation at section "[Authentication Chain](#)"

From the **GeoNode backend (server) side**, the server will make use of **Basic Authentication** with administrator credentials to configure the GeoServer catalog. GeoServer must be reachable by GeoNode of course, and GeoNode must know the internal GeoServer admin credentials.

From the **GeoNode frontend (browser and GUI) side**, the *Authentication* goal is to allow GeoServer to recognize as valid a user which has been already logged into GeoNode, providing kind of an [SSO](#) mechanism between the two applications.

GeoServer must know and must be able to access GeoNode via HTTP/HTTPS. In other words, an external user connected to GeoNode must be authenticated to GeoServer with same permissions. This is possible through the **OAuth2 Authentication** Protocol.

GeoNode / GeoServer Authentication Mechanism

GeoNode as OAuth2 Provider (OP)

OpenID Connect is an identity framework built on OAuth 2.0 protocol which extends the authorization of OAuth 2.0 processes to implement its authentication mechanism. OpenID Connect adds a discovery mechanism allowing users to use an external trusted authority as an identity provider. From another point of view, this can be seen as a single sign on (SSO) system.

OAuth 2.0 is an authorization framework which is capable of providing a way for clients to access a resource with restricted access on behalf of the resource owner. OpenID Connect allows clients to verify the users with an authorization server based authentication.

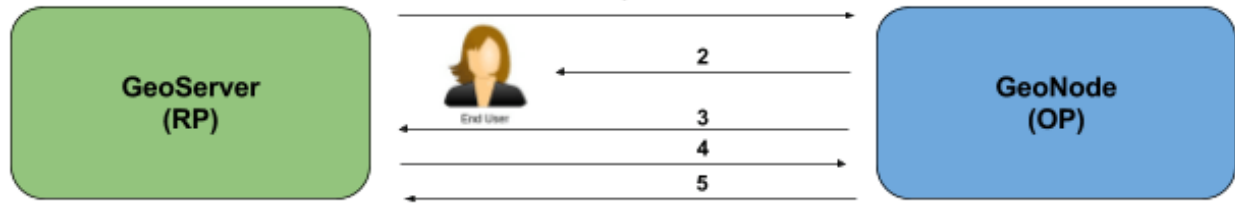
As an OP, GeoNode will be able to act as trusted identity provider, thus allowing the system working on an isolated environment and/or allow GeoNode to authenticate private users managed by the local Django auth subsystem.

GeoServer as OAuth2 Relying Party (RP)

Thanks to the **OAuth2 Authentication** GeoServer is able to retrieve an end user's identity directly from the OAuth2 Provider (OP).

With GeoNode acting as an OP, the mechanism will avoid the use of cookies relying, instead, on the OAuth2 secure protocol.

How the OAuth2 Protocol works:



1. The relying party sends the request to the OAuth2 provider to authenticate the end user
2. The OAuth2 provider authenticates the user
3. The OAuth2 provider sends the ID token and access token to the relying party
4. The relying party sends a request to the user info endpoint with the access token received from OAuth2 provider
5. The user info endpoint returns the claims.

GeoNode / GeoServer Authorization Mechanism

Allowing GeoServer to make use of a OAuth2 in order to act as an OAuth2 RP, is not sufficient to map a user identity to its roles though.

On GeoServer side we will still need to a `RoleService` which would be able to talk to GeoNode and transform the tokens into a User Principal to be used within the GeoServer Security subsystem itself.

In other words after a successful Authentication, GeoServer needs to Authorize the user in order to understand which resources he is enable to access or not. A REST based `RoleService` on GeoNode side, allows GeoServer to talk to GeoNode via [REST](#) to get the current user along with the list of its Roles.

Nevertheless knowing the Roles associated to a user is not sufficient. The complete GeoServer Authorization needs to catch a set of `Access Rules`, associated to the Roles, in order to establish which resources and data are accessible by the user.

The GeoServer Authorization is based on Roles only, therefore for each authenticated user we need also to know:

1. The Roles associated to a valid user session
2. The access permissions associated to a GeoServer Resource

The Authentication mechanism above allows GeoServer to get information about the user and his Roles, which addresses point 1.

About point 2, GeoServer makes use of the [GeoFence Embedded Server](#) plugin. GeoFence is a java web application that provides an advanced authentication / authorization engine for GeoServer using the interface described in [here](#). GeoFence has its own rules database for the management of Authorization rules, and overrides the standard GeoServer security management system by implementing a sophisticated Resource Access Manager. Least but not last, GeoFence implements and exposes a [REST API](#) allowing remote authorized clients to read / write / modify security rules.

The advantages using such plugin are multiple:

1. The Authorizations rules have a fine granularity. The security rules are handled by GeoFence in a way similar to the iptables ones, and allow to define security constraints even on sub-regions and attributes of layers.
2. GeoFence exposes a REST interface to its internal rule database, allowing external managers to update the security constraints programmatically

3. GeoFence implements an internal caching mechanism which improves considerably the performances under load.

GeoNode interaction with GeoFence

GeoNode itself is able to push/manage Authorization rules to GeoServer through the GeoFence [REST API](#), acting as an administrator for GeoServer. GeoNode properly configures the GeoFence rules anytime it is needed, i.e. the permissions of a Resource / Layer are updated.

GeoServer must know and must be able to access GeoNode via HTTP/HTTPS. In other words, an external user connected to GeoNode must be authenticated to GeoServer with same permissions. This is possible through the **GeoNodeCookieProcessingFilter**.

Summarizing we will have different ways to access GeoNode Layers:

1. Through GeoNode via Django Authentication and **GeoNodeCookieProcessingFilter**; basically the users available in GeoNode are also valid for GeoServer or any other backend.

Warning: If a GeoNode user has “administrator” rights, he will be able to administer GeoServer too.

2. Through GeoServer Security Subsystem; it will be always possible to access to GeoServer using its internal security system and users, unless explicitly disabled (**warning** this is dangerous, you must know what you are doing).

Lets now see in details how the single pieces are configured and how they can be configured.

Django OAuth Toolkit Setup and Configuration

As stated above, GeoNode makes use of the OAuth2 protocol for all the frontend interactions with GeoServer. GeoNode must be configured as an OAuth2 Provider and provide a `Client ID` and a `Client Secret` keys to GeoServer. This is possible by enabling and configuring the [Django OAuth Toolkit Plugin](#).

Warning: GeoNode and GeoServer won't work at all if the following steps are not executed at the first installation.

Default `settings.py` Security Settings for OAuth2

Double check that the OAuth2 Provider and Security Plugin is enabled and that the settings below are correctly configured.

AUTH_IP_WHITELIST

`AUTH_IP_WHITELIST` property limits access to users/groups REST Role Service endpoints to the only whitelisted IP addresses. Empty list means ‘allow all’. If you need to limit ‘api’ REST calls to only some specific IPs fill the list like this: `AUTH_IP_WHITELIST = ['192.168.1.158', '192.168.1.159']`

Default values are:

```
...
AUTH_IP_WHITELIST = []
...
```

INSTALLED_APPS

In order to allow GeoNode to act as an OAuth2 Provider, we need to enable the `oauth2_provider` Django application provided by the “Django OAuth Toolkit”.

Default values are:

```
...
INSTALLED_APPS = (

    'modeltranslation',

    ...
    'guardian',
    'oauth2_provider',
    ...

) + GEONODE_APPS
...
```

MIDDLEWARE_CLASSES

Installing the `oauth2_provider` Django application is not sufficient to enable the full functionality. We need also GeoNode to include additional entities to its internal model.

Default values are:

```
...
MIDDLEWARE_CLASSES = (
    'django.middleware.common.CommonMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',

    # The setting below makes it possible to serve different languages per
    # user depending on things like headers in HTTP requests.
    'django.middleware.locale.LocaleMiddleware',
    'pagination.middleware.PaginationMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',

    # This middleware allows to print private layers for the users that have
    # the permissions to view them.
    # It sets temporary the involved layers as public before restoring the
    →permissions.
    # Beware that for few seconds the involved layers are public there could be risks.
    # 'geonode.middleware.PrintProxyMiddleware',

    # If you use SessionAuthenticationMiddleware, be sure it appears before
    →OAuth2TokenMiddleware.
    # SessionAuthenticationMiddleware is NOT required for using django-oauth-toolkit.
    'django.contrib.auth.middleware.SessionAuthenticationMiddleware',
    'oauth2_provider.middleware.OAuth2TokenMiddleware',
)
...
```


AUTHENTICATION_BACKENDS

In order to allow GeoNode to act as an OAuth2 Provider, we need to enable the `oauth2_provider.backends.OAuth2Backend` Django backend provided by the “Django OAuth Toolkit”. Also notice that we need to specify the OAuth2 Provider scopes and declare which generator to use in order to create OAuth2 Client IDs.

Default values are:

```
...
# Replacement of default authentication backend in order to support
# permissions per object.
AUTHENTICATION_BACKENDS = (
    'oauth2_provider.backends.OAuth2Backend',
    'django.contrib.auth.backends.ModelBackend',
    'guardian.backends.ObjectPermissionBackend',
)

OAUTH2_PROVIDER = {
    'SCOPES': {
        'read': 'Read scope',
        'write': 'Write scope',
        'groups': 'Access to your groups'
    },

    'CLIENT_ID_GENERATOR_CLASS': 'oauth2_provider.generators.ClientIdGenerator',
}
...
```

Django OAuth Toolkit Admin Setup

Once the `settings.py` and `local_settings.py` have been correctly configured for your system:

1. Complete the GeoNode setup steps

- Prepare the model

```
python manage.py makemigrations
python manage.py migrate
python manage.py syncdb
```

- Prepare the static data

```
python manage.py collectstatic
```

- Make sure the database has been populated with initial default data

Warning: *Deprecated* this command will be replaced by migrations in the future, so be careful.

```
python manage.py loaddata initial_data.json
```

- Make sure there exists a superuser for your environment

Warning: *Deprecated* this command will be replaced by migrations in the future, so be careful.

```
python manage.py createsuperuser
```

Note: Read the base tutorials on GeoNode Developer documentation for details on the specific commands and how to use them.

2. Start the application

Start GeoNode accordingly on how the setup has been done; run debug mode through `paver`, or proxied by an HTTP Server like Apache2 HTTPD, Nginx or others.

3. Finalize the setup of the OAuth2 Provider

First of all you need to configure and create a new OAuth2 Application called `GeoServer` through the GeoNode Admin Dashboard

- Access the GeoNode Admin Dashboard
- Go to Django OAuth Toolkit > Applications
- Update or create the Application named `GeoServer`

Warning: The Application name **must** be `GeoServer`

- `Client id`; An alphanumeric code representing the OAuth2 Client Id. `GeoServer OAuth2 Plugin` **will** use **this** value.

Warning: In a production environment it is **highly** recommended to modify the default value provided with GeoNode installation.











- `User`; Search for the admin user. Its ID will be automatically updated into the form.
- `Redirect uris`; It is possible to specify many URIs here. Those must coincide with the `GeoServer` instances URIs.
- `Client type`; Choose `Confidential`
- `Authorization grant type`; Choose `Authorization code`
- `Client secret`; An alphanumeric code representing the OAuth2 Client Secret. `GeoServer OAuth2 Plugin` **will** use **this** value.

Warning: In a production environment it is **highly** recommended to modify the default value provided with GeoNode installation.

- `Name`; **Must** be `GeoServer`

Menu




-  Upload Layers
-  Profile
-  Recent Activity
-  Inbox
-  Announcements
-  Remote Services
-  Invite User
-  GeoServer
-  **Admin**
-  Help

Log out

Django OAuth Toolkit	
Access tokens	+ Add ✎ Change
Applications	+ Add ✎ Change
Grants	+ Add ✎ Change
Refresh tokens	+ Add ✎ Change

Change application

Client id:	<input type="text" value="Jrchz2oPY3akmzndmgUTYrs9gcZlgoV2I"/>
User:	<input type="text" value="2"/>  admin
Redirect uris:	<div><div>http://localhost:8080/geoserver http://localhost:8080/geoserver/ <a href="http://<host_name_or_ip>/geoserver">http://<host_name_or_ip>/geoserver <a href="http://<host_name_or_ip>/geoserver/">http://<host_name_or_ip>/geoserver/</div><div>Allowed URIs list, space separated</div></div>
Client type:	<input type="text" value="Confidential"/>
Authorization grant type:	<input type="text" value="Authorization code"/>
Client secret:	<input type="text" value="rCnp5txobUo83EpQEblM8fVj3QT5zb5ql"/>
Name:	<input type="text" value="GeoServer"/>
<input type="checkbox"/> Skip authorization	

GeoServer Security Backend

GeoServer Security Subsystem

GeoServer has a robust security subsystem, modeled on Spring Security. Most of the security features are available through the Web administration interface.

For more details on how this works and how to configure and modify it, please refer to the official GeoServer guide <http://docs.geoserver.org/stable/en/user/security/webadmin/index.html>

By using the `GeoServer Data Dir` provided with GeoNode build, the following configuration are already available. You will need just to update them accordingly to your environment (like IP addresses and Host names, OAuth2 Keys, and similar things). However it is recommended to read carefully all the following passages in order to understand exactly how the different component are configured and easily identify any possible issue during the deployment.

The main topics of this section are:

1. Connection to the GeoNode REST Role Service
2. Setup of the GeoServer OAuth2 Authentication Filter
3. Configuration of the GeoServer Filter Chains
4. Setup and test of the GeoFence Server and Default Rules

Connection to the GeoNode REST Role Service

Preliminary checks

- GeoServer is up and running and you have admin rights
- GeoServer must reach the GeoNode instance via HTTP
- The GeoServer Host IP Address must be allowed to access the GeoNode Role Service APIs (see the section `AUTH_IP_WHITELIST` above)

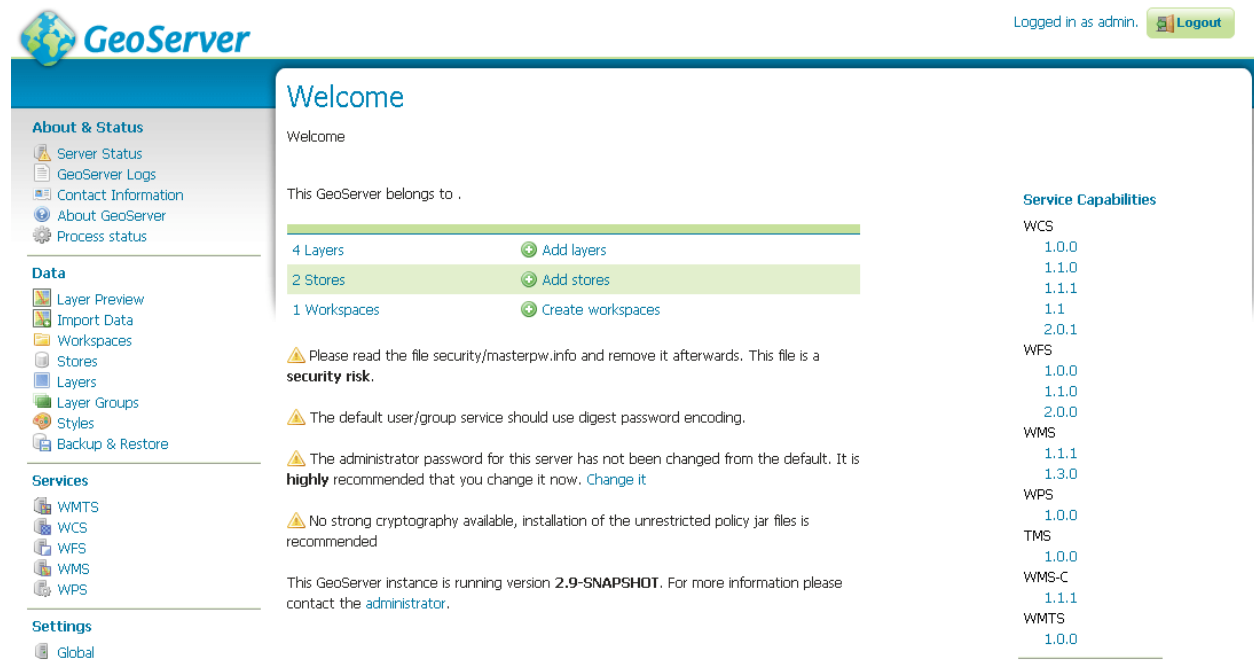
Setup of the GeoNode REST Role Service

1. Login as `admin` to the GeoServer GUI

Warning: In a production system remember to change the default admin credentials `admin` `geoserver`

2. Access the `Security > Users, Groups, Roles` section
3. **If not yet configured** the service `geonode REST role service`, click on `Role Services > Add new`

Note: This passage is **not** needed if the `geonode REST role service` has been already created. If so it will be displayed among the Role Services list



GeoServer

Logged in as admin. [Logout](#)

Welcome

Welcome

This GeoServer belongs to .

Count	Action
4 Layers	Add layers
2 Stores	Add stores
1 Workspaces	Create workspaces

Service Capabilities

WCS	1.0.0
	1.1.0
	1.1.1
	1.1
	2.0.1
WFS	1.0.0
	1.1.0
	2.0.0
WMS	1.1.1
	1.3.0
WPS	1.0.0
TMS	1.0.0
WMS-C	1.1.1
WMTS	1.0.0

About & Status

- Server Status
- GeoServer Logs
- Contact Information
- About GeoServer
- Process status

Data

- Layer Preview
- Import Data
- Workspaces
- Stores
- Layers
- Layer Groups
- Styles
- Backup & Restore

Services

- WMTS
- WCS
- WFS
- WMS
- WPS

Settings

- Global

Please read the file security/masterpw.info and remove it afterwards. This file is a **security risk**.











The default user/group service should use digest password encoding.

The administrator password for this server has not been changed from the default. It is **highly** recommended that you change it now. [Change it](#)

No strong cryptography available, installation of the unrestricted policy jar files is recommended

This GeoServer instance is running version **2.9-SNAPSHOT**. For more information please contact the [administrator](#).

Security

-  [Settings](#)
-  [Authentication](#)
-  [Passwords](#)
-  [Users, Groups, Roles](#)
-  [Data](#)
-  [Manage users, groups, and roles](#)
-  [WPS security](#)
-  [GeoFence](#)
-  [GeoFence Data Rules](#)
-  [GeoFence Admin Rules](#)

Role Services

- [Add new](#)
- [Remove selected](#)

Search		
<input type="checkbox"/> Name	Type	Administrator Role
<input type="checkbox"/> default	Default XML role service	ADMIN
<input type="checkbox"/> geonode REST role service	AuthKEY REST Role Service	ROLE_ADMIN

[<<](#)
[<](#)
[1](#)
[>](#)
[>>](#)
 Results 1 to 2 (out of 2 items)



Users, Groups, and Roles






Manage user group and role services



User Group Services

-  [Add new](#)
-  [Remove selected](#)

	Name	Type
	default	Default XML user/group service

     Results 1 to 1 (out of 1 ite

Role Services

-  [Add new](#)
-  [Remove selected](#)

4. If not yet configured the service geonode REST role service, choose AuthKEY REST - Role service from REST endpoint

New Role Service

Create and configure a new Role Service

XML - Default role service stored as XML

J2EE - Role service extracting roles from web.xml

AuthKEY REST - Role service from REST endpoint

JDBC - Role service stored in database

LDAP - Role service stored in LDAP repository

5. Create / update the geonode REST role service accordingly

- Name; **Must be** geonode REST role service
- Base Server URL; **Must point to the GeoNode instance base URL** (e.g. `http://<geonode_host_url>`)
- Roles REST Endpoint; **Enter** `/api/roles`
- Admin Role REST Endpoint; **Enter** `/api/adminRole`
- Users REST Endpoint; **Enter** `/api/users`
- Roles JSON Path; **Enter** `$.groups`
- Admin Role JSON Path; **Enter** `$.adminRole`
- Users JSON Path; **Enter** `$.users[0].groups`

Once everything has been setup and it is working, choose the Administrator role and Group administrator role as `ROLE_ADMIN`

Allow GeoFence to validate rules with ROLES

Warning: The following instruction are different accordingly to the GeoServer version you are currently using.

GeoServer 2.9.x and 2.10.x

1. Access the Security > Settings section
2. Choose the geonode REST role service as Active role service

AuthKEY REST Role Service

Role service from REST endpoint

Settings

Roles

Name

geonode REST role service

Administrator role

ROLE_ADMIN ▼

Group administrator role

ROLE_ADMIN ▼

REST Role Service Settings

Base Server URL

http://<geonode_host_url>

Roles REST Endpoint

/api/roles

Admin Role REST Endpoint

/api/adminRole

Users REST Endpoint

/api/users

Roles JSON Path

\$.groups

Admin Role JSON Path

\$.adminRole

Users JSON Path

\$.users

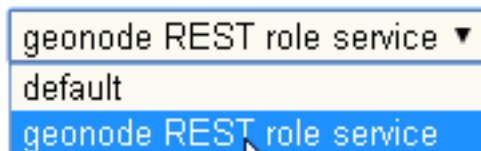
Security



Security Settings

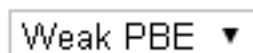
Configure security settings

Active role service



☐ Encrypt web admin URL parameters

Password encryption



⚠ No strong cryptography available

Save

Cancel

GeoServer 2.12.x and above

With the latest updates to GeoFence Plugin, the latter no more recognizes the Role Service from the default settings but from the `geofence-server.properties` file.

That said, it is important that the `Security > Settings` role service will be set to **default**, in order to allow GeoServer following the standard authorization chain.

On the other side, you will need to be sure that the `geofence-server.properties` file under the `$GEOSERVER_DATA_DIR/geofence` folder, contains the two following additional properties:

```
gwc.context.suffix=gwc
org.geoserver.rest.DefaultUserGroupServiceName=geonode REST role service
```

Setup of the GeoServer OAuth2 Authentication Filter

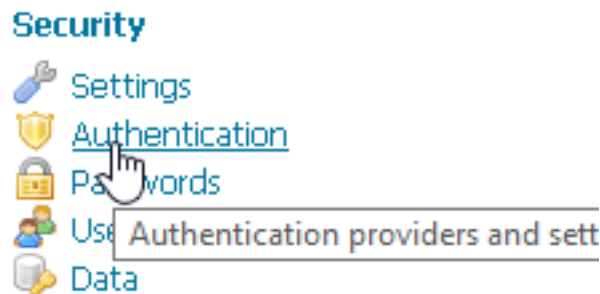
It is necessary now check that GeoServer can connect to OAuth2 Providers (specifically to GeoNode OP), and being able to Authenticate users through it.

Preliminary checks

- GeoServer is up and running and you have admin rights
- GeoServer must reach the GeoNode instance via HTTP
- OAuth2 Client ID and Client Secret have been generated on GeoNode and known

Setup of the GeoNode OAuth2 Security Filter

1. Access the `Security > Authentication` section



2. If **not yet configured** the Authentication Filter `geonode-oauth2 - Authentication` using a GeoNode OAuth2, click on `Authentication Filters > Add new`

Note: This passage is **not** needed if the `geonode-oauth2 - Authentication` using a GeoNode OAuth2 has been already created. If so it will be displayed among the Authentication Filters list

Authentication Filters

 Add new

 Remove selected

<input type="checkbox"/> Name	Type
<input type="checkbox"/> anonymous	Anonymous authentication
<input type="checkbox"/> basic	Basic HTTP authentication
<input type="checkbox"/> form	Form authentication
<input type="checkbox"/> geonode-oauth2	Authentication using a GeoNode OAuth2
<input type="checkbox"/> geonodeAnonymousFilter	org.geonode.security.GeoNodeAnonymousProcessingFilter
<input type="checkbox"/> geonodeCookieFilter	org.geonode.security.GeoNodeCookieProcessingFilter
<input type="checkbox"/> rememberme	Remember me authentication

<<
<
1
>
>>
Results 1 to 7 (out of 7 items)

Authentication Filters

 Add new
 Remove selected

3. If not yet configured the Authentication Filter `geonode-oauth2` - Authentication using a GeoNode OAuth2, choose GeoNode OAuth2 - Authenticates by looking up for a valid GeoNode OAuth2 `access_token` key sent as URL parameter

New Authentication Filter

Create and configure a new Authentication Filter




`J2EE` - Delegates to servlet container for authenticationGeoNode OAuth2 - Authenticates by looking up for a valid GeoNode OAuth2 `access_token` key sent as URL parameter

4. Create / update the `geonode-oauth2` - Authentication using a GeoNode OAuth2 accordingly

- Name; **Must** be `geonode-oauth2`
- Enable Redirect Authentication EntryPoint; It is recommended to put this to `False`, otherwise GeoServer won't allow you to connect to its Admin GUI through the `Form` but only through GeoNode
- Login Authentication EndPoint; Unless you have specific needs, keep the default value `/j_spring_oauth2_geonode_login`
- Logout Authentication EndPoint; Unless you have specific needs, keep the default value `/j_spring_oauth2_geonode_logout`

Authentication using a GeoNode OAuth2 geonode-oauth2

Authenticates by looking up for a valid GeoNode OAuth2 access_token key sent as URL parameter

Name	geonode-oauth2	
OAuth2 provider connection		
Enable Redirect Authentication EntryPoint	<input type="checkbox"/>	
Login Authentication EndPoint	/j_spring_oauth2_geonode_login	
Logout Authentication EndPoint	/j_spring_oauth2_geonode_logout	
Force Access Token URI HTTPS Secured Protocol	<input type="checkbox"/>	
Access Token URI		

- Force Access Token URI HTTPS Secured Protocol; This must be False unless you enabled a Secured Connection on GeoNode. In that case you will need to trust the GeoNode Certificate on the GeoServer JVM Keystore. Please see details below
- Access Token URI; Set this to `http://<geonode_host_base_url>/o/token/`
- Force User Authorization URI HTTPS Secured Protocol; This must be False unless you enabled a Secured Connection on GeoNode. In that case you will need to trust the GeoNode Certificate on the GeoServer JVM Keystore. Please see details below
- User Authorization URI; Set this to `http://<geonode_host_base_url>/o/authorize/`
- Redirect URI; Set this to `http://<geoserver_host>/geoserver`. This address **must** be present on the Redirect URIs of GeoNode OAuth2 > Applications > GeoServer (see above)
- Check Token Endpoint URL; Set this to `http://<geonode_host_base_url>/api/o/v4/tokeninfo/`
- Logout URI; Set this to `http://<geonode_host_base_url>/account/logout/`
- Scopes; Unless you have specific needs, keep the default value `read,write,groups`
- Client ID; The Client id alphanumeric key generated by the GeoNode OAuth2 > Applications > GeoServer (see above)
- Client Secret; The Client secret alphanumeric key generated by the GeoNode OAuth2 > Applications > GeoServer (see above)
- Role source; In order to authorize the user against GeoNode, choose Role service > geonode REST role service

Configuration of the GeoServer Filter Chains

The following steps ensure GeoServer can adopt more Authentication methods. As stated above, it is possible to Authenticate to GeoServer using different protocols.

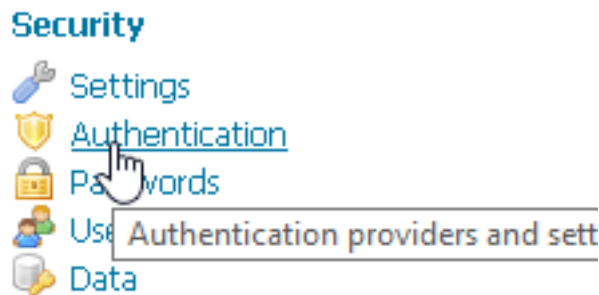
GeoServer scans the authentication filters chain associated to the specified path and tries them one by one sequentially. The first one matching the protocol and able to grant access to the user, breaks the cycle by creating a `User Principal` and injecting it into the `GeoServer SecurityContext`. The Authentication process, then, ends here and the control goes to the Authorization one, which will try to retrieve the authenticated user's Roles through the available GeoServer Role Services associated to the Authentication Filter that granted the access.

Preliminary checks

- GeoServer is up and running and you have admin rights
- GeoServer must reach the GeoNode instance via HTTP
- The `geonode-oauth2` - Authentication using a GeoNode OAuth2 Authentication Filter and the `geonode REST` role service have been correctly configured

Setup of the GeoServer Filter Chains

1. Access the `Security > Authentication` section



2. Identify the section `Filter Chains`
3. Make sure the web `Filter Chain` is configured as shown below

Warning: Every time you modify a Filter Chain, **don't forget to save** the Authentication settings. This **must** be done for **each** change.

4. Make sure the `rest` Filter Chain is configured as shown below

Warning: Every time you modify a Filter Chain, **don't forget to save** the Authentication settings. This **must** be done for **each** change.











5. Make sure the `gwc` Filter Chain is configured as shown below


Warning: Every time you modify a Filter Chain, **don't forget to save** the Authentication settings. This **must** be done for **each** change.

Filter Chains





 [Add service chain](#)





 [Add HTML chain](#)

Position	Name	Patterns
	web	/web/**,/gwc/rest/web/**,/,
 	webLogin	/j_spring_security_check,/j_spring_security_check/,/,
 	webLogout	/j_spring_security_logout,/j_spring_security_logout/,
 	rest	/rest/**
 	gwc	/gwc/rest/**
	default	/**



1


 Results 1 to 6 (out of 6 items)

Available		Selected
basic geonodeAnonymousFilter geonodeCookieFilter	   	geonode-oauth2 rememberme form anonymous

Available		Selected
geonodeAnonymousFilter geonodeCookieFilter	   	basic geonode-oauth2 anonymous

Available		Selected
anonymous geonodeAnonymousFilter geonodeCookieFilter	⇒ ⇐ ↑ ↓	basic geonode-oauth2

6. Make sure the default Filter Chain is configured as shown below

Available		Selected
geonodeAnonymousFilter geonodeCookieFilter	⇒ ⇐ ↑ ↓	basic geonode-oauth2 anonymous

Warning: Every time you modify a Filter Chain, **don't forget to save** the Authentication settings. This **must** be done for **each** change.

7. Add the GeoNode Login Endpoints to the comma-delimited list of the webLogin Filter Chain

Warning: Every time you modify a Filter Chain, **don't forget to save** the Authentication settings. This **must** be done for **each** change.

8. Add the GeoNode Logout Endpoints to the comma-delimited list of the webLogout Filter Chain

Warning: Every time you modify a Filter Chain, **don't forget to save** the Authentication settings. This **must** be done for **each** change.

9. Add the GeoNode Logout Endpoints to the comma-delimited list of the formLogoutChain XML node in `<GEOSERVER_DATA_DIR>/security/filter/formLogout/config.xml`

You will need a text editor to modify the file.

Note: If the `<formLogoutChain>` XML node does not exist at all, create a **new one** as specified below

Filter chain

Configure an individual filter chain

Chain settings

Name

Comma delimited list of ANT patterns (with optional query string)

- ☐ Disable security for this chain
- ☐ Allow creation of an HTTP session for storing the authentication token
- ☐ Accept only SSL requests

Role filter

Filter chain

Configure an individual filter chain

Chain settings

Name

Comma delimited list of ANT patterns (with optional query string)

- ☐ Disable security for this chain
- ☐ Allow creation of an HTTP session for storing the authentication token
- ☐ Accept only SSL requests

Role filter

```
<logoutFilter>
...
<redirectURL>/web/</redirectURL>
<formLogoutChain>/j_spring_security_logout,/j_spring_security_logout/,/
↪j_spring_oauth2_geonode_logout,/j_spring_oauth2_geonode_logout/</
↪formLogoutChain>
</logoutFilter>
```

Warning: The value `j_spring_oauth2_geonode_logout` **must** be the same specified as Logout Authentication EndPoint in the `geonode-oauth2 - Authentication` using a GeoNode OAuth2 above.

Setup and test of the GeoFence Server and Default Rules

In order to work correctly, GeoServer needs the [GeoFence Embedded Server](#) plugin to be installed and configured on the system.

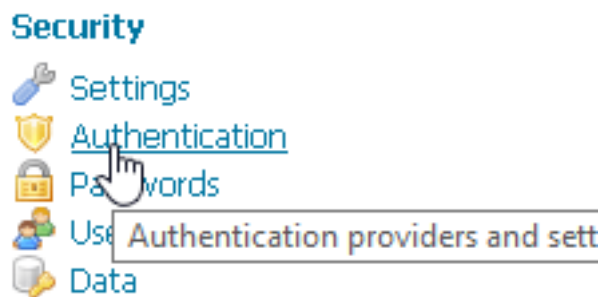
The GeoServer configuration provided for GeoNode, has the plugin already installed with a default configuration. In that case double check that the plugin works correctly and the default rules have been setup by following the next steps.

Preliminary checks

- GeoServer is up and running and you have admin rights
- The [GeoFence Embedded Server](#) plugin has been installed on GeoServer

Setup of the GeoServer Filter Chains

1. Access the Security > Authentication section



2. Identify the section Authentication Providers and make sure the geofence Authentication Provider is present
3. Make sure the Provider Chain is configured as shown below

Authentication Providers

[+ Add new](#)[- Remove selected](#)

Search	
<input type="checkbox"/> Name	Type
<input type="checkbox"/> default	Basic username/password authentication
<input type="checkbox"/> geofence	org.geoserver.geoserver.authentication.auth.GeoFenceAuthenticationProvider
<input type="checkbox"/> geonodeAuthProvider	org.geonode.security.GeoNodeAuthenticationProvider

<< < 1 > >> Results 1 to 3 (out of 3 items)

Provider Chain

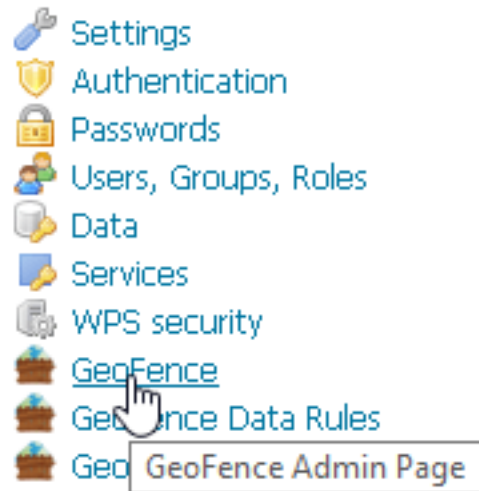
Available		Selected
geonodeAuthProvider	⇒ ⇐ ⇕ ⇓	default geofence

Warning: Every time you modify a Authentication Providers, **don't forget to save** the Authentication settings. This **must** be done for **each** change.

Setup of the GeoFence Server and Rules

1. Make sure GeoFence server works and the default settings are correctly configured
 - Access the Security > GeoFence section
 - Make sure the Options are configured as follows and the server works well when performing a Test Connection
 - Allow remote and inline layers in SLD; **Set it to True**
 - Allow SLD and SLD_BODY parameters in requests; **Set it to True**
 - Authenticated users can write; **Set it to True**
 - Use GeoServer roles to get authorizations; **Set it to False**
2. Check the GeoFence default Rules
 - Access the Security > GeoFence Data Rules section
 - Make sure the DENY ALL Rule is present by default, otherwise your data will be accessible to everyone

Security



Note: This rule is **always** the last one

Warning: If that rule does not exist **at the very bottom** (this rule is **always** the last one), add it manually.

- Access the `Security > GeoFence Admin Rules` section
- No Rules needed here

Troubleshooting and Advanced Features

Common Issues and Fixes

- GeoServer/GeoNode OAuth2 does not authenticate as Administrator even using GeoNode `admin` users

Symptoms

When trying to authenticate with an `admin` user using OAuth2, the process correctly redirects to GeoServer page but I'm not a GeoServer Administrator.

Cause

That means that somehow GeoServer could not successfully complete the Authorization and Authentication process.

The possible causes of the problem may be the following ones:

1. The OAuth2 Authentication fails on GeoServer side

This is usually due to an exception while trying to complete the Authentication process.

Connection successful

GeoFence Admin Page

GeoFence options Administration Page

General settings

GeoServer Instance name for GeoFence

GeoFence services URL (GeoServer restart is required if changed)

Test Connection



Options

- ☒ Allow remote and inline layers in SLD
- ☒ Allow SLD and SLD_BODY parameters in requests
- ☒ Authenticated users can write
- ☐ Use GeoServer roles to get authorizations



GeoFence Data Rules

Configure data rules for the internal GeoFence server.

- Add new rule
- Remove selected rules

<< < 1 > >>

Results 1 to 1 (out of 1 items)

Search

<input type="checkbox"/>	P	Role	User	Service	Request	Workspace	Layer	Access	
<input type="checkbox"/>	0	*	*	*	*	*	*	DENY	

<< < 1 > >>

Results 1 to 1 (out of 1 items)



GeoFence Admin Rules

Configure admin rules for the internal GeoFence server.

-  Add new rule
- Remove selected rules

<div> <div> <div><<</div> <div><</div> <div>></div> <div>>></div> </div> <div>Results 0 to 0 (out of 0 items)</div> <div> <div>Search</div> </div> </div>				
<input type="checkbox"/>	P	Role	User	Workspace
<div> <div> <div><<</div> <div><</div> <div>></div> <div>>></div> </div> <div>Results 0 to 0 (out of 0 items)</div> </div>				

- A typical cause is that GeoServer tries to use HTTPS connections but the GeoNode certificate is not trusted;

In that case please refer to the section below. Also take a look at the logs (in particular the GeoServer one) as explained in *Debugging GeoNode Installations*. The GeoServer logs should contain a detailed Exception explaining the cause of the problem. If no exception is listed here (even after raised the log level to *DEBUG*), try to check for the GeoNode Role Service as explained below.

- Another possible issue is that somehow the OAuth2 handshake cannot complete successfully;

1. Login into GeoServer as administrator through its WEB login form.
2. Double check that all the `geonode-oauth2` - Authentication using a GeoNode OAuth2 parameters are correct. If everything is ok, take a look at the logs (in particular the GeoServer one) as explained in *Debugging GeoNode Installations*. The GeoServer logs should contain a detailed Exception explaining the cause of the problem. If no exception is listed here (even after raised the log level to *DEBUG*), try to check for the GeoNode Role Service as explained below.

- ## 2. GeoServer is not able to retrieve the user Role from a Role Service

Always double check both HTTP Server and GeoServer log as specified in section *Debugging GeoNode Installations*. This might directly guide you to the cause of the problem.

- Check that the GeoServer host is granted to access GeoNode Role Service REST APIs in the `AUTH_IP_WHITELIST` of the `settings.py`
- Check that the `geonode rest role service` is the default Role service and that the GeoServer OAuth2 Plugin has been configured to use it by default
- Check that the GeoNode REST Role Service APIs are functional and produce correct JSON.

This is possible by using simple `cURL` GET calls like

```
curl http://localhost/api/adminRole  
$> {"adminRole": "admin"}  
  
curl http://localhost/api/users  
$> {"users": [{"username": "AnonymousUser", "groups": ["anonymous"]}, {"username": "afaf"}, {"username": "test"}], {"username": "admin", "groups": ["anonymous", "test", "admin"]}]}  
⏮️ ⏪️ ⏩️ ⏭️ 🔍 🔄
```

(continues on next page)

(continued from previous page)

```
curl http://localhost/api/roles
$> {"groups": ["anonymous", "test", "admin"]}

curl http://localhost/api/users/admin
$> {"users": [{"username": "admin", "groups": [
  ↪ "anonymous", "test", "admin"]}]}
```

How to setup HTTPS secured endpoints

In a production system it is a good practice to encrypt the connection between GeoServer and GeoNode. That would be possible by enabling HTTPS Protocol on the GeoNode REST Role Service APIs and OAuth2 Endpoints.

Most of the times you will rely on a self-signed HTTPS connection using a generated certificate. That makes the connection *untrusted* and you will need to tell to the GeoServer Java Virtual Machine to trust it.

This can be done by following the steps below.

For any issue take a look at the logs (in particular the GeoServer one) as explained in *Debugging GeoNode Installations*. The GeoServer logs should contain a detailed Exception explaining the cause of the problem.

SSL Trusted Certificates

When using a custom `Keystore` or trying to access a non-trusted or self-signed SSL-protected OAuth2 Provider from a non-SSH connection, you will need to add the certificates to the JVM `Keystore`.

In order to do this you can follow the next steps:

In this example we are going to

1. Retrieve SSL Certificate from GeoNode domain:

“Access Token URI” = `https://<geonode_host_base_url>/o/token/` therefore we need to trust `https://<geonode_host_base_url>` or `(<geonode_host_base_url>:443)`

Note: You will need to get and trust certificates from every different HTTPS URL used on OAuth2 Endpoints.

2. Store SSL Certificates on local hard-disk
3. Add SSL Certificates to the Java Keystore
4. Enable the JVM to check for SSL Certificates from the Keystore

1. Retrieve the SSL Certificate from GeoNode domain

Use the `openssl` command in order to dump the certificate

For `https://<geonode_host_base_url>`

```
openssl s_client -connect <geonode_host_base_url>:443
```

2. Store SSL Certificate on local hard-disk


```

Loading 'screen' into random state - done
CONNECTED(00000188)
depth=3 C = US, O = Equifax, OU = Equifax Secure Certificate Authority
verify return:1
depth=2 C = US, O = GeoTrust Inc., CN = GeoTrust Global CA
verify return:1
depth=1 C = US, O = Google Inc, CN = Google Internet Authority G2
verify return:1
depth=0 C = US, ST = California, L = Mountain View, O = Google Inc, CN = accounts.google.com
verify return:1
---
Certificate chain
 0 s:/C=US/ST=California/L=Mountain View/O=Google Inc/CN=accounts.google.com
  i:/C=US/O=Google Inc/CN=Google Internet Authority G2
 1 s:/C=US/O=Google Inc/CN=Google Internet Authority G2
  i:/C=US/O=GeoTrust Inc./CN=GeoTrust Global CA
 2 s:/C=US/O=GeoTrust Inc./CN=GeoTrust Global CA
  i:/C=US/O=Equifax/OU=Equifax Secure Certificate Authority
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIETCCA4mgAwIBAgIIeEP870cN1RQwDQYJKoZIhvcNAQELBQAwSTELMAkGA1UE
BhMCVVMxEzARBgNVBAoTCkdvb2dsZSBJbmMxJTAjBgNVBAMTHEdvd2dsZSBJbnRl
cm5ldCBBdXRob3JpdHkgRzIwHhcNMjYxMDE5MTcxNjU3WjcNMTEwMTcxMzAw
WjBtMQswCQYDVQQGEwJVUzETMBEGA1UECAwKQ2FsaWZvcn5pYTEwMBQGA1UEBwwN
TW91bnRhaW4gVmlldzETMBEGA1UECgwKR29vZ2x1IEluYzEcMBoGA1UEAwwTYWNj
b3VudHMuzZ29vZ2x1LmNvbTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEB
AK71x4DYCS7NcMqyp8UAq7067Wb7LLhkgE5QJcUVdvFfLrXmG3PAIcfiLMvdiK+F
RODn5d79I18qgn90XgB/xyFgx381MR/HoglavblCTfgbiotvhAMsxyY/h55efbcn
i2GGOXCWVcx119xPonWFvd7aUaOLSutMeE27T4Hqj3o0WFYVJq+dQNGdkGwqtUap
bosQFuyWg+rKINM6Opvy6ay8fRU/4JeKNVcaZUuIu12bPchBzIXiKtLkMJSIYzRz
ySjJmFLm++/ipMQ549xeyviB4pSz44athBuIgeNhgE8Xb7eTS5TPuWFjJ913nhdT
IDsa1PMX2PpsinA1IF8Z+kECAwEAaOCwggFjMB0GA1UdJQQWMBQGCCsGAQUF
BwMBBggrBgEFBQcDAjA1BgNVHREELjAsghNhY2NvdW50cy5nb29nbGUuY29tghUq

```

Copy-and-paste the section `-BEGIN CERTIFICATE-`, `-END CERTIFICATE-` and save it into a `.cert` file

Note: `.cert` file are plain text files containing the ASCII characters included on the `-BEGIN CERTIFICATE-`, `-END CERTIFICATE-` sections

`geonode.cert` (or whatever name you want with `.cert` extension)

```

1 -----BEGIN CERTIFICATE-----
2 MIIeOTCCA4mgAwIBAgIIeEP870cN1RQwDQYJKoZIhvcNAQELBQAwSTELMAkGA1UE
3 BhMCVVMxEzARBgNVBAoTCkdvb2dsZSBjb250aW50aW50aW50aW50aW50aW50aW50
4 cm5ldCBkdXRob3RpdHkgRzIwHhcNMjYxMDE5MTUxMjUzWhcNMjYxMDE5MTUxMjUz
5 WjBtMQswCQYDVQQGEwJVUzETMBEGA1UECgwKQ2FsaWZvcn5pYTEWMBQGA1UEBwwN
6 TW91bnRhaW4gVmlldzETMBEGA1UECgwKR29vZ2x1IEluYzEzMBQGA1UEAwwTYWVudHM
7 b3VudHMU229vZ2x1LmNvbTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEB
8 AK71x4DYCS7NcMqyp8UAq7067Wb7LLhkgE5QJcUVdvFfLrXmG3PAIcfilMvdIK+F
9 RODn5d79I18qgn90XgB/xyFgx381MR/Hoglavb1CTfgbiotvhAMxyY/h55efbcn
10 i2GGOXCWVcx119xPOnWFvd7aUa0LSutMeEZ7T4Hqj3o0WFYVJq+dQNGdkGwqtUap
11 bosQFuyWg+rKINM6Opvy6ay8fRU/4JeKNVcaZUuIu12bPchBzIXiKtLkMJSIYzRz
12 ySjJmFlm++/ipMQ549xyviB4pSz44athBuIgeNhgE8Xb7eTS5TPuWFjJ913nhdT
13 IDsa1PMX2PpsinA1IF8Z+kECawEAaOCaWcwgGfjMBOGA1UdJQQWMBQGCCsGAQUF
14 BwMBBggrBgEFBQcDAjA1BgNVHREELjAsghNhY2NvdW50cy5nb29nbGUuY29tghUq
15 LnBhenRuZXIuYW5kcm9pZC5jb20waA1IKwYBBQUHAQEEXDBaMCsGCCsGAQUFBzAC
16 hh9odHRwOi8vcGtpLmdvb2dsZS5jb20vRO1BRzIuY3JOMCsGCCsGAQUFBzABhh9o
17 dHRwOi8vY2xpZW50cy5nb29vZ2x1LmNvbS9vY3NmMBQGA1UdDgQWBBsf3MI/auhw
18 EZz38qRDr+B1Le11yTAMBgNVHRMBAf8EAjAAMB8GA1UdIwQYMBaAFerdBhYbvPZo
19 tXb1gba7Yhg6WoEvMCEGA1UdIAQAMBgwDAYKKwYBBAHWeQIFATAIBgZngQWBAgIw
20 MAYDVR0rBCKwIzA1oCQGVYVfaHR0cDovL3BraS5nb29nbGUuY29tL0dJIOIcwlMnW

```

3. Add SSL Certificates to the Java Keystore

You can use the Java command `keytool` like this

`geonode.cert` (or whatever name you want with `.cert` extension)

```
keytool -import -noprompt -trustcacerts -alias geonode -
→file geonode.cert -keystore ${KEYSTOREFILE} -storepass $
→{KEYSTOREPASS}
```

or, alternatively, you can use some graphic tool which helps you managing the SSL Certificates and Keystores, like [Portecle](#)

```
java -jar c:\apps\portecle-1.9\portecle.jar
```

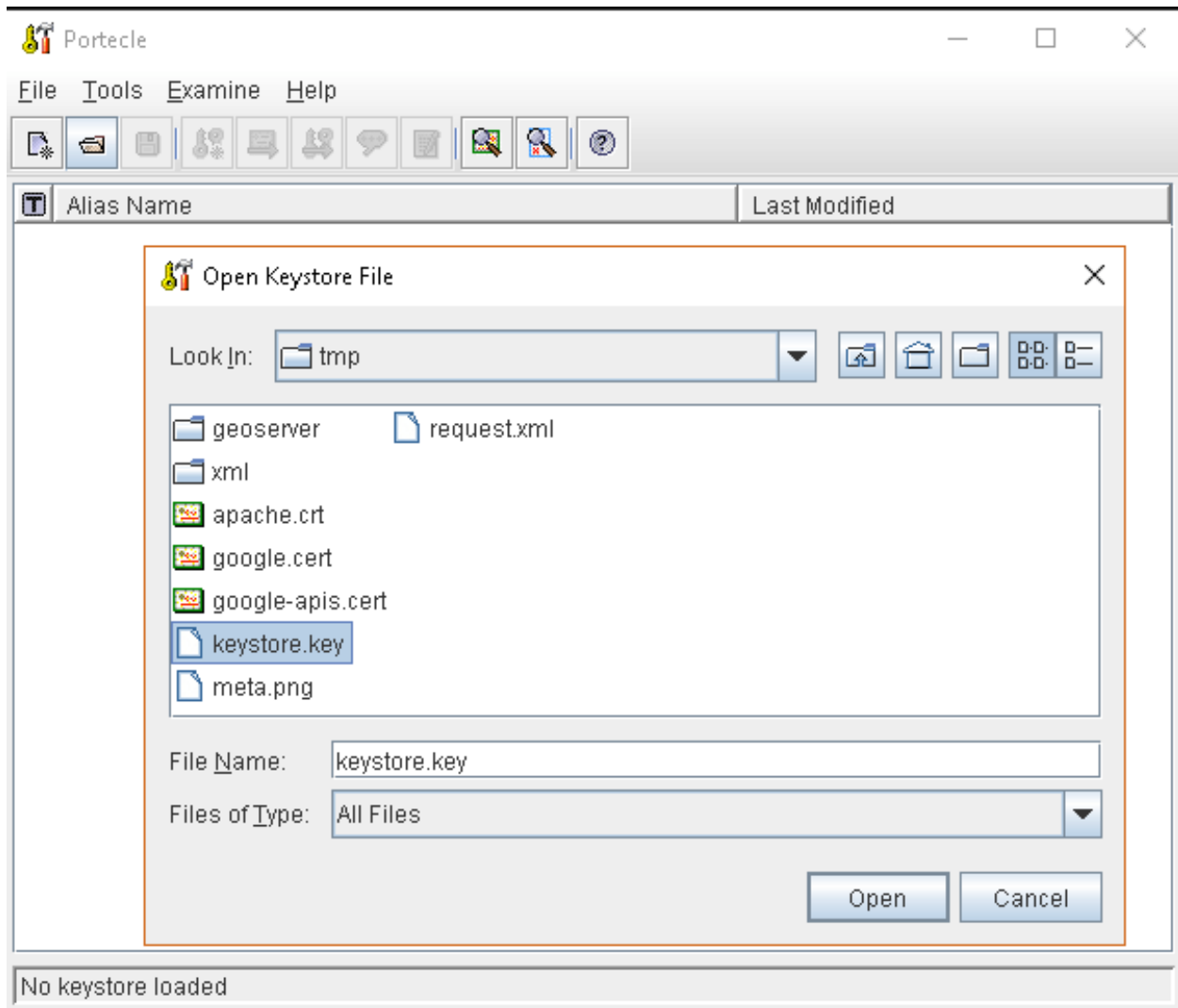
4. Enable the JVM to check for SSL Certificates from the Keystore

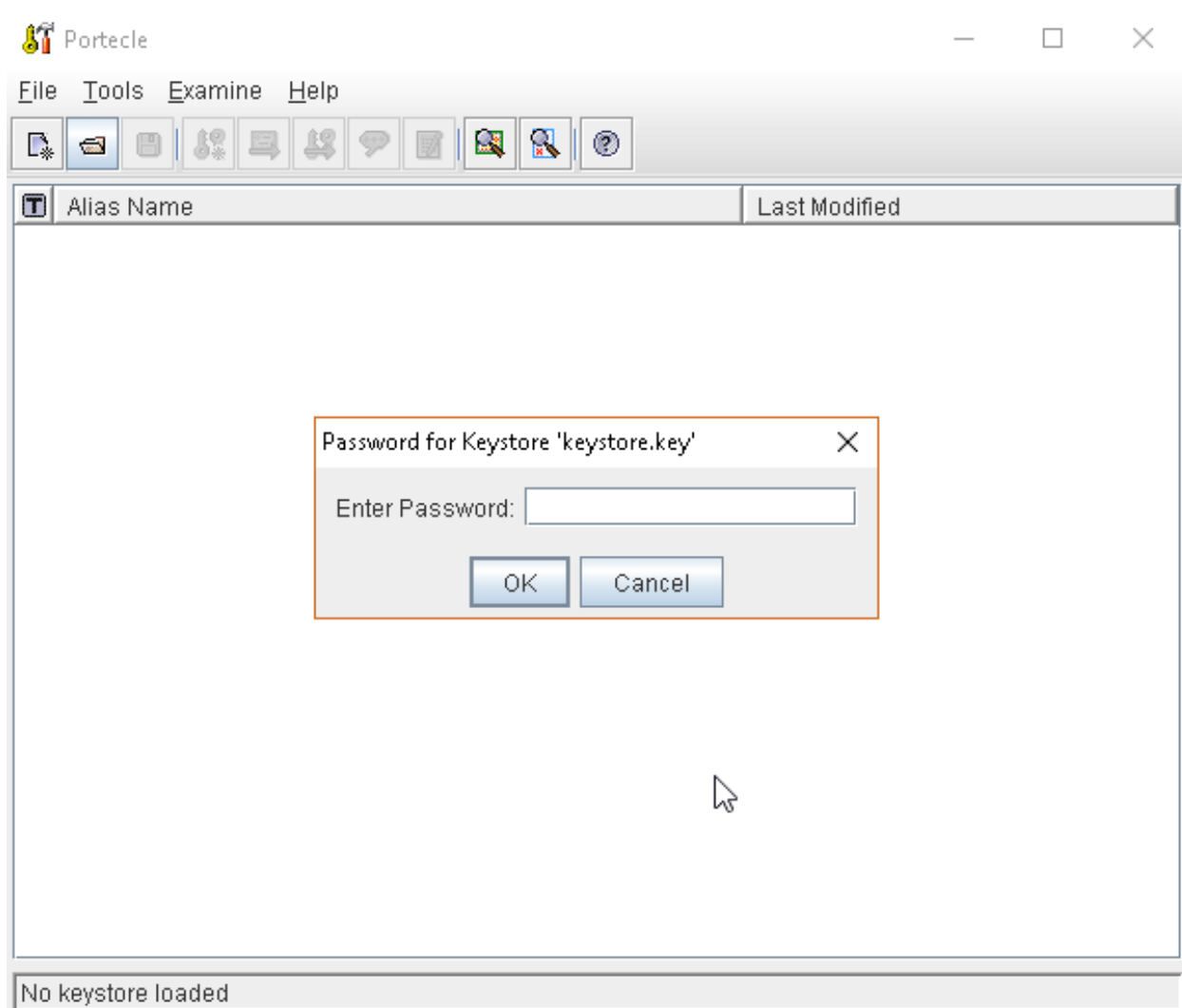
In order to do this, you need to pass a `JAVA_OPTION` to your JVM:

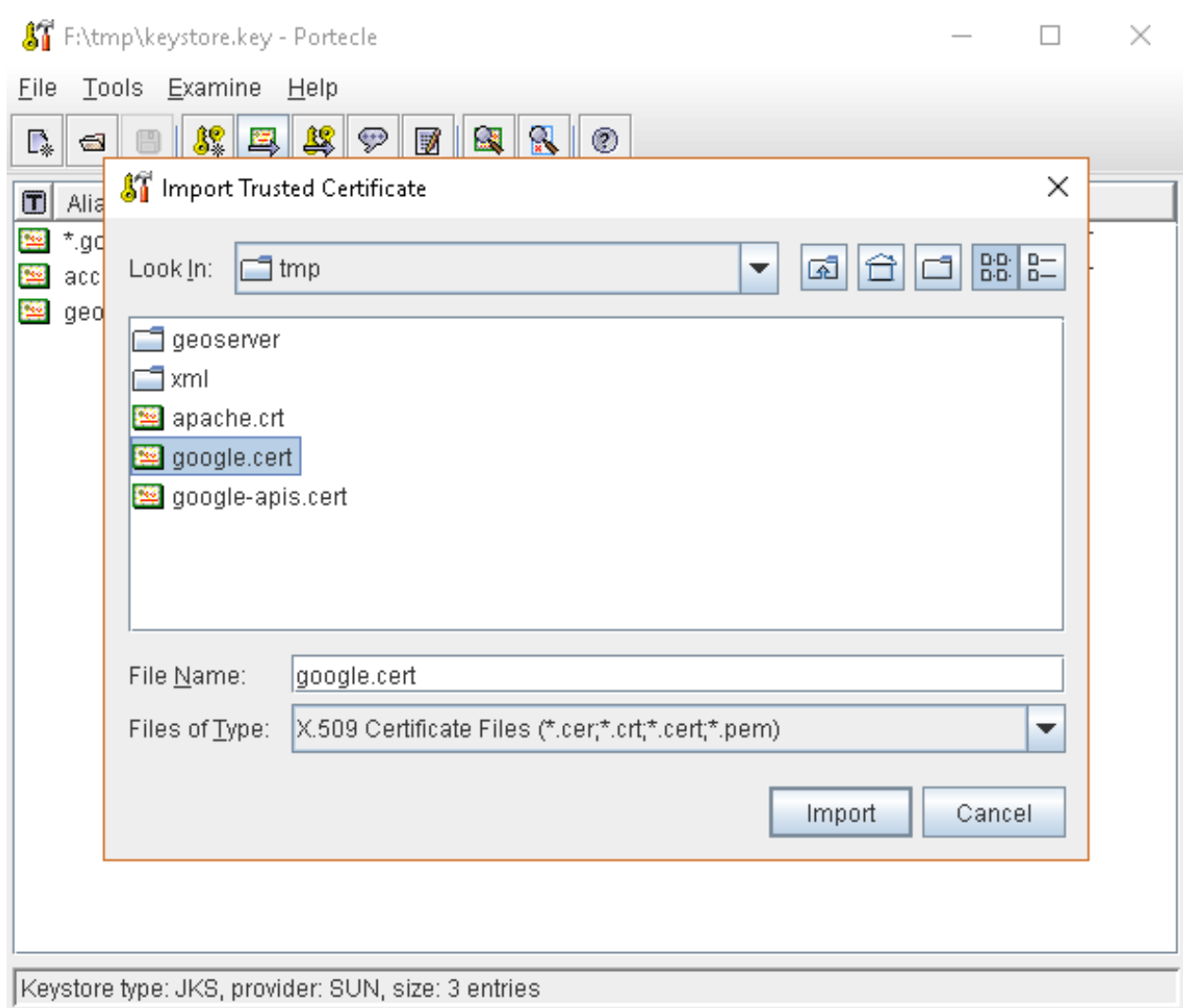
```
-Djavax.net.ssl.trustStore=F:\tmp\keystore.key
```

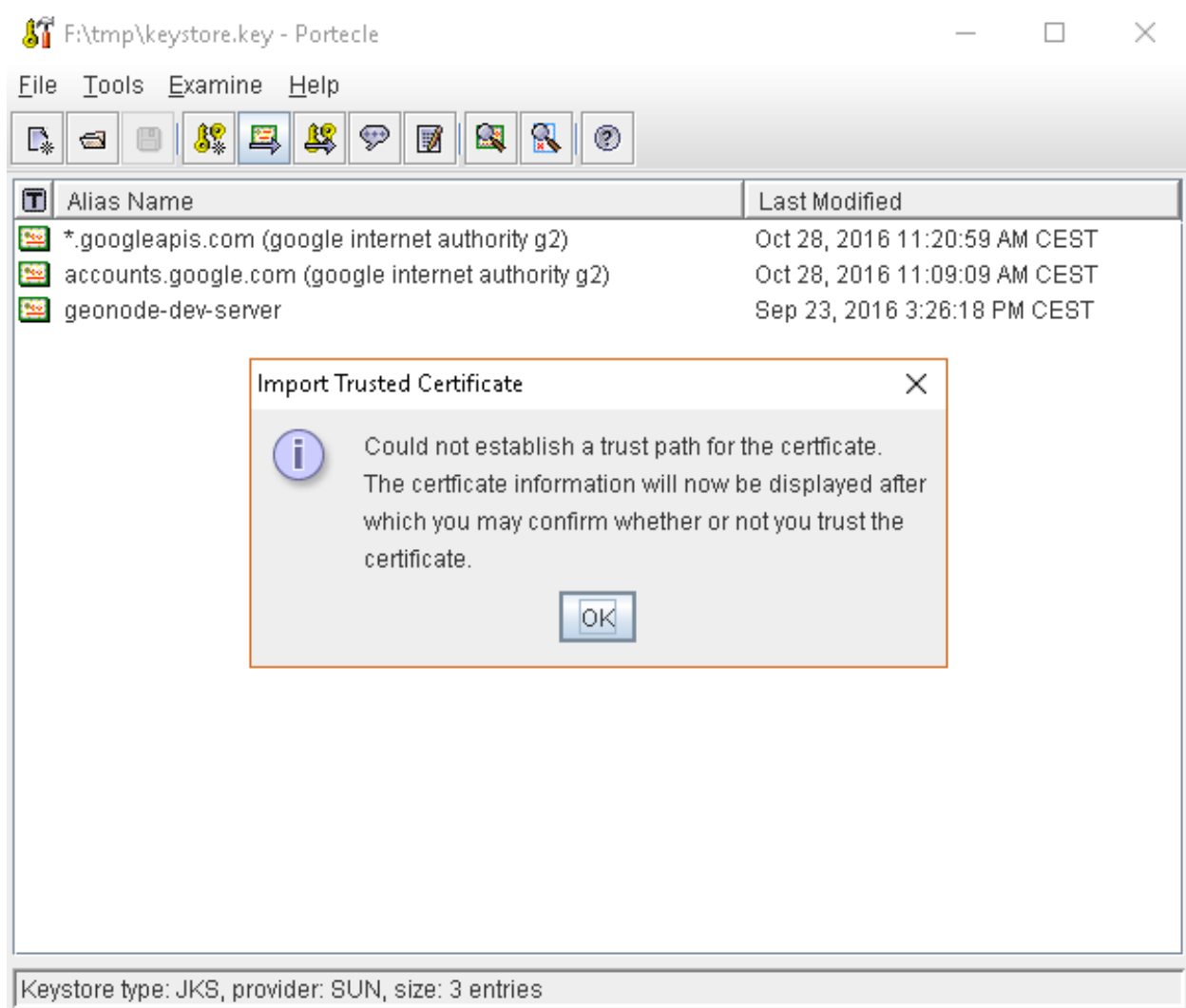
5. Restart your server

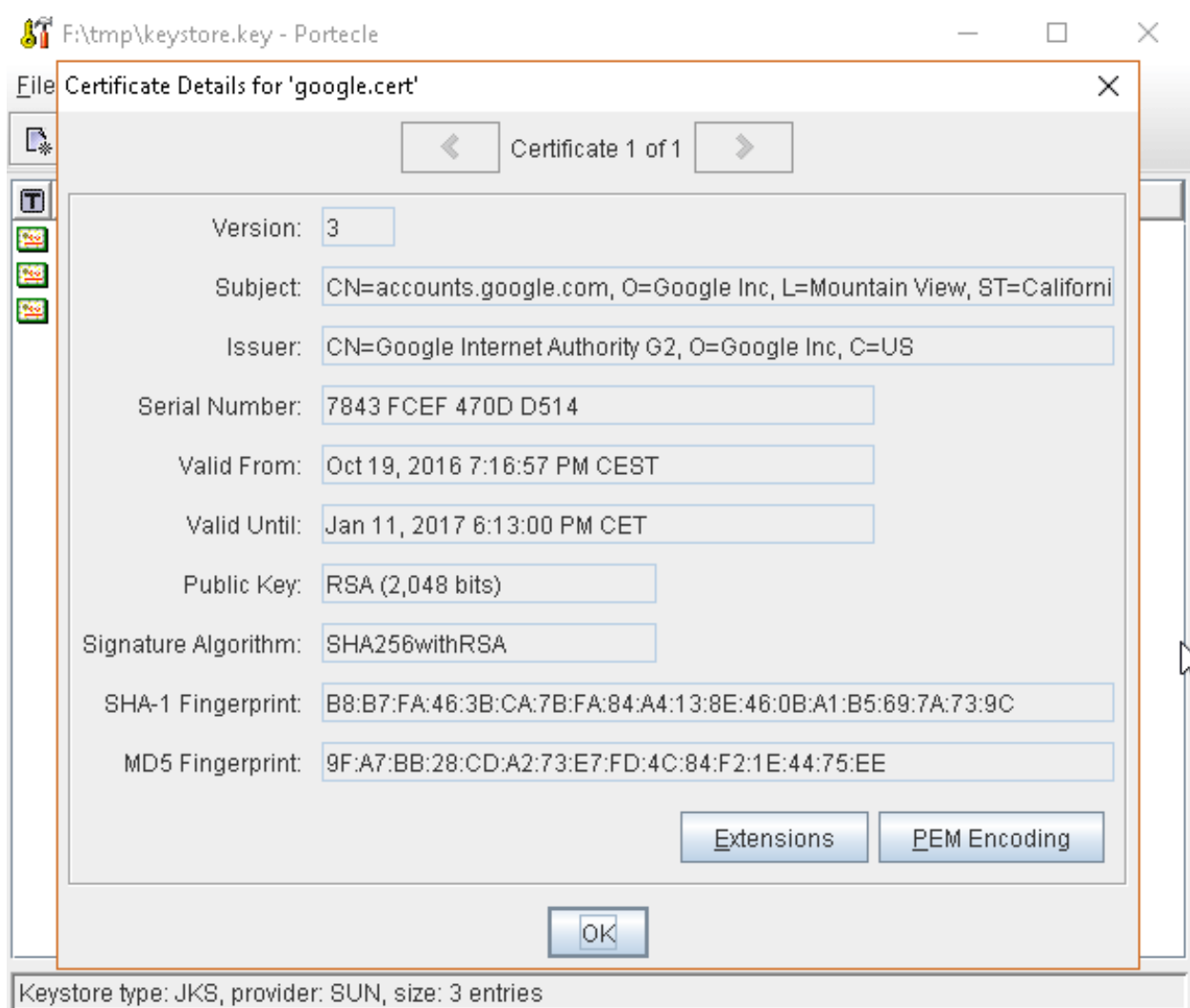
Note: Here below you can find a bash script which simplifies the Keystore SSL Certificates importing. Use it at your convenience.

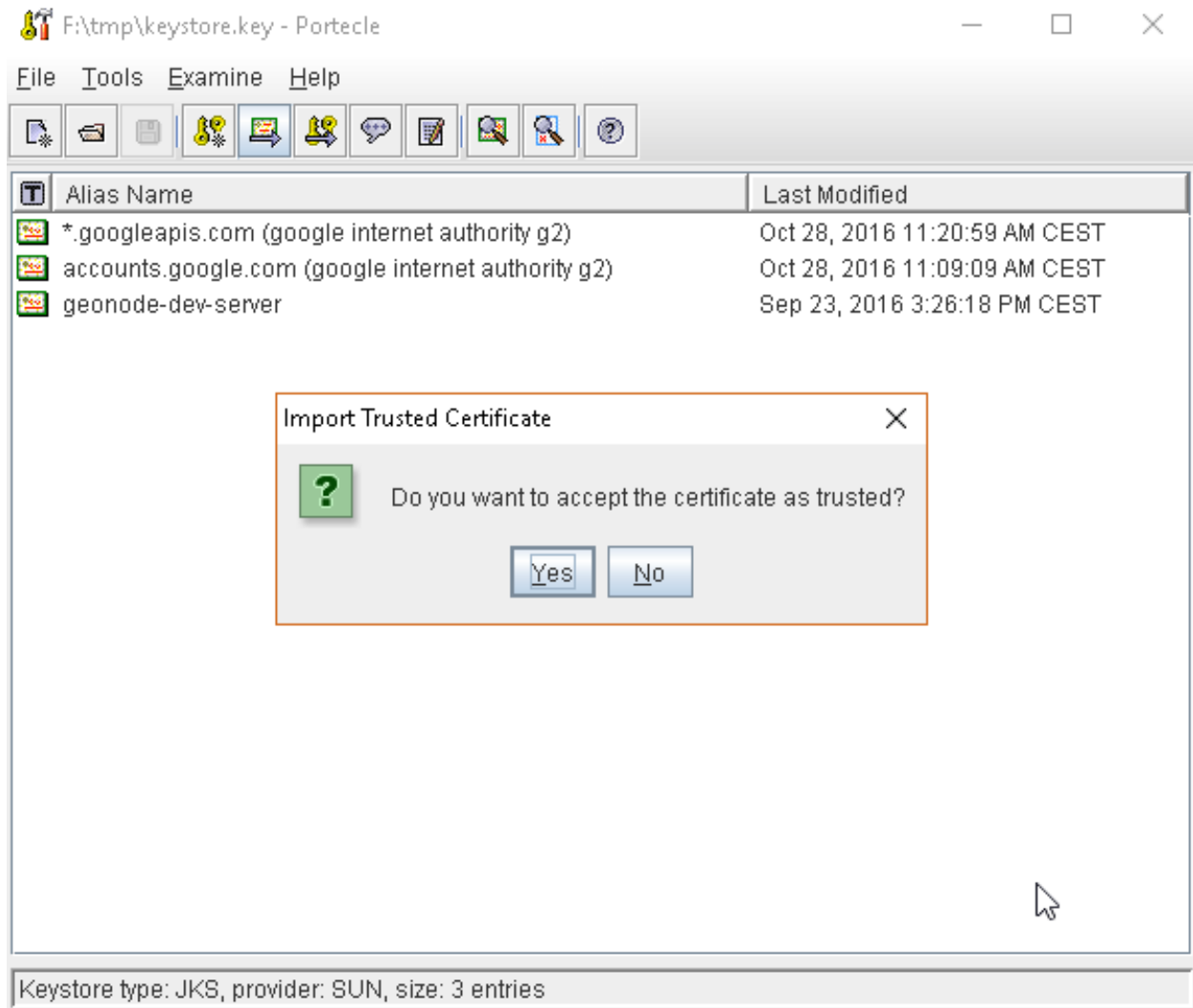


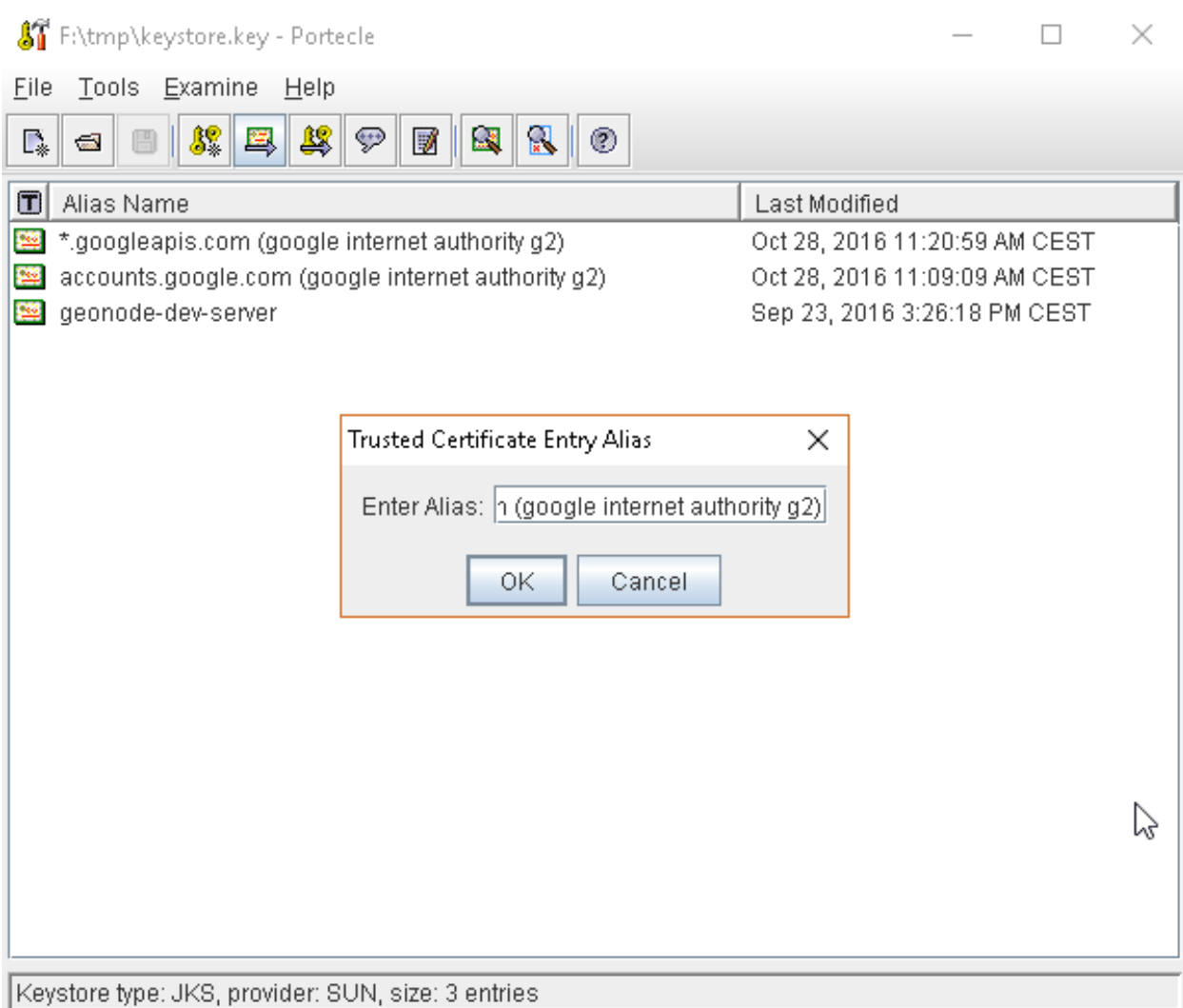


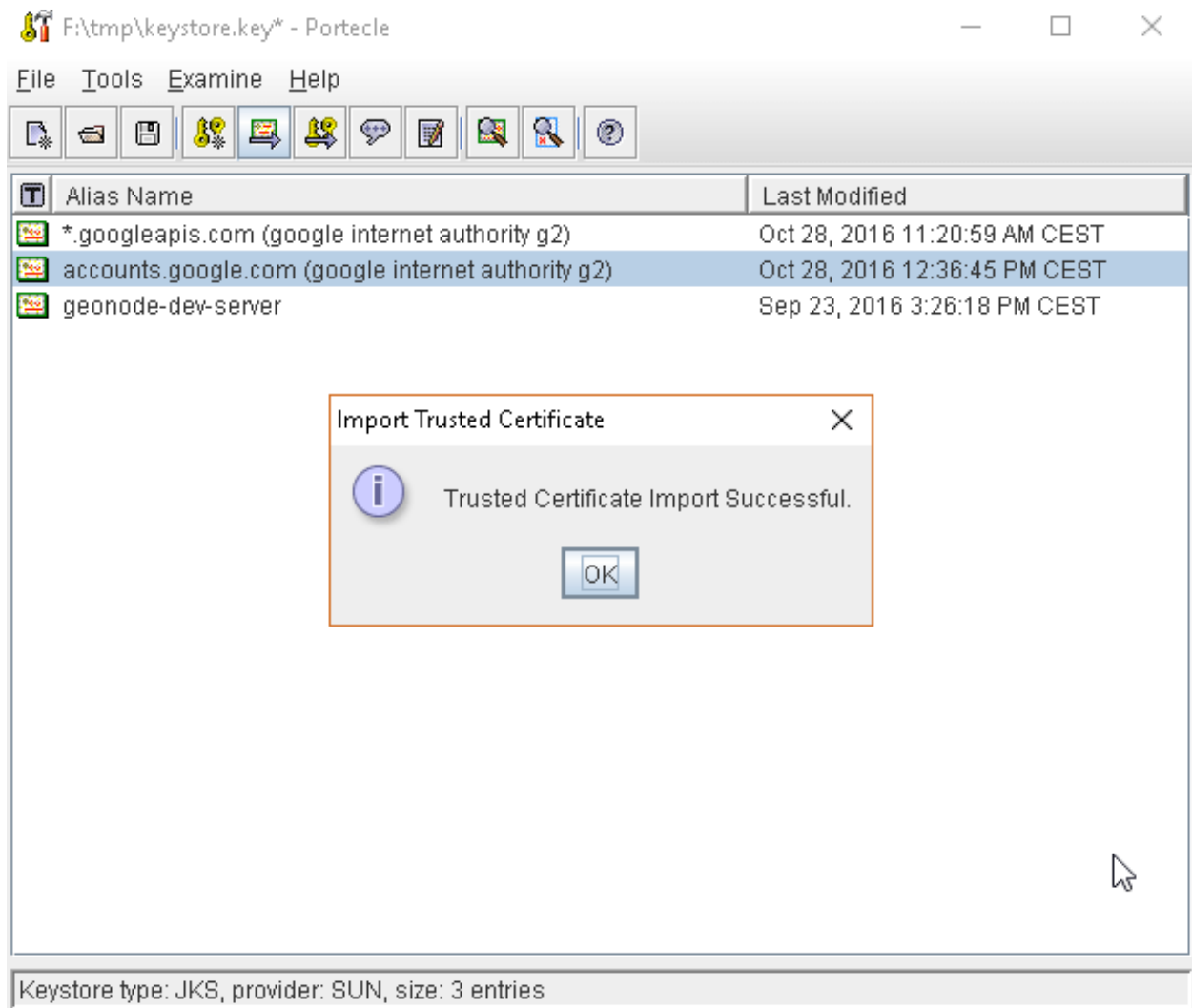


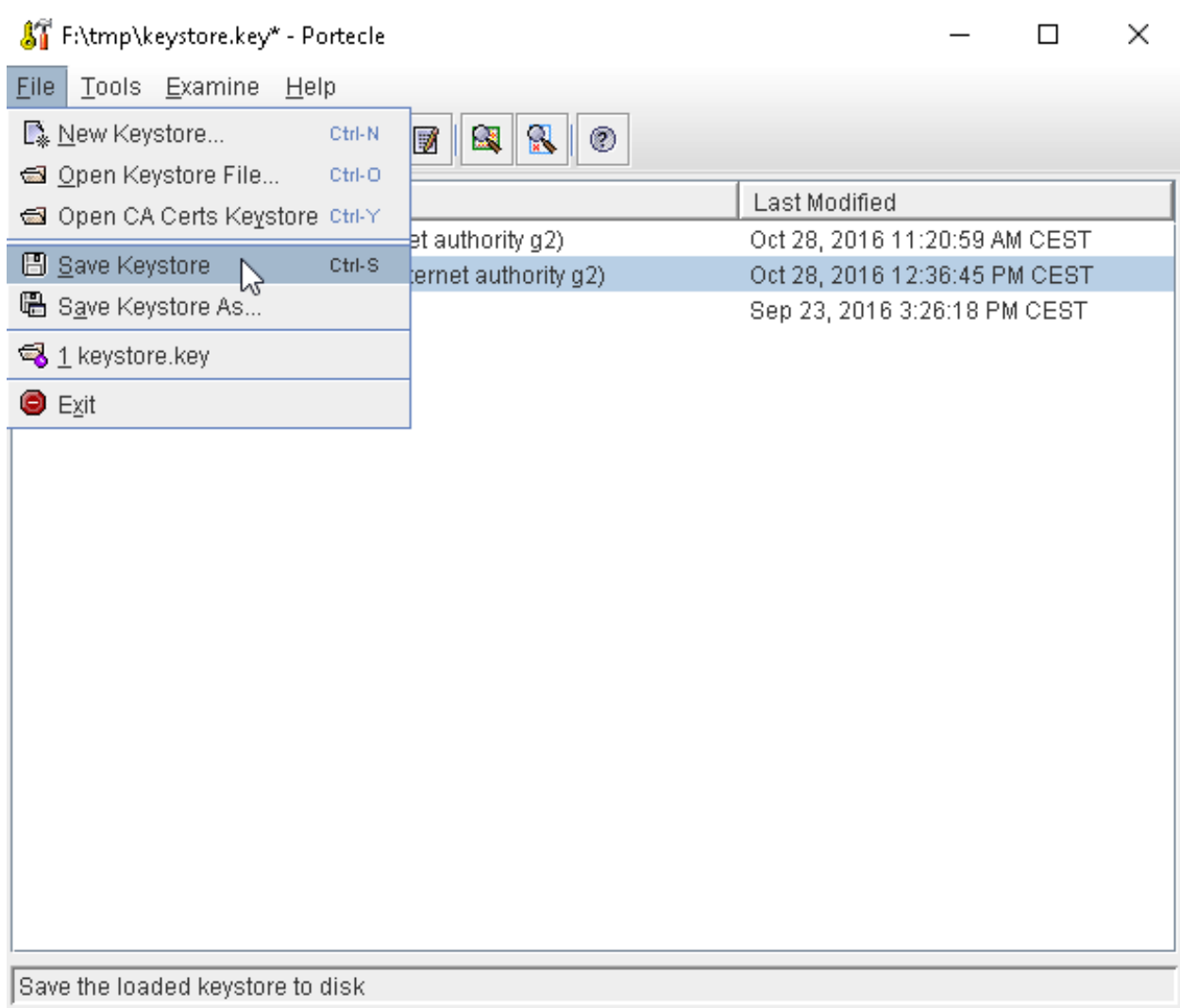












```
HOST=myhost.example.com
PORT=443
KEYSTOREFILE=dest_keystore
KEYSTOREPASS=changeme

# get the SSL certificate
openssl s_client -connect ${HOST}:${PORT} </dev/null \
    | sed -ne '/-BEGIN CERTIFICATE-/,/-END CERTIFICATE-/p' > ${HOST}.cert

# create a keystore and import certificate
keytool -import -noprompt -trustcacerts \
    -alias ${HOST} -file ${HOST}.cert \
    -keystore ${KEYSTOREFILE} -storepass ${KEYSTOREPASS}

# verify we've got it.
keytool -list -v -keystore ${KEYSTOREFILE} -storepass ${KEYSTOREPASS} -alias ${HOST}
```

GeoFence Advanced Features

GeoFence Rules Management and Tutorials

- [This tutorial](#) shows how to install and configure the Geofence Internal Server plug-in. It shows how to create rules in two ways: using the GUI and REST methods.
- GeoFence Rules can be created / updated / deleted through a REST API, accessible only by a GeoServer Admin user. You can find more details on how the GeoFence REST API works [here](#).

GeoFence Rules Storage Configuration

By default GeoFence is configured to use a filesystem based DB stored on the GeoServer Data Dir <GEOSERVER_DATA_DIR/geofence.

- It is possible also to configure GeoFence in order to use an external PostgreSQL / PostGIS Database. In order to do that please refer to the official GeoFence documentation [here](#).

3.2.1.4.2 Usage of the GeoNode's Django Administration Panel

GeoNode has an administration panel based on the Django admin which can be used to do some database operations. Although most of the operations can and should be done through the normal GeoNode interface, the admin panel provides a quick overview and management tool over the database.

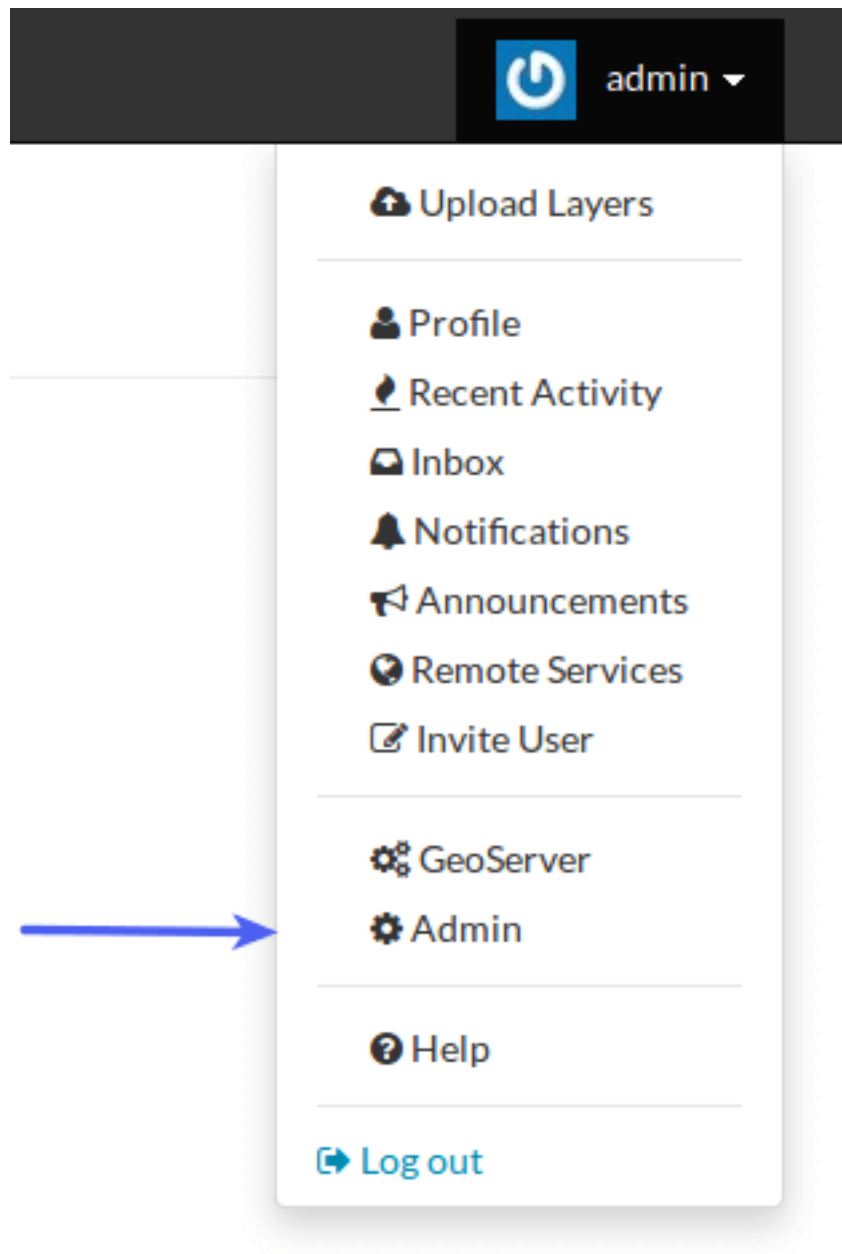
It should be highlighted that the sections not covered in this guide are meant to be managed through GeoNode.

Accessing the admin panel

Only the staff users (including the superusers) can access the admin interface.

Note: User's staff membership can be set by the admin panel itself, see how in the *Manage users and groups through the admin panel* section.

The link to access the admin interface can be found by clicking in the upper right corner on the user name, see figure



Manage users and groups through the admin panel

The admin section called Auth has the link to access the Groups while the section called People has the link to access the Users, see figure

Users



Adding a user

By clicking on the “add” link on the right of the Users link is possible to add a new user to the GeoNode site. A simple form asking for username and password will be presented, see figure

Add user
First, enter a username and password. Then, you'll be able to edit more user options.

Username:	<input type="text"/>
<small>Required. 30 characters or fewer. Letters, digits and @/./+/-/_ only.</small>	
Password:	<input type="password"/>
Password confirmation:	<input type="password"/>
<small>Enter the same password as above, for verification.</small>	

[Save and add another](#) [Save and continue editing](#) [Save](#)

Upon clicking “save” a new form will be presented asking for some personal information and the rights the user should have.

For a normal, not privileged user is enough to just fill the personal information and then confirm with “save”.

If the user has to access the admin panel or be a superuser it's enough just to tick the “staff” and “superuser” check-boxes.

Changing a user

To modify an existing user click on “Users” then on a username in the list. The same form will be presented.

Groups

Although the “Groups” permissions system is not implemented yet in GeoNode is possible to create new groups with set of permissions which will be inherited by all the group members.

The creation and management of a Group is done in a very similar way that the user one.

Manage profiles using the admin panel

So far GeoNode implements two distinct roles, that can be assigned to resources such as layers, maps or documents:

- party who authored the resource
- party who can be contacted for acquiring knowledge about or acquisition of the resource

These two profiles can be set in the GeoNode interface by accessing the metadata page and setting the “Point of Contact” and “Metadata Author” fields respectively.

Django administration

Home > Auth > Users > bob

✓ The user "bob" was added successfully. You may edit it again below.

Change user

Username:
 Required. 30 characters or fewer. Letters, digits and @/./+/-/_ only.

Password: **algorithm:** pbkdf2_sha256 **iterations:** 10000 **salt:** W7MOfl***** **hash:** 76eKsA*****
 Raw passwords are not stored, so there is no way to see this user's password, but you can change the password using [this form](#).

Personal info

First name:

Last name:

Email address:

Permissions

☒ **Active**
 Designates whether this user should be treated as active. Unselect this instead of deleting accounts.

☐ **Staff status**
 Designates whether the user can log into this admin site.

☐ **Superuser status**
 Designates that this user has all permissions without explicitly assigning them.

Groups: The groups this user belongs to. A user will get all permissions granted to each of his/her group. Hold down "Control", or "Command" on a Mac, to select more than one.

Available groups ⓘ

🔍

Chosen groups ⓘ

+

Django administration

Welcome, **Leonardo**. Change password / Log out

Home > Auth > Users

✓ The user "admin" was changed successfully.

Select user to change

🔍 Search

Action: 0 of 5 selected Go

Username	Email address	First name	Last name	Staff status
<input checked="" type="checkbox"/> admin	leonardo@example.com	Leonardo	da Vinci	✓
<input type="checkbox"/> colombo	colombo@example.com	Cristoforo	Colombo	✗
<input type="checkbox"/> galileo	galileo@example.com	Galileo	Galilei	✗
<input type="checkbox"/> magellano	magellano@example.com	Fernando	Magellano	✓
<input type="checkbox"/> pitagora	pitagora@example.com	Pitagora		✗

5 users

Filter

By staff status

All
 Yes
 No

By superuser status

All
 Yes
 No

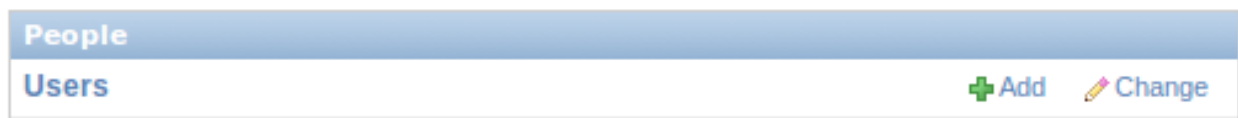
By active

All
 Yes
 No

Is possible for an administrator to add new roles if needed, by clicking on the “Add Role” button in the “Base” -> “Contact Roles” section:

The screenshot shows the 'Add contact role' form in the GeoNode administration interface. The form has three main sections: 'Resource' with a text input field labeled 'Resource name..', 'Contact' with a text input field labeled 'Type some text to search', and 'Role' with a dropdown menu. Below the dropdown menu, there is a small text label 'function performed by the responsible party'. At the bottom right of the form, there are three buttons: 'Save and add another', 'Save and continue editing', and 'Save'.

Clicking on the “People” section (see figure) will open a web for with some personal information plus a section called “Users”.



Is important that this last section is not modified here unless the administrator is very confident in that operation.

The screenshot shows the 'Contact roles' table in the GeoNode administration interface. The table has two main columns: 'Resource' and 'Role'. Each column contains a dropdown menu with a plus icon to its right. At the bottom left of the table, there is a button labeled 'Add another Contact Role'. At the bottom right, there is a 'Delete?' link.

Manage the metadata categories using the admin panel

In the “Base” section of the admin panel there are the links to manage the metadata categories used in GeoNode

The metadata categories are:

- Regions
- Restriction Code Types
- Spatial Representation Types
- Topic Categories





The other links available should not be used.

Regions

The Regions can be updated, deleted and added on needs. Just after a GeoNode fresh installation the regions contain all of the world countries, identified by their ISO code.

Restriction Code Types

Being GeoNode strictly tied to the standards, the restrictions cannot be added/deleted or modified in their identifier. This behavior is necessary to keep the consistency in case of federation with the CSW catalogues.

Base		
Contact roles	 Add	 Change
Licenses	 Add	 Change
Links	 Add	 Change
Metadata Regions	 Add	 Change
Metadata Restriction Code Types		 Change
Metadata Spatial Representation Types		 Change
Metadata Topic Categories		 Change
Resource bases	 Add	 Change
Thumbnails	 Add	 Change

Django administration Welcome, **Leonardo**. [Change password](#) / [Log out](#)

[Home](#) > [Base](#) > [Metadata Regions](#)

Select region to change

Search Add region +

Action: Go 0 of 100 selected

<input type="checkbox"/>	Code	Name
<input type="checkbox"/>	AFG	Afghanistan
<input type="checkbox"/>	ALA	Aland Islands
<input type="checkbox"/>	ALB	Albania
<input type="checkbox"/>	DZA	Algeria
<input type="checkbox"/>	ASM	American Samoa
<input type="checkbox"/>	AND	Andorra
<input type="checkbox"/>	AGO	Angola
<input type="checkbox"/>	AIA	Anguilla
<input type="checkbox"/>	ATG	Antigua and Barbuda
<input type="checkbox"/>	ARG	Argentina

The Restrictions GeoNode description field can in any case be modified if some kind of customisation is necessary, since it's just the string that will appear on the layer metadata page. If some of the restrictions are not needed within the GeoNode instance, it is possible to hide them by unchecking the “Is choice” field.

Django administration

Welcome, **Leonardo**. [Change password](#) / [Log out](#)

Home > Base > Metadata Restriction Code Types

Select restriction code type to change

Action: Go 0 of 8 selected

Identifier	Description	GeoNode description	Is choice
<input type="checkbox"/> copyright	exclusive right to the publication, production, or sale of the rights to a literary, dramatic, musical, or artistic work, or to the use of a commercial print or label, granted by law for a specified period of time to an author, composer, artist, distributor	exclusive right to the publication, production, or sale of the rights to a literary, dramatic, musical, or artistic work, or to the use of a commercial print or label, granted by law for a specified period of time to an author, composer, artist, distributor	<input checked="" type="checkbox"/>
<input type="checkbox"/> intellectualPropertyRights	rights to financial benefit from and control of distribution of non-tangible property that is a result of creativity	rights to financial benefit from and control of distribution of non-tangible property that is a result of creativity	<input checked="" type="checkbox"/>
<input type="checkbox"/> license	formal permission to do something	formal permission to do something	<input checked="" type="checkbox"/>
<input type="checkbox"/> limitation not listed	otherRestrictions	otherRestrictions	<input checked="" type="checkbox"/>
<input type="checkbox"/> patent	government has granted exclusive right to make, sell, use or license an invention or discovery	government has granted exclusive right to make, sell, use or license an invention or discovery	<input checked="" type="checkbox"/>
<input type="checkbox"/> patentPending	produced or sold information awaiting a patent	produced or sold information awaiting a patent	<input checked="" type="checkbox"/>
<input type="checkbox"/> restricted	withheld from general circulation or disclosure	withheld from general circulation or disclosure	<input checked="" type="checkbox"/>
<input type="checkbox"/> trademark	a name, symbol, or other device identifying a product, officially registered and legally restricted to the use of the owner or manufacturer	a name, symbol, or other device identifying a product, officially registered and legally restricted to the use of the owner or manufacturer	<input checked="" type="checkbox"/>

8 Metadata Restriction Code Types

Spatial Representation Types

For this section the same concepts of the Restriction Code Types applies.

Django administration

Welcome, **Leonardo**. [Change password](#) / [Log out](#)

Home > Base > Metadata Spatial Representation Types

Select spatial representation type to change

Action: Go 0 of 6 selected

Identifier	Description	GeoNode description	Is choice
<input type="checkbox"/> grid	grid data is used to represent geographic data	grid data is used to represent geographic data	<input checked="" type="checkbox"/>
<input type="checkbox"/> stereoModel	three-dimensional view formed by the intersecting homologous rays of an overlapping pair of images	three-dimensional view formed by the intersecting homologous rays of an overlapping pair of images	<input checked="" type="checkbox"/>
<input type="checkbox"/> textTable	textual or tabular data is used to represent geographic data	textual or tabular data is used to represent geographic data	<input checked="" type="checkbox"/>
<input type="checkbox"/> tin	triangulated irregular network	triangulated irregular network	<input checked="" type="checkbox"/>
<input type="checkbox"/> vector	vector data is used to represent geographic data	vector data is used to represent geographic data	<input checked="" type="checkbox"/>
<input type="checkbox"/> vldio	scene from a video recording	scene from a video recording	<input checked="" type="checkbox"/>

6 Metadata Spatial Representation Types

Topic Categories

Also for the Topic Categories the only part editable is the GeoNode description. Being standard is assumed that every possible data type will fall under these category identifiers. If some of the categories are not needed within the GeoNode instance, it is possible to hide them by unchecking the “Is choice” field.

Django administration

Welcome, **Leonardo**. [Change password](#) / [Log out](#)

Home > Base > Metadata Topic Categories

Select topic category to change

Action: Go 0 of 20 selected

Identifier	Description	GeoNode description	Is choice
<input type="checkbox"/> biota	flora and/or fauna in natural environment. Examples: wildlife, vegetation, biological sciences, ecology, wilderness, seallife, wetlands, habitat	Biota	<input checked="" type="checkbox"/>
<input type="checkbox"/> boundaries	legal land descriptions. Examples: political and administrative boundaries	Boundaries	<input checked="" type="checkbox"/>
<input type="checkbox"/> climatologyMeteorologyAtmosphere	processes and phenomena of the atmosphere. Examples: cloud cover, weather, climate, atmospheric conditions, climate change, precipitation	Climatology Meteorology Atmosphere	<input checked="" type="checkbox"/>
<input type="checkbox"/> economy	economic activities, conditions and employment. Examples: production, labour, revenue, commerce, industry, tourism and ecotourism, forestry, fisheries, commercial or subsistence hunting, exploration and exploitation of resources such as minerals, oil and gas	Economy	<input checked="" type="checkbox"/>
<input type="checkbox"/> elevation	height above or below sea level. Examples: altitude, bathymetry, digital elevation models, slope, derived products	Elevation	<input checked="" type="checkbox"/>
<input type="checkbox"/> environment	environmental resources, protection and conservation. Examples: environmental pollution, waste storage and treatment, environmental impact assessment, monitoring environmental risk, nature reserves, landscape	Environment	<input checked="" type="checkbox"/>
<input type="checkbox"/> farming	rearing of animals and/or cultivation of plants. Examples: agriculture, irrigation, aquaculture, plantations, herding, pests and diseases affecting crops and livestock	Farming	<input checked="" type="checkbox"/>
<input type="checkbox"/> foo	baz	bar	<input checked="" type="checkbox"/>

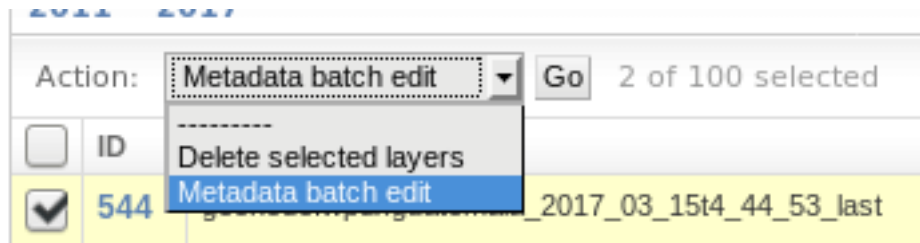
Manage layers using the admin panel

Some of the layers information can be edited directly through the admin interface although the best place is in the layer -> metadata page in GeoNode.

Layers	
Attributes	+ Add Change
Layers	+ Add Change
Styles	+ Add Change
Upload sessions	+ Add Change

Is not recommended to modify the Attributes neither the Styles.

Clicking on the Layers link will present a list of layers. Metadata information can be changed for multiple layers at once with “Metadata batch edit” action. By clicking one of the layers presents you with the page for modifying some of the information like metadata, keywords etc. It’s strongly recommended to limit the edits to the metadata and similar information.



Manage the maps using the admin panel

Currently the maps admin panel allows more metadata options that the GeoNode maps metadata page. Thus is a good place where to add some more detailed information.

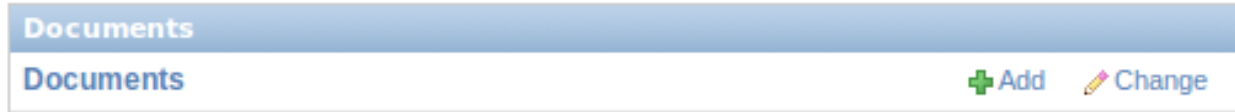
Maps	
Map layers	+ Add Change
Map snapshots	+ Add Change
Maps	+ Add Change

The “Map Layers” section should not be used.

By clicking on a map in the maps list the metadata web form will be presented. Is possible to add or modify the information here. As for the layers, the more specific entries like the layers stack or the map coordinates should not be modified. Just like for layers, you can edit metadata for multiple maps at once with “Metadata batch edit” action.

Manage the documents using the admin panel

As for the layers, most of the information related to the documents can and should be modified using the GeoNode's document metadata page.



Through the document detail page is possible to edit the metadata information. The fields related to the bounding box or the file attached should not be edited directly. Metadata information can be changed for multiple documents at once with “Metadata batch edit” action, just like for layers.

3.2.1.4.3 Management Commands for GeoNode

GeoNode comes with administrative commands to help with day to day tasks.

Below is the list of the ones that come from the GeoNode application, the full list can be obtained by doing:

```
python manage.py help
```

importlayers

Imports a file or folder with geospatial files to GeoNode.

It supports data in Shapefile and GeoTIFF format. It also picks up the styles if a `.sld` file is present.

Usage:

```
python manage.py importlayers <data_dir>
```

Additional options:

```
Usage: manage.py importlayers [options] path [path...]
```

Brings a data file **or** a directory full of data files into a GeoNode site. Layers are
→ added to the Django database, the GeoServer configuration, **and** the GeoNetwork
→ metadata index.

Options:

```
-v VERBOSITY, --verbosity=VERBOSITY      Verbosity level; 0=minimal output, 1=normal output,
                                           2=verbose output, 3=very verbose output
--settings=SETTINGS                       The Python path to a settings module, e.g.
                                           "myproject.settings.main". If this isn't provided, the
                                           DJANGO_SETTINGS_MODULE environment variable will be
                                           used.
--pythonpath=PYTHONPATH                   A directory to add to the Python path, e.g.
                                           "/home/djangoprojects/myproject".
--traceback                               Raise on exception
-u USER, --user=USER                     Name of the user account which should own the imported
                                           layers
```

(continues on next page)

(continued from previous page)

```

-i, --ignore-errors    Stop after any errors are encountered.
-o, --overwrite        Overwrite existing layers if discovered (defaults
                        False)
-k KEYWORDS, --keywords=KEYWORDS
                        The default keywords, separated by comma, for the
                        imported layer(s). Will be the same for all imported
                        layers if multiple imports are
                        done in one command
-c CATEGORY, --category=CATEGORY
                        The category for the imported
                        layer(s). Will be the same for all imported layers
                        if multiple imports are done in one command
-r REGIONS, --regions=REGIONS
                        The default regions, separated by comma, for the
                        imported layer(s). Will be the same for all imported
                        layers if multiple imports are
                        done in one command
-t TITLE, --title=TITLE
                        The title for the imported
                        layer(s). Will be the same for all imported layers
                        if multiple imports are done in one command
-p, --private          Make layer viewable only to owner
--version              show program's version number and exit
-h, --help             show this help message and exit

```

updatelayers

Update the GeoNode application with data from GeoServer.

This is useful to add data in formats that are not supported in GeoNode by default, and for example to link it to ArcSDE datastores. The updatelayers command provides several options that can be used to control how layer information is read from GeoServer and updated in GeoNode. Refer to 'Additional Options'.

Usage:

```
python manage.py updatelayers
```

Additional options:

```
Usage: manage.py updatelayers [options]
```

Update the GeoNode application with data from GeoServer

Options:

```

-v VERBOSITY, --verbosity=VERBOSITY
                        Verbosity level; 0=minimal output, 1=normal output,
                        2=verbose output, 3=very verbose output
--settings=SETTINGS    The Python path to a settings module, e.g.
                        "myproject.settings.main". If this isn't provided, the
                        DJANGO_SETTINGS_MODULE environment variable will be
                        used.
--pythonpath=PYTHONPATH
                        A directory to add to the Python path, e.g.
                        "/home/djangoprojects/myproject".
--traceback            Raise on exception

```

(continues on next page)

(continued from previous page)

```

-i, --ignore-errors    Stop after any errors are encountered.
--skip-unadvertised    Skip processing unadvertised layers from GeoServer.
--skip-geonode-registered
                        Just processing GeoServer layers still not registered
                        in GeoNode.
--remove-deleted       Remove GeoNode layers that have been deleted from
                        GeoServer.
-u USER, --user=USER  Name of the user account which should own the imported
                        layers
-f FILTER, --filter=FILTER
                        Only update data the layers that match the given
                        filter
-s STORE, --store=STORE
                        Only update data the layers for the given geoserver
                        store name
-w WORKSPACE, --workspace=WORKSPACE
                        Only update data on specified workspace
--version              show program's version number and exit
-h, --help             show this help message and exit

```

createvectorlayer

Create an empty PostGIS vector layer in GeoNode.

Usage:

```
python manage.py createvectorlayer name [options]
```

Additional options:

```

manage.py createvectorlayer [-h] [--version] [-v {0,1,2,3}]
                        [--settings SETTINGS]
                        [--pythonpath PYTHONPATH] [--traceback]
                        [--no-color] [--user USER]
                        [--geometry GEOMETRY]
                        [--attributes ATTRIBUTES] [--title TITLE]
                        name

```

Create an empty PostGIS vector layer in GeoNode.

positional arguments:

name

optional arguments:

```

-h, --help            show this help message and exit
--version            show program's version number and exit
-v {0,1,2,3}, --verbosity {0,1,2,3}
                        Verbosity level; 0=minimal output, 1=normal output,
                        2=verbose output, 3=very verbose output
--settings SETTINGS  The Python path to a settings module, e.g.
                        "myproject.settings.main". If this isn't provided, the
                        DJANGO_SETTINGS_MODULE environment variable will be
                        used.
--pythonpath PYTHONPATH
                        A directory to add to the Python path, e.g.

```

(continues on next page)

(continued from previous page)

```

--traceback           "/home/djangoprojects/myproject".
                      Raise on CommandError exceptions
--no-color            Don't colorize the command output.
--user USER           Name of the user account which should own the created
                      layer
--geometry GEOMETRY   Geometry type of the layer to be created. Can be
                      Point, LineString or Polygon. Default is Point
--attributes ATTRIBUTES
                      A json representation of the attributes to create.
                      Example: { "field_str": "string", "field_int":
                      "integer", "field_date": "date", "field_float":
                      "float"}
--title TITLE         Title for the layer to be created.

```

fixsitename

Uses SITENAME and SITEURL to set the values of the default site object.

This information is used in the page titles and when sending emails from GeoNode, for example, new registrations.

Usage:

```
python manage.py fixsitename
```

Additional options:

```
Usage: manage.py fixsitename [options]
```

Options:

```

-v VERBOSITY, --verbosity=VERBOSITY
                      Verbosity level; 0=minimal output, 1=normal output,
                      2=verbose output, 3=very verbose output
--settings=SETTINGS  The Python path to a settings module, e.g.
                      "myproject.settings.main". If this isn't provided, the
                      DJANGO_SETTINGS_MODULE environment variable will be
                      used.
--pythonpath=PYTHONPATH
                      A directory to add to the Python path, e.g.
                      "/home/djangoprojects/myproject".
--traceback           Raise on exception
--version             show program's version number and exit
-h, --help            show this help message and exit

```

delete_orphaned_files

Deletes orphaned files of deleted documents.

Usage:

```
python manage.py delete_orphaned_files
```

delete_orphaned_thumbs

Deletes orphaned thumbnails of deleted GeoNode resources (Layers, Maps and Documents).

Usage:

```
python manage.py delete_orphaned_thumbs
```

fix_baselayers

Fix base layers for all of the GeoNode maps or for a given map.

Usage:

```
# fix base layers for all of the GeoNode map
python manage.py fix_baselayers

# fix base layers for a given map
python manage.py fix_baselayers map_id
```

sync_geofence

Synchronize GeoNode permissions in GeoFence. This can be useful when upgrading GeoNode.

Usage:

```
# synchronize all layers
python manage.py sync_geofence

# synchronize all layers which contain a given search string in their name
python manage.py sync_geofence --layername cambridge
```

find_geoserver_broken_layers

Find GeoNode layers with a missing GeoServer resource.

Usage:

```
# search the whole catalog
python manage.py find_geoserver_broken_layers

# search all layers which contain a given search string in their name and owned by a
→given user
python manage.py sync_geofence --layername cambridge --owner bob

# remove layers which are broken
python manage.py find_geoserver_broken_layers --remove
```

3.2.1.4.4 Configuring Alternate CSW Backends

`pycsw` is the default CSW server implementation provided with GeoNode. This section will explain how to configure GeoNode to operate against alternate CSW server implementations.

Supported CSW server implementations

GeoNode additionally supports the following CSW server implementations:

- [GeoNetwork opensource](#)
- [deegree](#)

Since GeoNode communicates with alternate CSW configurations via HTTP, the CSW server can be installed and deployed independent of GeoNode if desired.

Installing the CSW

GeoNetwork opensource Installation

- Deploy GeoNetwork opensource by downloading `geonetwork.war` (see <http://geonetwork-opensource.org/downloads.html>) and deploying into your servlet container
- Follow the instructions at <http://geonetwork-opensource.org/manuals/3.0.5/eng/users/user-guide/quick-start/index.html> to complete the installation
- test the server with a GetCapabilities request (<http://localhost:8080/geonetwork/srv/en/csw?service=CSW&version=2.0.2&request=GetCapabilities>)

See <http://geonetwork-opensource.org/docs.html> for further documentation.

deegree Installation

- Deploy deegree by downloading the `deegree3 cswDemo .war` (see <http://wiki.deegree.org/deegreeWiki/DownloadPage>) and deploying into your servlet container
- Create a PostGIS-enabled PostgreSQL database
- Follow the instructions at http://wiki.deegree.org/deegreeWiki/deegree3/CatalogueService#Run_your_own_installation to complete the installation
- test the server with a GetCapabilities request (<http://localhost:8080/deegree-csw-demo-3.0.4/services?service=CSW&version=2.0.2&request=GetCapabilities>)

See <http://wiki.deegree.org/deegreeWiki/deegree3/CatalogueService> for further documentation.

Customizing GeoNode CSW configuration

At this point, the CSW alternate backend is ready for GeoNode integration. GeoNode's CSW configuration (in `geonode/settings.py`) must be updated to point to the correct CSW. The example below exemplifies GeoNetwork as an alternate CSW backend:

```
# CSW settings
CATALOGUE = {
    'default': {
        # The underlying CSW implementation
        # default is pycsw in local mode (tied directly to GeoNode Django DB)
        #'ENGINE': 'geonode.catalogue.backends.pycsw_local',
        # pycsw in non-local mode
        #'ENGINE': 'geonode.catalogue.backends.pycsw',
```

(continues on next page)

(continued from previous page)

```
# GeoNetwork opensource
'ENGINE': 'geonode.catalogue.backends.geonetwork',
# deegree and others
#'ENGINE': 'geonode.catalogue.backends.generic',

# The FULLY QUALIFIED base url to the CSW instance for this GeoNode
#'URL': '%scatalogue/csw' % SITEURL,
'URL': 'http://localhost:8080/geonetwork/srv/en/csw',
#'URL': 'http://localhost:8080/deegree-csw-demo-3.0.4/services',

# login credentials (for GeoNetwork)
'USER': 'admin',
'PASSWORD': 'admin',
}
}
```

3.2.1.4.5 LDAP configuration

Library Dependencies

LDAP support requires LDAP development libraries

Centos/RHEL

```
$sudo yum install -y openldap-devel
```

Ubuntu

```
$sudo apt-get install -y libldap2-dev
```

Install

```
$pip install python-ldap django-auth-ldap
```

Configure

Add the following to your `local_settings.py`

Note: Example only - more details can be found [here](#)

```
import ldap
from django_auth_ldap.config import LDAPSearch

AUTHENTICATION_BACKENDS = (
    'django_auth_ldap.backend.LDAPBackend',
    'django.contrib.auth.backends.ModelBackend',
    'guardian.backends.ObjectPermissionBackend',
)
AUTH_LDAP_SERVER_URI = 'ldap://ldap.example.com'
LDAP_SEARCH_DN = 'ou=users,dc=example,dc=com'
AUTH_LDAP_USER = '(uid=%(user)s)'
```

(continues on next page)

(continued from previous page)

```

AUTH_LDAP_BIND_DN = '{ADD_BIND_DN_IF_REQUIRED}'
AUTH_LDAP_BIND_PASSWORD = '{ADD_BIND_PASSWORD_IF_REQUIRED}'
AUTH_LDAP_USER_ATTR_MAP = {
    'first_name': 'givenName', 'last_name': 'sn', 'email': 'mail',
}
AUTH_LDAP_USER_SEARCH = LDAPSearch(LDAP_SEARCH_DN,
                                   ldap.SCOPE_SUBTREE, AUTH_LDAP_USER)

```

3.2.1.4.6 Customize the look and feel

Warning: These instructions are only valid if you've installed GeoNode following the guide at [Setup & Configure HTTPD !!](#)

Since version 2.8.1, it is possible to do change some elements of look and feel (such as colors, logo, etc.) using the Geonode Theme application from the administration. For more advanced changes, you first have to set up your own geonode project from a template. If you've successfully done this, you can go further and start theming your geonode project.

Setup steps

Warning: These instructions are only valid if you've installed GeoNode following the guide at [Setup & Configure HTTPD !!](#)

If you are working remotely, you should first connect to the machine that has your GeoNode installation. You will need to perform the following steps in a directory where you intend to keep your newly created project.

```

1 $ apt-get install python-django
2 $ django-admin startproject my_geonode --template=https://github.com/GeoNode/geonode-
  ↳ project/archive/master.zip -epy,rst
3 $ cd my_geonode
4 $ sudo pip install -e . --upgrade --no-cache
5 $ python manage.py migrate

```

Note: You should NOT use the name *geonode* for your project as it will conflict with your default geonode package name.

These commands create a new template based on the geonode example project.

Make sure that the directories are reachable and have the correct rights for the users **geonode** and **www-data**:

```

1 $ sudo chown -Rf geonode: *
2 $ sudo chmod -Rf 775 my_geonode

```

If you have a brand new installation of GeoNode, rename the `/home/geonode/geonode/local_settings.py.sample` to `local_settings.py` and edit it's content by setting the SITEURL and SITENAME. This file will be your main settings file for your project. It inherits all the settings from the original one plus you can override the ones that you need.

Note: You can also decide to copy the `/home/geonode/geonode/local_settings.py.sample` to `/path/to/my_geonode/my_geonode/local_settings.py` in order to keep all the custom settings confined into the new project.

Warning: In order for the edits to the `local_settings.py` file to take effect, you have to restart apache.

Edit the file `/etc/apache2/sites-available/geonode.conf` and change the following directive from:

```
WSGIScriptAlias / /home/geonode/geonode/wsgi/geonode.wsgi
```

to:

```
WSGIScriptAlias / /path/to/my_geonode/my_geonode/wsgi.py
```

Edit the file `/path/to/my_geonode/my_geonode/wsgi.py` and add the following lines at the beginning:

```
1 from django.core.wsgi import get_wsgi_application
```

Edit the file `/etc/apache2/sites-available/geonode.conf` and modify the **DocumentRoot** as follows:

Note: It's a good practice to make copies and backups of the configuration files before modifying or updating them in order to revert the configuration at the previous state if something goes wrong.

```
1 <VirtualHost *:80>
2     ServerName http://localhost
3     ServerAdmin webmaster@localhost
4     DocumentRoot /home/geonode/my_geonode/my_geonode
5
6     ErrorLog /var/log/apache2/error.log
7     LogLevel warn
8     CustomLog /var/log/apache2/access.log combined
9
10    WSGIProcessGroup geonode
11    WSGIPassAuthorization On
12    WSGIScriptAlias / /home/geonode/my_geonode/my_geonode/wsgi.py
13
14    <Directory "/home/geonode/my_geonode/my_geonode/">
15        <Files wsgi.py>
16            Order deny,allow
17            Allow from all
18            Require all granted
19        </Files>
20
21        Order allow,deny
22        Options Indexes FollowSymLinks
23        Allow from all
24        IndexOptions FancyIndexing
25    </Directory>
26
27    ...
```

Then regenerate the static **JavaScript** and **CSS** files from `/path/to/my_geonode/` and restart apache

```
1 $ python manage.py collectstatic
2 $ sudo service apache2 restart
```

Customize the Look & Feel

Now you can edit the templates in `my_geonode/templates`, the CSS and images to match your needs like shown in `customize.theme_admin!`

Note: After going through the theming guide you'll have to return to this site to execute one more command in order to finish the theming!

When you've done the changes, run the following command in the `my_geonode` folder:

```
1 $ cd /home/geonode/my_geonode
2 $ python manage.py collectstatic
```

And now you should see all the changes you've made to your GeoNode.

Source code revision control

It is recommended that you immediately put your new project under source code revision control. The GeoNode development team uses Git and GitHub and recommends that you do the same. If you do not already have a GitHub account, you can easily set one up. A full review of Git and distributed source code revision control systems is beyond the scope of this tutorial, but you may find the [Git Book](#) useful if you are not already familiar with these concepts.

1. Create a new repository in GitHub. You should use the GitHub user interface to create a new repository for your new project.

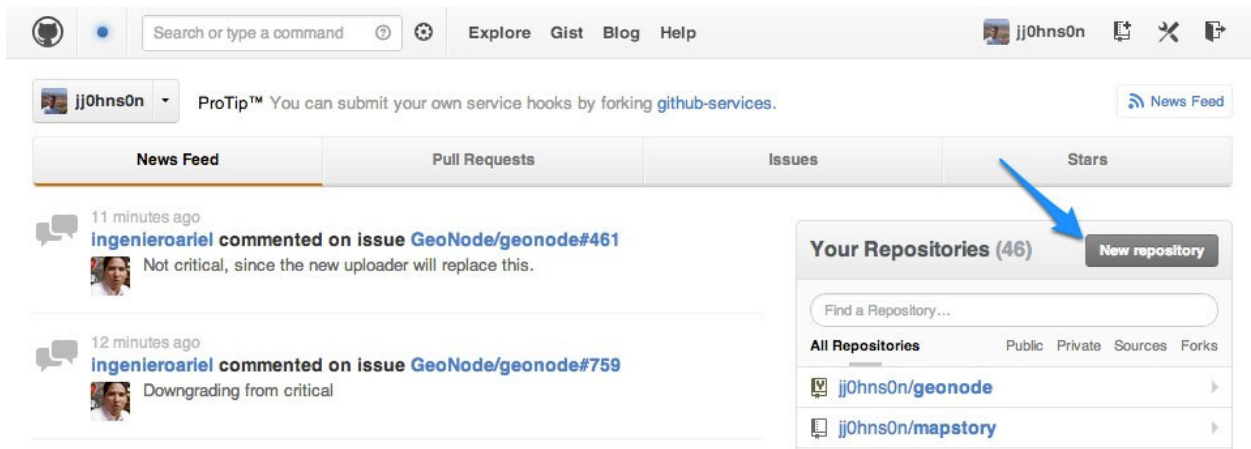


Fig. 75: Creating a new GitHub Repository From GitHub's Homepage

2. Initialize your own repository in the `my_geonode` folder:

```
1 $ sudo git init
```

3. Add the remote repository reference to your local git configuration:

Owner: **jj0hns0n** / Repository name: **my_geonode** ✓

Great repository names are short and memorable. Need inspiration? How about **north-american-octo-cyril**.

Description (optional): My GeoNode Project

☒ **Public**
Anyone can see this repository. You choose who can commit.

☐ **Private**
You choose who can see and commit to this repository.

☐ **Initialize this repository with a README**
This will allow you to `git clone` the repository immediately.

Add .gitignore: **None**

Create repository →

Fig. 76: Specifying new GitHub Repository Parameters

jj0hns0n / my_geonode build status: unknown

Unwatch Star 0

Code Network Pull Requests 0 Issues 0 Wiki Graphs Settings

Quick setup — if you've done this kind of thing before

Setup in Mac or HTTP SSH `https://github.com/jj0hns0n/my_geonode.git`

We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

Create a new repository on the command line

```
touch README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/jj0hns0n/my_geonode.git
git push -u origin master
```

Push an existing repository from the command line

```
git remote add origin https://github.com/jj0hns0n/my_geonode.git
git push -u origin master
```

Fig. 77: Your new Empty GitHub Repository

```
1 $ sudo git remote add origin <https url of your custom repo>
```

4. Add your project files to the repository:

```
1 $ sudo git add .
```

5. Commit your changes:

```
1 # Those two command must be issued ONLY once
2 $ sudo git config --global user.email "my@email"
3 $ sudo git config --global user.name "myuser"
4
5 $ sudo git commit -am "Initial commit"
```

6. Push to the remote repository:

```
1 $ sudo git push origin master
```

Further Reading

- If you want more information on how to GitHub works and how to contribute to GeoNode project, go to the section “[Contributing to GeoNode](#)”
- If you want to customize the Logo and Style of **my_geonode**, go to the section “[Theming your GeoNode project](#)”

Here below you can find some more details about the custom project structure and info on some of the most important Python files you may want to edit.

The following section is mostly oriented to advanced users and developers.

Project structure

Your GeoNode project will now be structured as depicted below:

```
|-- README.rst
|-- manage.py
|-- my_geonode
|   |-- __init__.py
|   |-- settings.py
|   |-- local_settings.py
|   |-- static
|   |   |-- README
|   |   |-- css
|   |   |   |-- site_base.css
|   |   |-- img
|   |   |   |-- README
|   |   |-- js
|   |       |-- README
|   |-- templates
|   |   |-- site_base.html
|   |   |-- site_index.html
|   |-- urls.py
|   |-- wsgi.py
|-- setup.py
```

You can also view your project on GitHub.

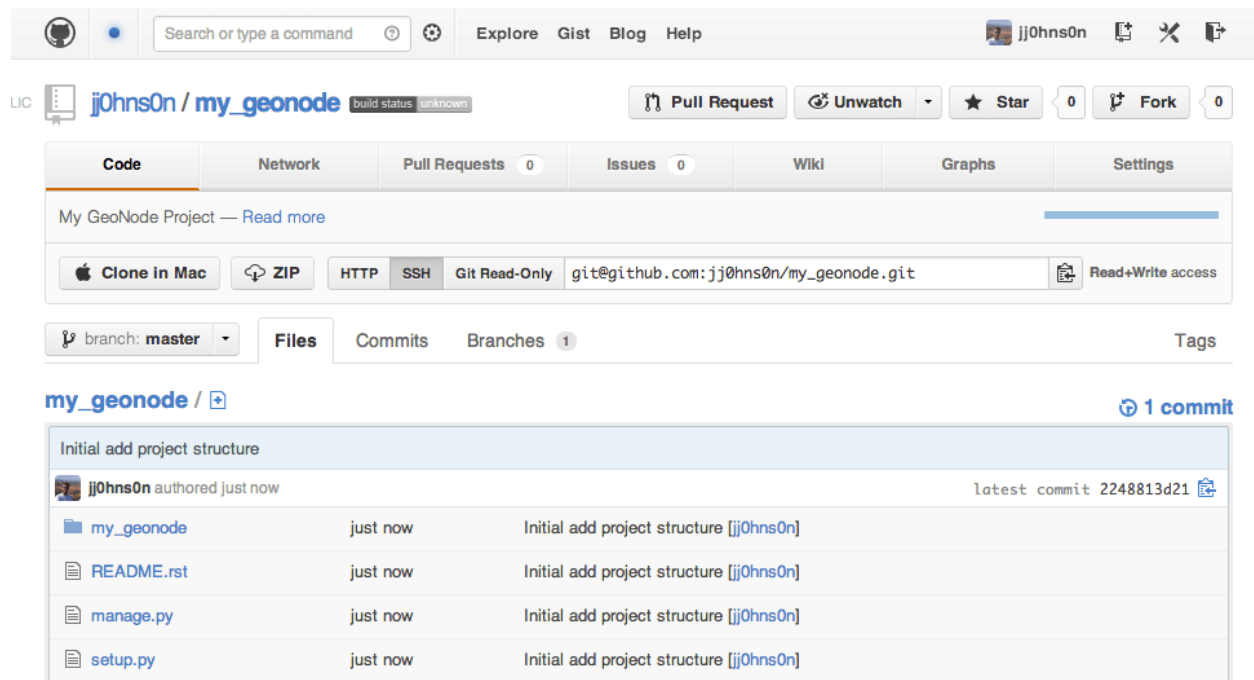


Fig. 78: Viewing your project on GitHub

Each of the key files in your project are described below.

manage.py

`manage.py` is the main entry point for managing your project during development. It allows running all the management commands from each app in your project. When run with no arguments, it will list all of the management commands.

settings.py

`settings.py` is the primary settings file for your project. It imports the settings from the system geonode and adds the local paths. It is quite common to put all sensible defaults here and keep deployment specific configuration in the `local_settings.py` file. All of the possible settings values and their meanings are detailed in the Django documentation.

A common paradigm for handling ‘local settings’ (and in other areas where some python module may not be available) is:

```
try: from local_settings import *
except: pass
```

This is not required and there are many other solutions to handling varying deployment configuration requirements.

urls.py

`urls.py` is where your application specific URL routes go. Additionally, any *overrides* can be placed here, too.

wsgi.py

This is a generated file to make deploying your project to a WSGI server easier. Unless there is very specific configuration you need, `wsgi.py` can be left alone.

setup.py

There are several packaging options in python but a common approach is to place your project metadata (version, author, etc.) and dependencies in `setup.py`.

This is a large topic and not necessary to understand while getting started with GeoNode development but will be important for larger projects and to make development easier for other developers.

More: <https://docs.python.org/2/distutils/setupscript.html>

static

The `static` directory will contain your fixed resources: CSS, HTML, images, etc. Everything in this directory will be copied to the final media directory (along with the *static* resources from other apps in your project).

templates

All of your projects templates go in the `templates` directory. While no organization is required for your project specific templates, when overriding or replacing a template from another app, the path must be the same as the template to be replaced.

Staying in sync with mainline GeoNode

Warning: These instructions are only valid if you've installed GeoNode using **apt-get** !!

One of the primary reasons to set up your own GeoNode project using this method is so that you can stay in sync with the mainline GeoNode as the core development team makes new releases. Your own project should not be adversely affected by these changes, but you will receive bug fixes and other improvements by staying in sync.

Upgrade GeoNode:

```
$ apt-get update
$ apt-get install geonode
```

Verify that your new project works with the upgraded GeoNode:

```
$ python manage.py runserver
```

Navigate to <http://localhost:8000>.

Warning: These instructions are only valid if you've installed GeoNode following the guide at *Setup & Configure HTTPD* !!

Upgrading from source code repo:

Upgrade GeoNode:

```
$ cd /home/geonode/geonode
$ git pull origin master
```

Verify that your new project works with the upgraded GeoNode:

```
$ python manage.py runserver
```

Navigate to <http://localhost:8000>.

Theming your GeoNode project

There are a range of options available to you if you want to change the default look and feel of your GeoNode project. Since GeoNode's style is based on [Bootstrap](#) you will be able to make use of all that Bootstrap has to offer in terms of theme customization. You should consult Bootstrap's documentation as your primary guide once you are familiar with how GeoNode implements Bootstrap and how you can override GeoNode's theme and templates in your own project.

Logos and graphics

GeoNode intentionally does not include a large number of graphics files in its interface. This keeps page loading time to a minimum and makes for a more responsive interface. That said, you are free to customize your GeoNode's interface by simply changing the default logo, or by adding your own images and graphics to deliver a GeoNode experience the way you envision it.

Your GeoNode project has a directory already set up for storing your own images at `<my_geonode>/static/img`. You should place any image files that you intend to use for your project in this directory.

Let's walk through an example of the steps necessary to change the default logo.

1. Change to the `img` directory:

```
$ cd <my_geonode>/static/img
```

2. If you haven't already, obtain your logo image. The URL below is just an example, so you will need to change this URL to match the location of your file or copy it to this location:

```
$ sudo wget http://www2.sta.uwi.edu/~anikov/UWI-logo.JPG
$ sudo chown -Rf geonode: .
```

3. Change to the `css` directory:

```
$ cd ../../..
```

4. Override the CSS that displays the logo by editing `<my_geonode>/static/css/site_base.css` with your favorite editor and adding the following lines, making sure to update the width, height, and URL to match the specifications of your image.

```
$ sudo vi site_base.css
```

```
.navbar-brand {
  width: 373px;
  height: 79px;
  background: transparent url("../img/UWI-logo.JPG") no-repeat scroll 15px 0px;
}
```

5. Restart your GeoNode project and look at the page in your browser:

```
$ cd /home/geonode
$ sudo rm -Rf geonode/geonode/static_root/*
$ cd my_geonode
$ python manage.py collectstatic
$ sudo service apache2 restart
```

Note: It is a good practice to cleanup the **static_folder** and the Browser Cache before reloading in order to be sure that the changes have been correctly taken and displayed on the screen.

Visit your site at <http://localhost/> or the remote URL for your site.

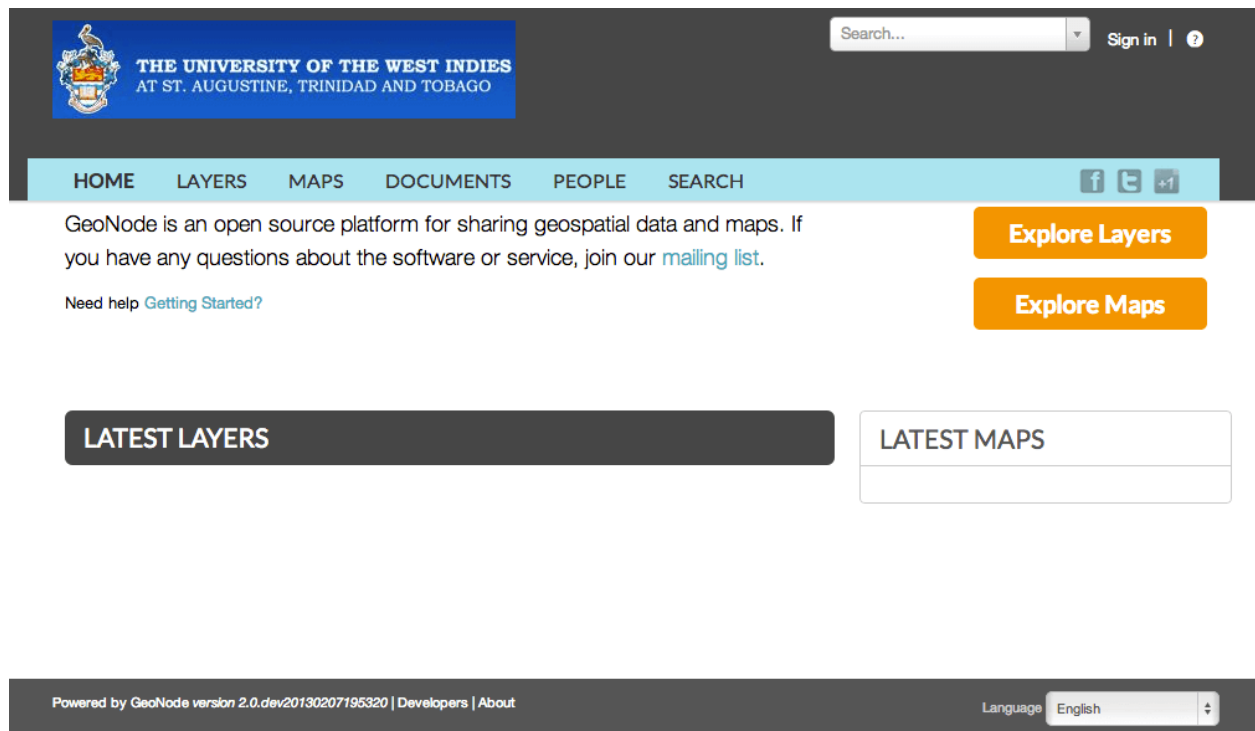


Fig. 79: Custom logo

You can see that the header has been expanded to fit your graphic. In the following sections you will learn how to customize this header to make it look and function the way you want.

Note: You should commit these changes to your repository as you progress through this section, and get in the habit of committing early and often so that you and others can track your project on GitHub. Making many atomic commits

and staying in sync with a remote repository makes it easier to collaborate with others on your project.

Cascading Style Sheets

In the last section you already learned how to override GeoNode's default CSS rules to include your own logo. You are able to customize any aspect of GeoNode's appearance this way. In the last screenshot, you saw that the main area in the homepage is covered up by the expanded header.

First, we'll walk through the steps necessary to displace it downward so it is no longer hidden, then change the background color of the header to match the color in our logo graphic.

1. Reopen `<my_geonode>/static/css/site_base.css` in your editor and add the following rule after the one added in the previous step:

```
$ cd /home/geonode/my_geonode/my_geonode/static/css
$ sudo vi site_base.css
```

```
#wrap {
    margin: 75px 75px;
}
```

2. Add a rule to change the background color of the header to match the logo graphic we used:

```
.navbar-inverse {
    background: #0e60c3;
}
```

3. Your project CSS file should now look like this:

```
.navbar-brand {
    width: 373px;
    height: 79px;
    background: url(../img/UWI-logo.JPG) no-repeat;
}

#wrap {
    margin: 75px 75px;
}

.navbar-inverse {
    background: #0e60c3;
}
```

4. Restart the development server and reload the page:

```
$ python manage.py collectstatic
$ sudo service apache2 restart
```

Note: You can continue adding rules to this file to override the styles that are in the GeoNode base CSS file which is built from `base.less`. You may find it helpful to use your browser's development tools to inspect elements of your site that you want to override to determine which rules are already applied. See the screenshot below. Another section of this workshop covers this topic in much more detail.

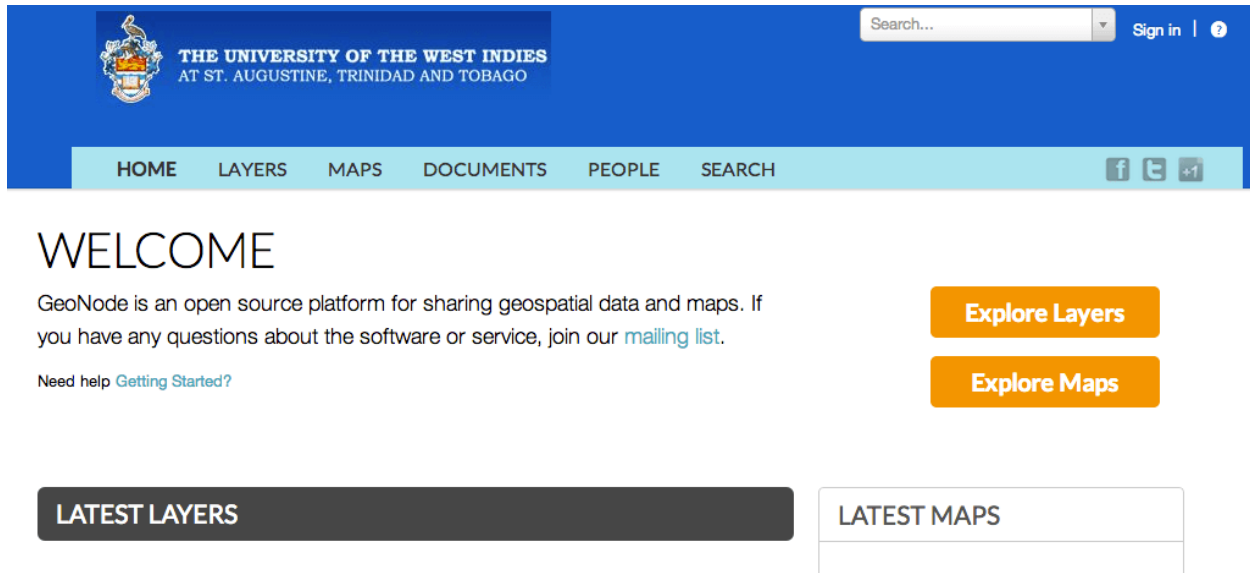


Fig. 80: CSS overrides

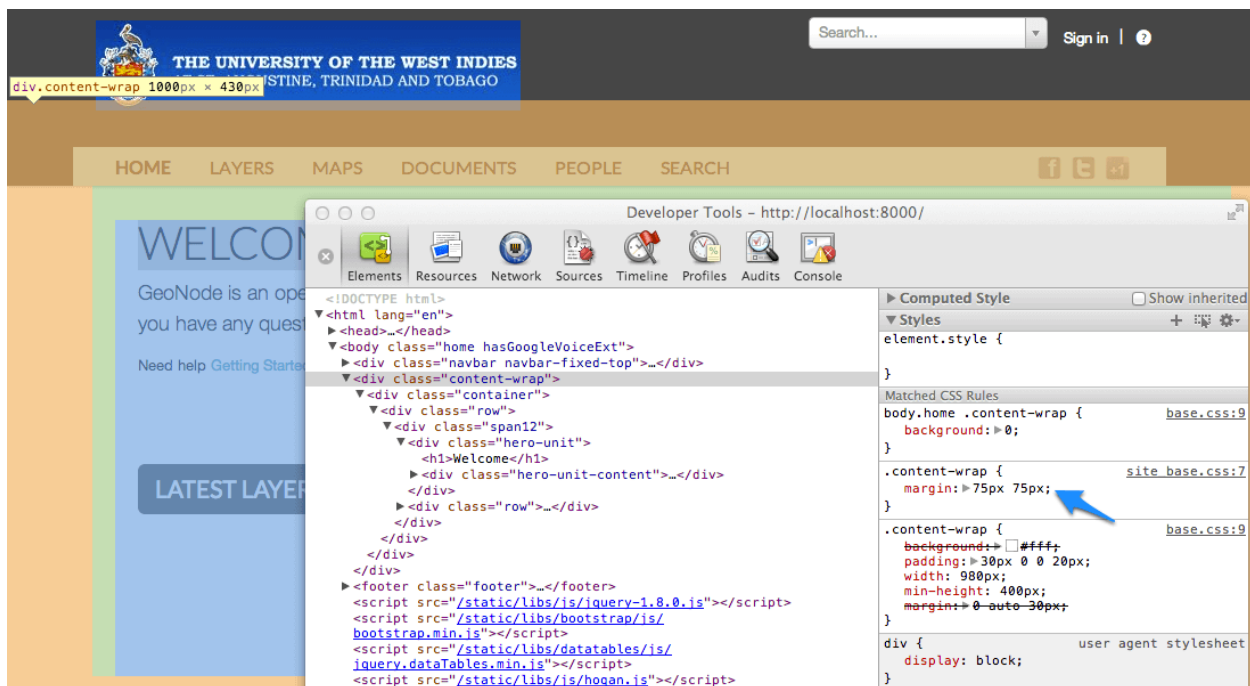


Fig. 81: Screenshot of using Chrome's debugger to inspect the CSS overrides

Templates and static pages

Now that we have changed the default logo and adjusted our main content area to fit the expanded header, the next step is to update the content of the homepage itself. Your GeoNode project includes two basic templates that you will use to change the content of your pages.

The file `site_base.html` (in `<my_geonode>/templates/`) is the basic template that all other templates inherit from and you will use it to update things like the header, navbar, site-wide announcement, footer, and also to include your own JavaScript or other static content included in every page in your site. It's worth taking a look at [GeoNode's base file on GitHub](#). You have several blocks available to you to for overriding, but since we will be revisiting this file in future sections of this workshop, let's just look at it for now and leave it unmodified.

Open `<my_geonode>/templates/site_base.html` in your editor:

```
$ cd /home/geonode/my_geonode/my_geonode/templates
$ sudo vi site_base.html

.. code-block:: html

{% extends "base.html" %}
{% block extra_head %}
    <link href="{{ STATIC_URL }}css/site_base.css" rel="stylesheet"/>
{% endblock %}
```

You will see that it extends from `base.html`, which is the GeoNode template referenced above and it currently only overrides the `extra_head` block to include our project's `site_base.css` which we modified in the previous section. You can see on [line 22 of the GeoNode base.html template](#) that this block is included in an empty state and is set up specifically for you to include extra CSS files as your project is already set up to do.

Now that we have looked at `site_base.html`, let's actually override a different template.

The file `site_index.html` is the template used to define your GeoNode project's homepage. It extends GeoNode's default `index.html` template and gives you the option to override specific areas of the homepage like the hero area, but also allows you leave area like the "Latest Layers" and "Maps" and the "Contribute" section as they are. You are of course free to override these sections if you choose and this section shows you the steps necessary to do that below.

1. Open `<my_geonode>/templates/site_index.html` in your editor.
2. Edit the `<h1>` element on line 9 to say something other than "Welcome":

```
<h1>{% trans "UWI GeoNode" %}</h1>
```

3. Edit the introductory paragraph to include something specific about your GeoNode project:

```
<p>
    {% blocktrans %}
    UWI's GeoNode is setup for students and faculty to collaboratively
    create and share maps for their class projects. It is maintained by the
    UWI Geographical Society.
    {% endblocktrans %}
</p>
```

4. Change the *Getting Started* link to point to another website:

```
<span>
    For more information about the UWI Geographical society,
    <a href="http://uwigsmona.weebly.com/">visit our website</a>
</span>
```

5. Add a graphic to the hero area above the paragraph replaced in step 3:

```
<img src = 'http://uwigsmona.weebly.com/uploads/1/3/2/4/13241997/1345164334.png'>
```

6. Your edited `site_index.html` file should now look like this:

```
{% extends 'index.html' %}
{% load i18n %}
{% comment %}
This is where you can override the hero area block. You can simply modify the
↪content below or replace it wholesale to meet your own needs.
{% endcomment %}
{% block hero %}
<div class="jumbotron">
  <div class="container">
    <h1>{% trans "UWI GeoNode" %}</h1>
    <div class="hero-unit-content"/>
    <div class="intro">
      <img src = 'http://uwigsmona.weebly.com/uploads/1/3/2/4/13241997/
↪1345164334.png'>
    </div>
    <p>
      {% blocktrans %}
      UWI's GeoNode is setup for students and faculty to collaboratively
      create and share maps for their class projects. It is maintained by
↪the
      UWI Geographical Society.
      {% endblocktrans %}
    </p>
    <span>
      For more information about the UWI Geographical society,
      <a href="http://uwigsmona.weebly.com/">visit our website</a>
    </span>
    </div>
  </div>
{% endblock %}
```

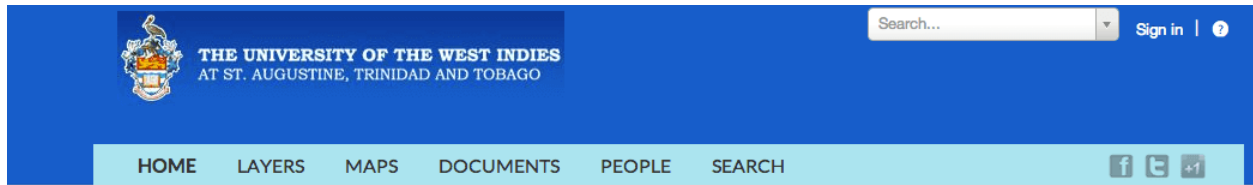
7. Refresh your GeoNode project and view the changes in your browser at <http://localhost/> or the remote URL for your site:

```
$ python manage.py collectstatic
$ sudo service apache2 restart
```

From here you can continue to customize your `site_index.html` template to suit your needs. This workshop will also cover how you can add new pages to your GeoNode project site.

Other theming options

You are able to change any specific piece of your GeoNode project's style by adding CSS rules to `site_base.css`, but since GeoNode is based on Bootstrap, there are many pre-defined themes that you can simply drop into your project to get a whole new look. This is very similar to [WordPress](#) themes and is a powerful and easy way to change the look of your site without much effort.



UWI GEONODE



UWI's GeoNode is setup for students and faculty to collaboratively create and share maps for their class projects. It is maintained by the UWI Geographical Society.

For more information about the UWI Geographical society, [visit our website](#)

Explore Layers

Explore Maps

Bootswatch

[Bootswatch](#) is a site where you can download ready-to-use themes for your GeoNode project site. The following steps will show you how to use a theme from Bootswatch in your own GeoNode site.

1. Visit <http://bootswatch.com> and select a theme (we will use Sandstone for this example). Select the *download bootstrap.css* option in the menu:
2. Put this file into `<my_geonode>/static/css`.

```
$ cd /home/geonode/my_geonode/my_geonode/static/css
```

3. Update the `site_base.html` template to include this file. It should now look like this:

```
$ cd /home/geonode/my_geonode/my_geonode/templates
$ sudo vi site_base.html
```

```
{% extends "base.html" %}
{% block extra_head %}
    <link href="{% STATIC_URL %}css/site_base.css" rel="stylesheet"/>
    <link href="{% STATIC_URL %}css/bootstrap.css" rel="stylesheet"/>
{% endblock %}
```

4. Refresh the development server and visit your site:

```
$ python manage.py collectstatic
$ sudo service apache2 restart
```

Your GeoNode project site is now using the Sandstone theme in addition to the changes you have made.

Bootswatch

THEMES ▾

HELP

BLOG

SANDSTONE ▾

OPEN SANDBOX

BOOTSTRAP.MIN.CSS

BOOTSTRAP.CSS

VARIABLES.LESS

BOOTSWATCH.LESS

_VARIABLES.SCSS

_BOOTSWATCH.SCSS

Sandstone

A touch of warmth

Navbar

Buttons

Typography

Tables

Forms

Navs

WELCOME! - example.com

localhost

Search

☆

📁


📧

⬇

🏠

💬

☰

 THE UNIVERSITY OF THE WEST INDIES
AT ST. AUGUSTINE, TRINIDAD AND TOBAGO

LAYERS

MAPS

DOCUMENTS

PEOPLE


GROUPS

Type your search here ...

SEARCH

SIGN IN

UWI GeoNode



UWI's GeoNode is setup for students and faculty to collaboratively create and share maps for their class projects. It is maintained by the UWI Geographical Society.

For more information about the UWI Geographical society, [visit our website](#)

Setup steps Setup your own geonode project

Theming your GeoNode project Theme your geonode project

3.2.1.4.7 Debugging GeoNode Installations

There are several mechanisms to debug GeoNode installations, the most common ones are discussed in the following sections.

Viewing the logs

There are many kinds of logs in GeoNode, most of them are located in `/var/log/geonode/` and will be explained below in order of relevance:

- **GeoNode main log:** This is the output of the Django application generated by Apache, it may contain detailed information about uploads and high level problems.

The default location is `/var/log/geonode/apache.log` or `/var/log/apache2/error.log`.

It is set to a very low level (not very much information is logged) by default, but it's output can be increased by setting the logging level to `DEBUG` in `/etc/geonode/local_settings.py`.

- **GeoServer log:** It contains most of the information related to problems with data, rendering and styling errors.

This one can be accessed at `GEOSERVER_DATA_DIR/logs/geoserver.log`, which is usually `/var/lib/tomcat7/webapps/geoserver/data/logs/geoserver.log` or `/var/lib/geoserver/geonode-data/logs/geoserver.log`.

It may also be symlinked in `/var/log/geonode/geoserver.log`.

- **Tomcat logs:** Tomcat logs could indicate problems loading GeoServer.

They can be found at `/var/lib/tomcat7/logs/catalina.out` or `/var/lib/tomcat/geoserver/logs/catalina.out`.

- **PostgreSQL logs:** PostgreSQL is accessed by GeoServer and Django, therefore information about errors which are very hard to debug may be found by looking at PostgreSQL's logs.

They are located at `/var/log/postgresql/postgresql-$(psql_version)-main.log` where `$(psql_version)` depends on your local installation.

Enabling DEBUG mode

Django can be set to return nicely formatted exceptions which are useful for debugging instead of generic 500 errors.

This is enabled by setting `DEBUG=True` in `/home/geonode/geonode/geonode/local_settings.py` (or `/etc/geonode/local_settings.py` if GeoNode has been installed using **apt-get**).

After enabling `DEBUG`, the Apache server has to be restarted for the changes to be picked up. In Ubuntu:

```
service apache2 restart
```

Other tips and tricks

Modifying GeoServer's output strategy

Up to version 1.1, GeoNode used by default the `SPEED` output strategy of GeoServer, this meant that proper error messages were being sacrificed for performance. Unfortunately, this caused many errors to be masked as XML parsing errors when layers were not properly configured.

It is recommended to verify the output strategy is set at least to `PARTIAL_BUFFER2` (or a safer one, e.g. `“FILE”`) with a high value for the buffer size. More information about the different strategies and the performance vs correctness trade off is available at GeoServer's `web.xml` file.

The typical location of the file that needs to be modified is `/var/lib/tomcat7/webapps/geoserver/WEB-INF/web.xml` as shown below:

```
<context-param>
  <param-name>serviceStrategy</param-name>
  <param-value>FILE</param-value>
</context-param>
```

Add the Django Debug Toolbar

Warning: The Debug Toolbar module **must** be disabled whe running the server in production (with Apache2 HTTPD Server WSGI)

The Django Debug Toolbar offers a lot of information on about how the page you are seeing is created and used. From the database hits to the views involved. It is a configurable set of panels that display various debug information about the current request/response and when clicked, display more details about the panel's content.

To install it:

```
$ pip install django-debug-toolbar
```

1. Then edit your settings `/home/geonode/geonode/geonode/settings.py` (or `/etc/geonode/settings.py` if GeoNode has been installed using **apt-get**) and add the following to the bottom of the file:

```
#debug_toolbar settings
if DEBUG:
    INTERNAL_IPS = ('127.0.0.1',)
    MIDDLEWARE_CLASSES += (
        'debug_toolbar.middleware.DebugToolbarMiddleware',
    )

    INSTALLED_APPS += (
        'debug_toolbar',
    )

    DEBUG_TOOLBAR_PANELS = [
        'debug_toolbar.panels.versions.VersionsPanel',
        'debug_toolbar.panels.timer.TimerPanel',
        'debug_toolbar.panels.settings.SettingsPanel',
        'debug_toolbar.panels.headers.HeadersPanel',
        'debug_toolbar.panels.request.RequestPanel',
```

(continues on next page)

(continued from previous page)

```

'debug_toolbar.panels.sql.SQLPanel',
'debug_toolbar.panels.staticfiles.StaticFilesPanel',
'debug_toolbar.panels.templates.TemplatesPanel',
'debug_toolbar.panels.cache.CachePanel',
'debug_toolbar.panels.signals.SignalsPanel',
'debug_toolbar.panels.logging.LoggingPanel',
'debug_toolbar.panels.redirects.RedirectsPanel',
]

DEBUG_TOOLBAR_CONFIG = {
    'INTERCEPT_REDIRECTS': False,
}

```

2. Stop Apache and start the server in **Development Mode**:

```

$ service apache2 stop
$ python manage.py runserver

```

3. Redirect the browser to **http://localhost:8000**. You should be able to see the Debug Panel on the right of the screen.

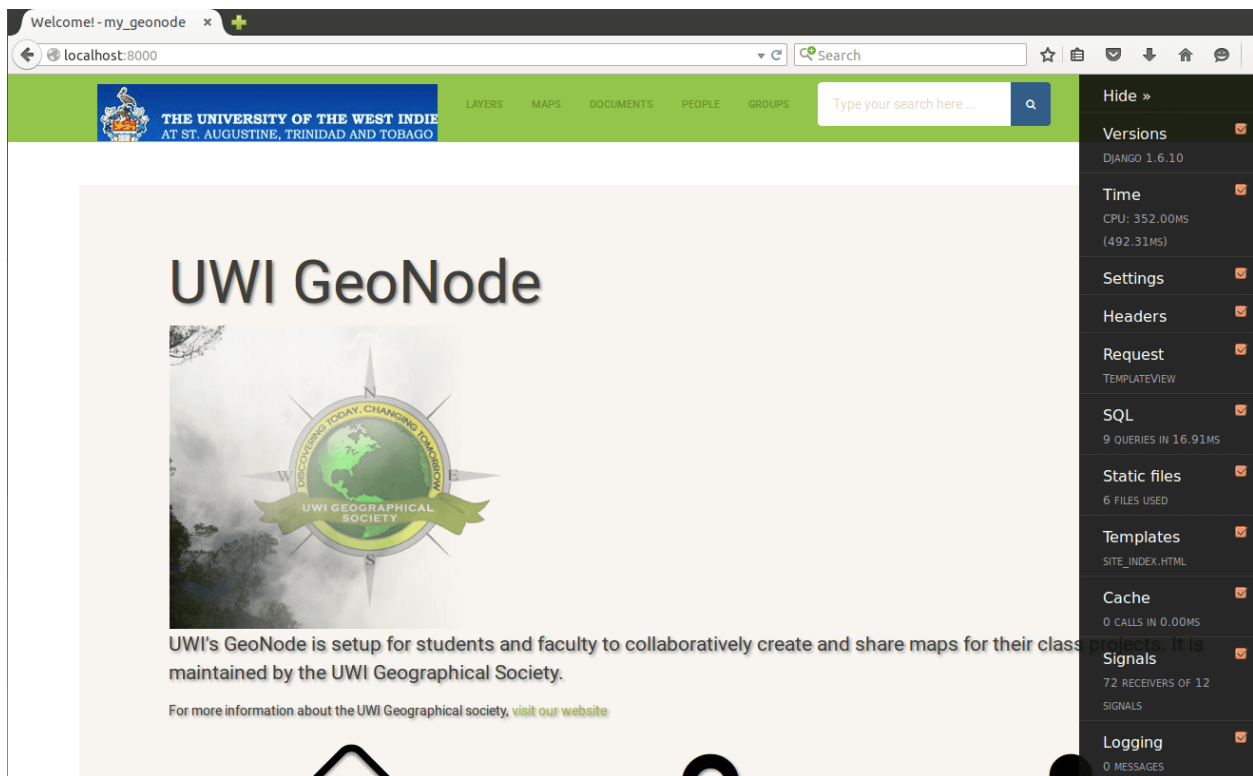


Fig. 82: Django Debug Toolbar Enabled In Devel Mode

More: For more set up and customize the panels read the official docs here

<http://django-debug-toolbar.readthedocs.io/en/latest/>

3.2.1.4.8 Changing the Default Language

GeoNode's default language is English, but GeoNode users can change the interface language with the pulldown menu at the top-right of most GeoNode pages. Once a user selects a language GeoNode remembers that language for subsequent pages.

GeoNode Configuration

As root edit the geonode config file `/home/geonode/geonode/geonode/settings.py` (or `/etc/geonode/settings.py` if GeoNode has been installed using **apt-get**) and change `LANGUAGE_CODE` to the desired default language.

Note: A list of language codes can be found in the global django config file `/usr/local/lib/python2.7/dist-packages/django/conf/global_settings.py` (or `/var/lib/geonode/lib/python2.7/site-packages/django/conf/global_settings.py` if GeoNode has been installed using **apt-get**).

For example, to make French the default language use:

```
LANGUAGE_CODE = 'fr'
```

Unfortunately Django overrides this setting, giving the language setting of a user's browser priority. For example, if `LANGUAGE_CODE` is set to French, but the user has configured their operating system for Spanish they may see the Spanish version when they first visit GeoNode.

Additional Steps

If this is not the desired behaviour, and all users should initially see the default `LANGUAGE_CODE`, regardless of their browser's settings, do the following steps to ensure Django ignores the browser language settings. (Users can always use the pulldown language menu to change the language at any time.)

As **root** create a new directory within GeoNode's site packages:

```
mkdir /usr/lib/python2.7/dist-packages/setmydefaultlanguage
```

or:: `mkdir /var/lib/geonode/lib/python2.7/site-packages/setmydefaultlanguage`

if GeoNode has been installed using **apt-get**.

As root create and edit a new file `/usr/lib/python2.7/dist-packages/setmydefaultlanguage/__init__.py` and add the following lines:

```
class ForceDefaultLanguageMiddleware(object):
    """
    Ignore Accept-Language HTTP headers

    This will force the I18N machinery to always choose settings.LANGUAGE_CODE
    as the default initial language, unless another one is set via sessions or cookies

    Should be installed *before* any middleware that checks request.META['HTTP_ACCEPT_
    →LANGUAGE'],
    namely django.middleware.locale.LocaleMiddleware
    """
    def process_request(self, request):
```

(continues on next page)

(continued from previous page)

```
if request.META.has_key('HTTP_ACCEPT_LANGUAGE') :  
    del request.META['HTTP_ACCEPT_LANGUAGE']
```

At the end of the GeoNode configuration file `/home/geonode/geonode/geonode/settings.py` (or `/etc/geonode/settings.py` if GeoNode has been installed using **apt-get**) add the following lines to ensure the above class is executed:

```
MIDDLEWARE_CLASSES += (  
    'setmydefaultlanguage.ForceDefaultLanguageMiddleware',  
)
```

Restart

Finally restart Apache as root with:

```
service apache2 restart
```

Please refer to Translating GeoNode for information on editing GeoNode pages in different languages and create new GeoNode Translations.

3.2.1.4.9 More on Security and Permissions

Security and Permissions

This tutorial will guide you through the steps that can be done in order to restrict the access on your data uploaded to geonode.

First of all it will be shown how a user can be created and what permissions he can have. Secondly we will take a closer look on to layers, maps and documents and the different opportunities you have in order to ban certain users from viewing or editing your data.

Users

Your first step will be to create a user. There are three options to do so, depending on which kind of user you want to create you may choose a different option. We will start with creating a *superuser*, because this user is the most important. A superuser has all the permissions without explicitly assigning them.

The easiest way to create a superuser (in linux) is to open your terminal and type:

```
$ python manage.py createsuperuser
```

You will be asked a username (in this tutorial we will call the superuser you now create *your_superuser*), an email address and a password.

Now you've created a superuser you should become familiar with the *Django Admin Interface*. As a superuser you are having access to this interface, where you can manage users, layers, permission and more. To learn more detailed about this interface check this [LINK](#). For now it will be enough to just follow the steps. To attend the *Django Admin Interface*, go to your geonode website and *sign in* with *your_superuser*. Once you've logged in, the name of your user will appear on the top right. Click on it and the following menu will show up:

Clicking on *Admin* causes the interface to show up.

Menu



 Upload Layers

 Profile

 Recent Activity

 Inbox

 Announcements

 Remote Services

 Invite User

 GeoServer

 Admin



3.2. Tutorials **Help**

Django administration

Site administration

Account		
Account deletions	+ Add	✎ Change
Accounts	+ Add	✎ Change
Signup codes	+ Add	✎ Change
Actstream		
Actions	+ Add	✎ Change
Follows	+ Add	✎ Change
Announcements		
Announcements	+ Add	✎ Change
Dismissals	+ Add	✎ Change
Auth		
Groups	+ Add	✎ Change
Users	+ Add	✎ Change
Avatar		
Avatars	+ Add	✎ Change
Base		
Contact roles	+ Add	✎ Change
Links	+ Add	✎ Change
Metadata Regions	+ Add	✎ Change

Go to *Auth -> Users* and you will see all the users that exist at the moment. In your case it will only be *your_superuser*. Click on it, and you will see a section on *Personal Info*, one on *Permissions* and one on *Important dates*. For the moment, the section on *Permissions* is the most important.

Permissions

☒ **Active**
Designates whether this user should be treated as active. Unselect this instead of deleting accounts.

☒ **Staff status**
Designates whether the user can log into this admin site.

☒ **Superuser status**
Designates that this user has all permissions without explicitly assigning them.

As you can see, there are three boxes that can be checked and unchecked. Because you've created a superuser, all three boxes are checked as default. If only the box *active* would have been checked, the user would not be a superuser and would not be able to access the *Django Admin Interface* (which is only available for users with the *staff* status). Therefore keep the following two things in mind:

- a superuser is able to access the *Django Admin Interface* and he has all permissions on the data uploaded to GeoNode.
- an ordinary user (created from the GeoNode interface) only has *active* permissions by default. The user will not have the ability to access the *Django Admin Interface* and certain permissions have to be added for him.

Until now we've only created superusers. So how do you create an ordinary user? You have two options:

1. Django Admin Interface

First we will create a user via the *Django Admin Interface* because we've still got it open. Therefore go back to *Auth -> Users* and you should find a button on the right that says *Add user*.

Django administration Welcome, **barbara**. Change password / Log out

Home > Auth > Users

Select user to change Add user +

Click on it and a form to fill out will appear. Name the new user *test_user*, choose a password and click *save* at the right bottom of the site.

Django administration Welcome, **barbara**. Change password / Log out

Home > Auth > Users > Add user

Add user

First, enter a username and password. Then, you'll be able to edit more user options.

Username:
Required. 30 characters or fewer. Letters, digits and @/./+/-/_ only.

Password:

Password confirmation:
Enter the same password as above, for verification.

Save and add another Save and continue editing Save

Now you should be directed to the site where you could change the permissions on the user *test_user*. As default only *active* is checked. If you want this user also to be able to attend this admin interface you could also check *staff status*. But for now we leave the settings as they are!

To test whether the new user was successfully created, go back to the GeoNode web page and try to sign in.

2. GeoNode website

To create an ordinary user you could also just use the GeoNode website. If you installed GeoNode using a release, you should see a *Register* button on the top, beside the *Sign in* button (you might have to log out before).



Hit the button and again a form will appear for you to fill out. This user will be named *geonode_user*

SIGN UP

Username

Password

Password (again)

Email

[Sign up](#)

By hitting *Sign up* the user will be signed up, as default only with the status *active*.

As mentioned before, this status can be changed as well. To do so, sign in with *your_superuser* again and attend the admin interface. Go again to *Auth -> Users*, where now three users should appear:

We now want to change the permission of the *geonode_user* so that he will be able to attend the admin interface as well. Click on to *geonode_user* and you will automatically be moved to the site where you can change the permissions. Check the box *staff status* and hit *save* to store the changes.

To sum it up, we have now created three users with different kind of permissions.

- **your_superuser:** This user is allowed to attend the admin interface and has all available permissions on layers, maps etc.
- **geonode_user:** This user is permitted to attend the admin interface, but permissions on layers, maps etc. have to be assigned.
- **test_user:** This user is not able to attend the admin interface, permissions on layers, maps etc. have also to be assigned.

You should now have an overview over the different kinds of users and how to create and edit them. You've also learned about the permissions a certain user has and how to change them using the *Django Admin Interface*.

Note: If you've installed GeoNode in developing mode, the *Register* button won't be seen from the beginning. To add this button to the website, you have to change the *REGISTRATION_OPEN = False* in the *settings.py* to *REGISTRATION_OPEN = True*. Then reload GeoNode and you should also be able to see the *Register* button.

Layers

Now that we've already created some users, we will take a closer look on the security of layers, how you can protect your data not to be viewed or edited by unwanted users.

Hint: As already mentioned before it is important to know that a superuser does have unrestricted access to all your uploaded data. That means you cannot ban a superuser from viewing, downloading or editing a layer!

The permissions on a certain layer can already be set when uploading your files. When the upload form appears (*Layers -> Upload Layer*) you will see the permission section on the right side:

As it can be seen here, the access on your layer is split up into three groups:

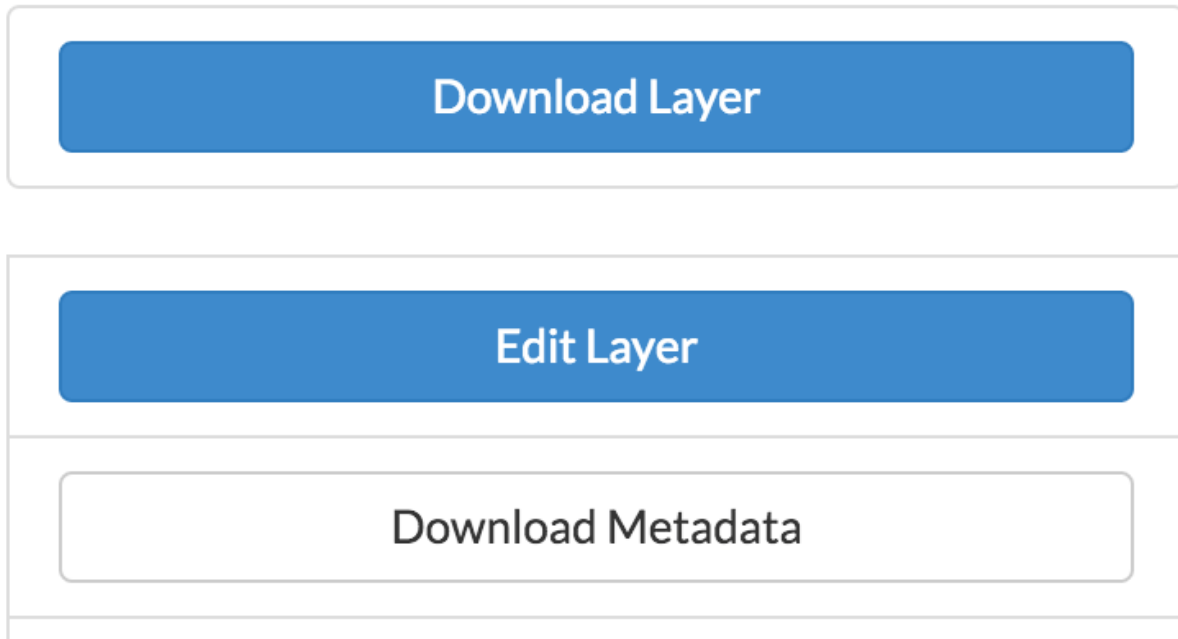
- view and download data
- edit data
- manage and edit data

The difference between *manage and edit layer* and *edit layer* is that a user assigned to *edit layer* is not able to change the permissions on the layer whereas a user assigned to *manage and edit layer* can change the permissions. You can now choose whether you want your layer to be viewed and downloaded by

- anyone
- any registered user
- a certain user (or group)

We will now upload our **test layer** like shown HERE. If you want your layer only be viewed by certain users or a group, you have to choose *Only users who can edit* in the part *Who can view and download this data*. In the section *Who can edit this data* you write down the names of the users you want to have admission on this data. For this first layer we will choose the settings like shown in the following image:

If you now log out, your layer can still be seen, but the unregistered users won't be able to edit your layer. Now sign in as *geonode_user* and click on the **test layer**. Above the layer you can see this:



The *geonode_user* is able to edit the **test_layer**. But before going deeper into this, we have to first take a look on another case. As an administrator you might also upload your layers to GeoServer and then make them available on GeoNode using *updatelayers*. Or you even add the layers via the terminal using *importlayers* (LINK TUTORIAL). To set the permissions on this layer, click on the **test layer** (you've uploaded via *updatelayers*) and you will see the same menu as shown in the image above. Click *Edit layer* and the menu will appear.

Choose *edit permissions* and a window with the permission settings will appear. This window can also be opened by scrolling down the website. On the right-hand side of the page you should be able to see a button like this.

Click on it and you will see the same window as before.

Now set the permissions of this layer using the following settings:

When you assign a user to be able to edit your data, this user is allowed to execute all of the following actions:

- edit metadata
- edit styles
- manage styles
- replace layer
- remove layer

So be aware that each user assigned to edit this layer can even remove it! In our case, only the user *test_user* and *your_superuser* do have the rights to do so. *Geonode_user* is neither able to view nor to download or edit this layer.

Now you are logged in as the user *test_user*. Below the **test_layer** you can see the following:

By clicking *Edit Layer* and *Edit Metadata* on top of the layer, you can change this information. The *test_user* is able to change all the metadata of this layer. We now want to change to *point of contact*, therefore scroll down until you see this:

Change the *point of contact* from *_who_ever_created_this* to *test_user*. Save your changes and you will now be able to see the following:

Edit Layer



Metadata

Edit



Styles

Edit

Manage



Thumbnail

Set



Layer

Replace

Edit data

Remove

Close

Point Of Contact

bobby (None)



Metadata Author

bobby (None)

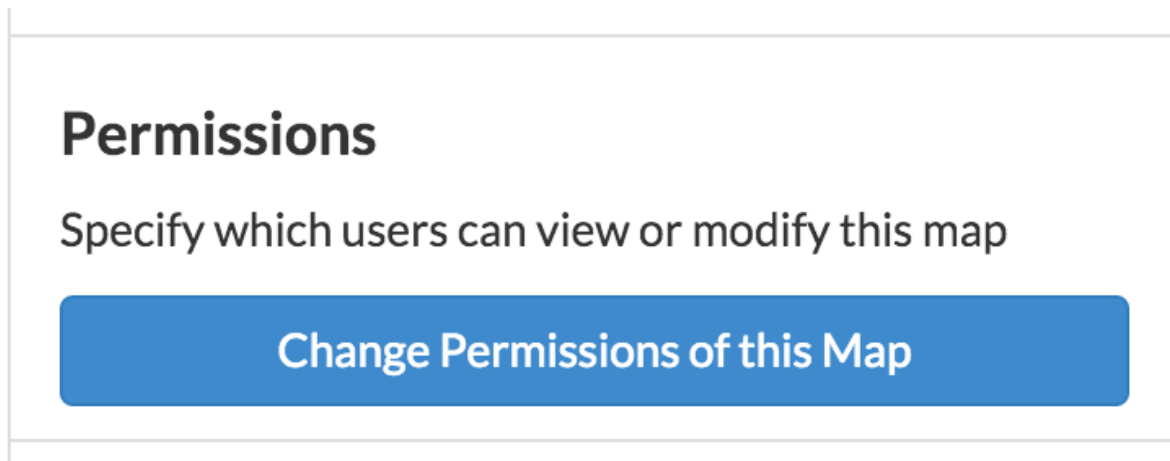


Warning: If you allow a user to view and download a layer, this user will also be able to edit the styles, even if he is not assigned to edit the layer! Keep this in mind!

To learn how you can edit metadata or change the styles go to this section [LINK](#).

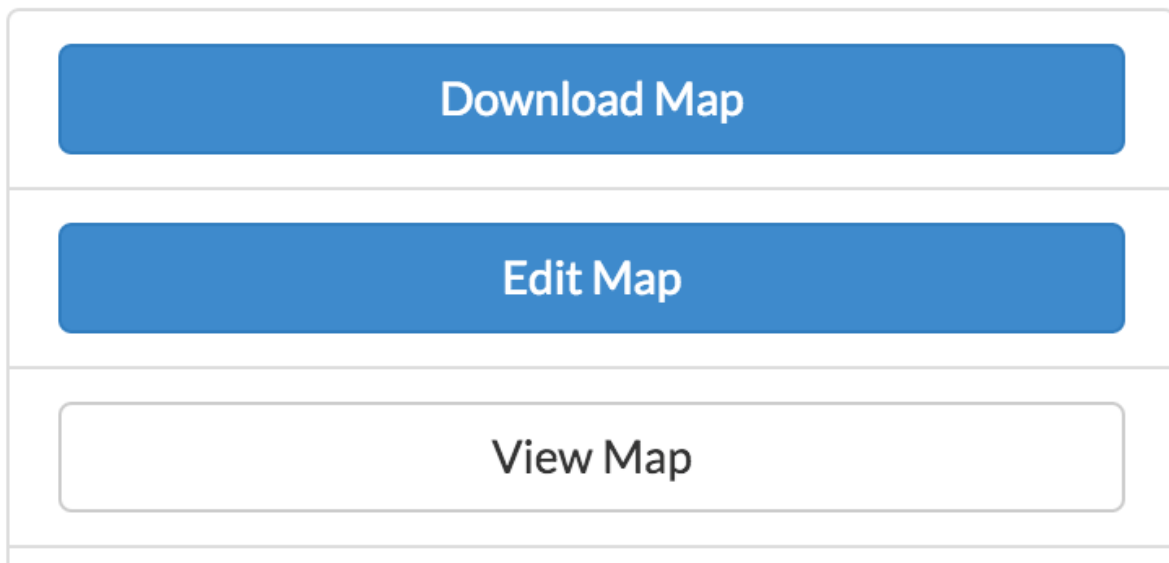
Maps

The permission on maps are basically the same as on layers, just that there are fewer options on how to edit the map. Let's create a map (or already TUTORIAL?). Click on **test_map** and scroll down till you see this:



Here you can set the same permissions as known from the layer permissions! Set the permissions of this map as seen here:

Save your changes and then log out and log in as *test_user*. You should now be able to view the *test_map* and click on to *Edit map*.



As you may recognize, this user is not able to change the permissions on this map. If you log in as the user *geonode_user* you should be able to see the button *change map permissions* when you scroll down the page.

Documents

All the same is also valid for your uploaded documents.

3.2.1.4.10 Loading Data into a GeoNode

This module will walk you through the various options available to load data into your GeoNode from GeoServer, on the command-line or programmatically. You can choose from among these techniques depending on what kind of data you have and how you have your geonode setup.

Warning: Some parts of this section have been taken from the [GeoServer](#) project and training documentation.

Using importlayers to import Data into GeoNode

The geonode.layers app includes 2 management commands that you can use to load or configure data in your GeoNode. Both of these are invoked by using the manage.py script. This section will walk you through how to use the importlayers management command and the subsequent section will lead you through the process of using updatelayers.

The first thing to do is to use the `--help` option to the importlayers command to investigate the options to this management command. You can display this help by executing the following command:

```
$ python manage.py importlayers --help
```

This will produce output that looks like the following:

```
Usage: manage.py importlayers [options] path [path...]

Brings a data file or a directory full of data files into aGeoNode site. Layers are
↳added to the Django database, theGeoServer configuration, and the GeoNetwork_
↳metadata index.

Options:
  --version                show program's version number and exit
  -h, --help              show this help message and exit
  -v VERBOSITY, --verbosity=VERBOSITY
                          Verbosity level; 0=minimal output, 1=normal output,
                          2=verbose output, 3=very verbose output
  --settings=SETTINGS     The Python path to a settings module, e.g.
                          "myproject.settings.main". If this isn't provided, the
                          DJANGO_SETTINGS_MODULE environment variable will be
                          used.
  --pythonpath=PYTHONPATH
                          A directory to add to the Python path, e.g.
                          "/home/djangoprojects/myproject".
  --traceback             Print traceback on exception
  --no-color              Don't colorize the command output.
  -u USER, --user=USER   Name of the user account which should own the imported
                          layers
  -i, --ignore-errors     Stop after any errors are encountered.
  -o, --overwrite         Overwrite existing layers if discovered (defaults
                          False)
  -k KEYWORDS, --keywords=KEYWORDS
```

(continues on next page)

(continued from previous page)

```

        The default keywords for the imported layer(s). Will
        be the same for all imported layers if multiple
        imports are done in one command
-l LICENSE, --license=LICENSE
        The license for the imported layer(s). Will be the
        same for all imported layers if multiple imports are
        done in one command
-c CATEGORY, --category=CATEGORY
        The category for the imported layer(s). Will be the
        same for all imported layers if multiple imports are
        done in one command
-r REGIONS, --regions=REGIONS
        The default regions, separated by comma, for the
        imported layer(s). Will be the same for all imported
        layers if multiple imports are done in one command
-n LAYERNAME, --name=LAYERNAME
        The name for the imported layer(s). Can not be used
        with multiple imports
-t TITLE, --title=TITLE
        The title for the imported layer(s). Will be the same
        for all imported layers if multiple imports are done
        in one command
-a ABSTRACT, --abstract=ABSTRACT
        The abstract for the imported layer(s). Will be the
        same forall imported layers if multiple imports are
        donein one command
-d DATE, --date=DATE The date and time for the imported layer(s). Will be
        the same for all imported layers if multiple imports
        are done in one command. Use quotes to specify both
        the date and time in the format 'YYYY-MM-DD HH:MM:SS'.
-p, --private Make layer viewable only to owner
-m, --metadata_uploaded_preserve
        Force metadata XML to be preserved
-C CHARSET, --charset=CHARSET
        Specify the charset of the data

```

While the description of most of the options should be self explanatory, its worth reviewing some of the key options in a bit more detail.

- The **-i** option will force the command to stop when it first encounters an error. Without this option specified, the process will skip over errors that have layers and continue loading the other layers.
- The **-o** option specifies that layers with the same name as the base name will be loaded and overwrite the existing layer.
- The **-u** option specifies which will be the user that owns the imported layers. The same user will be the point of contact and the metadata author as well for that layer
- The **-k** option is used to add keywords for all of the layers imported.
- The **-C** option specifies the character encoding of the data.

The import layers management command is invoked by specifying options as described above and specifying the path to a single layer file or to a directory that contains multiple files. For purposes of this exercise, lets use the default set of testing layers that ship with geonode. You can replace this path with the directory to your own shapefiles:

```

$ python manage.py importlayers -v 3 /var/lib/geonode/lib/python2.7/site-packages/
↪gisdata/data/good/vector/

```


This command will produce the following output to your terminal:

```
Verifying that GeoNode is running ...
Found 8 potential layers.
No handlers could be found for logger "pycsw"
[created] Layer for '/Users/jjohnson/.venvs/geonode/lib/python2.7/site-packages/
↳gisdata/data/good/vector/san_andres_y_providencia_administrative.shp' (1/8)
[created] Layer for '/Users/jjohnson/.venvs/geonode/lib/python2.7/site-packages/
↳gisdata/data/good/vector/san_andres_y_providencia_coastline.shp' (2/8)
[created] Layer for '/Users/jjohnson/.venvs/geonode/lib/python2.7/site-packages/
↳gisdata/data/good/vector/san_andres_y_providencia_highway.shp' (3/8)
[created] Layer for '/Users/jjohnson/.venvs/geonode/lib/python2.7/site-packages/
↳gisdata/data/good/vector/san_andres_y_providencia_location.shp' (4/8)
[created] Layer for '/Users/jjohnson/.venvs/geonode/lib/python2.7/site-packages/
↳gisdata/data/good/vector/san_andres_y_providencia_natural.shp' (5/8)
[created] Layer for '/Users/jjohnson/.venvs/geonode/lib/python2.7/site-packages/
↳gisdata/data/good/vector/san_andres_y_providencia_poi.shp' (6/8)
[created] Layer for '/Users/jjohnson/.venvs/geonode/lib/python2.7/site-packages/
↳gisdata/data/good/vector/san_andres_y_providencia_water.shp' (7/8)
[created] Layer for '/Users/jjohnson/.venvs/geonode/lib/python2.7/site-packages/
↳gisdata/data/good/vector/single_point.shp' (8/8)

Detailed report of failures:

Finished processing 8 layers in 30.0 seconds.

8 Created layers
0 Updated layers
0 Skipped layers
0 Failed layers
3.750000 seconds per layer
```

If you encounter errors while running this command, you can use the `-v` option to increase the verbosity of the output so you can debug the problem. The verbosity level can be set from 0-3 with 0 being the default. An example of what the output looks like when an error is encountered and the verbosity is set to 3 is shown below:

```
Verifying that GeoNode is running ...
Found 8 potential layers.
[failed] Layer for '/Users/jjohnson/.venvs/geonode/lib/python2.7/site-packages/
↳gisdata/data/good/vector/san_andres_y_providencia_administrative.shp' (1/8)
[failed] Layer for '/Users/jjohnson/.venvs/geonode/lib/python2.7/site-packages/
↳gisdata/data/good/vector/san_andres_y_providencia_coastline.shp' (2/8)
[failed] Layer for '/Users/jjohnson/.venvs/geonode/lib/python2.7/site-packages/
↳gisdata/data/good/vector/san_andres_y_providencia_highway.shp' (3/8)
[failed] Layer for '/Users/jjohnson/.venvs/geonode/lib/python2.7/site-packages/
↳gisdata/data/good/vector/san_andres_y_providencia_location.shp' (4/8)
[failed] Layer for '/Users/jjohnson/.venvs/geonode/lib/python2.7/site-packages/
↳gisdata/data/good/vector/san_andres_y_providencia_natural.shp' (5/8)
[failed] Layer for '/Users/jjohnson/.venvs/geonode/lib/python2.7/site-packages/
↳gisdata/data/good/vector/san_andres_y_providencia_poi.shp' (6/8)
[failed] Layer for '/Users/jjohnson/.venvs/geonode/lib/python2.7/site-packages/
↳gisdata/data/good/vector/san_andres_y_providencia_water.shp' (7/8)
[failed] Layer for '/Users/jjohnson/.venvs/geonode/lib/python2.7/site-packages/
↳gisdata/data/good/vector/single_point.shp' (8/8)

Detailed report of failures:
```

(continues on next page)

(continued from previous page)

```
/Users/jjohnson/.venvs/geonode/lib/python2.7/site-packages/gisdata/data/good/vector/  
↳san_andres_y_providencia_administrative.shp  
=====
```

```
Traceback (most recent call last):  
  File "/Users/jjohnson/projects/geonode/geonode/layers/utils.py", line 682, in upload  
    keywords=keywords,  
  File "/Users/jjohnson/projects/geonode/geonode/layers/utils.py", line 602, in file_  
↳upload  
    keywords=keywords, title=title)  
  File "/Users/jjohnson/projects/geonode/geonode/layers/utils.py", line 305, in save  
    store = cat.get_store(name)  
  File "/Users/jjohnson/.venvs/geonode/lib/python2.7/site-packages/geoserver/catalog.  
↳py", line 176, in get_store  
    for ws in self.get_workspaces():  
  File "/Users/jjohnson/.venvs/geonode/lib/python2.7/site-packages/geoserver/catalog.  
↳py", line 489, in get_workspaces  
    description = self.get_xml("%s/workspaces.xml" % self.service_url)  
  File "/Users/jjohnson/.venvs/geonode/lib/python2.7/site-packages/geoserver/catalog.  
↳py", line 136, in get_xml  
    response, content = self.http.request(rest_url)  
  File "/Library/Python/2.7/site-packages/httplib2/__init__.py", line 1445, in request  
    (response, content) = self._request(conn, authority, uri, request_uri, method, _  
↳body, headers, redirections, cachekey)  
  File "/Library/Python/2.7/site-packages/httplib2/__init__.py", line 1197, in _  
↳request  
    (response, content) = self._conn_request(conn, request_uri, method, body, headers)  
  File "/Library/Python/2.7/site-packages/httplib2/__init__.py", line 1133, in _conn_  
↳request  
    conn.connect()  
  File "/Library/Python/2.7/site-packages/httplib2/__init__.py", line 799, in connect  
    raise socket.error, msg  
error: [Errno 61] Connection refused
```

Note: This last section of output will be repeated for all layers, and only the first one is show above.

This error indicates that GeoNode was unable to connect to GeoServer to load the layers. To solve this, you should make sure GeoServer is running and re-run the command.

If you encounter errors with this command that you cannot solve, you should bring them up on the geonode users mailing list.

You should now have the knowledge necessary to import layers into your GeoNode project from a directory on the servers filesystem and can use this to load many layers into your GeoNode at once.

Note: If you do not use the -u command option, the ownership of the imported layers will be assigned to the primary superuser in your system. You can use GeoNodes Django Admin interface to modify this after the fact if you want them to be owned by another user.

GeoServer Data Configuration

While it is possible to import layers directly from your servers filesystem into your GeoNode, you may have an existing GeoServer that already has data in it, or you may want to configure data from a GeoServer which is not directly supported by uploading data.

GeoServer supports a wide range of data formats and connections to database, and while many of them are not supported as GeoNode upload formats, if they can be configured in GeoServer, you can add them to your GeoNode by following the procedure described below.

GeoServer supports 3 types of data: Raster, Vector and Databases. For a list of the supported formats for each type of data, consult the following pages:

- <http://docs.geoserver.org/latest/en/user/data/vector/index.html#data-vector>
- <http://docs.geoserver.org/latest/en/user/data/raster/index.html>
- <http://docs.geoserver.org/latest/en/user/data/database/index.html>

Note: Some of these raster or vector formats or database types require that you install specific plugins in your GeoServer in order to use the. Please consult the GeoServer documentation for more information.

Lets walk through an example of configuring a new PostGIS database in GeoServer and then configuring those layers in your GeoNode.

First visit the GeoServer administration interface on your server. This is usually on port 8080 and is available at <http://localhost:8080/geoserver/web/>

You should login with the superuser credentials you setup when you first configured your GeoNode instance.

Once you are logged in to the GeoServer Admin interface, you should see the following.

The screenshot shows the GeoServer administration interface. At the top, the GeoServer logo is on the left, and the user is logged in as 'admin' with a 'Logout' button on the right. The main content area is titled 'Welcome' and shows the following information:

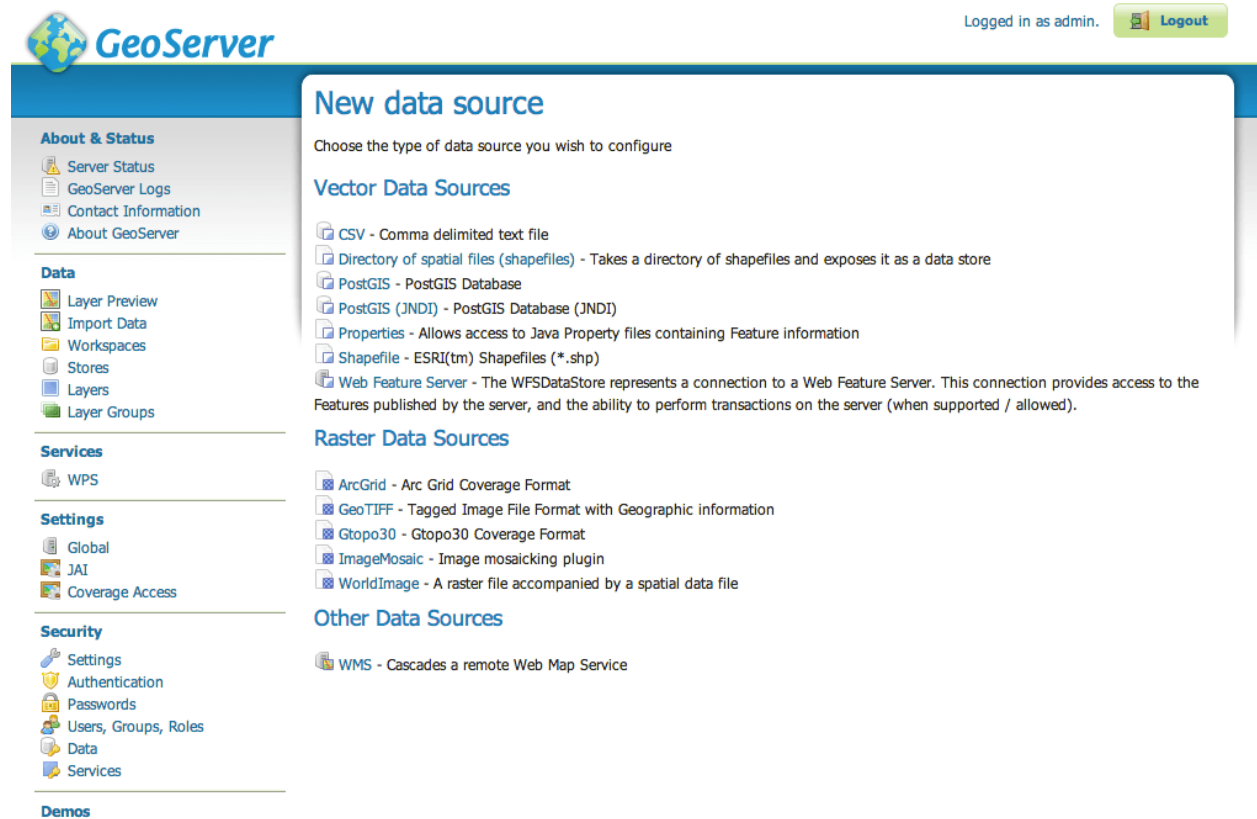
- Server Status:** 10 Layers, 10 Stores, 3 Workspaces. Each item has a corresponding '+ Add' button.
- Service Capabilities:**
 - WCS: 1.0.0, 1.1.1
 - WFS: 1.0.0, 1.1.0, 2.0.0
 - WMS: 1.1.1, 1.3.0
 - WPS: 1.0.0
- Security Notices:**
 - Please read the file /Users/jjohnson/projects/geonode/geoserver/data/security/masterpw.info and remove it afterwards. This file is a **security risk**.
 - The default user/group service should use digest password encoding.
 - The administrator password for this server has not been changed from the default. It is **highly** recommended that you change it now. [Change it](#)
 - Strong cryptography available
- Version:** This GeoServer instance is running version **2.2**. For more information please contact the administrator.

The left sidebar contains a navigation menu with the following sections:

- About & Status:** Server Status, GeoServer Logs, Contact Information, About GeoServer
- Data:** Layer Preview, Import Data, Workspaces, Stores, Layers, Layer Groups
- Services:** WPS
- Settings:** Global, JAI, Coverage Access
- Security:** Settings, Authentication, Passwords, Users, Groups, Roles, Data, Services
- Demos:**

Note: The number of stores, layers and workspaces may be different depending on what you already have configured in your GeoServer.

Next you want to select the “Stores” option in the left hand menu, and then the “Add new Store” option. The following screen will be displayed.



In this case, we want to select the PostGIS store type to create a connection to our existing database. On the next screen you will need to enter the parameters to connect to your PostGIS database (alter as necessary for your own database).

Note: If you are unsure about any of the settings, leave them as the default.


The next screen lets you configure the layers in your database. This will of course be different depending on the layers in your database.

Select the “Publish” button for one of the layers and the next screen will be displayed where you can enter metadata for this layer. Since we will be managing this metadata in GeoNode, we can leave these alone for now.

The things that *must* be specified are the Declared SRS and you must select the “Compute from Data” and “Compute from native bounds” links after the SRS is specified.

Click save and this layer will now be configured for use in your GeoServer.

The next step is to configure these layers in GeoNode. The `updatelayers` management command is used for this purpose. As with `importlayers`, its useful to look at the command line options for this command by passing the `-help` option:


GeoServer

Logged in as admin. [Logout](#)

New Vector Data Source

Add a new vector data source

PostGIS
PostGIS Database

Basic Store Info

Workspace *
geonode

Data Source Name *
workshop_postgis

Description

☒ Enabled

Connection Parameters

host *
localhost

port *
5432

database
workshop

schema
public

user *
workshop

passwd

About & Status

- Server Status
- GeoServer Logs
- Contact Information
- About GeoServer

Data

- Layer Preview
- Import Data
- Workspaces
- Stores
- Layers
- Layer Groups

Services

- WPS


Settings

- Global
- JAI
- Coverage Access

Security

- Settings
- Authentication
- Passwords
- Users, Groups, Roles
- Data
- Services

Demos


GeoServer

Logged in as admin. [Logout](#)

New Layer

Add a new layer

You can create a new feature type by manually configuring the attribute names and types. [Create new feature type...](#)
On databases you can also create a new feature type by configuring a native SQL statement. [Configure new SQL view...](#)
Here is a list of resources contained in the store 'workshop'. Click on the layer you wish to configure

[<<](#)
[<](#)
[1](#)
[>](#)
[>>](#)
 Results 1 to 5 (out of 5 items)

Published	Layer name	action
	boundaries_l	Publish
	ontdrainage	Publish
	popplaces	Publish
	qcdrainage	Publish
	roads	Publish

[<<](#)
[<](#)
[1](#)
[>](#)
[>>](#)
 Results 1 to 5 (out of 5 items)

About & Status

- Server Status
- GeoServer Logs
- Contact Information
- About GeoServer

Data

- Layer Preview
- Import Data
- Workspaces
- Stores
- Layers
- Layer Groups

Services

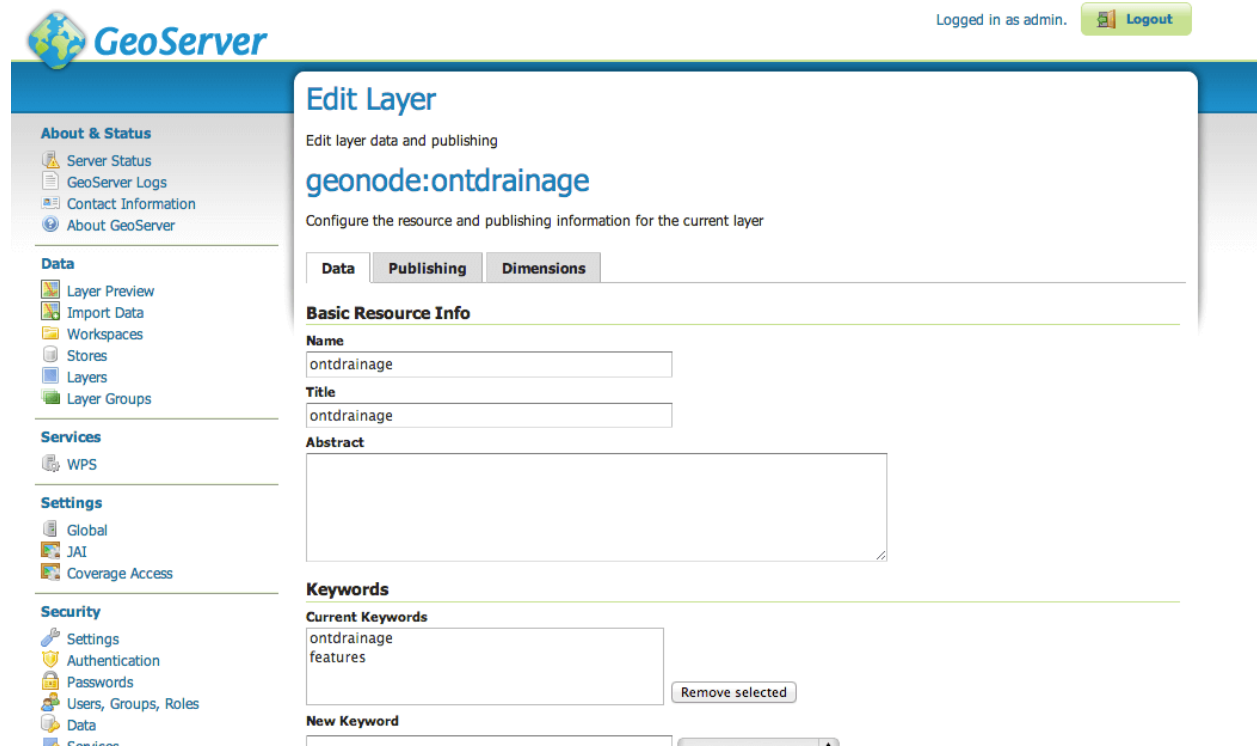
- WPS

Settings

- Global
- JAI
- Coverage Access

Security

- Settings
- Authentication
- Passwords



The screenshot shows the GeoServer web interface. On the left is a sidebar with a tree view containing categories: About & Status (Server Status, GeoServer Logs, Contact Information, About GeoServer), Data (Layer Preview, Import Data, Workspaces, Stores, Layers, Layer Groups), Services (WPS), Settings (Global, JAI, Coverage Access), and Security (Settings, Authentication, Passwords, Users, Groups, Roles, Data, Feeds). The main panel is titled 'Edit Layer' and has a subtitle 'Edit layer data and publishing'. Below this is the layer name 'geonode:ontdrainage' and a description 'Configure the resource and publishing information for the current layer'. There are three tabs: 'Data' (selected), 'Publishing', and 'Dimensions'. Under the 'Data' tab, there is a section 'Basic Resource Info' with input fields for 'Name' (ontdrainage), 'Title' (ontdrainage), and a large text area for 'Abstract'. Below this is a 'Keywords' section with a list of 'Current Keywords' (ontdrainage, features) and a 'Remove selected' button. At the bottom, there is a 'New Keyword' input field.

```
$ python manage.py updatelayers --help
```

This help option displays the following:

```
Usage: manage.py updatelayers [options]

Update the GeoNode application with data from GeoServer

Options:
  -v VERBOSITY, --verbosity=VERBOSITY
                                Verbosity level; 0=minimal output, 1=normal output,
                                2=verbose output, 3=very verbose output
  --settings=SETTINGS           The Python path to a settings module, e.g.
                                "myproject.settings.main". If this isn't provided, the
                                DJANGO_SETTINGS_MODULE environment variable will be
                                used.
  --pythonpath=PYTHONPATH       A directory to add to the Python path, e.g.
                                "/home/djangoprojects/myproject".
  --traceback                   Print traceback on exception
  -i, --ignore-errors           Stop after any errors are encountered.
  -u USER, --user=USER         Name of the user account which should own the imported
                                layers
  -w WORKSPACE, --workspace=WORKSPACE
                                Only update data on specified workspace
  --version                     show program's version number and exit
  -h, --help                   show this help message and exit
```

For this sample, we can use the default options. So enter the following command to configure the layers from our GeoServer into our GeoNode:

Add Keyword

Metadata links

No metadata links so far

Add link *Note only FGDC and TC211 metadata links show up in WMS 1.1.1 capabilities*

Coordinate Reference Systems

Native SRS

...

Declared SRS

EPSG:3978 [Find...](#) [EPSG:NAD83 / Canada Atlas Lambert...](#)

SRS handling

Force declared

Bounding Boxes

Native Bounding Box

Min X	Min Y	Max X	Max Y
1,362,796.5	-323,704.125	1,751,462.75	63,159.9765625

[Compute from data](#)

Lat/Lon Bounding Box

Min X	Min Y	Max X	Max Y
-78.0029900468	43.46292555508	-71.8184077161	47.79032394627

[Compute from native bounds](#)

Feature Type Details

Property	Type	Nullable	Min/Max Occurrences
gid	Integer	false	1/1
area	BigDecimal	true	0/1
perimeter	BigDecimal	true	0/1
oncart_	BigDecimal	true	0/1

GeoServer

Logged in as admin. [Logout](#)

About & Status

- Server Status
- GeoServer Logs
- Contact Information
- About GeoServer

Data

- Layer Preview
- Import Data
- Workspaces
- Stores
- Layers
- Layer Groups

Services

- WPS

Settings

- Global
- JAI
- Coverage Access

Security

- Settings
- Authentication
- Passwords
- Users, Groups, Roles
- Data

Layers

Manage the layers being published by GeoServer

[Add a new resource](#)
[Remove selected resources](#)

Results 1 to 11 (out of 11 items)

<input type="checkbox"/>	Type	Workspace	Store	Layer Name	Enabled?	Native SRS
<input type="checkbox"/>	geonode	Adult_Day_Care	Adult_Day_Care	Adult_Day_Care	✓	EPSG:2230
<input type="checkbox"/>	geonode	casinos	casinos	casinos	✓	EPSG:2230
<input type="checkbox"/>	geonode	san_andres_y_providencia_administrative	san_andres_y_providencia_administrative	san_andres_y_providencia_administrative	✓	EPSG:4326
<input type="checkbox"/>	geonode	san_andres_y_providencia_coastline	san_andres_y_providencia_coastline	san_andres_y_providencia_coastline	✓	EPSG:4326
<input type="checkbox"/>	geonode	san_andres_y_providencia_highway	san_andres_y_providencia_highway	san_andres_y_providencia_highway	✓	EPSG:4326
<input type="checkbox"/>	geonode	san_andres_y_providencia_location	san_andres_y_providencia_location	san_andres_y_providencia_location	✓	EPSG:4326
<input type="checkbox"/>	geonode	san_andres_y_providencia_natural	san_andres_y_providencia_natural	san_andres_y_providencia_natural	✓	EPSG:4326
<input type="checkbox"/>	geonode	san_andres_y_providencia_poi	san_andres_y_providencia_poi	san_andres_y_providencia_poi	✓	EPSG:4326
<input type="checkbox"/>	geonode	san_andres_y_providencia_water	san_andres_y_providencia_water	san_andres_y_providencia_water	✓	EPSG:4326
<input type="checkbox"/>	geonode	single_point	single_point	single_point	✓	EPSG:4326
<input type="checkbox"/>	geonode	workshop	ontdrainage	ontdrainage	✓	EPSG:3978

Results 1 to 11 (out of 11 items)

```
$ python manage.py updatelayers
```

The output will look something like the following:

```
[created] Layer Adult_Day_Care (1/11)
[created] Layer casinos (2/11)
[updated] Layer san_andres_y_providencia_administrative (3/11)
[updated] Layer san_andres_y_providencia_coastline (4/11)
[updated] Layer san_andres_y_providencia_highway (5/11)
[updated] Layer san_andres_y_providencia_location (6/11)
[updated] Layer san_andres_y_providencia_natural (7/11)
[updated] Layer san_andres_y_providencia_poi (8/11)
[updated] Layer san_andres_y_providencia_water (9/11)
[updated] Layer single_point (10/11)
[created] Layer ontdrainage (11/11)
```

Finished processing 11 layers in 45.0 seconds.

```
3 Created layers
8 Updated layers
0 Failed layers
4.090909 seconds per layer
```

Note: This example picked up 2 additional layers that were already in our GeoServer, but were not already in our GeoNode.

For layers that already exist in your GeoNode, they will be updated and the configuration synchronized between GeoServer and GeoNode.

You can now view and use these layers in your GeoNode.

Using GDAL and OGR to convert your Data for use in GeoNode

GeoNode supports uploading data in shapefiles, GeoTIFF, CSV and KML formats (for the last two formats only if you are using the `geonode.importer` backend in the `UPLOAD` variable in `settings.py`). If your data is in other formats, you will need to convert it into one of these formats for use in GeoNode. This section will walk you through the steps necessary to convert your data into formats suitable for uploading into GeoNode.

You will need to make sure that you have the GDAL library installed on your system. On Ubuntu you can install this package with the following command:

```
$ sudo apt-get install gdal-bin
```

OGR (Vector Data)

OGR is used to manipulate vector data. In this example, we will use MapInfo .tab files and convert them to shapefiles with the `ogr2ogr` command. We will use sample MapInfo files from the website linked below.

<http://services.land.vic.gov.au/landchannel/content/help?name=sampleddata>

You can download the Admin;(Postcode) layer by issuing the following command:

```
$ wget http://services.land.vic.gov.au/sampleddata/mif/admin_postcode_vm.zip
```

You will need to unzip this dataset by issuing the following command:

```
$ unzip admin_postcode_vm.zip
```

This will leave you with the following files in the directory where you executed the above commands:

```
|-- ANZVI0803003025.htm
|-- DSE_Data_Access_Licence.pdf
|-- VMADMIN.POSTCODE_POLYGON.xml
|-- admin_postcode_vm.zip
--- vicgrid94
    --- mif
        --- lga_polygon
            --- macedon\ ranges
                |-- EXTRACT_POLYGON.mid
                |-- EXTRACT_POLYGON.mif
            --- VMADMIN
                |-- POSTCODE_POLYGON.mid
                --- POSTCODE_POLYGON.mif
```

First, lets inspect this file set using the following command:

```
$ ogrinfo -so vicgrid94/mif/lga_polygon/macedon\ ranges/VMADMIN/POSTCODE_POLYGON.mid_
↳ POSTCODE_POLYGON
```

The output will look like the following:

```
Had to open data source read-only.
INFO: Open of `vicgrid94/mif/lga_polygon/macedon ranges/VMADMIN/POSTCODE_POLYGON.mid'
      using driver `MapInfo File' successful.
```

(continues on next page)

(continued from previous page)

```

Layer name: POSTCODE_POLYGON
Geometry: 3D Unknown (any)
Feature Count: 26
Extent: (2413931.249367, 2400162.366186) - (2508952.174431, 2512183.046927)
Layer SRS WKT:
PROJCS["unnamed",
  GEOGCS["unnamed",
    DATUM["GDA94",
      SPHEROID["GRS 80",6378137,298.257222101],
      TOWGS84[0,0,0,-0,-0,-0,0]],
    PRIMEM["Greenwich",0],
    UNIT["degree",0.0174532925199433]],
  PROJECTION["Lambert_Conformal_Conic_2SP"],
  PARAMETER["standard_parallel_1",-36],
  PARAMETER["standard_parallel_2",-38],
  PARAMETER["latitude_of_origin",-37],
  PARAMETER["central_meridian",145],
  PARAMETER["false_easting",2500000],
  PARAMETER["false_northing",2500000],
  UNIT["Meter",1]]
PFI: String (10.0)
POSTCODE: String (4.0)
FEATURE_TYPE: String (6.0)
FEATURE_QUALITY_ID: String (20.0)
PFI_CREATED: Date (10.0)
UFI: Real (12.0)
UFI_CREATED: Date (10.0)
UFI_OLD: Real (12.0)

```

This gives you information about the number of features, the extent, the projection and the attributes of this layer.

Next, lets go ahead and convert this layer into a shapefile by issuing the following command:

```
$ ogr2ogr -t_srs EPSG:4326 postcode_polygon.shp vicgrid94/mif/lga_polygon/macedon\
ranges/VMADMIN/POSTCODE_POLYGON.mid POSTCODE_POLYGON
```

Note that we have also reprojected the layer to the WGS84 spatial reference system with the `-t_srs` `ogr2ogr` option.

The output of this command will look like the following:

```

Warning 6: Normalized/laundered field name: 'FEATURE_TYPE' to 'FEATURE_TY'
Warning 6: Normalized/laundered field name: 'FEATURE_QUALITY_ID' to 'FEATURE_QU'
Warning 6: Normalized/laundered field name: 'PFI_CREATED' to 'PFI_CREATE'
Warning 6: Normalized/laundered field name: 'UFI_CREATED' to 'UFI_CREATE'

```

This output indicates that some of the field names were truncated to fit into the constraint that attributes in shapefiles are only 10 characters long.

You will now have a set of files that make up the `postcode_polygon.shp` shapefile set. We can inspect them by issuing the following command:

```
$ ogrinfo -so postcode_polygon.shp postcode_polygon
```

The output will look similar to the output we saw above when we inspected the MapInfo file we converted from:

```

INFO: Open of `postcode_polygon.shp'
      using driver `ESRI Shapefile' successful.

Layer name: postcode_polygon
Geometry: Polygon
Feature Count: 26
Extent: (144.030296, -37.898156) - (145.101137, -36.888878)
Layer SRS WKT:
GEOGCS["GCS_WGS_1984",
  DATUM["WGS_1984",
    SPHEROID["WGS_84",6378137,298.257223563]],
  PRIMEM["Greenwich",0],
  UNIT["Degree",0.017453292519943295]]
PFI: String (10.0)
POSTCODE: String (4.0)
FEATURE_TY: String (6.0)
FEATURE_QU: String (20.0)
PFI_CREATE: Date (10.0)
UFI: Real (12.0)
UFI_CREATE: Date (10.0)
UFI_OLD: Real (12.0)

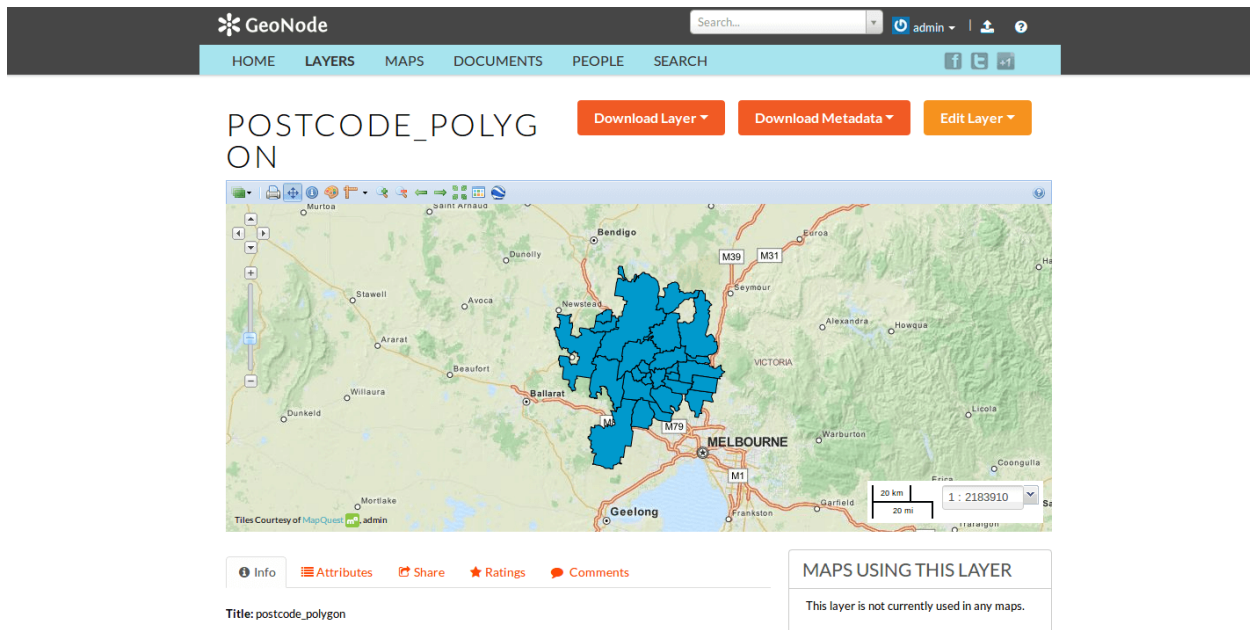
```

These files can now be loaded into your GeoNode instance via the normal uploader.

Visit the upload page in your GeoNode, drag and drop the files that composes the shapefile that you have generated using the GDAL ogr2ogr command (postcode_polygon.dbf, postcode_polygon.prj, postcode_polygon.shp, postcode_polygon.shx). Give the permissions as needed and then click the “Upload files” button.

The screenshot displays the GeoNode 'UPLOAD LAYERS' page. At the top, there's a navigation bar with 'HOME', 'LAYERS', 'MAPS', 'DOCUMENTS', 'PEOPLE', and 'SEARCH'. The main content area is titled 'UPLOAD LAYERS' and features a large dashed box for dropping files. Below this, there's a section for 'FILES TO BE UPLOADED' which lists four files: 'postcode_polygon.dbf', 'postcode_polygon.prj', 'postcode_polygon.shp', and 'postcode_polygon.shx'. To the right, the 'PERMISSIONS' section allows users to set access levels: 'Who can view and download this data?' (radio buttons for 'Anyone', 'Any registered user', 'Only users who can edit'), 'Who can edit this data?' (radio buttons for 'Any registered user', 'Only the following users or groups:'), and 'Who can manage and edit this data?'. At the bottom, there's a 'Select the charset or leave default' dropdown set to 'UTF-8/Unicode', and buttons for 'Clear' and 'Upload Files'. The footer shows 'Powered by GeoNode version 2.0.dev20130911150113| Developers | About' and a language selector set to 'English'.

As soon as the import process completes, you will have the possibility to go straight to the layer info page (“Layer Info” button), or to edit the metadata for that layer (“Edit Metadata” button), or to manage the styles for that layer (“Manage Styles”).



GDAL (Raster Data)

Now that we have seen how to convert vector layers into shapefiles using `ogr2ogr`, we will walk through the steps necessary to perform the same operation with Raster layers. For this example, we will work with Arc/Info Binary and ASCII Grid data and convert it into GeoTIFF format for use in GeoNode.

First, you need to download the sample data to work with it. You can do this by executing the following command:

```
$ wget http://84.33.2.26/geonode/sample_asc.tar
```

You will need to uncompress this file by executing this command:

```
$ tar -xvf sample_asc.tar
```

You will be left with the following files on your filesystem:

```
-- batemans_ele
|   |-- dblbnd.adf
|   |-- hdr.adf
|   |-- metadata.xml
|   |-- prj.adf
|   |-- sta.adf
|   |-- w001001.adf
|   |-- w001001x.adf
|-- batemans_elevation.asc
```

The file `batemans_elevation.asc` is an Arc/Info ASCII Grid file and the files in the `batemans_ele` directory are an Arc/Info Binary Grid file.

You can use the `gdalinfo` command to inspect both of these files by executing the following command:

```
$ gdalinfo batemans_elevation.asc
```

The output should look like the following:

```

Driver: AAIGrid/Arc/Info ASCII Grid
Files: batemans_elevation.asc
Size is 155, 142
Coordinate System is ``
Origin = (239681.0000000000000000,6050551.0000000000000000)
Pixel Size = (100.0000000000000000,-100.0000000000000000)
Corner Coordinates:
Upper Left  ( 239681.000, 6050551.000)
Lower Left  ( 239681.000, 6036351.000)
Upper Right ( 255181.000, 6050551.000)
Lower Right ( 255181.000, 6036351.000)
Center      ( 247431.000, 6043451.000)
Band 1 Block=155x1 Type=Float32, ColorInterp=Undefined
NoData Value=-9999

```

You can then inspect the batemans_ele files by executing the following command:

```
$ gdalinfo batemans_ele
```

And this should be the corresponding output:

```

Driver: AIG/Arc/Info Binary Grid
Files: batemans_ele
    batemans_ele/dblbnnd.adf
    batemans_ele/hdr.adf
    batemans_ele/metadata.xml
    batemans_ele/prj.adf
    batemans_ele/sta.adf
    batemans_ele/w001001.adf
    batemans_ele/w001001x.adf
Size is 155, 142
Coordinate System is:
PROJCS["unnamed",
    GEOGCS["GDA94",
        DATUM["Geocentric_Datum_of_Australia_1994",
            SPHEROID["GRS 1980",6378137,298.257222101,
                AUTHORITY["EPSG","7019"]],
            TOWGS84[0,0,0,0,0,0,0],
            AUTHORITY["EPSG","6283"]],
        PRIMEM["Greenwich",0,
            AUTHORITY["EPSG","8901"]],
        UNIT["degree",0.0174532925199433,
            AUTHORITY["EPSG","9122"]],
        AUTHORITY["EPSG","4283"]],
    PROJECTION["Transverse_Mercator"],
    PARAMETER["latitude_of_origin",0],
    PARAMETER["central_meridian",153],
    PARAMETER["scale_factor",0.9996],
    PARAMETER["false_easting",500000],
    PARAMETER["false_northing",1000000],
    UNIT["METERS",1]]
Origin = (239681.0000000000000000,6050551.0000000000000000)
Pixel Size = (100.0000000000000000,-100.0000000000000000)
Corner Coordinates:
Upper Left  ( 239681.000, 6050551.000) (150d 7'28.35"E, 35d39'16.56"S)
Lower Left  ( 239681.000, 6036351.000) (150d 7'11.78"E, 35d46'56.89"S)
Upper Right ( 255181.000, 6050551.000) (150d17'44.07"E, 35d39'30.83"S)

```

(continues on next page)

(continued from previous page)

```

Lower Right ( 255181.000, 6036351.000) (150d17'28.49"E, 35d47'11.23"S)
Center      ( 247431.000, 6043451.000) (150d12'28.17"E, 35d43'13.99"S)
Band 1 Block=256x4 Type=Float32, ColorInterp=Undefined
      Min=-62.102 Max=142.917
NoData Value=-3.4028234663852886e+38

```

You will notice that the `batemans_elevation.asc` file does *not* contain projection information while the `batemans_ele` file does. Because of this, let's use the `batemans_ele` files for this exercise and convert them to a GeoTiff for use in GeoNode. We will also reproject this file into WGS84 in the process. This can be accomplished with the following command.

```
$ gdalwarp -t_srs EPSG:4326 batemans_ele batemans_ele.tif
```

The output will show you the progress of the conversion and when it is complete, you will be left with a `batemans_ele.tif` file that you can upload to your GeoNode.

You can inspect this file with the `gdalinfo` command:

```
$ gdalinfo batemans_ele.tif
```

Which will produce the following output:

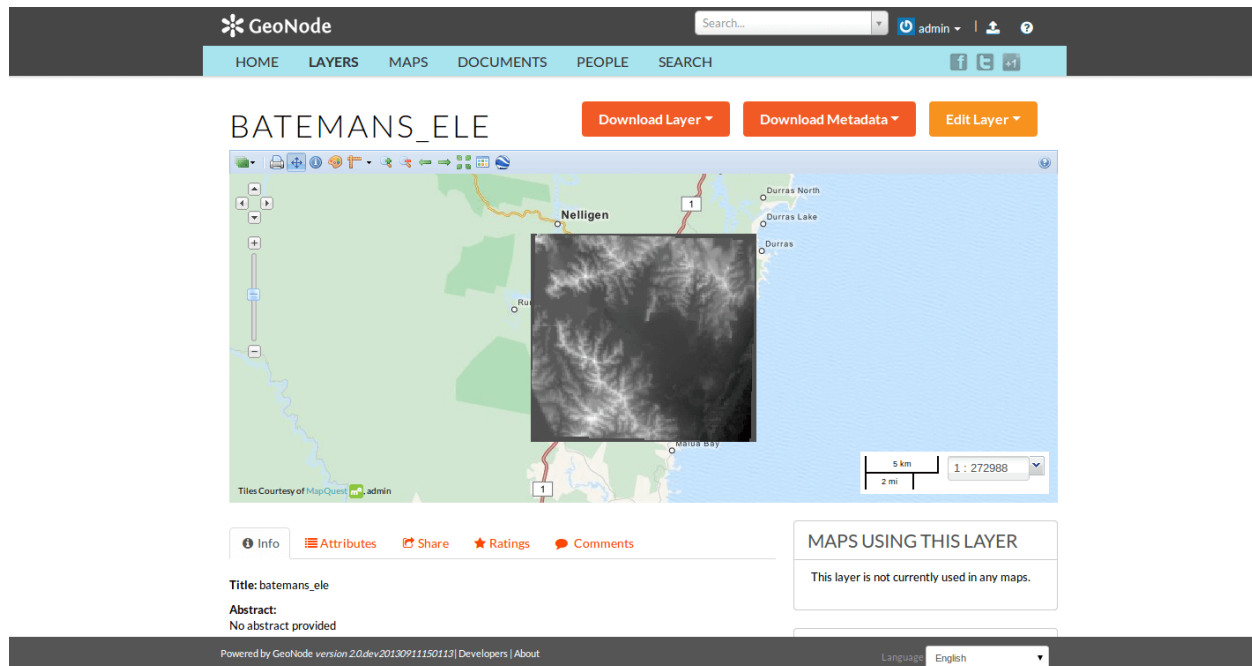
```

Driver: GTiff/GeoTIFF
Files: batemans_ele.tif
Size is 174, 130
Coordinate System is:
GEOGCS["WGS 84",
    DATUM["WGS_1984",
        SPHEROID["WGS 84",6378137,298.257223563,
            AUTHORITY["EPSG","7030"]],
        AUTHORITY["EPSG","6326"]],
    PRIMEM["Greenwich",0],
    UNIT["degree",0.0174532925199433],
    AUTHORITY["EPSG","4326"]]
Origin = (150.119938943722502,-35.654598806259330)
Pixel Size = (0.0010111114155919,-0.0010111114155919)
Metadata:
  AREA_OR_POINT=Area
Image Structure Metadata:
  INTERLEAVE=BAND
Corner Coordinates:
Upper Left  ( 150.1199389, -35.6545988) (150d 7'11.78"E, 35d39'16.56"S)
Lower Left  ( 150.1199389, -35.7860436) (150d 7'11.78"E, 35d47' 9.76"S)
Upper Right ( 150.2958728, -35.6545988) (150d17'45.14"E, 35d39'16.56"S)
Lower Right ( 150.2958728, -35.7860436) (150d17'45.14"E, 35d47' 9.76"S)
Center      ( 150.2079059, -35.7203212) (150d12'28.46"E, 35d43'13.16"S)
Band 1 Block=174x11 Type=Float32, ColorInterp=Gray

```

You can then follow the same steps we used above to upload the GeoTIFF file we created into the GeoNode, and you will see your layer displayed in the Layer Info page.

Now that you have seen how to convert layers with both OGR and GDAL, you can use these techniques to work with your own data and get it prepared for inclusion in your own GeoNode.



3.2.1.4.11 Implementing S3 Bucket for Static and Media Files

If you have access to an Amazon S3 bucket, using this resource for your site's static and media files can improve your site's performance. We'll assume you have an account and can create new users and S3 buckets already. This tutorial will walk you through implementing your GeoNode instance to use an S3 bucket for static and media files.

Configuring S3 Bucket

Before proceeding, preserving the security and manageability of the system should be considered. Therefore, we're going to suggest creating a new user account to have access to this S3 bucket. With the new username and password only being related to this one bucket, any compromise to the system will only affect this bucket. Additionally, creating a new user account just for the site will allow you to pass on the information to a new maintainer in the future.

Instructions

Creating Resources

- [Create the S3 bucket](#)
- Create a new user: Go to [AWS IAM](#). Select "Create new users" and follow the instructions, making sure you leave "Generate an access key for each User" selected.
- Download the user's access keys (access key and secret access key). Go to the new user's Security Credentials and click "Manage access keys". Download the credentials for the access key that was created and put the information somewhere safe. **You will not be able to download this information again.**
- Retrieve the new user's ARN (Amazon Resource Name). Go to the user's Summary tab to get the information. For example: `arn:aws:iam::123456789012:user/username`

Adding Bucket Policy

Next, the bucket policy needs to be set. In the [S3 management console](#) head to the bucket properties and add a new bucket policy with the information below. Use the name of the bucket you created for `S3_BUCKET_NAME` and the user's ARN retrieved in the previous step for `USER_ARN`.

```
{
  "Statement": [
    {
      "Sid": "PublicReadForGetBucketObjects",
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": ["s3:GetObject"],
      "Resource": ["arn:aws:s3:::S3_BUCKET_NAME/*"]
    },
    {
      "Action": "s3:*",
      "Effect": "Allow",
      "Resource": [
        "arn:aws:s3:::S3_BUCKET_NAME",
        "arn:aws:s3:::S3_BUCKET_NAME/*"
      ],
      "Principal": {
        "AWS": [
          "USER_ARN"
        ]
      }
    }
  ]
}
```

Applying CORS

Since assets are going to be served on the site from an external domain now (the S3 bucket), it needs to be configured with CORS. To do so, go to the S3 bucket's Properties > Permissions > Add CORS Configuration and paste this in:

```
<CORSConfiguration>
  <CORSRule>
    <AllowedOrigin>*</AllowedOrigin>
    <AllowedMethod>GET</AllowedMethod>
    <MaxAgeSeconds>3000</MaxAgeSeconds>
    <AllowedHeader>Authorization</AllowedHeader>
  </CORSRule>
</CORSConfiguration>
```

Additional Users

If you wish to configure more users, just follow all the steps after creating the S3 bucket with additional users. Add the following to the S3 bucket policy in the Statement:


```
{
  "Action": "s3:ListBucket",
  "Effect": "Allow",
  "Resource": "arn:aws:s3:::S3_BUCKET_NAME",
  "Principal": {
    "AWS": [
      "USER_ARN"
    ]
  }
}
```

Setting Environment Variables

GeoNode already has settings ready to be configured with the created S3 bucket. Simply set the following environment variables with the appropriate information gained from the steps above:

- `S3_BUCKET_NAME`, the name of what the bucket created in the first step.
- `AWS_ACCESS_KEY_ID`, the access key id downloaded earlier, e.g. `AKIAIOSFODNN7EXAMPLE`
- `AWS_SECRET_ACCESS_KEY`, the secret access key downloaded earlier, e.g. `wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY`

Static Files

In order to serve your **static files** through the S3 bucket, you'll additionally need to set the environment variable `S3_STATIC_ENABLED` to `True`.

Media Files

In order to serve your **media files** through the S3 bucket, you'll additionally need to set the environment variable `S3_MEDIA_ENABLED` to `True`.

Migrating an Existing Site's Data

If you already have a GeoNode site running and want to change it so your data is served through an S3 bucket instead, you will need to move your previously existing data into the bucket. Moving your data is beyond the scope of this tutorial, but Amazon provides helpful tools for managing your bucket such as the AWS CLI tools.

3.2.1.4.12 Backup & Restore GeoNode - Data Migration

The admin command to backup and restore GeoNode, allows to extract consistently the GeoNode and GeoServer data models in a serializable meta-format which is being interpreted later by the restore procedure in order to exactly rebuild the whole structure, according to the current instance version (which may also be different from the starting one).

In particular the tool helps developers and admins to correctly extract and serialize the following resources are on the storage and deserialize on the target GeoNode/GeoServer instance:

- **GeoNode** (Resource Base Model):
 1. Layers (both raster and vectors)

2. Maps
3. Documents
4. People with Credentials
5. Permissions
6. Associated Styles
7. Static data and templates

- **GeoServer** (Catalog):

1. OWS Services configuration and limits
2. Security model along with auth filters configuration, users and credentials
3. Workspaces
4. Stores (both DataStores and CoverageStores)
5. Layers
6. Styles

The tool exposes two GeoNode Management Commands, 'backup' and 'restore'.

The commands allow to:

1. Fully backup GeoNode data and fixtures on a zip archive
2. Fully backup GeoServer configuration (physical datasets - tables, shapefiles, geotiffs)
3. Fully restore GeoNode and GeoServer fixtures and catalog from the zip archive
4. Migrate fixtures from old GeoNode models to the new one

The usage of those commands is quite easy and straight. It is possible to run the backup and restore commands from the GeoNode Admin panel also.

The first step is to ensure that everything is correctly configured and the requisites respected in order to successfully perform a backup and restore of GeoNode.

Warning: It is worth notice that this functionality requires the latest [GeoServer Extension](#) (2.9.x or greater) for GeoNode in order to correctly work.

Note: GeoServer full documentation is also available here [GeoServer Docs](#)

Requisites and Setup

Before running a GeoNode backup / restore, it is necessary to ensure everything is correctly configured and setup.

Settings

Accordingly to the admin needs, the file `settings.ini` must be tuned up a bit before running a backup / restore.

It can be found at `geonode/base/management/commands/settings.ini` and by default it contains the following properties:

```

[database]
pgdump = pg_dump
pgrestore = pg_restore

[geoserver]
datadir = /opt/gs_data_dir
dumpvectordata = yes
dumprasterdata = yes

[fixtures]
#NOTE: Order is important
apps    = people,account,avatar.avatar,base.backup,base.license,base.topiccategory,
↳base.region,base.resourcebase,base.contactrole,base.link,base.restrictioncodetype,
↳base.spatialrepresentation,guardian.userobjectpermission,guardian.
↳groupobjectpermission,layers.uploadsession,layers.style,layers.layer,layers.
↳attribute,layers.layerfile,maps.map,maps.maplayer,maps.mapsnapshot,documents.
↳document,taggit

dumps   = people,accounts,avatars,backups,licenses,topiccategories,regions,
↳resourcebases,contactroles,links,restrictioncodetypes,spatialrepresentation,
↳userpermissions,grouppermissions,uploadsessions,styles,layers,attributes,
↳layerfiles,maps,maplayers,mapsnapshots,documents,tags

# Migrate from GN 2.0 to GN 2.4
#migrations = base.resourcebase, layers.layer, layers.attribute, maps.map, maps.maplayer
#manglers    = gn20_to_24.ResourceBaseMangler, gn20_to_24.LayerMangler, gn20_to_24.
↳LayerAttributesMangler, gn20_to_24.MapMangler, gn20_to_24.MapLayersMangler

# Migrate from GN 2.4 to GN 2.4
migrations = base.resourcebase, layers.layer, layers.attribute, maps.map, maps.maplayer
manglers    = gn24_to_24.ResourceBaseMangler, gn24_to_24.LayerMangler, gn24_to_24.
↳LayerAttributesMangler, gn24_to_24.DefaultMangler, gn24_to_24.MapLayersMangler

```

The `settings.ini` has few different sections that must carefully checked before running a backup / restore command.

Settings: [database] Section

```

[database]
pgdump = pg_dump
pgrestore = pg_restore

```

This section is quite simple. It contains only two (2) properties:

- `pgdump`; the path of the `pg_dump` local command.
- `pgrestore`; the path of the `pg_restore` local command.

Warning: Those properties are ignored in case GeoNode is not configured to use a DataBase as backend (see `settings.py` and `local_settings.py` sections)

Note: Database connection settings (both for GeoNode and GeoServer) will be taken from `settings.py` and `local_settings.py` configuration files. Be sure they are correctly configured (on the target GeoNode instance)

too) and the DataBase server is accessible while executing a backup / restore command.

Settings: [geoserver] Section

```
[geoserver]
datadir = /opt/gs_data_dir
dumpvectordata = yes
dumprasterdata = yes
```

This section allows to enable / disable a full data backup / restore of GeoServer.

- *datadir*; the full path of GeoServer Data Dir, by default `/opt/gs_data_dir`. The path **must** be accessible and **fully writable** by the geonode and / or httpd server users when executing a backup / restore command.
- *dumpvectordata*; a boolean allowing to disable dump of vectorial data from GeoServer (shapefiles or DB tables). If false (or no) vectorial data won't be stored / re-stored.
- *dumprasterdata*; a boolean allowing to disable dump of raster data from GeoServer (geotiffs). If false (or no) raster data won't be stored / re-stored.

Warning: Enabling those options **requires** that the GeoServer Data Dir is accessible and **fully writable** by the geonode and / or httpd server users when executing a backup / restore command.

Settings: [fixtures] Section

```
[fixtures]
#NOTE: Order is important
apps    = people,account,avatar.avatar,base.backup,base.license,base.topiccategory,
↳base.region,base.resourcebase,base.contactrole,base.link,base.restrictioncodetype,
↳base.spatialrepresentation,guardian.userobjectpermission,guardian.
↳groupobjectpermission,layers.uploadsession,layers.style,layers.layer,layers.
↳attribute,layers.layerfile,maps.map,maps.maplayer,maps.mapsnapshot,documents.
↳document,taggit

dumps   = people,accounts,avatars,backups,licenses,topiccategories,regions,
↳resourcebases,contactroles,links,restrictioncodetypes,spatialrepresentation,
↳userpermissions,grouppermissions,uploadsessions,styles,layers,attributes,
↳layerfiles,maps,maplayers,mapsnapshots,documents,tags

# Migrate from GN 2.0 to GN 2.4
#migrations = base.resourcebase, layers.layer, layers.attribute, maps.map, maps.maplayer
#manglers    = gn20_to_24.ResourceBaseMangler, gn20_to_24.LayerMangler, gn20_to_24.
↳LayerAttributesMangler, gn20_to_24.MapMangler, gn20_to_24.MapLayersMangler

# Migrate from GN 2.4 to GN 2.4
migrations = base.resourcebase, layers.layer, layers.attribute, maps.map, maps.maplayer
manglers    = gn24_to_24.ResourceBaseMangler, gn24_to_24.LayerMangler, gn24_to_24.
↳LayerAttributesMangler, gn24_to_24.DefaultMangler, gn24_to_24.MapLayersMangler
```

This section is the most complex one. Usually you don't need to modify it. Only an expert user who knows Python and GeoNode model structure should modify this section.

What its properties mean:

- *apps*; this is an ordered list of GeoNode Object Models (or Django apps). The backup / restore procedure will dump / restore the fixtures in a portable format.
- *dumps*; this is the list of files associated to the Django apps. The order **must** be the same of the property above. Each name represents the file name where to dump / read the single app fixture.
- *migrations*; some fixtures must be enriched or updated before restored on the target model. This section allows to associate specific manglers to the fixtures. Manglers are simple Python classes which simply converts some attributes to other formats.
- *manglers*; the Python mangler class to execute accordingly to the fixture indicated by the *migrations* property. Manglers classes **must** be located into the `geonode/base/management/commands/lib` folder.

Note: Manglers **must** be used when migrating from a GeoNode version to another one, i.e. where the original model differs from the target one. With the default distribution are provided manglers to convert from GeoNode 2.0 to GeoNode 2.4. Other versions may require other manglers or updates to the default ones.

Mangler Example

As specified on the section above, manglers are Python classes allowing developers to enrich / modify a fixture in order to fit the target GeoNode model.

The structure of a mangler is quite simple. Lets examine the `ResourceBaseMangler` of the `gn_20_to_24` library, a mangler used to convert a GeoNode 2.0 Resource Base to a GeoNode 2.4 one.

```
class ResourceBaseMangler(DefaultMangler):

    def default(self, obj):
        # Let the base class default method raise the TypeError
        return json.JSONEncoder.default(self, obj)

    def decode(self, json_string):
        """
        json_string is basically a string that you give to json.loads method
        """
        default_obj = super(ResourceBaseMangler, self).decode(json_string)

        # manipulate your object any way you want
        # ....
        upload_sessions = []
        for obj in default_obj:
            obj['pk'] = obj['pk'] + self.basepk

            obj['fields']['featured'] = False
            obj['fields']['rating'] = 0
            obj['fields']['popular_count'] = 0
            obj['fields']['share_count'] = 0
            obj['fields']['is_published'] = True
            obj['fields']['thumbnail_url'] = ''

            if 'distribution_url' in obj['fields']:
                if not obj['fields']['distribution_url'] is None and 'layers' in obj[
↪ 'fields']['distribution_url']:
```

(continues on next page)

(continued from previous page)

```

        obj['fields']['polymorphic_ctype'] = ["layers", "layer"]

        try:
            p = '(?P<protocol>http.*://)?(?P<host>[^:/ ]+).?(?P<port>[0-
↪9]*) (?P<details_url>.*)'
            m = re.search(p, obj['fields']['distribution_url'])
            if 'http' in m.group('protocol'):
                obj['fields']['detail_url'] = self.siteurl + m.group(
↪'details_url')
            else:
                obj['fields']['detail_url'] = self.siteurl + obj['fields
↪']['distribution_url']
        except:
            obj['fields']['detail_url'] = obj['fields']['distribution_url
↪']

        else:
            obj['fields']['polymorphic_ctype'] = ["maps", "map"]

        try:
            obj['fields'].pop("distribution_description", None)
        except:
            pass

        try:
            obj['fields'].pop("distribution_url", None)
        except:
            pass

        try:
            obj['fields'].pop("thumbnail", None)
        except:
            pass

        upload_sessions.append(self.add_upload_session(obj['pk'], obj['fields'][
↪'owner']))

        default_obj.extend(upload_sessions)

        return default_obj

    def add_upload_session(self, pk, owner):
        obj = dict()

        obj['pk'] = pk
        obj['model'] = 'layers.uploadsession'

        obj['fields'] = dict()
        obj['fields']['user'] = owner
        obj['fields']['traceback'] = None
        obj['fields']['context'] = None
        obj['fields']['error'] = None
        obj['fields']['processed'] = True
        obj['fields']['date'] = datetime.datetime.now().strftime("%Y-%m-%dT%H:%M:%S")

        return obj

```

1. It extends the DefaultMangler.

The DefaultMangler is a basic class implementing a JSONDecoder

```
class DefaultMangler(json.JSONDecoder):

    def __init__(self, *args, **kwargs):

        self.basepk = kwargs.get('basepk', -1)
        self.owner = kwargs.get('owner', 'admin')
        self.datastore = kwargs.get('datastore', '')
        self.siteurl = kwargs.get('siteurl', '')

        super(DefaultMangler, self).__init__(*args)

    def default(self, obj):
        # Let the base class default method raise the TypeError
        return json.JSONEncoder.default(self, obj)

    def decode(self, json_string):
        """
        json_string is basicly string that you give to json.
        ↪ loads method
        """
        default_obj = super(DefaultMangler, self).decode(json_
        ↪ string)

        # manipulate your object any way you want
        # ....

        return default_obj
```

By default this mangler unmarshalls GeoNode Object Model from JSON and returns it to the management command.

The GeoNode Object Model can be modified while decoding by extending the `def decode(self, json_string)` method.

- *json_string*; actual parameter contains the JSON representation of the fixture.
- *default_obj*; is the Python object decoded from the JSON representation of the fixture.

2. It overrides the `def decode(self, json_string)` method.

The decoded Python object can be enriched / modified before returning it to the management command.

From Command Line

The following sections shows instructions on how to perform backup / restore from the command line by using the Admin Management Commands.

In order to obtain a basic user guide for the management command from the command line, just run

```
python manage.py backup --help
python manage.py restore --help
```

`--help` will provide the list of available command line options with a brief description.

It is worth notice that both commands allows the following option

```
python manage.py restore

-f, --force                        # Forces the execution without asking_
↳for confirmation
-c CONFIG, --config=CONFIG        # Use custom settings.ini configuration_
↳file
--geoserver-data-dir=GS_DATA_DIR  # Geoserver data directory
--dump-geoserver-vector-data      # Dump geoserver vector data
--no-geoserver-vector-data        # Don't dump geoserver vector data
--dump-geoserver-raster-data      # Dump geoserver raster data
--no-geoserver-raster-data        # Don't dump geoserver raster data
--skip-geoserver                  # Skips geoserver backup
--backup-file=BACKUP_FILE         # Backup archive containing GeoNode_
↳data to restore.
--backup-dir=BACKUP_DIR           # Backup directory containing GeoNode_
↳data to restore.
```

```
python manage.py backup

-f, --force                        # Forces the execution without asking_
↳for confirmation
-c CONFIG, --config=CONFIG        # Use custom settings.ini configuration_
↳file
--geoserver-data-dir=GS_DATA_DIR  # Geoserver data directory
--dump-geoserver-vector-data      # Dump geoserver vector data
--no-geoserver-vector-data        # Don't dump geoserver vector data
--dump-geoserver-raster-data      # Dump geoserver raster data
--no-geoserver-raster-data        # Don't dump geoserver raster data
--skip-geoserver                  # Skips geoserver backup
--backup-dir=BACKUP_DIR           # Destination folder where to store the_
↳backup archive.
# It must be writable.
```

Which will instruct the management command to not ask for confirmation from the user. It enables basically a non-interactive mode.

Backup

In order to perform a backup just run the command:

```
python manage.py backup --backup-dir=<target_bk_folder_path>
```

The management command will automatically generate a .zip archive file on the target folder in case of success.

Restore

In order to perform a restore just run the command:

```
python manage.py restore --backup-file=<target_restore_file_path>
```

Restore requires the path of one .zip archive containing the backup fixtures.

Warning: The Restore will **overwrite** the whole target GeoNode / GeoServer users, catalog and database, so be very careful.

From GeoNode Admin GUI

1. Login as `admin` and click on Admin menu option
2. Look for Backups on Base section
3. Add a new backup
4. Insert a Name and a Description; also you **must** provide the Base folder where the backups will be stored

Warning: the Base folder **must** be **fully writable** from both `geonode` and `httpd` server system users.

5. Click on `save` and go back to the Backups list main section
6. The new Backup is not **ready** until you perform the Run Backup action; in order to do that select the backup to run and from the Action menu select Run the Backup

Note: A Backup is not *ready* until the `Location` attribute is filled

7. Click on `Yes, I'm sure` on the next section in order to perform the Backup

Note: The server page will wait for the Backup to finish (or fail).

8. The server page will wait for the Backup to finish (or fail); at the end of the Backup you will be redirected to the main list page.











Note: At a successfull run, the `Location` attribute is filled with the full path of the backup archive

Warning: A Backup can always being updated later and / or executed again. The `Location` attribute will be updated accordingly.

9. Execute as many Backups as you want; they can all point to the same Base Folder, the new backups will generate new unique archive files any time.

Menu



-  Upload Layers
-  Profile
-  Recent Activity
-  Inbox
-  Announcements
-  Remote Services
-  Invite User
-  GeoServer
-  Admin
-  Help

Log out

Avatar		
Avatars	+ Add	✎ Change
Base		
Backups	+ Add	✎ Change
Contact roles	+ Add	✎ Change
Hierarchical keywords	+ Add	✎ Change
Licenses	+ Add	✎ Change
Links	+ Add	✎ Change
Metadata Regions	+ Add	✎ Change
Metadata Restriction Code Types		✎ Change
Metadata Spatial Representation Types		✎ Change
Metadata Topic Categories		✎ Change
Dialogs		

Add backup +

Add backup en ▼

en	
Name:	<input type="text" value="Test Backup"/>
en	
Description:	<div>This is my backup for today</div>
Base folder:	<input type="text" value="/opt/backups"/>
Location:	(None)

Select backup to change

Add backup +

2016 September 19			
Action: <input type="text"/> Go 0 of 1 selected			
<input type="checkbox"/> ID	Name	Date	Location
<input type="checkbox"/> 2	Test Backup	Sept. 19, 2016, 8:29 a.m.	(None)
1 backup			

Select backup to change

2016 September 19			
Action: <input type="text"/> Go 1 of 1 selected			
<input checked="" type="checkbox"/>	ID		D.
<input checked="" type="checkbox"/>	2		Se
1 backup			

<input type="checkbox"/> Location
(None)

GeoNode administration

[Home](#) > Confirm run of Backups:

Confirm run of Backups:

Are you sure you want to run the selected Backups?

Test Backup

[Cancel](#)

[Home](#) > [Base](#) > Backups

✔ Executed Backup: Test Backup

Select backup to change< 2016 **September 19**Action: Go 0 of 1 selected

<input type="checkbox"/>	ID	Name	Date	Location
<input type="checkbox"/>	2	Test Backup	Sept. 19, 2016, 8:29 a.m.	/opt/backups/2016-09-19_083614.zip

1 backup

Location
/opt/backups/2016-09-19_083614.zip

Select backup to change< 2016 **September 19**Action: Go 0 of 3 selected

<input type="checkbox"/>	ID	Name	Date	Location
<input type="checkbox"/>	2	Test Backup	Sept. 19, 2016, 8:29 a.m.	/opt/backups/2016-09-19_083614.zip
<input type="checkbox"/>	3	Test Backup 2	Sept. 19, 2016, 8:40 a.m.	/opt/backups/2016-09-19_084041.zip
<input type="checkbox"/>	4	Test Backup N	Sept. 19, 2016, 8:41 a.m.	/opt/backups/2016-09-19_084140.zip

3 Backups

10. In order to Restore a zip archive, just select the instance to restore from the list and from the Action menu lunch the Run the Restore option.

Select backup to change

< 2016 September 19

Action: Go 1 of

<input type="checkbox"/>	ID	
<input type="checkbox"/>	2	
<input checked="" type="checkbox"/>	3	Test Backup 2
<input type="checkbox"/>	4	Test Backup N

3 Backups

11. Click on Yes, I'm sure on the next section in order to perform the Backup

GeoNode administration

[Home](#) > Confirm run of Backups:

Confirm run of Backups:

Are you sure you want to run the selected Backups?

Test Backup

[Cancel](#)

Note: The server page will wait for the Backup to finish (or fail).

Warning: The following target GeoNode folders **must** be **fully writable** from both geonode and httpd server system users

- geoserver_data_dir/data

- geonode / settings.MEDIA_ROOT
- geonode / settings.STATIC_ROOT
- geonode / settings.STATICFILES_DIRS
- geonode / settings.TEMPLATE_DIRS
- geonode / settings.LOCALE_PATHS

Warning: The Restore will **overwrite** the whole target GeoNode / GeoServer users, catalog and database, so be very careful.

3.2.1.4.13 GeoNode Monitoring

Contents

- *GeoNode Monitoring*
 - *Internal Monitoring Application (geonode.contrib.monitoring)*
 - * *Base concepts and objects*
 - * *Installation*
 - * *Enable the collect_metrics cron*
 - *cront job*
 - *supervisor*
 - * *Configuration*
 - * *Usage*
 - *Dashboard*
 - *Top bar and indicators*
 - *Software Performance*
 - *Hartware Performance*
 - *Errors*
 - *Alerts*
 - *Integration with GeoHealthCheck*

Internal Monitoring Application (geonode.contrib.monitoring)

Note: This application requires MaxMind's GeoIP database file.

Base concepts and objects

GeoNode monitoring is a configurable monitoring application, that allows internal resources and hardware resources monitoring for GeoNode installations, including GeoServer deployments.

Monitoring application is configurable, so different deployment scenarios could be handled - from GeoNode and GeoServer running on single host, through distributed installations, where GeoServer is deployed to several hosts.

Monitoring application uses three base entity classes to describe elements of reality: Host, Service Type and Service.

- Host is an object describing physical (or virtual) instance of operating system on which GN or GS is running. This object exists only for grouping and is not used directly by monitoring.
- Service Type is a description of kind of Service. Depending on service type, different metrics are stored, and different data collection mechanisms are used. Additionally, for system monitoring, it's not conducted directly, but with GeoNode or GeoServer as monitoring agent. That means, no additional software installation is needed to monitor system, but also, hosts that don't have GeoNode or GeoServer installed, won't be monitored. There are four service types:
 - hostgeonode, hostgeoserver - those types describe system monitoring probes that are running with GeoNode or GeoServer respectively,
 - geonode, geoserver - application-level probes that monitor one specific GeoNode or GeoServer instance.
- Service describes one specific instance of probe, either host-level or application-level. Service references Host and Service Type. Each service must be named, and name should be system-wide unique.

As mentioned above, each Service Type keeps a set of metrics, specific for that type. A metric is a description of measured value, for example: number of requests, response size or time, cpu usage, free memory etc. Each Service Type has it's own metrics set. Metric value may be either value counter (like country of user), numeric counter (like number of requests) or rate (like bytes in/out on network interface).

Besides metric data, monitoring will also store exception information for exceptions that were captured during request handling.

Data are collected periodically (at most every 1 minute), aggregated and stored in aggregated form. User can see data from predefined relative periods (last minute, last 10 minutes, last hour, last day, last week).

User can enable and configure automated checks, which will be run after each collection/aggregation cycle, and will emit notifications if metric values in that run exceed configured thresholds.

Installation

Warning: This plugin requires a Potgresql DB backend enabled

- ensure UTC Timezone to your DB

```
psql -c 'set timezone=UTC;'
```

- enable `MONITORING_ENABLED` flag and ensure that following code is in your settings:


```

CORS_ORIGIN_ALLOW_ALL = True

MONITORING_ENABLED = True
# add following lines to your local settings to enable monitoring
if MONITORING_ENABLED:
    INSTALLED_APPS += ('geonode.contrib.monitoring',)
    MIDDLEWARE_CLASSES += ('geonode.contrib.monitoring.middleware.MonitoringMiddleware
    ↪',)
    MONITORING_CONFIG = None
    MONITORING_HOST_NAME = 'localhost'
    MONITORING_SERVICE_NAME = 'local-geonode'
    MONITORING_HOST_NAME = SITE_HOST_NAME

INSTALLED_APPS += ('geonode.contrib.ows_api',)

GEOIP_PATH = os.path.join(os.path.dirname(__file__), '..', 'GeoLiteCity.dat')

```

- run `DJANGO_SETTINGS_MODULE=<project_name>.settings python manage.py migrate monitoring` to apply db schema changes and insert initial data
- run `DJANGO_SETTINGS_MODULE=<project_name>.settings python manage.py updategeoip` to fetch MaxMind's GeoIP database file. It will be written to path specified by `GEOIP_PATH` setting.
- run `DJANGO_SETTINGS_MODULE=<project_name>.settings python manage.py collect_metrics -n -t xml -f --since='<yyyy-mm-dd HH:mm:ss>'` to create first metrics.

Warning: Replace `<yyyy-mm-dd HH:mm:ss>` with a real date time to start with.

- update Sites from admin; make sure it contains a correct host name
- do not forget to enable notifications and configure them from user profile

Enable the `collect_metrics` cron

Warning: Here below you will find instructions for a Ubuntu 16.04 based machine, but the procedure is similar for other OSs. The basic concept is that you must allow the system to run the command `DJANGO_SETTINGS_MODULE=<project_name>.settings python manage.py collect_metrics -n -t xml` (without **-f** and **since**) every minute.

cront job

```
sudo crontab -e
```

```

# Add the following line at the bottom; this will run the supervisor command every_
↪minute
* * * * * supervisorctl start geonode-monitoring

```

supervisor

```
sudo apt install supervisor
sudo service supervisor restart
sudo update-rc.d supervisor enable
```

```
sudo vim /etc/supervisor/conf.d/geonode-monitoring.conf
```

```
[program:geonode-monitoring]
command=<path_to_virtualenv>/geonode/bin/python -W ignore <path_to_your_project>/
↳geonode/manage.py collect_metrics -n -t xml
directory = <path_to_your_project>
environment=DJANGO_SETTINGS_MODULE="<your_project>.settings"
user=<your_user>
numproc=1
stdout_logfile=/var/log/geonode-celery.log
stderr_logfile=/var/log/geonode-celery.log
autostart = true
autorestart = true
startsecs = 10
stopwaitsecs = 600
priority = 998
```

```
sudo service supervisor restart
sudo supervisorctl start geonode-monitoring
sudo supervisorctl status geonode-monitoring
```

```
sudo vim /etc/hosts
```

```
127.0.0.1      localhost
<public_ip>    <your_host.your_domain> <your_host>

# The following lines are desirable for IPv6 capable hosts
```

Configuration

In order to have working monitoring, at least Service should be configured. Let's assume following deployment scenario:

- there's one machine, geo01
- geo01 hosts both GeoNode and GeoServer (including PostgreSQL).
- applications are served with nginx+uwsgi, on port 80, but they are reachable on localhost address.
- GeoServer is served from /geoserver/ path
- GeoNode is served from / path

Here's step-by-step instruction how to create monitoring setup for deployment scenario:

1. Log in as admin, and go to admin section:

The screenshot shows the GeoNode web application interface. At the top, there is a dark blue header with the GeoNode logo, navigation links for 'Data', 'Maps', and 'About', a search bar, and a user profile dropdown for 'admin'. The main content area has a large 'Welcome' heading and a brief description of GeoNode as an open source platform for sharing geospatial data and maps. A 'Get Started »' button is visible. On the right side, a vertical menu is open, showing options like 'Profile', 'Recent Activity', 'Inbox', 'Admin' (highlighted with a red line and a red arrow), 'GeoServer', 'Monitoring', 'Help', and 'Logout'.

2. Go to **monitoring** section (or type `/admin/monitoring/` as a path in URL):

Layers		
Attributes	+ Add	✎ Change
Layers	+ Add	✎ Change
Styles	+ Add	✎ Change
Upload sessions	+ Add	✎ Change
Maps		
Map layers	+ Add	✎ Change
Map snapshots	+ Add	✎ Change
Maps	+ Add	✎ Change
Monitoring		
Exception events	+ Add	✎ Change
Hosts	+ Add	✎ Change
Metric labels	+ Add	✎ Change
Metric notification checks	+ Add	✎ Change
Metrics	+ Add	✎ Change
Monitored resources	+ Add	✎ Change

3. Go to **Hosts**:

Django administration
Welcome, **admin.**

Home > Monitoring

Monitoring administration

Monitoring		
Exception events	+ Add	✎ Change
<u>Hosts</u>	+ Add	✎ Change
Metric labels	+ Add	✎ Change
Metric notification checks	+ Add	✎ Change
Metrics	+ Add	✎ Change
Monitored resources	+ Add	✎ Change
Notification checks	+ Add	✎ Change
Notification metric definitions	+ Add	✎ Change
Notification receivers	+ Add	✎ Change
Ows services	+ Add	✎ Change
Request events	+ Add	✎ Change
Service type metrics	+ Add	✎ Change
Service types	+ Add	✎ Change
<u>Services</u>	+ Add	✎ Change

4. Click on **Add host +**:

Django administration
Welcome, **admin.** View site / Change password / Log out

Home > Monitoring > Hosts

Select host to change

Action: 0 of 1 selected

Add host +

5. Enter following information: * **host**: *localhost* * **ip**: *127.0.0.1* Note, that **host** value is arbitrary. You can enter other name if you like. Don't forget to save.

Django administration Welcome, **admin**. [View site](#) / [Change password](#) / [Log out](#)

[Home](#) > [Monitoring](#) > [Hosts](#) > Host: localhost (127.0.0.1)

Change host History

Name:	<input type="text" value="localhost"/>
Ip:	<input type="text" value="127.0.0.1"/>
<input checked="" type="checkbox"/> Active	

[✖ Delete](#)
[Save and add another](#)
[Save and continue editing](#)
[Save](#)

6. Go to **Services**:

Django administration Welcome, **admin**.

[Home](#) > [Monitoring](#)

Monitoring administration

Monitoring		
Exception events	+ Add	✎ Change
<u>Hosts</u>	+ Add	✎ Change
Metric labels	+ Add	✎ Change
Metric notification checks	+ Add	✎ Change
Metrics	+ Add	✎ Change
Monitored resources	+ Add	✎ Change
Notification checks	+ Add	✎ Change
Notification metric definitions	+ Add	✎ Change
Notification receivers	+ Add	✎ Change
Ows services	+ Add	✎ Change
Request events	+ Add	✎ Change
Service type metrics	+ Add	✎ Change
Service types	+ Add	✎ Change
<u>Services</u>	+ Add	✎ Change

7. Click on **Add service +**:


Django administration Welcome, **admin**. [View site](#) / [Change password](#) / [Log out](#)

[Home](#) > [Monitoring](#) > [Services](#)

Select service to change

Action: 0 of 6 selected

☐ **Name** ☐ **Active** ☐ **Host name** ☐ **Service type**

[Add service](#) 

8. Enter following information:



- **name:** *local-geonode*
- **host:** *localhost*
- **service type:** *geonode*

Django administration Welcome, **admin**. [View site](#) / [Change password](#) / [Log out](#)



[Home](#) > [Monitoring](#) > [Services](#) > [Service: local-geonode@localhost](#)



Change service History

Name:

Host:  

Check interval:

Last check: **Date:** [Today](#)  **Time:** [Now](#) 

Service type:  

☒ Active

Notes:

Url:

[Delete](#) [Save and add another](#) [Save and continue editing](#) [Save](#)







9. Add another **Service** Enter following information:




- **name:** *local-system-geonode*
- **host:** *localhost*
- **service type:** *hostgeonode*
- **url:** *http://localhost/* (should point to GeoNode home page)

Django administration

Welcome, **admin**. [View site](#) / [Change password](#)[Home](#) > [Monitoring](#) > [Services](#) > Service: local-system@localhost

Change service

Name:	<input type="text" value="local-system-geonode"/>
Host:	Host: localhost (127.0.0.1) ▼  
Check interval:	<input type="text" value="00:01:00"/>
Last check:	Date: <input type="text" value="2017-09-08"/> Today  Time: <input type="text" value="10:45:48"/> Now 
Service type:	Service Type: hostgeonode ▼  
<input checked="" type="checkbox"/> Active	
Notes:	<div><div></div></div>
Url:	Currently: http://localhost/ Change: <input type="text" value="http://localhost/"/>

 **Delete**
 **Save and add another**
 **Save and continue editing**

10. Add another **Service** and enter following information:

- **name:** *local-geoserver*
- **host:** *localhost*
- **service type:** *geoserver*
- **url:** *http://localhost/geoserver/* (should point to GeoServer home page)

Django administration
Welcome, **admin**. View site / Change password

Home > Monitoring > Services > Service: local-geoserver@localhost

Change service

Name:

Host:

Host: localhost (127.0.0.1) ▼ ✎ +

Check interval:

Last check:

Date: Today |

Time: Now |

Service type:

Service Type: geoserver ▼ ✎ +

☒ Active

Notes:

Url:

Currently: <http://localhost/geoserver/>

Change:

✖ Delete
Save and add another
Save and continue editing

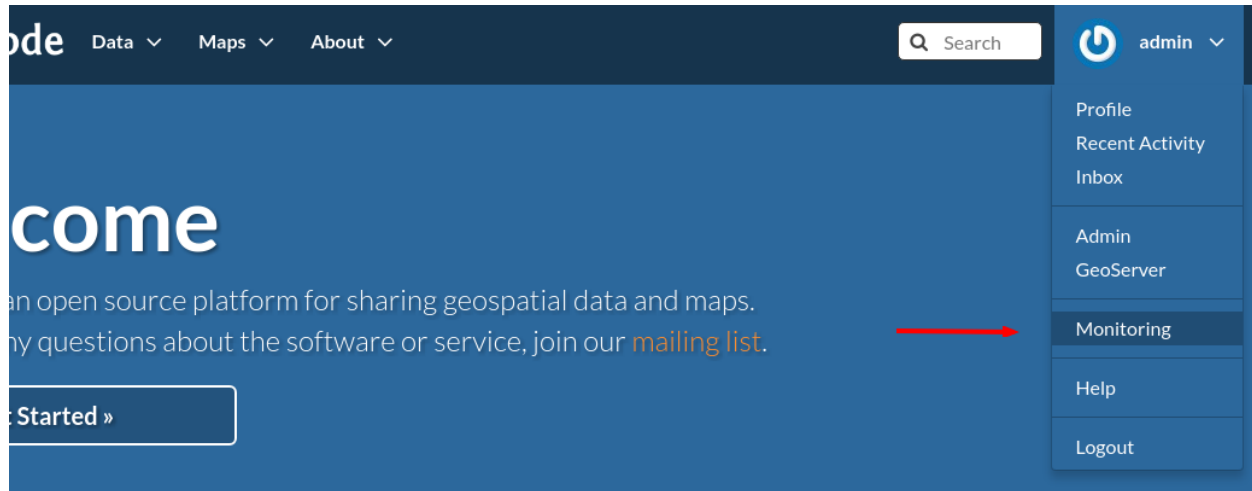
To summarize, following entries should be created in admin/monitoring:

- Host: localhost, with ip: 127.0.0.1
- **Service: local-geonode:**
 - host localhost
 - type geonode
- **Service: local-geoserver:**
 - url <http://localhost/geoserver/>
 - host localhost
 - type geoserver
- **Service: local-system-geonode**
 - url <http://localhost/>

- host localhost
- type hostgeonode

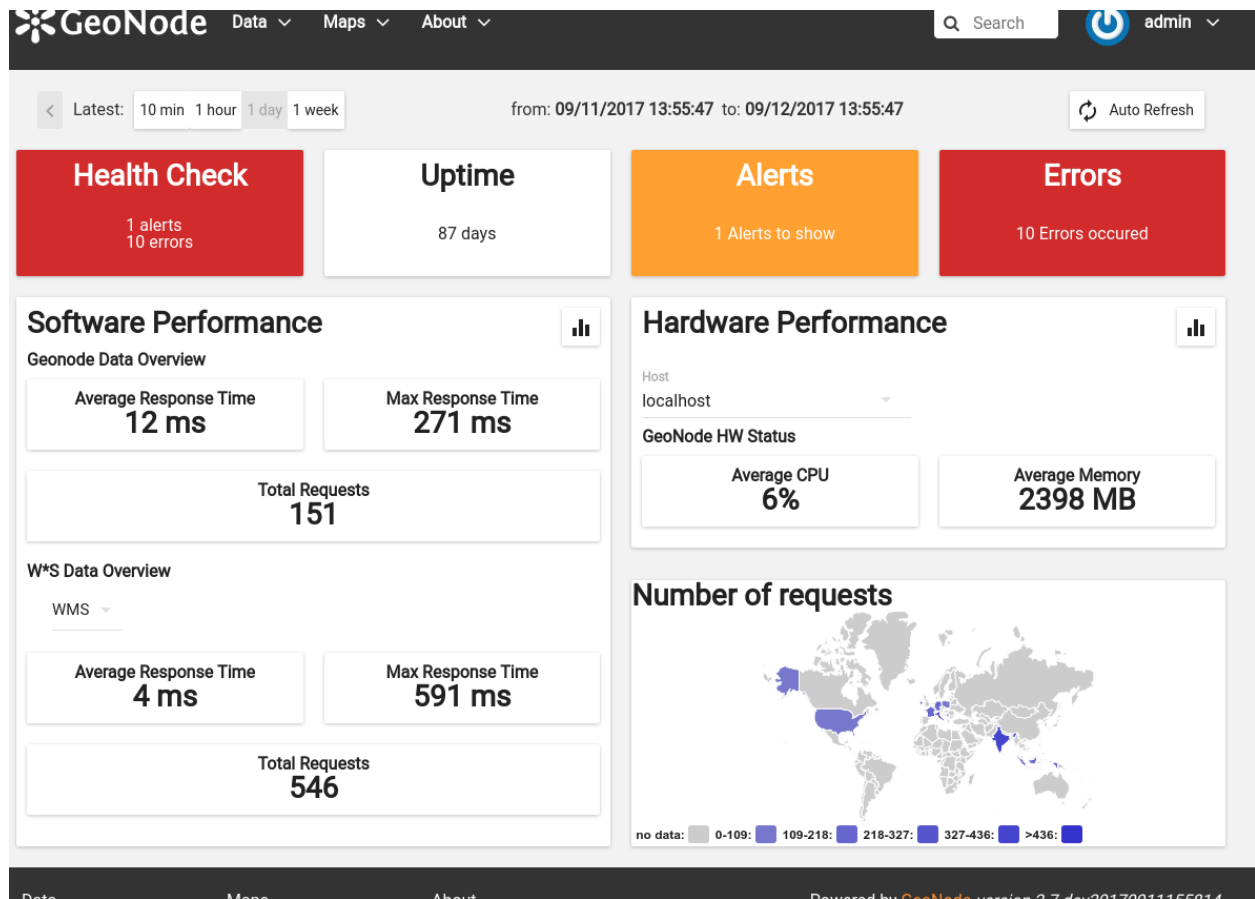
Usage

Monitoring interface is available for superusers only. It's available in profile menu:

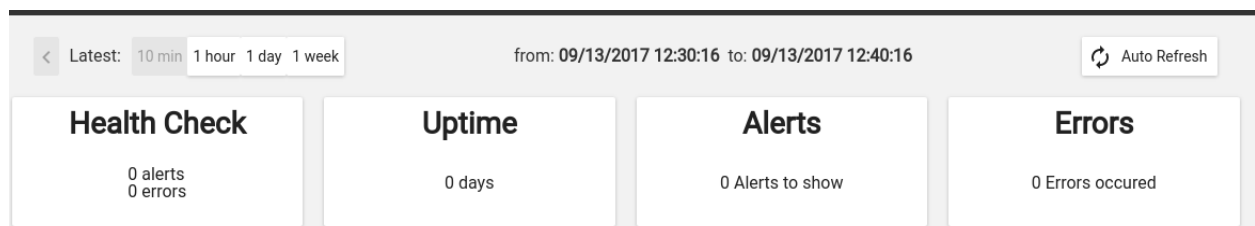


Dashboard

Main view offers overview of recent situation in GeoNode deployment.



Top bar and indicators



With top bar buttons User can:

- go back from nested interface elements (charts, alerts, errors)
- select time window from which data will be aggregated and shown (last 10 minutes, last 1 hour, last day or last week from now)
- see what's currently used time window
- enable/disable autorefresh

Below there are four main health indicators:

- **aggregated Health Check information.** This element will be:
 - *green* if there is no alerts nor errors
 - *yellow* if there are alerts

– *red* if there are errors

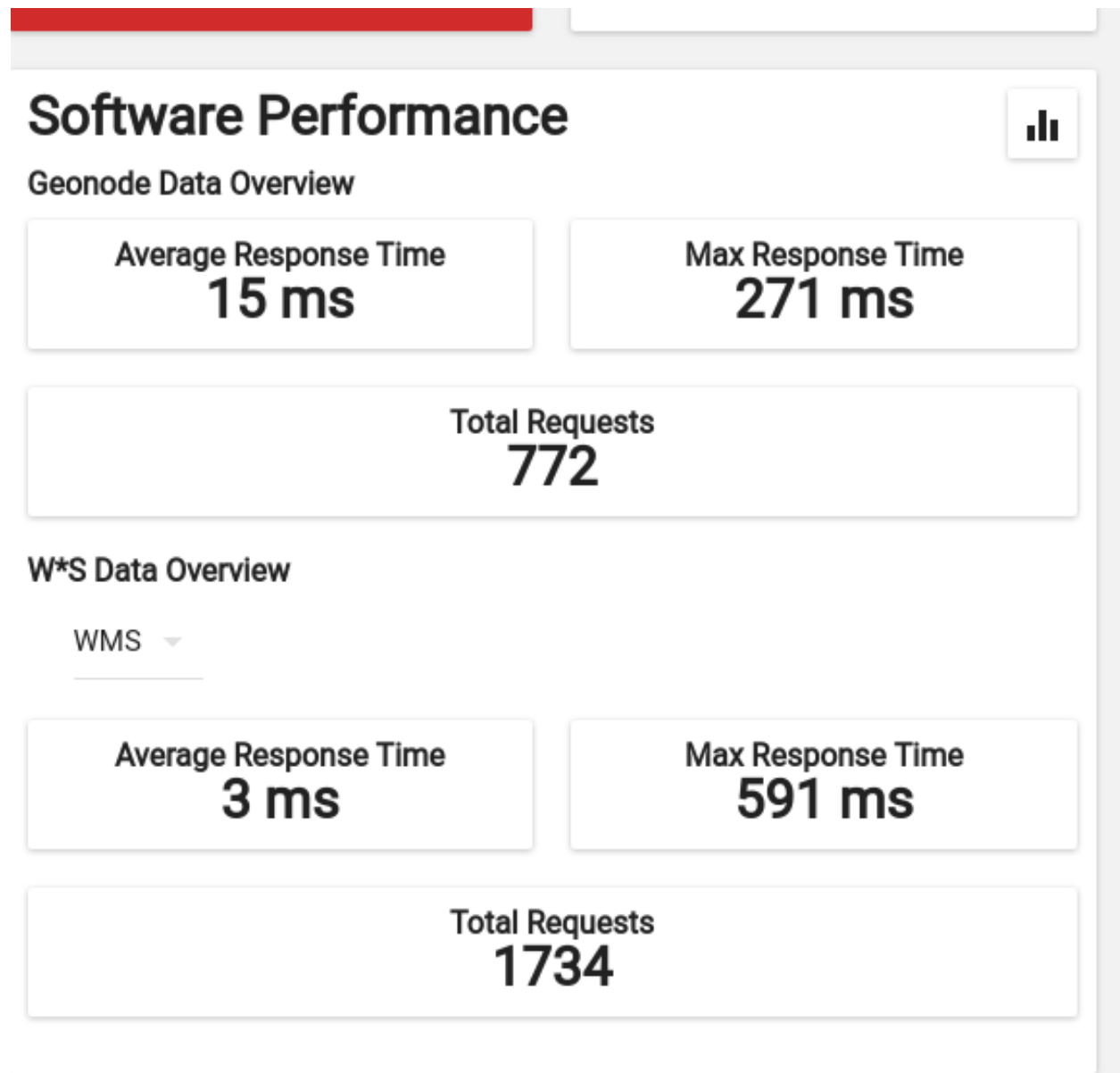
- **Uptime** that shows GeoNode's system uptime.
- **Alerts** shows number of notifications from defined checks. When clicked, Alerts box will show detailed information. See Notifications description for details.
- **Errors** - shows how many errors were captured during request processing. When clicked, Errors box will show detailed list of captured errors. See Errors description for details.



Indicators in error state

Software Performance

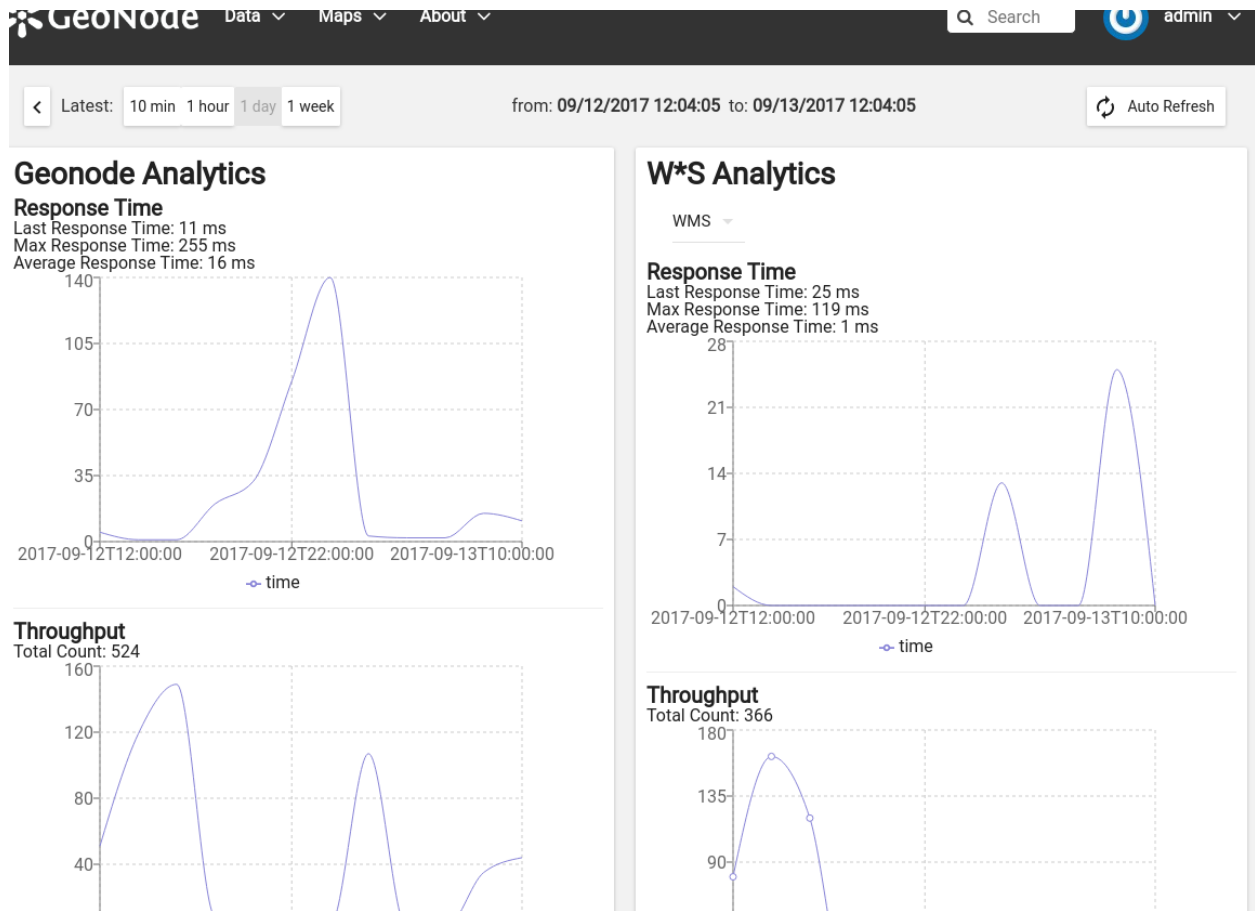
Software Performance view shows GeoServer web service statistics, for all requests monitored and detailed, OWS-specific, per service type (WMS, WFS, OCS etc).



Clicking on



will show charts with data history for overall performance and per-OWS performance:



Hardware Performance

Hardware performance box shows hardware usage statistics for selected host (monitored with any of hostgeonode or hostgeoserver type Service): % of CPU usage and average memory consumption. User can select from which host data will be presented.

017 12:30:16 to: 09/13/2017 12:40:16

 Auto Refresh

Alerts

0 Alerts to show

Errors

0 Errors occurred

Hardware Performance



Host

localhost

GeoNode HW Status

Average CPU
5%

Average Memory
1937 MB

Clicking on

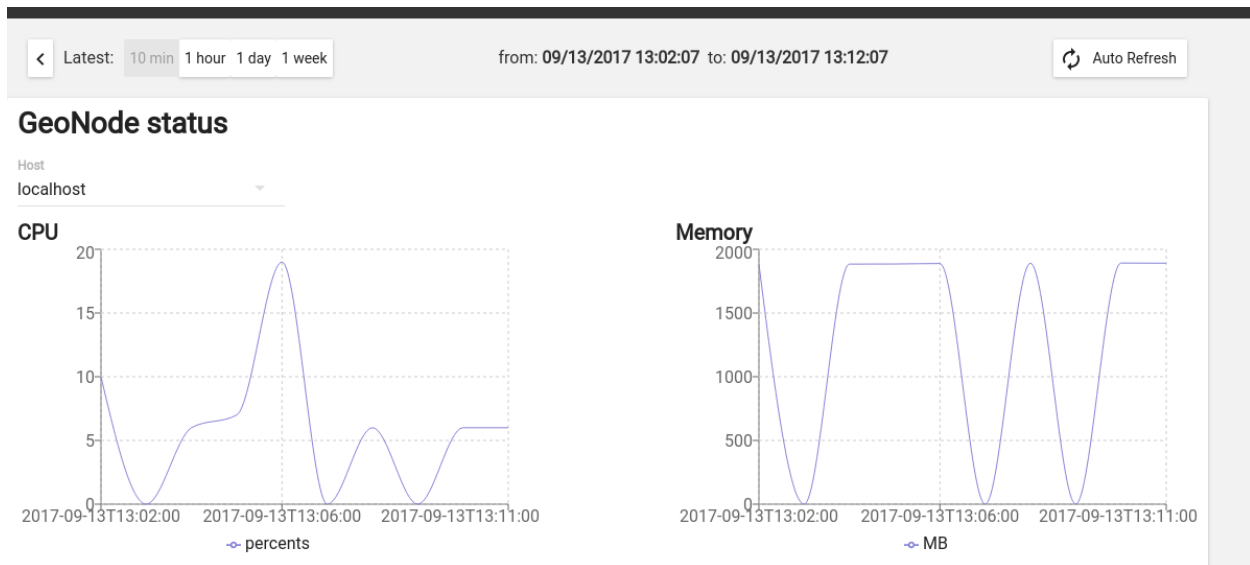
Hardware Performance



Host

localhost

will show charts with data history for selected host and time period



Errors

Errors view will show list of captured errors in GeoNode and GeoServer. List contents is displayed for selected time window.

GeoNode

Latest: 10 min 1 hour 1 day 1 week from: 09/05/2017 16:26:18 to: 09/12/2017 16:26:18 Auto Refresh

Errors

ID	Type	Service	Date
205	org.geoserver.platform.ServiceExcepti...	demo-geoserver	2017-09-08T23:58:03.823
206	org.geoserver.platform.ServiceExcepti...	demo-geoserver	2017-09-09T06:10:03.556
207	org.geoserver.platform.ServiceExcepti...	demo-geoserver	2017-09-10T04:10:03.870
208	org.geoserver.platform.ServiceExcepti...	demo-geoserver	2017-09-10T16:20:03.930
209	org.geoserver.platform.ServiceExcepti...	demo-geoserver	2017-09-11T08:16:08.702
210	org.geoserver.platform.ServiceExcepti...	demo-geoserver	2017-09-11T08:16:08.711
211	org.geoserver.platform.ServiceExcepti...	demo-geoserver	2017-09-11T08:16:08.774
212	org.geoserver.platform.ServiceExcepti...	demo-geoserver	2017-09-11T10:14:06.617
213	org.geoserver.platform.ServiceExcepti...	demo-geoserver	2017-09-11T10:14:06.729
214	django.template.base.TemplateSyntax...	local-geonode	2017-09-11T11:10:56.110
215	django.template.base.TemplateSyntax...	local-geonode	2017-09-11T11:11:06.116

For each error, details are available:

- error class, message and stack trace
- basic request context (IP, path, user agent)

<

Latest:

10 min

1 hour

1 day

1 week

from: 09/05/2017 16:26:50 to: 09/12/2017 16:26:50

↺

Auto Refresh

Error id: 206

METADATA	CLIENT
Date: 2017-09-09 06:10:03 Service: demo-geoserver Type: org.geoserver.platform.ServiceException Status code: 200 URL: /geosolutions/wms/reflect?format=application/atom+xml&layers=geosolutions:cleaned&featureid=cleaned.187	

Log

```
org.geoserver.wms.map.GetMapKvpRequestReader.parseLayers(GetMapKvpRequestReader.java:1357)
org.geoserver.wms.map.GetMapKvpRequestReader.read(GetMapKvpRequestReader.java:235)
org.geoserver.wms.map.GetMapKvpRequestReader.read(GetMapKvpRequestReader.java:85)
org.geoserver.ows.Dispatcher.parseRequestKVP(Dispatcher.java:1514)
org.geoserver.ows.Dispatcher.dispatch(Dispatcher.java:680)
org.geoserver.ows.Dispatcher.handleRequestInternal(Dispatcher.java:258)
org.springframework.web.servlet.mvc.AbstractController.handleRequest(AbstractController.java:147)
org.springframework.web.servlet.mvc.SimpleControllerHandlerAdapter.handle(SimpleControllerHandlerAdapter.java:50)
org.springframework.web.servlet.DispatcherServlet.doDispatch(DispatcherServlet.java:959)
org.springframework.web.servlet.DispatcherServlet.doService(DispatcherServlet.java:893)
org.springframework.web.servlet.FrameworkServlet.processRequest(FrameworkServlet.java:968)
org.springframework.web.servlet.FrameworkServlet.doGet(FrameworkServlet.java:859)
javax.servlet.http.HttpServlet.service(HttpServlet.java:621)
org.springframework.web.servlet.FrameworkServlet.service(FrameworkServlet.java:844)
javax.servlet.http.HttpServlet.service(HttpServlet.java:722)
org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:305)
org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:210)
org.geoserver.filters.ThreadLocalsCleanupFilter.doFilter(ThreadLocalsCleanupFilter.java:28)
org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:243)
org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:210)
org.geoserver.filters.SpringDelegatingFilter$Chain.doFilter(SpringDelegatingFilter.java:75)
org.geoserver.wms.animate.AnimatorFilter.doFilter(AnimatorFilter.java:71)
```

Alerts

An alert is a descriptive information on situation when observed metric contains values outside allowed range (for example, response time is above 30 seconds, or no requests were served within last 30 minutes). Alerts are generated by notifications mechanism described below.

Alerts view will show list of alerts for current moment (alerts that were generated in past are not displayed here):

GeoNode

Data ▾

Maps ▾

About ▾

Q

Search

⚙

admin ▾

<

Latest:

10 min

1 hour

1 day

1 week

from: 09/11/2017 13:56:39 to: 09/12/2017 13:56:39

↺

Auto Refresh

Alerts

⚙

Number of handled requests is lower than 800

←

Each alert contains more descriptive information what is wrong:

Notifications

Notification mechanism (not to be confused with notifications application in GeoNode) is a way to inform selected users about situations, where collected metric data would indicate a problem with deployment. Notifications are accessible from Alerts view:

There can be several notification configurations available.

Each notification configuration contains two main elements:

- list of email addressess which should be notified when alert is generated
- list of checks (at least one check must be in invalid state to generate alert)

GeoServer is not working ✓

detects when requests are not handled

Who to alert:

When to alert

- ☒ Number of handled requests is lower than 100
- ☒ No response for at least 86400
- ☒ Response time is higher than 500 s

User can add arbitrary number of emails. Email address doesn't need to point to user registered in GeoNode instance. If email provided doesn't belong to any of users, alert will be send as a regular email. If email provided can be associated with specific user, notifications application (and thus, notification settings for that user) will be used to send alert.

Integration with GeoHealthCheck

GeoNode can also be easily monitored with external tools, like [GeoHealthCheck](#). See [Documentation on adding resources](#) for details.

3.2.1.4.14 Use datastore shards in GeoNode

Many organizations have hundreds of layers uploaded to GeoNode. In such a case using the default GeoNode configuration, with just one PostGIS database and one GeoServer PostGIS store for all of the layers has several limitations, such as:

- **Layer upload and creation time become very large** Layer upload and creation time tends to become very large when the PostGIS database starts containing many layers. We have seen cases where even 4/5 minutes were needed for uploading a small sized shapefile). This issue is caused by the actual implementation of the PostgreSQL JDBC driver and has been reported in details in GeoServer bug GEOS-7533: <https://osgeo-org.atlassian.net/browse/GEOS-7533>
- **Large backups** When data are not edited in GeoNode, it is easier to backup smaller PostgreSQL databases rather than a big one, where only a few tables have been changed since the last backup. When using a single database for all of the uploads this is not possible

These problems can be tackled using the Datastore Shards GeoNode contrib module, which automatically creates new shards when some defined conditions changes.

How to use the `datastore_shards` module

As a first thing, add the module in the `INSTALLED_APPS` section of the settings file:

```
INSTALLED_APPS = (
    ...,
    'geonode.contrib.datastore_shards',
```

(continues on next page)

(continued from previous page)

```

    ...
)

```

Here is a typical extract of the settings that must be used for GeoNode to use the database shards module:

```

# SHARD DATABASES SETTINGS
# SHARD_STRATEGY may be yearly, monthly, layercount
SHARD_STRATEGY = 'layercount'
SHARD_LAYER_COUNT = 100
SHARD_PREFIX = 'wm_'
SHARD_SUFFIX = ''
DATASTORE_URL = 'postgis://user:password@localhost:5432/data'

```

Now synchronize the module models with the database:

```
python manage.py migrate datastore_shards
```

Now you are set and datastore shards will be used as soon as GeoNode is restarted. The `datastore_shards` application will automatically create a new shard whenever it is needed.

Note: The PostgreSQL ROLE which is used (user in `DATASTORE_URL`) must have `CREATEDB` option assigned in order to be able to create the PostgreSQL shards.

Database shards settings

SHARD_STRATEGY

This setting can currently be set to ‘yearly’, ‘monthly’, ‘layercount’:

- **yearly** a database shard is created and used each year. PostgreSQL database and GeoServer store name is in the form: `prefix_YYYY_suffix`
- **monthly** a database shard is created and used each month. PostgreSQL database and GeoServer store name is in the form: `prefix_YYYYMM_suffix`
- **layercount** a database shard is created when previous shard reaches a certain number of layers (which is set by the `SHARD_LAYER_COUNT` setting). PostgreSQL database and GeoServer store name is in the form: `prefix_01234_suffix`. 01234 is a progressive number starting from 0.

SHARD_PREFIX and SHARD_SUFFIX

When these settings are used, a prefix and a suffix is appended to the name of the shard.

For example, if `SHARD_PREFIX = ‘foo’` and `SHARD_SUFFIX = ‘bar’`, when using a `SHARD_STRATEGY` set to ‘yearly’, the shard for 2017 will be named `foo_2017_bar`.

SHARD_LAYER_COUNT

This setting is used when `SHARD_STRATEGY` is set to “layercount”, and it represents the maximum number a shard can contain before next shard is created and used.

Here is how it looks GeoServer when using a SHARD_STRATEGY set to “layercount” and SHARD_LAYER_COUNT set to 3:

Layers

Manage the layers being published by GeoServer

[+ Add a new layer](#)

[- Remove selected layers](#)

<< < 1 > >> Results 1 to 8 (out of 8 items)

<input type="checkbox"/>	Type	Title	Name	Store	Enabled	Native SRS
<input type="checkbox"/>	•	ukr_nhr_hotspots	geonode:ukr_nhr_hotspots	wm_00000	✓	EPSG:4326
<input type="checkbox"/>	•	ukr_nhr_hotspots_1	geonode:ukr_nhr_hotspots_1	wm_00000	✓	EPSG:4326
<input type="checkbox"/>	•	ukr_nhr_hotspots_2	geonode:ukr_nhr_hotspots_2	wm_00000	✓	EPSG:4326
<input type="checkbox"/>	•	ukr_nhr_hotspots_3	geonode:ukr_nhr_hotspots_3	wm_00001	✓	EPSG:4326
<input type="checkbox"/>	•	ukr_nhr_hotspots_4	geonode:ukr_nhr_hotspots_4	wm_00001	✓	EPSG:4326
<input type="checkbox"/>	•	ukr_nhr_hotspots_5	geonode:ukr_nhr_hotspots_5	wm_00001	✓	EPSG:4326
<input type="checkbox"/>	•	ukr_nhr_hotspots_6	geonode:ukr_nhr_hotspots_6	wm_00002	✓	EPSG:4326
<input type="checkbox"/>	•	ukr_nhr_hotspots_7	geonode:ukr_nhr_hotspots_7	wm_00002	✓	EPSG:4326

<< < 1 > >> Results 1 to 8 (out of 8 items)

As you can easily see a GeoServer PostGIS store is created every time the store contains three layers. Each store links to a different PostGIS database.

3.2.1.4.15 Asynchronous signals handling

Asynchronous signals handling is a part of GeoNode scalability architecture. It allows to offload post-processing tasks from main web application process, also to integrate closely with GeoServer.

In this setup GeoNode produces events (small messages with serialized configuration) that are queued by external broker (AMQP broker, preferably RabbitMQ), and later, are consumed (processed) by separate worker process. Additionally, GeoServer can be attached and used as producer as well, so two-way data synchronization between GeoNode and GeoServer can be established.

Note: Default configuration (in-memory queue) doesn't require AMQP broker, and will process signals synchronously. Also, default configuration won't allow two-way synchronization with GeoServer, even if GS is configured to work with external broker.

Async signals handling components

GeoNode

GeoNode can be switched to asynchronous signals processing by specifying `ASYNC_SIGNALS_BROKER_URL` setting in settings. See [GeoNode settings](#). This setting is consumed by [kombu library](#). You can set type of transport with connection details (credentials, host), for example: `amqp://localhost/` will connect to AMQP broker (RabbitMQ) running locally, without any authentication.

Note: while this is easiest to use asynchronous signals handling, it is not recommended setup due to security issues.

GeoServer (optional)

See [GeoServer documentation for Notification module](#).

RabbitMQ (AMQP broker)

Asynchronous signals infrastructure can be used with different protocols (supported by *Kombu* library), but AMQP with RabbitMQ broker is recommended.

Install and run RabbitMQ instance:

```
sudo apt-get install rabbitmq-server
sudo service rabbitmq-server start
```

Additional information about RabbitMQ configuration is available in [Documentation](#).

Consumer process

To handle events produced by GeoNode and GeoServer, a receiver, consumer process is needed. GeoNode provides *runmessaging* django command. This will start long-running process, which will handle incoming messages. For production deployments, it should be configured with process managers like supervisord or runit.

Following is sample configuration for supervisord:

```
[program:geonode-runmessaging]
command=/bin/bash -c "(source /path/to/virtualenv/bin/activate && django-admin.py_
↳runmessaging --autoscale 2,1 --loglevel DEBUG) "
directory = # /path/to/rundir
environment=DJANGO_SETTINGS_MODULE="geonode.local_settings"
user=geonode
numproc=1
stdout_logfile=/path/to/log/output.log
stderr_logfile=/path/to/log/output.log
autostart = true
autorestart = true
startsecs = 10
stopwaitsecs = 600
priority = 998
```

This file is also available in *scripts/misc/runmessaging.supervisord.conf* file in GeoNode repository.

To use it, you should have *supervisord* installed and configured:

```
sudo apt-get install supervisor
sudo supervisor start
vi /etc/supervisor/conf.d/runmessaging.supervisord.conf # adjust paths and variables.
sudo supervisorctl reload
```

Notes

Technical details and motivation is described in [GNIP](#).

3.2.1.4.16 GeoNode Social Accounts

Contents

- *GeoNode Social Accounts*
 - *Allow GeoNode to Login throguh Social Accounts (Facebook and Linkedin)*
 - * *Base concepts and objects*
 - * *Installation*
 - * *Configuration*
 - * *Usage*
 - *LinkedIn Application*
 - *Facebook Application*
 - * *Login by using Existing Accounts on GeoNode*

Allow GeoNode to Login throguh Social Accounts (Facebook and Linkedin)

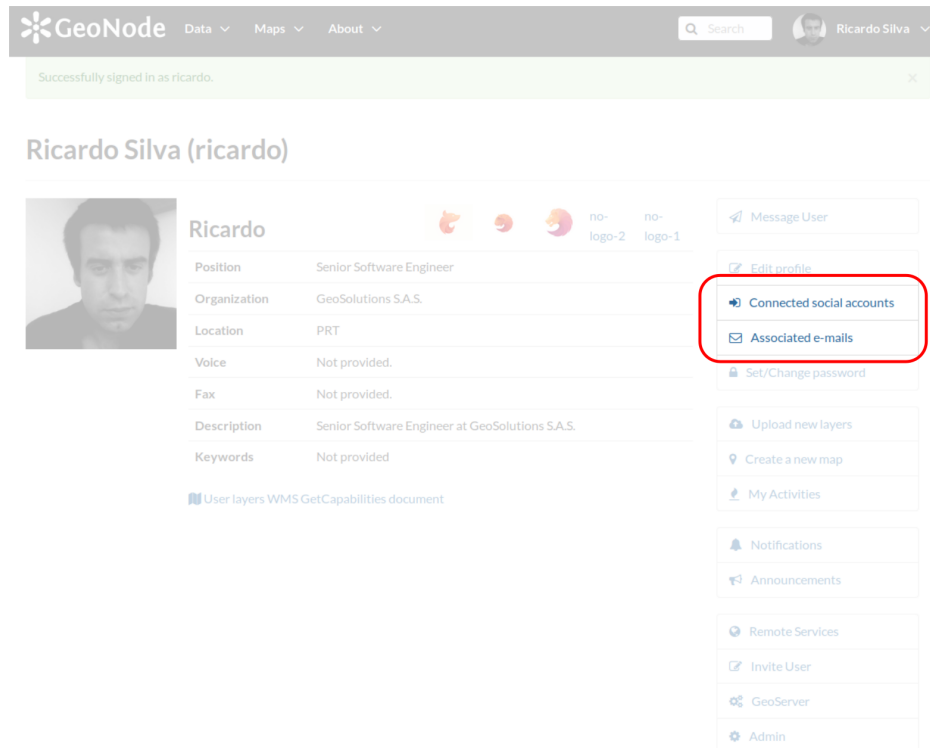
Base concepts and objects

In order to harmonize the various authentication flows between local accounts and remote social accounts, the whole user registration and authentication codebase has been refactored.

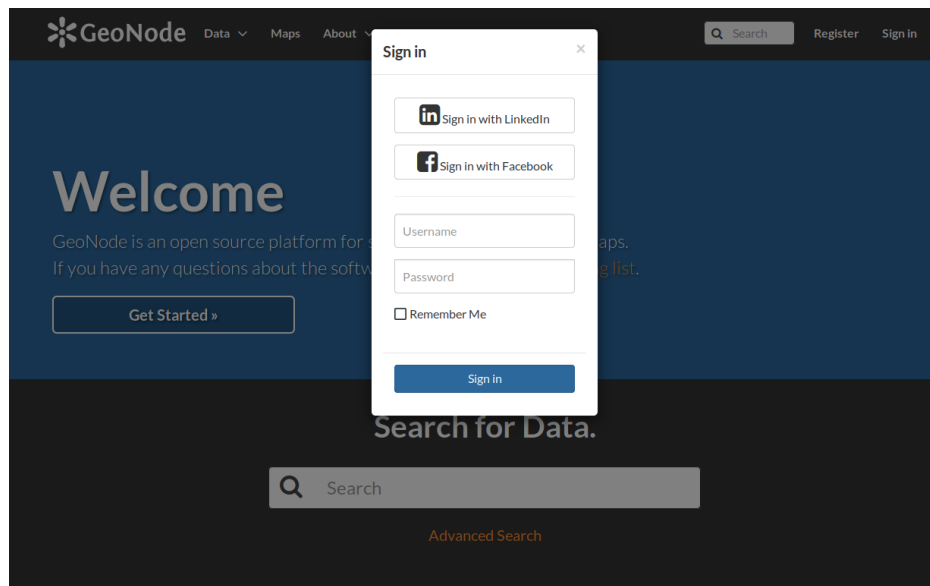
Major changes:

- [geonode-user-accounts](#) has been retired and is not used anymore. This app was only capable of managing local accounts;
- [django-allauth](#) has been integrated as a dependency of geonode. It provides a solution for managing both local and remote user accounts. It is now used for dealing with most user registration and auth flows;
- [django-invitations](#) has also been integrated as a dependency of geonode and is used for managing invitations to new users. This functionality was previously provided by [geonode-user-accounts](#);
- [django-allauth](#) has been extended in order to provide the following additional features:
 - Automatically registering an e-mail with a user when the e-mail is used to connect to a social account;
 - Automatically extract information from the user's social account and use that to enhance the user's profile fields on geonode. This was implemented in a pluggable way, allowing custom installs to configure it for other providers;
 - Allow approval of new registrations by staff members before allowing new users to login. This functionality was previously provided by [geonode-user-accounts](#).

- There are now extra sections on the user's profile to manage connected social accounts and e-mail accounts



- When properly configured, the login and register pages now display the possibility to login with social accounts



Installation

- Install the new `allauth` plugin and remove any of the old dependencies

```
pip install -r requirements.txt --upgrade
pip install -e . --upgrade --no-cache
pip uninstall geonode-user-accounts -y
pip uninstall django-user-accounts -y
```

- ensure sure the Django model is updated and the templates updated to the static folder

```
DJANGO_SETTINGS_MODULE=geonode.local_settings python -W ignore manage.py_
↳makemigrations
DJANGO_SETTINGS_MODULE=geonode.local_settings python -W ignore manage.py_
↳migrate
DJANGO_SETTINGS_MODULE=geonode.local_settings python -W ignore manage.py_
↳collectstatic --noinput
```

- ensure that Social Providers are enabled in your settings:

```
# prevent signing up by default
ACCOUNT_OPEN_SIGNUP = True
ACCOUNT_EMAIL_REQUIRED = True
ACCOUNT_EMAIL_VERIFICATION = 'optional'
ACCOUNT_EMAIL_CONFIRMATION_EMAIL = True
ACCOUNT_EMAIL_CONFIRMATION_REQUIRED = True
ACCOUNT_CONFIRM_EMAIL_ON_GET = True
ACCOUNT_APPROVAL_REQUIRED = True

SOCIALACCOUNT_ADAPTER = 'geonode.people.adapters.SocialAccountAdapter'

SOCIALACCOUNT_AUTO_SIGNUP = False

INSTALLED_APPS += (
    'allauth.socialaccount.providers.linkedin_oauth2',
    'allauth.socialaccount.providers.facebook',
)

SOCIALACCOUNT_PROVIDERS = {
    'linkedin_oauth2': {
        'SCOPE': [
            'r_emailaddress',
            'r_basicprofile',
        ],
        'PROFILE_FIELDS': [
            'emailAddress',
            'firstName',
            'headline',
            'id',
            'industry',
            'lastName',
            'pictureUrl',
            'positions',
            'publicProfileUrl',
            'location',
            'specialties',
            'summary',
        ],
    },
    'facebook': {
        'METHOD': 'oauth2',
```

(continues on next page)

(continued from previous page)

```

        'SCOPE': [
            'email',
            'public_profile',
        ],
        'FIELDS': [
            'id',
            'email',
            'name',
            'first_name',
            'last_name',
            'verified',
            'locale',
            'timezone',
            'link',
            'gender',
        ]
    },
}

# Comment out this in case you want to diable Social login
SOCIALACCOUNT_PROFILE_EXTRACTORS = {
    "facebook": "geonode.people.profileextractors.FacebookExtractor",
    "linkedin_oauth2": "geonode.people.profileextractors.
    ↪LinkedInExtractor",
}

```

Configuration

1. Go to GeoNode/Django Admin Dashboard and add the Social Apps you want to configure:

admin/socialaccount/socialapp/

Django administration

Home > Social Accounts > Social applications

Select social application to change

Action: 0 of 2 selected

<input type="checkbox"/>	Name	Provider
<input type="checkbox"/>	Facebook	Facebook
<input type="checkbox"/>	LinkedIn	LinkedIn

2 social applications

- LinkedIn

Change social application

Provider:	<div>LinkedIn ▾</div>
Name:	<div>LinkedIn</div>
Client id:	<div></div> <div>App ID, or consumer key</div>
Secret key:	<div></div> <div>API secret, client secret, or consumer secret</div>
Key:	<div></div> <div>Key</div>
Sites:	<div><div>Available sites ⓘ</div><div><div>Filter</div></div><div></div><div>Choose all ⓘ</div></div> <div><div>Chosen sites ⓘ</div><div></div><div>Remove all</div></div>

Hold down "Control", or "Command" on a Mac, to select more than one.

- Facebook

Change social application

Provider:	<div>Facebook ▾</div>
Name:	<div>Facebook</div>
Client id:	<div></div> <div>App ID, or consumer key</div>
Secret key:	<div></div> <div>API secret, client secret, or consumer secret</div>
Key:	<div></div> <div>Key</div>
Sites:	<div><div>Available sites ⓘ</div><div><div>Filter</div></div><div></div><div>Choose all ⓘ</div></div> <div><div>Chosen sites ⓘ</div><div></div><div>Remove all</div></div>

Hold down "Control", or "Command" on a Mac, to select more than one.

Warning: Make sure to add the sites you want to enable.

Usage

You need first to create and configure OAuth2 Applications on your Social Providers.

This will require a personal or business account, which can access to the developers sections of LinkedIn and Facebook and create and configure new Applications.

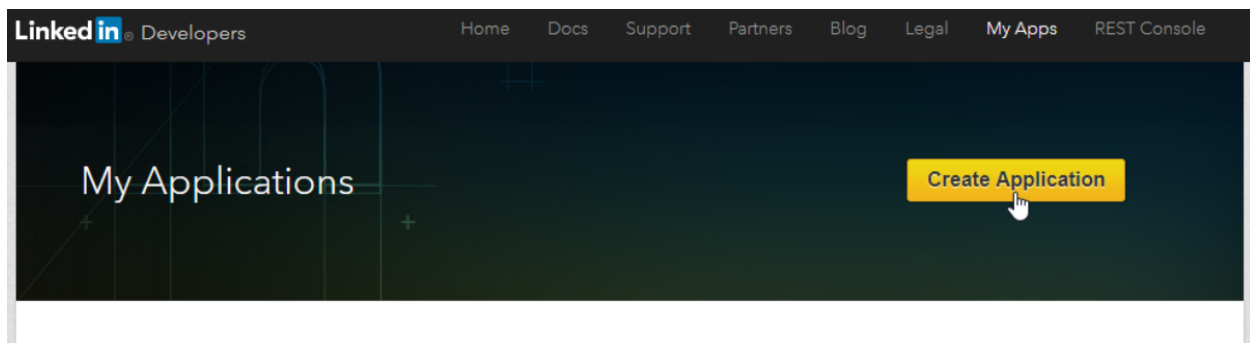
That account won't be visible to the GeoNode users. This is needed only to generate OAuth2 Client ID and Client Secret Authorization Keys.

In the following sections we will see in details how to configure them for both LinkedIn and Facebook.

LinkedIn Application

(ref.: <http://django-allauth.readthedocs.io/en/latest/providers.html>)

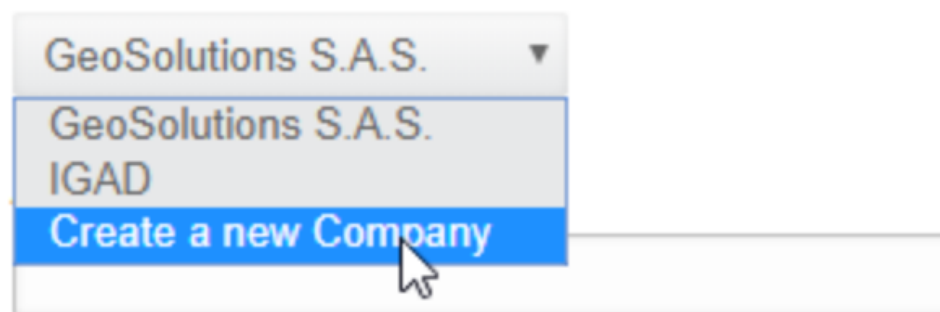
1. Go to <https://www.linkedin.com/developer/apps> and select Create Application



2. Create a new Company

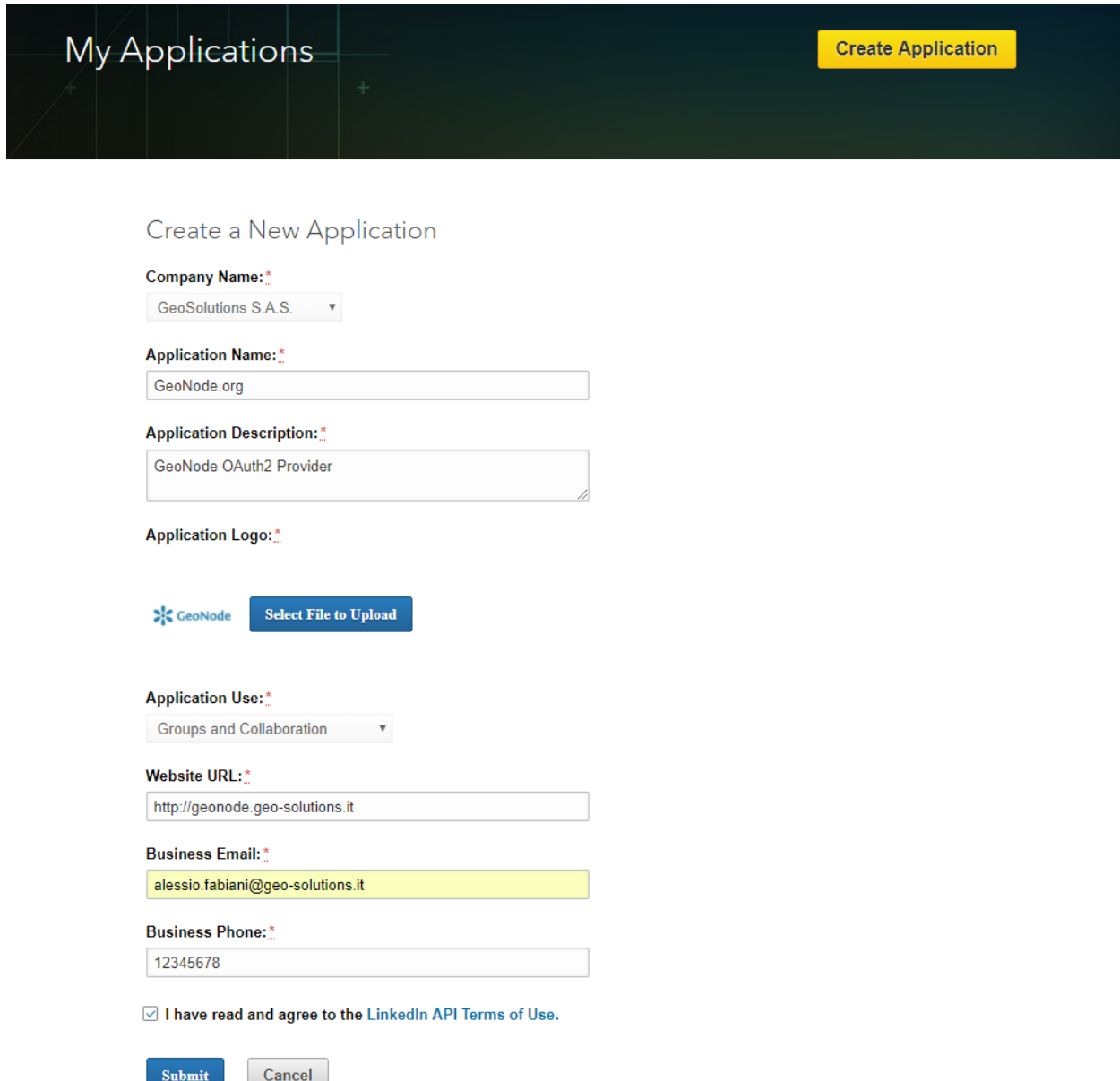
Create a New Application

Company Name: *



3. Fill the informations

Note: The logo must have precise square dimensions



The screenshot shows the 'My Applications' page header with a 'Create Application' button. Below is the 'Create a New Application' form with the following fields and values:

- Company Name:** GeoSolutions S.A.S.
- Application Name:** GeoNode.org
- Application Description:** GeoNode OAuth2 Provider
- Application Logo:** GeoNode logo and 'Select File to Upload' button
- Application Use:** Groups and Collaboration
- Website URL:** http://geonode.geo-solutions.it
- Business Email:** alessio.fabiani@geo-solutions.it
- Business Phone:** 12345678
- ☒ I have read and agree to the [LinkedIn API Terms of Use](#).
- Buttons:** Submit, Cancel

4. Select the following Default Application Permissions

Warning: Be sure to select the `r_basicprofile` and `r_emailaddress` application permissions.

Default Application Permissions

☒ **r_basicprofile**
☐ **w_share**
☒ **r_emailaddress**

5. Add OAuth 2.0 Authorized Redirect URLs:

```
http://geonode.geo-solutions.it/account/linkedin_oauth2/login/callback/
http://geonode.geo-solutions.it/account/linkedin/login/callback/
```

Default Application Permissions

☒ **r_basicprofile**
☒ **r_emailaddress**
☐ **rw_c**
☐ **w_share**

OAuth 2.0

Authorized Redirect URLs:

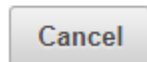
6. Save

<http://geonode.geo-solutions.it/account/linkedin/login>

OAuth 1.0a

Default "Accept" Redirect URL:

Default "Cancel" Redirect URL:



7. Take note of the Authentication Keys

Authentication Keys

Client ID: 

Client Secret: 

Default Application Permissions

8. Go to GeoNode/Django admin, Social Applications and select the LinkedIn one (/admin/socialaccount/socialapp/)

Django administration

Home > Social Accounts > Social applications

Select social application to change

Action: 0 of 2 selected

<input type="checkbox"/>	Name
<input type="checkbox"/>	Facebook
<input type="checkbox"/>	LinkedIn

2 social applications

9. Cut and Paste the Client ID and Client Secret on the related fields

Change social application

Provider:

Name:

Client id:

App ID, or consumer key

Secret key:

API secret, client secret, or consumer secret

10. Save

Facebook Application

(ref.: <http://django-allauth.readthedocs.io/en/latest/providers.html>) (ref.: <https://www.webforefront.com/django/setupdjangosocialauthentication.html>)

1. Go to <https://developers.facebook.com/apps> and Add a New Application



2. Create the App ID and go to the Dashboard

Crea un nuovo ID app

Inizia l'integrazione di Facebook nella tua app o nel sito Web

Nome visualizzato

IHP-WINS-DEV

Indirizzo e-mail di contatto

alessio.fabiani@gmail.com

Procedendo, accetti le [Normative della Piattaforma Facebook](#)

Annulla

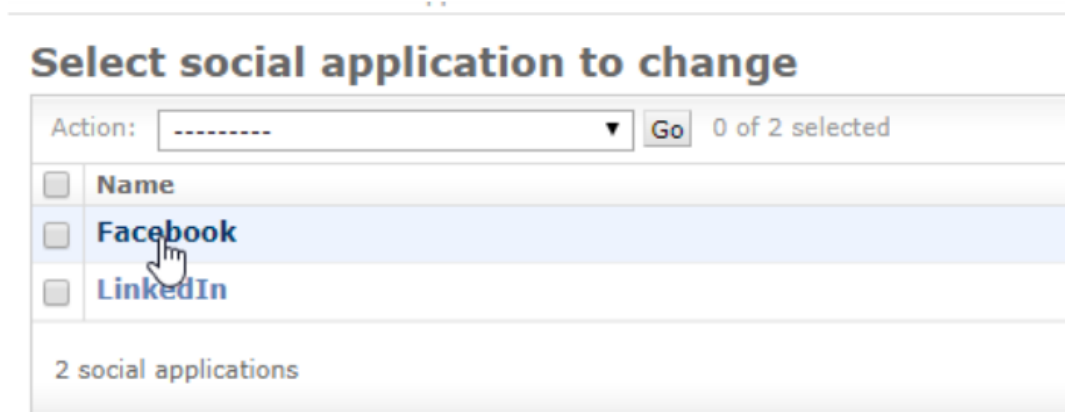
Crea ID app



3. Take note of the Authentication Keys



4. Go to GeoNode/Django admin, Social Applications and select the LinkedIn one
(/admin/socialaccount/socialapp/)



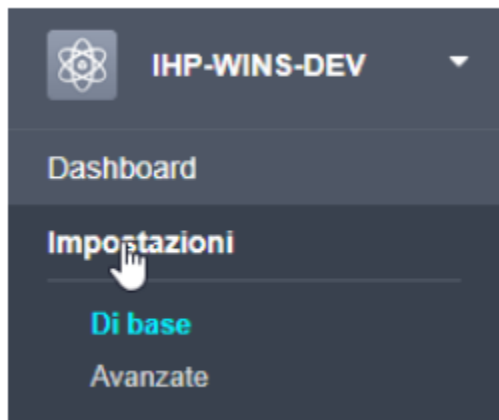
5. Cut and Paste the App ID and Secret Key on the related fields

ClientID	<-->	App Id
Client Secret	<-->	Secret Key

Change social application

Provider:	Facebook ▼
Name:	Facebook
Client id:	<div></div> App ID, or consumer key
Secret key:	<div></div> API secret, client secret, or consumer secret

6. Save
7. Go back to the Facebook Application Dashboard and select Settings



8. Add your App Domain

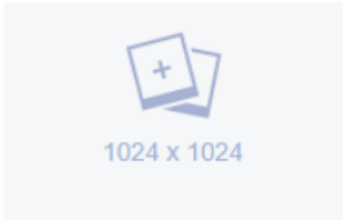
ID app

Nome visualizzato

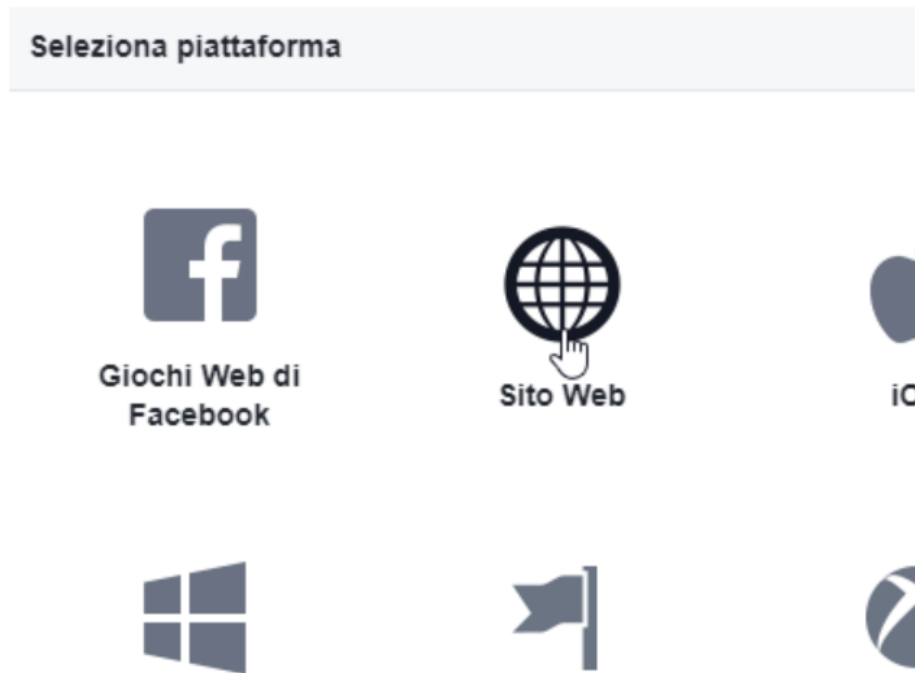
Domini app

Normativa sulla privacy per la finestra di dialogo di accesso e per

9. Click on Add Platform

Domini app	Indirizzo e-mail
<input type="text" value="http://igad-dev.geo-solutions.it/ x"/>	<input type="text" value="alessio.fabiani"/>
URL Normativa sulla privacy	URL delle Condizioni d'uso
<input type="text" value="Normativa sulla privacy per la finestra di dialogo di accesso e per i de"/>	<input type="text" value="Condizioni d'u"/>
Icona dell'app (1024 x 1024)	Categoria
	<input type="text" value="Scegli una cate"/>
	Ottieni maggiori informazioni
<input type="button" value="+ Aggiungi piattaforma"/>	

10. Select Web Site



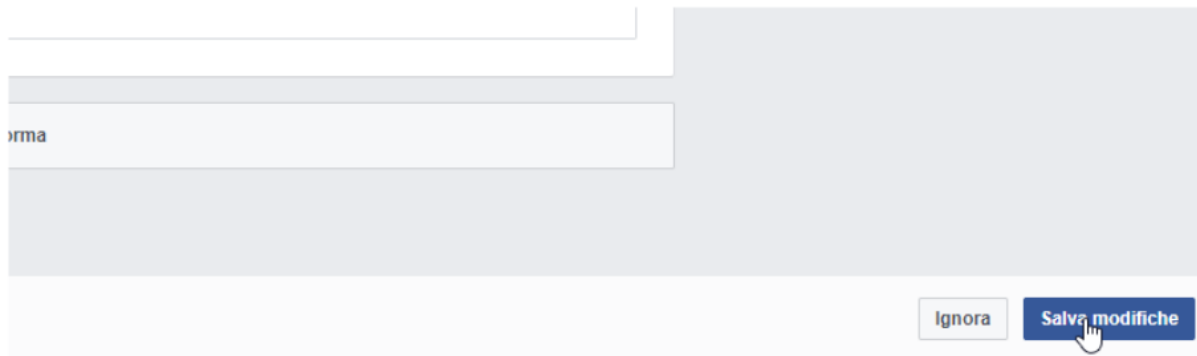
11. Add the URL

Sito Web

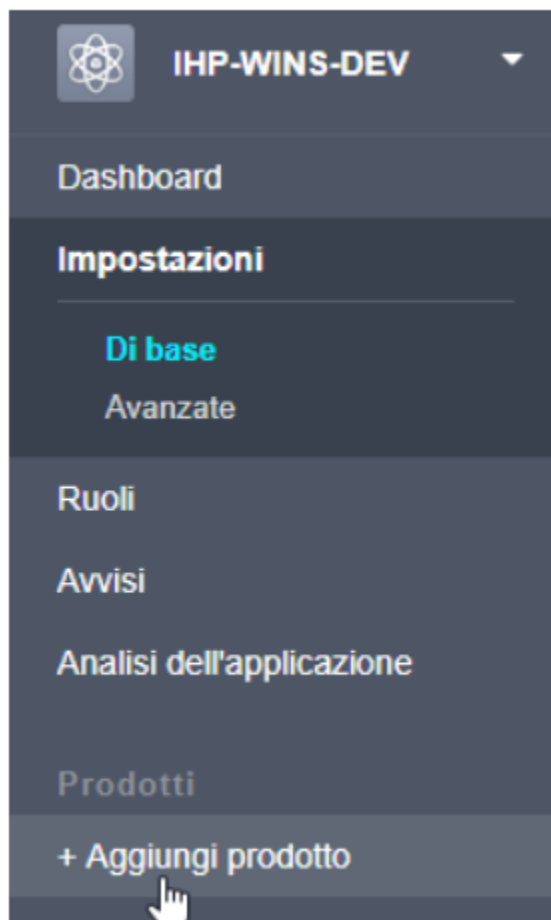
URL del sito

Save

12. And Save



13. Go to Add Product



14. Select Facebook Login

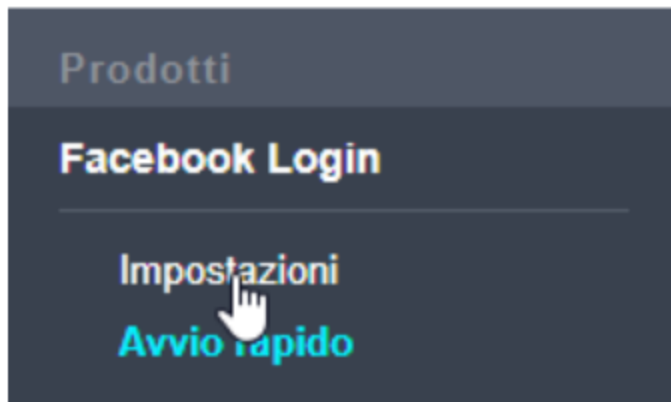


15. Select Web

questa app.



16. Go to Settings



17. Make sure Allow client OAuth and Access via OAuth Web are enabled

<input checked="" type="checkbox"/> Sì	Accesso client OAuth Abilita il flow standard del token client OAuth. Proteggi la tua applicazione con il controllo del browser per l'accesso al client.
<input checked="" type="checkbox"/> Sì	Accesso a OAuth Web Abilita l'accesso al client OAuth basato sul web. [?]
<input type="checkbox"/> No	Accesso OAuth al browser incorporato Abilita l'URI di reindirizzamento per il controllo del browser per l'accesso al client.

18. Add the valid redirect URIs:

```
http://geonode.geo-solutions.it/account/facebook/login/callback/
http://geondoe.geo-solutions.it/account/login/
```

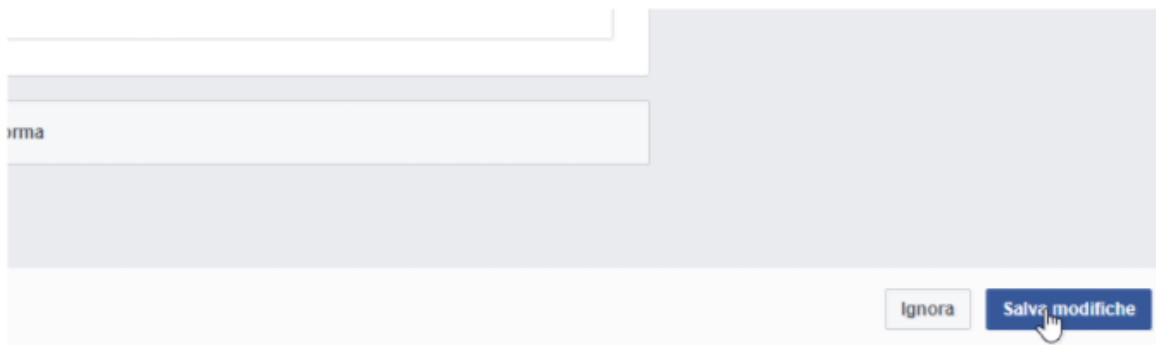
URI di reindirizzamento OAuth validi

http://hp-wins-dev.geo-solutions.it/account/facebook/login/callback/ ×

http://hp-wins-dev.geo-solutions.it/account/login/ ×

<input type="checkbox"/> No	Accesso dai dispositivi Abilita il flusso di accesso del client OAuth per dispositivi come smart TV [?]
------------------------------------	---

19. Save

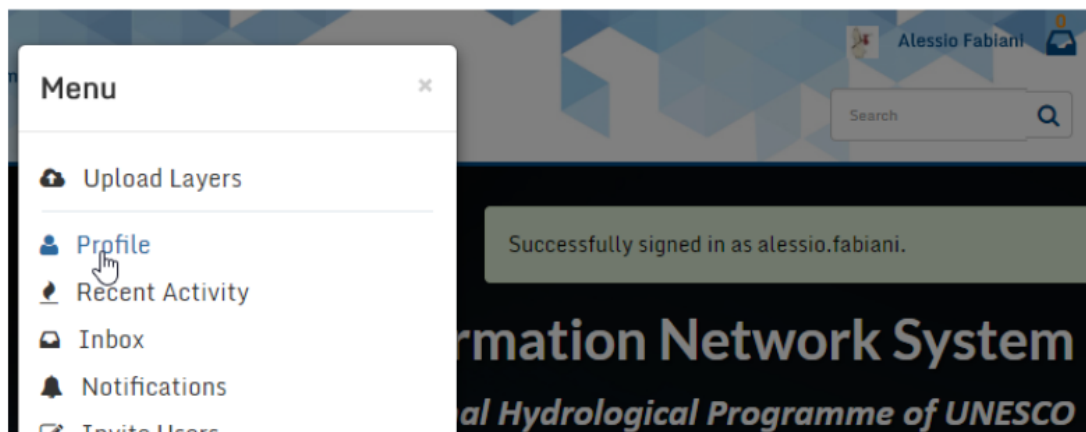


Login by using Existing Accounts on GeoNode

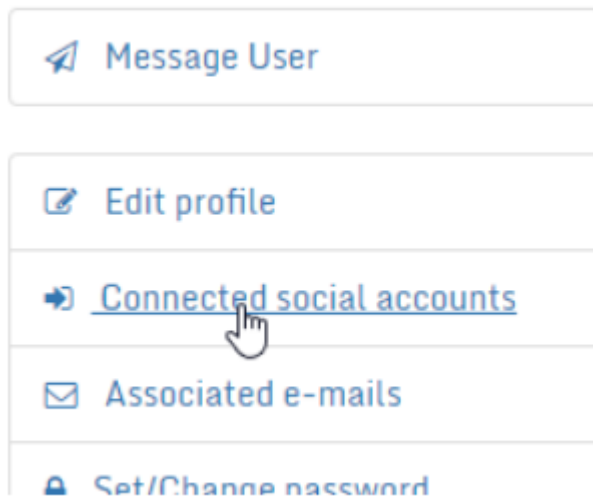
If you want to enable an already existing user account to login through social apps, you need to associate it to social accounts.

Usually this could be done only by the current user, since this operation requires authentication on its social accounts.

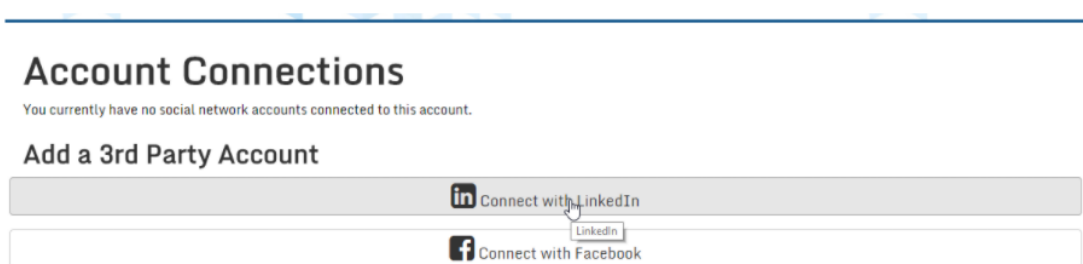
In order to do that you need to go to the User Profile Settings



Click on “Connected social accounts”





And actually connect them



Account Connections

You can sign in to your account using any of the following already connected third party accounts:

- ☐  Facebook account: Alessio Fabiani
- ☐  LinkedIn account: Alessio Fabiani

[Remove](#)

GeoNode and GeoServer Advanced Security GeoNode interacts with GeoServer through an advanced security mechanism based on OAuth2 Protocol and GeoFence. This section is a walk through of the configuration and setup of GeoNode and GeoServer Advanced Security.

Usage of the GeoNode's Django Administration Panel GeoNode has an administration panel based on the Django admin which can be used to do some database operations. Although most of the operations can and should be done through the normal GeoNode interface, the admin panel provides a quick overview and management tool over the database.

Management Commands for GeoNode GeoNode comes with administrative commands to help with day to day tasks. This section shows the list of the ones that come from the GeoNode application.

Configuring Alternate CSW Backends `pycsw` is the default CSW server implementation provided with GeoNode. This section will explain how to configure GeoNode to operate against alternate CSW server implementations.

LDAP configuration This module will allow you to add LDAP authentication to your GeoNode instance.

Customize the look and feel You might want to change the look of GeoNode, editing the colors and the logo of the website and adjust the templates for your needs. To do so, you first have to set up your own geonode project from a template. If you've successfully done this, you can go further and start theming your geonode project.

Debugging GeoNode Installations There are several mechanisms to debug GeoNode installations, the most common ones are discussed in this section.

Changing the Default Language GeoNode's default language is English, but GeoNode users can change the interface language with the pulldown menu at the top-right of most GeoNode pages. Once a user selects a language GeoNode remembers that language for subsequent pages.

More on Security and Permissions This tutorial will guide you through the steps that can be done in order to restrict the access on your data uploaded to geonode.

First of all it will be shown how a user can be created and what permissions he can have. Secondly we will take a closer look on to layers, maps and documents and the different opportunities you have in order to ban certain users from viewing or editing your data.

Loading Data into a GeoNode This module will walk you through the various options available to load data into your GeoNode from GeoServer, on the command-line or programmatically. You can choose from among these techniques depending on what kind of data you have and how you have your geonode setup.

Implementing S3 Bucket for Static and Media Files This section will show you how to configure your GeoNode instance to utilize an Amazon S3 Bucket for your site's static and media files.

Backup & Restore GeoNode - Data Migration How to perform a full backup / restore of GeoNode and GeoServer catalogs and how to migrate data. Customization backup / restore fixtures and data manglers.

GeoNode Monitoring How to set up and use internal monitoring application.

Use datastore shards in GeoNode How to use the PostGIS datastore shards contrib application in order to have GeoServer stores containing a fewer number of layers. Useful for organizations with hundreds of layers.

Asynchronous signals handling How to set up asynchronous signals handling.

GeoNode Social Accounts How to set up and login through LinkedIn and Facebook Social Accounts.

3.2.1.5 Developers Workshop

Welcome to the GeoNode Training *Developers Workshop* documentation v2.8.

This workshop will teach how to develop with and for the **GeoNode** software application. This module will introduce you to the components that GeoNode is built with, the standards that it supports and the services it provides based on those standards, and an overview its architecture.

Prerequisites GeoNode is a web based GIS tool, and as such, in order to do development on GeoNode itself or to integrate it into your own application, you should be familiar with basic web development concepts as well as with general GIS concepts.

3.2.1.5.1 Introduction to GeoNode development

This module will introduce you to the components that GeoNode is built with, the standards that it supports and the services it provides based on those standards, and an overview its architecture.

GeoNode is a web based GIS tool, and as such, in order to do development on GeoNode itself or to integrate it into your own application, you should be familiar with basic web development concepts as well as with general GIS concepts.

A set of reference links on these topics is included at the end of this module.

Standards

GeoNode is based on a set of Open Geospatial Consortium (OGC) standards. These standards enable GeoNode installations to be interoperable with a wide variety of tools that support these OGC standards and enable federation with other OGC compliant services and infrastructure. Reference links about these standards are also included at the end of this module.

GeoNode is also based on Web Standards ...

Open Geospatial Consortium (OGC) Standards

Web Map Service (WMS)

The Web Map Service (WMS) specification defines an interface for requesting rendered map images across the web. It is used within GeoNode to display maps in the pages of the site and in the GeoExplorer application to display rendered layers based on default or custom styles.

Web Feature Service (WFS)

The Web Feature Service (WFS) specification defines an interface for reading and writing geographic features across the web. It is used within GeoNode to enable downloading of vector layers in various formats and within GeoExplorer to enable editing of Vector Layers that are stored in a GeoNode.

Web Coverage Service (WCS)

The Web Coverage Service (WCS) specification defines an interface for reading and writing geospatial raster data as “coverages” across the web. It is used within GeoNode to enable downloading of raster layers in various formats.

Catalogue Service for Web (CSW)

The Catalogue Service for Web (CSW) specification defines an interface for exposing a catalogue of geospatial metadata across the web. It is used within GeoNode to enable any application to search GeoNode’s catalogue or to provide federated search that includes a set of GeoNode layers within another application.

Tile Mapping Service (TMS/WMTS)

The Tile Mapping Service (TMS) specification defines an interface for retrieving rendered map tiles over the web. It is used within GeoNode to enable serving of a cache of rendered layers to be included in GeoNode’s web pages or within the GeoExplorer mapping application. Its purpose is to improve performance on the client vs asking the WMS for rendered images directly.

Web Standards

HTML

HyperText Markup Language, commonly referred to as **HTML**, is the standard markup language used to create web pages.¹ Web browsers can read HTML files and render them into visible or audible web pages. HTML describes the structure of a website semantically along with cues for presentation, making it a markup language, rather than a programming language.

HTML elements form the building blocks of all websites. HTML allows images and objects to be embedded and can be used to create interactive forms. It provides a means to create structured documents by denoting structural semantics for text such as headings, paragraphs, lists, links, quotes and other items.

The language is written in the form of HTML elements consisting of *tags* enclosed in angle brackets (like `< >`). Browsers do not display the HTML tags and scripts, but use them to interpret the content of the page.

HTML can embed scripts written in languages such as JavaScript which affect the behavior of HTML web pages. Web browsers can also refer to Cascading Style Sheets (CSS) to define the look and layout of text and other material. The World Wide Web Consortium (W3C), maintainer of both the HTML and the CSS standards, has encouraged the use of CSS over explicit presentational HTML since 1997.

CSS

Cascading Style Sheets (CSS) is a style sheet language used for describing the presentation of a document written in a markup language.² Although most often used to set the visual style of web pages and user interfaces written in HTML and XHTML, the language can be applied to any XML document, including plain XML, SVG and XUL, and is applicable to rendering in speech, or on other media. Along with HTML and JavaScript, CSS is a cornerstone technology used by most websites to create visually engaging webpages, user interfaces for web applications, and user interfaces for many mobile applications.³

CSS is designed primarily to enable the separation of document content from document presentation, including aspects such as the layout, colors, and fonts.⁴ This separation can improve content accessibility, provide more flexibility and control in the specification of presentation characteristics, enable multiple HTML pages to share formatting by specifying the relevant CSS in a separate .css file, and reduce complexity and repetition in the structural content, such as semantically insignificant tables that were widely used to format pages before consistent CSS rendering was available in all major browsers. CSS makes it possible to separate presentation instructions from the HTML content in a separate file or style section of the HTML file. For each matching HTML element, it provides a list of formatting instructions. For example, a CSS rule might specify that “all heading 1 elements should be bold”, leaving pure semantic HTML markup that asserts “this text is a level 1 heading” without formatting code such as a `<bold>` tag indicating how such text should be displayed.

This separation of formatting and content makes it possible to present the same markup page in different styles for different rendering methods, such as on-screen, in print, by voice (when read out by a speech-based browser or screen reader) and on Braille-based, tactile devices. It can also be used to display the web page differently depending on the screen size or device on which it is being viewed. Although

¹ Hypertext Markup Language | Definition of hypertext markup language by Merriam-Webster

² “CSS developer guide”. Mozilla Developer Network. Retrieved 2015-09-24

³ “Web-based Mobile Apps of the Future Using HTML 5, CSS and JavaScript”. HTMLGoodies. Retrieved October 2014.

⁴ “What is CSS?”. World Wide Web Consortium. Retrieved December 2010.

REST

In computing, **Representational State Transfer (REST)** is the software architectural style of the World Wide Web.⁵⁶⁷ REST gives a coordinated set of constraints to the design of components in a distributed hypermedia system that can lead to a higher performing and more maintainable architecture.

To the extent that systems conform to the constraints of REST they can be called RESTful. RESTful systems typically, but not always, communicate over the Hypertext Transfer Protocol with the same HTTP verbs (GET, POST, PUT, DELETE, etc.) which web browsers use to retrieve web pages and to send data to remote servers.⁸ REST interfaces usually involve collections of resources with identifiers, for example /people/tom, which can be operated upon using standard verbs, such as DELETE /people/tom.

Exercises

Components and Services

Note: Hint, if `bash-completion` is installed, try `<TAB><TAB>` to get completions.

1. Start/stop services

```
$ sudo service apache2
$ sudo service apache2 reload
$ sudo service tomcat7
$ sudo service postgresql
```

2. Basic psql interactions

```
$ sudo su - postgres
$ psql
=> help                # get help
=> \?                  # psql specific commands
=> \l                   # list databases
=> \c geonode           # switch database
=> \ds                  # list tables
=> \ds layers_layer     # describe table
```

OGC Standards

WMS

1. Use the layer preview functionality in GeoServer to bring up a web map.

⁵ Fielding, R. T.; Taylor, R. N. (2000). "Principled design of the modern Web architecture". pp. 407–416. doi:10.1145/337180.337228.

⁶ Richardson, Leonard; Sam Ruby (2007), *RESTful web service*, O'Reilly Media, ISBN 978-0-596-52926-0, retrieved 18 January 2011, The main topic of this book is the web service architectures which can be considered RESTful: those which get a good score when judged on the criteria set forth in Roy Fielding's dissertation."

⁷ Richardson, Leonard; Mike Amundsen (2013), *RESTful web APIs*, O'Reilly Media, ISBN 978-1-449-35806-8, retrieved 15 September 2015, The Fielding dissertation explains the decisions behind the design of the Web."

⁸ Fielding, Roy Thomas (2000). "Chapter 5: Representational State Transfer (REST)". *Architectural Styles and the Design of Network-based Software Architectures* (Ph.D.). University of California, Irvine. This chapter introduced the Representational State Transfer (REST) architectural style for distributed hypermedia systems. REST provides a set of architectural constraints that, when applied as a whole, emphasizes scalability of component interactions, generality of interfaces, independent deployment of components, and intermediary components to reduce interaction latency, enforce security, and encapsulate legacy systems."

2. Copy the URL for the image in the map.
3. Alter URL parameters for the request.
4. Use *curl* to get the capabilities document

```
$ curl 'http://localhost/geoserver/wms?request=getcapabilities'
```

More: <http://docs.geoserver.org/stable/en/user/services/wms/index.html>

WFS

1. Describe a feature type using curl (replace ws:name with your layer)

```
$ curl 'http://localhost/geoserver/wfs?request=describefeaturetype&
↪name=ws:name'
```

More: <http://docs.geoserver.org/stable/en/user/services/wfs/reference.html>

Development References

Basic Web based GIS Concepts and Background

- OGC Services
 - <http://www.opengeospatial.org/>
 - https://en.wikipedia.org/wiki/Open_Geospatial_Consortium
- Web Application Architecture
 - https://en.wikipedia.org/wiki/Web_application
 - <http://www.w3.org/2001/tag/2010/05/WebApps.html>
 - <http://www.amazon.com/Web-Application-Architecture-Principles-Protocols/dp/047051860X>
- AJAX and REST
 - [https://en.wikipedia.org/wiki/Ajax_\(programming\)](https://en.wikipedia.org/wiki/Ajax_(programming))
 - https://en.wikipedia.org/wiki/Representational_state_transfer
- OpenGeo Suite
 - <http://workshops.boundlessgeo.com/suiteintro/>
 - <http://suite.opengeo.org/opengeo-docs/>
- GeoServer Administration
 - <http://suite.opengeo.org/opengeo-docs/geoserver/>
 - <http://suite.opengeo.org/docs/sysadmin/index.html#sysadmin>
- PostgreSQL and PostGIS Administration - <http://workshops.boundlessgeo.com/postgis-intro/> - <http://workshops.boundlessgeo.com/postgis-spatialdbtips/>

Core development tools and libraries

- Python
 - <https://docs.python.org/2/tutorial/>
 - <http://www.learnpython.org/>
 - <https://learnpythonthehardway.org/book/>
 - <http://www.guru99.com/python-tutorials.html>
- Django
 - <https://docs.djangoproject.com/en/dev/intro/tutorial01/>
 - <https://code.djangoproject.com/wiki/Tutorials>
- Javascript
 - <http://www.crockford.com/javascript/inheritance.html>
 - <http://geoext.org/v1/tutorials/quickstart.html>
- jQuery
 - <http://www.w3schools.com/jquery/default.asp>
 - <http://learn.jquery.com/using-jquery-core/>
 - <http://www.jquery-tutorial.net/>
- Bootstrap
 - <http://getbootstrap.com/>
 - <http://www.w3resource.com/twitter-bootstrap/tutorial.php>
- GeoTools/GeoScript/GeoServer
 - <http://docs.geotools.org/stable/tutorials/feature/csv2shp.html>
 - <http://geoscript.org/tutorials/index.html>
 - <http://docs.geotools.org/stable/tutorials/>
 - <https://github.com/boundlessgeo/gsconfig/blob/master/README.rst>
- geopython
 - <http://pycsw.org/docs/>
 - <http://geopython.github.io/OWSLib/>
 - <https://github.com/toblerity/shapely>
 - <https://github.com/sgillies/Fiona>
 - <https://pypi.python.org/pypi/pyproj>
- GDAL/OGR
 - http://www.gdal.org/gdal_utilities.html
 - http://www.gdal.org/ogr_utilities.html

3.2.1.5.2 Django Overview

This section introduces some basic concepts of Django, the Python based web framework on top of which GeoNode has been developed.

The main objective of Django is to facilitate the creation of complex sites oriented databases. Django emphasizes reusability and “pluggability” of components, rapid development, and the principle of not repeating yourself. Python is used everywhere, even for settings, files, and data models.

Django also provides an administrative interface to create, read, update and delete models that is dynamically generated by introspection and configured through the Administrative Templates.

Warning: Some parts of this section have been taken from the [Django](#) project and training documentation.

Getting Started With Django

Object-relational mapper

Data models can be defined entirely in Python. Django makes available a rich, dynamic database-access API for free, but it is still possible to write SQL if needed.

Hint: The following documentation is based on official [documentation](#) of the project Django.

Note: The following examples are taken from the official Django documentation for the sole purpose of introducing the general concepts.

```
1 class Band(models.Model):
2     """A model of a rock band."""
3     name = models.CharField(max_length=200)
4     can_rock = models.BooleanField(default=True)
5
6
7 class Member(models.Model):
8     """A model of a rock band member."""
9     name = models.CharField("Member's name", max_length=200)
10    instrument = models.CharField(choices=(
11        ('g', "Guitar"),
12        ('b', "Bass"),
13        ('d', "Drums"),
14    ),
15        max_length=1
16    )
17    band = models.ForeignKey("Band")
```

Models

A model is a Python class containing the essential fields and behaviors of the data stored on the DB. Generally, each model maps to a single database table.

- Each model is a Python class that subclasses `django.db.models.Model`.
- Each attribute of the model represents a database field.
- A model is an automatically-generated database-access API; see [Making queries](#).

Quick example

Note: The following examples are taken from the official Django documentation for the sole purpose of introducing the general concepts.

This example model defines a `Person`, which has a **first_name** and **last_name**:

```
1 from django.db import models
2
3 class Person(models.Model):
4     first_name = models.CharField(max_length=30)
5     last_name = models.CharField(max_length=30)
```

first_name and **last_name** are fields of the model. Each field is specified as a class attribute, and each attribute maps to a database column.

The above `Person` model would create a database table like this:

```
1 CREATE TABLE myapp_person (
2     "id" serial NOT NULL PRIMARY KEY,
3     "first_name" varchar(30) NOT NULL,
4     "last_name" varchar(30) NOT NULL
5 );
```

Some technical notes:

- The name of the table, **myapp_person**, is automatically derived from some model metadata but can be overridden.
- An **id** field is added automatically, but this behavior can be overridden.
- The **CREATE TABLE** SQL in this example is formatted using PostgreSQL syntax, but it's worth noting Django uses SQL tailored to the database backend specified in the settings file.

Using models

Once models have been defined, Django must be instructed on how to use those models. This is possible by editing the Django settings file and changing the **INSTALLED_APPS** setting to add the name of the module that contains the **model class**.

For example, if the models for the application is defined in the module **myapp.models**, **INSTALLED_APPS** should read, in part:

```
1 INSTALLED_APPS = (
2     #...
3     'myapp',
4     #...
5 )
```

Warning: When you add new apps to **INSTALLED_APPS**, be sure to run **manage.py migrate**, optionally making migrations for them first with **manage.py makemigrations**.

Note: GeoNode uses the specific command **manage.py migrate** to perform the models update and migration.

Fields

The list of DB fields is reflected (and specified) by the model class attributes.

Warning: Be careful not to choose field names that conflict with the models API like **clean**, **save**, or **delete**.

Example:

Note: The following examples are taken from the official Django documentation for the sole purpose of introducing the general concepts.

```
1 from django.db import models
2
3 class Musician(models.Model):
4     first_name = models.CharField(max_length=50)
5     last_name = models.CharField(max_length=50)
6     instrument = models.CharField(max_length=100)
7
8 class Album(models.Model):
9     artist = models.ForeignKey(Musician)
10    name = models.CharField(max_length=100)
11    release_date = models.DateField()
12    num_stars = models.IntegerField()
```

More: [Field Types](#)

Model methods

Custom methods on a model can be used to add custom “row-level” functionality to an object. This is a valuable technique for keeping business logic in one place.

For example, the following model has a few **custom methods**:

Note: The following examples are taken from the official Django documentation for the sole purpose of introducing the general concepts.

```
1 from django.db import models
2
3 class Person(models.Model):
4     first_name = models.CharField(max_length=50)
5     last_name = models.CharField(max_length=50)
```

(continues on next page)

(continued from previous page)

```

6     birth_date = models.DateField()
7
8     def baby_boomer_status(self):
9         "Returns the person's baby-boomer status."
10        import datetime
11        if self.birth_date < datetime.date(1945, 8, 1):
12            return "Pre-boomer"
13        elif self.birth_date < datetime.date(1965, 1, 1):
14            return "Baby boomer"
15        else:
16            return "Post-boomer"
17
18    def _get_full_name(self):
19        "Returns the person's full name."
20        return '%s %s' % (self.first_name, self.last_name)
21    full_name = property(_get_full_name)

```

The last method in this example is a `property`.

The `model` instance reference has a complete list of methods automatically given to each model. It is possible to override most of these; see [overriding predefined model methods](#)

More: [Models Methods](#)

Making queries

Django automatically gives a database-abstraction API that allows to create, retrieve, update and delete objects.

As an example:

Note: The following examples are taken from the official Django documentation for the sole purpose of introducing the general concepts.

```

1  from django.db import models
2
3  class Blog(models.Model):
4      name = models.CharField(max_length=100)
5      tagline = models.TextField()
6
7      def __str__(self):                # __unicode__ on Python 2
8          return self.name
9
10 class Author(models.Model):
11     name = models.CharField(max_length=50)
12     email = models.EmailField()
13
14     def __str__(self):                # __unicode__ on Python 2
15         return self.name
16
17 class Entry(models.Model):
18     blog = models.ForeignKey(Blog)
19     headline = models.CharField(max_length=255)
20     body_text = models.TextField()
21     pub_date = models.DateField()

```

(continues on next page)

(continued from previous page)

```
22     mod_date = models.DateField()
23     authors = models.ManyToManyField(Author)
24     n_comments = models.IntegerField()
25     n_pingbacks = models.IntegerField()
26     rating = models.IntegerField()
27
28     def __str__(self):                # __unicode__ on Python 2
29         return self.headline
```

Creating objects

As already said before, a model class represents a database table, and an instance of that class represents a particular record in the database table.

To create an object, instantiate it using keyword arguments to the model class, then call **save()** to save it to the database.

Assuming models live in a file **mysite/blog/models.py**, here's an example:

Note: The following examples are taken from the official Django documentation for the sole purpose of introducing the general concepts.

```
1 >>> from blog.models import Blog
2 >>> b = Blog(name='Beatles Blog', tagline='All the latest Beatles news.')
3 >>> b.save()
```

This performs an **INSERT SQL** statement behind the scenes. Django doesn't hit the database until you explicitly call **save()**. The **save()** method has no return value.

```
1 >>> b5.name = 'New name'
2 >>> b5.save()
```

This performs an **UPDATE SQL** statement behind the scenes. Django doesn't hit the database until you explicitly call **save()**.

Retrieving objects

Retrieving objects from the database can be done by constructing a **QuerySet** via a **Manager** on the model class.

A **QuerySet** represents a collection of objects from the database. It can have zero, one or many filters. Filters narrow down the query results based on the given parameters. In SQL terms, a **QuerySet** equates to a **SELECT** statement, and a filter is a limiting clause such as **WHERE** or **LIMIT**.

Each model has at least one **Manager**, and it's called **objects** by default.

It can be accessed directly via the model class, like so:

Note: The following examples are taken from the official Django documentation for the sole purpose of introducing the general concepts.

```

1 >>> Blog.objects
2 <django.db.models.manager.Manager object at ...>
3 >>> b = Blog(name='Foo', tagline='Bar')
4 >>> b.objects
5 Traceback:
6 ...
7 AttributeError: "Manager isn't accessible via Blog instances."
8 Note

```

Managers are accessible only via model classes, rather than from model instances, to *enforce a separation* between “table-level” operations and “record-level” operations. The **Manager** is the main source of **QuerySets** for a model. For example, `Blog.objects.all()` returns a **QuerySet** that contains all Blog objects in the database.

The simplest way to retrieve objects from a table is to get all of them. To do this, use the `all()` method on a **Manager**:

Note: The following examples are taken from the official Django documentation for the sole purpose of introducing the general concepts.

```

1 >>> all_entries = Entry.objects.all()

```

The `all()` method returns a **QuerySet** of all the objects in the database. The **QuerySet** returned by `all()` describes all objects in the database table. To select only a subset of the complete set of objects, it must be refined by adding filter conditions.

The two most common ways to refine a **QuerySet** are:

filter(kwargs)**

Returns a new **QuerySet** containing objects that match the given lookup parameters.

exclude(kwargs)**

Returns a new **QuerySet** containing objects that do not match the given lookup parameters. The lookup parameters (****kwargs** in the above function definitions) should be in the format described in **Field** lookups below.

For example, to get a **QuerySet** of blog entries from the year 2006, use `filter()` like so:

Note: The following examples are taken from the official Django documentation for the sole purpose of introducing the general concepts.

```

1 Entry.objects.filter(pub_date__year=2006)

```

With the default manager class, it is the same as:

```

1 Entry.objects.all().filter(pub_date__year=2006)

```

The result of refining a **QuerySet** is itself a **QuerySet**, so it's possible to chain refinements together.

For example:

Note: The following examples are taken from the official Django documentation for the sole purpose of introducing the general concepts.

```
1 >>> Entry.objects.filter(  
2     ...     headline__startswith='What'  
3     ... ).exclude(  
4     ...     pub_date__gte=datetime.date.today()  
5     ... ).filter(  
6     ...     pub_date__gte=datetime(2005, 1, 30)  
7     ... )
```

This takes the initial **QuerySet** of all entries in the database, adds a filter, then an exclusion, then another filter. The final result is a **QuerySet** containing all entries with a headline that starts with “What”, that were published between January 30, 2005, and the current day.

More: [Making queries](#)

URLs and views

A clean elegant URL scheme is an important detail in a high-quality Web application. Django encourages beautiful URL design and does not put junk in URLs, like .php or .asp.

In Django, a Python module called `urls.py` is like a table of contents for the application. It contains a simple mapping between URL patterns and **views**.

Note: The following examples are taken from the official Django documentation for the sole purpose of introducing the general concepts.

```
1 from django.conf.urls import url  
2 from . import views  
3  
4 urlpatterns = [  
5     url(r'^bands/$', views.band_listing, name='band-list'),  
6     url(r'^bands/(\d+)/$', views.band_detail, name='band-detail'),  
7     url(r'^bands/search/$', views.band_search, name='band-search'),  
8 ]
```

```
1 from django.shortcuts import render  
2  
3 def band_listing(request):  
4     """A view of all bands."""  
5     bands = models.Band.objects.all()  
6     return render(request, 'bands/band_listing.html', {'bands': bands})
```

More: [URL dispatcher](#)

Templates

Django’s template language allows developers to put logic into the HTML:

Note: The following examples are taken from the official Django documentation for the sole purpose of introducing the general concepts.

```

1 <html>
2   <head>
3     <title>Band Listing</title>
4   </head>
5   <body>
6     <h1>All Bands</h1>
7     <ul>
8       {% for band in bands %}
9         <li>
10          <h2><a href="{{ band.get_absolute_url }}">{{ band.name }}</a></h2>
11          {% if band.can_rock %}<p>This band can rock!</p>{% endif %}
12        </li>
13      {% endfor %}
14    </ul>
15  </body>
16</html>

```

More: [Templates](#)

Forms

Django provides a library that handles rendering HTML forms, validation of data submitted by users, and converting the data to native Python types. Django also provides a way to generate forms from your existing models and to use these forms to create and update data.

Note: The following examples are taken from the official Django documentation for the sole purpose of introducing the general concepts.

```

1 from django import forms
2
3 class BandContactForm(forms.Form):
4     subject = forms.CharField(max_length=100)
5     message = forms.CharField()
6     sender = forms.EmailField()
7     cc_myself = forms.BooleanField(required=False)

```

GET and POST

GET and **POST** are the only HTTP methods to use when dealing with forms.

Django's login form is returned using the **POST** method, in which the browser bundles up the form data, encodes it for transmission, sends it to the server, and then receives back its response.

GET, by contrast, bundles the submitted data into a string, and uses this to compose a URL. The URL contains the address where the data must be sent, as well as the data keys and values. You can see this in action if you do a search in the Django documentation, which will produce a URL of the form `https://docs.djangoproject.com/search/?q=forms&release=1`.

GET and **POST** are typically used for different purposes.

Any request that could be used to change the state of the system - for example, a request that makes changes in the database - should use **POST**. **GET** should be used only for requests that do not affect the state of the system.

GET would also be unsuitable for a password form, because the password would appear in the URL, and thus, also in browser history and server logs, all in plain text. Neither would it be suitable for large quantities of data, or for binary data, such as an image. A Web application that uses **GET** requests for admin forms is a security risk: it can be easy for an attacker to mimic a form's request to gain access to sensitive parts of the system. **POST**, coupled with other protections like Django's CSRF protection offers more control over access.

On the other hand, **GET** is suitable for things like a web search form, because the URLs that represent a **GET** request can easily be bookmarked, shared, or resubmitted.

More: [Working With Forms](#)

Authentication

Django supports a full-featured and secure authentication system. It handles user accounts, groups, permissions and cookie-based user sessions.

Note: The following examples are taken from the official Django documentation for the sole purpose of introducing the general concepts.

```
1 from django.contrib.auth.decorators import login_required
2 from django.shortcuts import render
3
4 @login_required
5 def my_protected_view(request):
6     """A view that can only be accessed by logged-in users"""
7     return render(request, 'protected.html', {'current_user': request.user})
```

More: [User authentication in Django](#)

Admin

One of the most powerful parts of Django is its automatic admin interface. It reads metadata from models in order to provide a powerful and ready-to-use GUI for **CRUD** operations against the model.

Note: The following examples are taken from the official Django documentation for the sole purpose of introducing the general concepts.

```
1 from django.contrib import admin
2 from bands.models import Band, Member
3
4 class MemberAdmin(admin.ModelAdmin):
5     """Customize the look of the auto-generated admin for the Member model"""
6     list_display = ('name', 'instrument')
7     list_filter = ('band',)
8
9 admin.site.register(Band) # Use the default options
10 admin.site.register(Member, MemberAdmin) # Use the customized options
```

Note: The advanced workshop for Developers will provide more details on GeoNode specific models and admin interface

More: [The Django admin site](#)

Internationalization

Django offers full support for translating text into different languages, plus locale-specific formatting of dates, times, numbers and time zones. It lets developers and template authors specify which parts of their apps should be translated or formatted for local languages and cultures, and it uses these hooks to localize Web applications for particular users according to their preferences.

Note: The following examples are taken from the official Django documentation for the sole purpose of introducing the general concepts.

```

1 from django.shortcuts import render
2 from django.utils.translation import ugettext
3
4 def homepage(request):
5     """
6     Shows the homepage with a welcome message that is translated in the
7     user's language.
8     """
9     message = ugettext('Welcome to our site!')
10    return render(request, 'homepage.html', {'message': message})

```

```

1 {% load i18n %}
2 <html>
3   <head>
4     <title>{% trans 'Homepage - Hall of Fame' %}</title>
5   </head>
6   <body>
7     {# Translated in the view: #}
8     <h1>{{ message }}</h1>
9     <p>
10      {% blocktrans count member_count=bands.count %}
11      Here is the only band in the hall of fame:
12      {% plural %}
13      Here are all the {{ member_count }} bands in the hall of fame:
14      {% endblocktrans %}
15    </p>
16    <ul>
17      {% for band in bands %}
18        <li>
19          <h2><a href="{{ band.get_absolute_url }}">{{ band.name }}</a></h2>
20          {% if band.can_rock %}<p>{% trans 'This band can rock!' %}</p>{% endif %}
21        </li>
22      {% endfor %}
23    </ul>
24  </body>
25 </html>

```

Note: The advanced workshop for Developers will provide more details on how to create languages and translations on GeoNode using [Transifex](#)

More: [Internationalization and localization](#)

Security

Django provides multiple protections against:

- **Clickjacking** Clickjacking is a type of attack where a malicious site wraps another site in a frame. This attack can result in an unsuspecting user being tricked into performing unintended actions on the target site.

The **X-Frame-Options** middleware contained in a form allow a supporting browser to prevent a site from being rendered inside a frame

- **Cross site scripting (XSS)** XSS attacks allow a user to inject client side scripts into the browsers of other users. This is usually achieved by storing the malicious scripts in the database where it will be retrieved and displayed to other users, or by getting users to click a link which will cause the attacker's JavaScript to be executed by the user's browser. However, XSS attacks can originate from any untrusted source of data, such as cookies or Web services, whenever the data is not sufficiently sanitized before including in a page.

- **Cross site request forgery (CSRF)** CSRF attacks allow a malicious user to execute actions using the credentials of another user without that user's knowledge or consent.

CSRF protection works by checking for a nonce in each POST request. This ensures that a malicious user cannot simply "replay" a form POST to your Web site and have another logged in user unwittingly submit that form. The malicious user would have to know the nonce, which is user specific (using a cookie).

- **SQL injection** SQL injection is a type of attack where a malicious user is able to execute arbitrary SQL code on a database. This can result in records being deleted or data leakage.

- **Host header validation** Django uses the **Host** header provided by the client to construct URLs in certain cases. While these values are sanitized to prevent Cross Site Scripting attacks, a fake **Host** value can be used for Cross-Site Request Forgery, cache poisoning attacks, and poisoning links in emails.

Because even seemingly-secure web server configurations are susceptible to fake **Host** headers, Django validates **Host** headers against the **ALLOWED_HOSTS** setting in the `django.http.HttpRequest.get_host()` method.

This validation only applies via `get_host()`; if your code accesses the Host header directly from `request.META` you are bypassing this security protection.

- **SSL/HTTPS** It is always better for security, though not always practical in all cases, to deploy your site behind HTTPS. Without this, it is possible for malicious network users to sniff authentication credentials or any other information transferred between client and server, and in some cases – active network attackers – to alter data that is sent in either direction.

Django provides some settings to secure your site under SSL/HTTPS.

Warning: While Django provides good security protection out of the box, it is still important to properly deploy your application and take advantage of the security protection of the Web server, operating system and other components.

- Make sure that your Python code is outside of the Web server's root. This will ensure that your Python code is not accidentally served as plain text (or accidentally executed).
- Take care with any user uploaded files.
- Django does not throttle requests to authenticate users. To protect against brute-force attacks against the authentication system, you may consider deploying a Django plugin or Web server module to throttle these requests.
- Keep your **SECRET_KEY** a secret.

- It is a good idea to limit the accessibility of your caching system and database using a firewall.

More: [Security in Django](#)

3.2.1.5.3 Development Prerequisites and Core Modules

This module will introduce you to the basic tools and skills required to start actively developing GeoNode.

GeoNode's Development Prerequisites

Basic Shell Tools

ssh and sudo

SSH and sudo are very basic terminal skills which you will need to deploy, maintain and develop with GeoNode. If you are not already familiar with their usage, you should review the basic descriptions below and follow the external links to learn more about how to use them effectively as part of your development workflow.

ssh is the network protocol used to connect to a remote server where you run your GeoNode instance whether on your own network or on the cloud. You will need to know how to use the *ssh* command from the terminal on your Unix machine or how to use a *ssh* client like *putty* or *WinSCP* on windows. You may need to use PKI certificates to connect to your remote server, and should be familiar with the steps and options necessary to connect this way. More information about *ssh* can be found in the links below.

- <http://winscp.net/eng/docs/ssh>

sudo is the command used to execute a terminal command as the superuser when you are logged in with a normal user. You will use *sudo* in order to start, stop and restart key services on your GeoNode instance. If you are not able to grant yourself these privileges on the machine you are using for your GeoNode instance, you may need to consult with your network administrator to arrange for your user to be granted *sudo* permissions. More information about *sudo* can be found in the links below.

- <https://en.wikipedia.org/wiki/Sudo>

bash

Bash is the most common Unix shell which will usually be the default on servers where you will be deploying your GeoNode instance. You should be familiar with the most common *bash* commands in order to be able to deploy, maintain and modify a *geonode* instance. More information about *Bash* and common *bash* commands can be found in the links below.

- [https://en.wikipedia.org/wiki/Bash_\(Unix_shell\)](https://en.wikipedia.org/wiki/Bash_(Unix_shell))

apt

apt is the packaging tool that is used to install GeoNode on ubuntu and other Debian based systems. You will need to be familiar with adding Personal Package Archives to your list of install sources, and will need to be familiar with basic *apt* commands. More information about *apt* can be found in the links below.

- https://en.wikipedia.org/wiki/Advanced_Packaging_Tool

Python Development Tools

The GeoNode development process relies on several widely used python development tools in order to make things easier for developers and other users of the systems that GeoNode developers work on or where GeoNodes are deployed. They are considered best practices for modern python development, and you should become familiar with these basic tools and be comfortable using them on your own projects and systems.

virtualenv

virtualenv is a tool used to create isolated python development environments such that the versions of project dependencies are sandboxed from the system-wide python packages. This eliminates the commonly encountered problem of different projects on the same system using different versions of the same library. You should be familiar with how to create and activate virtual environments for the projects you work on. More information about virtualenv can be found in the links below.

- <https://virtualenv.pypa.io/en/stable/>
- <https://pypi.python.org/pypi/virtualenv>

virtualenvwrapper is a wrapper around the virtualenv package that makes it easier to create and switch between virtual environments as you do development. Using it will make your life much easier, so its recommended that you install and configure it and use its commands as part of your virtualenv workflow. More info about virtualenvwrapper can be found in the links below.

- <https://bitbucket.org/dhellmann/virtualenvwrapper>

pip

pip is a tool for installing and managing python packages. Specifically it is used to install and upgrade packages found in the Python Package Index (PyPI). GeoNode uses pip to install itself, and to manage all of the python dependencies that are needed as part of a GeoNode instance. As you learn to add new modules to your geonode, you will need to become familiar with the use of pip and about basic python packaging usage. More information about pip can be found in the links below.

- <https://pip.pypa.io/en/latest/>
- <https://pypi.python.org/pypi/pip>
- [https://en.wikipedia.org/wiki/Pip_\(package_manager\)](https://en.wikipedia.org/wiki/Pip_(package_manager))

miscellaneous

ipython is a set of tools to make your python development and debugging experience easier. The primary tool you want to use is an interactive shell that adds introspection, integrated help and command completion and more. While not strictly required to do GeoNode development, learning how to use ipython will make your development more productive and pleasant. More information about ipython can be found in the links below.

- <http://ipython.org/>
- <https://pypi.python.org/pypi/ipython>
- <https://github.com/ipython/ipython>
- <https://en.wikipedia.org/wiki/IPython>

pdb is a standard python module that is used to interactively debug your python code. It supports setting conditional breakpoints so you can step through the code line by line and inspect your variables and perform arbitrary execution of statements. Learning how to effectively use *pdb* will make the process of debugging your application code significantly easier. More information about *pdb* can be found in the links below.

- <https://docs.python.org/2/library/pdb.html>

Django

GeoNode is built on top of the *Django web framework*, and as such, you will need to become generally familiar with Django itself in order to become a productive GeoNode developer. Django has excellent documentation, and you should familiarize yourself with Django by following the Django workshop and reading through its documentation as required.

Model Template View

Django is based on the Model Template View paradigm (more commonly called Model View Controller). Models are used to define objects that you use in your application and Django's ORM is used to map these models to a database. Views are used to implement the business logic of your application and provide objects and other context for the templates. Templates are used to render the context from views into a page for display to the user. You should become familiar with this common paradigm used in most modern web frameworks, and how it is specifically implemented and used in Django. The Django tutorial itself is a great place to start. More information about MTV in Django can be found in the links below.

- <https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller>
- <https://blog.codinghorror.com/understanding-model-view-controller/>
- <https://docs.djangoproject.com/en/1.8/>

HTTP Request Response

Django and all other web frameworks are based on the HTTP Request Response cycle. Requests come in to the server from remote clients which are primarily web browsers, but also can be API clients, and the server returns with a Response. You should be familiar with these very basic HTTP principles and become familiar with the way that Django implements them. More information about HTTP, Requests and Responses and Django's implementation in the links below.

- <https://www.jmarshall.com/easy/http/>
- https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol
- <https://docs.djangoproject.com/en/dev/ref/request-response/>

Management Commands

Django projects have access to a set of management commands that are used to manage your project. Django itself provides a set of these commands, and Django apps (including GeoNode) can provide their own. Management commands are used to do things like synchronize your models with your database, load data from fixtures or back up your database with fixtures, start the development server, initiate the debugger and many other things. GeoNode provides management commands for synchronizing with a GeoServer or updating the layers already in your GeoNode. You should become familiar with the basic management commands that come with Django, and specifically with the commands that are part of GeoNode. The GeoNode specific commands are covered in section. More information about management commands can be found in the links below.

- <https://docs.djangoproject.com/en/dev/ref/django-admin/>

Django Admin Interface

Django provides a build-in management console that administrators and developers can use to look at the data in the database that is part of the installed applications. Administrators can use this console to perform many common administration tasks that are a necessary part of running a GeoNode instance, and as a developer, you will use this interface during your development process to inspect the database and the data stored in your models. More information about the Django admin interface can be found in the links below.

- <https://docs.djangoproject.com/en/dev/ref/contrib/admin/>

Template Tags

Django templates make use of a set of tags to inject, file and format content into a rendered HTML page. Django itself includes a set of built-in template tags and filters that you will use in your own templates, and GeoNode provides a geonode specific set of tags that are used in the GeoNode templates. You should become familiar with the built-in tag set and with GeoNode's specific tags as you work on developing your own templates or extending from GeoNode's. More information about Django template tags can be found in the links below.

- <https://docs.djangoproject.com/en/dev/ref/templates/builtins/>

GeoNode's Core Modules

GeoNode is made up of a set of core Django pluggable modules (known as apps in Django) that provide the functionality of the application. Together they make up the key components of a GeoNode site. While your own use case and implementation may not require that you work directly on these models, it is important that you become familiar with their layout, structure and the functionality that they provide. You may need to import these apps into your own apps, and as such, becoming familiar with them is an important step in becoming a proficient GeoNode developer.

geonode.layers

geonode.layers is the most key GeoNode module. It is used to represent layers of data stored in a GeoNode's paired GeoServer. The layer model class inherits fields from the ResourceBase class which provides all of the fields necessary for the metadata catalogue, and adds fields that map the object to its corresponding layer in GeoServer. When your users upload a layer via the user interface, the layer is imported to GeoServer and a record is added to GeoNode's database to represent that GeoServer layer within GeoNode itself.

The Layer model class provides a set of helper methods that are used to perform operations on a Layer object, and also to return things such as the list of Download or Metadata links for that layer. Additional classes are used to model the layers Attributes, Styles, Contacts and Links. The Django signals framework is used to invoke specific functions to synchronize with GeoServer before and after the layer is saved.

The views in the layers app are used to perform functions such as uploading, replacing, removing or changing the points of contact for a layer, and views are also used to update layer styles, download layers in bulk or change a layers permissions.

The forms module in the layer app is used to drive the user interface forms necessary for performing the business logic that the views provide.

The Layers app also includes a set of templates that are paired with views and used to drive the user interface. A small set of layer template tags is also used to help drive the layer explore and search pages.

Some helper modules such as `geonode.layers.metadata` and `geonode.layers.ows` are used by the layer views to perform specific functions and help keep the main views module more concise and legible.

Additionally, the GeoNode specific management commands are a part of the `geonode.layers` app.

You should spend some time to review the layers app through GitHub's code browsing interface.

<https://github.com/GeoNode/geonode/tree/master/geonode/layers>

geonode.maps

The `geonode.maps` app is used to group together GeoNode's multi layer map functionality. The `Map` and `MapLayer` objects are used to model and implement maps created with the GeoExplorer application. The `Map` object also extends from the `ResourceBase` class which provides the ability to manage a full set of metadata fields for a `Map`.

The views in the maps app perform many of the same functions as the views in the layers app such as adding, changing, replacing or removing a map and also provide the endpoints for returning the map configuration from the database that is used to initialize the GeoExplorer app.

The maps app also includes a set of forms, customization of the Django admin, some utility functions and a set of templates and template tags.

You can familiarize yourself with the maps app on GitHub.

<https://github.com/GeoNode/geonode/tree/master/geonode/maps>

geonode.security

The `geonode.security` app is used to provide object level permissions within the GeoNode Django application. It is a custom Django authentication backend and is used to assign Generic, User and Group Permissions to Layers, Maps and other objects in the GeoNode system. Generic permissions are used to enable public anonymous or authenticated viewing and/or editing of your data layers and maps, and User and Group specific permissions are used to allow specific users or groups to access and edit your layers.

geonode.search

The `geonode.search` module provides the search API that is used to drive the GeoNode search pages. It is configured to index layers, maps, documents and profiles, but is extensible to allow you to use it to index your own model classes. This module is currently based on the Django ORM and as such has a limited set of search features, but the GeoNode development team is actively working on making it possible to use this module with more feature-rich search engines.

geonode.catalogue

The `geonode.catalogue` app provides a key set of metadata catalogue functions within GeoNode itself. GeoNode is configured to use an integrated version of the `pycsw` library to perform these functions, but can also be configured to use any OGC compliant CS-W implementation such as `GeoNetwork` or `Deegree`. The metadata app allows users to import and/or edit metadata for their layers, maps and documents, and it provides an OGC compliant search interface for use in federating with other systems.

geonode.geoserver

The `geonode.geoserver` module is used to interact with GeoServer from within GeoNode's python code. It relies heavily on the `gsconfig` library which addresses GeoServer's REST configuration API. Additionally, the `geonode.geoserver.uploader` module is used to interact with GeoServer's Importer API for uploading and configuring layers.

geonode.people

The `geonode.people` module is used to model and store information about both GeoNode users and people outside of the system who are listed as Points of Contact for particular layers. It is the foundational module for GeoNode's social features. It provides a set of forms for users to edit and manage their own profiles as well as to view and interact with the profiles of other users.

geoexplorer

GeoNode's core GIS client functions are performed by GeoExplorer. The GeoExplorer app is in turn based on GeoExt, OpenLayers and ExtJS. It provides functionality for constructing maps, styling layers and connecting to remote services. GeoExplorer is the reference implementation of the OpenGeo Suite SDK which is based on GXP. GeoNode treats GeoExplorer as an external module that is used out of the box in GeoNode, but it is possible for you to create your own Suite SDK app and integrate it with GeoNode.

Static Site

The front end of GeoNode is composed of a set of core templates, specific templates for each module, cascading style sheets to style those pages and a set of Javascript modules that provide the interactive functionality in the site.

Templates

GeoNode includes a basic set of core templates that use Django's template inheritance system to provide a modular system for constructing the web pages in GeoNode's interface. These core templates drive the overall page layout and things like the home page. You will start the process of customizing your GeoNode instance by overriding these templates, so you should familiarize yourself with their structure and how they inherit from each other to drive the pages.

Additionally, most of the apps described above have their own set of templates that are used to drive the pages for each module. You may also want to override these templates for your own purposes and as such should familiarize yourself with a few of the key ones.

CSS

GeoNode's CSS is based on Twitter's Bootstrap Library which uses the `lessc` dynamic stylesheet language. GeoNode extends from the basic Bootstrap style and you are able to create your own bootstrap based style to customize the look and feel of your own GeoNode instance. Sites like bootswatch.com also provide ready made styles that you can simply drop in to your project to change the style.

Javascript

The interactive functionality in GeoNode pages is provided by the jQuery Javascript framework and a set of jQuery plugins. The core set of GeoNode Javascript modules closely aligns with the apps described above, and there are also a few pieces of functionality provided as Javascript modules that are used through out all of the apps. You are able to add your own jQuery code and/or plugins to perform interactive functionality in your own application.

Exercises

Shell and Utilities

1. *ssh* into your virtual machine or other instance
2. *sudo* to modify the *sshd_config* settings to verify disabling of DNS resolution (UseDNS=no)
3. Install a command line helper

```
$ sudo apt-get install bash-completion
```

4. Exercise command completion

```
$ apt-get install <TAB><TAB>
```

5. Activate/deactivate the *virtualenv* on your instance

```
$ source /var/lib/geonode/bin/activate
$ deactivate
```

6. Set the *DJANGO_SETTINGS_MODULE* environment variable

```
$ export DJANGO_SETTINGS_MODULE=geonode.settings
```

7. Install the *httplib* utility via pip

```
$ pip install httplib
$ http http://localhost/geoserver/rest
$ http -a admin http://localhost/geoserver/rest
<type in password - geoserver>
```

Python

1. Launch *ipython* and experiment

```
> x = "some text"
> x.<TAB><TAB>
> x.split.__doc__
> ?
```

2. Execute a script with *ipython* and open the REPL

```
$ echo "twos = [ x*2 for x in range(5)]" > test.py
$ ipython -i test.py
> twos
```

3.2.1.5.4 Install GeoNode for Development

In order to install Geonode 2.4 in development mode on Ubuntu 14.04 the following steps are required:

For Windows: (win_devinstall)

For CentOS 7: (*Install GeoNode on CentOS 7 (dev mode)*)

Summary of the installation steps

1. Retrieve latest apt-get list
2. Install build tools and libraries
3. Install dependencies (Python, Postgresql and Java) and supporting tools
4. Add Node.js PPA and other tools required for static development
5. Set up a virtual environment (virtualenv)
6. Clone geonode from GitHub and install it in the virtual environment
7. Run paver to get install GeoServer and start the development servers
8. Compile and Start the server
9. Start Geonode instance
10. To stop the server
11. Next create a Django superuser for your GeoNode

Note: The following steps have to be executed in your terminal. The steps have to be done as a **root user**, therefore don't forget to type sudo in front!

Warning: Don't forget to stop the **GeoNode Production** services if enabled

```
service apache2 stop
service tomcat7 stop
```

1. If possible log as **root** user, open a terminal and `cd /home/geonode/dev`
2. Retrieve latest apt-get list

```
$ sudo apt-get update
```

3. Install build tools and libraries

```
$ sudo apt-get install -y build-essential libxml2-dev libxslt1-dev libpq-dev
↳zlib1g-dev
```

4. Install dependencies

Python native dependencies

```
$ sudo apt-get install -y python-dev python-imaging python-lxml python-pyproj
↳python-shapely python-nose python-httplib2 python-pip python-software-properties
```

Install Python Virtual Environment

```
$ sudo pip install virtualenvwrapper
```

PostgreSQL & PostGIS

Note: The following steps must be executed **only** if you don't have PostgreSQL and PostGIS already installed on your system (see [Install GeoNode Application](#))

```
$ sudo apt-get install -y postgresql-9.3-postgis-2.1 postgresql-9.3-postgis-scripts
```

Change postgres UNIX password

```
$ sudo passwd -u postgres # change password expiry information

$ sudo passwd postgres # change unix password for postgres

Create GeoNode role and database

.. code-block:: console

$ su postgres
$ createdb geonode_dev
$ createdb geonode_dev-imports
$ psql
postgres=#
postgres=# \password postgres
postgres=# CREATE USER geonode_dev WITH PASSWORD 'geonode_dev';
→# should be same as password in setting.py
postgres=# GRANT ALL PRIVILEGES ON DATABASE "geonode_dev" to_
→geonode_dev;
postgres=# GRANT ALL PRIVILEGES ON DATABASE "geonode_dev-imports
→" to geonode_dev;
postgres=# \q

$ psql -d geonode_dev-imports -c 'CREATE EXTENSION postgis;'
$ psql -d geonode_dev-imports -c 'GRANT ALL ON geometry_columns_
→TO PUBLIC;'
$ psql -d geonode_dev-imports -c 'GRANT ALL ON spatial_ref_sys TO_
→PUBLIC;'

$ exit
```

Edit PostgreSQL configuration file

```
sudo gedit /etc/postgresql/9.3/main/pg_hba.conf
```

Scroll to the bottom of the file and edit this line

```
# "local" is for Unix domain socket connections only
local    all                                all                                peer
```

As follows

```
# "local" is for Unix domain socket connections only
local    all                                all                                trust
```

Restart PostgreSQL to make the changes effective

```
sudo service postgresql restart
```

Java dependencies

Note: The following steps must be executed **only** if you don't have a Java JDK or JRE already installed on your system (see *Install GeoNode Application*)

```
$ sudo apt-get install -y openjdk-7-jdk --no-install-recommends
```

Supporting tools

```
$ sudo apt-get install -y git gettext libjpeg62 libjpeg62-dev
```

5. Set up a virtual environment

This is the local environment where Geonode will later be running.

Add the virtualenvwrapper to your new environment.

```
$ cd /home/geonode/dev

$ export VIRTUALENVWRAPPER_PYTHON=/usr/bin/python
$ export WORKON_HOME=/home/geonode/dev/.venvs
$ source /usr/local/bin/virtualenvwrapper.sh
$ export PIP_DOWNLOAD_CACHE=$HOME/.pip-downloads
```

On Ubuntu, you can add the above settings to your `.bashrc` file and reload the settings running

```
$ echo export VIRTUALENVWRAPPER_PYTHON=/usr/bin/python >> ~/.bashrc
$ echo export WORKON_HOME=/home/geonode/dev/.venvs >> ~/.bashrc
$ echo source /usr/local/bin/virtualenvwrapper.sh >> ~/.bashrc
$ echo export PIP_DOWNLOAD_CACHE=$HOME/.pip-downloads >> ~/.bashrc

$ source ~/.bashrc
```

Set up the local virtual environment for Geonode

```
$ mkvirtualenv geonode
$ workon geonode # or $ source /home/geonode/dev/.venvs/geonode/bin/
→activate
```

This creates a new directory where you want your project to be and creates a new virtualenvironment

6. Get the GeoNode source code

To download the latest GeoNode version from GitHub, the command `clone` is used

Note: If you are following the GeoNode training, skip the following command. You can find the cloned repository in `/home/geonode/dev`

```
$ git clone https://github.com/GeoNode/geonode.git
```

7. Add Node.js PPA and other tools required for static development

This is required for static development

Note: If you are following GeoNode's training, nodejs is already installed in the Virtual Machine skip the first three command and jump to `cd geonode/geonode/static`

```
$ sudo add-apt-repository -y ppa:chris-lea/node.js
$ sudo apt-get update
$ sudo apt-get install -y nodejs
$ cd geonode/geonode/static
$ npm install --save-dev

# If the last command does not work, you can run it manually
↳ like this:

$ npm install bower --save-dev
$ npm install grunt-cli --save-dev
$ npm install grunt-contrib-jshint --save-dev
$ npm install grunt-contrib-less --save-dev
$ npm install grunt-contrib-concat --save-dev
$ npm install grunt-contrib-copy --save-dev
$ npm install grunt-text-replace --save-dev
$ npm install grunt-contrib-uglify --save-dev
$ npm install grunt-contrib-cssmin --save-dev
$ npm install grunt-contrib-watch --save-dev
```

Note: Every time you want to update the static files after making changes to the sources, go to `geonode/static` and run 'grunt production'.

8. Install GeoNode in the new active local virtualenv

```
$ cd /home/geonode/dev
$ pip install pip --upgrade
$ pip install -e geonode --use-mirrors

$ cd geonode
```

If the install fails because of an error related to pyproj not being verified (happens on pip 1.5), use the following:

```
$ pip install -e geonode --use-mirrors --allow-external pyproj --allow-unverified_
↳ pyproj
```

9. Create local_settings.py

Add the `local_settings.py` to your GeoNode installation

```
$ cd /home/geonode/dev/geonode
$ cp geonode/local_settings.py.sample geonode/local_settings.py
$ gedit geonode/local_settings.py
```

Add the following lines to the `local_settings.py`

```

...

SITEURL = "http://localhost:8000/"

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'geonode_dev',
        'USER': 'geonode_dev',
        'PASSWORD': 'geonode_dev',
    },
    # vector datastore for uploads
    'datastore': {
        'ENGINE': 'django.contrib.gis.db.backends.postgis',
        # 'ENGINE': '', # Empty ENGINE name disables
        'NAME': 'geonode_dev-imports',
        'USER': 'geonode_dev',
        'PASSWORD': 'geonode_dev',
        'HOST': 'localhost',
        'PORT': '5432',
    }
}

# OGC (WMS/WFS/WCS) Server Settings
OGC_SERVER = {
    'default': {
        'BACKEND': 'geonode.geoserver',
        'LOCATION': 'http://localhost:8080/geoserver/',
        'PUBLIC_LOCATION': 'http://localhost:8080/geoserver/',
        'USER': 'admin',
        'PASSWORD': 'geoserver',
        'MAPFISH_PRINT_ENABLED': True,
        'PRINT_NG_ENABLED': True,
        'GEONODE_SECURITY_ENABLED': True,
        'GEOGIG_ENABLED': False,
        'WMST_ENABLED': False,
        'BACKEND_WRITE_ENABLED': True,
        'WPS_ENABLED': False,
        'LOG_FILE': '%s/geoserver/data/logs/geoserver.log' % os.path.abspath(os.
↪path.join(PROJECT_ROOT, os.pardir)),
        # Set to name of database in DATABASES dictionary to enable
        'DATASTORE': 'datastore',
    }
}

CATALOGUE = {
    'default': {
        # The underlying CSW implementation
        # default is pycsw in local mode (tied directly to GeoNode Django DB)
        'ENGINE': 'geonode.catalogue.backends.pycsw_local',
        # pycsw in non-local mode
        # 'ENGINE': 'geonode.catalogue.backends.pycsw_http',
        # GeoNetwork opensource
        # 'ENGINE': 'geonode.catalogue.backends.geonetwork',
        # deegree and others
        # 'ENGINE': 'geonode.catalogue.backends.generic',
    }
}

```

(continues on next page)

(continued from previous page)

```

# The FULLY QUALIFIED base url to the CSW instance for this GeoNode
'URL': '%scatalogue/csw' % SITEURL,
# 'URL': 'http://localhost:8080/geonetwork/srv/en/csw',
# 'URL': 'http://localhost:8080/deegree-csw-demo-3.0.4/services',

# login credentials (for GeoNetwork)
'USER': 'admin',
'PASSWORD': 'admin',
    }
}
...

```

10. Compile and Start the server for the first time

Align the database structure

```

$ cd /home/geonode/dev/geonode
$ python manage.py migrate

```

Warning: If the start fails because of an import error related to osgeo, then please consult the `install_gdal_devmode`.

The last step is to compile GeoServer and setup, then initialize the database

```

$ paver setup
$ paver sync

```

11. Now we can start our GeoNode instance

Warning: Don't forget to stop the **GeoNode Production** services if enabled

```

service apache2 stop
service tomcat7 stop

```

```

$ paver start

```

Visit the GeoNode site by typing <http://localhost:8000> into your browser window.

If you are using a different IP address (e.g 1.1.1.1), then start paver using the command below.

```

$ paver start -b 1.1.1.1:8000

```

Warning: If the start fails because of an import error related to osgeo, then please consult the `install_gdal_devmode`.

12. To stop the server

Press **Ctrl-C** on your keyboard to stop the server.

then type:

```
$ paver stop      # to stop all django, geoserver services
```

13. Create a Django superuser for your GeoNode

Create a superuser so you can log on to your local GeoNode installation at <http://localhost:8000>

```
$ python manage.py createsuperuser
```

3.2.1.5.5 Start working on Geonode the next day after install

With every restart of your machine, you have to restart GeoNode as well. That means, you will not be able to open <http://localhost:8000> directly after starting your machine new. In order to be able to use GeoNode now, you have to activate your virtualenv and to start the development servers.

Note: *username* is the name of your machine and personal folder!

1. Activate virtualenv

To activate your virtualenv you just need to type

```
$ workon geonode
```

or

```
$ source /home/geonode/dev/.venvs/geonode/bin/activate
```

Note: Be careful with the path, it might not be the same for you!

2. Start the server

Warning: Don't forget to stop the **GeoNode Production** services if enabled

```
service apache2 stop
service tomcat7 stop
```

```
$ cd geonode
$ paver start_geoserver
$ paver start_django
```

Now you are able to access <http://localhost:8000> again.

Note: Remember that you have to do these steps each time you restart your machine!!

Hint: Now you've followed these installation instructions, GeoNode is running in development mode. This also means that you are using all the default settings of GeoNode. If you want to change them, e.g use Tomcat instead of Jetty, or Postgresql instead of sqlite3, you may follow the steps from the section **Configure Manually** in `custom_install`.

3.2.1.5.6 GeoNode debugging techniques

GeoNode can be difficult to debug as there are several different components involved:

- Browser - includes HTML/CSS issues, JavaScript, etc.
- Django - GeoNode HTML views and web APIs
- GeoServer - Core Wxx services and Platform REST APIs

When attempting to diagnose a specific problem, often the order of investigation mirrors the order above - that is, start with the client: Is this a bug in code running on the browser. If not, step to the next level: the Django responses to client requests. Often this is possible via the browser using the correct tools. Many requests require Django communications with GeoServer. This is the next stage of investigation if a specific bug does not appear to originate in Django or the client.

The following section covers techniques to help diagnose and debug errors.

Debugging GeoNode in the Browser

This section covers some techniques for debugging browser and Django related response bugs using the Firefox web browser extension named Firebug. The concepts covered apply to other browser's tools but may vary in terminology.

Another Firefox extension worth noting is 'JSONView'. This extension supports formatted viewing of JSON responses and integrates well with Firebug.

References:

- <https://getfirebug.com/faq/>
- <https://jsonview.com/>

Net Tab

The net tab allows viewing all of the network traffic from the browser. The subtabs (like the selected "Images" tab) allow filtering by the type of traffic.

In this screen-shot, the mouse hover displays the image content and the full URL requested. One can right-click to copy-paste the URL or view in a separate tab. This is useful for obtaining test URLs. The grayed out entries show that the resource was cached via conditional-get (the 304 not modified). Other very useful advanced information includes the size of the response and the loading indicator graphics on the right. At the bottom, note the total size and timing information.

Net Tab Exercises

1. Go to layers/maps/search pages and look at the various requests. Note the XHR subtab. Look at the various request specific tabs: headers, params, etc.
2. Use the 'disable browser cache' option and see how it affects page loads. Discuss advantages/challenges of caching.

DOM Tab

The DOM tab displays all of the top-level window objects. By drilling down, this can be a useful way to find out what's going on in a page.

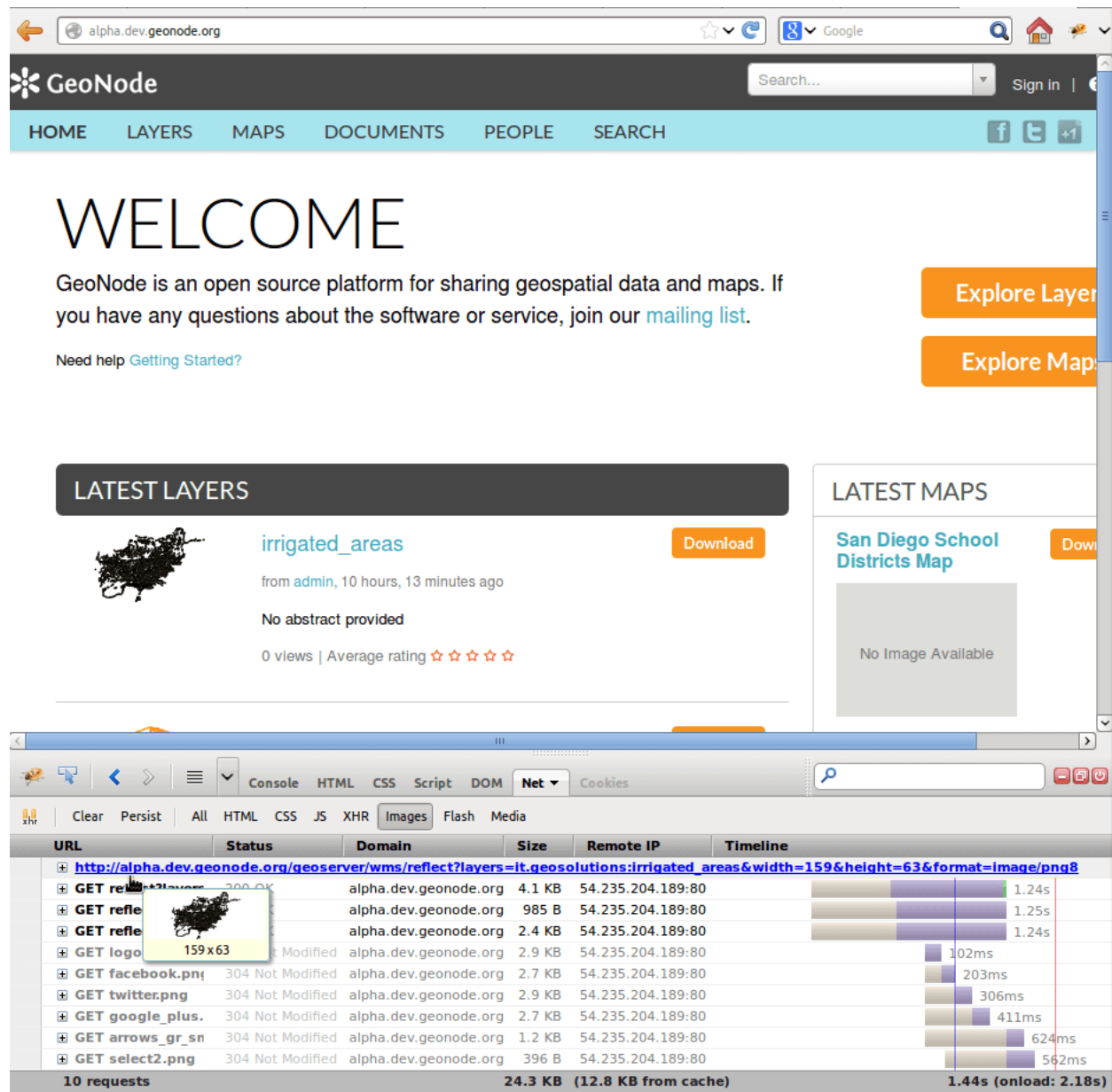


Fig. 83: Firebug Net Tab

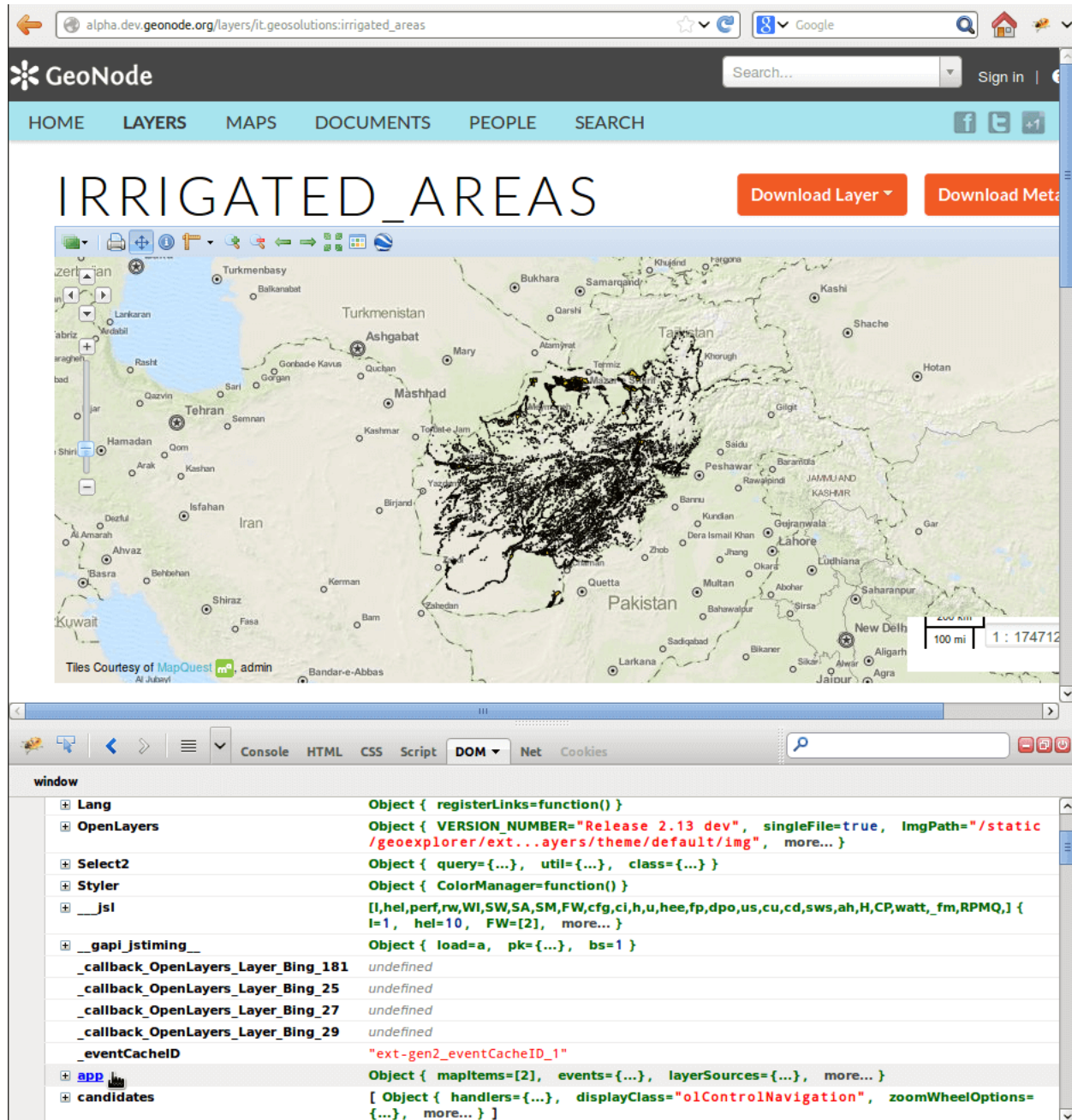


Fig. 84: Firebug DOM Tab

In this example, the mouse is hovering over the `app` object. Note the high level view of objects and their fields. The console tab allows interacting with the objects.

DOM Tab Exercises

1. Drill down in the DOM tab.
2. Use the console to interactively exercise jQuery.
3. Use the console to interact with the `app/map` or other page objects

Script Tab

The script tab allows viewing scripts and debugging.

The screen-shot displays a breakpoint set at line 3, the current code is stopped at line 8 and the mouse hover is displaying the value of the variable `'class_list'`. On the right, the `'Watch'` tab displays the various variables and scopes and offers a drill down view similar to the DOM view. The stack tab displays the execution stack context outside the current frame.

Script Tab Exercises

1. Step through some code
2. Look at various features: variables, scopes, DOM drill-down

HTML Tab

The HTML tag allows viewing and drilling down into the DOM. This is an incredibly useful feature when doing CSS or HTML work.

The screen-shot displays a search result `'article'` element highlighted with padding and margin in yellow and purple. The DOM structure is displayed on the left and the right panel displays the specific style rules while the computed tab displays the effective style rules. The layout tab displays rulers and property values while the DOM tab displays a debug/DOM-like view of the actual object's properties.

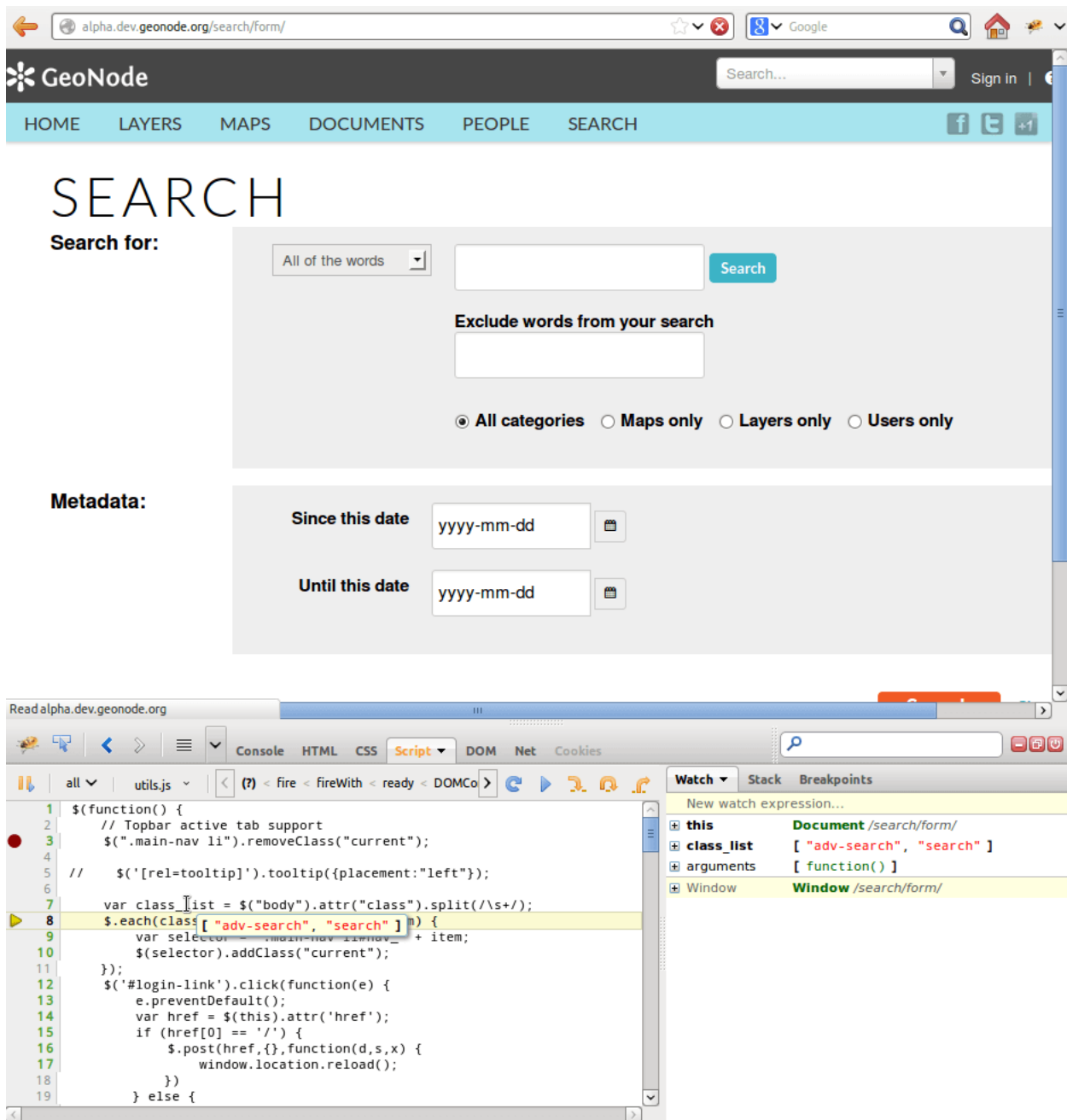
HTML Tab Exercises

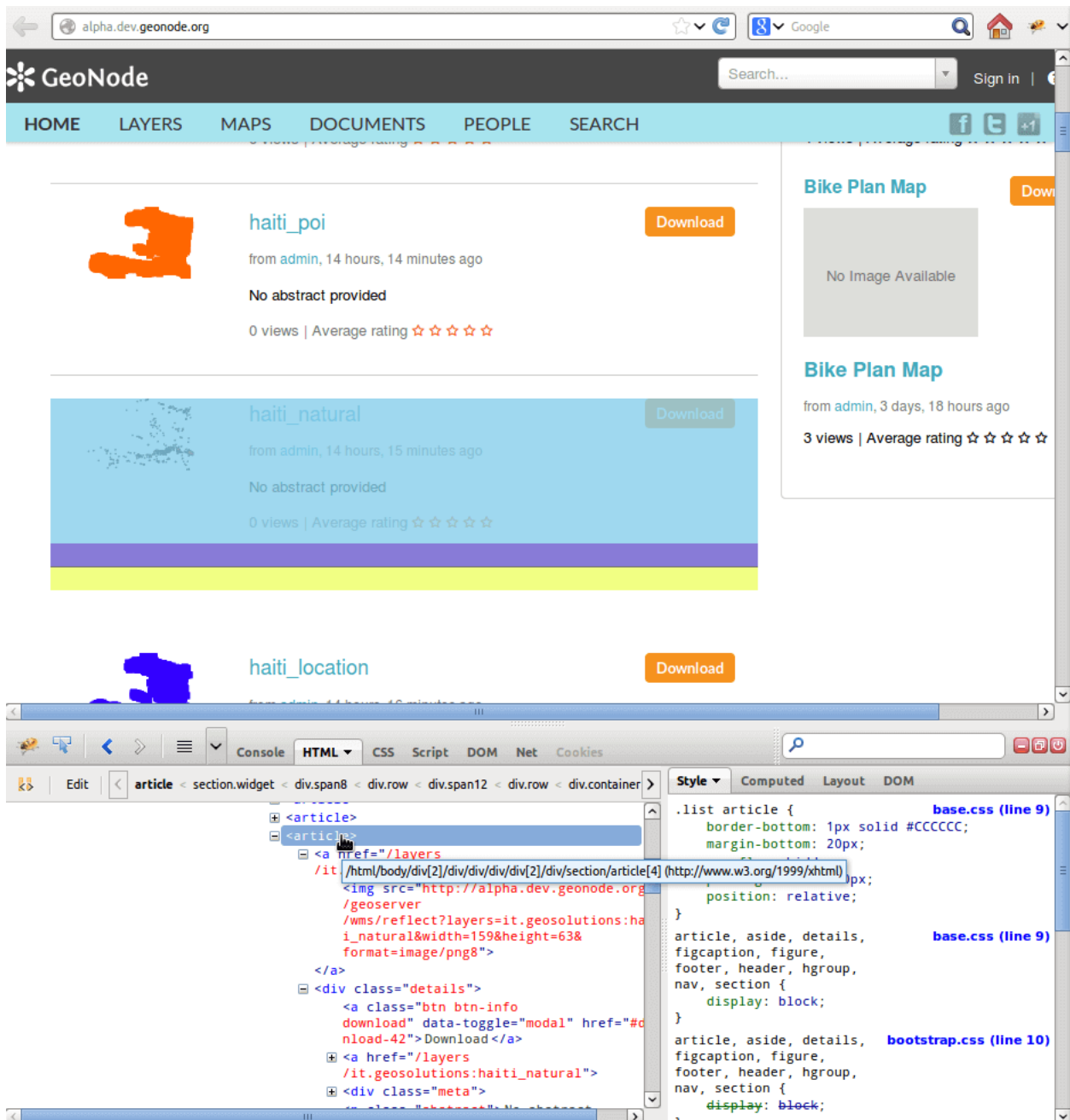
1. Identify elements by looking at the tabs on the right.
2. Change styles, adding new rules and styles.
3. Edit existing HTML elements via the raw view and the tree view.

Debugging GeoExplorer

In case you want to debug the GeoExplorer behaviour in your browser with Firebug of Chromium Developer toolbar, you may do the following:

Install Boundless Suite:





```
$ git clone git://github.com/GeoNode/suite.git
$ cd suite
$ git submodule update --init --recursive
```

Run GeoExplorer in debug mode:

```
$ cd geoexplorer
$ ant debug
```

Check if GeoExplorer is running at this URL: <http://localhost:9080>

Edit the `layers/templates/layers/layer_geoext_map.html` file and replace this line:

```
{% include "geoext/geo_header.html" %}
```

with this one:

```
{% include "geoext/geo_header_debug.html" %}
```

Debugging GeoNode's Python Components

Logging

References:

- <https://docs.python.org/2/library/logging.html>
- <https://docs.djangoproject.com/en/1.8/topics/logging/>

Logging is controlled by the contents of the logging data structure defined in the `settings.py`. The default settings distributed with GeoNode are configured to only log errors. During development, it's a good idea to override the logging data structure with something a bit more verbose.

Output

In production, logging output will go into the Apache error log. This is located in `/var/log/apache2/error.log`. During development, logging output will, by default, go to standard error.

Configuring

- Ensure the 'console' handler is at the appropriate level. It will ignore log messages below the set level.
- Ensure the specific logger you'd like to use is set at the correct level.
- If attempting to log SQL, ensure `DEBUG=True` in your `local_settings.py`.

Debugging SQL

- To trace all SQL in Django, configure the `django.db.backends` logger to `DEBUG`
- To examine a specific query object, you can use the `query` field: `str(Layer.objects.all().query)`
- You can gather more information by using `django.db.connection.queries`. When `DEBUG` is enabled, query SQL and timing information is stored in this list.

Hints

- Don't use print statements. They are easy to use in development mode but in production they will cause failure.
- Take advantage of python. Instead of:

```
logging.info('some var ' + x + ' is not = ' + y)
```

Use:

```
logging.info('some var %s is not = %s', x, y)
```

Excercises:

1. Enable logging of all SQL statements. Visit some pages and view the logging output.
2. Using the python shell, use the `queries` object to demonstrate the results of specific queries.

PDB

Reference:

- <https://docs.python.org/2/library/pdb.html>

For the adventurous, pdb allows for an interactive debugging session. This is only possible when running in a shell via `manage.py runserver` or `paver runserver`.

To set a breakpoint, insert the following code before the code to debug.

```
import pdb; pdb.set_trace()
```

When execution reaches this statement, the debugger will activate. The commands are noted in the link above. In addition to those debugger specific commands, general python statements are supported. For example, typing the name of a variable in scope will yield the value via string coercion. Typing “n” will execute the next line, “c” will continue the execution of the program, “q” will quit.

Debugging GeoServer

Resources:

- <http://docs.geoserver.org/latest/en/user/production/troubleshooting.html>
- <http://docs.geoserver.org/latest/en/user/production/troubleshooting.html>

This section does not attempt to cover developer-level debugging in GeoServer as this is a much larger topic involving many more tools. The goal here is to provide ‘black-box’ techniques to help resolve and report problems.

Logging

GeoServer logging, while sometimes containing too much information, is the best way to start diagnosing an issue in GeoNode once the other. To create a proper error report for use in requesting support, providing any contextual logging information is critical.

When using a standard GeoServer installation, the GeoServer logs are located at `/usr/share/geoserver/data/logs/geoserver.log`. The properties files that control the varying rules are also located here.

Exercises

1. Switch logging levels for various loggers.
2. Look at the different logging profiles and discuss the loggers and levels.
3. Learn how to read stack traces, nested or otherwise.

Advanced Troubleshooting

JVM diagnostics and advanced troubleshooting techniques are covered in the GeoServer documents linked to above. When providing information for a bug report, these can be helpful but in-depth Java knowledge is required to fully comprehend the output from some of these tools.

Exercises

1. Look at jstack output

Using Django to Help Debug

The `gsconfig` library provides a rich interface to interacting with GeoServer's REST API. This allows high-level functions as well as viewing raw REST responses.

```
cat = Layer.objects.gs_catalog
cat.get_layers() # list of gsconfig layer objects
# OR, for a specific layer
lyr = Layer.objects.get(id=1)
lyr.resource # specific gsconfig layer object
lyr.resource.fetch() # get the XML from REST
lyr.resource.dom # reference to the parsed XML
from xml.etree.ElementTree import tostring
tostring(lyr.resource.dom)
```

3.2.1.5.7 GeoNode APIs

GeoServer REST interface

This module is a walkthrough the GeoServer REST capabilities and APIs. Here also will be presented and deeply inspected several methods and frameworks to handle with REST APIs and functions.

What you will learn

In this section you will learn:

Introducing REST concepts

REST (REpresentational State Transfer) is a simple approach to web services strongly based on the basic HTTP infrastructure, such as URLs, HTTP methods and HTTP response codes.

The basic elements of a REST service are:

- **Resource:** each business entity is linked to a unique URL that represents it, and allows for its retrieval and eventual modification. In GeoServer such resources are layers, stores, styles and so on
- **Connectedness:** the various resources are linked to one another following significant relationships. For example, in GeoServer a store contains a list of feature types or coverages, a layer is linked to a style and a feature type/coverage, and so on (in other terms, the set of resources is supposed to be crawlable just like a web site).
- **Representation:** each resource can be represented in one or more way. For example in GeoServer resources are normally represented as HTML, XML and JSON.
- **Stateless-ness:** each communication with the server is atomic and not related to the communications happened before or after it. Whatever state needs to be managed needs to be stored as a publicly accessible resource.
- **HTTP methods reuse:** each resource is manipulated via the common HTTP methods each having a common meaning, summarized by the following table

Method	Description
GET	Retrieves the resource in the specified representation. Query parameters are often used to filter the contents of the returned resource, and sometimes to specify the desired representation format.
HEAD	Similar to GET, but instead of returning the full response it returns only the HTTP headers, which might contain information such as the last modification date of the resource
PUT	Stores the representation of a resource at a given URL. Used when the client already knows what the final URL of the resource will be
POST	Creates a new resource by either getting its contents in the request, or having some parameters to compute it. The main different is that the final URL of the created resource is not known to the client, and is returned by the server after creation via a redirect. It is also used to have the server perform certain actions that cannot be encoded as another method, for example, have it send a SMS (assuming creating a resource representing the SMS is not desirable)
DELETE	Destroys the specified resource.

The above results in a web services protocols that is easy to understand, implement and connect to from various languages, and with good scalability characteristics.

The GeoServer rest interface is located at <http://localhost:8083/geoserver/rest>, by default a browser will show resources in HTML format allowing for a simple browsable interface to the GeoServer configuration.

<http://localhost:8083/geoserver/rest>

Follow the links into `workspaces` and then `geosolutions` and switch the format from `.html` to `xml` to see the XML representation:

<http://localhost:8083/geoserver/rest/workspaces/geosolutions.xml>

Using REST module

This section contains a number of examples which illustrate various uses of the REST data configuration API.

The GeoServer REST configuration module uses the REST principles to expose services allowing to edit the catalog, in particular to manage workspaces, stores, layers, styles and groups.

Geoserver Configuration API

- [workspaces](#)
- [namespaces](#)
- [styles](#)
- [layers](#)
- [layergroups](#)
- [reload](#)
- [reset](#)
- [about/manifest](#)
- [about/version](#)
- [settings](#)
- [settings/contact](#)
- [services/wms/settings](#)
- [services/wfs/settings](#)
- [services/wcs/settings](#)
- [templates](#)

Fig. 85: Browsing the REST interface with HTML format

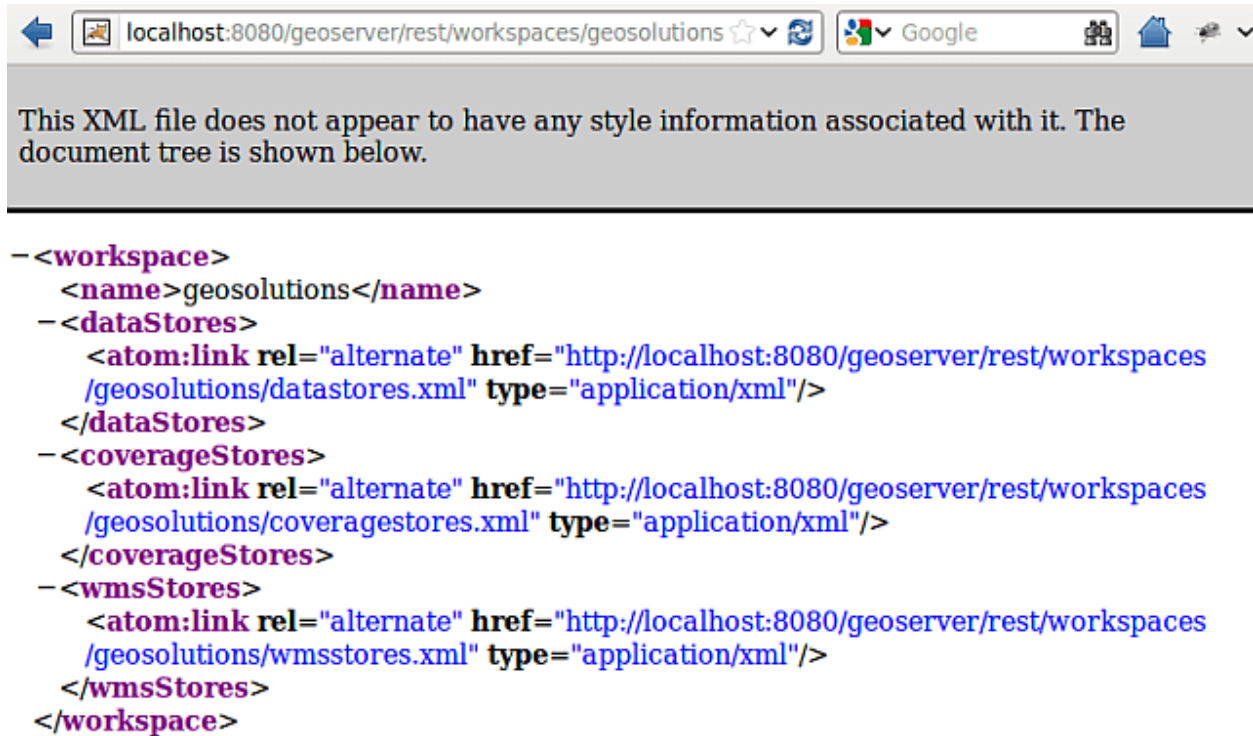


Fig. 86: The GeoSolutions workspace represented as XML

Note: The REST configuration extension has normally to be installed separately, it is not come out of the box.

The examples in this section use the `cURL` utility, which is a handy command line tool for executing HTTP requests and transferring files.

1. Open the Terminal and enter the following command:

```
curl -u admin:geoserver -v -XPOST -H "Content-type: text/xml" -d "
↳ <workspace><name>myworkspace</name></workspace>" http://localhost:8083/
↳ geoserver/rest/workspaces
```

The response should contain the following:

1. Go to the Workspaces section via Web interface to show the new workspace created
2. Get the new created workspace details entering the following:

```
curl -u admin:geoserver -XGET -H "Accept: text/xml" http://
↳ localhost:8083/geoserver/rest/workspaces/myworkspace
```

3. Publish the shapefile pointlands using the myworkspace workspace entering the following
 - Linux:

```

* About to connect() to localhost port 8080 (#0)
* Trying ::1... connected
* Connected to localhost (::1) port 8080 (#0)
* Server auth using Basic with user 'admin'
> POST /geoserver/rest/workspaces HTTP/1.1
> Authorization: Basic YWRtaW46R2Vvcw==
> User-Agent: curl/7.19.5 (i486-pc-linux-gnu) libcurl/7.19.5 OpenSSL/0.9.8g zlib/1.2.3.3 libidn/1.15
> Host: localhost:8080
> Accept: */*
> Content-type: text/xml
> Content-Length: 47
>
< HTTP/1.1 201 Created
< Server: Apache-Coyote/1.1
< Date: Thu, 01 Apr 2010 15:22:57 GMT
< Location: http://localhost:8080/geoserver/rest/workspaces/myworkspace
< Server: Noelios-Restlet-Engine/1.0..8
< Transfer-Encoding: chunked
<
* Connection #0 to host localhost left intact
* Closing connection #0

```

Fig. 87: Create a new workspace via REST

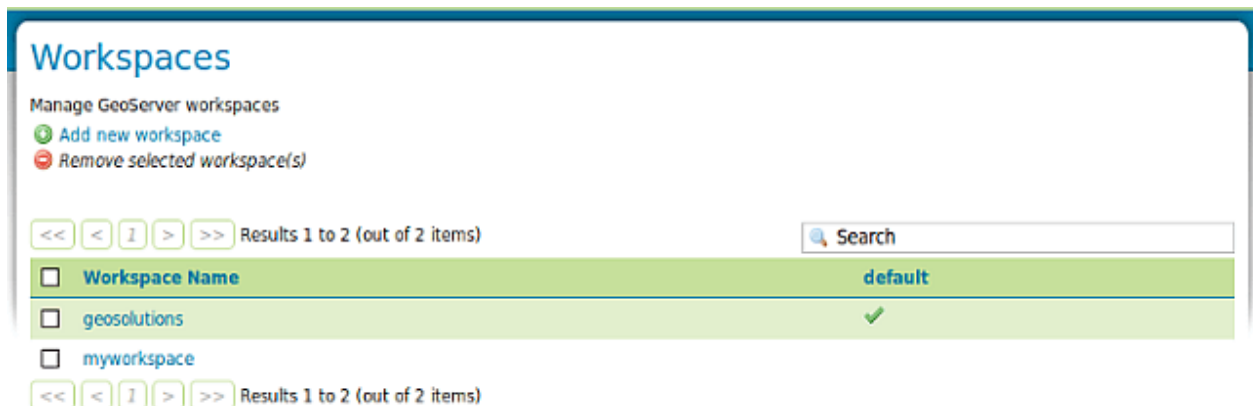


Fig. 88: GET request to obtain new workspace details

```

<workspace>
  <name>myworkspace</name>
  <dataStores>
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate" href="http://localhost:8080/geoserver/rest/workspaces/myworkspace/datastores.xml" type="application/xml"/>
  </dataStores>
  <coverageStores>
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate" href="http://localhost:8080/geoserver/rest/workspaces/myworkspace/coveragestores.xml" type="application/xml"/>
  </coverageStores>
</workspace>

```

Fig. 89: GET request to obtain new workspace details

```
curl -u admin:geoserver -H "Content-type: application/zip" -T ../
↪pointlands.zip http://localhost:8083/geoserver/rest/workspaces/
↪myworkspace/datastores/pointlands/file.shp
```

- Windows:

```
curl -u admin:geoserver -H "Content-type: application/zip" -T ../
↪pointlands.zip http://localhost:8083/geoserver/rest/workspaces/
↪myworkspace/datastores/pointlands/file.shp
```

4. Go to the **Layer Preview** to show the layers in a OpenLayers Map.

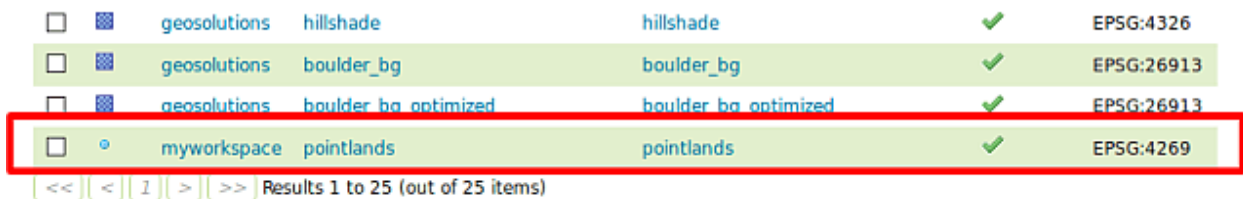


Fig. 90: Showing the new layer created

Note: If you previously followed the security portion of the workshop the layer won't be accessible because the administrator does not have the required roles. Go back in the service security section and remove the rule limiting the GetMap requests.

5. Retrieves the created data store as XML entering the following:

```
curl -u admin:geoserver -XGET http://localhost:8083/geoserver/rest/
↪workspaces/myworkspace/datastores/pointlands.xml
```

```
<dataStore>
  <name>pointlands</name>
  <type>Shapefile</type>
  <enabled>true</enabled>
  <workspace>
    <name>myworkspace</name>
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel=
↪"alternate" href="http://localhost:8083/geoserver/rest/workspaces/
↪myworkspace.xml" type="application/xml"/>
  </workspace>
  <connectionParameters>
    <entry key="url">file:${TRAINING_ROOT}/geoserver_data/data/
↪myworkspace/pointlands/</entry>
    <entry key="namespace">http://myworkspace</entry>
  </connectionParameters>
  <__default>false</__default>
  <featureTypes>
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel=
↪"alternate" href="http://localhost:8083/geoserver/rest/workspaces/
↪myworkspace/datastores/pointlands/featuretypes.xml" type="application/
↪xml"/>
  </featureTypes>
</dataStore>
```

(continues on next page)

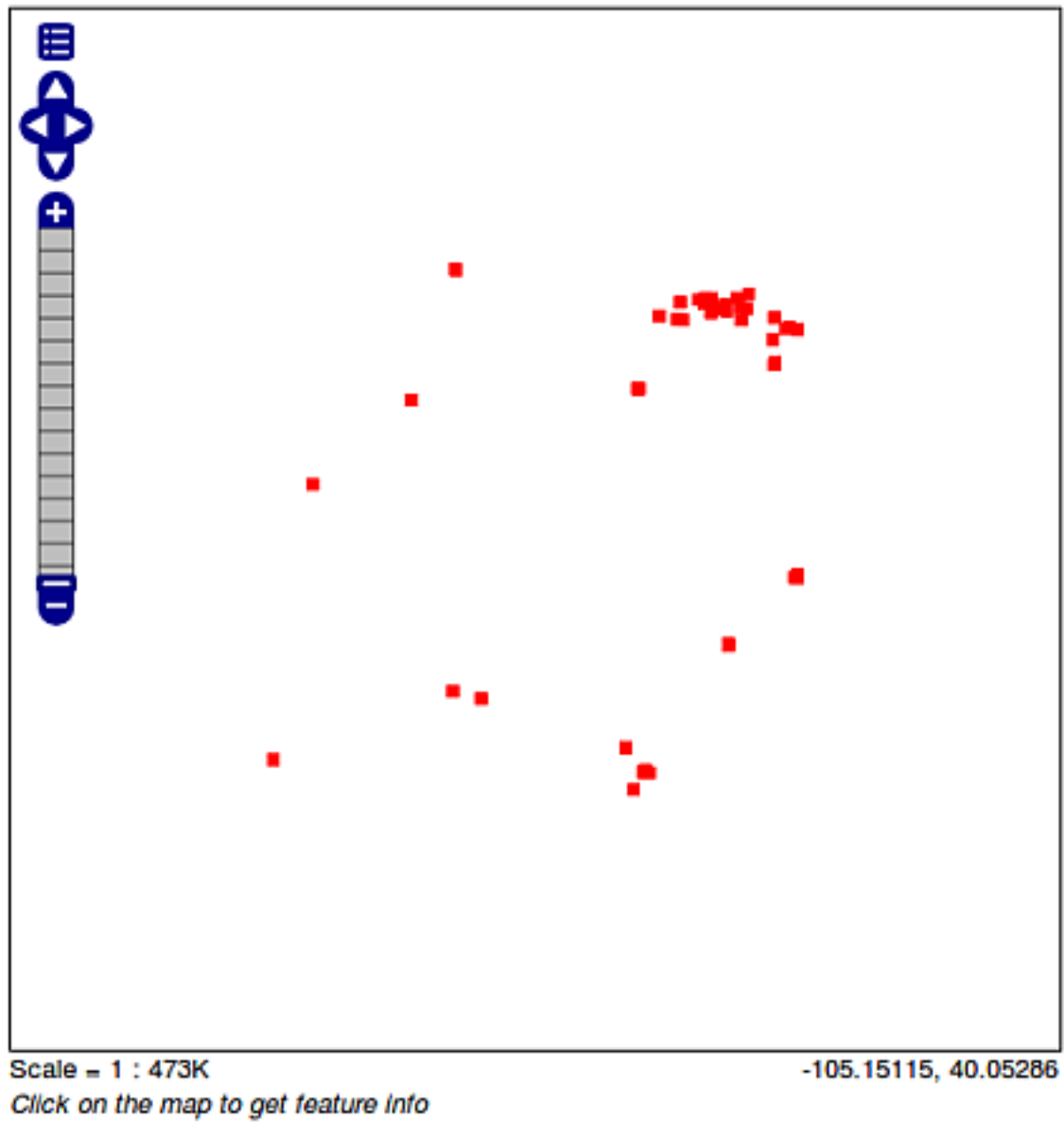


Fig. 91: The new layers created

(continued from previous page)

```
</featureTypes>
</dataStore>
```

Note: By default when a shapefile is uploaded a feature type resource and the associated layer are automatically created.

6. Retrieve the layer as XML entering the following:

```
curl -u admin:geoserver -XGET http://localhost:8083/geoserver/rest/
↳ layers/myworkspace:pointlands.xml
```

```
<layer>
  <name>pointlands</name>
  <type>VECTOR</type>
  <defaultStyle>
    <name>point</name>
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel=
↳ "alternate" href="http://localhost:8083/geoserver/rest/styles/point.xml
↳ " type="application/xml"/>
  </defaultStyle>
  <resource class="featureType">
    <name>pointlands</name>
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel=
↳ "alternate" href="http://localhost:8083/geoserver/rest/workspaces/
↳ myworkspace/datastores/pointlands/featuretypes/pointlands.xml" type=
↳ "application/xml"/>
  </resource>
  <attribution>
    <logoWidth>0</logoWidth>
    <logoHeight>0</logoHeight>
  </attribution>
</layer>
```

Note: When the layer is created a default style named point is assigned to it.

7. Create a new style named landmarks with the following SLD (using the GeoServer Admin UI):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<StyledLayerDescriptor version="1.0.0"
  xsi:schemaLocation="http://www.opengis.net/sld StyledLayerDescriptor.xsd
↳ "
  xmlns="http://www.opengis.net/sld"
  xmlns:ogc="http://www.opengis.net/ogc"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <!-- a Named Layer is the basic building block of an SLD document -->
  <NamedLayer>
    <Name>default_point</Name>
    <UserStyle>
      <!-- Styles can have names, titles and abstracts -->
      <Title>Default Point</Title>
      <Abstract>A sample style that draws a point</Abstract>
```

(continues on next page)

(continued from previous page)

```

        <!-- FeatureTypeStyles describe how to render different_
→ features -->
        <!-- A FeatureTypeStyle for rendering points -->
        <FeatureTypeStyle>
            <Rule>
                <Name>rule1</Name>
                <Title>Red Square</Title>
                <Abstract>A 6 pixel square with a red fill and no_
→ stroke</Abstract>
                <PointSymbolizer>
                    <Graphic>
                        <Mark>
                            <WellKnownName>triangle</WellKnownName>
                            <Stroke>
                                <CssParameter name="stroke">
→ #66FF66</CssParameter>
                                </Stroke>
                                <Fill>
                                    <CssParameter name="fill">#66FF66
→ </CssParameter>
                                </Fill>
                            </Mark>
                            <Size>10</Size>
                        </Graphic>
                    </PointSymbolizer>
                </Rule>
            </FeatureTypeStyle>
        </UserStyle>
    </NamedLayer>
</StyledLayerDescriptor>

```

8. Apply the existing landmarks style to the layer created myworkspace:pointlands (this operation does not overwrite the entire layer definition, updates it instead):

```

curl -u admin:geoserver -XPUT -H "Content-type: text/xml" -d "<layer>
→ <defaultStyle><name>landmarks</name></defaultStyle><enabled>true</
→ enabled></layer>" http://localhost:8083/geoserver/rest/layers/
→ myworkspace:pointlands

```

9. Go to the **Layer Preview** to show the layers with the new landmarks style.

REST configuration examples

This section contains a number of examples which illustrate various uses of the REST configuration API. The examples are grouped by the language or environment used.

cURL

The examples in this section use [cURL](#), a command line tool for executing HTTP requests and transferring files, to generate requests to GeoServer's REST interface. Although the examples are based on cURL, they could be adapted for any HTTP-capable tool or library. Please be aware that cURL doesn't act exactly the same as a web browser. In contrast to Mozilla Firefox or Google Chrome, cURL will not escape special characters in your request string automatically. To make sure, that your requests can be processed correctly, make sure, that characters like parenthesis, commas and the like are escaped before

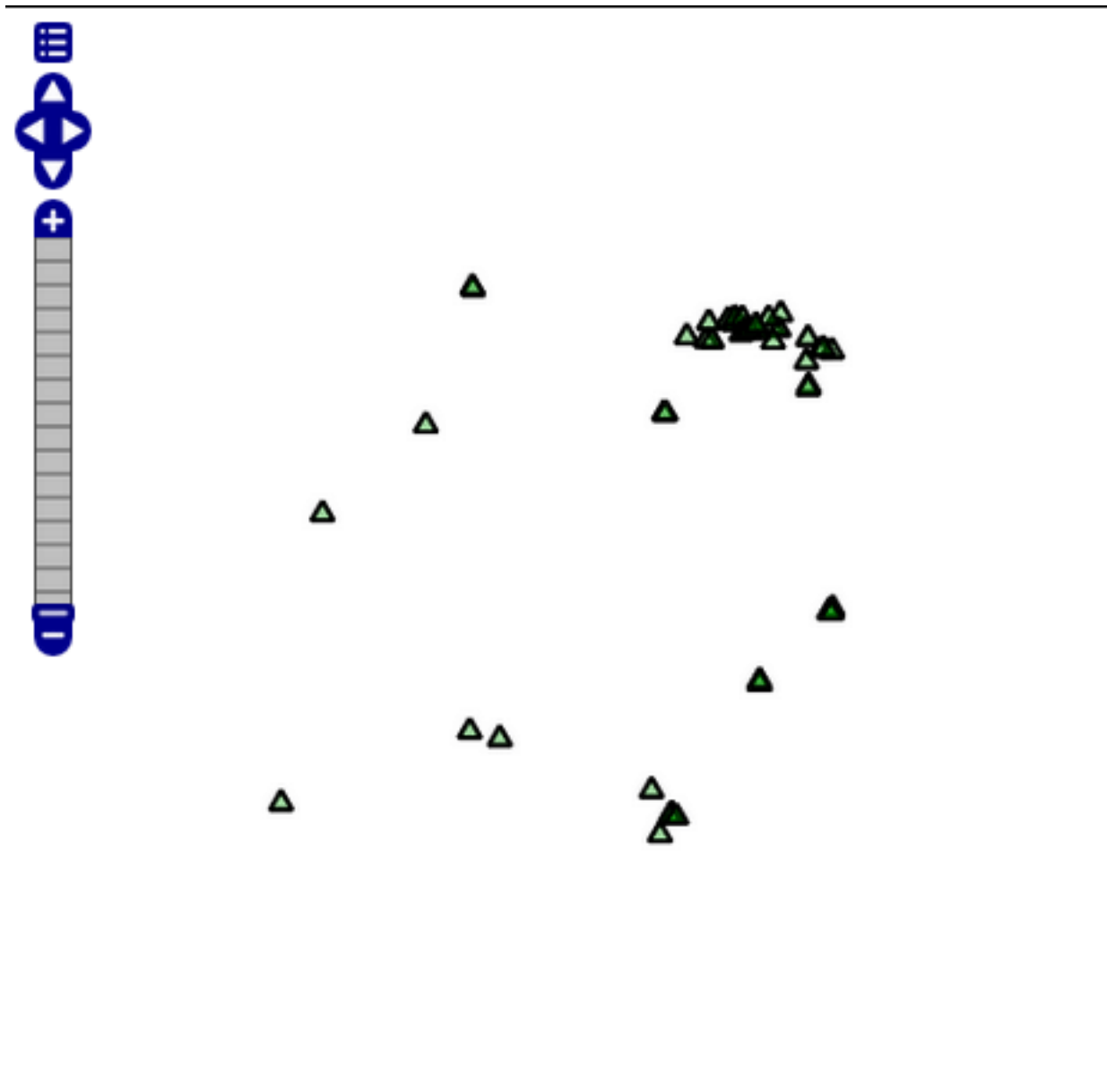


Fig. 92: Viewing the layers with the new created style landmarks

sending them via cURL. If you use libcurl in PHP 5.5 or newer you can prepare the URL-string using the function `curl_escape`. In older versions of PHP `htmlspecialchars` should do the job also.

Adding a new workspace

The following creates a new workspace named “acme” with a POST request:

Note: Each code block below contains a single command that may be extended over multiple lines.

```
curl -v -u admin:geoserver -XPOST -H "Content-type: text/xml" -d "<workspace>
↳<name>acme</name></workspace>" http://localhost/geoserver/rest/workspaces
```

If executed correctly, the response should contain the following:

```
< HTTP/1.1 201 Created
...
< Location: http://localhost/geoserver/rest/workspaces/acme
```

Note the `Location` response header, which specifies the location (URI) of the newly created workspace.

The workspace information can be retrieved as XML with a GET request:

```
curl -v -u admin:geoserver -XGET -H "Accept: text/xml" http://localhost/
↳geoserver/rest/workspaces/acme
```

The response should look like this:

```
<workspace>
  <name>acme</name>
  <dataStores>
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate"
      href="http://localhost/geoserver/rest/workspaces/acme/datastores.xml"
      type="application/xml"/>
  </dataStores>
  <coverageStores>
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate"
      href="http://localhost/geoserver/rest/workspaces/acme/coveragestores.xml"
↳"
      type="application/xml"/>
  </coverageStores>
  <wmsStores>
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate"
      href="http://localhost/geoserver/rest/workspaces/acme/wmsstores.xml"
      type="application/xml"/>
  </wmsStores>
</workspace>
```

This shows that the workspace can contain “dataStores” (for vector data), “coverageStores” (for raster data), and “wmsStores” (for cascaded WMS servers).

Note: The `Accept` header is optional. The following request omits the `Accept` header, but will return the same response as above.

```
curl -v -u admin:geoserver -XGET http://localhost/geoserver/rest/workspaces/
↳acme.xml
```

Uploading a shapefile

In this example, a new store will be created by uploading a shapefile.

The following request uploads a zipped shapefile named `roads.zip` and creates a new store named `roads`.

Note: Each code block below contains a single command that may be extended over multiple lines.

```
curl -v -u admin:geoserver -XPUT -H "Content-type: application/zip" --data-  
↪binary @roads.zip http://localhost/geoserver/rest/workspaces/acme/  
↪datastores/roads/file.shp
```

The `roads` identifier in the URI refers to the name of the store to be created. To create a store named `somethingelse`, the URI would be `http://localhost/geoserver/rest/workspaces/acme/datastores/somethingelse/file.shp`

If executed correctly, the response should contain the following:

```
< HTTP/1.1 201 Created
```

The store information can be retrieved as XML with a GET request:

```
curl -v -u admin:geoserver -XGET http://localhost/geoserver/rest/workspaces/  
↪acme/datastores/roads.xml
```

The response should look like this:

```
<dataStore>  
  <name>roads</name>  
  <type>Shapefile</type>  
  <enabled>true</enabled>  
  <workspace>  
    <name>acme</name>  
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate"  
      href="http://localhost/geoserver/rest/workspaces/acme.xml" type=  
↪"application/xml"/>  
  </workspace>  
  <connectionParameters>  
    <entry key="url">file:/C:/path/to/data_dir/data/acme/roads/</entry>  
    <entry key="namespace">http://acme</entry>  
  </connectionParameters>  
  <__default>false</__default>  
  <featureTypes>  
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate"  
      href="http://localhost/geoserver/rest/workspaces/acme/datastores/roads/  
↪featuretypes.xml"  
      type="application/xml"/>  
  </featureTypes>  
</dataStore>
```

By default when a shapefile is uploaded, a feature type is automatically created. The feature type information can be retrieved as XML with a GET request:

```
curl -v -u admin:geoserver -XGET http://localhost/geoserver/rest/workspaces/
↳acme/datastores/roads/featuretypes/tiger_roads.xml
```

If executed correctly, the response will be:

```
<featureType>
  <name>roads</name>
  <nativeName>roads</nativeName>
  <namespace>
    <name>acme</name>
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate"
      href="http://localhost/geoserver/rest/namespaces/acme.xml" type=
↳"application/xml"/>
    </namespace>
    ...
  </featureType>
```

The remainder of the response consists of layer metadata and configuration information.

Note: Notice that the name of the Layer (and of the FeatureType) corresponds to the physical name of the ShapeFile contained into the archive.

Adding an existing shapefile

In the previous example a shapefile was uploaded directly to GeoServer by sending a zip file in the body of a PUT request. This example shows how to publish a shapefile that already exists on the server.

Consider a directory on the server /data/shapefiles that contains the shapefile `rivers.shp`. The following adds a new store for the shapefile:

Note: In order to execute the exercise, create a folder `shapefiles` somewhere on the server and extract there the `shapefiles.zip`.

```
curl -v -u admin:geoserver -XPUT -H "Content-type: text/plain" -d "file:///
↳home/geonode/data/shapefiles/rivers.shp" http://localhost/geoserver/rest/
↳workspaces/acme/datastores/rivers/external.shp
```

The `external.shp` part of the request URI indicates that the file is coming from outside the catalog.

If executed correctly, the response should contain the following:

```
< HTTP/1.1 201 Created
```

The shapefile will be added to the existing store and published as a layer.

To verify the contents of the store, execute a GET request. Since the XML response only provides details about the store itself without showing its contents, execute a GET request for HTML:

```
curl -v -u admin:geoserver -XGET http://localhost/geoserver/rest/workspaces/
↳acme/datastores/rivers.html
```

Adding a directory of existing shapefiles

This example shows how to load and create a store that contains a number of shapefiles, all with a single operation. This example is very similar to the example above of adding a single shapefile.

Consider a directory on the server `/data/shapefiles` that contains multiple shapefiles. The following adds a new store for the directory.

Note: In order to execute the exercise, create a folder `shapefiles` somewhere on the server and extract there the `shapefiles.zip`.

```
curl -v -u admin:geoserver -XPUT -H "Content-type: text/plain" -d "file:///
↪home/geonode/data/shapefiles/" "http://localhost/geoserver/rest/workspaces/
↪acme/datastores/shapefiles/external.shp?configure=all"
```

Note the `configure=all` query string parameter, which sets each shapefile in the directory to be loaded and published.

If executed correctly, the response should contain the following:

```
< HTTP/1.1 201 Created
```

To verify the contents of the store, execute a GET request. Since the XML response only provides details about the store itself without showing its contents, execute a GET request for HTML:

```
curl -v -u admin:geoserver -XGET http://localhost/geoserver/rest/workspaces/
↪acme/datastores/shapefiles.html
```

Creating a layer style

This example will create a new style on the server and populate it the contents of a local SLD file.

The following creates a new style named `roads_style`:

Note: Each code block below contains a single command that may be extended over multiple lines.

```
curl -v -u admin:geoserver -XPOST -H "Content-type: text/xml" -d "<style>
↪<name>roads_style</name><filename>roads.sld</filename></style>" http://
↪localhost/geoserver/rest/styles
```

If executed correctly, the response should contain the following:

```
< HTTP/1.1 201 Created
```

This request uploads a file called `roads.sld` file and populates the `roads_style` with its contents:

```
curl -v -u admin:geoserver -XPUT -H "Content-type: application/vnd.ogc.
↪sld+xml" -d @roads.sld http://localhost/geoserver/rest/styles/roads_style
```

If executed correctly, the response should contain the following:

```
< HTTP/1.1 200 OK
```

The SLD itself can be downloaded through a GET request:

```
curl -v -u admin:geoserver -XGET http://localhost/geoserver/rest/styles/
↳ roads_style.sld
```

Changing a layer style

This example will alter a layer style. Prior to making any changes, it is helpful to view the existing configuration for a given layer.

Note: Each code block below contains a single command that may be extended over multiple lines.

The following retrieves the “acme:roads” layer information as XML:

```
curl -v -u admin:geoserver -XGET "http://localhost/geoserver/rest/layers/
↳ acme:tiger_roads.xml"
```

The response in this case would be:

```
<layer>
  <name>tiger_roads</name>
  <type>VECTOR</type>
  <defaultStyle>
    <name>line</name>
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate" href=
↳ "http://localhost/geoserver/rest/styles/line.xml" type="application/xml"/>
  </defaultStyle>
  <resource class="featureType">
    <name>tiger_roads</name>
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate" href=
↳ "http://localhost/geoserver/rest/workspaces/acme/datastores/roads/
↳ featuretypes/tiger_roads.xml" type="application/xml"/>
  </resource>
  <attribution>
    <logoWidth>0</logoWidth>
    <logoHeight>0</logoHeight>
  </attribution>
</layer>
```

When the layer is created, GeoServer assigns a default style to the layer that matches the geometry of the layer. In this case a style named `line` is assigned to the layer. This style can be viewed with a WMS request:

```
http://localhost/geoserver/wms/reflect?layers=acme:tiger_roads
```

In this next example a new style will be created called `roads_style` and assigned to the “acme:roads” layer:

```
curl -v -u admin:geoserver -XPUT -H "Content-type: text/xml" -d "<layer>
↳ <defaultStyle><name>roads_style</name></defaultStyle></layer>" http://
↳ localhost/geoserver/rest/layers/acme:tiger_roads
```

If executed correctly, the response should contain the following:

```
< HTTP/1.1 200 OK
```

The new style can be viewed with the same WMS request as above:

```
http://localhost/geoserver/wms/reflect?layers=acme:tiger_roads
```

Note that if you want to upload the style in a workspace (ie, not making it a global style), and then assign this style to a layer in that workspace, you need first to create the style in the given workspace:

```
curl -u admin:geoserver -XPOST -H 'Content-type: text/xml' -d '<style><name>
↪roads_style</name><filename>roads.sld</filename></style>' http://localhost/
↪geoserver/rest/workspaces/acme/styles
```

Upload the file within the workspace:

```
curl -u admin:geoserver -XPUT -H 'Content-type: application/vnd.ogc.sld+xml'
↪-d @roads.sld http://localhost/geoserver/rest/workspaces/acme/styles/roads_
↪style
```

And finally apply that style to the layer. Note the use of the `<workspace>` tag in the XML:

```
curl -u admin:geoserver -XPUT -H 'Content-type: text/xml' -d '<layer>
↪<defaultStyle><name>roads_style</name><workspace>acme</workspace></
↪defaultStyle></layer>' http://localhost/geoserver/rest/layers/acme:tiger_
↪roads
```

Adding a PostGIS database

In this example, a PostGIS database named `nyc` will be added as a new store.

Warning: This section assumes that a PostGIS database named `nyc` is present on the local system and is accessible by the user `bob`.

Note: In order to create and setup the database locally, follow the instructions at `setup_nyc_db`

Create a new text file and add the following content to it. This will represent the new store. Save the file as `nycDataStore.xml`.

```
<dataStore>
  <name>nyc</name>
  <connectionParameters>
    <host>localhost</host>
    <port>5432</port>
    <database>nyc</database>
    <user>bob</user>
    <passwd>postgres</passwd>
    <dbtype>postgis</dbtype>
  </connectionParameters>
</dataStore>
```

The following will add the new PostGIS store to the GeoServer catalog:

Note: Each code block below contains a single command that may be extended over multiple lines.

```
curl -v -u admin:geoserver -XPOST -T nycDataStore.xml -H "Content-type: text/
↪xml" http://localhost/geoserver/rest/workspaces/acme/datastores
```

If executed correctly, the response should contain the following:

```
< HTTP/1.1 200 OK
```

The store information can be retrieved as XML with a GET request:

```
curl -v -u admin:geoserver -XGET http://localhost/geoserver/rest/workspaces/
↪acme/datastores/nyc.xml
```

The response should look like the following:

```
<dataStore>
  <name>nyc</name>
  <type>PostGIS</type>
  <enabled>true</enabled>
  <workspace>
    <name>acme</name>
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate"
      href="http://localhost/geoserver/rest/workspaces/acme.xml" type=
↪"application/xml"/>
  </workspace>
  <connectionParameters>
    <entry key="port">5432</entry>
    <entry key="dbtype">postgis</entry>
    <entry key="host">localhost</entry>
    <entry key="user">bob</entry>
    <entry key="database">nyc</entry>
    <entry key="namespace">http://acme</entry>
  </connectionParameters>
  <__default>>false</__default>
  <featureTypes>
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate"
      href="http://localhost/geoserver/rest/workspaces/acme/datastores/nyc/
↪featuretypes.xml"
      type="application/xml"/>
  </featureTypes>
</dataStore>
```

Adding a PostGIS table

In this example, a table from the PostGIS database created in the previous example will be added as a featuretype.

Warning: This example assumes the table has already been created and the `tiger_roads` Layer deleted in case you have executed the previous steps.

The following adds the table `tiger_roads` as a new feature type:

Note: Each code block below contains a single command that may be extended over multiple lines.

```
curl -v -u admin:geoserver -XPOST -H "Content-type: text/xml" -d "  
↪<featureType><name>tiger_roads</name></featureType>" http://localhost/  
↪geoserver/rest/workspaces/acme/datastores/nyc/featuretypes
```

The featuretype information can be retrieved as XML with a GET request:

```
curl -v -u admin:geoserver -XGET http://localhost/geoserver/rest/workspaces/  
↪acme/datastores/nyc/featuretypes/tiger_roads.xml
```

This layer can viewed with a WMS GetMap request:

```
http://localhost/geoserver/wms/reflect?layers=acme:tiger_roads
```

Creating a PostGIS table

In the previous example, a new feature type was added based on a PostGIS table that already existed in the database. The following example will not only create a new feature type in GeoServer, but will also create the PostGIS table itself.

Create a new text file and add the following content to it. This will represent the definition of the new feature type and table. Save the file as `annotations.xml`.

```
<featureType>  
  <name>annotations</name>  
  <nativeName>annotations</nativeName>  
  <title>Annotations</title>  
  <srs>EPSG:4326</srs>  
  <attributes>  
    <attribute>  
      <name>the_geom</name>  
      <binding>com.vividsolutions.jts.geom.Point</binding>  
    </attribute>  
    <attribute>  
      <name>description</name>  
      <binding>java.lang.String</binding>  
    </attribute>  
    <attribute>  
      <name>timestamp</name>  
      <binding>java.util.Date</binding>  
    </attribute>  
  </attributes>  
</featureType>
```

This request will perform the feature type creation and add the new table:

Note: Each code block below contains a single command that may be extended over multiple lines.

```
curl -v -u admin:geoserver -XPOST -T annotations.xml -H "Content-type: text/  
↪xml" http://localhost/geoserver/rest/workspaces/acme/datastores/nyc/  
↪featuretypes
```

The result is a new, empty table named “annotations” in the “nyc” database, fully configured as a feature type.

The featuretype information can be retrieved as XML with a GET request:

```
curl -v -u admin:geoserver -XGET http://localhost/geoserver/rest/workspaces/
↳acme/datastores/nyc/featuretypes/annotations.xml
```

Creating a layer group

Warning: This example assumes the tables has already been created and the `tiger_roads`, `poly_landmarks`, `poi`, `giant_polygon` Layers have been created.

```
$ curl -v -u admin:geoserver -XPOST -H "Content-type: text/xml" -d "
↳<featureType><name>giant_polygon</name></featureType>" http://localhost/
↳geoserver/rest/workspaces/acme/datastores/nyc/featuretypes

$ curl -v -u admin:geoserver -XPOST -H "Content-type: text/xml" -d "
↳<featureType><name>poi</name></featureType>" http://localhost/geoserver/
↳rest/workspaces/acme/datastores/nyc/featuretypes

$ curl -v -u admin:geoserver -XPOST -H "Content-type: text/xml" -d "
↳<featureType><name>poly_landmarks</name></featureType>" http://localhost/
↳geoserver/rest/workspaces/acme/datastores/nyc/featuretypes
```

In this example, a layer group will be created, based on layers that already exist on the server.

Create a new text file and add the following content to it. This file will represent the definition of the new layer group. Save the file as `nycLayerGroup.xml`.

```
<layerGroup>
  <name>nyc</name>
  <layers>
    <layer>poi</layer>
    <layer>poly_landmarks</layer>
    <layer>tiger_roads</layer>
  </layers>
  <styles>
    <style>point</style>
    <style>polygon</style>
    <style>roads_style</style>
  </styles>
</layerGroup>
```

The following request creates the new layer group:

Note: Each code block below contains a single command that may be extended over multiple lines.

```
curl -v -u admin:geoserver -XPOST -d @nycLayerGroup.xml -H "Content-type:
↳text/xml" http://localhost/geoserver/rest/layergroups
```

Note: The argument `-d@filename.xml` in this example is used to send a file as the body of an HTTP

request with a POST method. The argument `-T filename.xml` used in the previous example was used to send a file as the body of an HTTP request with a PUT method.

This layer group can be viewed with a WMS GetMap request:

```
http://localhost/geoserver/wms/reflect?layers=nyc&format=openlayers
```

Retrieving component versions

This example shows how to retrieve the versions of the main components: GeoServer, GeoTools, and GeoWebCache:

Note: The code block below contains a single command that is extended over multiple lines.

```
curl -v -u admin:geoserver -XGET -H "Accept: text/xml" http://localhost/  
↪geoserver/rest/about/version.xml
```

The response will look something like this:

```
<about>  
  <resource name="GeoServer">  
    <Build-Timestamp>04-Aug-2015 11:00</Build-Timestamp>  
    <Git-Revision>bca94d09e2e18839814a4b663ba8b0fca2130e47</Git-Revision>  
    <Version>2.7-SNAPSHOT</Version>  
  </resource>  
  <resource name="GeoTools">  
    <Build-Timestamp>29-Jul-2015 10:13</Build-Timestamp>  
    <Git-Revision>f50be97a039cd06d43a87ec3cc101626f0ac9fd2</Git-Revision>  
    <Version>13-SNAPSHOT</Version>  
  </resource>  
  <resource name="GeoWebCache">  
    <Git-Revision>f6e0d39c29c2317d2839c52a84676935e5b046cf/  
↪f6e0d39c29c2317d2839c52a84676935e5b046cf</Git-Revision>  
    <Version>1.7-SNAPSHOT</Version>  
  </resource>  
</about>
```

Retrieving manifests

This collection of examples shows how to retrieve the full manifest and subsets of the manifest as known to the ClassLoader.

Note: The code block below contains a single command that is extended over multiple lines.

```
curl -v -u admin:geoserver -XGET -H "Accept: text/xml" http://localhost/  
↪geoserver/rest/about/manifest.xml
```

The result will be a very long list of manifest information. While this can be useful, it is often desirable to filter this list.

Filtering over resource name

It is possible to filter over resource names using regular expressions. This example will retrieve only resources where the name attribute matches `gwc-.*`:

Note: The code block below contains a single command that is extended over multiple lines.

```
curl -v -u admin:geoserver -XGET -H "Accept: text/xml" http://localhost/  
↪geoserver/rest/about/manifest.xml?manifest=gwc-.*
```

The result will look something like this (edited for brevity):

```
<about>  
  <resource name="gwc-2.3.0">  
    ...  
  </resource>  
  <resource name="gwc-core-1.4.0">  
    ...  
  </resource>  
  <resource name="gwc-diskquota-core-1.4.0">  
    ...  
  </resource>  
  <resource name="gwc-diskquota-jdbc-1.4.0">  
    ...  
  </resource>  
  <resource name="gwc-georss-1.4.0">  
    ...  
  </resource>  
  <resource name="gwc-gmaps-1.4.0">  
    ...  
  </resource>  
  <resource name="gwc-kml-1.4.0">  
    ...  
  </resource>  
  <resource name="gwc-rest-1.4.0">  
    ...  
  </resource>  
  <resource name="gwc-tms-1.4.0">  
    ...  
  </resource>  
  <resource name="gwc-ve-1.4.0">  
    ...  
  </resource>  
  <resource name="gwc-wms-1.4.0">  
    ...  
  </resource>  
  <resource name="gwc-wmts-1.4.0">  
    ...  
  </resource>  
</about>
```

Filtering over resource properties

Filtering is also available over resulting resource properties. This example will retrieve only resources with a property equal to GeoServerModule.

Note: The code blocks below contain a single command that is extended over multiple lines.

```
curl -u admin:geoserver -XGET -H "Accept: text/xml" http://localhost/  
↳geoserver/rest/about/manifest.xml?key=GeoServerModule
```

The result will look something like this (edited for brevity):

```
<about>  
  <resource name="control-flow-2.3.0">  
    <GeoServerModule>extension</GeoServerModule>  
    ...  
  </resource>  
  ...  
  <resource name="wms-2.3.0">  
    <GeoServerModule>core</GeoServerModule>  
    ...  
  </resource>  
</about>
```

It is also possible to filter against both property and value. To retrieve only resources where a property named GeoServerModule has a value equal to extension, append the above request with &value=extension:

```
curl -u admin:geoserver -XGET -H "Accept: text/xml" http://localhost/  
↳geoserver/rest/about/manifest.xml?key=GeoServerModule&value=extension
```

Uploading and modifying a image mosaic

The following command uploads a polyphemus.zip file containing the definition of a mosaic (along with at least one granule of the mosaic to initialize the resolutions, overviews and the like) and will configure all the coverages in it as new layers.

Note: The code blocks below contain a single command that is extended over multiple lines.

```
curl -u admin:geoserver -XPUT -H "Content-type:application/zip" --data-  
↳binary @polyphemus.zip http://localhost/geoserver/rest/workspaces/topp/  
↳coveragestores/polyphemus/file.imagemosaic
```

The following instead instructs the mosaic to harvest (or re-harvest) a single file into the mosaic, collecting its properties and updating the mosaic index:

```
curl -v -u admin:geoserver -XPOST -H "Content-type: text/plain" -d "file:///   
↳path/to/the/file/polyphemus_20130302.nc" "http://localhost/geoserver/rest/  
↳workspaces/topp/coveragestores/poly-incremental/external.imagemosaic"
```

Harvesting can also be directed towards a whole directory, as follows:

```
curl -v -u admin:geoserver -XPOST -H "Content-type: text/plain" -d "file:///
↳path/to/the/mosaic/folder" "http://localhost/geoserver/rest/workspaces/
↳topp/coveragestores/poly-incremental/external.imagemosaic"
```

The image mosaic index structure can be retrieved using something like:

```
curl -v -u admin:geoserver -XGET "http://localhost/geoserver/rest/workspaces/
↳topp/coveragestores/polyphemus-v1/coverages/NO2/index.xml"
```

Which will result in the following:

```
<Schema>
<attributes>
  <Attribute>
    <name>the_geom</name>
    <minOccurs>0</minOccurs>
    <maxOccurs>1</maxOccurs>
    <nillable>true</nillable>
    <binding>com.vividsolutions.jts.geom.Polygon</binding>
  </Attribute>
  <Attribute>
    <name>location</name>
    <minOccurs>0</minOccurs>
    <maxOccurs>1</maxOccurs>
    <nillable>true</nillable>
    <binding>java.lang.String</binding>
  </Attribute>
  <Attribute>
    <name>imageindex</name>
    <minOccurs>0</minOccurs>
    <maxOccurs>1</maxOccurs>
    <nillable>true</nillable>
    <binding>java.lang.Integer</binding>
  </Attribute>
  <Attribute>
    <name>time</name>
    <minOccurs>0</minOccurs>
    <maxOccurs>1</maxOccurs>
    <nillable>true</nillable>
    <binding>java.sql.Timestamp</binding>
  </Attribute>
  <Attribute>
    <name>elevation</name>
    <minOccurs>0</minOccurs>
    <maxOccurs>1</maxOccurs>
    <nillable>true</nillable>
    <binding>java.lang.Double</binding>
  </Attribute>
  <Attribute>
    <name>fileDate</name>
    <minOccurs>0</minOccurs>
    <maxOccurs>1</maxOccurs>
    <nillable>true</nillable>
    <binding>java.sql.Timestamp</binding>
  </Attribute>
  <Attribute>
    <name>updated</name>
```

(continues on next page)

(continued from previous page)

```

    <minOccurs>0</minOccurs>
    <maxOccurs>1</maxOccurs>
    <nillable>true</nillable>
    <binding>java.sql.Timestamp</binding>
  </Attribute>
</attributes>
  <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate" href=
  ↪ "http://localhost/geoserver/rest/workspaces/topp/coveragestores/polyphemus-
  ↪ v1/coverages/NO2/index/granules.xml" type="application/xml"/>
</Schema>

```

Listing the existing granules can be performed as follows:

```

curl -v -u admin:geoserver -XGET "http://localhost/geoserver/rest/workspaces/
  ↪ topp/coveragestores/polyphemus-v1/coverages/NO2/index/granules.xml?limit=2"

```

This will result in a GML description of the granules, as follows:

```

<?xml version="1.0" encoding="UTF-8"?>
<wfs:FeatureCollection xmlns:gf="http://www.geoserver.org/rest/granules"
  ↪ xmlns:ogc="http://www.opengis.net/ogc" xmlns:wfs="http://www.opengis.net/
  ↪ wfs" xmlns:gml="http://www.opengis.net/gml">
  <gml:boundedBy>
    <gml:Box srsName="http://www.opengis.net/gml/srs/epsg.xml#4326">
      <gml:coord>
        <gml:X>5.0</gml:X>
        <gml:Y>45.0</gml:Y>
      </gml:coord>
      <gml:coord>
        <gml:X>14.875</gml:X>
        <gml:Y>50.9375</gml:Y>
      </gml:coord>
    </gml:Box>
  </gml:boundedBy>
  <gml:featureMember>
    <gf:NO2 fid="NO2.1">
      <gf:the_geom>
        <gml:Polygon>
          <gml:outerBoundaryIs>
            <gml:LinearRing>
              <gml:coordinates>5.0,45.0 5.0,50.9375 14.875,50.9375 14.875,45.
  ↪ 0 5.0,45.0</gml:coordinates>
            </gml:LinearRing>
          </gml:outerBoundaryIs>
        </gml:Polygon>
      </gf:the_geom>
      <gf:location>polyphemus_20130301.nc</gf:location>
      <gf:imageindex>336</gf:imageindex>
      <gf:time>2013-03-01T00:00:00Z</gf:time>
      <gf:elevation>10.0</gf:elevation>
      <gf:fileDate>2013-03-01T00:00:00Z</gf:fileDate>
      <gf:updated>2013-04-11T10:54:31Z</gf:updated>
    </gf:NO2>
  </gml:featureMember>
  <gml:featureMember>
    <gf:NO2 fid="NO2.2">
      <gf:the_geom>

```

(continues on next page)

(continued from previous page)

```

    <gml:Polygon>
      <gml:outerBoundaryIs>
        <gml:LinearRing>
          <gml:coordinates>5.0,45.0 5.0,50.9375 14.875,50.9375 14.875,45.
↪0 5.0,45.0</gml:coordinates>
        </gml:LinearRing>
      </gml:outerBoundaryIs>
    </gml:Polygon>
  </gf:the_geom>
  <gf:location>polyphemus_20130301.nc</gf:location>
  <gf:imageindex>337</gf:imageindex>
  <gf:time>2013-03-01T00:00:00Z</gf:time>
  <gf:elevation>35.0</gf:elevation>
  <gf:fileDate>2013-03-01T00:00:00Z</gf:fileDate>
  <gf:updated>2013-04-11T10:54:31Z</gf:updated>
</gf:NO2>
</gml:featureMember>
</wfs:FeatureCollection>

```

Removing all the granules originating from a particular file (a NetCDF file can contain many) can be done as follows:

```

curl -v -u admin:geoserver -XDELETE "http://localhost/geoserver/rest/
↪workspaces/topp/coveragestores/polyphemus-v1/coverages/NO2/index/granules.
↪xml?filter=location='polyphemus_20130301.nc'"

```

Creating an empty mosaic and harvest granules

The next command uploads an `empty.zip` file. This archive contains the definition of an empty mosaic (no granules in this case) through the following files:

```

datastore.properties (the postgis datastore connection params)
indexer.xml (The mosaic Indexer, note the CanBeEmpty=true parameter)
polyphemus-test.xml (The auxiliary file used by the NetCDF reader to parse_
↪schemas and tables)

```

Note: Make sure to update the `datastore.properties` file with your connection params and refresh the zip when done, before uploading it.

Note: The code blocks below contain a single command that is extended over multiple lines.

Note: The `configure=none` parameter allows for future configuration after harvesting

```

curl -u admin:geoserver -XPUT -H "Content-type:application/zip" --data-
↪binary @empty.zip http://localhost/geoserver/rest/workspaces/topp/
↪coveragestores/empty/file.imagemosaic?configure=none

```

The following instead instructs the mosaic to harvest a single `polyphemus_20120401.nc` file into the mosaic, collecting its properties and updating the mosaic index:

```
curl -v -u admin:geoserver -XPOST -H "Content-type: text/plain" -d "file:///
↳path/to/the/file/polyphemus_20120401.nc" "http://localhost/geoserver/rest/
↳workspaces/topp/coveragestores/empty/external.imagemosaic"
```

Once done you can get the list of coverages/granules available on that store.

```
curl -v -u admin:geoserver -XGET "http://localhost/geoserver/rest/workspaces/
↳topp/coveragestores/empty/coverages.xml?list=all"
```

Which will result in the following:

```
<list>
  <coverageName>NO2</coverageName>
  <coverageName>O3</coverageName>
</list>
```

Next step is configuring ONCE for coverage (as an instance NO2), an available coverage.

```
curl -v -u admin:geoserver -XPOST -H "Content-type: text/xml" -d @"path/to/
↳coverageconfig.xml" "http://localhost/geoserver/rest/workspaces/topp/
↳coveragestores/empty/coverages"
```

Where coverageconfig.xml may look like this

```
<coverage>
  <name>NO2</name>
</coverage>
```

Note: When specifying only the coverage name, the coverage will be automatically configured

Master Password Change

The master password can be fetched with a GET request.

```
curl -v -u admin:geoserver -XGET http://localhost/geoserver/rest/security/
↳masterpw.xml
```

A generated master password may be -"}3a^Kh. Next step is creating an XML file.

File changes.xml

```
<masterPassword>
  <oldMasterPassword>-"}3a^Kh</oldMasterPassword>
  <newMasterPassword>geoserver1</newMasterPassword>
</masterPassword>
```

Changing the master password using the file:

```
curl -v -u admin:geoserver -XPUT -H "Content-type: text/xml" -d @change.xml
↳http://localhost/geoserver/rest/security/masterpw.xml
```

PHP

The examples in this section use the server-side scripting language [PHP](#), a popular language for dynamic webpages. PHP has [cURL functions](#), as well as [XML functions](#), making it a convenient method for performing batch processing through the GeoServer REST interface. The following scripts execute single requests, but can be easily modified with looping structures to perform batch processing.

Note: In order to execute the examples just copy the script content into a `test.php` file and execute the following command:

```
$ php test.php
```

POST with PHP/cURL

The following script attempts to add a new workspace.

```
<?php
// Open log file
$logfh = fopen("GeoserverPHP.log", 'w') or die("can't open log file");

// Initiate cURL session
$service = "http://localhost:8080/geoserver/"; // replace with your URL
$request = "rest/workspaces"; // to add a new workspace
$url = $service . $request;
$ch = curl_init($url);

// Optional settings for debugging
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true); //option to return string
curl_setopt($ch, CURLOPT_VERBOSE, true);
curl_setopt($ch, CURLOPT_STDERR, $logfh); // logs curl messages

//Required POST request settings
curl_setopt($ch, CURLOPT_POST, true);
$passwordStr = "admin:geoserver"; // replace with your username:password
curl_setopt($ch, CURLOPT_USERPWD, $passwordStr);

//POST data
curl_setopt($ch, CURLOPT_HTTPHEADER,
    array("Content-type: application/xml"));
$xmlStr = "<workspace><name>test_ws</name></workspace>";
curl_setopt($ch, CURLOPT_POSTFIELDS, $xmlStr);

//POST return code
$successCode = 201;

$buffer = curl_exec($ch); // Execute the curl request

// Check for errors and process results
$info = curl_getinfo($ch);
if ($info['http_code'] != $successCode) {
    $msgStr = "# Unsuccessful cURL request to ";
    $msgStr .= $url." [".$info['http_code']. "]\n";
    fwrite($logfh, $msgStr);
}
```

(continues on next page)

(continued from previous page)

```

    } else {
        $msgStr = "# Successful cURL request to ".$url."\n";
        fwrite($logfh, $msgStr);
    }
    fwrite($logfh, $buffer."\n");

    curl_close($ch); // free resources if curl handle will not be reused
    fclose($logfh); // close logfile

?>

```

The logfile should look something like:

```

* About to connect() to www.example.com port 80 (#0)
* Trying 123.456.78.90... * connected
* Connected to www.example.com (123.456.78.90) port 80 (#0)
* Server auth using Basic with user 'admin'
> POST /geoserver/rest/workspaces HTTP/1.1
Authorization: Basic sDsdFjkLDFOiedlsdkfj
Host: www.example.com
Accept: */*
Content-type: application/xml
Content-Length: 43

< HTTP/1.1 201 Created
< Date: Fri, 21 May 2010 15:44:47 GMT
< Server: Apache-Coyote/1.1
< Location: http://www.example.com/geoserver/rest/workspaces/test_ws
< Content-Length: 0
< Content-Type: text/plain
<
* Connection #0 to host www.example.com left intact
# Successful cURL request to http://www.example.com/geoserver/rest/workspaces

* Closing connection #0

```

If the cURL request fails, a code other than 201 will be returned. Here are some possible values:

Code	Meaning
0	Couldn't resolve host; possibly a typo in host name
201	Successful POST
30x	Redirect; possibly a typo in the URL
401	Invalid username or password
405	Method not Allowed: check request syntax
500	GeoServer is unable to process the request, e.g. the workspace already exists, the XML is malformed, ...

For other codes see [cURL Error Codes](#) and [HTTP Codes](#).

GET with PHP/cURL

The script above can be modified to perform a GET request to obtain the names of all workspaces by replacing the code blocks for required settings, data and return code with the following:

```
<?php
// Required GET request settings
// curl_setopt($ch, CURLOPT_GET, True); // CURLOPT_GET is True by default

//GET data
curl_setopt($ch, CURLOPT_HTTPHEADER, array("Accept: application/xml"));

//GET return code
$successCode = 200;
?>
```

The logfile should now include lines like:

```
> GET /geoserver/rest/workspaces HTTP/1.1
< HTTP/1.1 200 OK
```

DELETE with PHP/cURL

To delete the (empty) workspace we just created, the script is modified as follows:

```
<?php
$request = "rest/workspaces/test_ws"; // to delete this workspace
?>
```

```
<?php
//Required DELETE request settings
curl_setopt($ch, CURLOPT_CUSTOMREQUEST, "DELETE");
$passwordStr = "admin:geoserver"; // replace with your username:password
curl_setopt($ch, CURLOPT_USERPWD, $passwordStr);

//DELETE data
curl_setopt($ch, CURLOPT_HTTPHEADER,
    array("Content-type: application/atom+xml"));

//DELETE return code
$successCode = 200;
?>
```

The log file will include lines like:

```
> DELETE /geoserver/rest/workspaces/test_ws HTTP/1.1
< HTTP/1.1 200 OK
```

Python

We are looking for volunteers to flesh out this section with examples.

In the meantime, anyone looking to do python scripting of the GeoServer REST config API should use `gsconfig.py`. It is quite capable, and is used in production as part of [GeoNode](#), so examples can be found in that codebase. Documentation and examples can be found at the section [GeoNode's Ad-Hoc API](#).

Java

We are looking for volunteers to flesh out this section with examples.

In the meantime, anyone looking to do Java scripting of the GeoServer REST API should use [GeoServer Manager](#), a REST client library with minimal dependencies on external libraries.

Another option is [gsrj](#). This project is a GeoServer REST client written in Java with no extra dependencies on GeoServer/GeoTools, unlike the standard GeoServer REST module. The project has minimal documentation, but does include a [Quickstart](#).

Ruby

The examples in this section use [rest-client](#), a REST client for Ruby. There is also a project to create a GeoServer-specific REST client in Ruby: [RGeoServer](#).

Once installed on a system, `rest-client` can be included in a Ruby script by adding `require 'rest-client'`.

GET and PUT Settings

Note: In order to execute the example just copy the script content into a `test.ruby` file and execute the following command:

```
$ ruby test.ruby
```

This example shows how to read the settings using GET, make a change and then use PUT to write the change to the server.

```
require 'json'
require 'rest-client'

url = 'http://admin:geoserver@localhost:8080/geoserver/rest/'

# get the settings and parse the JSON into a Hash
json_text = RestClient.get(url + 'settings.json')
settings = JSON.parse(json_text)

# settings can be found with the appropriate keys
global_settings = settings["global"]
jai_settings = global_settings["jai"]

# change a value
jai_settings["allowInterpolation"] = true

# put changes back to the server
RestClient.put(url + 'settings', settings.to_json, :content_type => :json)
```

GeoServer Importer

The Importer extension gives a GeoServer administrator an alternate, more-streamlined method for uploading and configuring new layers.

There are two primary advantages to using the Importer over the standard GeoServer data-loading workflow:

1. **Supports batch operations** (loading and publishing multiple spatial files or database tables in one operation)
2. **Creates unique styles** for each layer, rather than linking to the same (existing) styles.

This section will discuss the Importer extension.

Installing the Importer extension

The Importer extension is an official extension, available on the [GeoServer download](#) page.

1. Download the extension for your version of GeoServer. (If you see an option, select *Core*.)

Warning: Make sure to match the version of the extension to the version of GeoServer.

2. Extract the archive and copy the contents into the GeoServer `WEB-INF/lib` directory.
3. Restart GeoServer.
4. To verify that the extension was installed successfully, open the `web_admin` and look for an *Import Data* option in the *Data* section on the left-side menu.

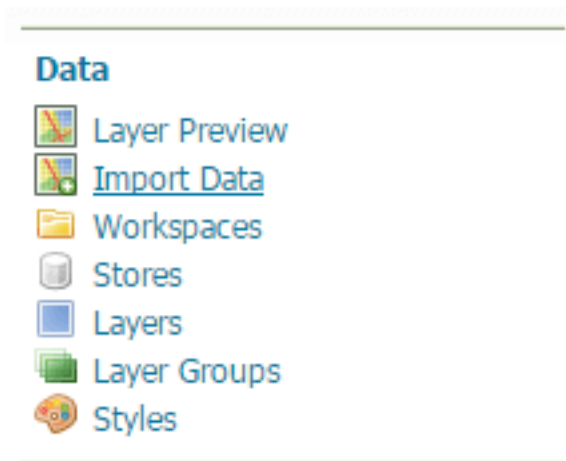


Fig. 93: Importer extension successfully installed.

For additional information please see the section on *Using the Importer extension*.

Using the Importer extension

Here are step-by-step instructions to import multiple shapefiles in one operation. For more details on different types of operations, please see the *Importer interface reference*

1. Find a directory of shapefiles and copy into your `data_directory`.

Note: You can always use the [Natural Earth Quickstart](#) data for this task.

2. Log in as an administrator and navigate to the *Data* → *Import Data* page.

3. For select *Spatial Files* as the data source.

1. Choose a data source to import from






- ☒  Spatial Files - Files from a directory or archive
- ☐  Mosaic - Raster files from a directory composing a mosaic
- ☐  PostGIS - Tables from PostGIS database
- ☐  Oracle - Tables from Oracle database ([Install plugin and drivers.](#))
- ☐  SQL Server - Tables from Microsoft SQL Server database ([Install plugin and drivers.](#))

Fig. 94: Data source

4. Click *Browse* to navigate to the directory of shapefiles to be imported.
5. The web-based file browser will show as options your home directory, data directory, and the root of your file system (or drive). In this case, select *Data directory*

2. Configure the data source

Choose a file or directory

natural_earth_quickstart/110m_cultural [Browse...](#)

Fig. 95: Directory

6. Back on the main form, select *Create new* next to *Workspace*, and enter `ne` to denote the workspace.

Note: Make sure the *Store* field reads *Create new* as well.

3. Specify the target for the import

Workspace

Store

Fig. 96: Import target workspace

7. Click *Next* to start the import process.
8. On the next screen, any layers available for import will be shown.

Note: Non-spatial files will be ignored.


9. In most cases, all files will be ready for import, but if the spatial reference system (SRS) is not recognized, you will need to manually input this but clicking *Advanced*













Note: You will need to manually input the SRS if you used the Natural Earth data above. For each layer click on *Advanced* and set reprojection to EPSG:4326.

10. Check the box next to each layer you wish to import.

Import 0

Status	Created	Last Updated
PENDING	moments ago	moments ago

 /Users/jody/Data/natural_earth/110m_cultural
 Select: [All](#) | [None](#) | [Ready](#)

<input type="checkbox"/> Layer	Status	Actions
<input type="checkbox"/>  ne_110m_admin_0_countries	 Ready	Advanced...
<input type="checkbox"/>  ne_110m_admin_0_boundary_lines_land	 Ready	Advanced...
<input type="checkbox"/>  ne_110m_admin_0_tiny_countries	 Ready	Advanced...
<input type="checkbox"/>  ne_110m_populated_places	 Ready	Advanced...
<input type="checkbox"/>  ne_110m_admin_0_pacific_groupings	 Ready	Advanced...
<input type="checkbox"/>  ne_110m_admin_1_states_provinces_shp	 Ready	Advanced...

Results 1 to 6 (out of 6 items)

Fig. 97: Import layer list

Advanced Import Settings

Reprojection

☒ Enabled

From ...

To [EPSG:WGS 84...](#)

Attribute Remapping

 [Add](#)

Fig. 98: Advanced import settings

Import 0

Status	Created	Last Updated
PENDING	8 minutes ago	8 minutes ago

/Users/fjody/Data/natural_earth/110m_cultural Select: All | None | Ready

Layer	Status	Actions
<input checked="" type="checkbox"/> ne_110m_admin_0_countries	Ready	Advanced...
<input checked="" type="checkbox"/> ne_110m_admin_0_boundary_lines_land	Ready	Advanced...
<input checked="" type="checkbox"/> ne_110m_admin_0_tiny_countries	Ready	Advanced...
<input type="checkbox"/> ne_110m_populated_places	Ready	Advanced...
<input type="checkbox"/> ne_110m_admin_0_pacific_groupings	Ready	Advanced...
<input type="checkbox"/> ne_110m_admin_1_states_provinces_shp	Ready	Advanced...

<< < | > >> Results 1 to 6 (out of 6 items)

Fig. 99: Setting the layers to import

- When ready, click *Import*.

Warning: Don't click *Done* at this point, otherwise the import will be canceled.

- The results of the import process will be shown next to each layer.
- When finished, click *Done*.

Note: Recent import processes are listed at the bottom of the page. You may wish to visit these pages to check if any difficulties were encountered during the import process or import additional layers.

Recent Imports Remove

Import	Status	Last Updated
0	PENDING	6 minutes ago

<< < | > >> Results 1 to 1 (out of 1 items)

Fig. 100: Recent imports

Importer interface reference

The Layer Importer user interface is a component of the GeoServer web interface. You can access it from the GeoServer web interface by clicking the *Import Data* link, found on the left side of the screen after logging in.

Data sources page

The front page of the Layer Importer is where the data source and format are set. The following options are displayed:

Choose a data source to import from

Select one of the following data sources to use for the import:

- *Spatial Files* (see *Supported data formats* for more details)
- *PostGIS* database
- *Oracle* database
- *SQL Server* database

Import Data

1. Choose a data source to import from





- ☒  Spatial Files - Files from a directory or archive
- ☐  PostGIS - Tables from PostGIS database
- ☐  Oracle - Tables from Oracle database
- ☐  SQL Server - Tables from Microsoft SQL Server database

Fig. 101: Choose a data source

The contents of the next section is dependent on the data source chosen here.

Configure the data source: Spatial Files

There is a single box for selecting a file or directory. Click the *Browse* link to bring up a file chooser. To select a file, click on it. To select a directory, click on a directory name to open it and then click *OK*.

2. Configure the data source

Choose a file or directory



Fig. 102: Spatial file data source

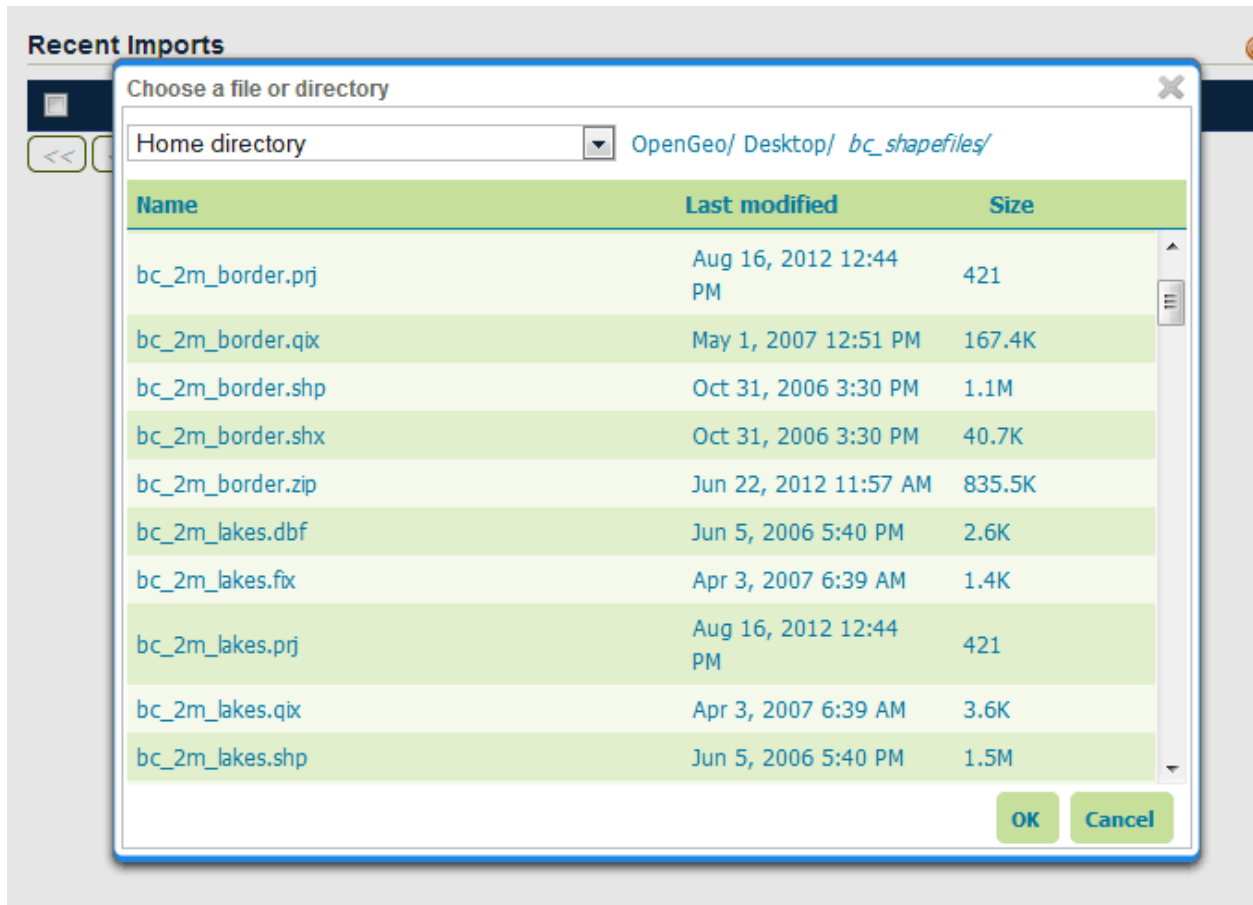


Fig. 103: File chooser for selecting spatial files

Configure the data source: PostGIS

Fill out fields for *Connection type* (Default or JNDI) *Host*, *Port*, *Database name*, *Schema*, *Username* to connect with, and *Password*.

There are also advanced connection options, which are common to the standard PostGIS store loading procedure. (See the [PostGIS data store](#) page in the GeoServer reference documentation.)

2. Configure the data source

Connection type *

Default ▼

Host *

localhost

Port *

54321

Database *

OpenGeo

Schema

public

Username *

OpenGeo

Password

••••••••

▶ Connection pooling

▶ Advanced

Fig. 104: PostGIS data source connection

Configure the data source: Oracle

The parameter fields for the Oracle import are identical to that of PostGIS. The fields aren't populated with default credentials with the exception of the port, which is set to **1521** by default.

Note: This option is only enabled if the Oracle extension is installed.

2. Configure the data source

Connection type *

Default ▼

Host *

localhost

Port *

1521

Database *

Schema

Username *

Password

▶ Connection pooling

▶ Advanced

Fig. 105: Oracle data source connection

Configure the data source: SQL Server

The parameter fields for the SQL Server import are identical to that of PostGIS. The fields aren't populated with default credentials with the exception of the port, which is set to **4866** by default.

Note: This option is only enabled if the SQL Server extension is installed.

2. Configure the data source

Connection type *

Default ▾

Host *

localhost

Port *

4866

Database *

Schema

Username *

Password

▸ [Connection pooling](#)

▸ [Advanced](#)

Fig. 106: SQL Server data source connection

Specify the target for the import

This area specifies where in the GeoServer catalog the new data source will be stored. This does not affect file placement.

Select the name of an existing workspace and store.

3. Specify the target for the import

Workspace

opengeo ▾

Store

postgis ▾

Fig. 107: Target workspace and store in GeoServer

Alternately, select *Create New* and type in a names for a new workspace or store. During the import process, these will be created.

3. Specify the target for the import

Workspace

Store

Fig. 108: Creating a new workspace and store

Recent imports

This section will list previous imports, and whether they were successful or not. Items can be removed from this list with the *Remove* button, but otherwise cannot be edited.

Recent Imports Remove		
Import	Status	Last Updated
<input type="checkbox"/> 0	COMPLETE	moments ago
<input type="button" value="Previous"/> <input type="button" value="First"/> <input type="button" value="Next"/> <input type="button" value="Last"/> <input type="button" value="Refresh"/> Results 1 to 1 (out of 1 items)		

Fig. 109: Recent imports

When ready to continue to the next page, click *Next*.

Layer listing page

On the next page will be a list of layers found by the Layer Importer. The layers will be named according to the source content's name (file name of database table name). For each entry there will be a *Status* showing if the source is ready to be imported.

All layers will be selected for import by default, but can be deselected here by unchecking the box next to each entry.

A common issue during the import process is when a CRS cannot be determined for a given layer. In this case, a dialog box will display where the CRS can be declared explicitly. Enter the CRS and Click *Apply*.

When ready to perform the import, click *Import*.

Each selected layer will be added to the GeoServer catalog inside a new or existing store, and published as a layer.

After the import is complete the status area will refresh showing if the import was successful for each layer. If successful, a dialog box for previewing the layer will be displayed, with options for *Layer Preview* (OpenLayers), *Google Earth*, and *GeoExplorer*.

Advanced import settings page

The *Advanced* link next to each layer will lead to the Advanced import settings page.

On this page, data can be set to be reprojected from one CRS to another during the import process. To enable reprojection, select the *Reprojection* box, and enter the source and target CRS.

Import 1

StatusCreatedLast Updated

COMPLETEmoments agomoments ago

bc_shapefiles

Select: AllNoneReady

Importing into new store bc_shapefiles

Layer	Status	
<input checked="" type="checkbox"/> intersection	➔ Ready for import.	Advanced...
<input checked="" type="checkbox"/> bc_hospitals	➔ Ready for import.	Advanced...
<input checked="" type="checkbox"/> bc_2m_lakes	➔ Ready for import.	Advanced...
<input checked="" type="checkbox"/> bc_2m_rivers	➔ Ready for import.	Advanced...
<input checked="" type="checkbox"/> bc_elections_1996	➔ Ready for import.	Advanced...
<input checked="" type="checkbox"/> bc_parks_2001	➔ Ready for import.	Advanced...
<input checked="" type="checkbox"/> bc_roads	➔ Ready for import.	Advanced...
<input checked="" type="checkbox"/> bc_municipality	➔ Ready for import.	Advanced...
<input checked="" type="checkbox"/> bc_pubs	➔ Ready for import.	Advanced...
<input checked="" type="checkbox"/> bc_elections_nad83	➔ Ready for import.	Advanced...
<input checked="" type="checkbox"/> bc_2m_border	➔ Ready for import.	Advanced...
<input checked="" type="checkbox"/> bc_2m_rivwide	➔ Ready for import.	Advanced...

<<<>>>

Results 1 to 12 (out of 12 items)

Import

Cancel

Fig. 110: List of layers to be imported

Layer

Status

☐ bc_2m_border

⚠ Projection could not be determined.

EPSG:4269

Find...

EPSG:NAD83...

➔ Apply

Fig. 111: Declaring a CRS

Import 1

StatusCreatedLast Updated

INCOMPLETEmoments agomoments ago

bc_shapefiles

Select: AllNoneReady

Importing into new store bc_shapefiles

Layer	Status	
<input checked="" type="checkbox"/> intersection	✔ Import successful.	View in Layer Preview Go
<input checked="" type="checkbox"/> bc_hospitals	✔ Import successful.	View in Layer Preview GeoExplorer Google Earth Go
<input checked="" type="checkbox"/> bc_2m_lakes	✔ Import successful.	View in Layer Preview Go
<input checked="" type="checkbox"/> bc_2m_rivers	✔ Import successful.	View in Layer Preview Go
<input checked="" type="checkbox"/> bc_elections_1996	✔ Import successful.	View in Layer Preview Go

Fig. 112: Layers successfully imported

In addition, on this page attributes can be renamed and their type changed. Click on the *Add* link under *Attribute Remapping* to select the attribute to alter, its type, and its new name. Click *Apply* when done.

Click *Save* when finished.

Advanced Import Settings

Reprojection

☒ Enabled

From

EPSG:4269

Find...

EPSG:NAD83...

To

EPSG:3005

Find...

EPSG:NAD83 / BC Albers...

Attribute Remapping

BORDER_ID ▾

Double ▾

ID

➔ Apply ⏹ Cancel

➕ Add

Save

Cancel

Fig. 113: Advanced layer list page

Supported data formats

The importer supports any format that GeoServer can use a data store or coverage store. These include the most commonly used formats:

- Shapefile
- GeoTIFF

And a few additional formats:

- CSV
- KML

The following databases are supported:

- PostGIS
- Oracle
- Microsoft SQL Server

Note: Oracle and SQL Server require extra drivers to be installed.

- Install instructions for Oracle
 - Install instructions for SQL Server
-

REST API

Importer concepts

The importer REST API is built around a tree of objects representing a single import, structured as follows:

- **import**
 - target workspace
 - data
 - **task (one or more)**
 - * data
 - * layer
 - * transformation (one or more)

An **import** refers to the top level object and is a “session” like entity the state of the entire import. It maintains information relevant to the import as a whole such as user information, timestamps along with optional information that is uniform along all tasks, such as a target workspace, the shared input data (e.g., a directory, a database). An import is made of any number of task objects.

A **data** is the description of the source data of a import (overall) or a task. In case the import has a global data definition, this normally refers to an aggregate store such as a directory or a database, and the data associated to the tasks refers to a single element inside such aggregation, such as a single file or table.

A **task** represents a unit of work to the importer needed to register one new layer, or alter an existing one, and contains the following information:

- The data being imported
- The target store that is the destination of the import
- The target layer
- The data of a task, referred to as its source, is the data to be processed as part of the task.
- The transformations that we need to apply to the data before it gets imported

This data comes in a variety of forms including:

- A spatial file (Shapefile, GeoTiff, KML, etc. . .)
- A directory of spatial files
- A table in a spatial database
- A remote location that the server will download data from

A task is classified as either “direct” or “indirect”. A *direct task* is one in which the data being imported requires no transformation to be imported. It is imported directly. An example of such a task is one that involves simply importing an existing Shapefile as is. An *indirect task* is one that does require a **transformation** to the original import data. An example of an indirect task is one that involves importing a Shapefile into an existing PostGIS database. Another

example of indirect task might involve taking a CSV file as an input, turning a x and y column into a Point, remapping a string column into a timestamp, and finally import the result into a PostGIS.

REST API Reference

All the imports

/imports

Method	Action	Status Code/Headers	Input	Output	Parameters
GET	Retrieve all imports	200	n/a	Import Collection	n/a
POST	Create a new import	201 with Location header	n/a	Imports	async=false/true

Retrieving the list of all imports

```
GET /imports
```

results in:

```
Status: 200 OK
Content-Type: application/json

{
  "imports": [{
    "id": 0,
    "state": "COMPLETE",
    "href": "http://localhost:8080/geoserver/rest/imports/0"
  }, {
    "id": 1,
    "state": "PENDING",
    "href": "http://localhost:8080/geoserver/rest/imports/1"
  }]
}
```

Creating a new import

Posting to the /imports path a import JSON object creates a new import session:

```
Content-Type: application/json

{
  "import": {
    "targetWorkspace": {
      "workspace": {
        "name": "scratch"
      }
    },
    "targetStore": {
      "dataStore": {
```

(continues on next page)

(continued from previous page)

```

        "name": "shapes"
    },
    "data": {
        "type": "file",
        "file": "/data/spearfish/archsites.shp"
    }
}

```

The parameters are:

Name	Optional	Description
targetWorkspace	N	The target workspace to import to
targetStore	Y	The target store to import to
data	Y	The data to be imported

The mere creation does not start the import, but it may automatically populate its tasks depending on the target. For example, by referring a directory of shapefiles to be importer, the creation will automatically fill in a task to import each of the shapefiles as a new layer.

The response to the above POST request will be:

```

Status: 201 Created
Location: http://localhost:8080/geoserver/rest/imports/2
Content-Type: application/json

{
  "import": {
    "id": 2,
    "href": "http://localhost:8080/geoserver/rest/imports/2",
    "state": "READY",
    "targetWorkspace": {
      "workspace": {
        "name": "scratch"
      }
    },
    "targetStore": {
      "dataStore": {
        "name": "shapes",
        "type": "PostGIS"
      }
    },
    "data": {
      "type": "file",
      "format": "Shapefile",
      "href": "http://localhost:8080/geoserver/rest/imports/2/data",
      "file": "archsites.shp"
    },
    "tasks": [
      {
        "id": 0,
        "href": "http://localhost:8080/geoserver/rest/imports/2/tasks/0",
        "state": "READY"
      }
    ]
  }
}

```

(continues on next page)

(continued from previous page)

```

    }
}

```

The operation of populating the tasks can require time, especially if done against a large set of files, or against a “remote” data (more on this later), in this case the POST request can include `?async=true` at the end of the URL to make the importer run it asynchronously. In this case the import will be created in INIT state and will remain in such state until all the data transfer and task creation operations are completed. In case of failure to fetch data the import will immediately stop, the state will switch to the `INIT_ERROR` state, and a error message will appear in the import context “message” field.

Import object

`/imports/<importId>`

Method	Action	Status Code/Headers	In-put	Out-put	Pa-ram-e-ters
GET	Retrieve import with id <importId>	200	n/a	Im-ports	n/a
POST	Execute import with id <importId>	204	n/a	n/a	async=true/false
PUT	Create import with proposed id <importId>. If the proposed id is ahead of the current (next) id, the current id will be advanced. If the proposed id is less than or equal to the current id, the current will be used. This allows an external system to dictate the id management.	201 with Location header	n/a	Im-ports	n/a
DELETE	Remove import with id <importId>	200	n/a	n/a	n/a

The representation of a import is the same as the one contained in the import creation response. The execution of a import can be a long task, as such, it’s possible to add `async=true` to the request to make it run in a asynchronous fashion, the client will have to poll the import representation and check when it reaches the “COMPLETE” state.

Data

A import can have a “data” representing the source of the data to be imported. The data can be of different types, in particular, “file”, “directory”, “mosaic”, “database” and “remote”. During the import initialization the importer will scan the contents of said resource, and generate import tasks for each data found in it.

Most data types are discussed in the task section, the only type that’s specific to the whole import context is the “remote” one, that is used to ask the importer to fetch the data from a remote location autonomously, without asking the client to perform an upload.

The representation of a remote resource looks as follows:

```

"data": {
  "type": "remote",
  "location": "ftp://fthost/path/to/importFile.zip",
  "username": "user",
  "password": "secret",
  "domain" : "mydomain"
}

```

The location can be any URI supported by Commons VFS, including HTTP and FTP servers. The username, password and domain elements are all optional, and required only if the remote server demands an authentication of sorts. In case the referred file is compressed, it will be unpacked as the download completes, and the tasks will be created over the result of unpacking.

Tasks

/imports/<importId>/tasks

Method	Action	Status Code/Headers	Input	Output
GET	Retrieve all tasks for import with id <importId>	200	n/a	Task Collection
POST	Create a new task	201 with Location header	<i>Multipart form data</i>	Tasks

Getting the list of tasks

```
GET /imports/0/tasks
```

Results in:

```
Status: 200 OK
Content-Type: application/json

{
  "tasks": [
    {
      "id": 0,
      "href": "http://localhost:8080/geoserver/rest/imports/2/tasks/0",
      "state": "READY"
    }
  ]
}
```

Creating a new task as a file upload

A new task can be created by issuing a POST to imports/<importId>/tasks as a “Content-type: multipart/form-data” multipart encoded data as defined by RFC 2388. One or more file can be uploaded this way, and a task will be created for importing them. In case the file being uploaded is a zip file, it will be unzipped on the server side and treated as a directory of files.

The response to the upload will be the creation of a new task, for example:

```
Status: 201 Created
Location: http://localhost:8080/geoserver/rest/imports/1/tasks/1
Content-type: application/json

{
  "task": {
```

(continues on next page)

(continued from previous page)

```

    "id": 1,
    "href": "http://localhost:8080/geoserver/rest/imports/2/tasks/1",
    "state": "READY",
    "updateMode": "CREATE",
    "data": {
      "type": "file",
      "format": "Shapefile",
      "href": "http://localhost:8080/geoserver/rest/imports/2/tasks/1/data",
      "file": "bugsites.shp"
    },
    "target": {
      "href": "http://localhost:8080/geoserver/rest/imports/2/tasks/1/target",
      "dataStore": {
        "name": "shapes",
        "type": "PostGIS"
      }
    },
    "progress": "http://localhost:8080/geoserver/rest/imports/2/tasks/1/progress",
    "layer": {
      "name": "bugsites",
      "href": "http://localhost:8080/geoserver/rest/imports/2/tasks/1/layer"
    },
    "transformChain": {
      "type": "vector",
      "transforms": []
    }
  }
}

```

Creating a new task from form upload

This creation mode assumes the POST to `imports/<importId>/tasks` of form URL encoded data containing a `url` parameter:

```

Content-type: application/x-www-form-urlencoded

url=file:///data/spearfish/

```

The creation response will be the same as the multipart upload.

Single task resource

`/imports/<importId>/task/<taskId>`

Method	Action	Status Code/Headers	In-put	Out-put
GET	Retrieve task with id <taskId> within import with id <importId>	200	n/a	Task
PUT	Modify task with id <taskId> within import with id <importId>	200	Task	Task
DELETE	Remove task with id <taskId> within import with id <importId>	200	n/a	n/a

The representation of a task resource is the same one reported in the task creation response.

Updating a task

A PUT request over an existing task can be used to update its representation. The representation can be partial, and just contains the elements that need to be updated.

The updateMode of a task normally starts as “CREATE”, that is, create the target resource if missing. Other possible values are “REPLACE”, that is, delete the existing features in the target layer and replace them with the task source ones, or “APPEND”, to just add the features from the task source into an existing layer.

The following PUT request updates a task from “CREATE” to “APPEND” mode:

```
Content-Type: application/json

{
  "task": {
    "updateMode": "APPEND"
  }
}
```

Directory files representation

The following operations are specific to data objects of type `directory`.

`/imports/<importId>/task/<taskId>/data/files`

Method	Action	Status Code/Headers	In-put	Out-put
GET	Retrieve the list of files for a task with id <taskId> within import with id <importId>	200	n/a	Task

The response to a GET request will be:

```
Status: 200 OK
Content-Type: application/json

{
  files: [
    {
      file: "tasmania_cities.shp",
      href: "http://localhost:8080/geoserver/rest/imports/0/tasks/0/data/
↪files/tasmania_cities.shp"
    },
    {
      file: "tasmania_roads.shp",
      href: "http://localhost:8080/geoserver/rest/imports/0/tasks/0/data/
↪files/tasmania_roads.shp"
    },
    {
      file: "tasmania_state_boundaries.shp",
      href: "http://localhost:8080/geoserver/rest/imports/0/tasks/0/data/
↪files/tasmania_state_boundaries.shp"
```

(continues on next page)

(continued from previous page)

```

    },
    {
      file: "tasmania_water_bodies.shp",
      href: "http://localhost:8080/geoserver/rest/imports/0/tasks/0/data/
↪files/tasmania_water_bodies.shp"
    }
  ]
}

```

/imports/<importId>/task/<taskId>/data/files/<fileId>

Method	Action	Status Code/Headers	In-put	Out-put
GET	Retrieve the file with id <fileId> from the data of a task with id <taskId> within import with id <importId>	200	n/a	Task
DELETE	Remove a specific file from the task with id <taskId> within import with id <importId>	200	n/a	n/a

Following the links we'll get to the representation of a single file, notice how in this case a main file can be associate to sidecar files:

```

Status: 200 OK
Content-Type: application/json

{
  type: "file",
  format: "Shapefile",
  location: "C:\\devel\\gs_data\\release\\data\\taz_shapes",
  file: "tasmania_cities.shp",
  href: "http://localhost:8080/geoserver/rest/imports/0/tasks/0/data/files/
↪tasmania_cities.shp",
  prj: "tasmania_cities.prj",
  other: [
    "tasmania_cities.dbf",
    "tasmania_cities.shx"
  ]
}

```

Mosaic extensions

In case the input data is of `mosaic` type, we have all the attributes typical of a directory, plus support for directly specifying the timestamp of a particular granule.

In order to specify the timestamp a PUT request can be issued against the granule:

```

Content-Type: application/json

{
  "timestamp": "2004-01-01T00:00:00.000+0000"
}

```

and the response will be:

```
Status: 200 OK
Content-Type: application/json

{
  "type": "file",
  "format": "GeoTIFF",
  "href": "http://localhost:8080/geoserver/rest/imports/0/tasks/0/data/files/bm_
↪200401.tif",
  "location": "/data/bluemarble/mosaic",
  "file": "bm_200401.tif",
  "prj": null,
  "other": [],
  "timestamp": "2004-01-01T00:00:00.000+0000"
}
```

Database data

The following operations are specific to data objects of type `database`. At the time of writing, the REST API does not allow the creation of a database data source, but it can provide a read only description of one that has been created using the GUI.

`/imports/<importId>/tasks/<taskId>/data`

Method	Action	Status Code/Headers	In-put	Output
GET	Retrieve the database connection parameters for a task with id <code><taskId></code> within import with id <code><importId></code>	200	n/a	List of database connection parameters and available tables

Performing a GET on a database type data will result in the following response:

```
{
  type: "database",
  format: "PostGIS",
  href: "http://localhost:8080/geoserver/rest/imports/0/data",
  parameters: {
    schema: "public",
    fetch size: 1000,
    validate connections: true,
    Connection timeout: 20,
    Primary key metadata table: null,
    preparedStatements: true,
    database: "gttest",
    port: 5432,
    passwd: "cite",
    min connections: 1,
    dbtype: "postgis",
    host: "localhost",
    Loose bbox: true,
    max connections: 10,
    user: "cite"
  },
}
```

(continues on next page)

(continued from previous page)

```

    tables: [
      "geoline",
      "geopoint",
      "lakes",
      "line3d",
    ]
  }

```

Database table

The following operations are specific to data objects of type `table`. At the time of writing, the REST API does not allow the creation of a database data source, but it can provide a read only description of one that has been created using the GUI. A table description is normally linked to task, and refers to a database data linked to the overall import.

/imports/<importId>/tasks/<taskId>/data

Method	Action	Status Code/Headers	In-put	Output
GET	Retrieve the table description for a task with id <taskId> within import with id <importId>	200	n/a	A table representation

Performing a GET on a database type data will result in the following response:

```

{
  type: "table",
  name: "abc",
  format: "PostGIS",
  href: "http://localhost:8080/geoserver/rest/imports/0/tasks/0/data"
}

```

Task target layer

/imports/<importId>/tasks/<taskId>/layer

The layer defines how the target layer will be created

Method	Action	Status Code/Headers	In-put	Output
GET	Retrieve the layer of a task with id <taskId> within import with id <importId>	200	n/a	A layer JSON representation
PUT	Modify the target layer for a task with id <taskId> within import with id <importId>	200	Task	Task

Requesting the task layer will result in the following:

```

Status: 200 OK
Content-Type: application/json

```

(continues on next page)

(continued from previous page)

```

{
  layer: {
    name: "tasmania_cities",
    href: "http://localhost:8080/geoserver/rest/imports/0/tasks/0/layer",
    title: "tasmania_cities",
    originalName: "tasmania_cities",
    nativeName: "tasmania_cities",
    srs: "EPSG:4326",
    bbox: {
      minx: 147.2909004483,
      miny: -42.85110181689001,
      maxx: 147.2911004483,
      maxy: -42.85090181689,
      crs: "GEOGCS[\"WGS 84\", DATUM[\"World Geodetic System 1984\", SPHEROID[
↪ \"WGS 84\", 6378137.0, 298.257223563, AUTHORITY[\"EPSG\", \"7030\"]], AUTHORITY[\"EPSG\",
↪ \"6326\"]], PRIMEM[\"Greenwich\", 0.0, AUTHORITY[\"EPSG\", \"8901\"]], UNIT[\"degree\", 0.
↪ 017453292519943295], AXIS[\"Geodetic longitude\", EAST], AXIS[\"Geodetic latitude\",
↪ NORTH], AUTHORITY[\"EPSG\", \"4326\"]]"
    ],
    attributes: [
      {
        name: "the_geom",
        binding: "com.vividsolutions.jts.geom.MultiPoint"
      },
      {
        name: "CITY_NAME",
        binding: "java.lang.String"
      },
      {
        name: "ADMIN_NAME",
        binding: "java.lang.String"
      },
      {
        name: "CNTRY_NAME",
        binding: "java.lang.String"
      },
      {
        name: "STATUS",
        binding: "java.lang.String"
      },
      {
        name: "POP_CLASS",
        binding: "java.lang.String"
      }
    ],
    style: {
      name: "cite_tasmania_cities",
      href: "http://localhost:8080/geoserver/rest/imports/0/tasks/0/
↪ layer/style"
    }
  }
}

```

All the above attributes can be updated using a PUT request. Even if the above representation is similar to the REST config API, it should not be confused with it, as it does not support all the same properties, in particular the supported properties are all the ones listed above.

Task transformations

`/imports/<importId>/tasks/<taskId>/transforms`

Method	Action	Status Code/Headers	Input	Output
GET	Retrieve the list of transformations of a task with id <taskId> within import with id <importId>	200	n/a	A list of transformations in JSON format
POST	Create a new transformation and append it inside a task with id <taskId> within import with id <importId>	201	A JSON transformation representation	The transform location

Retrieving the transformation list

A GET request for the list of transformations will result in the following response:

```
Status: 200 OK
Content-Type: application/json

{
  "transforms": [
    {
      "type": "ReprojectTransform",
      "href": "http://localhost:8080/geoserver/rest/imports/0/tasks/1/transforms/0",
      "source": null,
      "target": "EPSG:4326"
    },
    {
      "type": "DateFormatTransform",
      "href": "http://localhost:8080/geoserver/rest/imports/0/tasks/1/transforms/1",
      "field": "date",
      "format": "yyyyMMdd"
    }
  ]
}
```

Appending a new transformation

Creating a new transformation requires posting a JSON document with a `type` property identifying the class of the transformation, plus any extra attribute required by the transformation itself (this is transformation specific, each one will use a different set of attributes).

The following POST request creates an attribute type remapping:

```
Content-Type: application/json

{
  "type": "AttributeRemapTransform",
  "field": "cat",
  "target": "java.lang.Integer"
}
```

The response will be:

```
Status: 201 OK
Location: http://localhost:8080/geoserver/rest/imports/0/tasks/1/transform/2
```

/imports/<importId>/tasks/<taskId>/transforms/<transformId>

Method	Action	Status Code/Headers	Input Headers	Output
GET	Retrieve a transformation identified by <transformId> inside a task with id <taskId> within import with id <importId>	200	n/a	A single transformation in JSON format
PUT	Modifies the definition of a transformation identified by <transformId> inside a task with id <taskId> within import with id <importId>	200	A JSON transformation representation (eventually just the portion of it that needs to be modified)	The full transformation representation
DELETE	Removes the transformation identified by <transformId> inside a task with id <taskId> within import with id <importId>	200	A JSON transformation representation (eventually just the portion of it that needs to be modified)	The full transformation representation

Retrieve a single transformation

Requesting a single transformation by identifier will result in the following response:

```
Status: 200 OK
Content-Type: application/json

{
  "type": "ReprojectTransform",
  "href": "http://localhost:8080/geoserver/rest/imports/0/tasks/1/transforms/0",
  "source": null,
  "target": "EPSG:4326"
}
```

Modify an existing transformation

Assuming we have a reprojection transformation, and that we need to change the target SRS type, the following PUT request will do the job:

```
Content-Type: application/json

{
  "type": "ReprojectTransform",
  "target": "EPSG:3005"
}
```

The response will be:

```
Status: 200 OK
Content-Type: application/json

{
  "type": "ReprojectTransform",
  "href": "http://localhost:8080/geoserver/rest/imports/0/tasks/1/transform/0",
  "source": null,
  "target": "EPSG:3005"
}
```

Transformation reference

AttributeRemapTransform

Remaps a certain field to a given target data type

Parameter	Optional	Description
field	N	The name of the field to be remapped
target	N	The “target” field type, as a fully qualified Java class name

AttributesToPointGeometryTransform

Transforms two numeric fields `latField` and `lngField` into a point geometry representation `POINT(lngField, latField)`, the source fields will be removed.

Parameter	Optional	Description
latField	N	The “latitude” field
lngField	N	The “longitude” field

CreateIndexTransform

For database targets only, creates an index on a given column after importing the data into the database

Parameter	Optional	Description
field	N	The field to be indexed

DateFormatTransform

Parses a string representation of a date into a `Date/Timestamp` object

Parameter	Optional	Description
field	N	The field to be parsed
format	Y	A date parsing pattern, setup using the Java SimpleDateFormat syntax. In case it’s missing, a number of built-in formats will be tried instead (short and full ISO date formats, dates without any separators).

IntegerFieldToDateTransform

Takes a integer field and transforms it to a date, interpreting the integer field as a date

Parameter	Optional	Description
field	N	The field containing the year information

ReprojectTransform

Reprojects a vector layer from a source CRS to a target CRS

Parameter	Optional	Description
source	Y	Identifier of the source coordinate reference system (the native one will be used if missing)
target	N	Identifier of the target coordinate reference system

GdalTranslateTransform

Applies `gdal_translate` to a single file raster input. Requires `gdal_translate` to be inside the PATH used by the web container running GeoServer.

Parameter	Optional	Description
options	N	Array of options that will be passed to <code>gdal_translate</code> (beside the input and output names, which are internally managed)

GdalWarpTransform

Applies `gdalwarp` to a single file raster input. Requires `gdalwarp` to be inside the PATH used by the web container running GeoServer.

Parameter	Optional	Description
options	N	Array of options that will be passed to <code>gdalwarp</code> (beside the input and output names, which are internally managed)

GdalAddoTransform

Applies `gdaladdo` to a single file raster input. Requires `gdaladdo` to be inside the PATH used by the web container running GeoServer.

Parameter	Optional	Description
options	N	Array of options that will be passed to <code>gdaladdo</code> (beside the input file name, which is internally managed)
levels	N	Array of integers with the overview levels that will be passed to <code>gdaladdo</code>

Importer REST API examples

Mass configuring a directory of shapefiles

In order to initiate an import of the `c:\data\tasmania` directory into the existing `tasmania` workspace the following JSON will be POSTed to GeoServer:

```
{
  "import": {
    "targetWorkspace": {
      "workspace": {
        "name": "tasmania"
      }
    },
    "data": {
      "type": "directory",
      "location": "C:/data/tasmania"
    }
  }
}
```

This curl command can be used for the purpose:

```
curl -u admin:geoserver -XPOST -H "Content-type: application/json" -d @import.json
→ "http://localhost:8080/geoserver/rest/imports"
```

The importer will locate the files to be imported, and automatically prepare the tasks, returning the following response:

```
{
  "import": {
    "id": 9,
    "href": "http://localhost:8080/geoserver/rest/imports/9",
    "state": "PENDING",
    "archive": false,
    "targetWorkspace": {
      "workspace": {
        "name": "tasmania"
      }
    },
    "data": {
      "type": "directory",
      "format": "Shapefile",
      "location": "C:\\data\\tasmania",
      "href": "http://localhost:8080/geoserver/rest/imports/9/data"
    },
    "tasks": [
      {
        "id": 0,
        "href": "http://localhost:8080/geoserver/rest/imports/9/tasks/0",
        "state": "READY"
      },
      {
        "id": 1,
        "href": "http://localhost:8080/geoserver/rest/imports/9/tasks/1",
        "state": "READY"
      }
    ]
  }
}
```

(continues on next page)

(continued from previous page)

```

        "id": 2,
        "href": "http://localhost:8080/geoserver/rest/imports/9/tasks/2",
        "state": "READY"
      },
      {
        "id": 3,
        "href": "http://localhost:8080/geoserver/rest/imports/9/tasks/3",
        "state": "READY"
      }
    ]
  }
}

```

After checking every task is ready, the import can be initiated by executing a POST on the import resource:

```
curl -u admin:geoserver -XPOST "http://localhost:8080/geoserver/rest/imports/9"
```

The resource can then be monitored for progress, and eventually final results:

```
curl -u admin:geoserver -XGET "http://localhost:8080/geoserver/rest/imports/9"
```

Which in case of successful import will look like:

```

{
  "import": {
    "id": 9,
    "href": "http://localhost:8080/geoserver/rest/imports/9",
    "state": "COMPLETE",
    "archive": false,
    "targetWorkspace": {
      "workspace": {
        "name": "tasmania"
      }
    }
  },
  "data": {
    "type": "directory",
    "format": "Shapefile",
    "location": "C:\\data\\tasmania",
    "href": "http://localhost:8080/geoserver/rest/imports/9/data"
  },
  "tasks": [
    {
      "id": 0,
      "href": "http://localhost:8080/geoserver/rest/imports/9/tasks/0",
      "state": "COMPLETE"
    },
    {
      "id": 1,
      "href": "http://localhost:8080/geoserver/rest/imports/9/tasks/1",
      "state": "COMPLETE"
    },
    {
      "id": 2,
      "href": "http://localhost:8080/geoserver/rest/imports/9/tasks/2",
      "state": "COMPLETE"
    },
    {

```

(continues on next page)

(continued from previous page)

```

        "id": 3,
        "href": "http://localhost:8080/geoserver/rest/imports/9/tasks/3",
        "state": "COMPLETE"
    }
]
}
}

```

Configuring a shapefile with no projection information

In this case, let's assume we have a single shapefile, `tasmania_cities.shp`, that does not have the ancillary `.prj` file (the example is equally good for any case where the `prj` file contents cannot be matched to an official EPSG code).

We are going to post the following import definition:

```

{
  "import": {
    "targetWorkspace": {
      "workspace": {
        "name": "tasmania"
      }
    },
    "data": {
      "type": "file",
      "file": "C:/data/tasmania/tasmania_cities.shp"
    }
  }
}

```

With the usual `curl` command:

```

curl -u admin:geoserver -XPOST -H "Content-type: application/json" -d @import.json
→ "http://localhost:8080/geoserver/rest/imports"

```

The response in case the CRS is missing will be:

```

{
  "import": {
    "id": 13,
    "href": "http://localhost:8080/geoserver/rest/imports/13",
    "state": "PENDING",
    "archive": false,
    "targetWorkspace": {
      "workspace": {
        "name": "tasmania"
      }
    },
    "data": {
      "type": "file",
      "format": "Shapefile",
      "file": "tasmania_cities.shp"
    },
    "tasks": [
      {
        "id": 0,

```

(continues on next page)

(continued from previous page)

```
        "href": "http://localhost:8080/geoserver/rest/imports/13/tasks/0",
        "state": "NO_CRS"
    }
  ]
}
}
```

Drilling down to the task layer, we can see the SRS information is missing:

```
{
  "layer": {
    "name": "tasmania_cities",
    "href": "http://localhost:8080/geoserver/rest/imports/13/tasks/0/layer",
    "title": "tasmania_cities",
    "originalName": "tasmania_cities",
    "nativeName": "tasmania_cities",
    "bbox": {
      "minx": 146.2910004483,
      "miny": -43.85100181689,
      "maxx": 148.2910004483,
      "maxy": -41.85100181689
    },
    "attributes": [
      {
        "name": "the_geom",
        "binding": "com.vividsolutions.jts.geom.MultiPoint"
      },
      {
        "name": "CITY_NAME",
        "binding": "java.lang.String"
      },
      {
        "name": "ADMIN_NAME",
        "binding": "java.lang.String"
      },
      {
        "name": "CNTRY_NAME",
        "binding": "java.lang.String"
      },
      {
        "name": "STATUS",
        "binding": "java.lang.String"
      },
      {
        "name": "POP_CLASS",
        "binding": "java.lang.String"
      }
    ],
    "style": {
      "name": "tasmania_tasmania_cities2",
      "href": "http://localhost:8080/geoserver/rest/imports/13/tasks/0/layer/style"
    }
  }
}
```

The following PUT request will update the SRS:

```
curl -u admin:geoserver -XPUT -H "Content-type: application/json" -d @layerUpdate.
↪json "http://localhost:8080/geoserver/rest/imports/13/tasks/0/layer/"
```

Where layerUpdate.json is:

```
{
  layer : {
    srs: "EPSG:4326"
  }
}
```

Getting the import definition again, we'll find it ready to execute:

```
{
  "import": {
    "id": 13,
    "href": "http://localhost:8080/geoserver/rest/imports/13",
    "state": "PENDING",
    "archive": false,
    "targetWorkspace": {
      "workspace": {
        "name": "tasmania"
      }
    },
    "data": {
      "type": "file",
      "format": "Shapefile",
      "file": "tasmania_cities.shp"
    },
    "tasks": [
      {
        "id": 0,
        "href": "http://localhost:8080/geoserver/rest/imports/13/tasks/0",
        "state": "READY"
      }
    ]
  }
}
```

A POST request will make it execute:

```
curl -u admin:geoserver -XPOST "http://localhost:8080/geoserver/rest/imports/13"
```

And eventually succeed:

```
{
  "import": {
    "id": 13,
    "href": "http://localhost:8080/geoserver/rest/imports/13",
    "state": "COMPLETE",
    "archive": false,
    "targetWorkspace": {
      "workspace": {
        "name": "tasmania"
      }
    },
    "data": {
```

(continues on next page)

(continued from previous page)

```

    "type": "file",
    "format": "Shapefile",
    "file": "tasmania_cities.shp"
  },
  "tasks": [
    {
      "id": 0,
      "href": "http://localhost:8080/geoserver/rest/imports/13/tasks/0",
      "state": "COMPLETE"
    }
  ]
}

```

Uploading a CSV file to PostGIS while transforming it

A remote sensing tool is generating CSV files with some locations and measurements, that we want to upload into PostGIS as a new spatial table. The CSV file looks as follows:

```

AssetID, SampleTime, Lat, Lon, Value
1, 2015-01-01T10:00:00, 10.00, 62.00, 15.2
1, 2015-01-01T11:00:00, 10.10, 62.11, 30.25
1, 2015-01-01T12:00:00, 10.20, 62.22, 41.2
1, 2015-01-01T13:00:00, 10.31, 62.33, 27.6
1, 2015-01-01T14:00:00, 10.41, 62.45, 12

```

First, we are going to create a empty import with an existing PostGIS store as the target:

```

curl -u admin:geoserver -XPOST -H "Content-type: application/json" -d @import.json
↪ "http://localhost:8080/geoserver/rest/imports"

```

Where import.json is:

```

{
  "import": {
    "targetWorkspace": {
      "workspace": {
        "name": "topp"
      }
    },
    "targetStore": {
      "dataStore": {
        "name": "gttest"
      }
    }
  }
}

```

Then, we are going to POST the CSV file to the tasks list, in order to create an import task for it:

```

curl -u admin:geoserver -F name=test -F filedata=@values.csv "http://localhost:8080/
↪ geoserver/rest/imports/0/tasks"

```

And we are going to get back a new task definition, with a notification that the CRS is missing:

```
{
  "task": {
    "id": 0,
    "href": "http://localhost:8080/geoserver/rest/imports/16/tasks/0",
    "state": "NO_CRS",
    "updateMode": "CREATE",
    "data": {
      "type": "file",
      "format": "CSV",
      "file": "values.csv"
    },
    "target": {
      "href": "http://localhost:8080/geoserver/rest/imports/16/tasks/0/target",
      "dataStore": {
        "name": "values",
        "type": "CSV"
      }
    },
    "progress": "http://localhost:8080/geoserver/rest/imports/16/tasks/0/progress",
    "layer": {
      "name": "values",
      "href": "http://localhost:8080/geoserver/rest/imports/16/tasks/0/layer"
    },
    "transformChain": {
      "type": "vector",
      "transforms": [

      ]
    }
  }
}
```

As before, we are going to force the CRS by updating the layer:

```
curl -u admin:geoserver -XPUT -H "Content-type: application/json" -d @layerUpdate.
↪json "http://localhost:8080/geoserver/rest/imports/0/tasks/0/layer/"
```

Where layerUpdate.json is:

```
{
  layer : {
    srs: "EPSG:4326"
  }
}
```

Then, we are going to create a transformation mapping the Lat/Lon columns to a point:

```
{
  "type": "AttributesToPointGeometryTransform",
  "latField": "Lat",
  "lngField": "Lon"
}
```

The above will be uploaded to GeoServer as follows:

```
curl -u admin:geoserver -XPOST -H "Content-type: application/json" -d @toPoint.json
↪"http://localhost:8080/geoserver/rest/imports/0/tasks/0/transforms"
```

Now the import is ready to run, and we'll execute it using:

```
curl -u admin:geoserver -XPOST "http://localhost:8080/geoserver/rest/imports/0"
```

If all goes well the new layer is created in PostGIS and registered in GeoServer as a new layer.

In case the features in the CSV need to be appended to an existing layer a PUT request against the task might be performed, changing its updateMode from “CREATE” to “APPEND”. Changing it to “REPLACE” instead will preserve the layer, but remove the old contents and replace them with the newly uploaded ones.

Uploading and optimizing a GeoTiff with ground control points

A data supplier is periodically providing GeoTIFFs that we need to configure in GeoServer. The GeoTIFF is referenced via Ground Control Points, is organized by stripes, and has no overviews. The objective is to rectify, optimize and publish it via the importer.

First, we are going to create an empty import with no store as the target:

```
curl -u admin:geoserver -XPOST -H "Content-type: application/json" -d @import.json  
↪ "http://localhost:8080/geoserver/rest/imports"
```

Where import.json is:

```
{  
  "import": {  
    "targetWorkspace": {  
      "workspace": {  
        "name": "sf"  
      }  
    }  
  }  
}
```

Then, we are going to POST the GeoTIFF file to the tasks list, in order to create an import task for it:

```
curl -u admin:geoserver -F name=test -F filedata=@box_gcp_fixed.tif "http://  
↪ localhost:8080/geoserver/rest/imports/0/tasks"
```

We are then going to append the transformations to rectify (gdalwarp), retile (gdal_translate) and add overviews (gdaladdo) to it:

```
curl -u admin:geoserver -XPOST -H "Content-type: application/json" -d @warp.json  
↪ "http://localhost:8080/geoserver/rest/imports/0/tasks/0/transforms"  
curl -u admin:geoserver -XPOST -H "Content-type: application/json" -d @gtx.json  
↪ "http://localhost:8080/geoserver/rest/imports/0/tasks/0/transforms"  
curl -u admin:geoserver -XPOST -H "Content-type: application/json" -d @gad.json  
↪ "http://localhost:8080/geoserver/rest/imports/0/tasks/0/transforms"
```

warp.json is:

```
{  
  "type": "GdalWarpTransform",  
  "options": [ "-t_srs", "EPSG:4326"  
]
```

gtx.json is:


```
{
  "type": "GdalTranslateTransform",
  "options": [ "-co", "TILED=YES", "-co", "BLOCKXSIZE=512", "-co", "BLOCKYSIZE=512"]
}
```

gad.json is:

```
{
  "type": "GdalAddoTransform",
  "options": [ "-r", "average"],
  "levels" : [2, 4, 8, 16]
}
```

Now the import is ready to run, and we'll execute it using:

```
curl -u admin:geoserver -XPOST "http://localhost:8080/geoserver/rest/imports/0"
```

A new layer `box_gcp_fixed` layer will appear in GeoServer, with an underlying GeoTIFF file ready for web serving.

Adding a new granule into an existing mosaic

A data supplier is periodically providing new time based imagery that we need to add into an existing mosaic in GeoServer. The imagery is in GeoTIFF format, and lacks a good internal structure, which needs to be aligned with the one into the other images.

First, we are going to create a import with an indication of where the granule is located, and the target store:

```
curl -u admin:geoserver -XPOST -H "Content-type: application/json" -d @import.json
"http://localhost:8080/geoserver/rest/imports"
```

Where import.json is:

```
{
  "import": {
    "targetWorkspace": {
      "workspace": {
        "name": "topp"
      }
    },
    "data": {
      "type": "file",
      "file": "/home/aaime/devel/gisData/ndimensional/data/world/world.200407.
↪3x5400x2700.tiff"
    },
    "targetStore": {
      "dataStore": {
        "name": "bluemarble"
      }
    }
  }
}
```

We are then going to append the transformations to harmonize the file with the rest of the mosaic:

```
curl -u admin:geoserver -XPOST -H "Content-type: application/json" -d @gtx.json
↪ "http://localhost:8080/geoserver/rest/imports/0/tasks/0/transforms"
curl -u admin:geoserver -XPOST -H "Content-type: application/json" -d @gad.json
↪ "http://localhost:8080/geoserver/rest/imports/0/tasks/0/transforms"
```

gtx.json is:

```
{
  "type": "GdalTranslateTransform",
  "options": [ "-co", "TILED=YES" ]
}
```

gad.json is:

```
{
  "type": "GdalAddoTransform",
  "options": [ "-r", "average" ],
  "levels" : [2, 4, 8, 16, 32, 64, 128]
}
```

Now the import is ready to run, and we'll execute it using:

```
curl -u admin:geoserver -XPOST "http://localhost:8080/geoserver/rest/imports/0"
```

The new granule will be ingested into the mosaic, and will thus be available for time based requests.

Asynchronously fetching and importing data from a remote server

We assume a remote FTP server contains multiple shapefiles that we need to import in GeoServer as new layers. The files are large, and the server has much better bandwidth than the client, so it's best if GeoServer performs the data fetching on its own.

In this case a asynchronous request using remote data will be the best fit:

```
curl -u admin:geoserver -XPOST -H "Content-type: application/json" -d @import.json
↪ "http://localhost:8080/geoserver/rest/imports?async=true"
```

Where import.json is:

```
{
  "import": {
    "targetWorkspace": {
      "workspace": {
        "name": "topp"
      }
    },
    "data": {
      "type": "remote",
      "location": "ftp://myserver/data/bc_shapefiles",
      "username": "dan",
      "password": "secret"
    }
  }
}
```

The request will return immediately with an import context in “INIT” state, and it will remain in such state until the data is fetched and the tasks created. Once the state switches to “PENDING” the import will be ready for execution. Since there is a lot of shapefiles to process, also the import run will be done in asynchronous mode:

```
curl -u admin:geoserver -XPOST "http://localhost:8080/geoserver/rest/imports/0?
  ↪async=true"
```

The response will return immediately in this case as well, and the progress can be followed as the tasks in the import switch state.

GeoNode’s Ad-Hoc API

gsconfig

gsconfig is a python library for manipulating a GeoServer instance via the GeoServer RESTConfig API.

The project is distributed under a [MIT License](#) .

Installing

```
pip install gsconfig
```

For developers:

```
git clone git@github.com:boundlessgeo/gsconfig.git
cd gsconfig
python setup.py develop
```

Getting Help

There is a brief manual at <http://boundlessgeo.github.io/gsconfig/> . If you have questions, please ask them on the GeoServer Users mailing list: <http://geoserver.org/comm/> .

Please use the GitHub project at <https://github.com/boundlessgeo/gsconfig> for any bug reports (and pull requests are welcome, but please include tests where possible.)

Sample Layer Creation Code

```
from geoserver.catalog import Catalog
cat = Catalog("http://localhost:8080/geoserver/")
topp = cat.get_workspace("topp")
shapefile_plus_sidecars = shapefile_and_friends("states")
# shapefile_and_friends should look on the filesystem to find a shapefile
# and related files based on the base path passed in
#
# shapefile_plus_sidecars == {
#     'shp': 'states.shp',
#     'shx': 'states.shx',
#     'prj': 'states.prj',
#     'dbf': 'states.dbf'
# }
```

(continues on next page)

(continued from previous page)

```
# 'data' is required (there may be a 'schema' alternative later, for creating empty_
↪featuretypes)
# 'workspace' is optional (GeoServer's default workspace is used by... default)
# 'name' is required
ft = cat.create_featurestore(name, workspace=topp, data=shapefile_plus_sidecars)
```

Running Tests

Since the entire purpose of this module is to interact with GeoServer, the test suite is mostly composed of [integration tests](#). These tests necessarily rely on a running copy of GeoServer, and expect that this GeoServer instance will be using the default data directory that is included with GeoServer. This data is also included in the GeoServer source repository as `/data/release/`. In addition, it is expected that there will be a PostgreSQL database available at `postgres:password@localhost:5432/db`. You can test connecting to this database with the `psql` command line client by running `$ psql -d db -Upostgres -h localhost -p 5432` (you will be prompted interactively for the password.)

To override the assumed database connection parameters, the following environment variables are supported:

- DATABASE
- DBUSER
- DBPASS

If present, `psycpg` will be used to verify the database connection prior to running the tests.

If provided, the following environment variables will be used to reset the data directory:

GEOSERVER_HOME Location of git repository to read the clean data from. If only this option is provided `git clean` will be used to reset the data.

GEOSERVER_DATA_DIR Optional location of the data directory GeoServer will be running with. If provided, `rsync` will be used to reset the data.

GS_VERSION Optional environment variable allowing the catalog test cases to automatically download and start a vanilla GeoServer WAR from the web. Be sure that there are no running services on HTTP port 8080.

Here are the commands that I use to reset before running the `gsconfig` tests:

```
$ cd ~/geoserver/src/web/app/
$ PGUSER=postgres dropdb db
$ PGUSER=postgres createdb db -T template_postgis
$ git clean -dxff -- ../../../../data/release/
$ git checkout -f
$ MAVEN_OPTS="-XX:PermSize=128M -Xmx1024M" \
GEOSERVER_DATA_DIR=../../../../data/release \
mvn jetty:run
```

At this point, GeoServer will be running foregrounded, but it will take a few seconds to actually begin listening for http requests. You can stop it with `CTRL-C` (but don't do that until you've run the tests!) You can run the `gsconfig` tests with the following command:

```
$ python setup.py test
```

Instead of restarting GeoServer after each run to reset the data, the following should allow re-running the tests:

```
$ git clean -dxff -- ../../../../data/release/
$ curl -XPOST --user admin:geoserver http://localhost:8080/geoserver/rest/reload
```

More Examples - Updated for GeoServer 2.4+

Loading the GeoServer catalog using `gsconfig` is quite easy. The example below allows you to connect to GeoServer by specifying custom credentials.

```
from geoserver.catalog import Catalog
cat = Catalog("http://localhost:8080/geoserver/rest/", "admin", "geoserver")
```

The code below allows you to create a `FeatureType` from a Shapefile

```
geosolutions = cat.get_workspace("geosolutions")
import geoserver.util
shapefile_plus_sidecars = geoserver.util.shapefile_and_friends("C:/work/gsconfig/test/
↳data/states")
# shapefile_and_friends should look on the filesystem to find a shapefile
# and related files based on the base path passed in
#
# shapefile_plus_sidecars == {
#     'shp': 'states.shp',
#     'shx': 'states.shx',
#     'prj': 'states.prj',
#     'dbf': 'states.dbf'
# }
# 'data' is required (there may be a 'schema' alternative later, for creating empty_
↳featuretypes)
# 'workspace' is optional (GeoServer's default workspace is used by... default)
# 'name' is required
ft = cat.create_featurestore("test", shapefile_plus_sidecars, geosolutions)
```

It is possible to create JDBC Virtual Layers too. The code below allow to create a new SQL View called `my_jdbc_vt_test` defined by a custom sql.

```
from geoserver.catalog import Catalog
from geoserver.support import JDBCVirtualTable, JDBCVirtualTableGeometry,
↳JDBCVirtualTableParam

cat = Catalog('http://localhost:8080/geoserver/rest/', 'admin', '****')
store = cat.get_store('postgis-geoserver')
geom = JDBCVirtualTableGeometry('newgeom', 'LineString', '4326')
ft_name = 'my_jdbc_vt_test'
epsg_code = 'EPSG:4326'
sql = 'select ST_MakeLine(wkb_geometry ORDER BY waypoint) As newgeom, assetid,
↳runtime from waypoints group by assetid, runtime'
keyColumn = None
parameters = None

jdbc_vt = JDBCVirtualTable(ft_name, sql, 'false', geom, keyColumn, parameters)
ft = cat.publish_featuretype(ft_name, store, epsg_code, jdbc_virtual_table=jdbc_vt)
```

This example shows how to easily update a layer property. The same approach may be used with every catalog resource

```
ne_shaded = cat.get_layer("ne_shaded")
ne_shaded.enabled=True
cat.save(ne_shaded)
cat.reload()
```

Deleting a store from the catalog requires to purge all the associated layers first. This can be done by doing something like this:

```
st = cat.get_store("ne_shaded")
cat.delete(ne_shaded)
cat.reload()
cat.delete(st)
cat.reload()
```

There are some functionalities allowing to manage the ImageMosaic coverages. It is possible to create new ImageMosaics, add granules to them, and also read the coverages metadata, modify the mosaic Dimensions and finally query the mosaic granules and list their properties.

The gsconfig methods map the [REST APIs for ImageMosaic](#)

In order to create a new ImageMosaic layer, you can prepare a zip file containing the properties files for the mosaic configuration. Refer to the GeoTools ImageMosaic Plugin guide in order to get details on the mosaic configuration. The package contains an already configured zip file with two granules. You need to update or remove the `datastore.properties` file before creating the mosaic otherwise you will get an exception.

```
from geoserver.catalog import Catalog
cat = Catalog("http://localhost:8180/geoserver/rest")
cat.create_imagemosaic("NOAAWW3_NCOMultiGrid_WIND_test", "NOAAWW3_NCOMultiGrid_WIND_
↳test.zip")
```

By default the `cat.create_imagemosaic` tries to configure the layer too. If you want to create the store only, you can specify the following parameter

```
cat.create_imagemosaic("NOAAWW3_NCOMultiGrid_WIND_test", "NOAAWW3_NCOMultiGrid_WIND_
↳test.zip", "none")
```

In order to retrieve from the catalog the ImageMosaic coverage store you can do this

```
store = cat.get_store("NOAAWW3_NCOMultiGrid_WIND_test")
```

It is possible to add more granules to the mosaic at runtime. With the following method you can add granules already present on the machine local path.

```
cat.harvest_externalgranule("file://D:/Work/apache-tomcat-6.0.16/instances/data/data/
↳MetOc/NOAAWW3/20131001/WIND/NOAAWW3_NCOMultiGrid__WIND_000_20131001T000000.tif",
↳store)
```

The method below allows to send granules remotely via POST to the ImageMosaic. The granules will be uploaded and stored on the ImageMosaic index folder.

```
cat.harvest_uploadgranule("NOAAWW3_NCOMultiGrid__WIND_000_20131002T000000.zip", store)
```

To delete an ImageMosaic store, you can follow the standard approach, by deleting the layers first. *ATTENTION:* at this time you need to manually cleanup the data directory from the mosaic granules and, in case you used a DB datastore, you must also drop the mosaic tables.

```
layer = cat.get_layer("NOAAWW3_NCOMultiGrid_WIND_test")
cat.delete(layer)
cat.reload()
cat.delete(store)
cat.reload()
```

The method below allows you to load and update the coverage metadata of the ImageMosaic. You need to do this for every coverage of the ImageMosaic of course.

```
coverage = cat.get_resource_by_url("http://localhost:8180/geoserver/rest/workspaces/
↳ natocmre/coveragestores/NOAAWW3_NCOMultiGrid_WIND_test/coverages/NOAAWW3_
↳ NCOMultiGrid_WIND_test.xml")
coverage.supported_formats = ['GEOTIFF']
cat.save(coverage)
```

By default the ImageMosaic layer has not the coverage dimensions configured. It is possible using the coverage metadata to update and manage the coverage dimensions. *ATTENTION*: notice that the presentation parameters accepts only one among the following values { 'LIST', 'DISCRETE_INTERVAL', 'CONTINUOUS_INTERVAL' }

```
from geoserver.support import DimensionInfo
timeInfo = DimensionInfo("time", "true", "LIST", None, "ISO8601", None)
coverage.metadata = ({'dirName': 'NOAAWW3_NCOMultiGrid_WIND_test_NOAAWW3_NCOMultiGrid_
↳ WIND_test', 'time': timeInfo})
cat.save(coverage)
```

Once the ImageMosaic has been configured, it is possible to read the coverages along with their granule schema and granule info.

```
from geoserver.catalog import Catalog
cat = Catalog("http://localhost:8180/geoserver/rest")
store = cat.get_store("NOAAWW3_NCOMultiGrid_WIND_test")
coverages = cat.mosaic_coverages(store)
schema = cat.mosaic_coverage_schema(coverages['coverages'][0]['name'],
↳ store)
granules = cat.mosaic_granules(coverages['coverages'][0]['name'], store)
```

The granules details can be easily read by doing something like this:

```
granules['crs']['properties']['name']
granules['features']
granules['features'][0]['properties']['time']
granules['features'][0]['properties']['location']
granules['features'][0]['properties']['run']
```

When the mosaic grows up and starts having a huge set of granules, you may need to filter the granules query through a CQL filter on the coverage schema attributes.

```
granules = cat.mosaic_granules(coverages['coverages'][0]['name'], store,
↳ "time >= '2013-10-01T03:00:00.000Z'")
granules = cat.mosaic_granules(coverages['coverages'][0]['name'], store,
↳ "time >= '2013-10-01T03:00:00.000Z' AND run = 0")
granules = cat.mosaic_granules(coverages['coverages'][0]['name'], store,
↳ "location LIKE '%20131002T000000.tif'")
```

gsimporter

gsimporter is a python library for using GeoServer's importer API.

Installing

pip install gsconfig

or

git clone <https://github.com/boundlessgeo/gsimporter> cd gsimporter pip install .

Getting Help

Please use the GitHub project at <https://github.com/boundlessgeo/gsimporter> for any bug reports (and pull requests are welcome, but please include tests where possible.)

Running Tests

The tests are integration tests. These require having a running GeoServer instance with the community/importer modules installed. Because some of the tests use a PostgreSQL database, a data base is required to run. **It is strongly advised to run with a data directory you don't care about.**

The test suite will first attempt to verify a connection to GeoServer and a connection to the database. If the default values are not appropriate, provide them via environment variables on the command line or via *export*. For example:

```
GEOSERVER_BASE_URL=http://localhost:8080 python setup.py test
```

A convenient way to deal with connection or other settings (besides setting things up to use the defaults) is to put them all in a bash (or other shell) script.

The tests are designed to create a workspace named *importer* and *importer2* for use in testing. *importer* will be set to the default workspace. As much as possible, things are cleaned up after test execution.

To run all of the tests, one way is via *setup.py*. *python setup.py test* should do the trick.

If developing and finer grained control is desired, specific tests and other flags can be provided using *python test/uploadtests.py*. Supported arguments are:

- **-clean** delete layers and stores in the test workspaces. Useful for cleanup.
- **-skip-teardown** don't delete things after running. May cause errors but useful for a single test.

To run a single case (or drop the method name to run the whole class):

```
python test/uploadtests.py ErrorTests.test_invalid_file
```

3.2.1.5.8 Testing in GeoNode

The community encourages Test Driven Development (TDD) and the contribution of new tests to extend test coverage. Ideally every model, view, and utility should be covered by tests.

GeoNode has Unit, Integration and Javascript tests. The Unit tests are located in the tests file of every Django app (Maps, Layers, Documents, Catalogue, Search, Security etc).

The Integration, CSW and smoke tests are located under the tests folder).

Warning: The tests are meant to be ran using the SQLite database, some of them may fail using PostgreSQL or others. Therefore **remove or rename your local_settings.py file before running the tests.**

If running them in development mode make sure to have the jetty server shut down otherwise the test could get stuck. To make sure it is run:

```
$ paver stop
```

Unit Tests

To run the unit tests make sure you have the virtualenv active (if running GeoNode under virtualenv) then run:

```
$ paver test # or python setup.py test when testing development versions
```

This will produce a detailed test report.

It's possible to run just specific apps tests by using the Django command:

```
$ python manage.py test app/tests.py
```

For example:

```
$ python manage.py test geonode.maps.tests
```

To run a single test case or method (omit the method name to run the whole class), for example:

```
$ python manage.py test geonode.maps.tests:MapsTest.test_maps_search
```

These tests are based on the Python/Django unit test suite.

Integration Tests

To run the unit tests make sure you have the virtualenv active (if running GeoNode under virtualenv) then run:

```
$ paver test_integration # or python setup.py test_integration when testing_
↪ development versions
```

To run the CSW integration test run:

```
$ paver test_integration -n geonode.tests.csw
```

Like the unit tests above, it is also possible to test specific modules, for example:

```
$ paver test_integration -n geonode.tests.integration:GeoNodeMapTest.test_
↪ search_result_detail
```

To test with with coverage:

```
$ python manage.py test geonode.maps.tests -- --with-coverage --cover-
↪ package=geonode.maps
```

These tests are based on the Python/Django unit test suite.

Javascript Tests

Note: Javascript tests has been currently disabled in GeoNode. There is a plan to improve and re-enable them in the future.

3.2.1.5.9 Pavement.py and Paver

Paver is a python module that automates repetitive tasks like running documentation generators, moving files around, testing and downloading things using the convenience of Python's syntax and massive library of code. GeoNode comes with several paver tasks which save administrators and developers from having to manually perform repetitive operations from the command line. The tasks are stored in the pavement.py file in your GeoNode root directory and can be run with ``paver <task_name>`` from that directory.

Pavement Tasks

Here's a list of Pavement tasks maintained by the GeoNode development team.

deb

```
`paver deb`
```

Creates Debian packages.

Use the `key` option (or its shorter version `-k`) to specify the GPG key to sign the package with.

Use the `ppa` option (or its shorter version `-p`) to specify the PPA the package should be published to.

package

```
`paver package`
```

Creates a distributable tarball for GeoNode.

reset

```
`paver reset`
```

Resets the GeoNode development environment by deleting the development database and re-deploying the GeoServer data directory.

reset_hard

```
`paver reset_hard`
```

Cleans the local GeoNode git repository and removes untracked directories.

test

```
`paver test`
```

Runs the GeoNode unit tests.

test_integration

```
`paver test_integration`
```

Runs the GeoNode integration tests.

setup

```
`paver setup`
```

Installs GeoNode's Python dependencies using pip.

setup_data

```
`paver setup_data`
```

Loads sample GIS data from the gisdata python package.

Use the `type` option (or its shorter version `-t`) to only import a specific data type. Supported types are “vector”, “raster”, and “time.”

setup_geoserver

```
`paver setup_geoserver`
```

Downloads GeoServer and the Jetty Runner and then moves the GeoServer data directory to the correct location.

start

```
`paver start`
```

Starts the GeoNode development web server and GeoServer.

start_django

```
`paver start_django`
```

Starts the GeoNode development web server on the local machine.

Use the `bind` option (or its shorter version `-b`) to bind the development server to a specific IP address and port number.

start_geoserver

```
`paver start_geoserver`
```

Runs the local GeoServer using Jetty.

stop

```
`paver stop`
```

Stops the GeoNode development web server and GeoServer.

stop_django

```
`paver stop_django`
```

Stops the GeoNode development web server.

stop_geoserver

```
`paver stop_geoserver`
```

Stops GeoServer.

sync

```
`paver sync`
```

Synchronizes the database according to the GeoNode models and loads the GeoNode sample data.

static

```
`paver static`
```

Note: This task requires the [Node Package Manager](#) to be installed.

Downloads and installs GeoNode's static file dependencies and creates the production assets.

upgrade_db

```
`paver upgrade_db`
```

Updates database schemas from legacy GeoNode versions.

Use the `version` option (or its shorter version `-v`) to specify the GeoNode version when running this task.

3.2.1.5.10 Introduction to GeoNode Projects

GeoNode enables you to set up a complete site simply by installing the packages and adding your data. If you want to create your own project based on GeoNode, there are a several options available that enable you to customize the look and feel of your GeoNode site. You can add additional modules that are necessary for your own use case and to integrate your GeoNode project with other external sites and services.

This module assumes that you have installed a GeoNode site with the Ubuntu Packages and that you have a working GeoNode based on that setup. If you want to follow this same methodology on a different platform, you can follow this module and adapt as necessary for your environment.

Overview

GeoNode is an out-of-the-box, full-featured Spatial Data Infrastructure solution, but many GeoNode implementations require either customization of the default site or the use of additional modules, whether they be third-party Django Pluggables or modules developed by a GeoNode implementer.

There are quite a few existing Downstream GeoNode projects some of which follow the methodology described in this module. You should familiarize yourself with these projects and how and why they extend GeoNode. You should also carefully think about what customization and additional modules you need for your own GeoNode-based project and research the options that are available to you. The [Django Packages](#) site is a great place to start looking for existing modules that may meet your needs.

Existing downstream GeoNode projects

- [Harvard Worldmap](#)
- [MapStory](#)
- [Risiko/SAFE](#)
- [MetroBoston DataCommon](#)
- [WFP GeoNode](#)
- [ADRIPLAN](#)
- [ROGUE GeoNode](#)

Django template projects

GeoNode follows the Django template projects paradigm introduced in Django 1.4. At a minimum, a Django project consists of a `settings.py` file and a `urls.py` file; Django apps are used to add specific pieces of functionality. The GeoNode development team has created a template project which contains these required files with all the GeoNode configuration you need to get up and running with your own GeoNode project. If you would like learn more about Django projects and apps, you should consult the [Django Documentation](#)

3.2.1.5.11 Make a GeoNode release

Making a GeoNode release requires a quite complex preparation of the environment while once everything is set up is a really easy and quick task. As said the complex part is the preparation of the environment since it involves, the generation of a password key to be uploaded to the Ubuntu servers and imported in launchpad.

If you have already prepared the environment then jump to the last paragraph.

Before start, make sure to have a [pypi](#) and a [launchpad](#) account.

Before doing the release, a GeoNode team member who can already make release has to add you as a launchpad GeoNode team member.

Creating and importing a gpg key

A GPG key is needed to push and publish the package. There is a complete guide on how to [create and import](#) a GPG key

Preparing the environment

Make sure to have a Ubuntu 12.04 machine. Install the following packages in addition to the python virtualenv tools:

```
$ sudo apt-get install git-core git-buildpackage dbhelper devscripts
```

Get the GeoNode code (from master) in a virtualenv:

```
$ mkvirtualenv geonode
$ git clone https://github.com/GeoNode/geonode.git
$ cd geonode
```

Edit the .bashrc file and add the following lines (the key ID can be found in “your personal keys” tab:

```
export GPG_KEY_GEONODE="your\_gpg\_key\_id"
export DEBEMAIL=yourmail@mail.com
export EDITOR=vim
export DEBFULLNAME="Your Full Name"
```

then close and:

```
$ source .bashrc
```

Type “env” to make sure all the variables are correctly exported

Set the correct git email:

```
$ git config --global user.email "yourmail@mail.com"
```

Register your credentials:

```
$ python setup.py register
```

Make the release

The followings are the only commands needed if the environment and the various registrations have already been done.

Make sure to have pulled the master to the desired commit. Edit the file geonode/__init__.py at line 21 and set the correct version.

Install GeoNode in the virtualenv (make sure to have the virtualenv activated and be in the geonode folder):

```
$ pip install -e geonode
```

Publish the package:

```
$ cd geonode
$ paver publish
```

The last command will:

- Tag the release and push it to GitHub
- Create the debian package and push it at ppa:geonode/testing in launchpad
- Create the .tar.gz sources and push them to Pypi
- Update the changelog and commit it to master

Introduction to GeoNode development This module will introduce you to the components that GeoNode is built with, the standards that it supports and the services it provides based on those standards, and an overview its architecture.

Pavement.py and Paver Here's a list of Pavement tasks maintained by the GeoNode development team.

Django Overview This section introduces some basic concepts of Django, the Python based web framework on top of which GeoNode has been developed.

Django's primary goal is to ease the creation of complex, database-driven websites. Django emphasizes reusability and "pluggability" of components, rapid development, and the principle of *don't repeat yourself*. Python is used throughout, even for settings, files, and data models.

Django also provides an optional administrative create, read, update and delete interface that is generated dynamically through introspection and configured via admin models.

Development Prerequisites and Core Modules This module will introduce you to the basic tools and skills required to start actively developing GeoNode.

install_devmode This module shows a step-by-step guide for the setup of a GeoNode Development Environment on an Ubuntu system.

For other Linux distributions the commands are similar, the difference is mainly on the packages names.

Note: For Windows: (*Windows Binary Installer*)

GeoNode debugging techniques GeoNode can be difficult to debug as there are several different components involved. This module shows some techniques to debug the different parts of GeoNode.

GeoNode APIs This module provides an overview of the core modules and libraries used by GeoNode and teach to the user how to use them through some guided examples.

Testing in GeoNode This section explain how to run the tests on GeoNode.

Introduction to GeoNode Projects GeoNode enables you to set up a complete site simply by installing the packages and adding your data.

If you want to create your own project based on GeoNode, there are a several options available that enable you to customize the look and feel of your GeoNode site.

You can add additional modules that are necessary for your own use case and to integrate your GeoNode project with other external sites and services.

Make a GeoNode release Making a GeoNode release.

3.2.1.6 Advanced Workshop

Welcome to the GeoNode Training *Advanced Workshop* documentation v2.8.

This module introduces advanced techniques and methodologies for the management of the geospatial data and the maintenance and tuning of the servers on *Production Environments*.

The last sections of the module will teach also you how to add brand new classes and functionalities to your GeoNode installation.

Prerequisites You should be familiar with GeoNode, GeoServer, Python framework and development concepts other than with system administrator and caching concepts and techniques.

3.2.1.6.1 Advanced Data Management and Processing

Warning: Some parts of this section have been taken from the [GeoServer](#) project and training documentation.

Loading OSM Data into GeoNode

In this section, we will walk through the steps necessary to load OSM data into your GeoNode project. As discussed in previous sections, your GeoNode already uses OSM tiles from MapQuest and the main OSM servers as some of the available base layers. This session is specifically about extracting actual data from OSM and converting it for use in your project and potentially for Geoprocessing tasks.

The first step in this process is to get the data from OSM. We will be using the OSM Overpass API since it lets us do more complex queries than the OSM API itself. You should refer to the OSM Overpass API documentation to learn about all of its features. It is an extremely powerful API that lets you extract data from OSM using a very sophisticated API.

- http://wiki.openstreetmap.org/wiki/Overpass_API
- http://wiki.openstreetmap.org/wiki/Overpass_API/Language_Guide

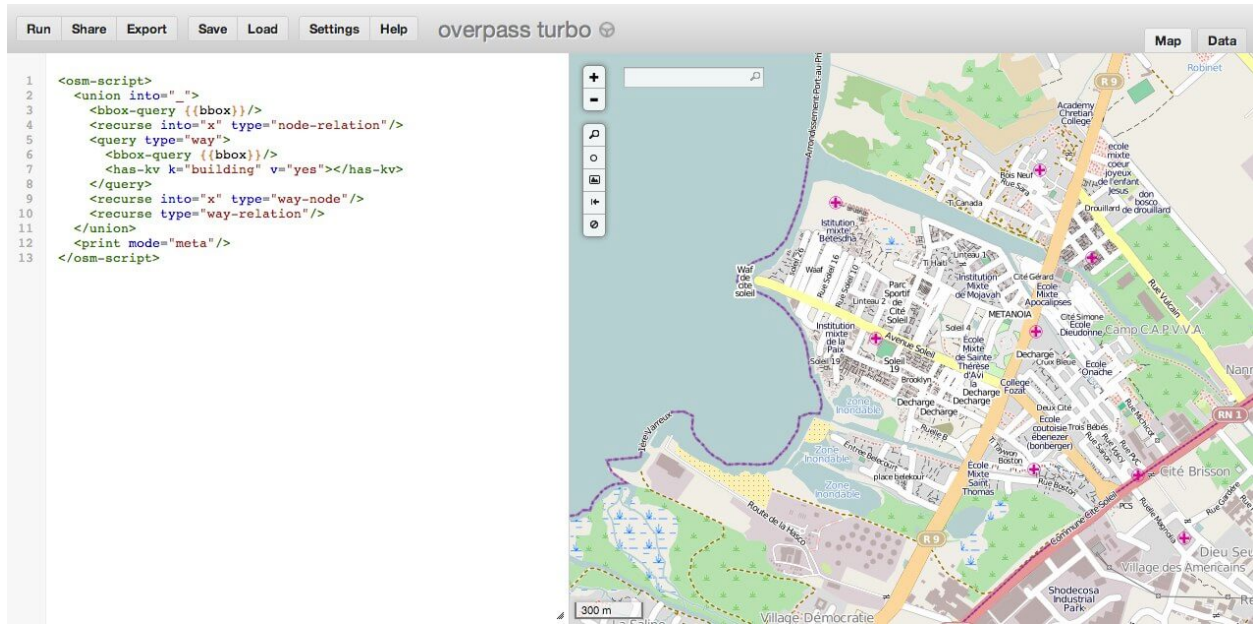
In this example, we will be extracting building footprint data around Port au Prince in Haiti. To do this we will use an interactive tool that makes it easy construct a Query against the Overpass API. Point your browser at <http://overpass-turbo.eu/> and use the search tools to zoom into Port Au Prince and Cité Soleil specifically.

You will need to cut and paste the query specified below to get all of the appropriate data under the bounding box:

```
<osm-script>
  <union into="_">
    <bbox-query {{bbox}}/>
    <recurse into="x" type="node-relation"/>
    <query type="way">
      <bbox-query {{bbox}}/>
      <has-kv k="building" v="yes"></has-kv>
    </query>
    <recurse into="x" type="way-node"/>
    <recurse type="way-relation"/>
  </union>
  <print mode="meta"/>
</osm-script>
```

This should look like the following.

When you have the bounding box and query set correctly, click the “Export” button on the menu to bring up the export menu, and then click the API interpreter link to download the OSM data base on the query you have specified.



This will download a file named 'export.osm' on your file system. You will probably want to rename it something else more specific. You can do that by issuing the following command in the directory where it was downloaded:

```
$ mv export.osm cite_soleil_buildings.osm
```

Note: You can also rename the file in your Operating Systems File management tool (Windows Explorer, Finder etc).

Exporting OSM data to shapefile using QGIS

Now that we have OSM data on our filesystem, we will need to convert it into a format suitable for uploading into your GeoNode. There are many ways to accomplish this, but for purposes of this example, we will use an OSM QGIS plugin that makes it fairly easy. Please consult the wiki page that explains how to install this plugin and make sure it is installed in your QGIS instance. Once its installed, you can use the Web Menu to load your file.

This will bring up a dialog box that you can use to find and convert the OSM file we downloaded.

When the process has completed, you will see your layers in the Layer Tree in QGIS.

Since we are only interested in the polygons, we can turn the other 2 layers off in the Layer Tree.

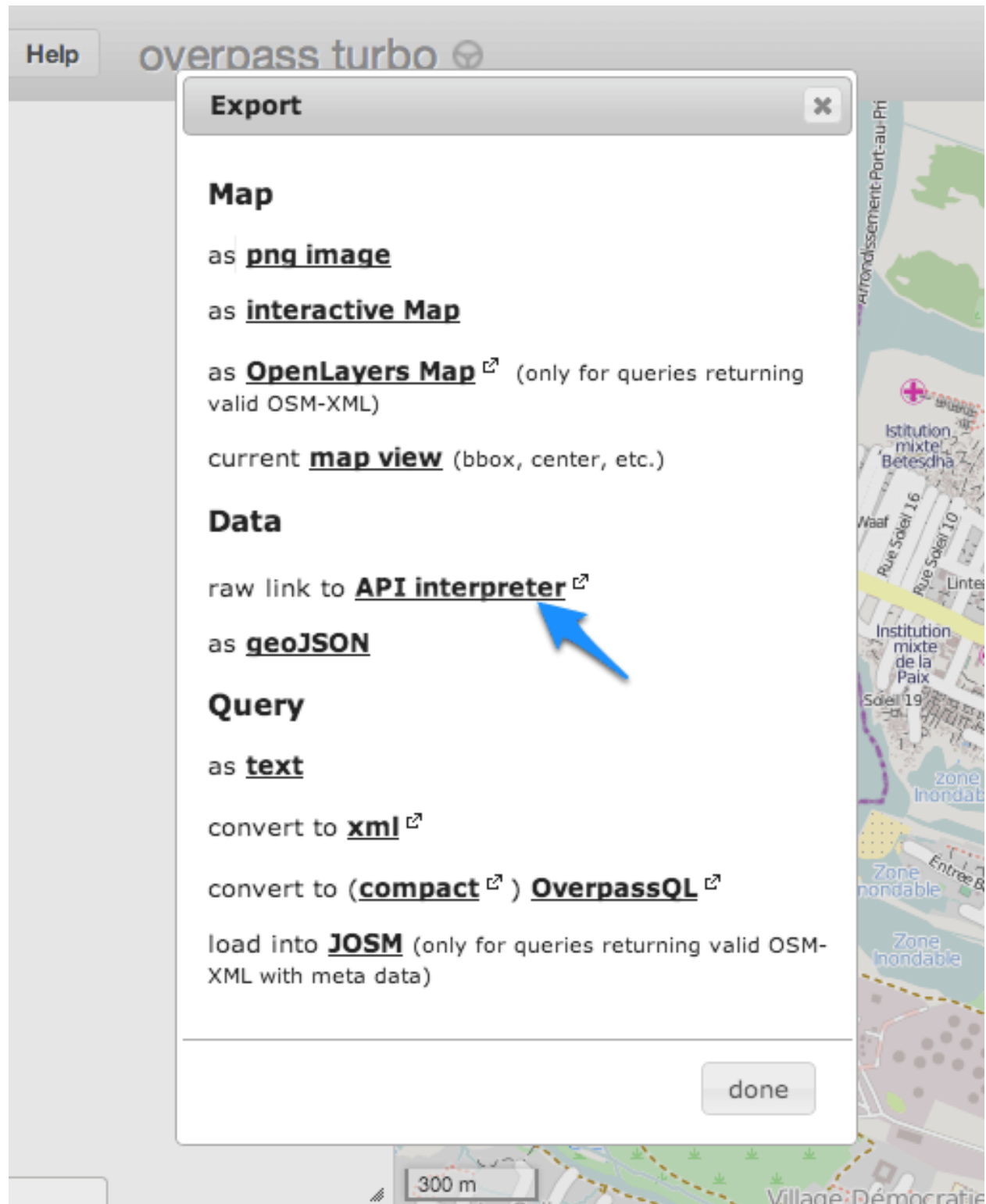
The next step is to use QGIS to convert this layer into a Shapefile so we can upload it into GeoNode. To do this, select the layer in the Layer tree, right click and then select the Save As option.

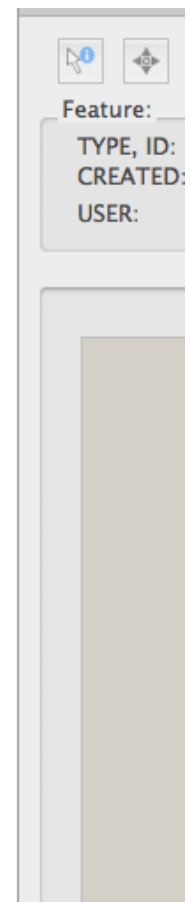
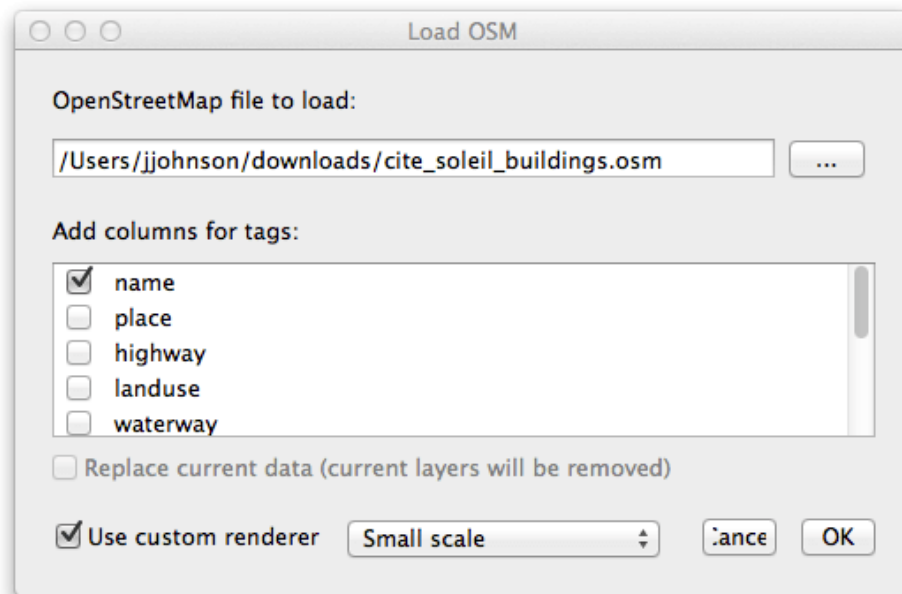
This will bring up the Save Vector Layer as Dialog.

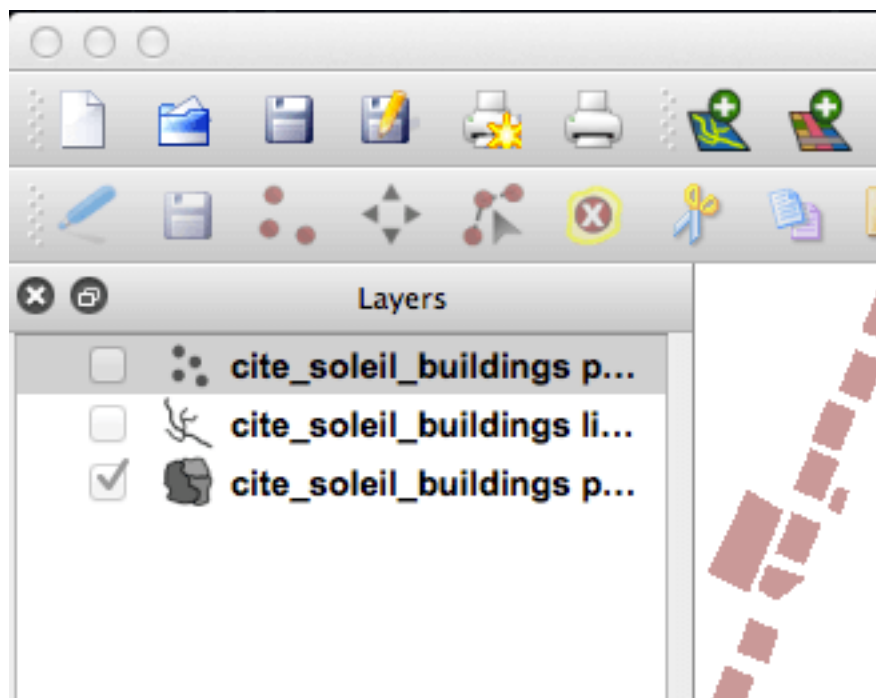
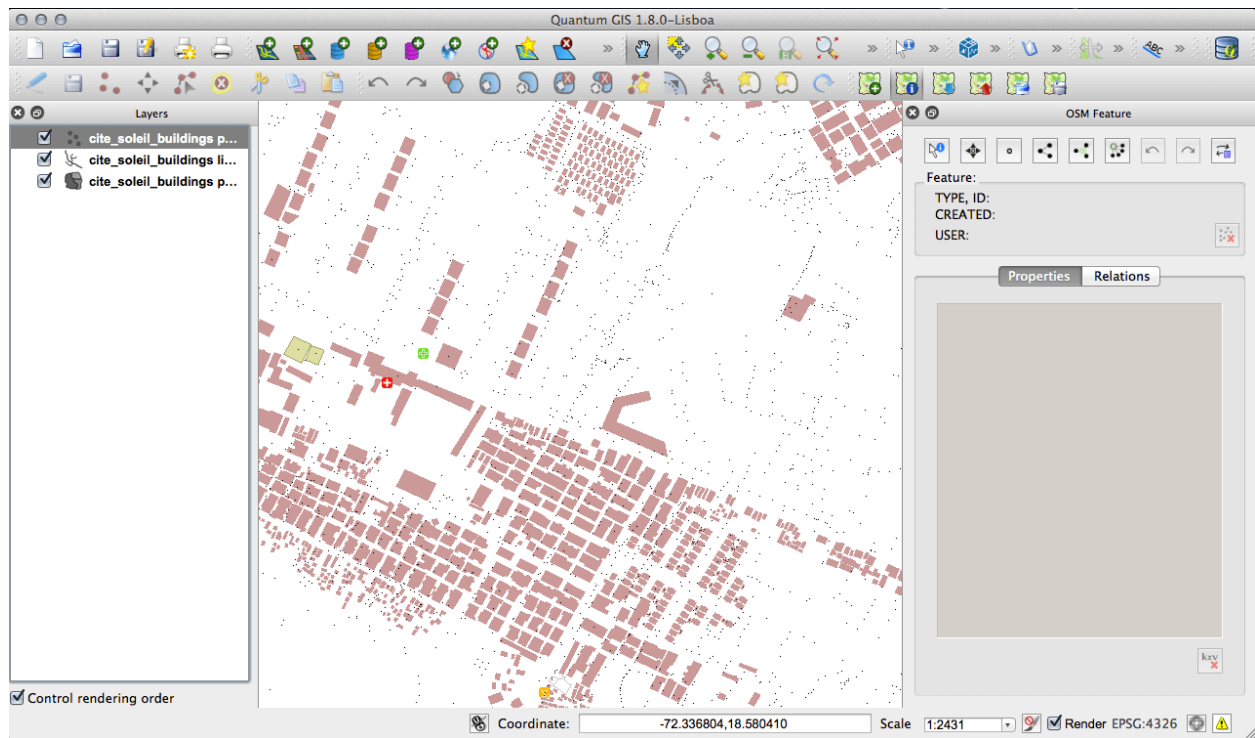
Specify where on disk you want your file saved, and hit Save then OK.

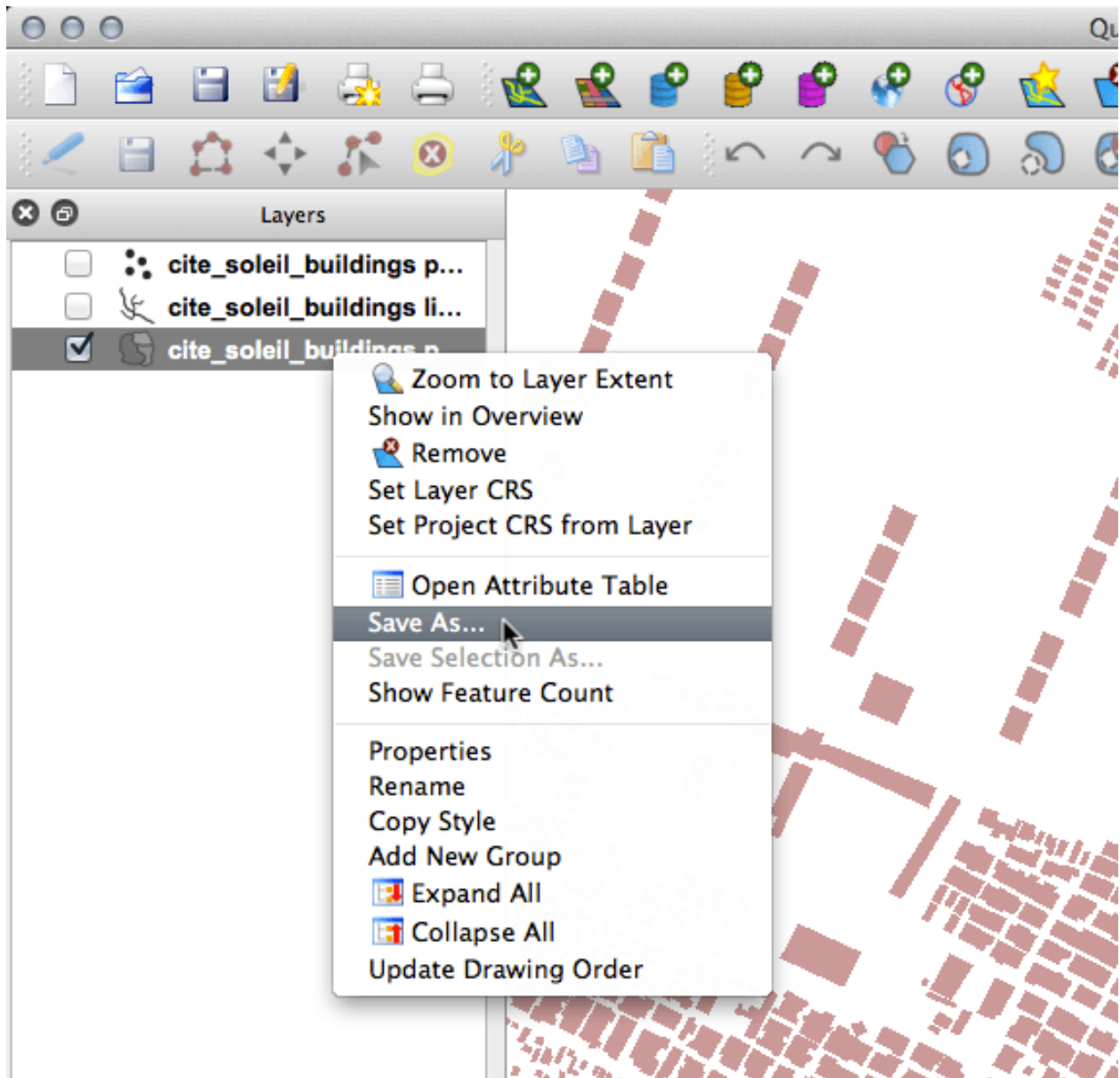
You now have a shapefile of the data you extracted from OSM that you can use to load into GeoNode. Use the GeoNode Layer Upload form to load the Shapefile parts into your GeoNode, and optionally edit the metadata and then you can view your layer in the Layer Info page in your geonode.

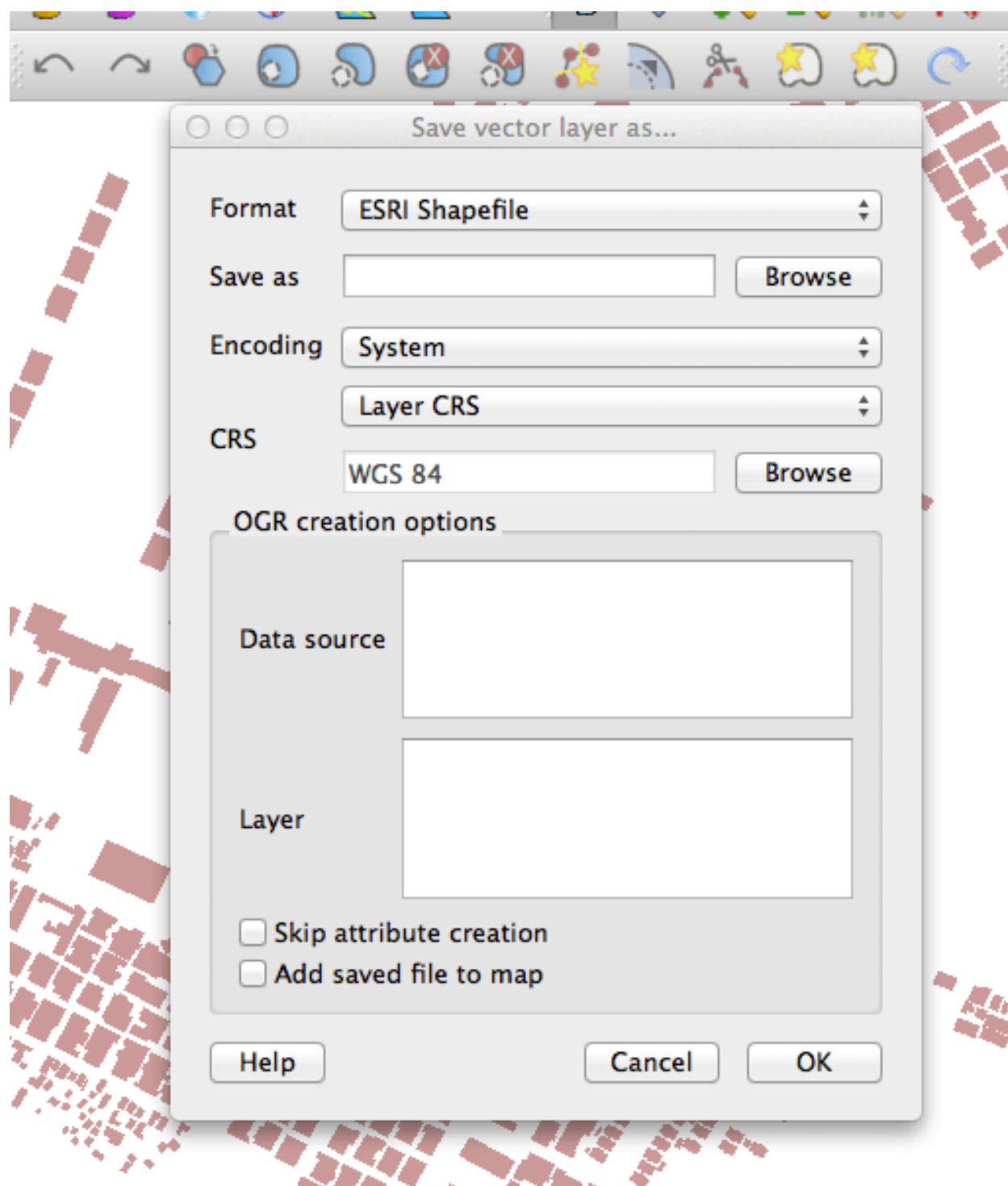
Note: You may want to switch to an imagery layer in order to more easily see the buildings on the OSM background.

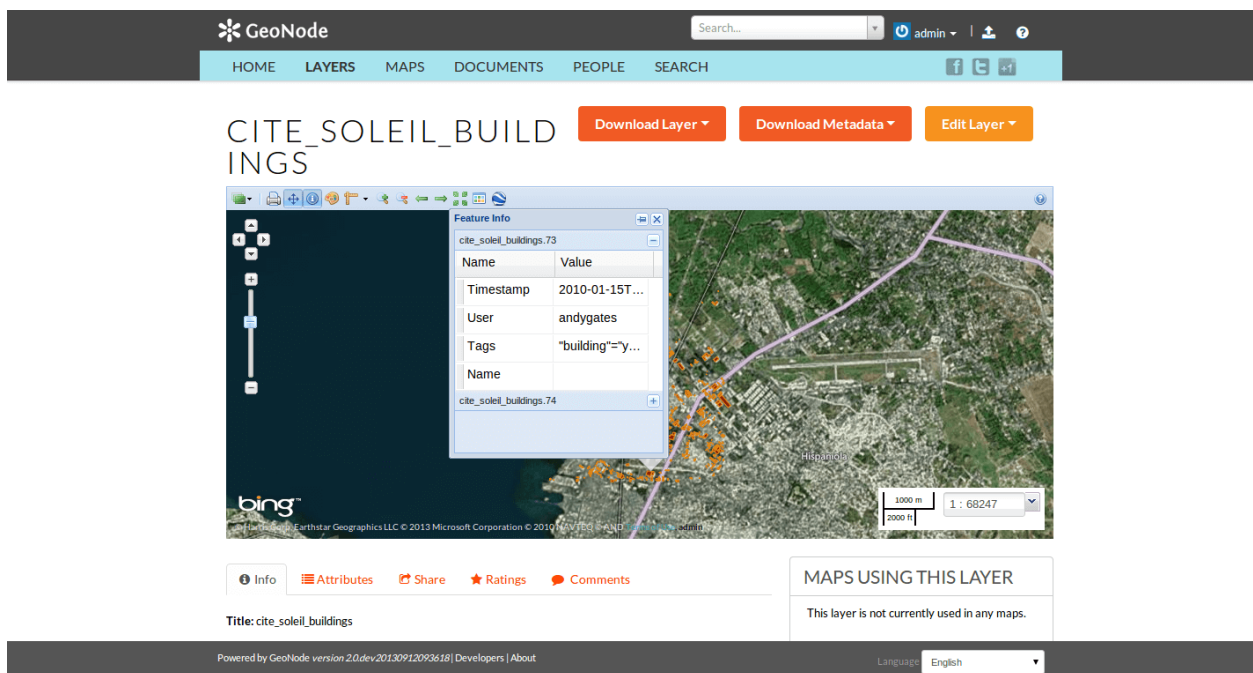
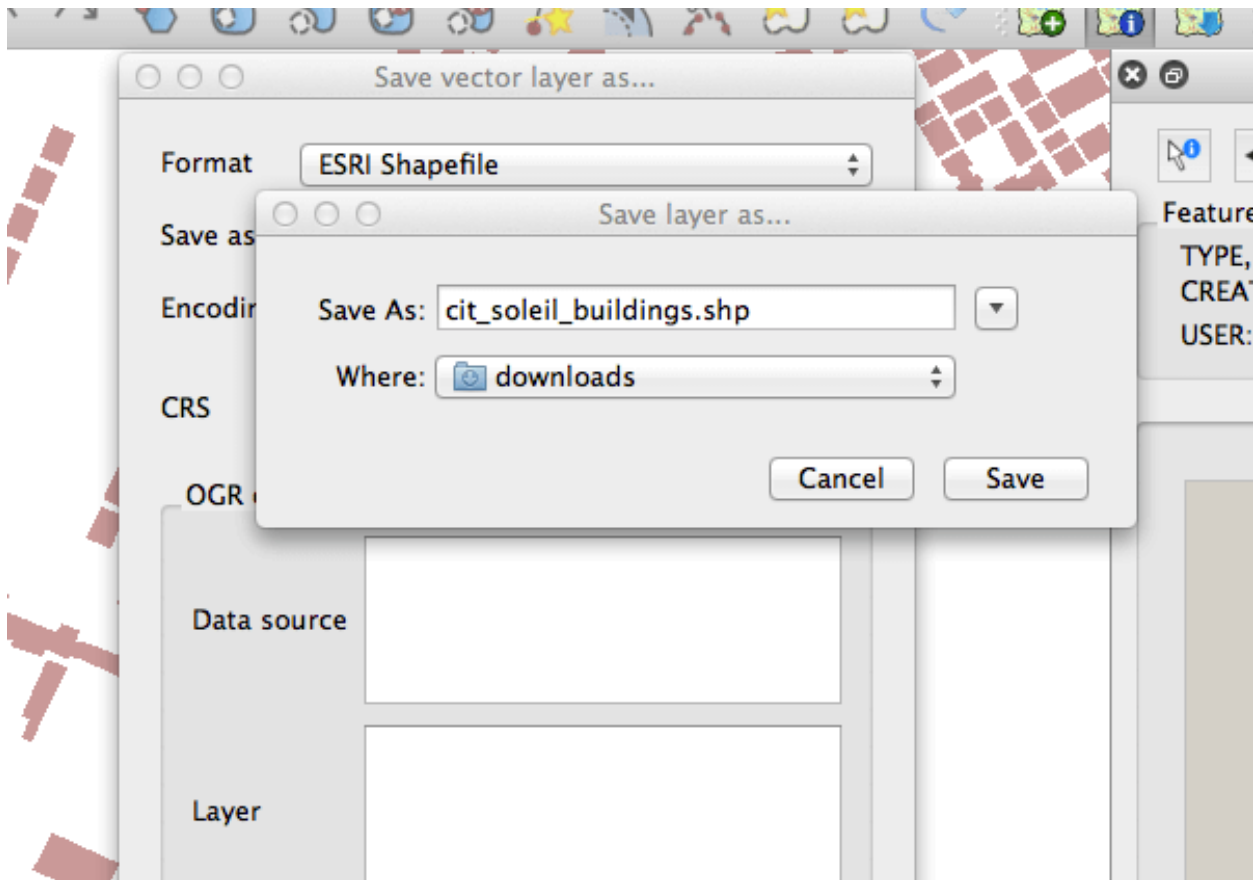












Exporting OSM data to shapefile using GDAL

An alternative way to export the .osm file to a shapefile is to use [ogr2ogr](#) combined with the [GDAL osm driver](#), available from GDAL version 1.10.

As a first step, inspect how the GDAL OSM driver sees the .osm file using the ogrinfo command:

```
$ ogrinfo cite_soleil_buildings.osm
Had to open data source read-only.
INFO: Open of `cite_soleil_buildings.osm'
      using driver `OSM' successful.
1: points (Point)
2: lines (Line String)
3: multilinestrings (Multi Line String)
4: multipolygons (Multi Polygon)
5: other_relations (Geometry Collection)
```

ogrinfo has detected 5 different geometric layers inside the OSM data source. As we are just interested in the buildings, you will just export the multipolygons layer to a new shapefile using the GDAL ogr2ogr command utility:

```
$ ogr2ogr cite_soleil_buildings cite_soleil_buildings.osm multipolygons
```

Now you can upload the shapefile to GeoNode using the GeoNode Upload form in the same manner as you did in the previous section.

Using GeoGig to load OSM Data into Manage OSM Data

Another alternative for working with OSM data in your GeoNode is to use [GeoGig](#). GeoGig is a tool that draws inspiration from Git but adapts its core concepts to handle distributed versioning of geospatial data. GeoGig allows you to load OpenStreetMap data into a repository on your server and either export that data into PostGIS for use in your GeoNode, or configure the GeoGig GeoServer extension and expose the repo directly from GeoServer. An article about this process can be found on the [Boundless Geo Blog](#), and will be described below. Much of the impetus for GeoGig came from the ROGUE JCTD and the technology has been incorporated into [Boundless Exchange](#).

Much more information about how to perform the steps below can be found in the GeoGig documentation page on [Using GeoGig with OpenStreetMap data](#). The instructions that follow are only a brief overview of the process.

Getting Started

You will first need to install the GeoGig command line tools. These can be found on the projects [SourceForge page](#). Follow the instructions contained in the installation file in order to install the CLI tools on your GeoNode server. Once you have the tools installed and added to your path, the *geogig* command should be available:

```
$ geogig --version
Project Version : 1.0-SNAPSHOT
Build Time : December 18, 2014 at 03:59:10 UTC
Build User Name : Unknown
Build User Email : Unknown
Git Branch : master
Git Commit ID : a4a80a8dd853dfe497729b35399594947866e8ae
Git Commit Time : December 18, 2014 at 03:44:24 UTC
Git Commit Author Name : Gabriel Roldan
Git Commit Author Email : gabriel.roldan@gmail.com
Git Commit Message : Synchronize DefaultPlatform.getTempDir() to avoid false_
↳ precondition check on concurrent access.
```


Once the GeoGig command is available, you will need to create an empty repository to hold your data. Change directories to a suitable location on your servers filesystem and issue the following command substituting *my_repo* for whatever name you choose:

```
$ cd /somewhere/on/file/system
$ mkdir my_repo
$ geogig init
Initialized empty Geogig repository in /somewhere/on/filesystem/my_repo/.geogig
```

Loading OSM Data into your Repository

Now that you have an empty repository to store our data, the next step is to load the current snapshot of OSM data into your repository using the *geogig osm download* command. At a minimum, you will want to use a bounding box filter to limit the downloaded data to the area of interest for your geonode installation. The example below is a bounding box that encompasses the country of Malawi. More information about the GeoGig OSM download command can be found in the [geogig docs](#):

```
$ geogig osm download --bbox -17.129459 32.668991 -9.364680 35.920441 --saveto ./mw-
↪osm-temp --keep-files

Connecting to http://overpass-api.de/api/interpreter...

Downloaded data will be kept in /somewhere/on/filesystem/my_repo/./mw-osm-temp

Importing into GeoGig repo...
1,164,420
1,164,420 entities processed in 5.892 min

Building trees for [node, way]

Trees built in 7.614 s
0%
Staging features...
100%

Committing features...
100%
Processed entities: 1,164,420.
  Nodes: 1,091,572.
  Ways: 72,848
```

GeoGig stores data in *trees* which are basically equivalent to *layers* in a normal geospatial context. At this point in the process, your repo contains 2 trees for *ways* and *nodes* from OSM. In order to convert these into layers that may be more familiar to your users like *roads*, *buildings* or *medical facilities*, you will need to apply a mapping that filters the complete list of nodes and ways and converts the tags into attributes. There are a great set of sample mappings in the US State Departments CyberGIS project. You can find them at this [link](#). You should clone this repository along side your GeoGig repository and then apply them as shown below:

```
$ geogig osm map ../cybergis-osm-mappings/mappings/basic/buildings_and_roads.json
$ geogig osm map ../cybergis-osm-mappings/mappings/health/medical_centers.json
$ geogig osm map ../cybergis-osm-mappings/mappings/education/schools.json
```

Now you can inspect the repository using the following commands:

```
$ geogig ls-tree
osm_roads
osm_buildings
node
way

$ geogig show osm_roads
TREE ID: 596a4f39ab9fadcbba6ffcaf5c135e29c2bc67d3
SIZE: 20288
NUMBER Of SUBTREES: 0
DEFAULT FEATURE TYPE ID: 0f7cbc6c114727858fb50668eb8a4448667bdc12

DEFAULT FEATURE TYPE ATTRIBUTES
id: <LONG>
geom: <LINESTRING>
status: <STRING>
media: <STRING>
name: <STRING>
ref: <STRING>
highway: <STRING>
lanes: <STRING>
oneway: <STRING>
surface: <STRING>
access: <STRING>
source: <STRING>
motor_vehicle: <STRING>
nodes: <STRING>
```

Updating the OSM Data in your Repository

As OpenStreetMap is a constantly changing dataset, you will want to periodically update your repo with the latest changes from OSM. GeoGig provides a way to do this using the *geogig osm download* command with the `--update` flag:

```
$ geogig osm download --update
```

Note: If you get an error that looks like the error below, this means that there are no changes in OSM since your last update and it can be ignored.

Committing features...

An unhandled error occurred: Nothing to commit after f249200302d5e808fb1b04f329b39b5853ffb7d0. See the log for more details.

Serving your GeoGig repository in GeoNode

At this point you have different options on how to serve this repository from your GeoNode. The most basic option is to *export* this data to PostGIS and configure the PostGIS database in GeoServer for use in GeoNode. First create a PostGIS database that will be used to store the data and then use the *geogig pg export* command to export the layers into this database for serving. Note you will need to replace the connection parameters below to match your servers setup:

```
$ geogig pg export --host localhost --port 5432 -- schema myschema --database my_osm_
↪database --user my_user --password my_password osm_train_stations osm_train_stations
```

Define your new primary key for the table we can't export (id is the osm id):

```
INSERT INTO gt_pk_metadata_table (table_schema, table_name, pk_column) VALUES (
↪'geogig_data', 'osm_train_stations', 'id');
```

Next you need to alter your OSM data table accordingly:

```
ALTER TABLE geogig_data.osm_train_stations DROP CONSTRAINT osm_train_stations_pkey;

ALTER TABLE geogig_data.osm_train_stations ADD PRIMARY KEY (id);

ALTER TABLE geogig_data.osm_train_stations DROP COLUMN fid;

ALTER TABLE geogig_data.osm_train_stations ADD COLUMN fid character varying(64);
```

Then you can run the *geogig pg export* command with the *-o* option to overwrite the table:

```
$ geogig pg export -o --host localhost --port 5432 -- schema myschema --database my_
↪osm_database --user my_user --password my_password osm_train_stations osm_train_
↪stations
```

At this point, you need to configure your PostGIS database connection in GeoServer. More information about this process can be found in the [GeoServer documentation](#).

Once the layers are configured in GeoServer. You want to issue the *updatelayers* to configure them in your GeoNode:

```
$ python manage.py updatelayers --store geogig_data --filter osm_train_stations
```

Using the GeoGig GeoServer Extension

The GeoGig project also contains a GeoServer extension that allows a GeoServer administrator to configure and serve the GeoGig store directly. This extension basically lets you treat your GeoGig repository as any other store of spatial data.

Note: This section is still to be completed.

Using the *osm-extract* script to download OSM Data into PostGIS

osm-extract is a script that allows to download data from OpenStreetMap, perform ETL procedures in order to classify data into layers and publish it in a PostgreSQL+PostGIS database. It is based on a fork from Terronodo and it is built around a Makefile with instructions that must be executed with the Linux *make* command (we assume you are working on a Linux based OS). Once OSM data have been loaded into PostGIS, they can be published in GeoNode.

The script processes the whole .pbk file provided in input, therefore the processing extent depends on the bounding box of the .pbk file itself. In addition data can be updated on a fixed frequency, by executing the Makefile with a scheduled cron job. In such case a sql instruction can be executed in order to update the publication date of the respective metadata published by GeoNode.

Steps for putting it in production

1. Download the repo
2. Install the dependencies: osmosis. (GeoNode is assumed to be up and running)
3. Launch the Makefile for the first time
4. Publish the layers in GeoNode, `update_layers`
5. Customize the sh file
6. Customize the SQL file
7. Schedule the shell file as a cron job

1. Download the repo:

```
git clone https://github.com/MalawiGeospatialTools/osm-extract.git
```

2. Install the dependencies

We assume that GeoNode is already installed on your machine. In addition to that you need to install osmosis, which is used by the Makefile to handle OSM data. In order to do so, follow the instructions at [Installing pre-built Osmosis](#)

3. Launch the Makefile for the first time

Set the current directory to the directory where the Makefile is stored. Then launch it by typing the following command:

```
make all NAME=<country> URL="<Planet.osm mirror>"
```

Substitute in the command `<country>` with the name of your country of interest (e.g. *malawi*), if you're working on a specific country (the name is used for naming the staging files created by the procedure).

Also replace `<Planet.osm mirror>` with one of the mirrors listed in <http://wiki.openstreetmap.org/wiki/Planet.osm#Downloading>; make sure that the mirror publishes extracts in .pbk format. One handy mirror is <http://download.openstreetmap.fr/extracts>, which published data on a country or area basis: this allows to reduce the processing steps to a single country or area of interest.

The procedure is going to create a new database in your PostgreSQL instance and store in it the OSM data for your country. Therefore you should run the Makefile with a user that has enough privileges.

The features and attributes to be included in each table and consequently in each GeoNode layer are defined in configuration files which are stored in the `conf` directory.

A few indications on the computing resources: 30 seconds of computing time are required for processing a .pbk file sized 38MB (the country of Malawi) with an Amazon m3 medium instance (1 Intel Xeon E5-2670 CPU, 3.75GB RAM).

4. Publish the layers in GeoNode

We propose to do so in two steps: firstly publish the layers in GeoServer and then in GeoNode. In GeoServer generate a new Store so that you can keep it separate from the default GeoNode Store. Then publish the layers of your interest from the ones that were created by the procedure at the previous step (please note that some of them may be empty, depending on the country of interest). In GeoNode take advantage of the `update_layers` command and publish all layers from the GeoServer Workspace created ad-hoc at the previous step. See the [update_layers documentation](#) for details.

5. Customize the sh file

Customize the `osm_update.sh` file in order to fit your server and software configuration, namely:

- define the installation path (on line 3) for the osmosis software, so that it can be found by the OS

- change the current directory to a working directory of your interest, where temporary files can be stored, deleted and updated (on line 4)
- define the name of the country of interest as well as the url (on line 6) as you did in step 3

6. Customize the SQL file

Customize the `set_pub_date.sql` file in order to fit it for your purpose. In particular substitute the store name `osm_extracts` with the name of the store in which your OSM data are in GeoServer.

7. Schedule the shell file as a cron job

Insert the `osm_update.sh` file in the crontab of your server as a scheduled job. In order to do so, please have a look at the official cron documentation. If you're using Ubuntu OS, please have a look [here](#).

Adding Data to GeoServer

Managing GeoServer Data Directory

This section explain how to manage the GeoServer Data Directory.

What you will learn

In this section you will:

Creating and setting a new GeoServer Data Directory

1. Generally if GeoServer is running in Web Archive mode inside of a servlet container, like in this Workshop, the data directory is located at `<web application root>/data` (the data directory contains the GeoServer configuration data).
2. The first thing to do is to correctly configure the `GEOSERVER_DATA_DIR`. To increase the portability of their data and to facilitate updates GeoServer, in the default Workshop configuration the `GEOSERVER_DATA_DIR` is configured under the directory:

```
${TRAINING_ROOT}/geoserver_data or %TRAINING_ROOT%\geoserver_data on Windows
```

Generally this is not an issue, but if you run the system from the Live DVD this folder resides in memory. The first thing to do is to move this folder into a local persistent storage.

- Move the `GEOSERVER_DATA_DIR` somewhere in the persistent storage using the command:

```
sudo mv -f ${TRAINING_ROOT}/geoserver_data <TARGET_DIR>
```

- Make a symbolic link to the `GEOSERVER_DATA_DIR` by issuing the command:

```
ln -s <TARGET_DIR> ${TRAINING_ROOT}/geoserver_data
```

Warning: Check that the user `geo` has permissions to read/write all the files/folder inside the `GEOSERVER_DATA_DIR`.

Note: Instead of creating a symbolic link you can configure GeoServer in order to allow it to point to the new GEOSERVER_DATA_DIR. To do that edit the file `/opt/tomcat-geoserver/webapps/geoserver/WEB-INF/web.xml` and modify the context param `GEOSERVER_DATA_DIR`.

Structure of the GeoServer Data Directory

The following is the `GEOSERVER_DATA_DIR` structure:

```
data_directory/  
  coverages/  
  data/  
  demo/  
  gwc/  
  gwc-layers/  
  layergroups/  
  logs/  
  palettes/  
  security/  
  styles/  
  temp/  
  user_projections/  
  validation/  
  workspaces/  
  www/  
  global.xml  
  gwc-gs.xml  
  logging.xml  
  wcs.xml  
  wfs.xml  
  wms.xml  
  wps.xml
```

File	Description
coverages	Contains some demo raster layers for this training
data	Not to be confused with the GeoServer data directory itself, the data directory is a location where actual data can be stored. This directory is commonly used to store shapefiles and raster files but can be used for any data that is file based. The main benefit of storing data files inside of the data directory is portability.
demo	The demo directory contains files which define the sample requests available in the Sample Request Tool.
gwc	This directory holds the cache created by the embedded GeoWebCache service.
gwc-layers	This directory holds the configuration files created by the embedded GeoWebCache service for each layer.
layergroups	Contains information on the layer groups configurations.
logs	This directory contains the GeoServer logging files (log file and logging properties files).
palettes	The palettes directory is used to store pre-computed Image Palettes. Image palettes are used by the GeoServer WMS as way to reduce the size of produced images while maintaining image quality.
security	The security directory contains all the files used to configure the GeoServer security subsystem. This includes a set of property files which define access roles, along with the services and data each role is authorized to access.
styles	The styles directory contains a number of Styled Layer Descriptor (SLD) files which contain styling information used by the GeoServer WMS.
temp	Temporary directory, used by the WPS service.
user_projections	The user_projections directory contains extra spatial reference system definitions. The epsg.properties can be used to add new spatial reference systems, whilst the epsg_override.properties file can be used to override an official definition with a custom one.
validation	This directory contains the validation rules
workspaces	The various workspaces directories contain metadata about stores and layers which are published by GeoServer. Each layer will have a layer.xml file associated with it, as well as either a coverage.xml or a featuretype.xml file depending on whether it's a raster or vector.
www	The www directory is used to allow GeoServer to act like a regular web server and serve regular files. While not a replacement for a full blown web server the www directory can be useful to easily serve OpenLayers map applications (this avoids the need to setup a proxy in order to respect the same origin policy).
global.xml	Contains settings that go across services, including contact information, JAI settings, character sets and verbosity.
gwc-gs.xml	Contains various settings for the GeoWebCache service.
logging.xml	Specifies the logging level, location, and whether it should log to std out.
wcs.xml	Contains the service metadata and various settings for the WCS service.
wfs.xml	Contains the service metadata and various settings for the WFS service.
wms.xml	Contains the service metadata and various settings for the WMS service.

Adding base types

This section explain how to add some of the base data types into GeoServer. As an example we will learn how to insert a shapefile and GeoTIFF into GeoServer, as well as how to import a Shapefile into PostGIS and then publish it from there.

What you will learn

In this section you will:

Adding a Shapefile

The task of adding a Shapefile is one that is core to any GIS tool. This section covers the task of adding and publishing a Shapefile with GeoServer.

1. Navigate to the workshop directory `$TRAINING_ROOT/data/user_data/` (on Windows `%TRAINING_ROOT%\data\user_data`) and find the following shapefiles:

```
Mainrd.shp
Mainrd.shx
Mainrd.dbf
Mainrd.prj
```

Copy the files to the following directory:

```
$GEOSERVER_DATA_DIR/data/boulder
```

for Windows:

```
%GEOSERVER_DATA_DIR%\data\boulder
```

Note: Ensure that all four parts of the shapefile are copied. This includes the `shp`, `shx`, `dbf`, and `prj` extensions.

2. Navigate to the GeoServer [Welcome Page](#).
3. On the Welcome page locate the *Login* form located at the top right corner, and enter the username “admin” and password “Geos”.
4. Click the *Add stores* link.
5. Select the *Shapefile* link and click it.

Note: The new data source menu contains a list of all the spatial formats supported by GeoServer. When creating a new data store one of these formats must be chosen. Formats like Shapefile and PostGIS are supported by default, and many other formats are available as extensions.

6. On the *Edit Vector Data Source* page enter “Mainrd” in the *Data Source Name* and *Description* fields. Finally click on browse link in order to set the Shapefile location in the *URL* field and click *Save*.

Note: The Mainrd.shp got just copied in the data directory, inside the “data/boulder” folder, and the file browser opens right in the data directory, so just click on “data” and then “boulder” and you’ll find it

7. After saving, you will be taken to a page that lists all the layers in the shapefile and gives you the option to publish any of them. Click *Publish*.
8. The *Coordinate Reference Systems* have to be manually populated. The *Name* and *Title* fields should be automatically populated.

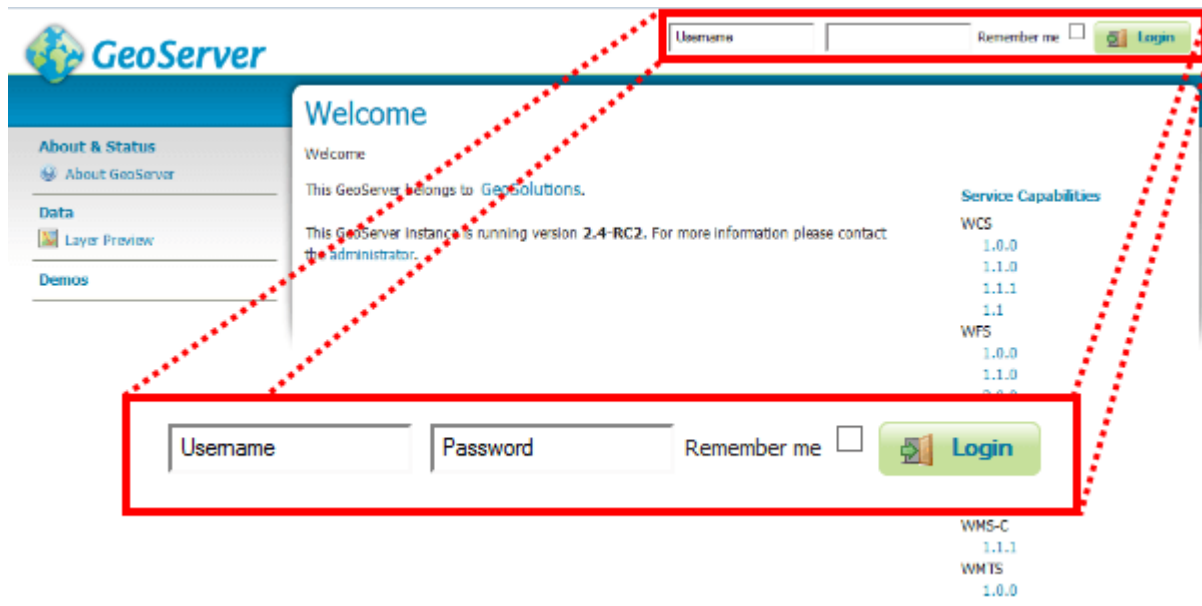


Fig. 114: GeoServer Login

Welcome

Welcome

This GeoServer belongs to [GeoSolutions](#).

24 Layers	+ Add layers
12 Stores	+ Add stores
1 Workspaces	+ Create workspaces

Fig. 115: Add stores link

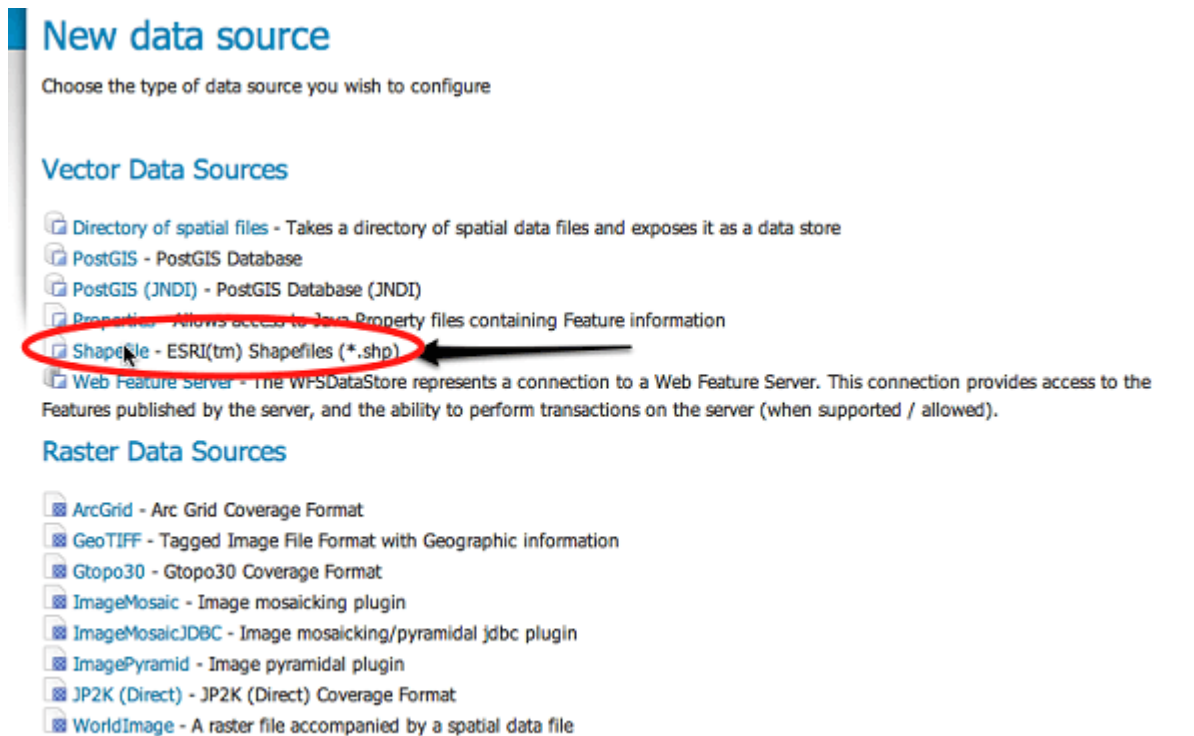


Fig. 116: Add a new shapefile

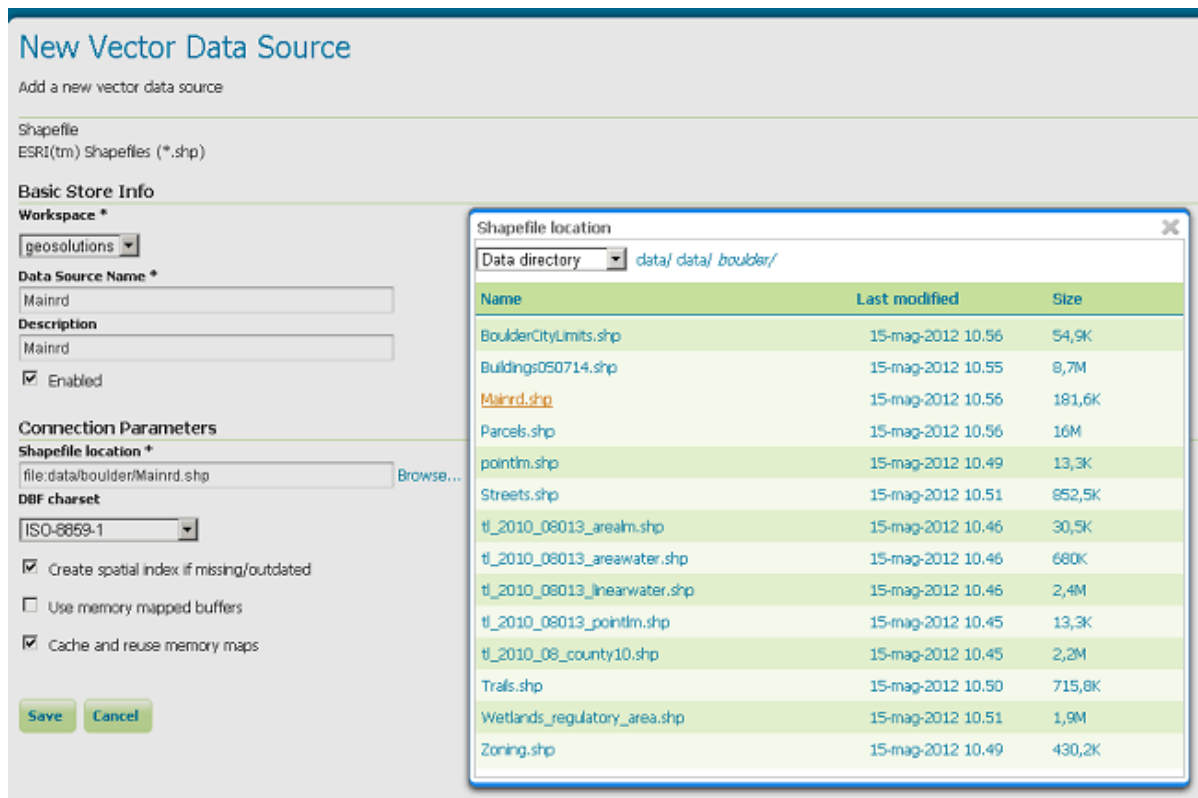


Fig. 117: Specifying Shapefile parameters

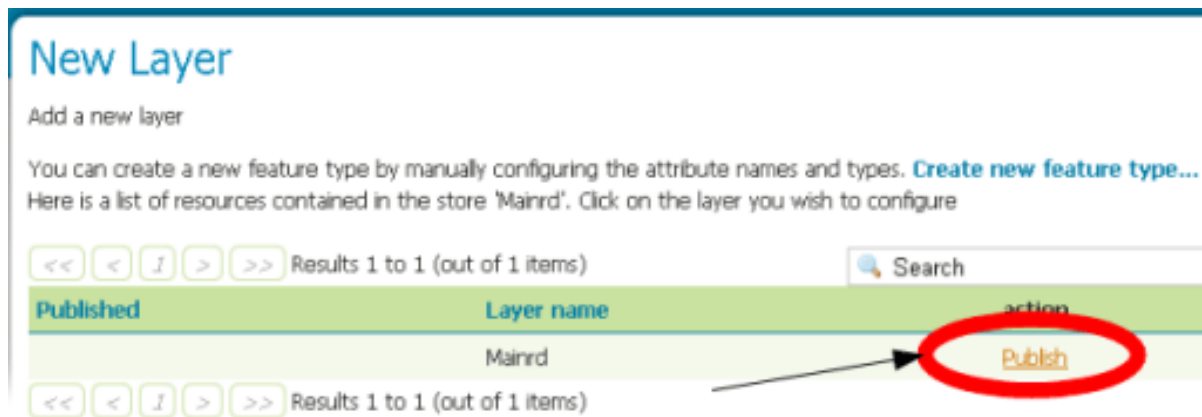


Fig. 118: Publishing a layer from the shapefile

Scroll down the page and generate the bounds for the layer by clicking the *Compute from data* button in the *Bounding Boxes* section.

9. Scroll to the bottom of the page, notice the read only *Feature Type Detail* table and then click *Save*.
 10. If all went well, you should see something like this:
At this point a shapefile has been added and is ready to be served by GeoServer.
 11. Choose the `preview` link in the main menu and filter the layer list with `mainrd`:
 12. Click on the `OpenLayers` link to preview the layer in an interactive viewer:
- In the *next* section we will see how to load a shapefile into PostGIS.

Loading a Shapefile into PostGIS

This task shows how to load a shapefile into PostGIS database:

1. Open the terminal window and enter the following command and press enter to creating a new database named 'shape':

- Linux:

```
createdb -U postgres -T postgis20 shape
```

- Windows:

```
setenv.bat
createdb -U postgres -T postgis20 shape
```

1. Enter the following command and press enter to load the shapefile into 'shape' database:

- Linux:

```
shp2pgsql -I ${TRAINING_ROOT}/data/user_data/Mainrd.shp public.main_
→roads | psql -d shape
```

- Windows:

Edit Layer
Edit layer data and publishing

geosolutions:Mainrd
Configure the resource and publishing information for the current layer

Data **Publishing**

Basic Resource Info

Name
Mainrd

Title
Mainrd

Abstract

Keywords

Current Keywords
features
Mainrd
Remove selected

New Keyword

Vocabulary

Add Keyword

Metadata links
No metadata links so far
Add link *Note only FGDC and TC211 metadata links show up in WMS 1.1.1 capabilities*

Coordinate Reference Systems

Native SRS
UNKNOWN [NAD_1983_HARN_StatePlane_Colorado_North_FIPS_0501...](#)

Declared SRS
EPSG:2876 [Find... EPSG:NAD83\(HARN\) / Colorado North \(ftUS\)...](#)

SRS handling
Force declared

Set the declared CRS to "EPSG:2876".

Fig. 119: Populate fields.

Bounding Boxes

Native Bounding Box

Min X	Min Y	Max X	Max Y
3.048.474,661	1.226.045,092	3.095.249	1.279.080,5

[Compute from data](#)

Lat/Lon Bounding Box

Min X	Min Y	Max X	Max Y
-105,32710918777	39,953380163165	-105,15953746505	40,099342759115

[Compute from native bounds](#)

Feature Type Details

Property	Type	Nullable	Min/Max Occurrences
the_geom	MultiLineString	true	0/1
LABEL_NAME	String	true	0/1

Use the links to autopopulate the bounding boxes.

Fig. 120: Generating the layer bounding box

Feature Type Details

Property	Type	Nullable	Min/Max Occurrences
the_geom	MultiLineString	true	0/1
LABEL_NAME	String	true	0/1

[Reload feature type](#) ⚠ ...

Save Button

Fig. 121: Submitting the layer configuration

Layers

Manage the layers being published by GeoServer

[Add a new resource](#)

[Remove selected resources](#)

Results 1 to 1 (out of 1 items)

<input type="checkbox"/>	Type	Workspace	Store	Layer Name	Enabled?	Native SRS
<input type="checkbox"/>		geosolutions	Mainrd	Mainrd	<input checked="" type="checkbox"/>	EPSG:2876

Results 1 to 1 (out of 1 items)

Fig. 122: After a successful save

Layers

Manage the layers being published by GeoServer

 [Add a new resource](#)

 [Remove selected resources](#)


 Results 1 to 16 (out of 16 items)


<input type="checkbox"/>	Type	Workspace	Store	Layer Name	Enabled?	Native SRS
<input type="checkbox"/>		geosolutions	Mainrd	Mainrd	✓	EPSG:2876
<input type="checkbox"/>		geosolutions	BoulderCityLimit	BoulderCityLimits	✓	EPSG:2876
<input type="checkbox"/>		geosolutions	Buildings050714	Buildings050714	✓	EPSG:2876
<input type="checkbox"/>		geosolutions	Parcels	Parcels	✓	EPSG:2876
<input type="checkbox"/>		geosolutions	pointlm	pointlm	✓	EPSG:2876
<input type="checkbox"/>		geosolutions	Streets	Streets	✓	EPSG:2876
<input type="checkbox"/>		geosolutions	Streets	Streets	✓	EPSG:2876
<input type="checkbox"/>		geosolutions	Streets	Streets	✓	EPSG:2876
<input type="checkbox"/>		geosolutions	tl_2010_08013_arealm	tl_2010_08013_arealm	✓	EPSG:2876
<input type="checkbox"/>		geosolutions	tl_2010_08013_areawater	tl_2010_08013_areawater	✓	EPSG:2876
<input type="checkbox"/>		geosolutions	tl_2010_08013_linearwater	tl_2010_08013_linearwater	✓	EPSG:2876
<input type="checkbox"/>		geosolutions	tl_2010_08013_pointlm	tl_2010_08013_pointlm	✓	EPSG:2876
<input type="checkbox"/>		geosolutions	tl_2010_08_county10	tl_2010_08_county10	✓	EPSG:2876
<input type="checkbox"/>		geosolutions	Trails	Trails	✓	EPSG:2876
<input type="checkbox"/>		geosolutions	Wetlands_regulatory_area	Wetlands_regulatory_area	✓	EPSG:2876
<input type="checkbox"/>		geosolutions	Zoning	Zoning	✓	EPSG:2876

 Results 1 to 16 (out of 16 items)

Layer Preview

List of all layers configured in GeoServer and provides previews in various formats for each.

 Results 1 to 20 (out of 0 matches from 20 items)

Type	Name	Title	Common Formats	All Formats
	geosolutions:Mainrd	Mainrd	OpenLayers KML GML	<input type="text" value="Select one"/>


 Results 1 to 20 (out of 0 matches from 20 items)

Fig. 123: Selecting the mainrd shapefile in the layer preview.

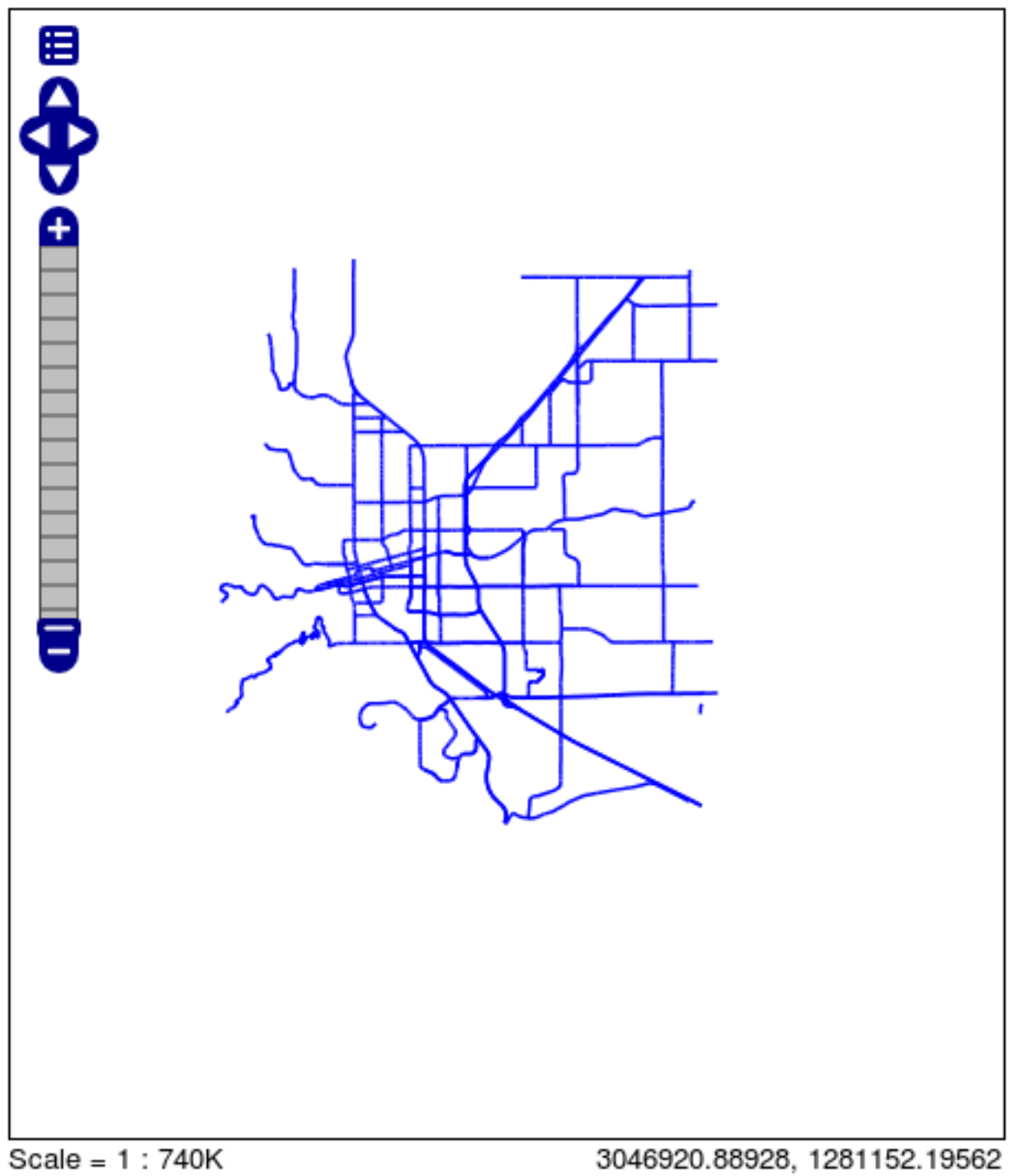


Fig. 124: The mainrd shapefile preview

```
shp2pgsql -I "%TRAINING_ROOT%\data\user_data\Mainrd.shp" public.main_
→roads | psql -U postgres -d shape
```

The shapefile will be loaded within the ‘main_roads’ table of the ‘shape’ database. The following screenshot shows some of the table contents in pgAdmin III

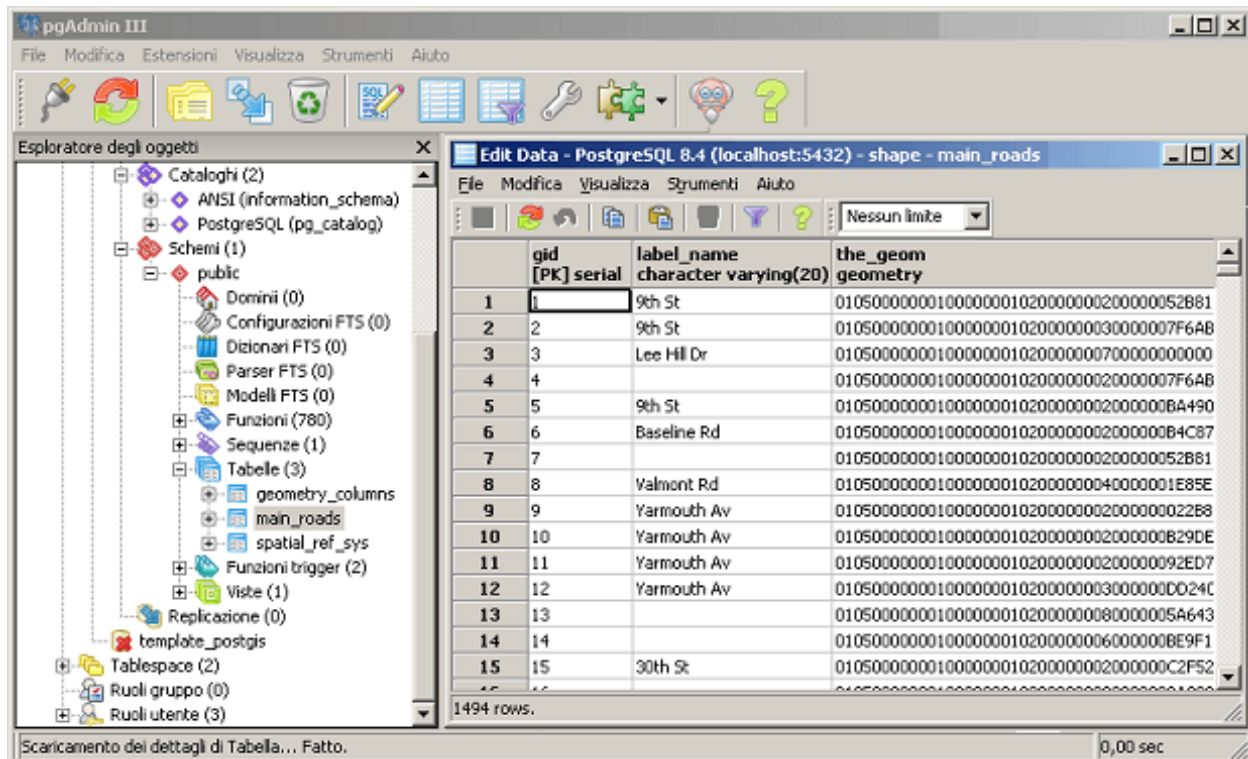


Fig. 125: A PostGIS table by ShapeFile

In the *next* section we will see how to add a PostGIS layer into GeoServer.

Adding a Postgis layer

This task shows how to add a PostGIS layer into GeoServer:

1. Navigate to the GeoServer [Welcome Page](#).
2. If you are not already logged in, on the Welcome page locate the *Login* form located at the top right corner, and enter the username “admin” and password “Geos”.
3. Click the *Add stores* link.
4. Select the *PostGIS* link and click it.
5. On the *New Vector Data Source* page fill the following parameter:
 - *Data source name*, ‘shape’
 - *port*, ‘5434’
 - *database*, ‘shape’ the name of the database created in previous workshop step.
 - *user*, ‘geonode’ the name of the user database owner.

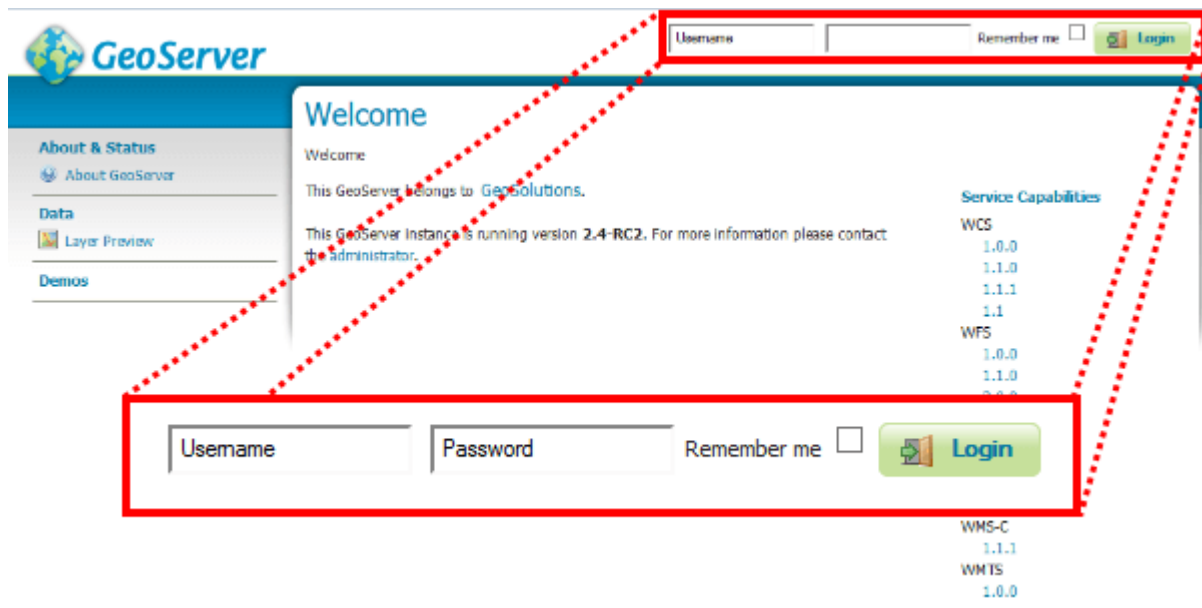


Fig. 126: GeoServer Login

Welcome

Welcome

This GeoServer belongs to GeoSolutions.

24 Layers

+ Add layers

12 Stores



+ Add stores

1 Workspaces

+ Create workspaces

Fig. 127: Add stores link

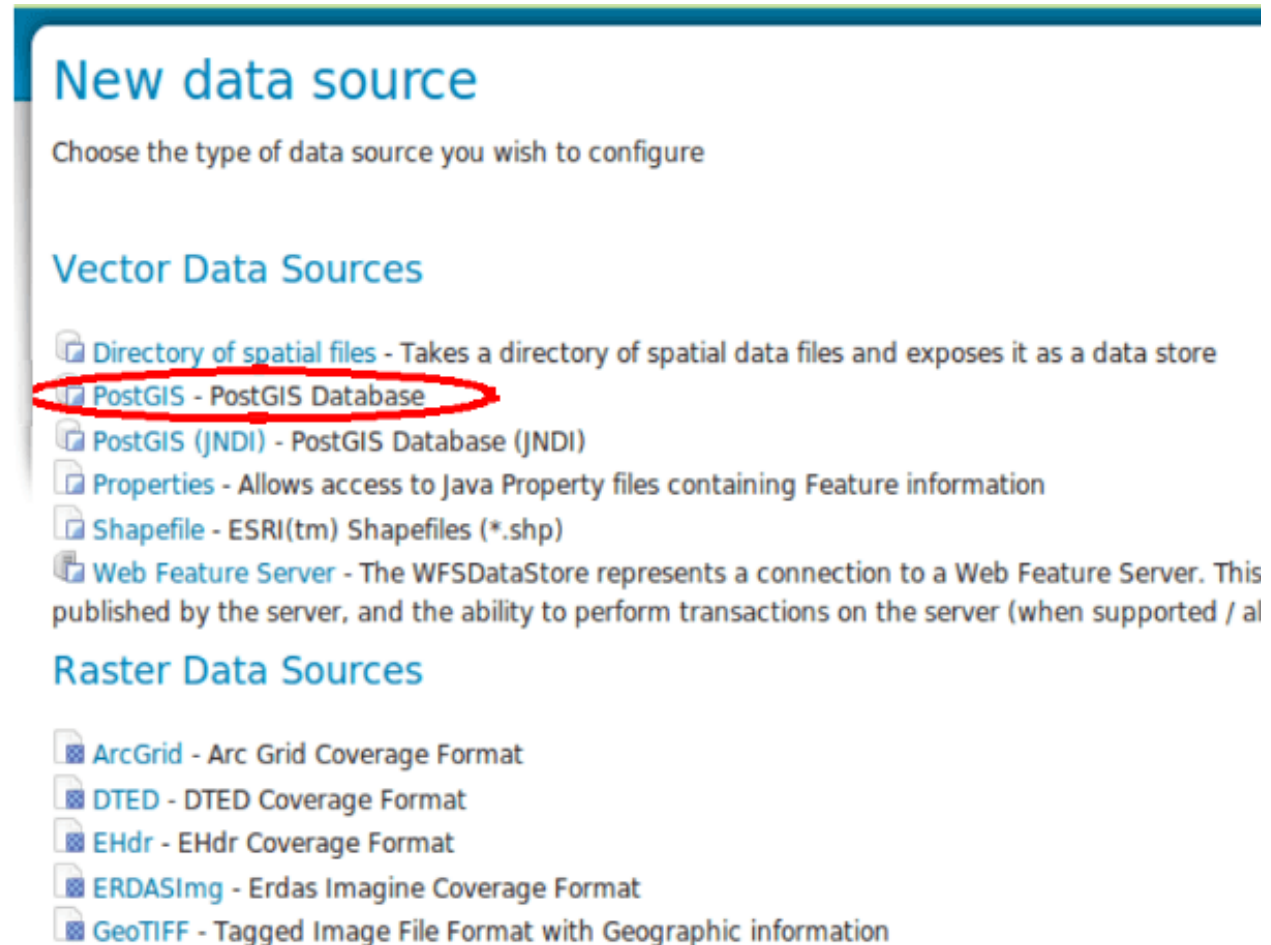


Fig. 128: Add new PostGIS Store

- *password*, ‘geonode’ the user password.

and click *Save*.

6. After saving, you will be taken to a page that lists all the layers in the PostGIS database and gives you the option to publish any of them. Click *Publish*.
7. The *Name* and *Title* fields should be automatically populated. Fill the *Declared SRS* field to set the Coordinate Reference Systems and generate the bounds for the layer by clicking the *Compute from data* and *Compute from native bounds* buttons in the *Bounding Boxes* section
8. Scroll to the bottom of the page, notice the read only *Feature Type Detail* table and then click *Save*.
9. If all went well, you should see something like this:
10. At this point the PostGIS layer has been added and is ready to be served by GeoServer. Use the layer preview to view its contents, filtering on the ‘main_road’ name.

Adding a GeoTIFF

GeoTIFF is a widely used geospatial raster data format. It is composed of a single file containing both the data and the georeferencing information (not to be confused with the .tiff/.tif/.prj file triplet, which is considered a “world image” file in GeoServer). This section provides instructions to add and publish a GeoTIFF file.

1. Open the web browser and navigate to the GeoServer [Welcome Page](#).
2. Select *Add stores* from the interface.
3. Select *GeoTIFF - Tagged Image File Format with Geographic information* from the set of available Raster Data Sources.
4. Specify a proper name (as an instance, 13tde815295_200803_0x6000m_cl) in the *Data Source Name* field of the interface.
5. Click on browse link in order to set the GeoTIFF location in the *URL* field.

Note: The 13tde815295_200803_0x6000m_cl.tif is located at \$TRAINING_ROOT/data/user_data/aerial/13tde815295_200803_0x6000m_cl.tif (on Windows %TRAINIG_ROOT%\data\user_data\airial\13tde815295_200803_0x6000m_cl.tif)

6. Click *Save*.
7. Publish the layer by clicking on the *publish* link.
8. Check the Coordinate Reference Systems and the Bounding Boxes fields are properly set and click on *Save*.
9. At this point the GeoTIFF is being published with GeoServer. You can use the layer preview to inspect the data.

Adding a WMS Cascade Layer

WMS cascading allows to expose layers coming from other WMS servers as if they were local layers. This provides for some interesting advantages:

- Clients connecting to your SDI need to care about less points of origin, which might be important for high security networks
- It is now possible to ask for maps in formats not supported by the original server, or to reproject the maps in projections not supported by the original server (GeoServer supports out of the box almost 5000 different coordinate reference systems)

New Vector Data Source

Add a new vector data source

PostGIS

PostGIS Database

Basic Store Info

Workspace *

geosolutions ▼

Data Source Name *

shape

Description

☒ Enabled

Connection Parameters

host *

localhost

port *

5434

database

shape

schema

public

user *

geosolutions

passwd

••••

Namespace *

<http://www.geo-solutions.it/workshop>

☐ Expose primary keys

max connections

10

Primary key metadata table**Session startup SQL****Session close-up SQL**

- ☒ Loose bbox
- ☒ Estimated extends
- ☐ preparedStatements

Max open prepared statements

- ☐ encode functions
- ☒ Support on the fly geometry simplification
- ☐ create database

create database params

Fig. 129: Setting database connection parameters

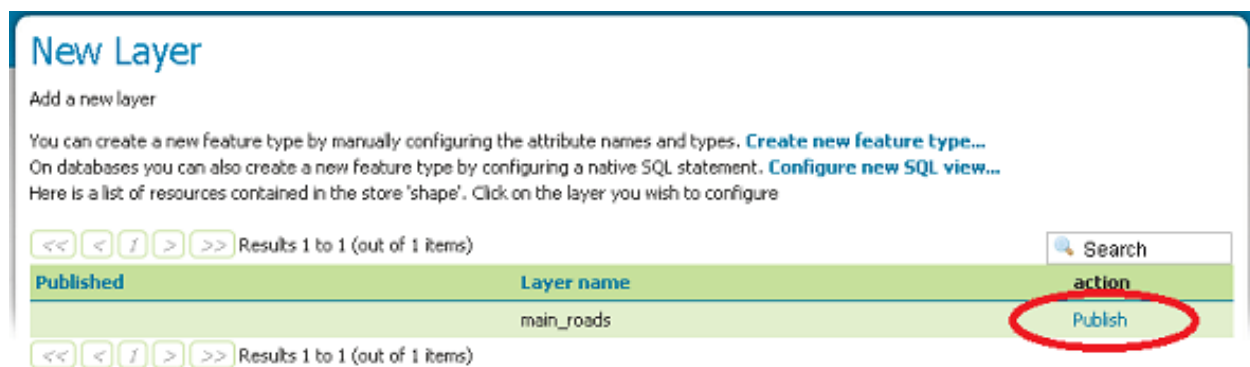
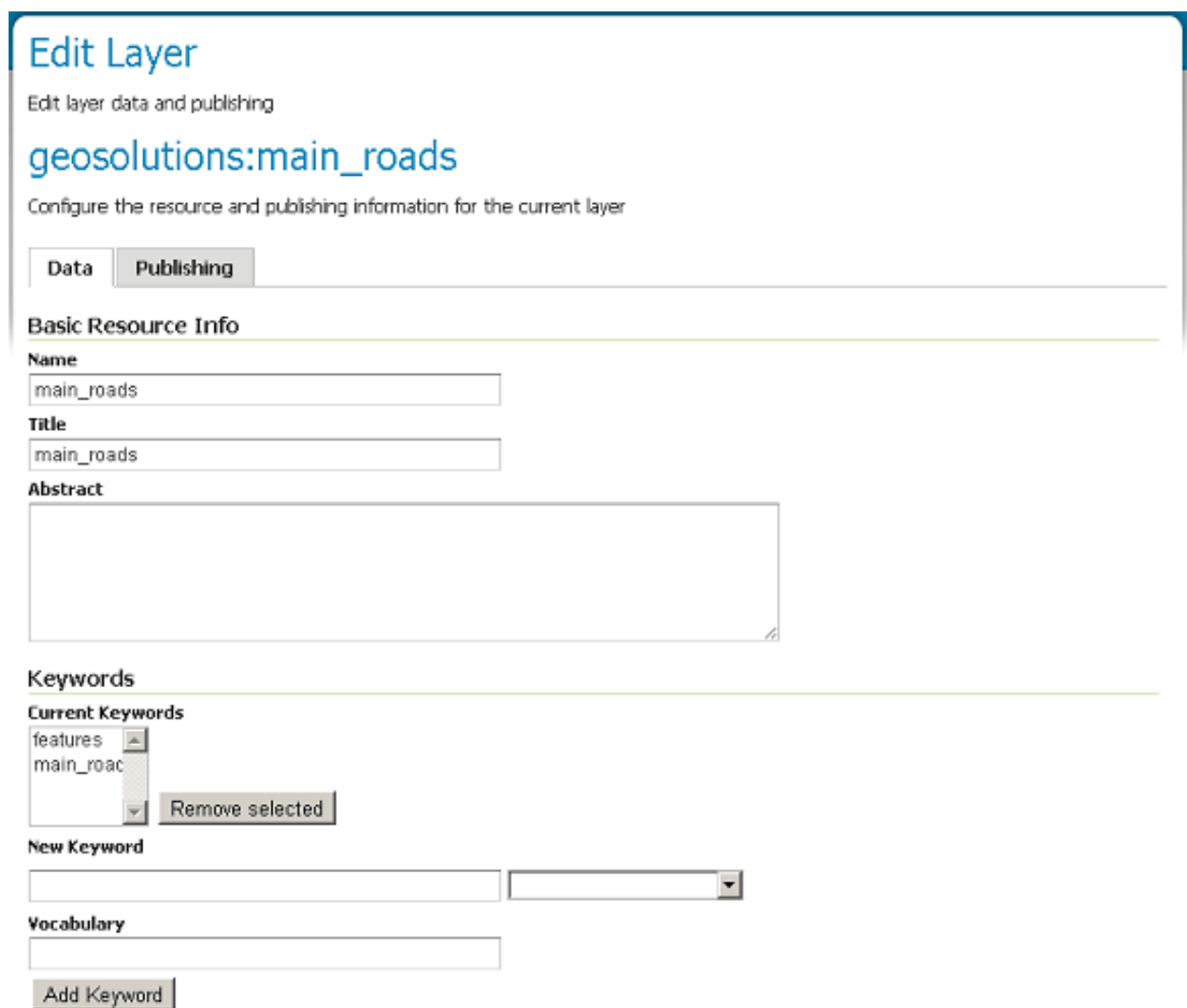


Fig. 130: Publishing a layer from the PostGIS table



Metadata links

No metadata links so far

[Add link](#) *Note only FGDC and TC211 metadata links show up in WMS 1.1.1 capabilities*

Coordinate Reference Systems

Native SRS

 ...

Declared SRS

EPSG:2876

[Find...](#)

EPSG:NAD83(HARN) / Colorado North (ftUS)...

SRS handling

Force declared

Set the Declared CRS to EPSG:2876 and after use the links to autopopulate the bounding boxes

Bounding Boxes

Native Bounding Box

Min X	Min Y	Max X	Max Y
3,048,474,5	1,226,045	3,095,249	1,279,080,5

[Compute from data](#)

Lat/Lon Bounding Box

Min X	Min Y	Max X	Max Y
-105,3271097627	39,953379910609	-105,15953746505	40,099342759979

[Compute from native bounds](#)

Fig. 131: Populating fields and generating the layer bounding box

Feature Type Details

Property	Type	Nullable	Min/Max Occurrences
label_name	String	true	0/1
the_geom	MultiLineString	true	0/1

[Reload feature type](#) ⚠ ...

Save

Cancel

Fig. 132: Submitting the layer configuration

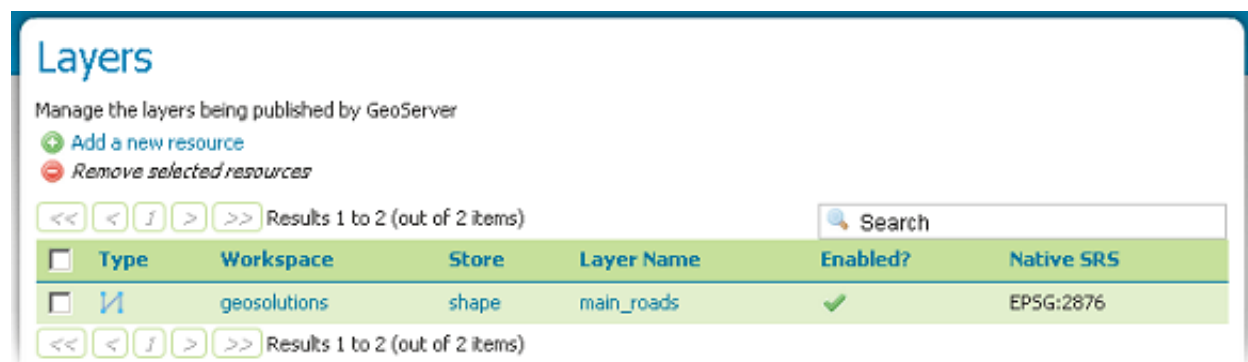


Fig. 133: After a successful save

This GeoServer belongs to [GeoSolutions](#).

21 Layers

[+ Add layers](#)

21 Stores

[+ Add stores](#)

1 Workspaces

[+ Create workspaces](#)

Raster Data Sources

- [ArcGrid](#) - Arc Grid Coverage Format
- [DTED](#) - DTED Coverage Format
- [EHdr](#) - EHdr Coverage Format
- [ERDASImg](#) - Erdas Imagine Coverage Format
- [GeoTIFF](#) - Tagged Image File Format with Geographic information
- [Gtopo30](#) - Gtopo30 Coverage Format
- [ImageMosaic](#) - Image mosaicking plugin
- [ImagePyramid](#) - Image pyramidal plugin
- [JP2K \(Direct\)](#) - JP2K (Direct) Coverage Format
- [JP2MrSID](#) - JP2K (MrSID) Coverage Format
- [MrSID](#) - MrSID Coverage Format
- [NITF](#) - NITF Coverage Format
- [WorldImage](#) - A raster file accompanied by a spatial data file

Add Raster Data Source

Description

GeoTIFF

Tagged Image File Format with Geographic Information

Basic Store Info

Workspace *

geosolutions

Data Source Name *

13tde815295_200803_0x6000m_cl

Description

13tde815295_200803_0x6000m_cl

☒ Enabled

Connection Parameters

URL *

file:data/example.extension

Save

Cancel

URL

13tde770310_200803_0x6000m_cl.tif	14-mag-2012 12.11	1,7M
13tde770325_200803_0x6000m_cl.tif	14-mag-2012 12.10	1,5M
13tde770340_200803_0x6000m_cl.tif	14-mag-2012 12.10	984,7K
13tde785280_200803_0x6000m_cl.tif	14-mag-2012 12.10	1,5M
13tde785295_200803_0x6000m_cl.tif	14-mag-2012 12.11	1,5M
13tde785310_200803_0x6000m_cl.tif	14-mag-2012 12.10	1,4M
13tde785325_200803_0x6000m_cl.tif	14-mag-2012 12.11	1,2M
13tde785340_200803_0x6000m_cl.tif	14-mag-2012 12.09	1,1M
13tde800280_200803_0x6000m_cl.tif	14-mag-2012 12.11	1,5M
13tde800295_200803_0x6000m_cl.tif	14-mag-2012 12.11	1,4M
13tde800310_200803_0x6000m_cl.tif	14-mag-2012 12.10	1,2M
13tde800325_200803_0x6000m_cl.tif	14-mag-2012 12.10	1,1M
13tde815280_200803_0x6000m_cl.tif	14-mag-2012 12.11	1,3M
<u>13tde815295_200803_0x6000m_cl.tif</u>	14-mag-2012 12.10	1,1M
13tde815310_200803_0x6000m_cl.tif	14-mag-2012 12.10	1,1M

New Layer

Add a new layer

Here is a list of resources contained in the store '13tde815295_200803_0x6000m_cl'.

Click on the layer you wish to configure

<< < I > >> Results 1 to 1 (out of 1 items)

Search

Published	Layer name	action
	13tde815295_200803_0x6000m_cl	Publish

<< < I > >> Results 1 to 1 (out of 1 items)

Coordinate Reference Systems

Native SRS

[EPSG:NAD83 / UTM zone 13N...](#)

Declared SRS

[EPSG:NAD83 / UTM zone 13N...](#)

SRS handling

Bounding Boxes

Native Bounding Box

Min X	Min Y	Max X	Max Y
481.500	4.429.500	483.000	4.431.000

[Compute from data](#)

Lat/Lon Bounding Box

Min X	Min Y	Max X	Max Y
-105,2168216144	40,015499345512	-105,1992022820	40,029045461096

[Compute from native bounds](#)

Coverage Parameters

InputTransparentColor

SUGGESTED_TILE_SIZE



Scale = 1 : 10K

482979.49219, 4429517.57813

Click on the map to get feature info

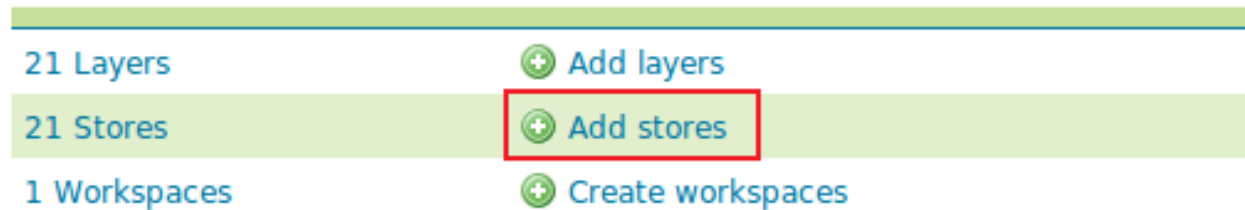
- It is now possible to mix the layers with local ones to generate print oriented formats such as PDF
- It is now possible to provide more informations about the layer, such as a better description, more keywords, which will benefit all clients, in particular catalogues harvesting informations from your capabilities document

Configuration

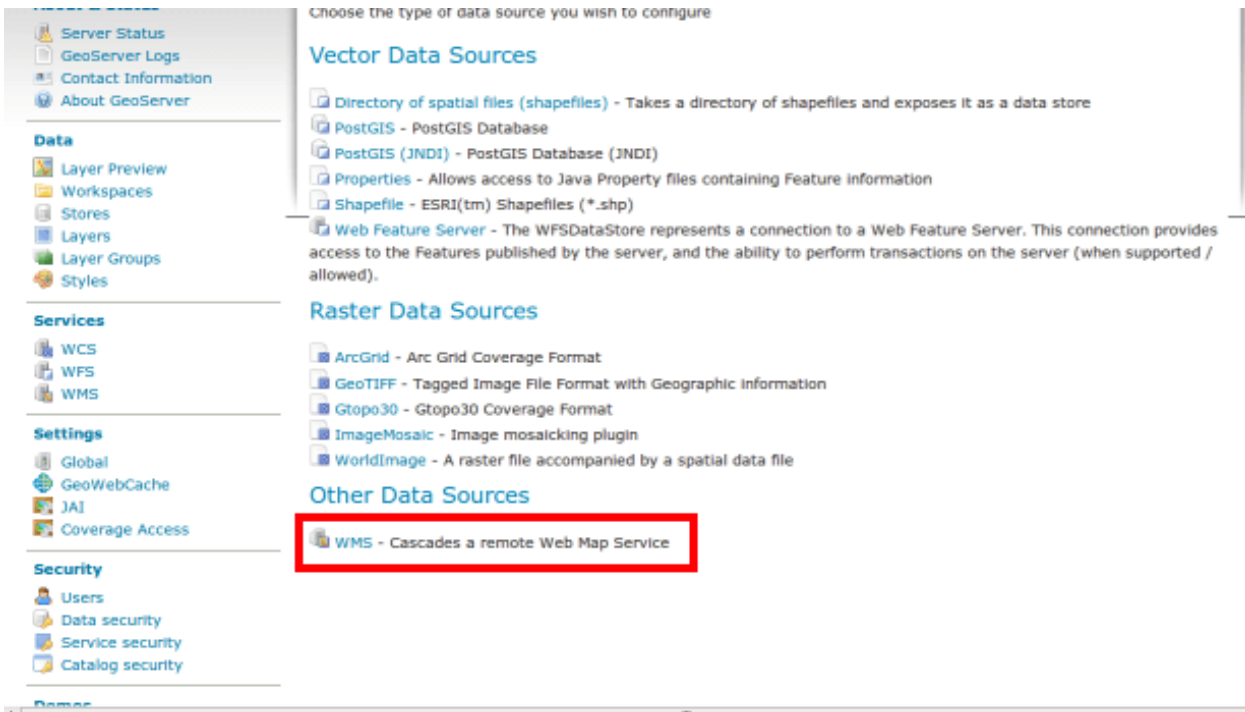
The configuration as usage of the cascaded layers follows GeoServer traditional ease of use.

1. Open the web browser and navigate to the GeoServer [Welcome Page](#).
2. Select *Add stores* from the interface.

This GeoServer belongs to [GeoSolutions](#).



3. Select *WMS - Cascades a remote Web Map Service* from the set of available Other Data Sources.



4. Specify a proper name (as an instance, geoserver-enterprise) in the *Data Source Name* field of the interface.
5. Specify `http://demo1.geo-solutions.it/geoserver-enterprise/ows?service=wms&version=1.1.1&request=GetCapabilities` as the URL of the sample data in the *Capabilities URL* field.
6. Click *Save*.

New WMS Connection

Edit the connection to a remote WMS Connection

Basic Store Info

Workspace *

it.geosolutions ▼

WMS Source Name *

geoserver-enterprise

Description

☒ Enabled

Connection Info

Capabilities URL *

http://demo1.geo-solutions.it/geoserver-enterprise/ows?

User Name

Password

☒ Use HTTP connection pooling

Max concurrent connections *

6

Connect timeout in seconds *

30

Read timeout in seconds *

60

Save

Cancel

- Publish the layer by clicking on the *publish* link near the *geosolutions:ne_shaded* layer name. Notice that you can also add more layers later.

Services	GeoSolutions:Raster_50000_Scale_Index	Publish
WCS	GeoSolutions:SRTM	Publish
WFS	GeoSolutions:adriatico-deparea	Publish
WMS	GeoSolutions:adriatico-depcontour	Publish
Settings	GeoSolutions:baleari-tirreno-deparea	Publish
Global	GeoSolutions:baleari-tirreno-depcontour	Publish
GeoWebCache	GeoSolutions:cities	Publish
JAI	GeoSolutions:coastline	Publish
Coverage Access	GeoSolutions:country	Publish
Security	GeoSolutions:elba-deparea	Publish
Users	GeoSolutions:elba-depcontour	Publish
Data security	GeoSolutions:laspezia-deparea	Publish
Service security	GeoSolutions:laspezia-depcontour	Publish
Catalog security	GeoSolutions:laspezia-marina-deparea	Publish
Demos	GeoSolutions:laspezia-marina-depcontour	Publish
	GeoSolutions:marligure-deparea	Publish
	GeoSolutions:marligure-depcontour	Publish
	GeoSolutions:marligure-tirreno-strettosicilia-contour	Publish
	GeoSolutions:marligure-tirreno-strettosicilia-deparea	Publish
	GeoSolutions:mjwater	Publish
	GeoSolutions:ne_shaded	Publish

<< < 1 2 > >> Results 1 to 25 (out of 47 items)

- Check the Coordinate Reference Systems and the Bounding Boxes fields are properly set and click on *Save*.
- At this point the new WMS Layer is being published with GeoServer. You can use the layer preview to inspect the data.

Adding a WFS Cascade Layer

GeoServer has the ability to load data from a remote Web Feature Server (WFS). This is useful if the remote WFS lacks certain functionality that GeoServer contains. For example, if the remote WFS is not also a Web Map Server (WMS), data from the WFS can be cascaded through GeoServer to utilize GeoServer's WMS. If the remote WFS has a WMS but that WMS cannot output KML, data can be cascaded through GeoServer's WMS to output KML.

Configuration

The configuration as usage of the cascaded layers follows GeoServer traditional ease of use.

- Open the web browser and navigate to the GeoServer [Welcome Page](#).
- Select *Add stores* from the interface.
- Select *Web Feature Server* from the set of available Vector Data Sources.
- Specify a proper name (as an instance, *wfs-cascade*) in the *Data Source Name* field of the interface.
- Specify `http://demo1.geo-solutions.it/geoserver-enterprise/ows?service=wfs&version=1.0.0&request=GetCapabilities` as the URL of the sample data in the *Capabilities URL* field.
- Make sure that the HTTP Authentication fields match the remote server authorization you have on it (In this case the server is open so we don't need them).

Coordinate Reference Systems

Native SRS

[EPSG:WGS 84...](#)

Declared SRS

[EPSG:WGS 84...](#)

SRS handling

Bounding Boxes

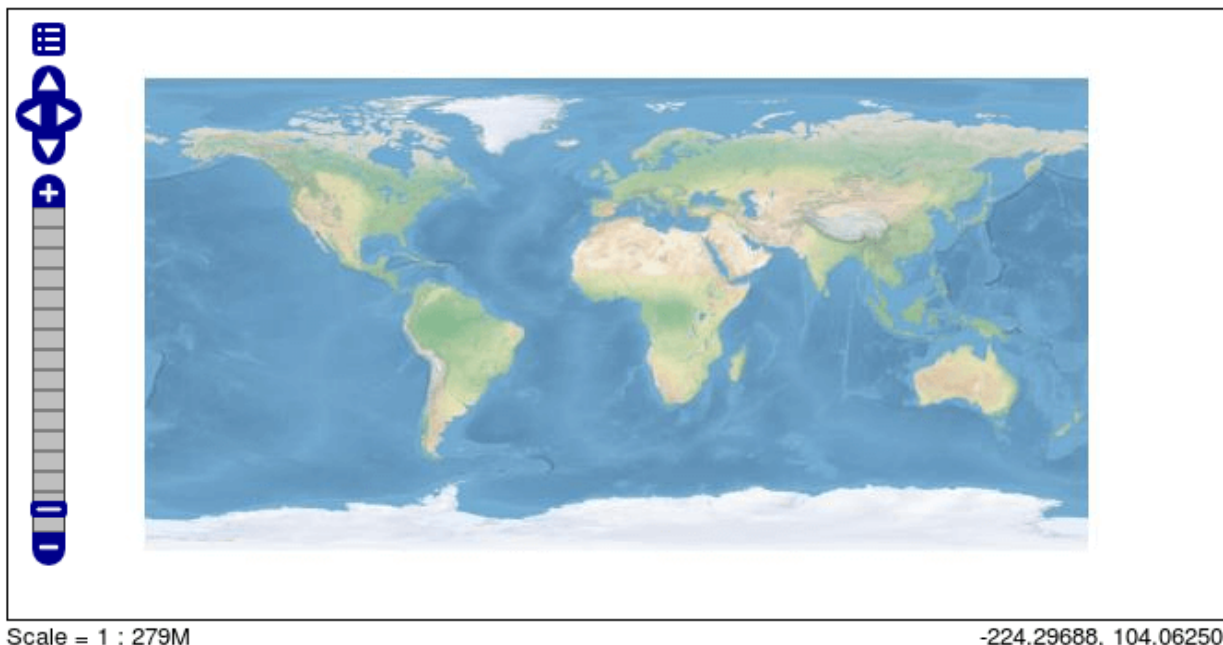
Native Bounding Box

Min X	Min Y	Max X	Max Y
-180	-90	180	90

[Compute from data](#)

Lat/Lon Bounding Box

Min X	Min Y	Max X	Max Y
-180	-90	180	90

[Compute from native bounds](#)


This GeoServer belongs to [GeoSolutions](#).

21 Layers	+ Add layers
21 Stores	+ Add stores
1 Workspaces	+ Create workspaces

New Vector Data Source

Add a new vector data source

Web Feature Server

The WFSDataStore represents a connection to a Web Feature Server. This connection provides access to the and the ability to perform transactions on the server (when supported / allowed).

Basic Store Info

Workspace *

geosolutions ▼

Data Source Name *

wfs-cascade

Description

☒ Enabled

Connection Parameters

WFS GetCapabilities URL *

vs?service=wfs&version=1.0.0&request=GetCapabilities

☐ Favor HTTP POST method over GET

HTTP Authentication user name

HTTP Authentication user password

Character encoding for XML messages

UTF-8 ▼

Connection and read timeout (ms)

3000

Use version 1.0.0

Leave these fields empty

- Click *Save*.
- Publish the layer by clicking on the *publish* link near the *geosolutions_country* layer name. Notice that you can also add more layers later.

New Layer

Add a new layer

Add layer from

You can create a new feature type by manually configuring the attribute names and types. [Create new feature type...](#)
Here is a list of resources contained in the store 'wfs-cascade'. Click on the layer you wish to configure

<< < 1 > >> Results 0 to 0 (out of 0 items) <input type="text" value="Search"/>		
Published	Layer name	Action
	geosolutions_country	Publish
	geosolutions_adriatico-deparea	Publish
	geosolutions_cities	Publish
	geosolutions_coastline	Publish
	geosolutions_elba-deparea	Publish
	geosolutions_elba-depcontour	Publish
	geosolutions_laspezia-deparea	Publish

- Check the Coordinate Reference Systems and the Bounding Boxes fields are properly set and click on *Save*.
- At this point the new WMS Layer is being published with GeoServer. You can use the layer preview to inspect the data.

Adding a SQL Parametric View based Layer

The traditional way to use database backed data is to configure either a table or a database view as a new layer in GeoServer. Starting with GeoServer 2.1.0 the user can also create a new layer by specifying a raw SQL query, without the need to actually creating a view in the database. The SQL can also be parametrized, and parameter values passed in along with a WMS or WFS request.

Creating a plain SQL view

- In order to create an SQL view the administrator can go into the *Add a new resource* from the *Layers* page.
- Upon selection of a database backed store a list of tables and views available for publication will appear, but at the bottom of it a new link, *Configure new SQL view*, will appear:
- Selecting the link *Configure new SQL view* will open a new page where the SQL statement can be specified:

```
SELECT st.obs_year,
        st.storm_num,
        st.storm_name,
        min(st.obs_datetime)
AS storm_start, max(st.obs_datetime)
AS storm_end, max(st.wind)
AS max_wind, st_makeline(st.geom)
AS the_route
FROM ( SELECT storm_obs.storm_num,
```

(continues on next page)

Coordinate Reference Systems

Native SRS

[EPSG:WGS 84...](#)

Declared SRS

[EPSG:WGS 84...](#)

SRS handling



Bounding Boxes

Native Bounding Box

Min X
Min Y
Max X
Max Y

[Compute from data](#)

Lat/Lon Bounding Box

Min X
Min Y
Max X
Max Y

[Compute from native bounds](#)


Scale = 1 : 279M

Click on the map to get feature info

94.21875, 60.07941

GeoServer

Logged in as geosolutions. [Logout](#)

Layers

Manage the layers being published by GeoServer

[Add a new resource](#) [Remove selected resources](#)

<< < 1 > >> Results 1 to 21 (out of 21 items)

<input type="checkbox"/>	Type	Workspace	Store	Layer Name	Enabled?	Native SRS
<input type="checkbox"/>	Point	geosolutions	Civici	Civici	✓	EPSG:3003
<input type="checkbox"/>	LineString	geosolutions	Spart_auto	Spart_auto	✓	EPSG:3003
<input type="checkbox"/>	LineString	geosolutions	Cinta_linea	Cinta_linea	✓	EPSG:3003
<input type="checkbox"/>	LineString	geosolutions	Limite_Comunale	Limite_Comunale	✓	EPSG:3003
<input type="checkbox"/>	Image	geosolutions	storms	vstorm_track	✓	EPSG:4326
<input type="checkbox"/>	Point	geosolutions	storms	storm_obs	✓	EPSG:4326
<input type="checkbox"/>	LineString	geosolutions	dettagli	dettagli	✓	EPSG:3003
<input type="checkbox"/>	LineString	geosolutions	P_Strade	P_Strade	✓	EPSG:3003
<input type="checkbox"/>	Point	geosolutions	Ospedale	Ospedale	✓	EPSG:3003
<input type="checkbox"/>	LineString	geosolutions	Ferrovia	Ferrovia	✓	EPSG:3003

GeoServer

Logged in as admin. [Logout](#)

New Layer

Add a new layer

Add layer from

- Sceglierne uno
- geosolutions:13tde815295_200803_0x6000m_cl
- geosolutions:Mainrd
- geosolutions:NaturalEarthCountries
- geosolutions:boulder_shapefiles
- geosolutions:dem
- geosolutions:nyc
- geosolutions:sf
- geosolutions:sfdem
- geosolutions:shape
- geosolutions:states
- geosolutions:storms**
- geosolutions:usa
- geosolutions:wind
- geosolutions:wind2

New Layer


Add a new layer

Add layer from

You can create a new feature type by manually configuring the attribute names and types. [Create new feature type...](#)

On databases you can also create a new feature type by configuring a native SQL statement [Configure new SQL view...](#)

Here is a list of resources contained in the store 'storms'. Click on the layer you wish to configure

<div> << < 1 > >> </div> <div>Results 0 to 0 (out of 0 items)</div> <div> <input type="text" value="Search"/> </div>		
Published	Layer name	
	storm_obs	Publish again

(continued from previous page)

```

                                storm_obs.storm_name,
                                storm_obs.wind,
                                storm_obs.press,
                                storm_obs.obs_datetime,
                                date_part('year'::text, storm_obs.obs_datetime)
                                AS obs_year, storm_obs.geom
FROM storm_obs
ORDER BY date_part('year'::text, storm_obs.obs_datetime),
                                storm_obs.storm_num,
                                storm_obs.obs_datetime) st
GROUP BY st.obs_year, st.storm_num, st.storm_name
ORDER BY st.obs_year, st.storm_num

```

Note: The query can be any SQL statement that can be validly executed as part of a subquery in the FROM clauses, that is `select * from (<the sql view>) [as] vtable`. This is true for most SQL statements, but specific syntax might be needed to call onto a stored procedure depending on the database. Also, all the columns returned by the SQL statement must have a name, in some databases aliasing is required when calling function names

- Once a valid SQL statement has been specified press the *refresh* link in the Attributes table to get a list of the feature type attributes:

Note: GeoServer will do its best to figure out automatically the geometry type and the native srid, but they should always be double checked and eventually corrected. In particular having the right SRID (spatial reference id) is key to have spatial queries actually work. In many spatial databases the SRID is equal to the EPSG code for the specific spatial reference system, but that is not always true (e.g., Oracle has a number of non EPSG SRID codes).

- Specify a valid SRID.

Note: If stable feature ids are desired for the view's features one or more column providing a unique identification for the features should be checked in the `Identifier` column. Always make sure those attributes generate a actually unique key, or filtering and WFS clients will mishbehave.

Create new SQL view

Define a new SQL view and configure its identified and geometry columns

View Name

SQL statement

```
SELECT st.obs_year,
        st.storm_num,
        st.storm_name,
        min(st.obs_datetime)
        AS storm_start, max(st.obs_datetime)
        AS storm_end, max(st.wind)
        AS max_wind, st_makeline(st.geom)
        AS the_route
FROM ( SELECT storm_obs.storm_num,
              storm_obs.storm_name,|
              storm_obs.wind,
              storm_obs.press,
              storm_obs.obs_datetime,
              date_part('year'::text,
storm_obs.obs_datetime)
              AS obs_year, storm_obs.geom
        FROM storm_obs
        ORDER BY date_part('year'::text, storm_obs.obs_datetime),
                  storm_obs.storm_num,
                  storm_obs.obs_datetime) st
GROUP BY st.obs_year, st.storm_num, st.storm_name
ORDER BY st.obs_year, st.storm_num
```

Fig. 134: Plain SQL View configuration

SQL view parameters[Guess parameters from SQL](#) [Add new parameter](#) [Remove selected](#)

<input type="checkbox"/>	Name	Default value	Validation regular expression
--------------------------	------	---------------	-------------------------------

Attributes**Refresh** ☒ Guess geometry type and srid

Name	Type	SRID	Identifier
obs_year	Double		<input type="checkbox"/>
storm_num	Integer		<input type="checkbox"/>
storm_name	String		<input type="checkbox"/>
storm_start	Timestamp		<input type="checkbox"/>
storm_end	Timestamp		<input type="checkbox"/>
max_wind	BigDecimal		<input type="checkbox"/>
the_route	<input type="text" value="LineString"/>	<input type="text"/>	<input type="checkbox"/>

the_route	<input type="text" value="LineString"/>	<input type="text" value="4326"/>	<input type="checkbox"/>
<input type="button" value="Save"/> <input type="button" value="Cancel"/>			

Fig. 135: Forcing manually 4326 SRID in this case

- Once the query and the attribute details are set press *Save* and the usual new layer configuration page will show up. That page will have a link to a SQL view editor at the bottom of the *Data* tab:

Feature Type Details

Property	Type	Nullable	Min/Max Occurrences
obs_year	Double	true	0/1
storm_num	Integer	false	1/1
storm_name	String	false	1/1
storm_start	Timestamp	true	0/1
storm_end	Timestamp	true	0/1
max_wind	BigDecimal	true	0/1
the_route	LineString	true	0/1

Edit sql view

- Make sure the CRS is EPSG:4326 and **write manually** $(-180, -90, 180, 90)$ values in the *Bounding Boxes* sections.
- Click *Save*.

At this point the new WMS Layer is being published with GeoServer.

Creating a parametric SQL view

Warning: As a rule of thumb use SQL parameter substitution only if the required functionality cannot be obtained with safer means, such as dynamic filtering (CQL filters) or SLD parameter substitution. Only use SQL parameters as a last resort, improperly validated parameters can open the door to [SQL injection attacks](#).

A parametric SQL view is based on a SQL query containing parameters whose values can be dynamically provided along WMS or WFS requests. A parameter is bound by % signs, can have a default value, and should always have a validation regular expression.

- In order to create a parametric SQL view performs the steps 1 and 2 like before and then insert the following parameters:

```
SELECT date_part('year'::text, t1.obs_datetime) AS obs_year, t1.storm_num, t1.
↳storm_name, t1.wind, t2.wind AS wind_end, t1.press, t2.press AS press_end, t1.
↳obs_datetime, t2.obs_datetime AS obs_datetime_end, st_makeline(t1.geom, t2.
↳geom) AS geom
FROM storm_obs t1
JOIN ( SELECT storm_obs.id, storm_obs.storm_num, storm_obs.storm_name, storm_obs.
↳wind, storm_obs.press, storm_obs.obs_datetime, storm_obs.geom
```

(continues on next page)

Coordinate Reference Systems

Native SRS

[EPSG:WGS 84...](#)

Declared SRS

[EPSG:WGS 84...](#)

SRS handling

Bounding Boxes

Native Bounding Box

Min X	Min Y	Max X	Max Y
-180	-90	180	90

[Compute from data](#)

Lat/Lon Bounding Box

Min X	Min Y	Max X	Max Y
-180	-90	180	90

[Compute from native bounds](#)

Edit SQL view

Update the definition of the SQL view and its metadata

View Name

SQL statement

```
SELECT date_part('year'::text,
t1.obs_datetime) AS obs_year, t1.storm_num,
t1.storm_name, t1.wind, t2.wind AS wind_end,
t1.press, t2.press AS press_end,
t1.obs_datetime, t2.obs_datetime AS
obs_datetime_end, st_makeline(t1.geom,
t2.geom) AS geom
FROM storm_obs t1
JOIN ( SELECT storm_obs.id,
storm_obs.storm_num, storm_obs.storm_name,
storm_obs.wind, storm_obs.press,
storm_obs.obs_datetime, storm_obs.geom
      FROM storm_obs) t2 ON
(t1.obs_datetime + '06:00:00'::interval) =
t2.obs_datetime AND t1.storm_name::text =
t2.storm_name::text
WHERE
```

Fig. 136: Parametric SQL View configuration

(continued from previous page)

```

FROM storm_obs) t2 ON (t1.obs_datetime + '06:00:00'::interval) = t2.
↪ obs_datetime AND t1.storm_name::text = t2.storm_name::text
WHERE
    date_part('year'::text, t1.obs_datetime) BETWEEN %MIN_OBS_YEAR% AND %MAX_
↪ OBS_YEAR%
ORDER BY date_part('year'::text, t1.obs_datetime), t1.storm_num, t1.obs_datetime

```

Note: The query defines two parameters %MIN_OBS_YEAR% and %MAX_OBS_YEAR%.

2. Click on the *Guess parameters from SQL*. GeoServer will automatically create fields with the parameters specified in the view:

SQL view parameters

[Guess parameters from SQL](#)
[Add new parameter](#)
[Remove selected](#)

<input type="checkbox"/>	Name	Default value	Validation regular expression
<input type="checkbox"/>	MAX_OBS_YEAR	2020	^[\\w\\d\\s]+\$
<input type="checkbox"/>	MIN_OBS_YEAR	0	^[\\w\\d\\s]+\$

Note: Always provide default values for each parameter in order to let the layer work properly and also be sure the regular expression for the values validation are correct.

Examples of Regular Expressions:

- `^[\\d\\.\\+\\-eE]+$` will check that the parameter value is composed with valid elements for a floating point number, eventually in scientific notation, but will not check that the provided value is actually a valid floating point number
- `[^;']+` will check the parameter value does not contain quotes or semicolon, preventing common sql injection attacks, without actually imposing much on the parameter value structure

3. Fill in some default values for the parameters, so that GeoServer can run the query and inspect the results in the next steps. Set MAX_OBS_YEAR to 2020 and MIN_OBS_YEAR to 0.
4. *Refresh* the attributes, check the Geometry SRID and publish the layer like before. Also assign the `storm_track_interval` style to the layer as Default Style.
5. Click on the *OpenLayers* on the *Layer Preview* list for `v_storm_track_interval` layer.
6. At a first glance you won't see anything since the layer is using the default parameters for the observation years. Specify two years for the view adding this parameter at the end of the GetMap Request:


```
&viewparams=MIN_OBS_YEAR:2000;MAX_OBS_YEAR:2000
```

You should obtain a request like this:


```


http://localhost:8083/geoserver/geosolutions/wms?service=WMS&version=1.1.0&
↪ request=GetMap&layers=geosolutions:v_storm_track_interval&styles=&bbox=-
↪ 180.0,-90.0,180.0,90.0&width=660&height=330&srs=EPSG:4326&
↪ format=application/openlayers&viewparams=MIN_OBS_YEAR:2000;MAX_OBS_
↪ YEAR:2000


```


 WMS

Settings


 Global


 GeoWebCache


 JAI


 Coverage Access

Security

 Users

 Data security

 Service security

 Catalog security

Demos

Cache Time (seconds)

Edit Layer

Per-Request Feature Limit

0

Maximum number of decimals


0


WMS Settings


☒ Queryable


Default Style


storm_track_interval


 Depression (<34 knots)


 Tropical Storm (34-63 knots)

 Hurricane Cat 1 (64-82 knots)

 Hurricane Cat 2 (83-95 knots)

 Hurricane Cat 3 (96-112 knots)

 Hurricane Cat 4 (113-135 knots)

 Hurricane Cat 5 (>135 knots)

Additional Styles

Available Styles

burg
capitals
Cinta Murria

3.2. Tutorials

623

Now you are able to see the hurricanes from the parametric view and also dynamically choose the observation years interval of interest.

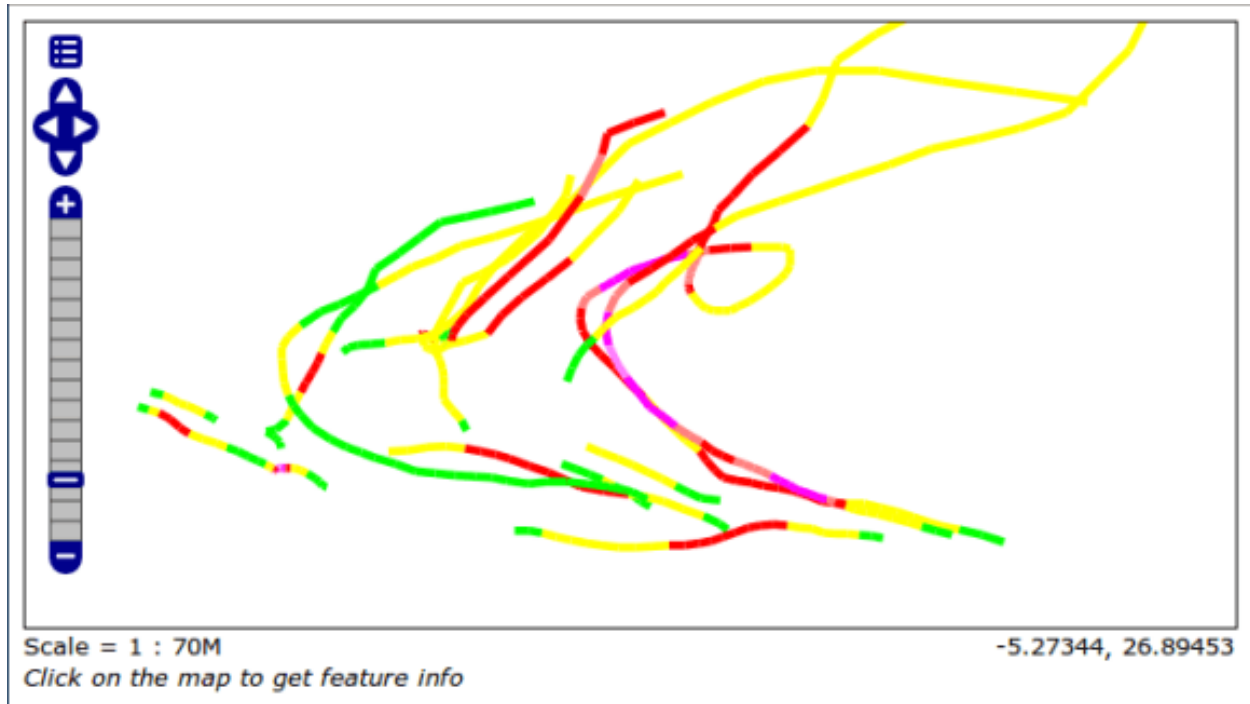


Fig. 137: Parametric SQL View OL preview

Adding an Image Mosaic to GeoServer

This section covers the task of adding and publishing a ImageMosaic file with GeoServer.

1. Navigate to the GeoServer [Welcome Page](#).
2. On the Welcome page locate the *Login* form located at the top right corner, and enter the username “admin” and password “Geos”.
3. Click the *Add stores* link.
4. Select the *ImageMosaic* link and click it.
5. On the *Add Raster Data Source* page enter `$TRAINING_ROOT/data/user_data/aerial` (on Windows `%TRAINING_ROOT%\data\user_data\airial\`) in the *URL* field (or browse the filesystem clicking on *Browse*), “boulder_bg” in the *Data Source Name* and *Description* fields, and click *Save*.
6. After saving, you will be taken to a page that lists all the layers in the store and gives you the option to publish any of them. Click *Publish*.
7. The *Coordinate Reference Systems* should be automatically populated, as well as the *Name*, *Title* and *Bounding Boxes* fields.

Note: Change the *Name* and *Title* into **boulder_bg** as shown in the figure.

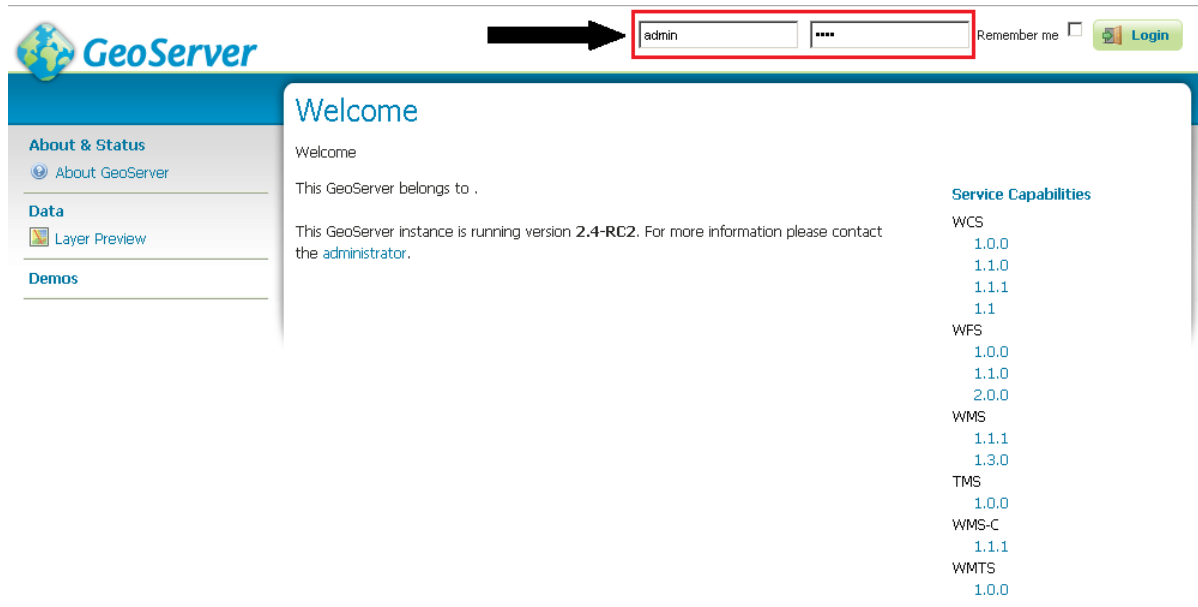


Fig. 138: GeoServer Login

Welcome

Welcome

This GeoServer belongs to .



Fig. 139: Add stores link

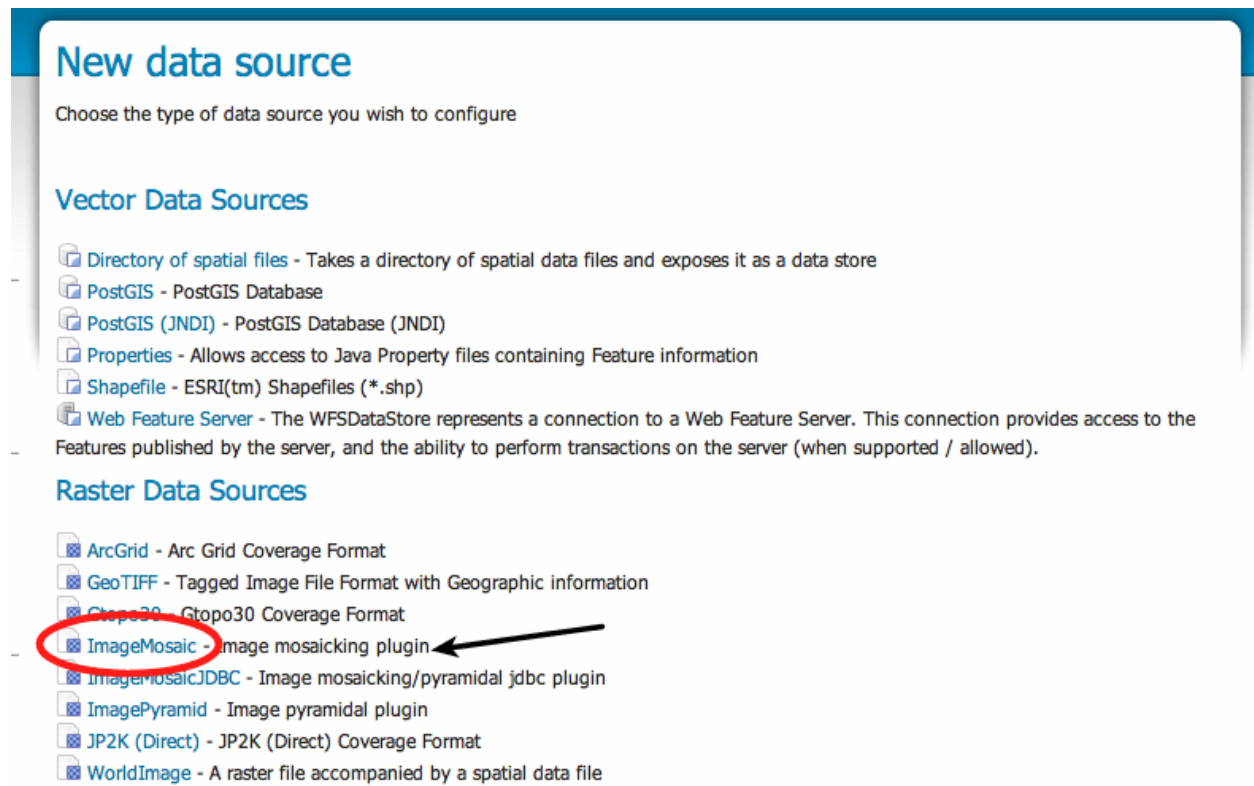


Fig. 140: Add a new Image Mosaic

The CRS and BBox fields are auto-filled with information taken from the underlying files. The coverage options section is filled with default parameters (which will be discussed later on in the training).

8. Scroll to the bottom of the page and then click *Save*. If all went well you should see something like this:
9. In the *Layer Preview* section click on the OpenLayers link to preview the layer in an interactive viewer, filtering by *boulder_bg* name:

Adding a GDAL Supported Format

In case the GDAL libraries are available, it is possible to access to several GDAL's supported data formats. Actually, the available GDAL plugins allow to support DTED, EHdr, ERDASImg, MrSID, JPEG 2000 (via MrSID Driver) and NITF data formats. Moreover, in case a valid license have been purchased and the proper native library is available, also ECW, JPEG 2000 (via ECW) and JPEG 2000 (via Kakadu) are supported. This section provides instructions to add and publish MrSID, ECW and JPEG 2000 datasets.

Warning: This assumes the GeoServer image GDAL plug-in is already installed. The GDAL plugin is normally an extension.

If the store described in this section are not available, install the 'geoserver-2.2-SNAPSHOT-gdal-plugin' from %TRAINING_ROOT%\data\plugins\. Just decompress the zip file into %TRAINING_ROOT%\webapps\geoserver\WEB-INF\lib\ and restart GeoServer.

Add Raster Data Source

Description

ImageMosaic

Image mosaicking plugin

Basic Store Info

Workspace *

geosolutions ▼

Data Source Name *

boulder_bg

Description

boulder_bg

☒ Enabled

Connection Parameters

URL *

file:data\user_data\airial\

[Browse...](#)

[Save](#)

[Cancel](#)

Fig. 141: Specifying store parameters

New Layer

Add a new layer

Here is a list of resources contained in the store 'boulder_bg'. Click on the layer you wish to configure

<<

<

1

>

>>

Results 1 to 1 (out of 1 items)

Search

Published	Layer name	action
	aerial	Publish

<<

<

1

>

>>

Results 1 to 1 (out of 1 items)

Fig. 142: Publishing a layer from the store

Data

Publishing

Dimensions

Tile Caching

Basic Resource Info

Name

☒ Enabled☒ Advertised

Title

Abstract

Keywords

Current Keywords

WCS

ImageMosaic

aerial

New Keyword

Vocabulary

Metadata links

No metadata links so far

Note only FGDC and TC211 metadata links show up in WMS 1.1.1 capabilities

Coordinate Reference Systems

Native SRS

[EPSG:NAD83 / UTM zone 13N...](#)

Declared SRS

[EPSG:NAD83 / UTM zone 13N...](#)

6285 handling

Coverage Parameters

Accurate resolution computation

AllowMultithreading

BackgroundValues

Filter

FootprintBehavior

InputTransparentColor

MaxAllowedTiles

MergeBehavior

OutputTransparentColor

SORTING

SUGGESTED_TILE_SIZE

USE_JAI_IMAGEREAD

Fig. 143: The coverage layer gui for the boulder_bg layer.

Coverage Band Details

Band	Data type	Null Values	minRange	maxRange	Unit
<input type="text" value="RED_BAND"/>	Unsigned 8 bits	<input type="text" value="0.0"/>	<input type="text" value="0"/>	<input type="text" value="0"/>	<input type="text" value="W.m-2.Sr-1"/>
<input type="text" value="GREEN_BAND"/>	Unsigned 8 bits	<input type="text" value="0.0"/>	<input type="text" value="0"/>	<input type="text" value="0"/>	<input type="text" value="W.m-2.Sr-1"/>
<input type="text" value="BLUE_BAND"/>	Unsigned 8 bits	<input type="text" value="0.0"/>	<input type="text" value="0"/>	<input type="text" value="0"/>	<input type="text" value="W.m-2.Sr-1"/>

Fig. 144: The coverage bands details



















Layers

Manage the layers being published by GeoServer

[Add a new resource](#)

[Remove selected resources](#)

<< < 1 > >> Results 1 to 18 (out of 18 items)

<input type="checkbox"/>	Type	Workspace	Store	Layer Name	Enabled?	Native SRS
<input type="checkbox"/>		geosolutions	boulder_bg	boulder_bg	✓	EPSG:26913
<input type="checkbox"/>		geosolutions	boulder_shapefiles	bbuildings	✓	EPSG:2876
<input type="checkbox"/>		geosolutions	boulder_shapefiles	blakes	✓	EPSG:4269
<input type="checkbox"/>		geosolutions	boulder_shapefiles	BoulderCityLimits	✓	EPSG:2876
<input type="checkbox"/>		geosolutions	boulder_shapefiles	bplandmarks	✓	EPSG:4269
<input type="checkbox"/>		geosolutions	boulder_shapefiles	bptlandmarks	✓	EPSG:4269
<input type="checkbox"/>		geosolutions	boulder_shapefiles	bptlandmarks_2876	✓	EPSG:2876
<input type="checkbox"/>		geosolutions	boulder_shapefiles	brivers	✓	EPSG:4269
<input type="checkbox"/>		geosolutions	boulder_shapefiles	bstreets	✓	EPSG:2876
<input type="checkbox"/>		geosolutions	boulder_shapefiles	ccounties	✓	EPSG:4269
<input type="checkbox"/>		geosolutions	boulder_shapefiles	Mainrd	✓	EPSG:2876
<input type="checkbox"/>		geosolutions	boulder_shapefiles	Parcels	✓	EPSG:2876
<input type="checkbox"/>		geosolutions	boulder_shapefiles	Trails	✓	EPSG:2876
<input type="checkbox"/>		geosolutions	boulder_shapefiles	Wetlands_regulatory_area	✓	EPSG:2876
<input type="checkbox"/>		geosolutions	dem	srtm	✓	EPSG:4326
<input type="checkbox"/>		geosolutions	hillshade	hillshade	✓	EPSG:4326
<input type="checkbox"/>		geosolutions	NaturalEarthCountries	WorldCountries	✓	EPSG:4326
<input type="checkbox"/>		geosolutions	states	states	✓	EPSG:4326

<< < 1 > >> Results 1 to 18 (out of 18 items)

Fig. 145: After a successful save.

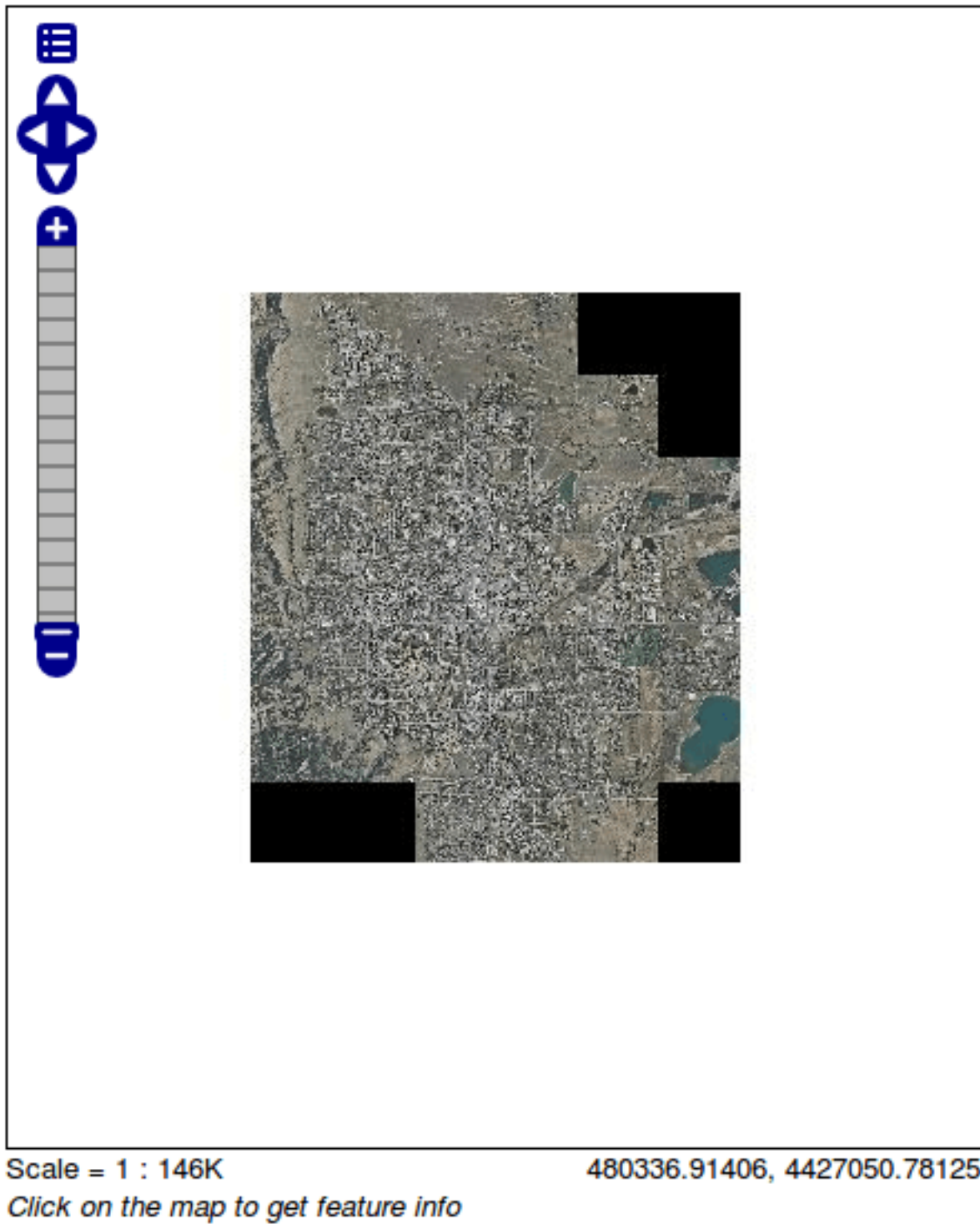
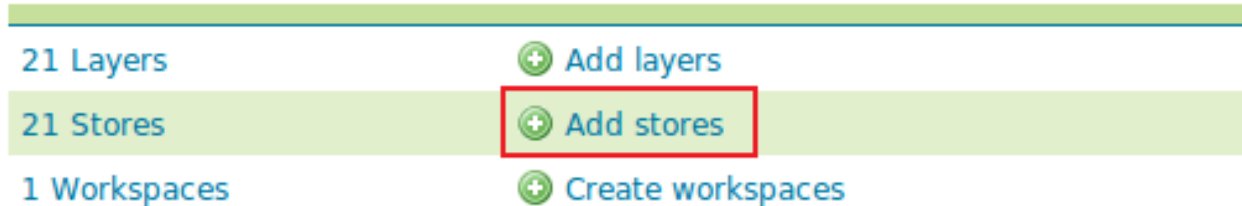


Fig. 146: Mosaic preview.

MrSID Data Set

1. Open the web browser and navigate to the GeoServer [Welcome Page](#).
2. Select *Add stores* from the interface.

This GeoServer belongs to [GeoSolutions](#).



3. Select *MrSID - MrSID Coverage Format* from the set of available Raster Data Sources.

Raster Data Sources



4. Specify a proper name (as an instance, `c3008957_nes_20`) in the *Data Source Name* field of the interface.
5. Specify `file:%TRAINING_ROOT%/data/user_data/c3008957_nes_20/c3008957_nes_20.sid` as URL of the sample data in the *Connections Parameter's - URL* field. (replace `%TRAINING_ROOT%` with your current training root directory)
6. Click *Save*.
7. Assign a proper layername (e.g `c3008957_nes_20`) then publish the layer by clicking on the *publish* link.

Add Raster Data Source

Description

MrSID

MrSID Coverage Format

Basic Store Info

Workspace *

geosolutions ▼

Data Source Name *

c3008957_nes_20

Description

☒ Enabled

Connection Parameters

URL *

data\user_data\c3008957_nes_20\c3008957_nes_20.

Save

Cancel

Edit Layer

Edit layer data and publishing

geosolutions:c3008957_nes_20

Configure the resource and publishing information for the current layer

Data

Publishing

Dimensions

Tile Caching

Basic Resource Info

Name

Title

Abstract

Keywords

Current Keywords

WCS
MrSID
c3008957_nes_20

Remove selected

New Keyword

Vocabulary

Add Keyword

Metadata links

No metadata links so far

Add link

Note only FGDC and TC211 metadata links show up in WMS 1.1.1 capabilities

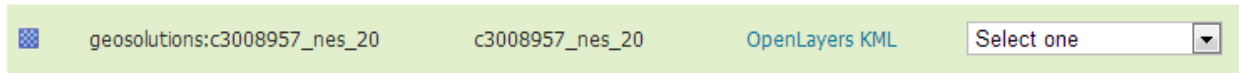
Coordinate Reference Systems

Native SRS

8. Click on *Save* when done.

At this point the MrSID data is being published with GeoServer.

1. Click the *Layer Preview* link in the left GeoServer menu.
2. Look for a *geosolutions:c3008957_nes_20* layer and click the *OpenLayers* link beside of it.



ECW Data Set

Warning: Attention, you need a license in order to use ECW data sets. Here we are using a free distributed ECW file only for demonstration.

ECW (Enhanced Compression Wavelet) is a proprietary wavelet compression image format optimized for aerial and satellite imagery.

1. Open the web browser and navigate to the GeoServer [Welcome Page](#).
2. Select *Add stores* from the interface.
3. Select *ECW - ECW Coverage Format* from the set of available Raster Data Sources.
4. Specify a proper name (as an instance, *TerraColor_Sydney_AU_15m*) in the *Data Source Name* field of the interface.
5. Specify `file:%TRAINING_ROOT%/data/user_data/tc_sydney_au_ecw/TerraColor_Sydney_AU_15m.ecw` as URL of the sample data in the *Connections Parameter's - URL* field (replace `%TRAINING_ROOT%` with your current training root directory).
6. Click *Save*.
7. Assign a proper layername (e.g *TerraColor_Sydney_AU_15m*) then publish the layer by clicking on the *publish* link.

At this point the ECW data is being published with GeoServer.

1. Click the *Layer Preview* link in the left GeoServer menu.
2. Look for a *geosolutions:TerraColor_Sydney_AU_15m* layer and click the *OpenLayers* link beside of it.

JP2K Data Set

JPEG 2000 is a image coding system that uses state-of-the-art compression techniques based on wavelet technology.

1. Open the web browser and navigate to the GeoServer [Welcome Page](#).
2. Select *Add stores* from the interface.
3. Select *JP2ECW - JP2 (ECW) Coverage Format* from the set of available Raster Data Sources.

Note: We used *JP2ECW - JP2 (ECW) Coverage Format* because *JP2MrSID - JP2 (MrSID) Coverage Format* is not fully stable, and may not work properly especially with several Linux distributions.



This GeoServer belongs to [GeoSolutions](#).

21 Layers	+ Add layers
21 Stores	+ Add stores
1 Workspaces	+ Create workspaces

Raster Data Sources

- [ArcGrid](#) - Arc Grid Coverage Format
- [DTED](#) - DTED Coverage Format
- [ECW](#) - ECW Coverage Format
- [EHdr](#) - EHdr Coverage Format
- [ERDASimg](#) - Erdas Imagine Coverage Format
- [GeoTIFF](#) - Tagged Image File Format with Geographic information
- [Gtopo30](#) - Gtopo30 Coverage Format
- [ImageMosaic](#) - Image mosaicking plugin
- [ImagePyramid](#) - Image pyramidal plugin
- [JP2ECW](#) - JP2K (ECW) Coverage Format
- [JP2K \(Direct\)](#) - JP2K (Direct) Coverage Format
- [JP2MrSID](#) - JP2K (MrSID) Coverage Format
- [MrSID](#) - MrSID Coverage Format
- [NITF](#) - NITF Coverage Format

Edit Raster Data Source

Description

ECW

ECW Coverage Format

Basic Store Info

Workspace *

geosolutions ▼

Data Source Name *

Terracolor_Sydney_AU_15m

Description

☒ Enabled

Connection Parameters

URL *

data/user_data/tc_sydney_au_ecw/Terracolor_Sydney_

Save

Cancel

Edit Layer

Edit layer data and publishing

geosolutions:Terracolor_Sydney_AU_15m

Configure the resource and publishing information for the current layer

Data

Publishing

Dimensions

Tile Caching

Basic Resource Info

Name

Terracolor_Sydney_AU_15m

Title

Terracolor_Sydney_AU_15m

Abstract

Keywords

Current Keywords

WCS
ECW
Terracolor_Sydney_AU_15m

Remove selected

New Keyword








Vocabulary

Add Keyword

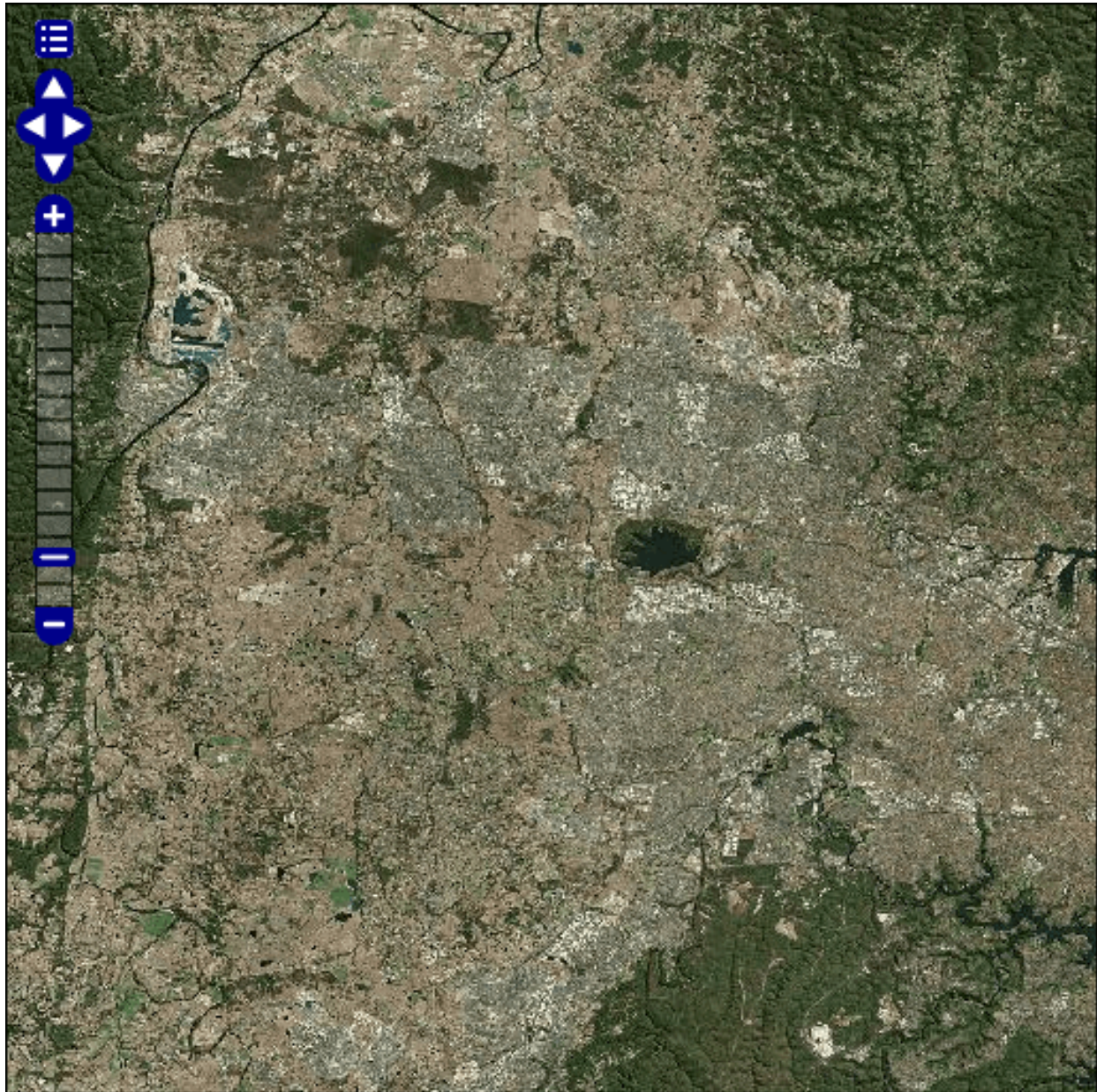
Metadata links

No metadata links so far

<< < 1 > >> Results 1 to 18 (out of 18 items)

Type	Name	Title	Common Formats	All Formats
	geosolutions:bbuildings	Boulder buildings	OpenLayers KML GML	Select one
	geosolutions:blakes	Lakes and other polygonal water entities	OpenLayers KML GML	Select one
	geosolutions:BoulderCityLimits	BoulderCityLimits	OpenLayers KML GML	Select one
	geosolutions:bp.landmarks	Boulder polygonal landmarks	OpenLayers KML GML	Select one
	geosolutions:bpt.landmarks	Point landmarks	OpenLayers KML GML	Select one
	geosolutions:bpt.landmarks_2876	pointlm	OpenLayers KML GML	Select one
	geosolutions:brivers	Rivers and other linear water entities	OpenLayers KML GML	Select one
	geosolutions:bstreets	Boulder streets	OpenLayers KML GML	Select one
	geosolutions:Parcels	Parcels	OpenLayers KML GML	Select one
	geosolutions:Trails	Trails	OpenLayers KML GML	Select one
	geosolutions:Wetlands_regulatory_area	Wetlands_regulatory_area	OpenLayers KML GML	Select one
	geosolutions:srtm	srtm	OpenLayers KML	Select one
	geosolutions:WorldCountries	countries	OpenLayers KML GML	Select one
	geosolutions:states	states	OpenLayers KML GML	Select one
	geosolutions:wind	wind	OpenLayers KML	Select one
	geosolutions:wind2	wind2	OpenLayers KML	Select one
	geosolutions:Terracolor_Sydney_AU_15m	Terracolor_Sydney_AU_15m	OpenLayers KML	Select one
	boulder		OpenLayers KML	Select one

<< < 1 > >> Results 1 to 18 (out of 18 items)



Scale = 1 : 388K

150.95305, -33.79501

Click on the map to get feature info

This GeoServer belongs to [GeoSolutions](#).

21 Layers


[+ Add layers](#)


21 Stores

[+ Add stores](#)


1 Workspaces

[+ Create workspaces](#)

 [PostGIS \(JNDI\)](#) - PostGIS Database (JNDI)

 [Properties](#) - Allows access to Java Property files containing Feature information


 [Shapefile](#) - ESRI(tm) Shapefiles (*.shp)

 [Web Feature Server](#) - The WFSDataStore represents a connection to a Web Feature Server. This connects to the server, and the ability to perform transactions on the server (when supported / allowed).

Raster Data Sources

 [AIG](#) - Arc/Info Binary Grid (AIG) Coverage Format

 [ArcGrid](#) - Arc Grid Coverage Format

 [DTED](#) - DTED Coverage Format

 [ECW](#) - ECW Coverage Format


 [EHdr](#) - EHdr Coverage Format


 [ENVIHdr](#) - ENVIHdr Coverage Format


 [ERDASImg](#) - Erdas Imagine Coverage Format


 [GeoTIFF](#) - Tagged Image File Format with Geographic information


 [Gtopo30](#) - Gtopo30 Coverage Format


 [ImageMosaic](#) - Image mosaicking plugin

 [ImagePyramid](#) - Image pyramidal plugin


 [JP2ECW](#) - JP2K (ECW) Coverage Format

 [JP2K \(Direct\)](#) - JP2K (Direct) Coverage Format


 [JP2MrSID](#) - JP2K (MrSID) Coverage Format

 [MrSID](#) - MrSID Coverage Format

 [NITF](#) - NITF Coverage Format

 [WorldImage](#) - A raster file accompanied by a spatial data file

Other Data Sources

 [WMS](#) - Cascades a remote Web Map Service

4. Specify a proper name (as an instance, TerraColor_Sydney_AU_15m_JP2K) in the *Data Source Name* field of the interface.
5. Specify `file:%TRAINING_ROOT%/data/user_data/tc_sydney_au_jp2/TerraColor_Sydney_AU_15m.jp2` as URL of the sample data in the *Connections Parameter's - URL* field. (replace `%TRAINING_ROOT%` with your current training root directory)

Add Raster Data Source

Description

JP2ECW
JP2K (ECW) Coverage Format

Basic Store Info

Workspace *

geosolutions ▼

Data Source Name *

TerraColor_Sydney_AU_15m_JP2K

Description

TerraColor_Sydney_AU_15m_JP2K

☒ Enabled

Connection Parameters

URL *

file:data/user_data/tc_sydney_au_jp2/TerraColor_Sydn [Browse...](#)

[Save](#) [Cancel](#)

6. Click *Save*.
7. Assign a proper layername (e.g TerraColor_Sydney_AU_15m_JP2K) then publish the layer by clicking on the *publish* link.

At this point the JPEG 2000 data is being published with GeoServer.

1. Click the *Layer Preview* link in the left GeoServer menu.

Edit Layer

Edit layer data and publishing

geosolutions:TerraColor_Sydney_AU_15m

Configure the resource and publishing information for the current layer

Data**Publishing****Dimensions****Tile Caching**

Basic Resource Info

Name☒ Enabled☒ Advertised**Title****Abstract**

Keywords

Current Keywords

WCS

JP2ECW

TerraColor_Sydney_AU_15m

Remove selected

New Keyword**Vocabulary**

Add Keyword

Metadata links

No metadata links so far

Add link

Note only FGDC and TC211 metadata links show up in WMS 1.1.1 capabilities

Coordinate Reference Systems

Native SRS

EPSG:4326

[EPSG:WGS 84...](#)

Declared SRS

EPSG:4326

Find...

[EPSG:WGS 84...](#)

SRS handling

Reproject native to declared ▼

Bounding Boxes

Native Bounding Box

Min X	Min Y	Max X	Max Y
150,38180451736	-34,35941880639	151,38171971736	-33,35950360639

[Compute from data](#)

Lat/Lon Bounding Box

Min X	Min Y	Max X	Max Y
150,38180451736	-34,35941880639	151,38171971736	-33,35950360639

[Compute from native bounds](#)

Coverage Parameters

SUGGESTED_TILE_SIZE

512,512

USE_JAI_IMAGEREAD

true

USE_MULTITHREADING

false

2. Look for a *geosolutions:TerraColor_Sydney_AU_15m_JP2K* layer and click the *OpenLayers* link beside of it.

Pretty maps with GeoServer

This module describes how to manage the GeoServer maps visualization. Will be discussed all those aspects which relate styles, decorations, Layer Groups and other interesting GeoServer features affecting the WMS protocol.

In this module you will:

Styling with SLD

This section introduces the concepts of the Styled Layer Descriptor (SLD) markup language. SLD is the styling engine used by GeoServer, and how all WMS portrayal is specified.

What you will learn

In this section you will:

Adding a Style

The most important function of a web map server is the ability to style and render data. This section covers the task of adding a new style to GeoServer and configuring the default style for a particular layer.

1. From the GeoServer [Welcome Page](#) navigate to *Style*.

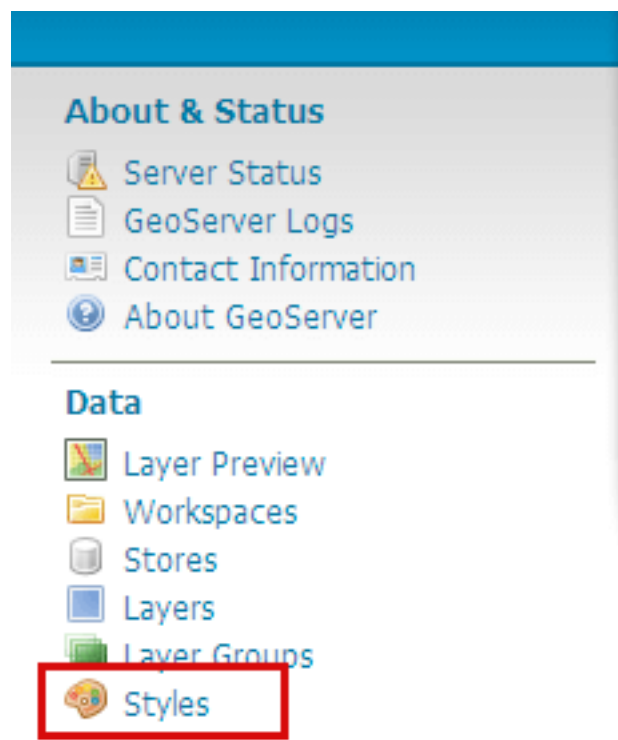


Fig. 147: Navigating to Style configuration

2. Click *New*



Fig. 148: Adding a new style

3. Enter “mainrd” in the *Name* field and notice the file upload dialogue *SLD file*.
4. Navigate to the workshop (on Linux) `$TRAINING_ROOT/data/user_data/` directory (on Windows `%TRAINING_ROOT%\data\user_data\`), select the `foss4g_mainrd.sld` file, and click *Upload*.

Note: In GeoServer, styles are represented via SLD (Styled Layer Descriptor) documents. SLD is an XML format for specifying the symbolization of a layer. When an SLD document is uploaded the contents are shown in the text editor. The editor can be used to edit the contents of the SLD directly.

5. Add the new style by clicking *Submit*. Once it's save, you should see something like this:
6. After having created the style, it's time to apply it to a vector layer. Click on the *Layers* link.
7. Select the “Mainrd” on the *Layers* page.
8. Select the *Publish* tab.
9. Assign the new created style “mainrd” as the default style.

Warning: Many new users mistake the *Available Styles* for the *Default Style*, please take into account that they are different, the default one allows that style to be used implicitly when no style is specified in a map request, while the available ones are just optional compatible styles.

Note: Geoserver 2.x assigns a default style depending on the geometry of the objects and the type, for example: *line, poly, raster, point*.

10. Scroll to the bottom of the page and hit *Save*.
11. Use the map preview to show how the style, please note you'll have to zoom in once to show the data due to the map scale filters (`MaxScaleDenominator` directive in the SLD).

Styling Vector data

In previous modules the style for a layer was configured by uploading an existing SLD. In this section the task of creating a new SLD document from scratch will be covered. In particular we are going to create some styles that can be applied to vectorial datasets, in the first case by drawing patterns and dash arrays to polygons and lines and in the second case drawing roads and labels to lines.

New style

Type a new SLD definition, or use an existing one as a template, or upload a ready made style from your file system. The editor can provide syntax highlight and be brought to full screen. Click on the "validate" button to verify the style is a valid SLD document.

Name

Workspace

Copy from existing style
 [Copy ...](#)

12pt

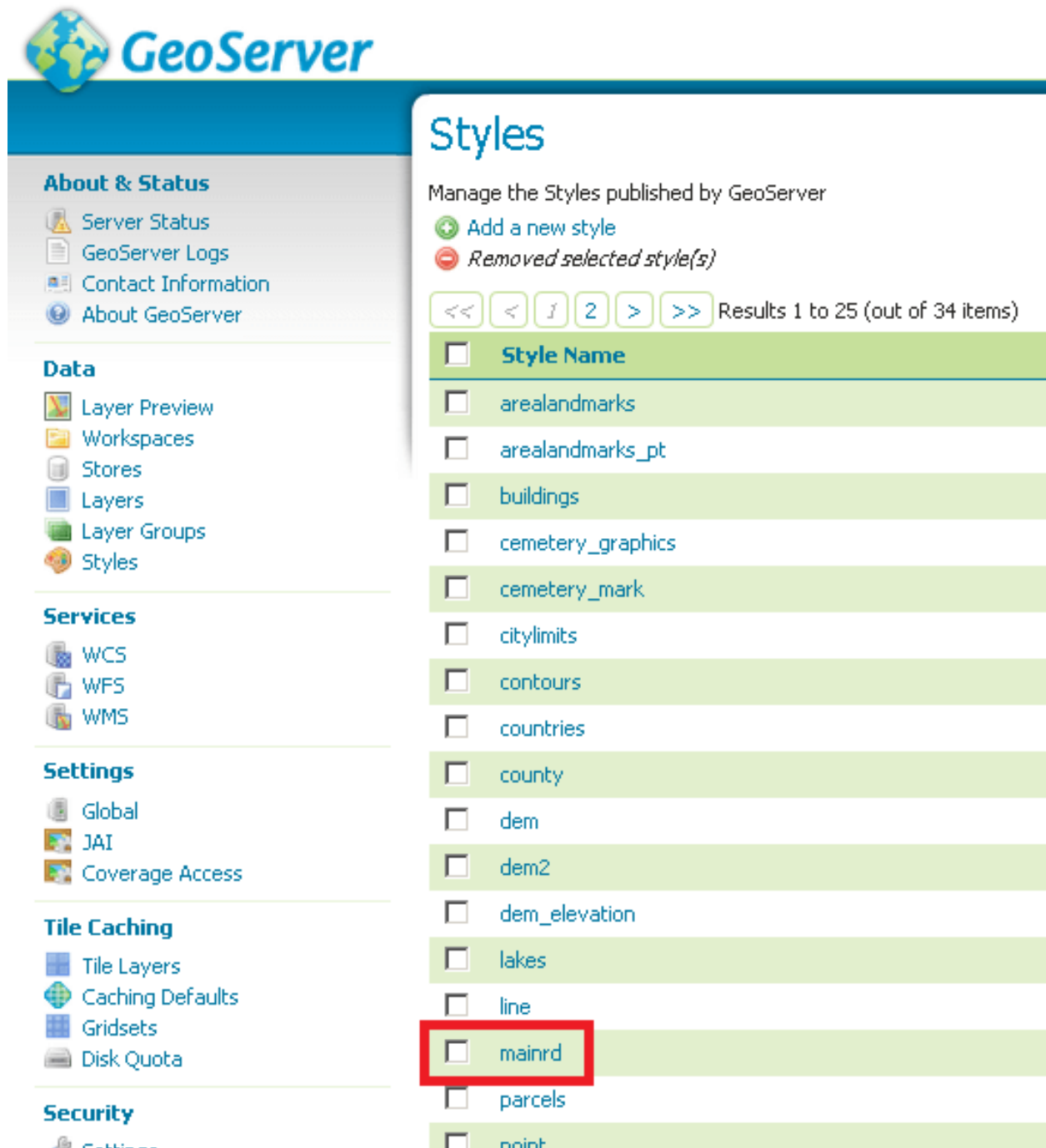
```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <sld:StyledLayerDescriptor xmlns="http://www.opengis.net/sld" xmlns:sld="http://www.opengis.net/sld"
3   <sld:UserLayer>
4     <sld:LayerFeatureConstraints>
5       <sld:FeatureTypeConstraint/>
6     </sld:LayerFeatureConstraints>
7     <sld:UserStyle>
8       <sld:Name>Mainrd</sld:Name>
9       <sld:Title/>
10      <sld:FeatureTypeStyle>
11        <sld:Name>group 0</sld:Name>
12        <sld:FeatureTypeName>Feature</sld:FeatureTypeName>
13        <sld:SemanticTypeIdentifier>generic:geometry</sld:SemanticTypeIdentifier>
14        <sld:SemanticTypeIdentifier>simple</sld:SemanticTypeIdentifier>
15        <sld:Rule>
16          <sld:Name>Outer</sld:Name>
17          <sld:MaxScaleDenominator>500000.0</sld:MaxScaleDenominator>
18          <sld:LineSymbolizer>
19            <sld:Stroke>
20              <sld:CssParameter name="stroke">#C08B00</sld:CssParameter>
21              <sld:CssParameter name="stroke-linecap">round</sld:CssParameter>
22              <sld:CssParameter name="stroke-linejoin">round</sld:CssParameter>
23              <sld:CssParameter name="stroke-width">6.0</sld:CssParameter>
24            </sld:Stroke>
25          </sld:LineSymbolizer>
26        </sld:Rule>
27      </sld:FeatureTypeStyle>
28    </sld:UserStyle>
29  </sld:UserLayer>
30 </sld:StyledLayerDescriptor>

```

SLD file
 Nessun file selezionato. [Upload ...](#)

Fig. 149: Specifying style name and populating from a file.



GeoServer

About & Status

- Server Status
- GeoServer Logs
- Contact Information
- About GeoServer

Data

- Layer Preview
- Workspaces
- Stores
- Layers
- Layer Groups
- Styles

Services

- WCS
- WFS
- WMS

Settings

- Global
- JAI
- Coverage Access

Tile Caching

- Tile Layers
- Caching Defaults
- Gridsets
- Disk Quota

Security

- Settings

Styles

Manage the Styles published by GeoServer

[Add a new style](#)

[Removed selected style\(s\)](#)

<< < 1 2 > >> Results 1 to 25 (out of 34 items)

<input type="checkbox"/>	Style Name
<input type="checkbox"/>	arealandmarks
<input type="checkbox"/>	arealandmarks_pt
<input type="checkbox"/>	buildings
<input type="checkbox"/>	cemetery_graphics
<input type="checkbox"/>	cemetery_mark
<input type="checkbox"/>	citylimits
<input type="checkbox"/>	contours
<input type="checkbox"/>	countries
<input type="checkbox"/>	county
<input type="checkbox"/>	dem
<input type="checkbox"/>	dem2
<input type="checkbox"/>	dem_elevation
<input type="checkbox"/>	lakes
<input type="checkbox"/>	line
<input type="checkbox"/>	mainrd
<input type="checkbox"/>	parcels
<input type="checkbox"/>	point

Fig. 150: Submitting style

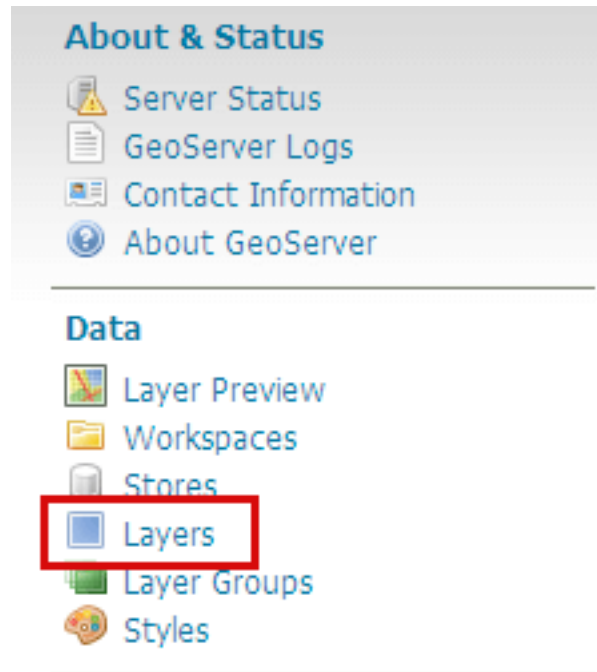


Fig. 151: Navigating to Layers

What you will learn

In this section you will:

Examine an existing style

1. From the GeoServer [Welcome Page](#) navigate to *Style*.
2. From the style list select the *citylimits* style
3. Inside the *Style Editor* we have the following style:

```
<?xml version="1.0" encoding="UTF-8"?>
<sld:StyledLayerDescriptor xmlns="http://www.opengis.net/sld"
  xmlns:sld="http://www.opengis.net/sld"
  xmlns:ogc="http://www.opengis.net/ogc"
  xmlns:gml="http://www.opengis.net/gml"
  version="1.0.0">
  <sld:UserLayer>
    <sld:LayerFeatureConstraints>
      <sld:FeatureTypeConstraint/>
    </sld:LayerFeatureConstraints>
    <sld:UserStyle>
      <sld:Name>BoulderCityLimits</sld:Name>
      <sld:Title/>
      <sld:IsDefault>1</sld:IsDefault>
      <sld:FeatureTypeStyle>
        <sld:Name>group 0</sld:Name>
        <sld:FeatureTypeName>Feature</sld:FeatureTypeName>
```

(continues on next page)

Layers

Manage the layers being published by GeoServer

[+ Add a new resource](#)

[- Remove selected resources](#)

<< < 1 > >> Results 1 to 17 (out of 17 items)


















<input type="checkbox"/>	Type	Workspace	Store	Layer Name
<input type="checkbox"/>		geosolutions	boulder_shapefiles	bbuildings
<input type="checkbox"/>		geosolutions	boulder_shapefiles	blakes
<input type="checkbox"/>		geosolutions	boulder_shapefiles	BoulderCityLimits
<input type="checkbox"/>		geosolutions	boulder_shapefiles	bplandmarks
<input type="checkbox"/>		geosolutions	boulder_shapefiles	bptlandmarks
<input type="checkbox"/>		geosolutions	boulder_shapefiles	bptlandmarks_2876
<input type="checkbox"/>		geosolutions	boulder_shapefiles	brivers
<input type="checkbox"/>		geosolutions	boulder_shapefiles	bstreets
<input type="checkbox"/>		geosolutions	boulder_shapefiles	ccounties
<input type="checkbox"/>		geosolutions	boulder_shapefiles	Parcels
<input type="checkbox"/>		geosolutions	boulder_shapefiles	Trails
<input type="checkbox"/>		geosolutions	boulder_shapefiles	Wetlands_regulatory_area
<input type="checkbox"/>		geosolutions	dem	srtm
<input type="checkbox"/>		geosolutions	hillshade	hillshade
<input type="checkbox"/>		geosolutions	NaturalEarthCountries	WorldCountries
<input type="checkbox"/>		geosolutions	states	states
<input type="checkbox"/>		geosolutions	boulder_shapefiles	Mainrd

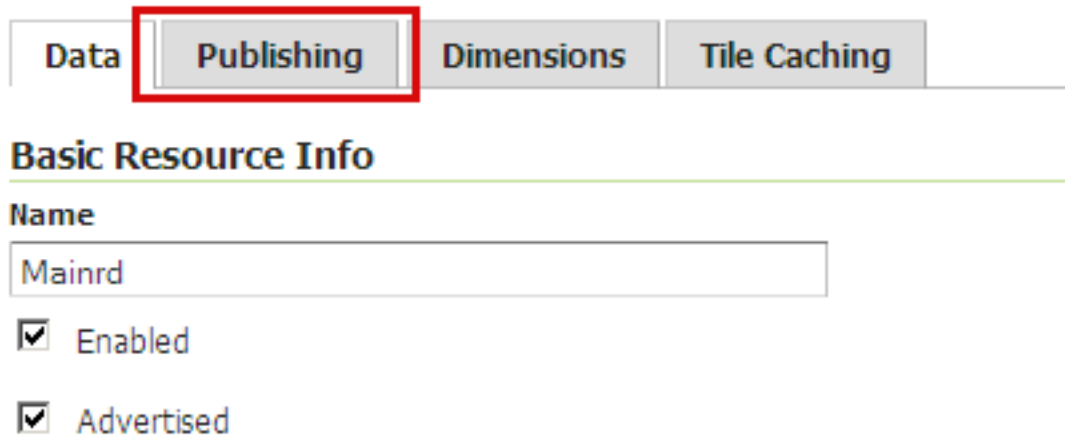
Fig. 152: Selecting a layer

Edit Layer

Edit layer data and publishing

geosolutions:Mainrd

Configure the resource and publishing information for the current layer



Data **Publishing** Dimensions Tile Caching

Basic Resource Info

Name

Mainrd

☒ Enabled

☒ Advertised

Fig. 153: Publish tab

WMS Settings

☒ Queryable

Default Style

mainrd

Outer

Label

Inner

Additional Styles

Available Styles

arealandmarks
arealandmarks_pt
buildings

Fig. 154: Publish tab

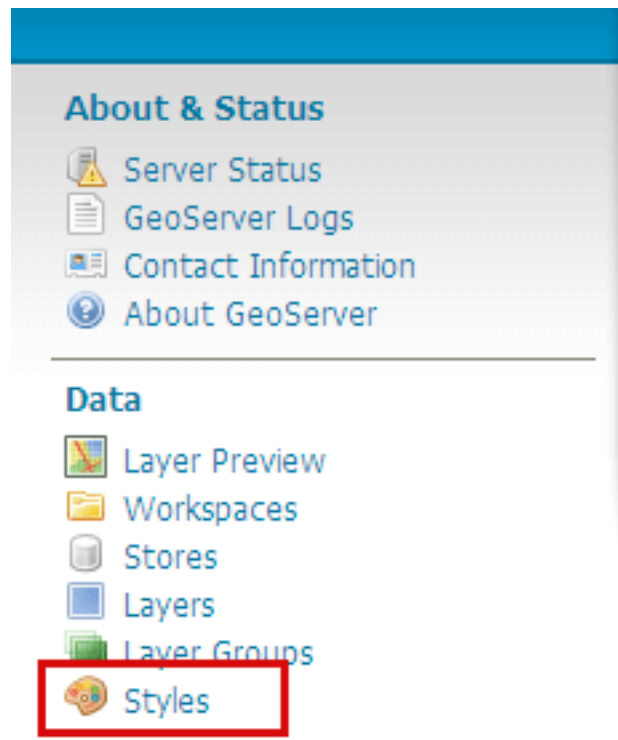


Fig. 155: Navigating to Style configuration

(continued from previous page)

```


<sld:SemanticTypeIdentifier>generic:geometry</
↪sld:SemanticTypeIdentifier>
<sld:SemanticTypeIdentifier>simple</sld:SemanticTypeIdentifier>
<sld:Rule>
  <sld:Name>Filled</sld:Name>
  <sld:MinScaleDenominator>75000</sld:MinScaleDenominator>
  <sld:PolygonSymbolizer>
    <sld:Fill>
      <sld:CssParameter name="fill">#7F7F7F</sld:CssParameter>
      <sld:CssParameter name="fill-opacity">0.5</
↪sld:CssParameter>
    </sld:Fill>
    <sld:Stroke>
      <sld:CssParameter name="stroke">#7F7F7F</
↪sld:CssParameter>
      <sld:CssParameter name="stroke-opacity">0.5</
↪sld:CssParameter>
      <sld:CssParameter name="stroke-width">2.0</
↪sld:CssParameter>
    </sld:Stroke>
  </sld:PolygonSymbolizer>
  <sld:TextSymbolizer>
    <sld:Label>
      <ogc:Literal>Boulder</ogc:Literal>
    </sld:Label>
    <sld:Font>
      <sld:CssParameter name="font-family">Arial</
↪sld:CssParameter>


```

(continues on next page)

Styles

Manage the Styles published by GeoServer

 [Add a new style](#)

 [Removed selected style\(s\)](#)

      Results 1 to 25 (out of 41 items)

<input type="checkbox"/>	Style Name
<input type="checkbox"/>	arealandmarks
<input type="checkbox"/>	arealandmarks_pt
<input type="checkbox"/>	buildings
<input type="checkbox"/>	cemetery_graphics
<input type="checkbox"/>	cemetery_mark
<input type="checkbox"/>	citylimits

Fig. 156: The styles list

(continued from previous page)

```

    <sld:CssParameter name="font-size">14.0</
↪sld:CssParameter>
    <sld:CssParameter name="font-style">normal</
↪sld:CssParameter>
    <sld:CssParameter name="font-weight">normal</
↪sld:CssParameter>
    </sld:Font>
    <sld:LabelPlacement>
      <sld:PointPlacement>
        <sld:AnchorPoint>
          <sld:AnchorPointX>
            <ogc:Literal>0.0</ogc:Literal>
          </sld:AnchorPointX>
          <sld:AnchorPointY>
            <ogc:Literal>0.5</ogc:Literal>
          </sld:AnchorPointY>
        </sld:AnchorPoint>
        <sld:Rotation>
          <ogc:Literal>0</ogc:Literal>
        </sld:Rotation>
      </sld:PointPlacement>
    </sld:LabelPlacement>
    <sld:Fill>
      <sld:CssParameter name="fill">#000000</sld:CssParameter>
    </sld:Fill>
    <sld:VendorOption name="maxDisplacement">200</
↪sld:VendorOption>
    <sld:VendorOption name="Group">true</sld:VendorOption>
  </sld:TextSymbolizer>
</sld:Rule>
</sld:FeatureTypeStyle>
</sld:UserStyle>
</sld:UserLayer>
</sld:StyledLayerDescriptor>

```

Note: The most important section are:

- The `<Rule>` tag combines a number of symbolizers (we have also the possibility to define the OGC filter) to define the portrayal of a feature.
- The `<PolygonSymbolizer>` styles polygons and contain styling information about their border (stroke) and their fill.
- The `<TextSymbolizer>` specifies text labels and their style:
 - `<Label>` Specifies the content of the text label
 - `` Specifies the font information for the labels.
 - `<LabelPlacement>` Sets the position of the label relative its associate feature.
 - `<Fill>` Determines the fill color of the text label.
 - `VendorOption maxDisplacement` Controls the displacement of the label along a line. Normally GeoServer would label a polygon in its centroid, provided the location is not busy with another label and that the label is not too big compare to the polygon, or not label it at all otherwise. When the `maxDisplacement` is set, the labeller will search for another location within `maxDisplacement` pixels from the pre-computed label point.

- VendorOption Group Sometimes you will have a set of related features that you only want a single label for. The grouping option groups all features with the same label text, then finds a representative geometry for the group.
 - The `<MaxScaleDenominator>` and `<MinScaleDenominator>` are used to apply a particular SLD rule to a specific scale. The above SLD makes sure that the Boulder border disappear once we zoom in enough to see the city details. An alternative approach could be to keep the layer showing, but switch it to a different style, for example a thick red line, so that the details of the city are not disturbed by the polygon fill.
-

4. Now from the style list select the *rivers* style.

5. Inside the *Style Editor* we have the following style:

```
<?xml version="1.0" encoding="UTF-8"?>
<sld:StyledLayerDescriptor xmlns="http://www.opengis.net/sld"
  xmlns:sld="http://www.opengis.net/sld"
  xmlns:ogc="http://www.opengis.net/ogc"
  xmlns:gml="http://www.opengis.net/gml"
  version="1.0.0">
  <sld:UserLayer>
    <sld:LayerFeatureConstraints>
      <sld:FeatureTypeConstraint/>
    </sld:LayerFeatureConstraints>
    <sld:UserStyle>
      <sld:Name>Hydrology Line</sld:Name>
      <sld:Title/>
      <sld:FeatureTypeStyle>
        <sld:Rule>
          <sld:Name>default rule</sld:Name>
          <sld:MaxScaleDenominator>75000</sld:MaxScaleDenominator>
          <sld:LineSymbolizer>
            <sld:Stroke>
              <sld:CssParameter name="stroke-width">0.5</
→sld:CssParameter>
              <sld:CssParameter name="stroke">#06607F</
→sld:CssParameter>
            </sld:Stroke>
          </sld:LineSymbolizer>
        </sld:Rule>
      </sld:FeatureTypeStyle>
    </sld:UserStyle>
  </sld:UserLayer>
</sld:StyledLayerDescriptor>
```

Note:

This is a very simple Line style. Take into account the LineSymbolizer that styles lines. Lines are one-dimensional geometry elements that contain position and length. Lines can be comprised of multiple line segments.

The outermost tag is the `<Stroke>` tag. This tag is required, and determines the visualization of the line:

- `stroke` Specifies the solid color given to the line, in the form `#RRGGBB`. Default is black (`#000000`).
- `stroke-width` Specifies the width of the line in pixels. Default is 1.

In this case `MaxScaleDenominator` is used to make sure that the rivers start showing up when we are zoomed in enough, and in particular as the city borders disappear

Create a simple style for points

1. From the GeoServer [Welcome Page](#) navigate to *Style*.

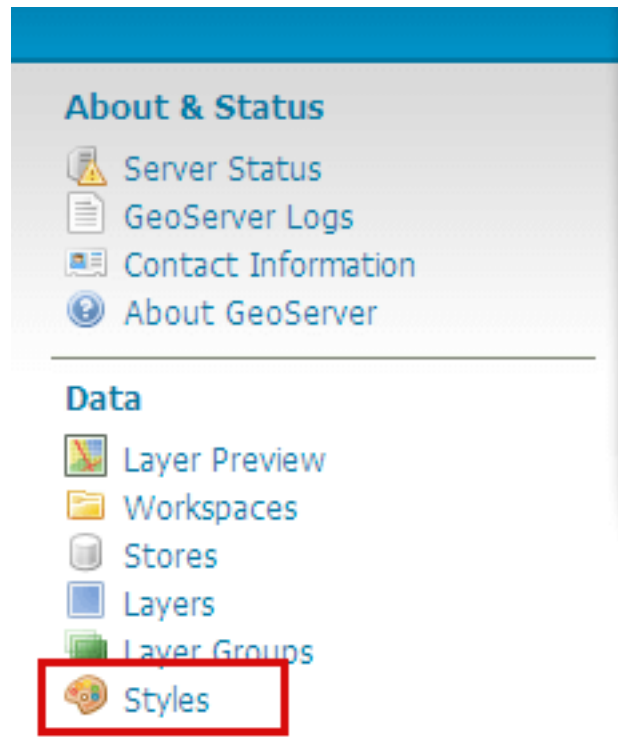


Fig. 157: Navigating to Style configuration

2. Click *New*



Fig. 158: Adding a new style

3. Enter “landmarks” in the *Name* field.
4. In the *SLD Editor* enter the following XML:

New style

Type a new SLD definition, or use an existing one as a template, or upload a ready made the "validate" button to verify the style is a valid SLD document.

Name

Copy from existing style

Scegliarne uno  Copy ...

    12pt 

1
2
3

Fig. 159: Creating a new style

```
<StyledLayerDescriptor xmlns="http://www.opengis.net/sld" xmlns:xsi="http://www.
↪w3.org/2001/XMLSchema-instance" version="1.0.0" xsi:schemaLocation="http://www.
↪opengis.net/sld http://schemas.opengis.net/sld/1.0.0/StyledLayerDescriptor.xsd">
  <NamedLayer>
    <Name>landmarks</Name>
    <UserStyle>
      <Name>landmarks</Name>
      <Title>Point Landmarks</Title>
      <FeatureTypeStyle>
        <Rule>
          <Name>default</Name>
          <Title>Landmarks</Title>
          <PointSymbolizer>
            <Graphic>
              <Mark>
                <WellKnownName>triangle</WellKnownName>
                <Fill>
                  <CssParameter name="fill">#009900</
↪CssParameter>
                  <CssParameter name="fill-opacity">0.2</
↪CssParameter>
                </Fill>
                <Stroke>
                  <CssParameter name="stroke">#000000</
↪CssParameter>
                  <CssParameter name="stroke-width">2</
↪CssParameter>
                </Stroke>
              </Mark>
            </Graphic>
          </PointSymbolizer>
        </Rule>
      </FeatureTypeStyle>
    </UserStyle>
  </NamedLayer>
</StyledLayerDescriptor>
```

(continues on next page)

(continued from previous page)

```

        <Size>12</Size>
      </Graphic>
    </PointSymbolizer>
  </Rule>
</FeatureTypeStyle>
</UserStyle>
</NamedLayer>
</StyledLayerDescriptor>

```

Note:

Take into account:

- **WellKnownName** The name of the common shape. Options are circle, square, triangle, star, cross, or x. Default is s
- **fill** Specifies how the symbolizer should be filled. Options are a `<CssParameter name="fill">` specifying a color in the form #RRGGBB, or `<GraphicFill>` for a fill made with a repeated graphic.
- **fill-opacity** Determines the opacity (transparency) of symbolizers. Values range from 0 (completely transparent) to 1 (completely opaque). Default is 1.

5. Then click *Save* button.
6. Open the `geosolutions:bptlandmarks` vector layer, but this time associate the style as a “Additional Style”:
7. Click on the *Save* button.
8. Preview the `geosolutions:bptlandmarks` layer, which with the default style should be empty due to scale dependencies. Then click the option button at the top left of the map and select the `landmarks` style in the style drop down:

Patterns and Hatches

1. Go and edit the configuration of the `bplandmarks` layer, enter the publish tab and associate the `cemetery_mark` and `cemetery_graphics` styles as “Additional styles” for the layer, then press “Save”
2. From the [Welcome Page](#) navigate to *Styles*.

Note: You have to be logged in as Administrator in order to activate this function.

3. Select “cemetery_graphics” from the list
4. In the *SLD Editor* you will see the following XML:

```

<?xml version="1.0" encoding="UTF-8"?>
  <sld:StyledLayerDescriptor
    xmlns="http://www.opengis.net/sld"
    xmlns:sld="http://www.opengis.net/sld"
    xmlns:ogc="http://www.opengis.net/ogc"
    xmlns:gml="http://www.opengis.net/gml"
    xmlns:xlink="http://www.w3.org/1999/xlink" version="1.0.0">









```

(continues on next page)

WMS Settings

☒ Queryable

Default Style

 shopping
 mountains
 jails
 government
 airport
 school
 cemetery
 gate

Additional Styles



Available Styles		Selected Styles
arealandmarks	 	landmarks
arealandmarks_pt		
buildings		
cemetery_graphics		
cemetery_mark		
citylimits		
contours		
countries		
county		
dem		

Fig. 160: Open the Layers Preview

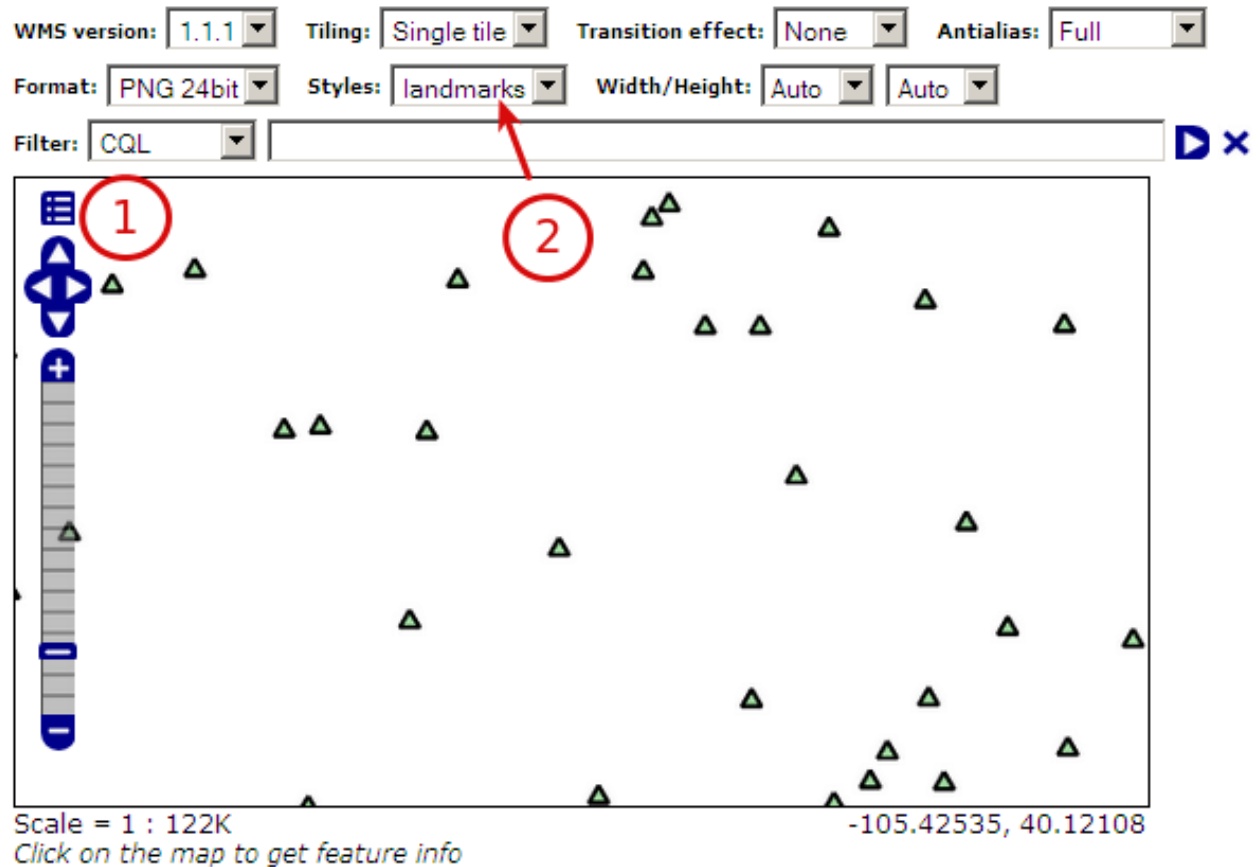


Fig. 161: Open the Layers Preview

WMS Settings

☒ Queryable

Default Style

arealandmarks

- ☐ default rule
- ☒ park
- ☒ nationalpark
- ☒ cemeters

Additional Styles


Available Styles	
arealandmarks	<input type="checkbox"/>
arealandmarks_pt	<input type="checkbox"/>
buildings	<input type="checkbox"/>
citylimits	<input type="checkbox"/>
contours	<input type="checkbox"/>
countries	<input type="checkbox"/>
county	<input type="checkbox"/>
dem	<input type="checkbox"/>
dem2	<input type="checkbox"/>
dem3	<input type="checkbox"/>










Selected Styles	
cemetery_mark	<input type="checkbox"/>
cemetery_graphics	<input type="checkbox"/>

Styles

Manage the Styles published by GeoServer

 Add a new style

 Removed selected style(s)

      Results 1 to 25 (out of 43 items)

<input type="checkbox"/>	Style Name
<input type="checkbox"/>	arealandmarks
<input type="checkbox"/>	arealandmarks_pt
<input type="checkbox"/>	buildings
<input type="checkbox"/>	cemetery_graphics
<input type="checkbox"/>	cemetery_mark
<input type="checkbox"/>	citylimits

Fig. 162: Patterns filling SLD

(continued from previous page)

```

<sld:UserLayer>
  <sld:UserStyle>
    <sld:Name>tl 2010 08013 arealm</sld:Name>
    <sld:Title/>
    <sld:FeatureTypeStyle>
      <sld:Rule>
        <sld:Name>cemeteries</sld:Name>
        <ogc:Filter>
          <ogc:PropertyIsEqualTo>
            <ogc:PropertyName>MTFCC</ogc:PropertyName>
            <ogc:Literal>K2582</ogc:Literal>
          </ogc:PropertyIsEqualTo>
        </ogc:Filter>
        <sld:MaxScaleDenominator>500000.0</sld:MaxScaleDenominator>
        <sld:PolygonSymbolizer>
          <sld:Fill>
            <sld:GraphicFill>
              <sld:Graphic>
                <sld:ExternalGraphic>
                  <sld:OnlineResource>
                    xlink:type="simple"
                    xlink:href="./img/landmarks/area/
→grave_yard.png" />
                  <sld:Format>image/png</sld:Format>
                </sld:ExternalGraphic>
              </sld:Graphic>
            </sld:GraphicFill>
          </sld:Fill>
        </sld:PolygonSymbolizer>
      </sld:Rule>
    </sld:FeatureTypeStyle>
  </sld:UserStyle>
</sld:UserLayer>
</sld:StyledLayerDescriptor>

```

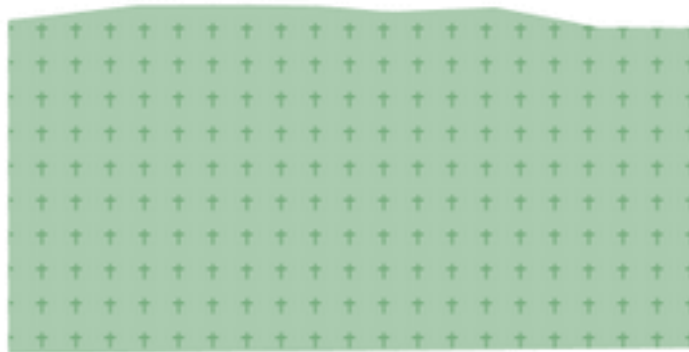


Fig. 163: Filling with patterns

Note: The above SLD defines a `<PolygonSymbolizer>` with a `<GraphicFill>` pointing to a png

`/img/landmarks/area/grave_yard.png` in the GeoServer data directory, which will be used by GeoServer as pattern to fill the polygon.

- Like before, select now “cemetery_mark” from the list

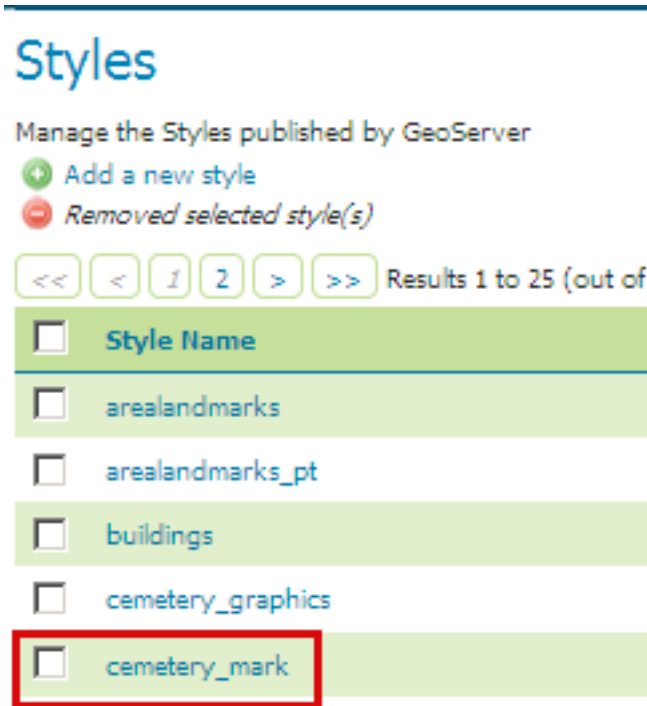


Fig. 164: True Type Font filling SLD

- In the *SLD Editor* you will see the following XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<sld:StyledLayerDescriptor
  xmlns="http://www.opengis.net/sld"
  xmlns:sld="http://www.opengis.net/sld"
  xmlns:ogc="http://www.opengis.net/ogc"
  xmlns:gml="http://www.opengis.net/gml"
  xmlns:xlink="http://www.w3.org/1999/xlink" version="1.0.0">

  <sld:UserLayer>
    <sld:Name>cemeteries</sld:Name>
    <sld:UserStyle>
      <sld:Name>t1 2010 08013 arealm</sld:Name>
      <sld:Title/>
      <sld:FeatureTypeStyle>

        <sld:Rule>
          <sld:Name>cemeteries</sld:Name>
          <ogc:Filter>
            <ogc:PropertyIsEqualTo>
              <ogc:PropertyName>MTFCC</ogc:PropertyName>
              <ogc:Literal>K2582</ogc:Literal>
            </ogc:PropertyIsEqualTo>
          </ogc:Filter>
        </sld:Rule>
      </sld:FeatureTypeStyle>
    </sld:UserStyle>
  </sld:UserLayer>
</sld:StyledLayerDescriptor>
```

(continues on next page)

(continued from previous page)

```

</ogc:Filter>
<sld:MaxScaleDenominator>500000.0</sld:MaxScaleDenominator>
<sld:PolygonSymbolizer>
  <sld:Fill>
    <sld:CssParameter name="fill">#D3FFD3</sld:CssParameter>
    <sld:CssParameter name="fill-opacity">0.5</sld:CssParameter>
  </sld:Fill>
  <sld:Stroke>
    <sld:CssParameter name="stroke">#6DB26D</sld:CssParameter>
  </sld:Stroke>
</sld:PolygonSymbolizer>
<sld:PolygonSymbolizer>
  <sld:Fill>
    <sld:GraphicFill>
      <sld:Graphic>
        <sld:Mark>
          <sld:WellKnownName>ttf://Wingdings#0x0055</sld:WellKnownName>
          <sld:Stroke>
            <sld:CssParameter name="stroke">#6DB26D</sld:CssParameter>
          </sld:Stroke>
        </sld:Mark>
        <sld:Size>16</sld:Size>
      </sld:Graphic>
    </sld:GraphicFill>
  </sld:Fill>
  <sld:VendorOption name="graphic-margin">8</sld:VendorOption>
</sld:PolygonSymbolizer>

</sld:Rule>

</sld:FeatureTypeStyle>
</sld:UserStyle>
</sld:UserLayer>
</sld:StyledLayerDescriptor>

```

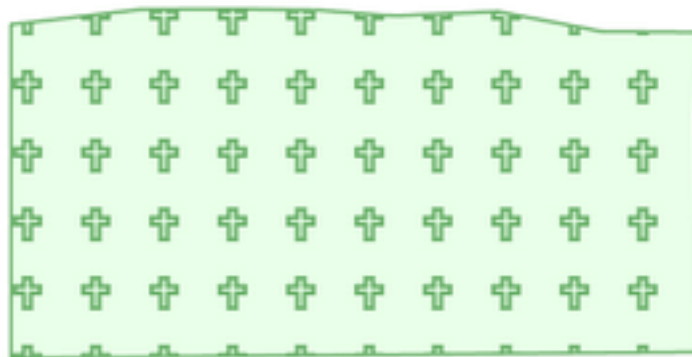


Fig. 165: Filling with TTF fonts

Note: The above SLD defines a `<PolygonSymbolizer>` with a `<GraphicFill>` looking for a specific *Windings* character which will be used by GeoServer as pattern to fill the polygon. The `graphic-margin`

`VendorOption` is used to add some space around symbols.

7. Lets now take a look at another way to fill polygons using patterns, the *Hatches*. From the [Welcome Page](#) navigate to *Styles* and select “wetlands” from the list.

Note: You may switch to the second page in order to find the style.



Fig. 166: Wetlands style with some hatches

```
<?xml version="1.0" encoding="UTF-8"?>
  <sld:StyledLayerDescriptor xmlns="http://www.opengis.net/sld" xmlns:sld=
    ↪ "http://www.opengis.net/sld" xmlns:ogc="http://www.opengis.net/ogc" xmlns:gml=
    ↪ "http://www.opengis.net/gml" version="1.0.0">
    <sld:UserLayer>
      <sld:LayerFeatureConstraints>
        <sld:FeatureTypeConstraint/>
      </sld:LayerFeatureConstraints>
      <sld:UserStyle>
        <sld:Name>Wetlands regulatory area</sld:Name>
        <sld:Title/>
        <sld:FeatureTypeStyle>
          <sld:Rule>
            <sld:Name>default rule</sld:Name>
            <sld:MaxScaleDenominator>10000.0</sld:MaxScaleDenominator>
            <sld:PolygonSymbolizer>
```

(continues on next page)

(continued from previous page)

```

        <sld:Fill>
          <sld:GraphicFill>
            <sld:Graphic>
              <sld:Mark>
                <sld:WellKnownName>shape://times</
↪sld:WellKnownName>

                <sld:Fill/>
                <sld:Stroke>
                  <sld:CssParameter name="stroke">

↪#ADD8E6</sld:CssParameter>

                  <sld:CssParameter name="stroke-width
↪">1.0</sld:CssParameter>

                  </sld:Stroke>
                </sld:Mark>
              <sld:Size>
                <ogc:Literal>8.0</ogc:Literal>
              </sld:Size>
            </sld:Graphic>
          </sld:GraphicFill>
        <!--
↪sld:CssParameter>
        <sld:CssParameter name="fill">#7CE3F8</
↪sld:CssParameter>
        <sld:CssParameter name="fill-opacity">0.5</
        -->
      </sld:Fill>
    </sld:PolygonSymbolizer>
  </sld:Rule>
</sld:FeatureTypeStyle>
</sld:UserStyle>
</sld:UserLayer>
</sld:StyledLayerDescriptor>

```

8. Comment out the following line in order to see the polygons at lower zoom levels too:

```
<!-- sld:MaxScaleDenominator>10000.0</sld:MaxScaleDenominator -->
```

9. Click *Submit* to add the new SLD.
10. To see how the styles work, make sure the default style of the *Wetlands_regulatory_area* feature type is set to *wetlands*.
11. Use the [Map Preview](#) to preview the new style.
12. On the previous example we used *times* as hatches mark. GeoServer makes available different kinds of hatches marks:

Dashes

1. Lets now familiarize a bit with *Dashes*. We are going to see how it's possible to draw several kind of dashes to represent different types of trails or roads.
2. From the [Welcome Page](#) navigate to *Styles*.

Note: You have to be logged in as Administrator in order to activate this function.

Edit Layer

Edit layer data and publishing

geosolutions:Wetlands_regulatory_area

Configure the resource and publishing information for the current layer

Data **Publishing**

Edit Layer

Name

☒ Enabled

☒ Advertised

HTTP Settings

☐ Response Cache Headers

Cache Time (seconds)

WFS Settings

Per-Request Feature Limit

Maximum number of decimals

WMS Settings

☒ Queryable

Default Style

Fig. 167: Changing the default style of the *Wetlands_regulatory_area* feature type to *wetlands*

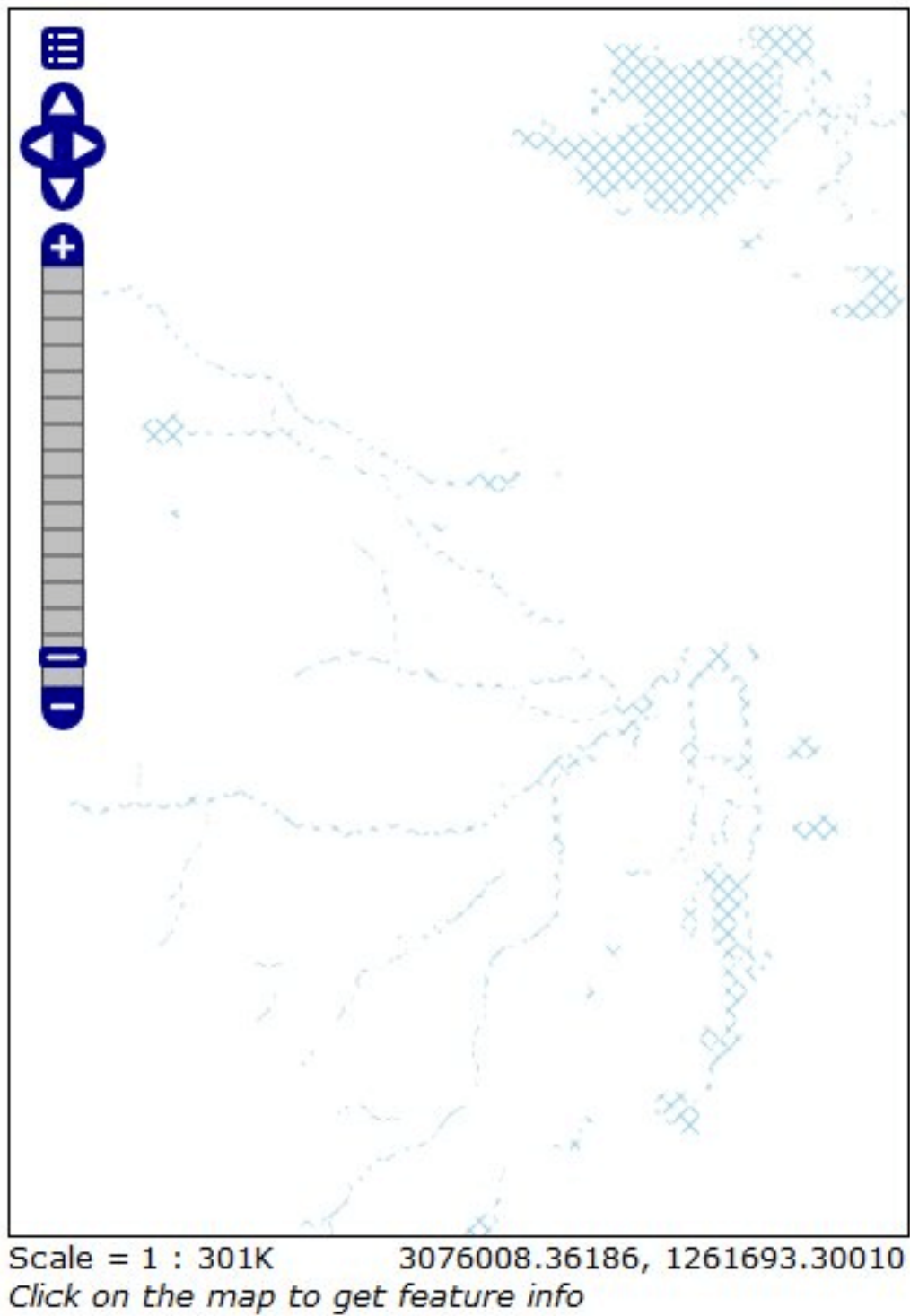


Fig. 168: Previewing the *bplandmarks* layer with the hatches applied

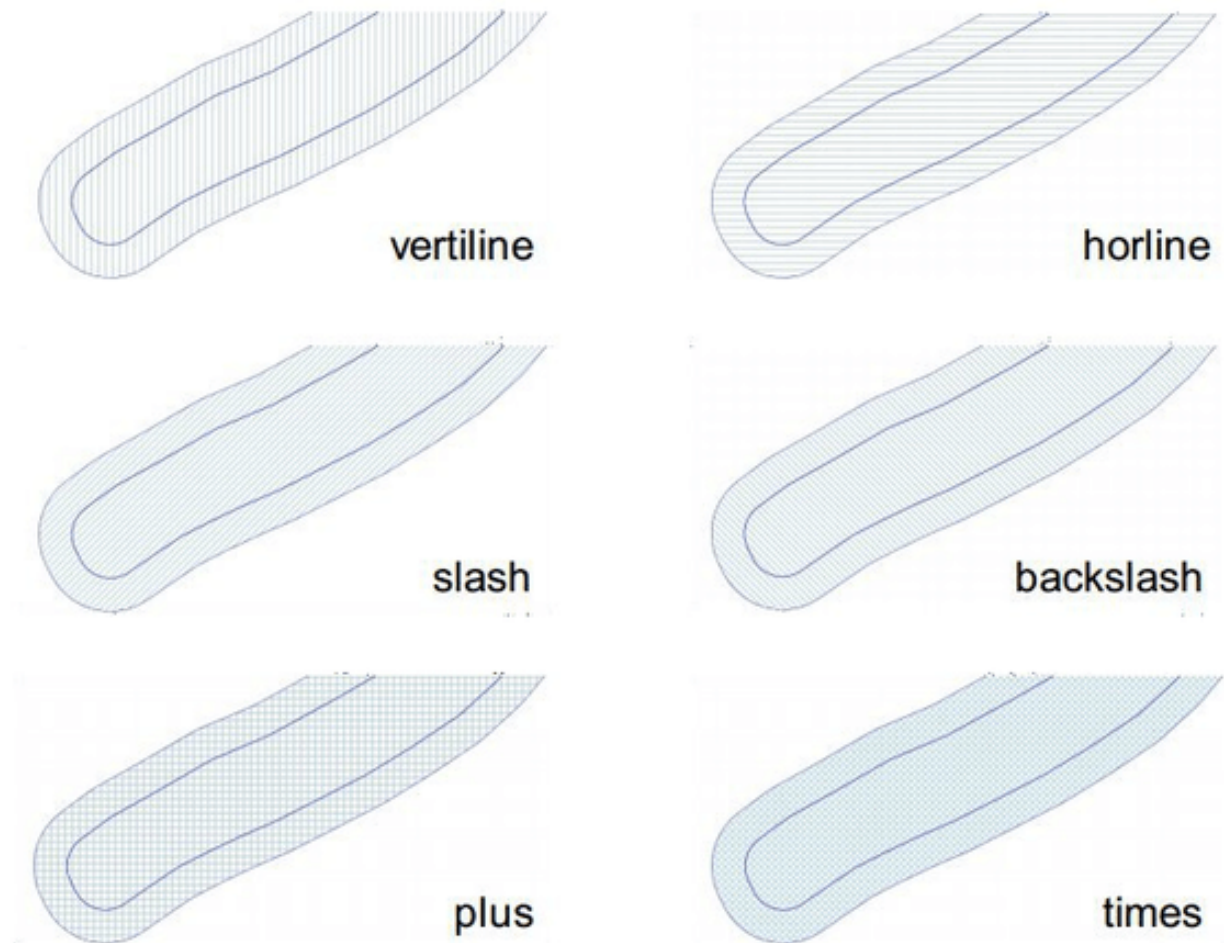


Fig. 169: Different types of hatches marks.

3. Select “trails” from the list

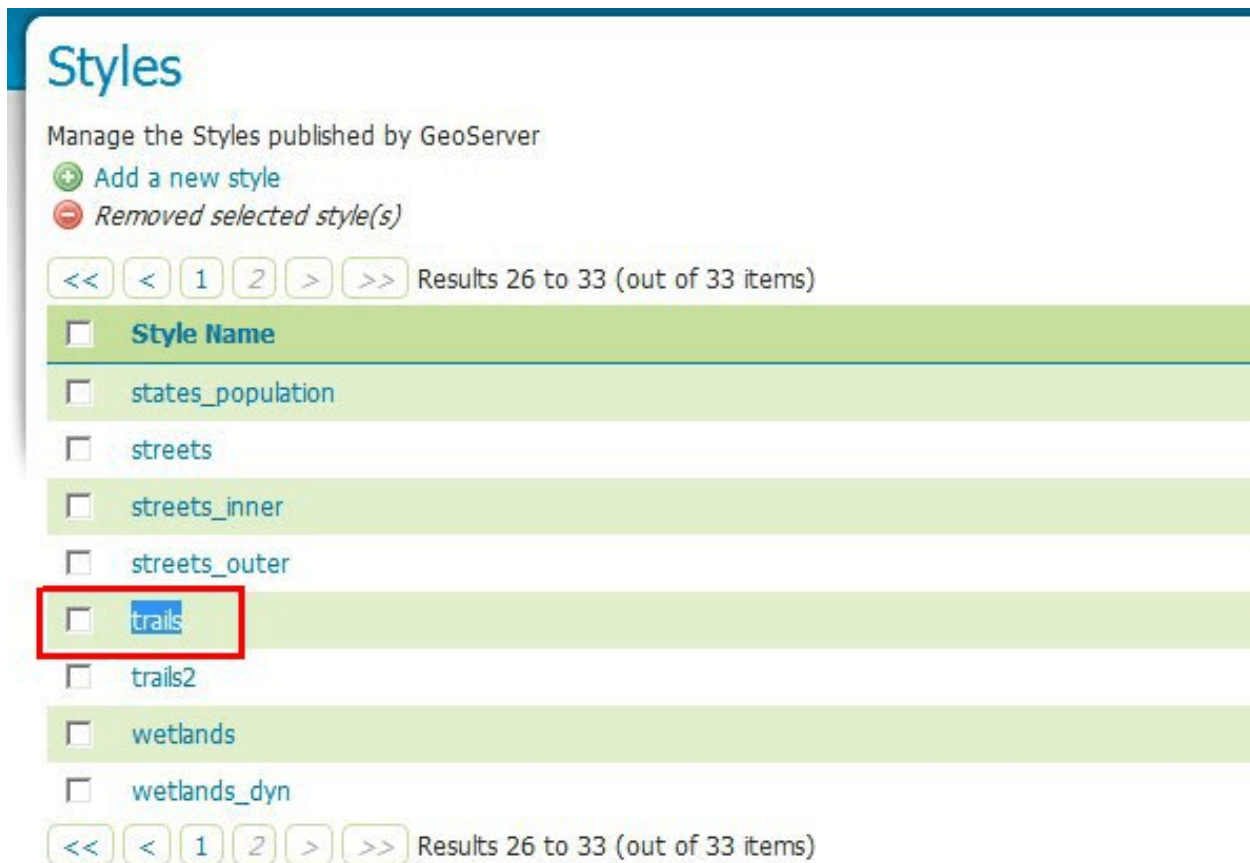


Fig. 170: Dashes SLD

4. In the *SLD Editor* you will see the following XML:

```
<?xml version="1.0" encoding="UTF-8"?>
  <sld:StyledLayerDescriptor xmlns="http://www.opengis.net/sld" xmlns:sld=
    ↪ "http://www.opengis.net/sld" xmlns:ogc="http://www.opengis.net/ogc" xmlns:gml=
    ↪ "http://www.opengis.net/gml" version="1.0.0">
    <sld:UserLayer>
      <sld:LayerFeatureConstraints>
        <sld:FeatureTypeConstraint/>
      </sld:LayerFeatureConstraints>
      <sld:UserStyle>
        <sld:Name>Trails</sld:Name>
        <sld:Title/>
        <sld:FeatureTypeStyle>
          <sld:Rule>
            <sld:MaxScaleDenominator>75000</sld:MaxScaleDenominator>
            <sld:LineSymbolizer>
              <sld:Stroke>
                <sld:CssParameter name="stroke">#6B4900</
    ↪ sld:CssParameter>
                <sld:CssParameter name="stroke-width">0.1</
    ↪ sld:CssParameter>
                <sld:CssParameter name="stroke-dasharray">2.0 </
    ↪ sld:CssParameter>
```

(continues on next page)

(continued from previous page)

```

        </sld:Stroke>
      </sld:LineSymbolizer>
    </sld:Rule>
  </sld:FeatureTypeStyle>
</sld:UserStyle>
</sld:UserLayer>
</sld:StyledLayerDescriptor>

```



Fig. 171: Simple dash-array

Note: The above SLD defines a `<LineSymbolizer>` with a `<Stroke>` using the CSS property `stroke-dasharray` to represent the trails like a simple gray dash.

Note: Encodes a dash pattern as a series of numbers separated by spaces. Odd-indexed numbers (first, third, etc) determine the length in pixels to draw the line, and even-indexed numbers (second, fourth, etc) determine the length in pixels to blank out the line. Default is an unbroken line. Starting from version 2.1 dash arrays can be combined with graphic strokes to generate complex line styles with alternating symbols or a mix of lines and symbols.

5. The Style above is the default one for the layer `geosolutions:Trails`. Let's have a look at a bit more complex example. From the [Welcome Page](#) navigate to *Styles* and select "trails2" from the list
6. In the *SLD Editor* you will see the following XML:

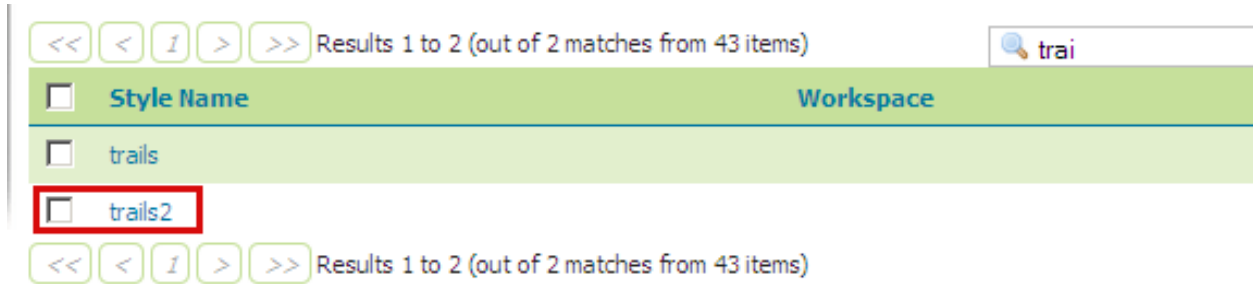


Fig. 172: Trails2 Style

```
<?xml version="1.0" encoding="UTF-8"?>
  <sld:StyledLayerDescriptor xmlns="http://www.opengis.net/sld" xmlns:sld=
    ↪ "http://www.opengis.net/sld" xmlns:ogc="http://www.opengis.net/ogc" xmlns:gml=
    ↪ "http://www.opengis.net/gml" version="1.0.0">
    <sld:UserLayer>
      <sld:LayerFeatureConstraints>
        <sld:FeatureTypeConstraint/>
      </sld:LayerFeatureConstraints>
      <sld:UserStyle>
        <sld:Name>Trails</sld:Name>
        <sld:Title/>
        <sld:FeatureTypeStyle>
          <sld:Rule>
            <sld:MaxScaleDenominator>75000</sld:MaxScaleDenominator>
            <sld:LineSymbolizer>
              <sld:Stroke>
                <sld:GraphicStroke>
                  <sld:Graphic>
                    <sld:Mark>
                      <sld:WellKnownName>circle</
    ↪ sld:WellKnownName>
                      <sld:Fill>
                        <sld:CssParameter name="fill">
    ↪ #AA0000</sld:CssParameter>
                      </sld:Fill>
                    </sld:Mark>
                  <sld:Size>
                    <ogc:Literal>6</ogc:Literal>
                  </sld:Size>
                </sld:Graphic>
              </sld:GraphicStroke>
              <sld:CssParameter name="stroke-dasharray">6 18</
    ↪ sld:CssParameter>
            </sld:Stroke>
          </sld:LineSymbolizer>
          <sld:LineSymbolizer>
            <sld:Stroke>
              <sld:CssParameter name="stroke">#AA0000</
    ↪ sld:CssParameter>
              <sld:CssParameter name="stroke-dasharray">10 14</
    ↪ sld:CssParameter>
              <sld:CssParameter name="stroke-dashoffset">14</
    ↪ sld:CssParameter>
```

(continues on next page)

(continued from previous page)

```
        </sld:Stroke>
      </sld:LineSymbolizer>
    </sld:Rule>
  </sld:FeatureTypeStyle>
</sld:UserStyle>
</sld:UserLayer>
</sld:StyledLayerDescriptor>
```

Note: We may notice two interesting things in this style, two `<LineSymbolizer>` the first one defining a *circle* Mark with a simple dasharray and the second one a simple stroke defining also a *dashoffset*. The latter specifies the distance in pixels into the dasharray pattern at which to start drawing. Default is 0.

7. Open the *geosolutions:Trails* layers and add *trails2* as an additional style, then go to the *Layer Preview* to see it in action

Warning: You have to zoom in from the layer preview in order to see the lines due to the *MaxScaleDenominator*

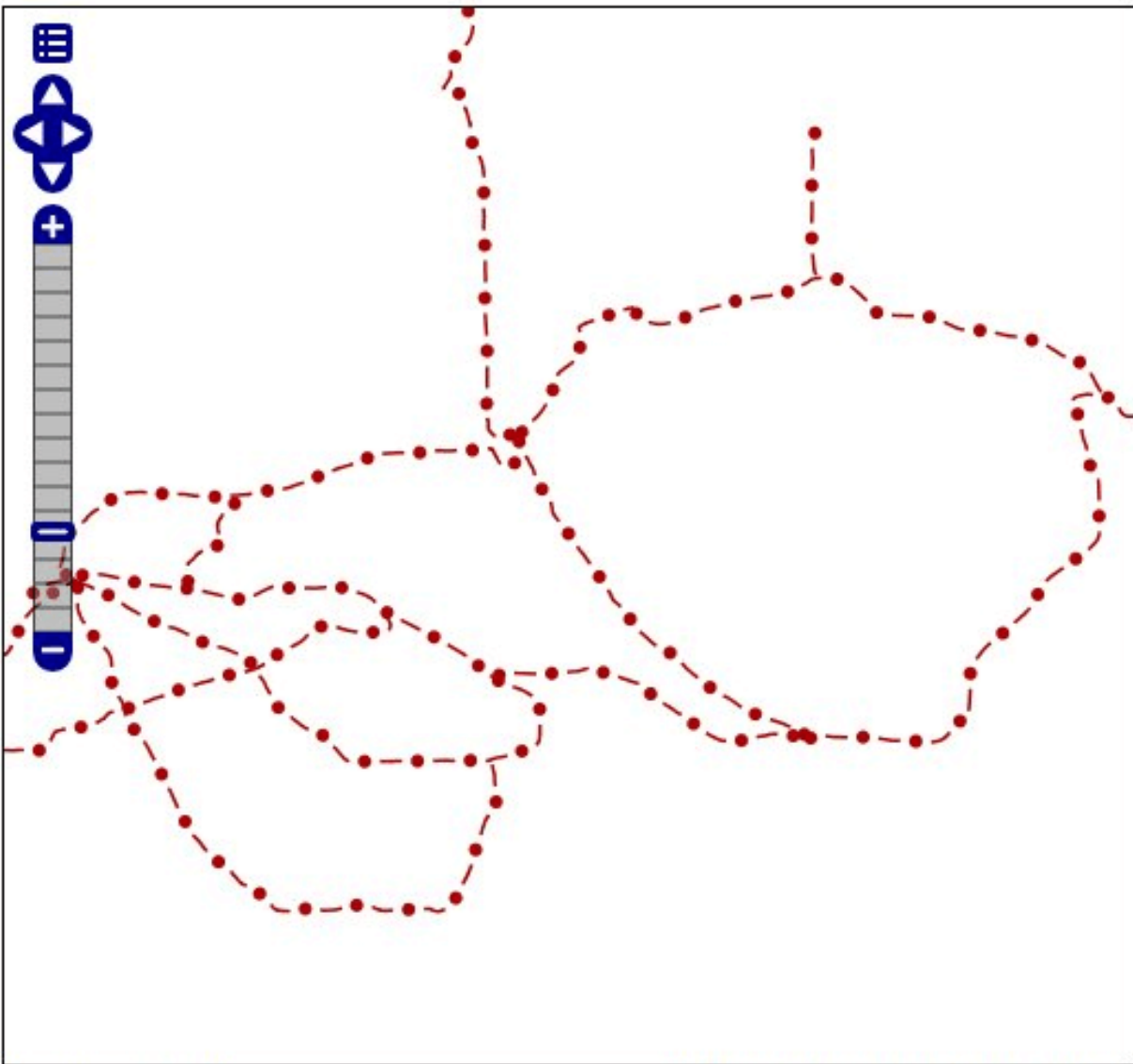
Roads and labelling roads

1. From the [Welcome Page](#) navigate to *Styles* → *mainrd* in order to edit the mainrd SLD.

Note: You have to be logged in as Administrator in order to activate this function.

2. In the *SLD Editor* find the *sld:TextSymbolizer* associated to the *ogc:PropertyName LABEL_NAME*

Note: The style defines a `` and an `<Halo>` in order to render the value of the property *LABEL_NAME* for that layer. The interesting part is at the bottom where several `<VendorOption>` are specified. Those options are GeoServer specific and allows us to have better and nicer result by tweaking the label renderer behaviour.



Scale = 1 : 67K

3066676.01511, 1271517.47177

Click on the map to get feature info



Fig. 173: Road style

Option	Description	Type
followLine	<p>The followLine option forces a label to follow the curve of the line.</p> <pre><VendorOption name="followLine">true</VendorOption></pre> <p>To use this option place the following in your <TextSymbolizer>. It is required to use <LinePlacement> along with this option to ensure that all labels are correctly following the lines:</p> <pre><LabelPlacement> <LinePlacement/> </LabelPlacement></pre>	boolean
repeat	<p>The repeat option determines how often GeoServer labels a line. Normally GeoServer would label each line only once, regardless of their length. Specify a positive value to make it draw the label every repeat pixels.</p> <pre><VendorOption name="repeat">100</VendorOption></pre>	number
group	<p>Sometimes you will have a set of related features that you only want a single label for. The grouping option groups all features with the same label text, then finds a representative geometry for the group.</p> <p>Roads data is an obvious example - you only want a single label for all of main street, not a label for every piece of main street.</p> <p>When the grouping option is off (default), grouping is not performed and each geometry is labelled (space permitting).</p> <p>With the grouping option on, all the geometries with the same label are grouped together and the label position is determined from ALL the geometries.</p> <ul style="list-style-type: none"> • Point Set first point inside the view rectangle is used. • Line Set lines are (a) networked together (b) clipped to the view rectangle (c) middle of the longest network path is used. • Polygon Set polygons are (a) clipped to the view rectangle (b) the centroid of the largest polygon is used. <pre><VendorOption name="group">yes</VendorOption></pre>	enum{yes/no}
676	<p>Chapter 3. Table of contents</p>	

Another important thing to notice in this style is the **road casing**, that is, the fact each road segment is painted by two overlapping strokes of different color and size.

Placing the strokes in the two separate feature type styles is crucial:

- with the symbolizers in two separate `FeatureTypeStyle` element all roads are painted with the large stroke, and then again with the thin, lighter one.
- if instead the two symbolizers were placed in the same `FeatureTypeStyle` element the result would be different, and not pleasing to see, since the renderer would take the first road, paint with the large and thin strokes in sequence, then move to the next one and repeat until the end

Styling point data

Point data in SLD can be depicted with `PointSymbolizer` and labelled with `TextSymbolizer`. This section describe an existing, realistic style, available in the data directory that depicts the *point landmarks* layer (`bptlandmarks`) with icons and labels.

The dataset

The `bptlandmarks` layer (Boulder point landmarks) contains the location of significant point entities such as malls, schools, airports and the like. The attribute structure is reported in the [GeoServer page for such layer](#):

The style will use the `MTFCC` code to categorize the various points in the different types (e.g., schools have `MTFCC = K2543`, and eventually use `FULLNAME` for the label. This results in the [following map](#):

The complete style we'll be referring to is named `point_landmark`, you can have a look at the full style in the [GeoServer style editor](#):

Point symbolizers

A point symbolizer depicts a symbol by means of a `Mark` or a `External Graphic`. The former is a built-in vector symbol that can be stroked and filled at the styler will, but only a handful of such symbols are available, whilst the latter can be a user provided image or SVG graphic.

The point landmark styles use the Open Street Map icons for most of the locations. The images have been added inside the data directory, inside `styles/im`, since this allows to refer them by relative path:

Given the above symbols a point symbolizer looks as follows:

```
<sld:PointSymbolizer>
  <sld:Graphic>
    <sld:ExternalGraphic>
      <sld:OnlineResource xlink:type="simple" xlink:href="./img/landmarks/
↪school.png" />
      <sld:Format>image/png</sld:Format>
    </sld:ExternalGraphic>
  </sld:Graphic>
  <VendorOption name="labelObstacle">true</VendorOption>
</sld:PointSymbolizer>
```

The icon is depicted on the screen as-is, at its natural resolutions. The `labelObstacle` vendor parameter, specific to GeoServer, makes sure the point is icon is treated as a [label obstacle](#), that is, makes sure no label will ever be depicted over the point.

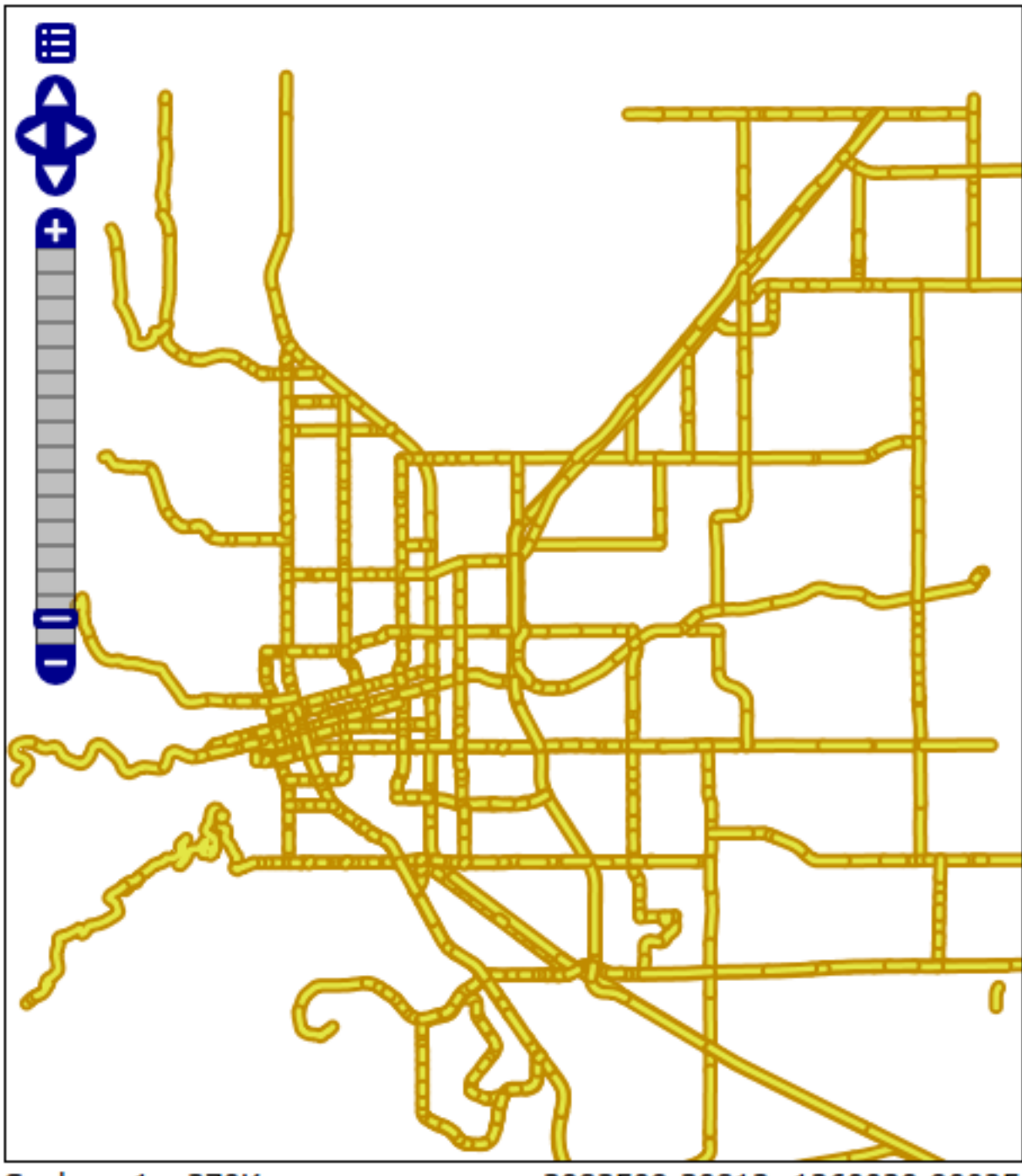


Fig. 174: Road casing with a single FeatureTypeStyle element

Feature Type Details

Property	Type	Nullable	Min/Max Occurrences
the_geom	Point	true	0/1
STATEFP	String	true	0/1
COUNTYFP	String	true	0/1
ANSICODE	String	true	0/1
POINTID	String	true	0/1
FULLNAME	String	true	0/1
MTFCC	String	true	0/1

Fig. 175: Point landmarks attribute structure

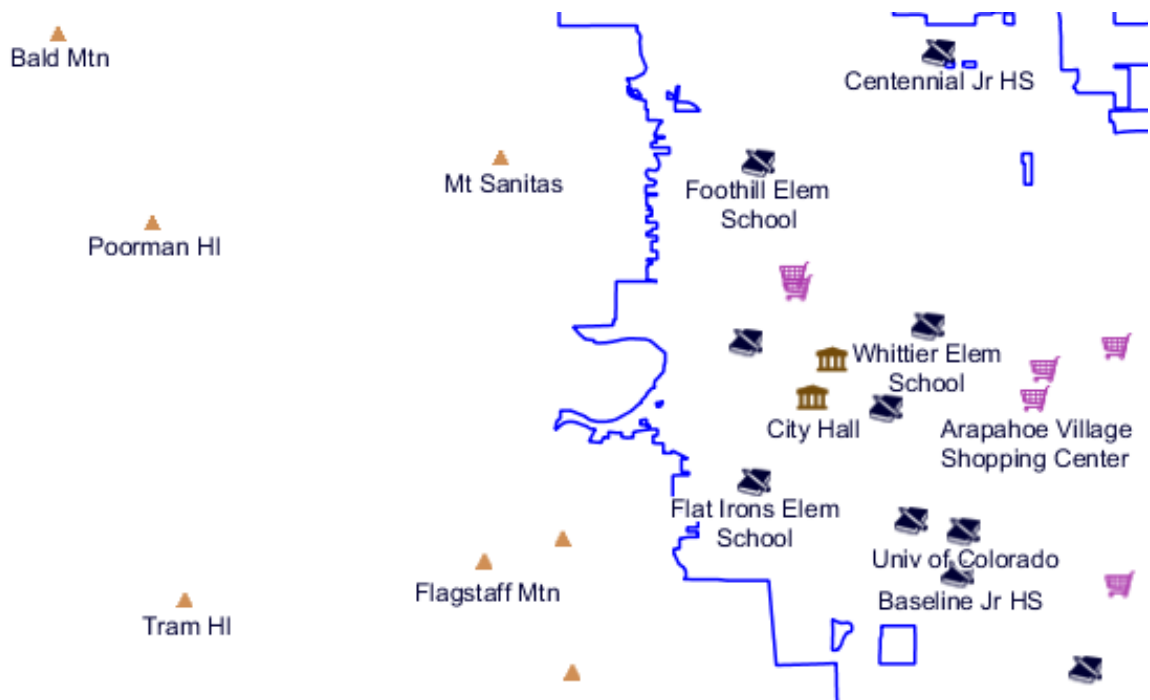


Fig. 176: Point landmarks in Boulder

```

336         </sld:Displacement>
337         <sld:Rotation>
338             <ogc:Literal>0.0</ogc:Literal>
339         </sld:Rotation>
340     </sld:PointPlacement>
341 </sld:LabelPlacement>
342 <sld:Fill>
343     <sld:CssParameter name="fill">#000033</sld:CssParameter>
344 </sld:Fill>
345 <sld:Priority>200000</sld:Priority>
346 <sld:VendorOption name="autoWrap">100</sld:VendorOption>
347 </sld:TextSymbolizer>
348 </sld:Rule>
349
350 <sld:Rule>
351     <sld:Name>school</sld:Name>
352     <ogc:Filter>
353         <ogc:PropertyIsEqualTo>
354             <ogc:PropertyName>MTFCC</ogc:PropertyName>
355             <ogc:Literal>K2543</ogc:Literal>
356         </ogc:PropertyIsEqualTo>
357     </ogc:Filter>
358     <sld:MaxScaleDenominator>100000</sld:MaxScaleDenominator>
359     <VendorOption name="labelObstacle">true</VendorOption>
360     <sld:PointSymbolizer>
361         <sld:Graphic>
362             <sld:ExternalGraphic>
363                 <sld:OnlineResource xlink:type="simple" xlink:href="./img/landmarks/school.png" />
364                 <sld:Format>image/png</sld:Format>
365             </sld:ExternalGraphic>
366         </sld:Graphic>

```

Fig. 177: Point landmarks style

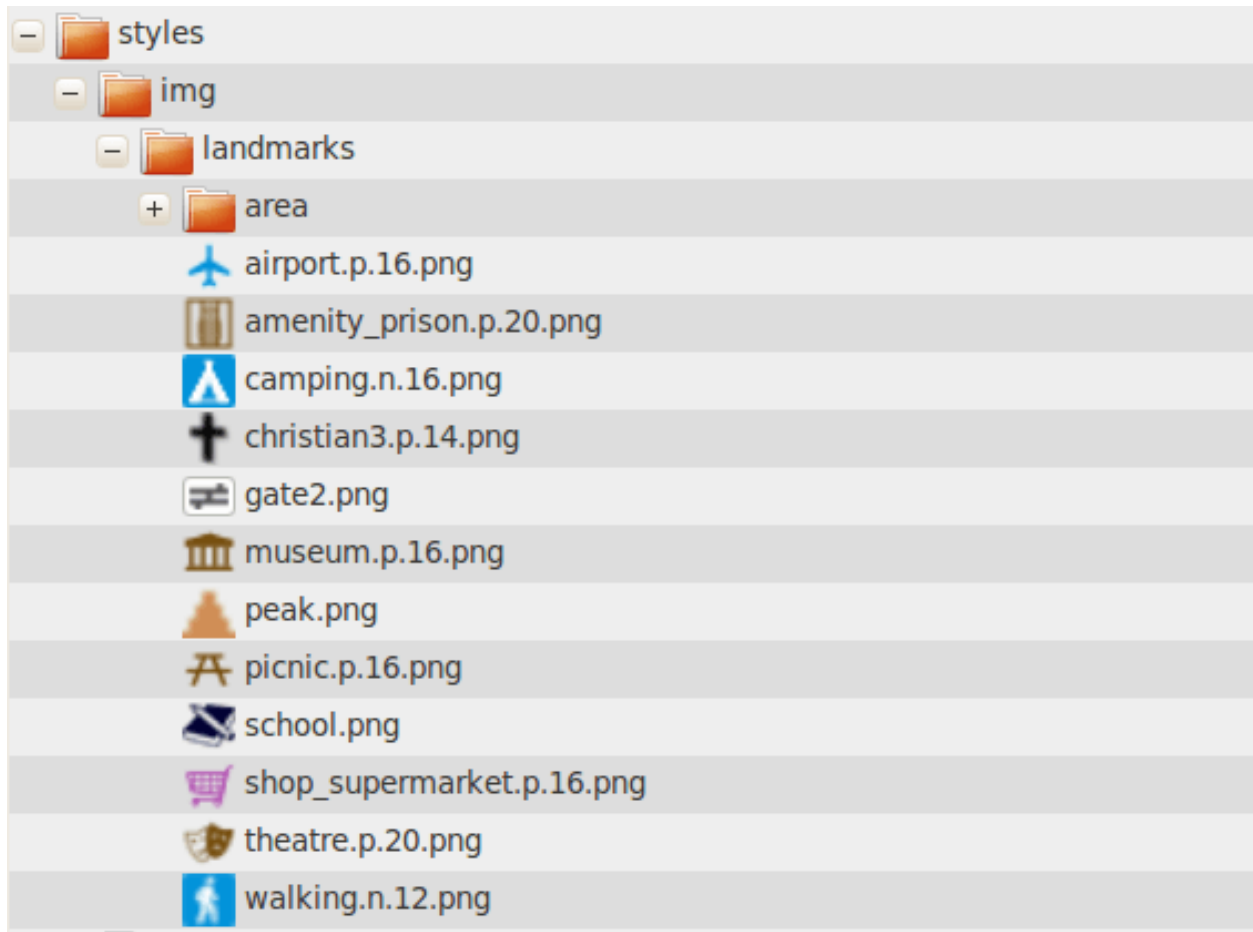


Fig. 178: Point landmarks style

Text symbolizers for points

The text symbolizer associates a label with a point using an attribute value as the label source. The following symbolizer is used to label schools:

```
<sld:TextSymbolizer>
  <sld:Label>
    <ogc:PropertyName>FULLNAME</ogc:PropertyName>
  </sld:Label>
  <sld:Font>
    <sld:CssParameter name="font-family">Arial</sld:CssParameter>
    <sld:CssParameter name="font-size">12.0</sld:CssParameter>
    <sld:CssParameter name="font-style">normal</sld:CssParameter>
    <sld:CssParameter name="font-weight">normal</sld:CssParameter>
  </sld:Font>
  <sld:LabelPlacement>
    <sld:PointPlacement>
      <sld:AnchorPoint>
        <sld:AnchorPointX>
          <ogc:Literal>0.5</ogc:Literal>
        </sld:AnchorPointX>
        <sld:AnchorPointY>
          <ogc:Literal>1.0</ogc:Literal>
        </sld:AnchorPointY>
      </sld:AnchorPoint>
      <sld:Displacement>
        <sld:DisplacementX>
          <ogc:Literal>0.0</ogc:Literal>
        </sld:DisplacementX>
        <sld:DisplacementY>
          <ogc:Literal>-10.0</ogc:Literal>
        </sld:DisplacementY>
      </sld:Displacement>
      <sld:Rotation>
        <ogc:Literal>0.0</ogc:Literal>
      </sld:Rotation>
    </sld:PointPlacement>
  </sld:LabelPlacement>
  <sld:Halo>
    <sld:Radius>
      <ogc:Literal>1.5</ogc:Literal>
    </sld:Radius>
    <sld:Fill>
      <sld:CssParameter name="fill">#FFFFFF</sld:CssParameter>
    </sld:Fill>
  </sld:Halo>
  <sld:Fill>
    <sld:CssParameter name="fill">#000033</sld:CssParameter>
  </sld:Fill>
  <sld:Priority>200000</sld:Priority>
  <sld:VendorOption name="autoWrap">100</sld:VendorOption>
</sld:TextSymbolizer>
```

Highlights about the above style:

- Uses FULLNAME as the label source
- Uses a Arial 12pt font

- Places the label below the point, and offsets it by 10 pixel to the south
- Applies a white halo to make it stand out of the background map
- Sets its priority to 200000 (high, important) to make sure the label is depicted in preference to others
- Uses the `autoWrap` option to make it wrap on the next line if it's larger than 100 pixels (the full list of labelling vendor options is available in the [GeoServer user guide](#)).

Using Rules to assign a different styling to each point

A Rule is a SLD construct allowing the style editor to control scale dependencies and filter data so that only certain data is depicted using the symbolizers contained in the rule.

The rule for the school points looks as follows:

```
<sld:Rule>
  <sld:Name>school</sld:Name>
  <ogc:Filter>
    <ogc:PropertyIsEqualTo>
      <ogc:PropertyName>MTFCC</ogc:PropertyName>
      <ogc:Literal>K2543</ogc:Literal>
    </ogc:PropertyIsEqualTo>
  </ogc:Filter>
  <sld:MaxScaleDenominator>100000</sld:MaxScaleDenominator>
  <sld:PointSymbolizer>
    <!-- same as above -->
  </sld:PointSymbolizer>
  <sld:TextSymbolizer>
    <!-- same as above -->
  </sld:TextSymbolizer>
</sld:Rule>
```

Highlights about the above rule:

- makes sure the symbolizers are applied only to the features whose `MTFCC = K2543`
- shows the symbols only when the scale denominator is below 100000 (e.g., shows them at 1:10000, but not at 1:2000000).

Using dynamic symbolizers to reduce the style size

The overall `point_landmark` style has 8 different rules using different symbols for each type and amounts to almost 550 lines of XML. The same style could be written in a much more compact way if we could store the symbol name in some attribute and expand it in the external graphic URL.

Standard SLD 1.0 does not allow for that, but GeoServer supports extensions to it known as *dynamic symbolizers* that allow for generic CQL expressions to be embedded in the URL. The data directory already contains a secondary layer, `bptlandmarks_2876`, which is using a different projection and has a `IMAGE` attribute containing the file names.

The style can then be reduced to a single rule using the following point symbolizer:

```
<sld:PointSymbolizer>
  <sld:Graphic>
    <sld:ExternalGraphic>
      <sld:OnlineResource xlink:type="simple" xlink:href="./img/landmarks/${IMAGE}" />
      <sld:Format>image/png</sld:Format>
    </sld:ExternalGraphic>
  </sld:Graphic>
</sld:PointSymbolizer>
```

(continues on next page)

(continued from previous page)

```

    </sld:ExternalGraphic>
  </sld:Graphic>
  <VendorOption name="labelObstacle">true</VendorOption>
</sld:PointSymbolizer>

```

Here is the overall style:

```

<?xml version="1.0" encoding="UTF-8"?>
<sld:StyledLayerDescriptor
  xmlns="http://www.opengis.net/sld"
  xmlns:sld="http://www.opengis.net/sld"
  xmlns:ogc="http://www.opengis.net/ogc"
  xmlns:gml="http://www.opengis.net/gml"
  xmlns:xlink="http://www.w3.org/1999/xlink" version="1.0.0">

  <sld:UserLayer>
    <sld:LayerFeatureConstraints>
      <sld:FeatureTypeConstraint/>
    </sld:LayerFeatureConstraints>
    <sld:UserStyle>
      <sld:Name>t1 2010 08013 pointlm</sld:Name>
      <sld:Title/>
      <sld:FeatureTypeStyle>
        <sld:Rule>
          <sld:Name>landmarks</sld:Name>
          <ogc:Filter>
            <ogc:Not>
              <ogc:PropertyIsNull>
                <ogc:PropertyName>IMAGE</ogc:PropertyName>
              </ogc:PropertyIsNull>
            </ogc:Not>
          </ogc:Filter>
          <sld:MaxScaleDenominator>100000</sld:MaxScaleDenominator>
          <sld:PointSymbolizer>
            <sld:Graphic>
              <sld:ExternalGraphic>
                <sld:OnlineResource xlink:type="simple" xlink:href="./img/landmarks/$
→ {IMAGE}" />
              <sld:Format>image/png</sld:Format>
            </sld:ExternalGraphic>
          </sld:Graphic>
          <VendorOption name="labelObstacle">true</VendorOption>
        </sld:PointSymbolizer>
        <sld:TextSymbolizer>
          <sld:Label>
            <ogc:PropertyName>FULLNAME</ogc:PropertyName>
          </sld:Label>
          <sld:Font>
            <sld:CssParameter name="font-family">Arial</sld:CssParameter>
            <sld:CssParameter name="font-size">12.0</sld:CssParameter>
            <sld:CssParameter name="font-style">normal</sld:CssParameter>
            <sld:CssParameter name="font-weight">normal</sld:CssParameter>
          </sld:Font>
          <sld:LabelPlacement>
            <sld:PointPlacement>
              <sld:AnchorPoint>
                <sld:AnchorPointX>

```

(continues on next page)

(continued from previous page)

```

        <ogc:Literal>0.5</ogc:Literal>
      </sld:AnchorPointX>
      <sld:AnchorPointY>
        <ogc:Literal>1.0</ogc:Literal>
      </sld:AnchorPointY>
    </sld:AnchorPoint>
    <sld:Displacement>
      <sld:DisplacementX>
        <ogc:Literal>0.0</ogc:Literal>
      </sld:DisplacementX>
      <sld:DisplacementY>
        <ogc:Literal>-14.0</ogc:Literal>
      </sld:DisplacementY>
    </sld:Displacement>
    <sld:Rotation>
      <ogc:Literal>0.0</ogc:Literal>
    </sld:Rotation>
  </sld:PointPlacement>
</sld:LabelPlacement>
<sld:Halo>
  <sld:Radius>
    <ogc:Literal>1.5</ogc:Literal>
  </sld:Radius>
  <sld:Fill>
    <sld:CssParameter name="fill">#FFFFFF</sld:CssParameter>
  </sld:Fill>
</sld:Halo>
<sld:Fill>
  <sld:CssParameter name="fill">#000033</sld:CssParameter>
</sld:Fill>
<sld:Priority>200000</sld:Priority>
<sld:VendorOption name="autoWrap">100</sld:VendorOption>
</sld:TextSymbolizer>
</sld:Rule>
</sld:FeatureTypeStyle>
</sld:UserStyle>
</sld:UserLayer>
</sld:StyledLayerDescriptor>

```

And [here](#) is a map using this alternate style:

Styling in real world units

By default SLD interprets all sizes expressed in the style sheet (e.g., line widths, symbol sizes) as being pixels on the map.

It is however possible to make the style sheet use real world units, e.g., meters or feet, by specifying the desired unit of measure as an attribute of the symbolizer. The supported unit of measure are:

- meter
- foot
- pixel

The following line style uses a line width of 40 meters:

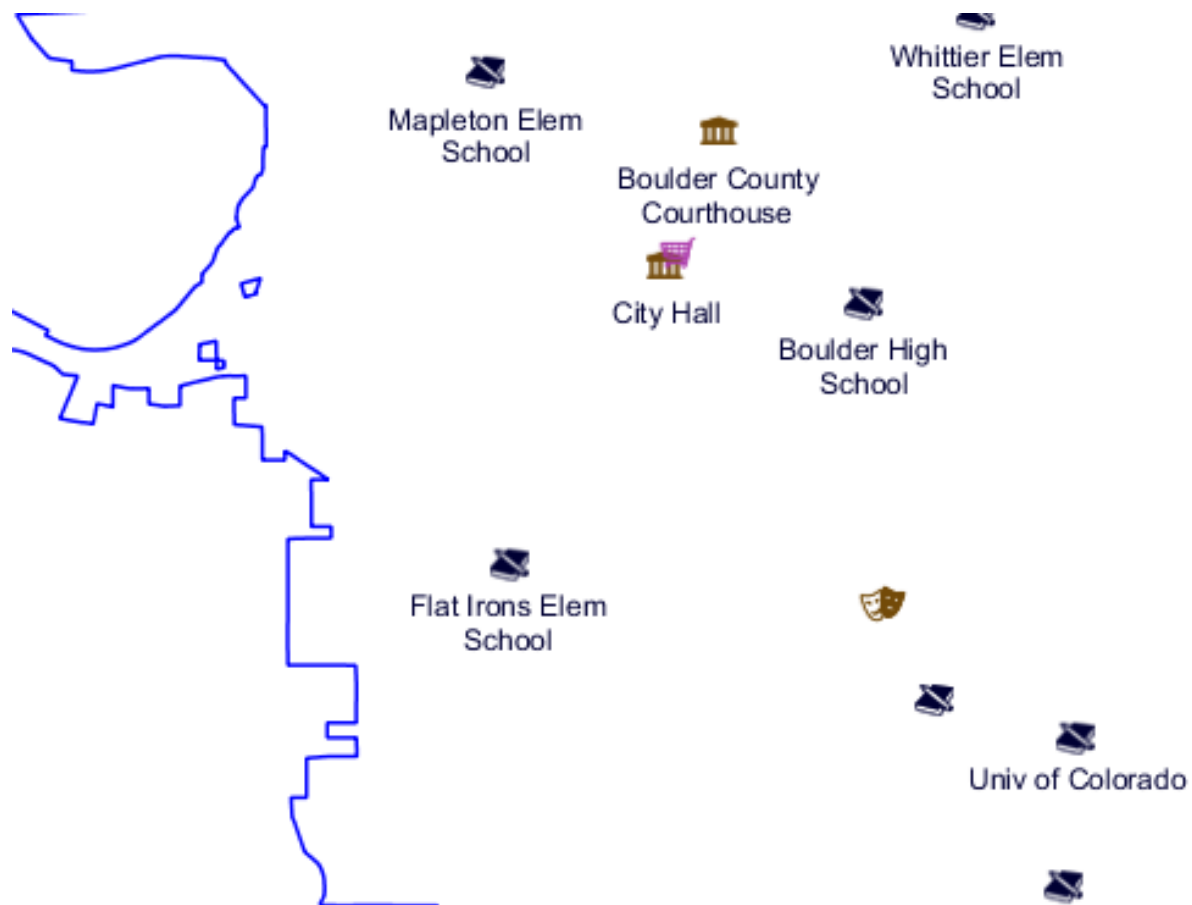


Fig. 179: Point landmarks using dynamic symbolizers

```
<LineSymbolizer uom="http://www.opengeospatial.org/se/units/metre">
  <Stroke>
    <CssParameter name="stroke">#000033</CssParameter>
    <CssParameter name="stroke-width">40</CssParameter>
  </Stroke>
</LineSymbolizer>
```

Setting up a uom based style in GeoServer

1. Create a new style named line40m using the following SLD:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<StyledLayerDescriptor version="1.0.0"
  xsi:schemaLocation="http://www.opengis.net/sld StyledLayerDescriptor.xsd"
  xmlns="http://www.opengis.net/sld"
  xmlns:ogc="http://www.opengis.net/ogc"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <NamedLayer>
    <Name>line40m</Name>
    <UserStyle>
      <Title>40 meter wide line</Title>
      <FeatureTypeStyle>
        <Rule>
          <LineSymbolizer uom="http://www.opengeospatial.org/se/
units/metre">
            <Stroke>
              <CssParameter name="stroke">#000033</
              <CssParameter name="stroke-width">40</
            </Stroke>
          </LineSymbolizer>
        </Rule>
      </FeatureTypeStyle>
    </UserStyle>
  </NamedLayer>
</StyledLayerDescriptor>
```

2. Associate the line40m to MainRd as a secondary style:
3. Preview the MainRd layer and switch to the line40m style:
4. Zoom in and out and observe how the width of the line on screen varies by changing the zoom level

Geometry transformations

This section show how to GeoServer provides a number of filter functions that can actually manipulate geometries by transforming them into something different: this is what we call *geometry transformations in SLD*.

Additional Styles



Fig. 180: Adding the line40m style as a secondary style for Mainrd

Extracting vertices

1. Using skills learned in the *adding styles* section, create a new style named *mainrd_transform* using the following SLD:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<StyledLayerDescriptor version="1.0.0"
  xmlns="http://www.opengis.net/sld" xmlns:ogc="http://www.opengis.net/ogc"
  xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsi="http://www.w3.org/2001/
  →XMLSchema-instance"
  xsi:schemaLocation="http://www.opengis.net/sld http://schemas.opengis.net/sld/1.
  →0.0/StyledLayerDescriptor.xsd">
  <NamedLayer>
    <Name>Roads and vertices</Name>
    <UserStyle>
      <FeatureTypeStyle>
        <Rule>
          <LineSymbolizer>
            <Stroke />
          </LineSymbolizer>
          <PointSymbolizer>
            <Geometry>
              <ogc:Function name="vertices">
                <ogc:PropertyName>the_geom</ogc:PropertyName>
              </ogc:Function>
            </Geometry>
            <Graphic>
              <Mark>
                <WellKnownName>circle</WellKnownName>
                <Fill>
                  <CssParameter name="fill">#FF0000</CssParameter>
                </Fill>
              </Mark>
              <Size>6</Size>
            </Graphic>
          </PointSymbolizer>
        </Rule>
      </FeatureTypeStyle>
    </UserStyle>
```

(continues on next page)

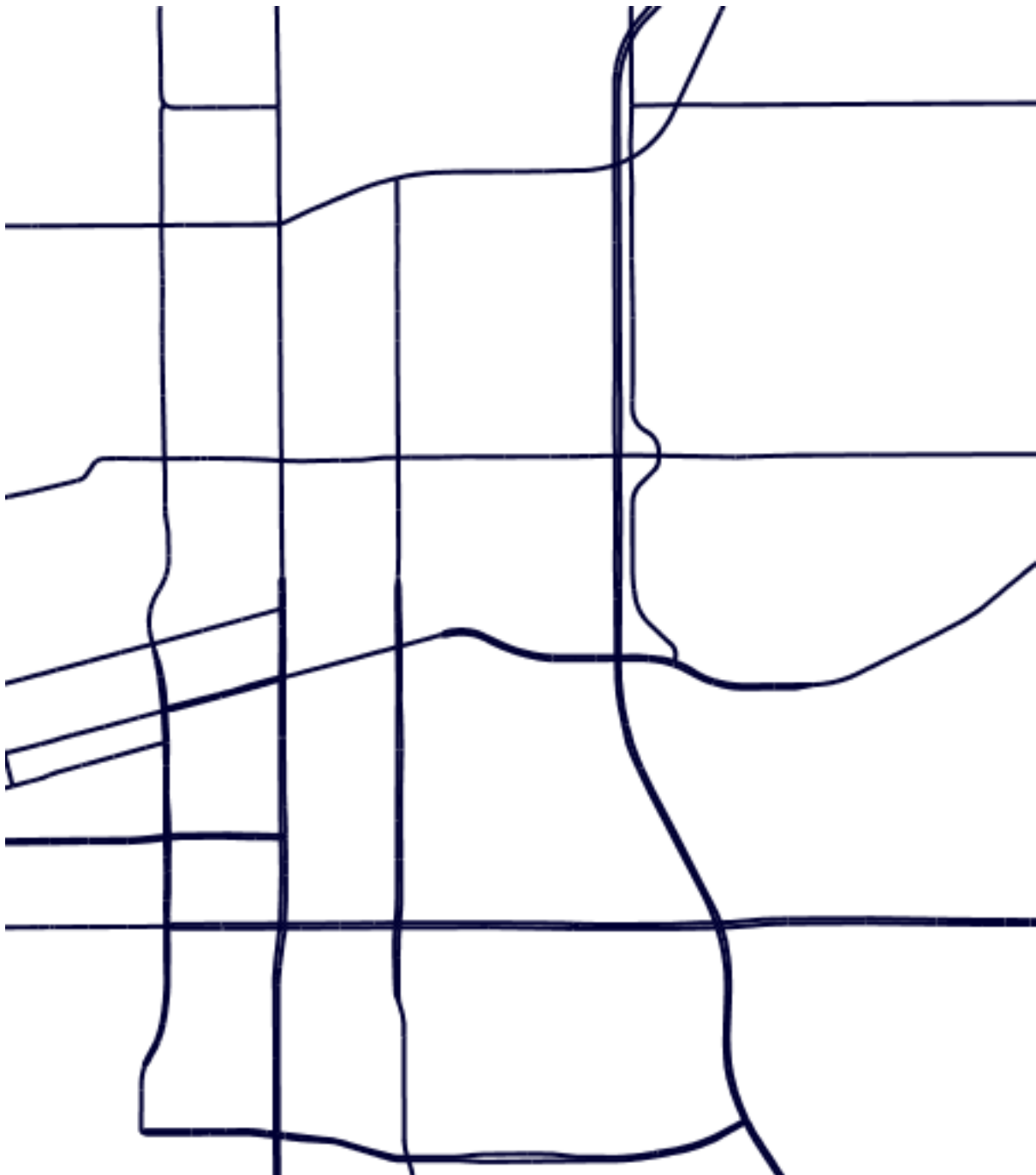


Fig. 181: A uom based line, zoomed out

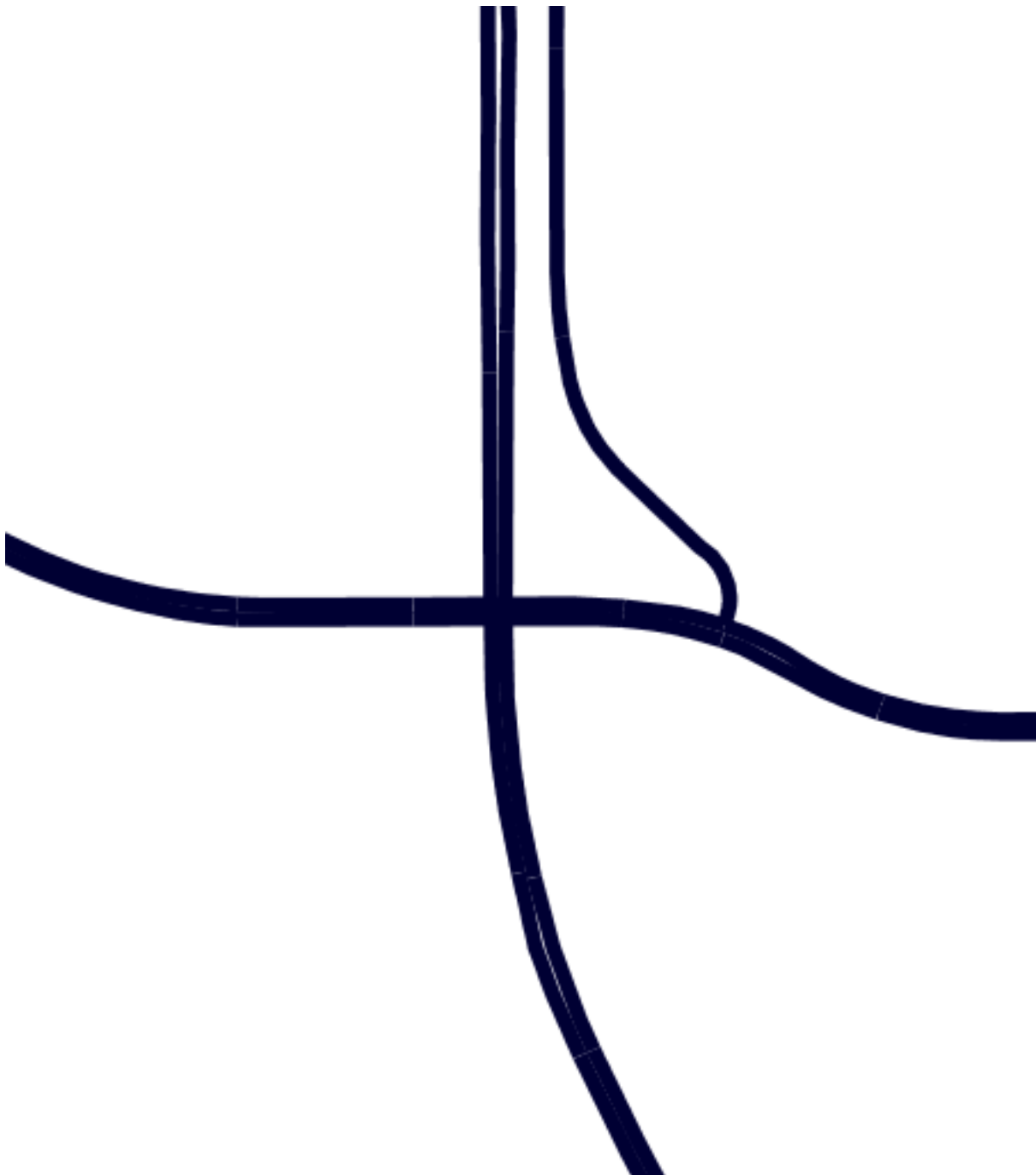


Fig. 182: Zooming in on the same line

(continued from previous page)

```
</NamedLayer>
</StyledLayerDescriptor>
```

Note: The `vertices` function returns a multi-point made with all the vertices of the original geometry

- Using skills learned in the [adding styles](#) section, modify the styling of the `Mainrd` layer and add `mainrd_transform` as an alternate style (hint, select the `mainrd_transform` style in the first list below “available styles” and then use the right arrow to move it in the “selected styles”):

WMS Settings

☒ Queryable

Default Style

mainrd



Additional Styles



Fig. 183: Adding the `mainrd_transform` style as a secondary style for `Mainrd`

- Use the Preview link to display the `Mainrd` layer, then open the options box and choose the alternate style from the drop down:

Line buffer

- Using skills learned in the [geoserver.addstyle](#) section, create a new style `mainrd_buffer` using the following SLD

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<StyledLayerDescriptor version="1.0.0"
  xmlns="http://www.opengis.net/sld" xmlns:ogc="http://www.opengis.net/ogc"
  xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsi="http://www.w3.org/
  ↪2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.opengis.net/sld http://schemas.opengis.
  ↪net/sld/1.0.0/StyledLayerDescriptor.xsd">
  <NamedLayer>
    <Name>Roads and vertices</Name>
    <UserStyle>
```

(continues on next page)

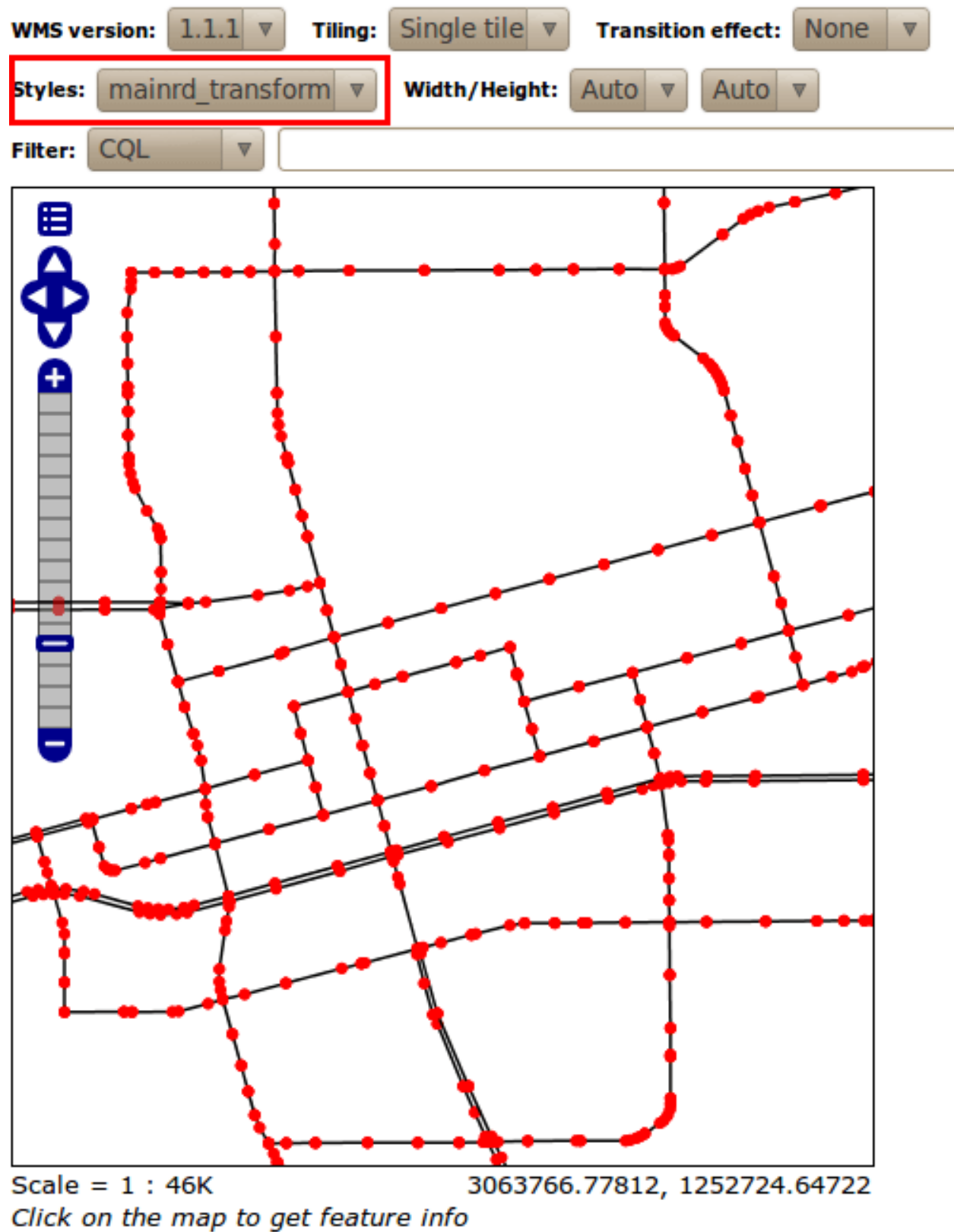


Fig. 184: Extracting and showing the vertices out of a geometry

(continued from previous page)

```

<FeatureTypeStyle>
  <Rule>
    <PolygonSymbolizer>
      <Geometry>
        <ogc:Function name="buffer">
          <ogc:PropertyName>the_geom</
→ogc:PropertyName>
          <ogc:Literal>200</ogc:Literal>
        </ogc:Function>
      </Geometry>
      <Fill>
        <CssParameter name="fill">#7F7F7F</
→CssParameter>
        <CssParameter name="fill-opacity">0.3</
→CssParameter>
      </Fill>
    </PolygonSymbolizer>
    <LineSymbolizer>
      <Stroke />
    </LineSymbolizer>
  </Rule>
</FeatureTypeStyle>
</UserStyle>
</NamedLayer>
</StyledLayerDescriptor>

```

Note: The `buffer` function builds a polygon of all the points that are within the specified distance from the original geometry.

2. As done previously, modify the styling of the `Mainrd` layer and add `mainrd_buffer` as an alternate style:

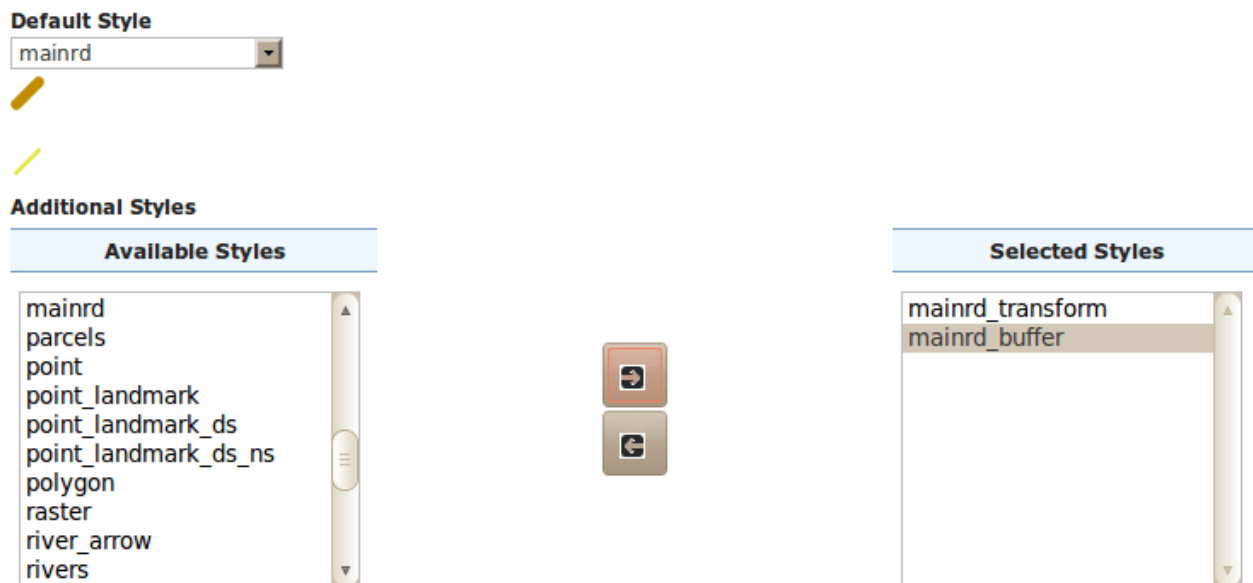


Fig. 185: Adding the `mainrd_buffer` style as a secondary style for `Mainrd`

1. Use the [Map Preview](#) to preview the new style.

Charting

GeoServer can produce maps with charts through the chart extension. Bundled with GeoServer is an open source version of the (deprecated) [Google Chart API](#) called [Eastwood Charts](#).

You can display bar or pie charts (Most Google Charts except for Google-o-meter and spider charts are supported by the Eastwood library but the same does not apply to the corresponding GeoServer extension) for each feature on your map. You can control colors or labels. You can use percentages that are in your data attributes or compute percentages from counts on the fly.

How Charting Works

The Charting Extension makes usage of a URL inside the `<ExternalGraphic>` element of SLD documents. The URL used follows the Google Chart API syntax, but the chart is generated internally in GeoServer, hence no call to external services made removing any privacy or security concern and providing maximum performance. All the information about the chart that you want, such as chart data, size, colors, and labels, are part of the URL.

Inside the URL we can use variable substitution for using the attributes of the underlying features that are read from the datasource allowing us to create stunning dynamic charts using our own data.

An example of a chart created using an `<ExternalGraphic>` element is shown here below:

```
<ExternalGraphic>
  <OnlineResource
    xlink:href="http://chart?cht=p&chd=t:${100 * MALE / PERSONS},${100 *
↪FEMALE / PERSONS}&chf=bg,s,FFFFFF00" />
  <Format>application/chart</Format>
</ExternalGraphic>
```

All URLs start with <https://chart?> followed by the parameters that specify chart data and appearance. Parameters are name=value pairs, separated by an ampersand character (&), and parameters can be in any order, after the ?. All charts require at minimum the following parameters: *cht* (chart type), *chd* (data), and *chs* (chart size). However, there are many more parameters for additional options, and you can specify as many additional parameters as the chart supports.

We are now going to see examples and explanation for the various types of charts supported. First of all we will start with the standard features support by all the charts.

Standard Features

All Chart URLs have the following format:

```
https://chart?cht=<chart_type>&chd=<chart_data>&chs=<chart_size>&...more_
↪parameters...
```

The standard parameters as part of the above URL have the following meaning:

- The *cht* parameter allows us to control the type of charts; as an example *cht=p* can be used for a 2D (flat) Pie.
- The *chs* parameter allows us to control the size of charts; as an example *chs=500x200* specifies the chart size (width x height), in pixels. As an alternative we can use the `<Size>` element of external graphics (we'll show an example in the following).



Fig. 186: Extracting start and end point of a line

- The *chd* parameter allows us to control the chart data; as an example *chd=t:60,40* can be used to provide tabular data to the diagram rendering system. We can use variable substitution and other GeoServer mechanisms to pass data sources value as the chart data. A typical example would be something like *chd=t:\${100 * MALE / PERSONS},\${100 * FEMALE / PERSONS}&* where *MALE*, *PERSONS* and *FEMALE* are attribute of GeoServer data sources.
- The *chl* parameter allows us to control the label of charts; as an example *chl=Male|Female* can be used to label a chart.

Pie Charts

Quoting Wikipedia,

“A pie chart (or a circle graph) is a circular chart divided into sectors, illustrating numerical proportion. In a pie chart, the arc length of each sector (and consequently its central angle and area), is proportional to the quantity it represents.”

Let us now create a sample map using the Pie Charts element leveraging on the data provided with the training. Afterwards we will review the various options.

To print dynamic charts on a map using a Pie symbol over the United States map add a new style called *statespies* by adding the SLD provided below as indicated in this picture.

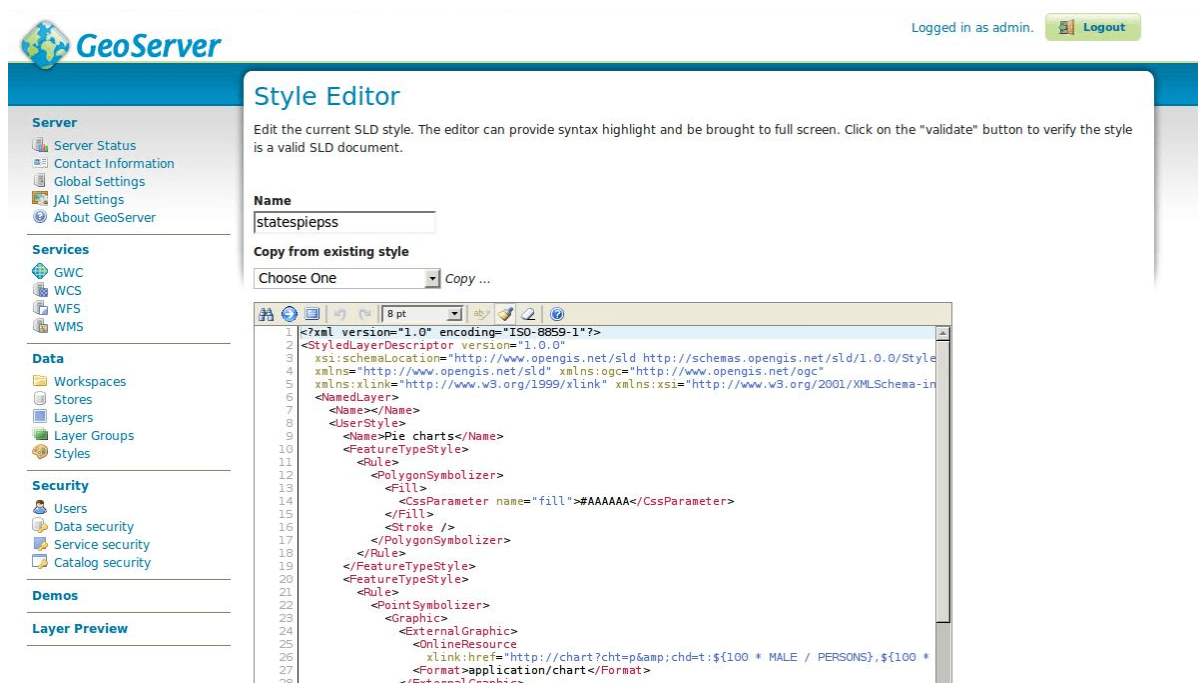


Fig. 187: Creating a new Dynamic Style

In the *SLD Editor* enter the following XML:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<StyledLayerDescriptor version="1.0.0"
  xsi:schemaLocation="http://www.opengis.net/sld http://schemas.opengis.net/
  sld/1.0.0/StyledLayerDescriptor.xsd"
  xmlns="http://www.opengis.net/sld" xmlns:ogc="http://www.opengis.net/ogc"
```

(continues on next page)

(continued from previous page)

```

xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsi="http://www.w3.org/
↪2001/XMLSchema-instance">
  <NamedLayer>
    <Name></Name>
    <UserStyle>
      <Name>Pie charts</Name>
      <FeatureTypeStyle>
        <Rule>
          <PolygonSymbolizer>
            <Fill>
              <CssParameter name="fill">#AAAAAA</CssParameter>
            </Fill>
            <Stroke />
          </PolygonSymbolizer>
        </Rule>
      </FeatureTypeStyle>
      <FeatureTypeStyle>
        <Rule>
          <PointSymbolizer>
            <Geometry>
              <ogc:Function name="centroid">
                <ogc:PropertyName>the_geom</ogc:PropertyName>
              </ogc:Function>
            </Geometry>
            <Graphic>
              <ExternalGraphic>
                <OnlineResource
                  xlink:href="http://chart?cht=p&chd=t:${100 * MALE / ↪
↪PERSONS},${100 * FEMALE / PERSONS}&chf=bg,s,FFFFFF00" />
                <Format>application/chart</Format>
              </ExternalGraphic>
              <Size>
                <ogc:Add>
                  <ogc:Literal>20</ogc:Literal>
                  <ogc:Mul>
                    <ogc:Div>
                      <ogc:PropertyName>PERSONS</ogc:PropertyName>
                      <ogc:Literal>20000000.0</ogc:Literal>
                    </ogc:Div>
                    <ogc:Literal>60</ogc:Literal>
                  </ogc:Mul>
                </ogc:Add>
              </Size>
            </Graphic>
          </PointSymbolizer>
        </Rule>
      </FeatureTypeStyle>
    </UserStyle>
  </NamedLayer>
</StyledLayerDescriptor>

```

In order to have the *states* layer use this style with no additional indications, modify the default style of the *states* layer using the user interface to point to the newly created *statespies*.

Now go to the **Layer Preview** to view the new style in action.

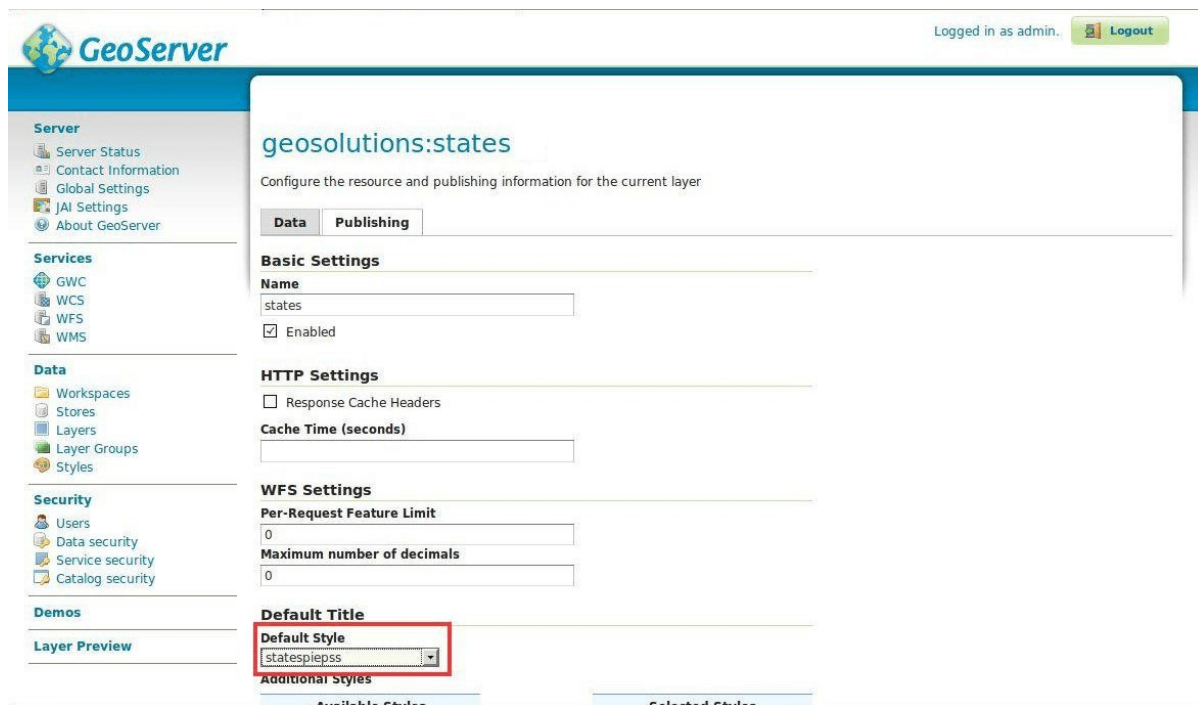
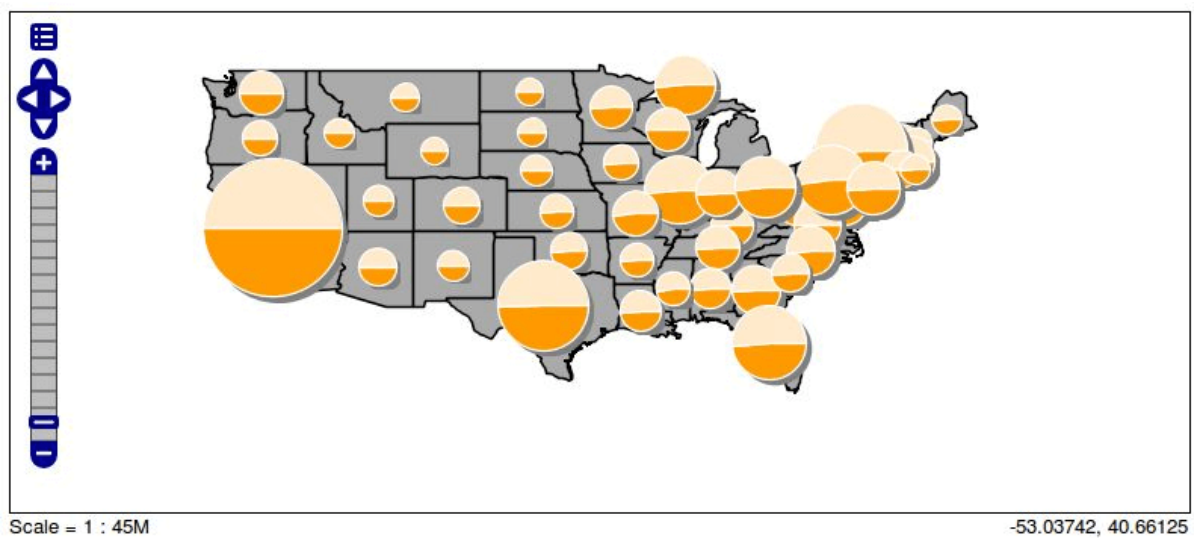


Fig. 188: Changing the default style of the states layer

Fig. 189: Previewing the states layer with the *statespies* applied

Pie Chart Options

Let us quickly analyse the components of the ExternalGraphic call, which follow the rules of a Google Charts API call:

Pie Chart Types

The *cht* parameter allows us to control the type of pie. Supported options are as follows:

- *cht=p* for a 2D (flat) Pie
- *cht=p3* for a 3D (flat) Pie
- *cht=pc* **is not supported**.

Pie Chart Data

`chd=t:${100 * MALE / PERSONS},${100 * FEMALE / PERSONS}` the chart data is expressed in “text” format, and in particular, the first value is the result of $100 * \text{MALE} / \text{PERSONS}$, where MALE and PERSONS are two attributes of feature being rendered

Pie Chart Background

`chf=bg,s,FFFFFF00`: we state that the chart background fill is solid, white and transparent. In particular, the color is expressed as RRGGBBAA, where AA is the alpha component, which controls transparency. In particular 0 is fully transparent, 255 is fully opaque

Pie Chart Size

The size of the chart is controlled using the usual `<Size>` element of external graphics, and in particular, it's setup so that it's proportional to the PERSONS attribute via the expression: $20 + (\text{PERSONS} / 20,000,000) * 60$.

Pie Chart Colors

We can specify the colors of all values, each value, or some values using the *chco* parameter. This overrides the usage of the default Background Fills *chf* parameter, hence it is optional.

Syntax is as follows:

```
chco=<color_slice_1>,<color_slice_2>
```

for specifying individual colors for slices and

```
chco=<color_1>|<color_2>
```

for specifying a gradient to be applied to the slices.

where *color* is in [RRGGBB hexadecimal format](#).

Pie Chart Labels

We can specify labels for individual pie chart slices using the *chl* parameter.

The syntax is as follows:

```
chl=<label_value>| ... |<label_value>
```

Pie Chart Rotation

Pie Chart Rotation can be achieved via the *chp* parameter. By default, the first series is drawn starting at 3:00, continuing clockwise around the chart.

The syntax is as follows:

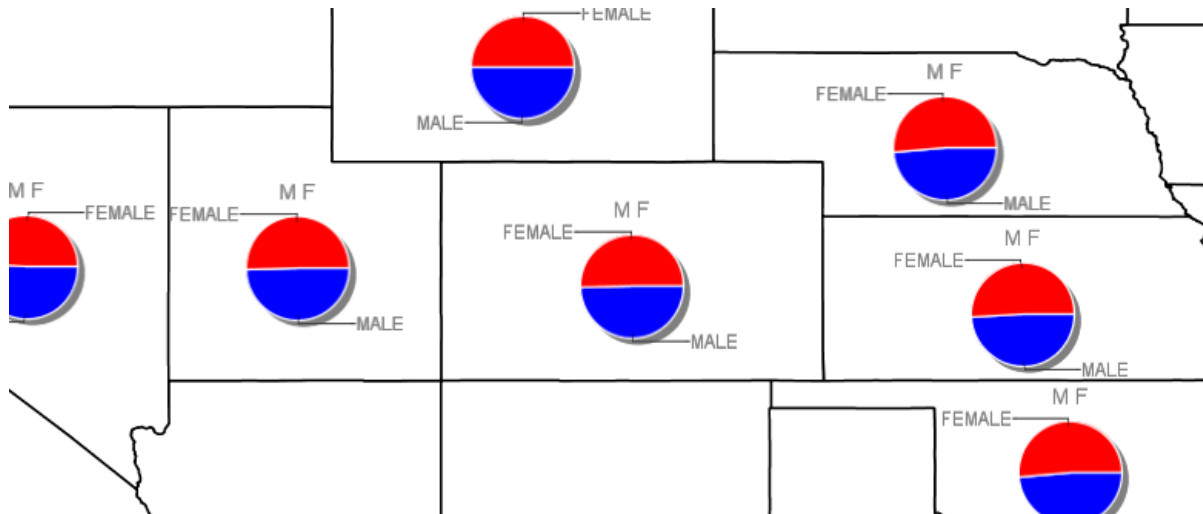
```
chp=<radians>
```

Additional information on creating pie charts can be found [on the official pie charts documentation](#)

A more comprehensive example can be found here below:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<StyledLayerDescriptor version="1.0.0"
  xsi:schemaLocation="http://www.opengis.net/sld http://schemas.opengis.net/
↳sld/1.0.0/StyledLayerDescriptor.xsd"
  xmlns="http://www.opengis.net/sld" xmlns:ogc="http://www.opengis.net/ogc"
  xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsi="http://www.w3.org/
↳2001/XMLSchema-instance">
  <NamedLayer>
    <Name></Name>
    <UserStyle>
      <Name>Pie charts</Name>
      <FeatureTypeStyle>
        <Rule>
          <PolygonSymbolizer>
            <Fill>
              <CssParameter name="fill">#ffffff</CssParameter>
            </Fill>
            <Stroke />
          </PolygonSymbolizer>
        </Rule>
      </FeatureTypeStyle>
      <FeatureTypeStyle>
        <Rule>
          <PointSymbolizer>
            <Geometry>
              <ogc:Function name="centroid">
                <ogc:PropertyName>the_geom</ogc:PropertyName>
              </ogc:Function>
            </Geometry>
            <Graphic>
              <ExternalGraphic>
                <OnlineResource
                  xlink:href="http://chart?cht=p&chf=bg,s,FFFFFF00&
↳chd=t:${100 * MALE / PERSONS},${100 * FEMALE / PERSONS}&
↳chl=MALE|FEMALE&chs=200x100&chco=0000ff,ff0000&chtt=M+F" />
                <Format>application/chart</Format>
              </ExternalGraphic>
            </Graphic>
          </PointSymbolizer>
        </Rule>
      </FeatureTypeStyle>
    </UserStyle>
  </NamedLayer>
</StyledLayerDescriptor>
```

The resulting image can be found here below:



Bar Charts

Quoting Wikipedia,

“A bar chart or bar graph is a chart with rectangular bars with lengths proportional to the values that they represent. The bars can be plotted vertically or horizontally. A vertical bar chart is sometimes called a column bar chart.”

Let us now create a sample map using the Bar Charts element leveraging on the data provided with the training. Afterwards we will review the various options.

To print dynamic charts on a map using a Bar symbol over the United States map add a new style called *statesbars* by adding the SLD provided below as indicated in this picture.

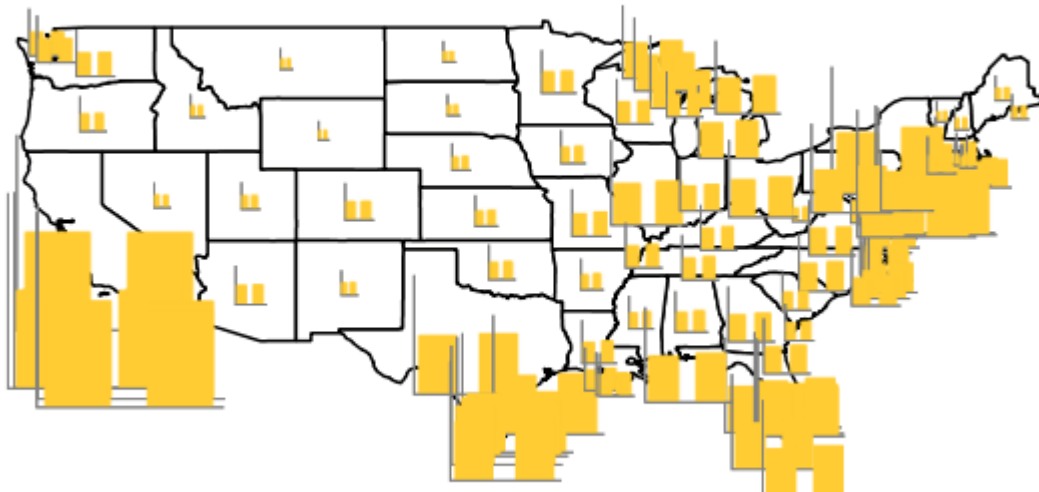


Fig. 190: Creating a new Dynamic Style with Bar Charts

In the *SLD Editor* enter the following XML:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<StyledLayerDescriptor version="1.0.0"
  xsi:schemaLocation="http://www.opengis.net/sld http://schemas.opengis.net/
  ↪sld/1.0.0/StyledLayerDescriptor.xsd"
  xmlns="http://www.opengis.net/sld" xmlns:ogc="http://www.opengis.net/ogc"
  xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsi="http://www.w3.org/
  ↪2001/XMLSchema-instance">
  <NamedLayer>
    <Name></Name>
    <UserStyle>
      <Name>Pie charts</Name>
      <FeatureTypeStyle>
        <Rule>
          <PolygonSymbolizer>
            <Fill>
              <CssParameter name="fill">#ffffff</CssParameter>
            </Fill>
            <Stroke />
          </PolygonSymbolizer>
        </Rule>
      </FeatureTypeStyle>
      <FeatureTypeStyle>
        <Rule>
          <PointSymbolizer>
            <Graphic>
              <Geometry>
                <ogc:Function name="centroid">
                  <ogc:PropertyName>the_geom</ogc:PropertyName>
                </ogc:Function>
              </Geometry>
              <ExternalGraphic>
                <OnlineResource
                  xlink:href="http://chart?cht=bvg&chf=bg,s,FFFFFF00&
  ↪chd=t:${100 * MALE / PERSONS},${100 * FEMALE / PERSONS}" />
                <Format>application/chart</Format>
              </ExternalGraphic>
              <Size>
                <ogc:Add>
                  <ogc:Literal>20</ogc:Literal>
                  <ogc:Mul>
                    <ogc:Div>
                      <ogc:PropertyName>PERSONS</ogc:PropertyName>
                      <ogc:Literal>20000000.0</ogc:Literal>
                    </ogc:Div>
                    <ogc:Literal>60</ogc:Literal>
                  </ogc:Mul>
                </ogc:Add>
              </Size>
            </Graphic>
          </PointSymbolizer>
        </Rule>
      </FeatureTypeStyle>
    </UserStyle>
  </NamedLayer>
</StyledLayerDescriptor>
```

Bar Chart Options

Let us quickly analyse the components of the ExternalGraphic call, which follow the rules of a Google Charts API call:

Bar Chart Types

The *cht* parameter allows us to control the type of pie. Supported options are as follows:

- *cht=bvg* for simple 2D vertical Bars layed out as groups.
- *cht=bhg* for simple 2D horizontal Bars layed out as groups.
- *cht=bvs* for simple 2D vertical Bars layed out as stacks.
- *cht=bvo* **is not supported**.

Bar Chart Data

`chd=t:${100 * MALE / PERSONS},${100 * FEMALE / PERSONS}` the chart data is expressed in “text” format, and in particular, the first value is the result of `100 * MALE / PERSONS`, where MALE and PERSONS are two attributes of feature being rendered. This type of sequence is good for grouped bar charts. Values for successive groups are separated by `|`. Values within the same group are separated by comma.

Bar Chart Colors

Note: Note that by default, all series are displayed in the same color; if you don’t specify different colors for different series, it will be hard to distinguish that there are multiple series in your chart.

You can specify the colors of individual bars, individual series, or multiple series using the *chco* parameter. If you don’t specify a different color for each series, all series will be the same color. Syntax is as follows:

```
chco=<series_1_color>, ..., <series_n_color>
```

or

```
chco=<series_1_bar_1>|<series_1_bar_2>|...|<series_1_bar_n>,<series_2>,...,
↪<series_n>
```

where *color* is in **RRGGBB** hexadecimal format.

Bar Chart Background

`chf=bg,s,FFFFFF00`: we state that the chart background fill is solid, white and transparent. In particular, the color is expressed as **RRGGBBAA**, where AA is the alpha component, which controls transparency. In particular 0 is fully transparent, 255 is fully opaque.

Bar Chart Size

The size of the chart is controlled using the usual `<Size>` element of external graphics, and in particular, it’s setup so that it’s proportional to the PERSONS attribute via the expression: `20 + (PERSONS / 20,000,000) * 60`.

Bar Chart Labels

Bar charts support standard axis labels, but labels along the base of the bars are assigned to individual bars, rather than spread out along the bar chart. (To spread out labels evenly, use the *chxp* parameter as described below.) If you specify axis labels but don’t specify custom labels along the bar axis, the bar labels will be the index number of each bar or group of bars. You can customize axis labels using the *chxl* parameter.

The syntax is as follows:

```
chl=<label_value>| ... |<label_value>
```

Additional information on creating pie charts can be found on the [official bar chart documentation](#)

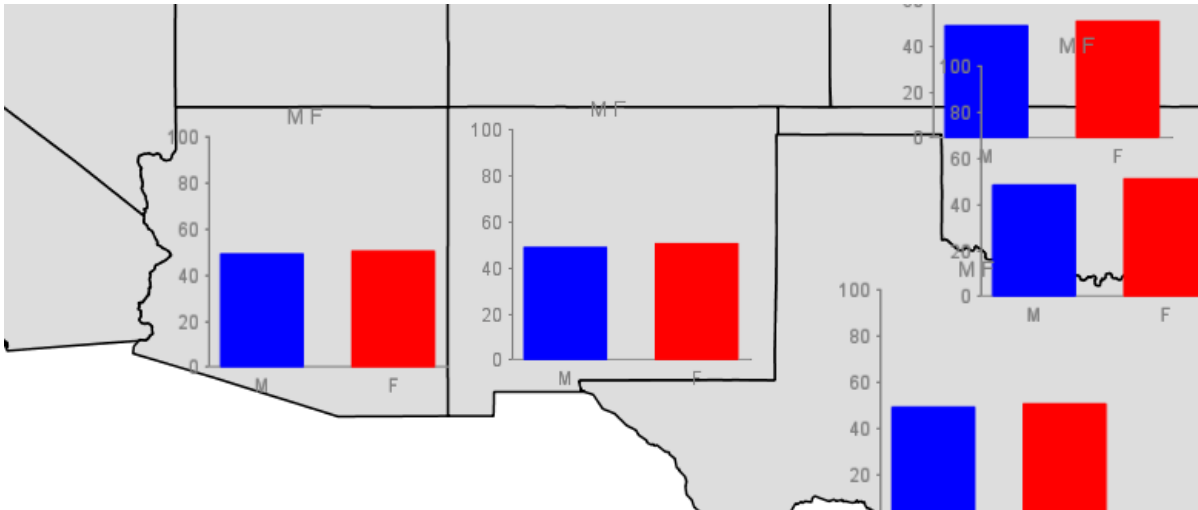
A more comprehensive example can be found here below:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<StyledLayerDescriptor version="1.0.0"
  xsi:schemaLocation="http://www.opengis.net/sld http://schemas.opengis.net/
  ↪sld/1.0.0/StyledLayerDescriptor.xsd"
  xmlns="http://www.opengis.net/sld" xmlns:ogc="http://www.opengis.net/ogc"
  xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsi="http://www.w3.org/
  ↪2001/XMLSchema-instance">
  <NamedLayer>
    <Name></Name>
    <UserStyle>
      <Name>Pie charts</Name>
      <FeatureTypeStyle>
        <Rule>
          <PolygonSymbolizer>
            <Fill>
              <CssParameter name="fill">#dddddd</CssParameter>
            </Fill>
            <Stroke />
          </PolygonSymbolizer>
        </Rule>
      </FeatureTypeStyle>
      <FeatureTypeStyle>
        <Rule>
          <PointSymbolizer>
            <Graphic>
              <ExternalGraphic>
                <OnlineResource
                  xlink:href="http://chart?chxt=x,y&chl=0:M|F&
  ↪amp;cht=bvg&chco=0000ff,ff0000&chf=bg,s,FFFFFF00&chd=t:${100 *
  ↪MALE / PERSONS}|${100 * FEMALE / PERSONS}&chs=200x200&chtt=M+F" />
                <Format>application/chart</Format>
              </ExternalGraphic>
            </Graphic>
          </PointSymbolizer>
        </Rule>
      </FeatureTypeStyle>
    </UserStyle>
  </NamedLayer>
</StyledLayerDescriptor>
```

The resulting image can be found here below:

Styling Raster data

In the previous section we have created and optimized some vector styles. In this section we will deal with a styled SRTM raster and we will see how to get a better visualization of that layer by adding hillshade.



1. From the [Welcome Page](#) navigate to *Layer Preview* and select the [OpenLayers](#) link for the `geosolutions:srtm` layer.

There is a DEM style associated to that SRTM dataset layer, resulting in such a colored rendering.

2. Return to the GeoServer *Welcome Page*, select the *Styles* and click the `dem` style to see which color map is applied.

Note: You have to be logged in as Administrator in order to edit/check styles.

Note the entries with `opacity = 0.0` which allow to make no data values as transparent.

The current DEM style allows to get a pleasant rendering of the SRTM dataset but we can get better results by combining it with an hillshade layer which will be created through another GDAL utility (`gdaldem`).

Adding hillshade

1. Open a shell and run:

```
* Linux

gdaldem hillshade -z 5 -s 111120 ${TRAINING_ROOT}/geoserver_data/data/boulder/
↪srtm_boulder.tiff ${TRAINING_ROOT}/geoserver_data/data/boulder/srtm_boulder_hs.
↪tiff -co tiled=yes

* Windows

gdaldem hillshade -z 5 -s 111120 %TRAINING_ROOT%\geoserver_data\data\boulder\srtm_
↪boulder.tiff %TRAINING_ROOT%\geoserver_data\data\boulder\srtm_boulder_hs.tiff -
↪co tiled=yes
```

Note: The `z` parameter exaggerates the elevation, the `s` parameter provides the ratio between the elevation units and the ground units (degrees in this case), `-co tiled=yes` makes `gdaldem` generate a TIFF with inner tiling. We'll investigate this last option better in the following pages.

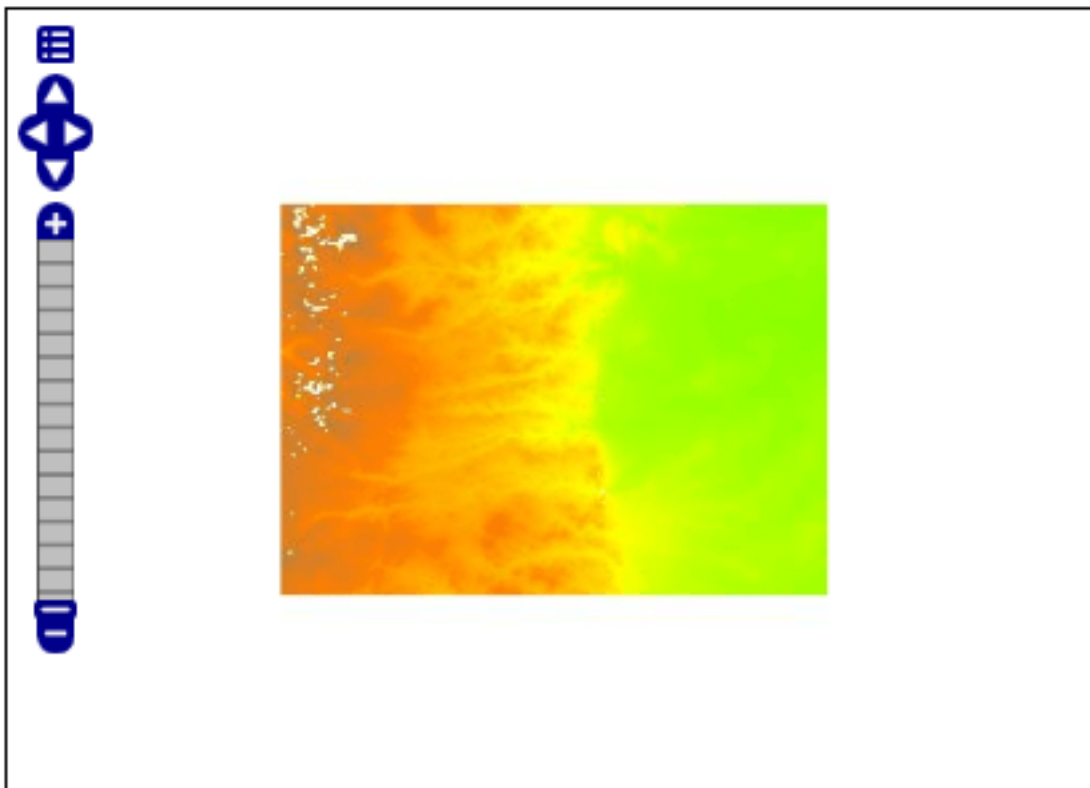


Fig. 191: SRTM rendering with DEM style

2. From the [Welcome Page](#) navigate to *Styles* and select *Add a new style* as previously seen in the [Adding a style](#) section.
3. In the *SLD Editor* enter the following XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<sld:StyledLayerDescriptor xmlns="http://www.opengis.net/sld" xmlns:sld="http://
↪www.opengis.net/sld" xmlns:ogc="http://www.opengis.net/ogc" xmlns:gml="http://
↪www.opengis.net/gml" version="1.0.0">
  <sld:UserLayer>
    <sld:LayerFeatureConstraints>
      <sld:FeatureTypeConstraint/>
    </sld:LayerFeatureConstraints>
    <sld:UserStyle>
      <sld:Title/>
      <sld:FeatureTypeStyle>
        <sld:Name>name</sld:Name>
        <sld:FeatureTypeName>Feature</sld:FeatureTypeName>
        <sld:Rule>
          <sld:MinScaleDenominator>75000</sld:MinScaleDenominator>
          <sld:RasterSymbolizer>
            <sld:Geometry>
              <ogc:PropertyName>grid</ogc:PropertyName>
            </sld:Geometry>
            <sld:ColorMap>
              <sld:ColorMapEntry color="#000000" opacity="0.0"
↪quantity="0.0"/>
            </sld:ColorMap>
          </sld:RasterSymbolizer>
        </sld:Rule>
      </sld:FeatureTypeStyle>
    </sld:UserStyle>
  </sld:UserLayer>
</sld:StyledLayerDescriptor>
```

(continues on next page)

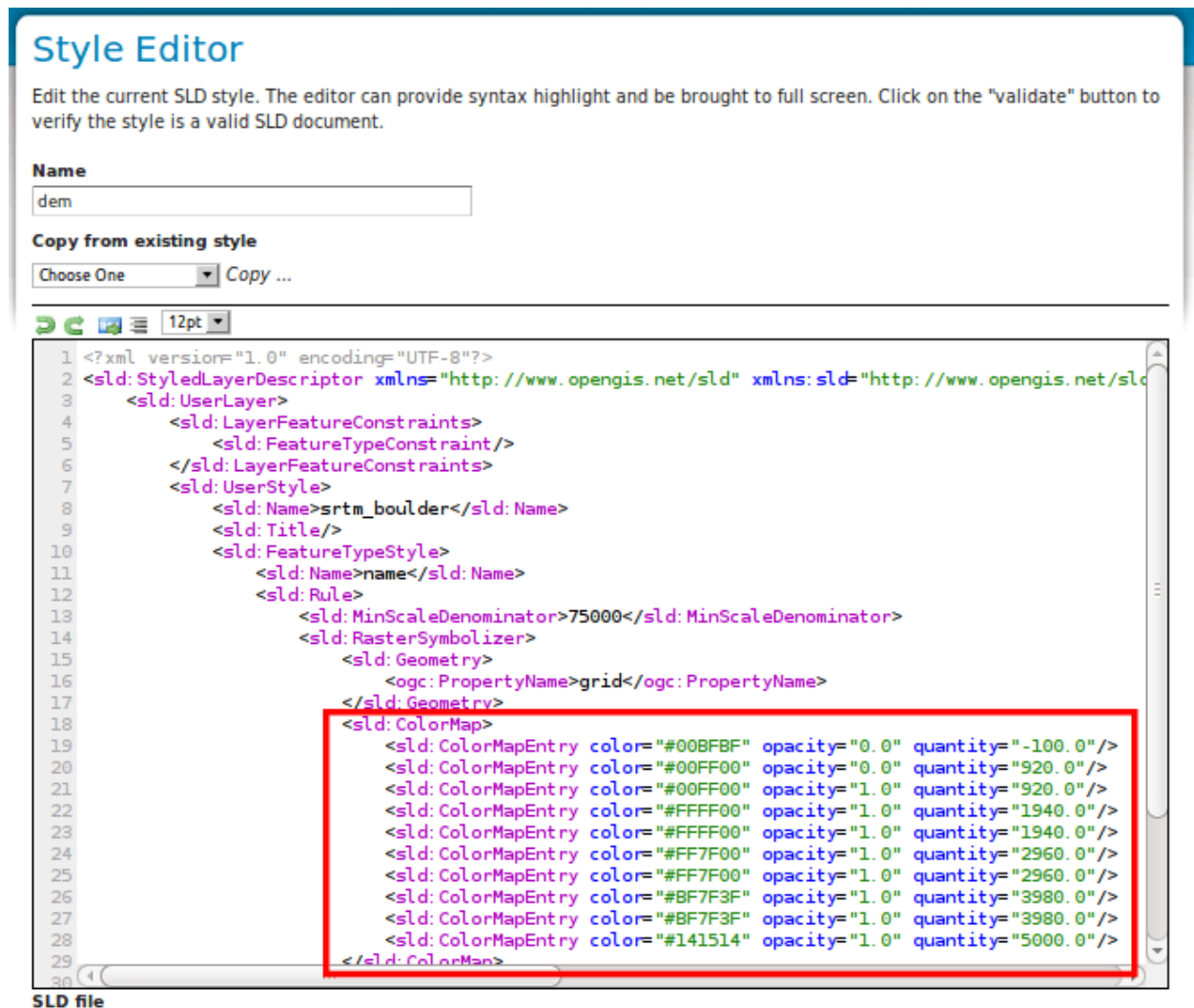


Fig. 192: Style editing

(continued from previous page)

```

<quantity="1.0"/>
<quantity="256.0"/>
    <sld:ColorMapEntry color="#999999" opacity="0.7"
    <sld:ColorMapEntry color="#FFFFFF" opacity="0.7"
    </sld:ColorMap>
    </sld:RasterSymbolizer>
    </sld:Rule>
    </sld:FeatureTypeStyle>
    </sld:UserStyle>
    </sld:UserLayer>
</sld:StyledLayerDescriptor>

```

Note: Note the opacity values being less than 1, in order to make it partially transparent which will allow to do overlaying on other layers

4. Set `hillshade` as name and then click the *Submit* button.
5. Select *Add stores* from the *GeoServer Welcome Page* to add the previously created `hillshade` raster.
6. Select *GeoTIFF - Tagged Image File Format with Geographic information* from the set of available Raster Data Sources.
7. Specify `hillshade` as name in the *Data Source Name* field of the interface.
8. Click on *browse* link in order to set the GeoTIFF location in the *URL* field.

Note: make sure to specify the `srtm_boulder_hs.tiff` previously created with `gdaldem`, which should be located at `$TRAINING_ROOT/geoserver_data/data/boulder`

9. Click *Save*.
10. Publish the layer by clicking on the *publish* link.

New Layer

Add a new layer

Here is a list of resources contained in the store 'hillshade'. Click on the layer you wish to configure

<< < 1 > >> Results 1 to 1 (out of 1 items)					Search
Published	Layer name				action
	hillshade				Publish

Fig. 193: Publishing Raster Layer

11. Set `SRTM Hillshade` as Title
12. Switch to *Publishing* tab
13. Make sure to set the default style to `hillshade` on the *Publishing* → *Default Style* section.
14. Click *Save* to create the new layer.
15. Use the **Layer Preview** to preview the new layer with the `hillshade` style.
16. Edit the Layer Preview URL in your browser by locating the `layers` parameter

Edit Layer

Edit layer data and publishing

geosolutions:hillshade

Configure the resource and publishing information for the current layer

Data **Publishing**

Edit Layer

Name

hillshade

☒ Enabled

☒ Advertised

WMS Settings

☒ Queryable

Default Style

hillshade

☒ 0.0 > x

☒ 1.0 = x

☒ 256.0 = x

Fig. 194: Editing Raster Publishing info

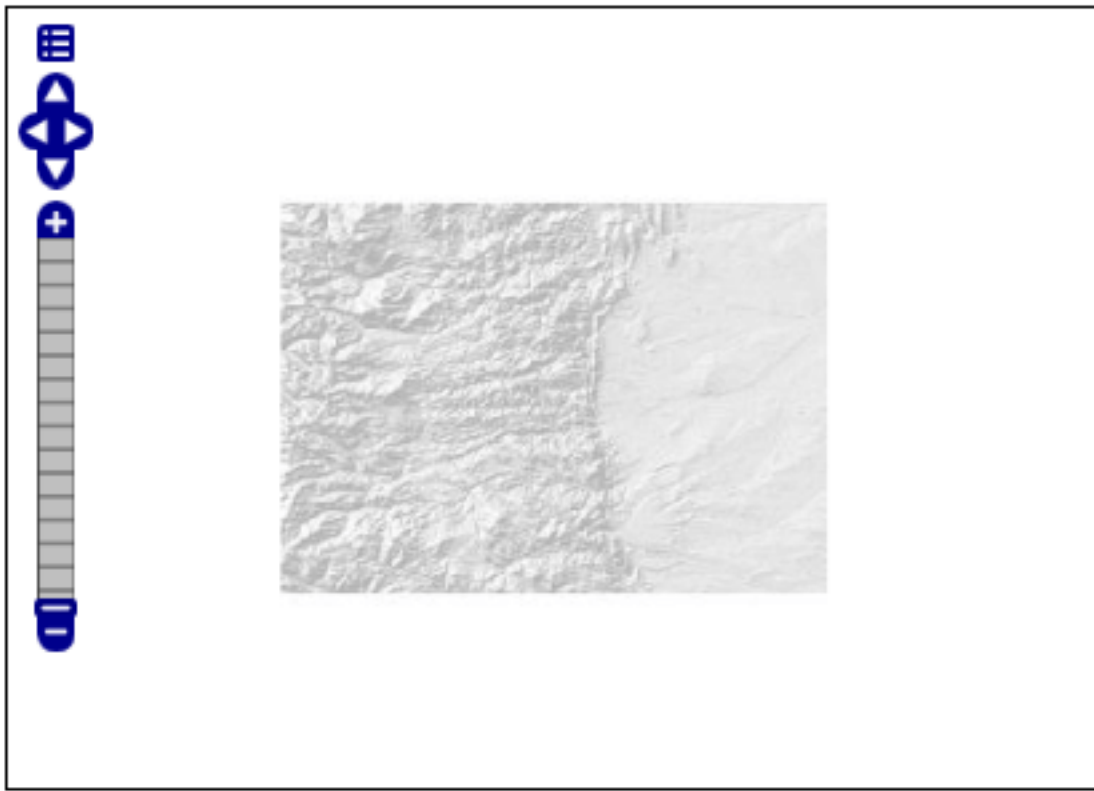
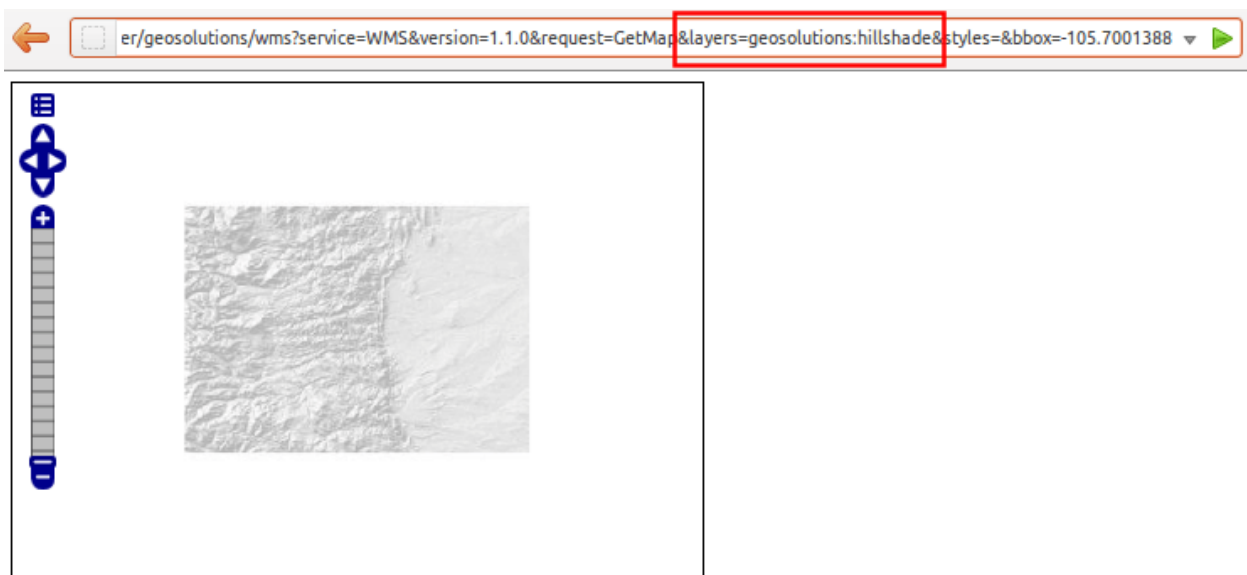


Fig. 195: Previewing the new raster layer with the hillshade style applied



17. Insert the *geosolutions:srtm*, additional layer (note the final comma) before the *geosolutions:hillshade* one, and in the *styles* parameter, add a comma before *hillshade* to make GeoServer use the default style for the srtm layer

```
GetMap&layers=geosolutions:srtm,geosolutions:srtm_boulder_hs&styles=,hillshade&bbox=-105.70013888888889,39.800138
```

18. Press Enter to send the updated request. The Layer Preview should change like this where you can see both the SRTM and hillshade layers.



Fig. 196: Layer preview with SRTM and hillshade being overlaid

Styling with CSS

The CSS extension module allows to build map styles using a compact, expressive styling language already well known to most web developers: Cascading Style Sheets.

The standard CSS language has been extended to allow for map filtering and managing all the details of a map production. In this section we'll experience creating a few simple styles with the CSS language.

Creating line styles

1. From the main menu bar select the [CSS styles](#) entry

2. Click on the “Choose a different layer” link and switch to the Mainrd layer
3. Click on the “Create a new style” link and input `css_mainrd` as the style name, then press the “Create” button

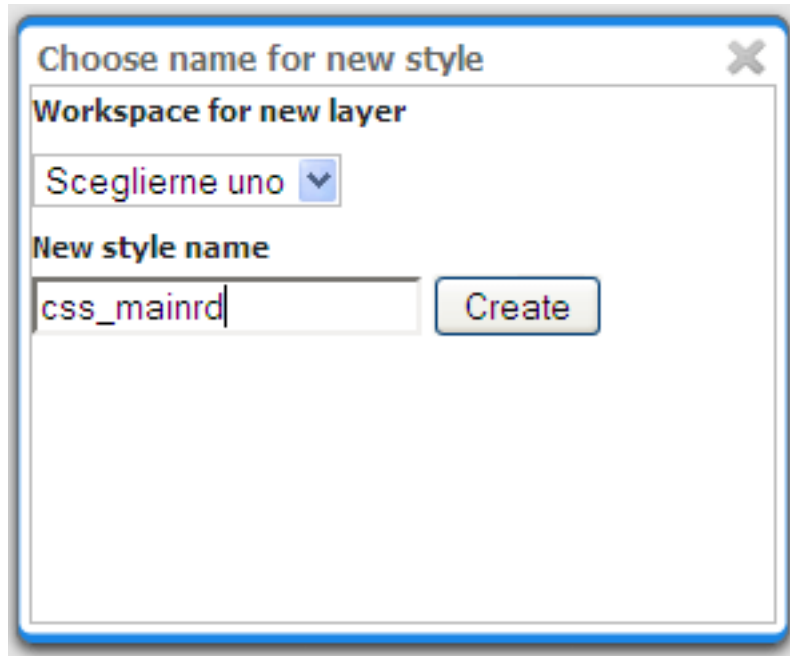


Fig. 197: Creating a new CSS style for the Mainrd layer

4. Set the style contents to the following, press submit and switch to the map preview

```
* {
  stroke: orange;
  stroke-width: 6;
  stroke-linecap: round;
}
```

5. Now let's create a cased line effect by adding a second set of colours and widths, and forcing two different z indexes for them. Press submit, look at the map and at the generated SLD

```
* {
  stroke: orange, yellow;
  stroke-width: 6, 2;
  stroke-linecap: round;
  z-index: 1, 2;
}
```

6. Finally, let's add a label that follows the road

```
* {
  stroke: orange, yellow;
  stroke-width: 6, 2;
  stroke-linecap: round;
  z-index: 1, 2;
  label: [LABEL_NAME];
  font-fill: black;
  font-family: Arial;
  font-size: 12;
}
```

(continues on next page)

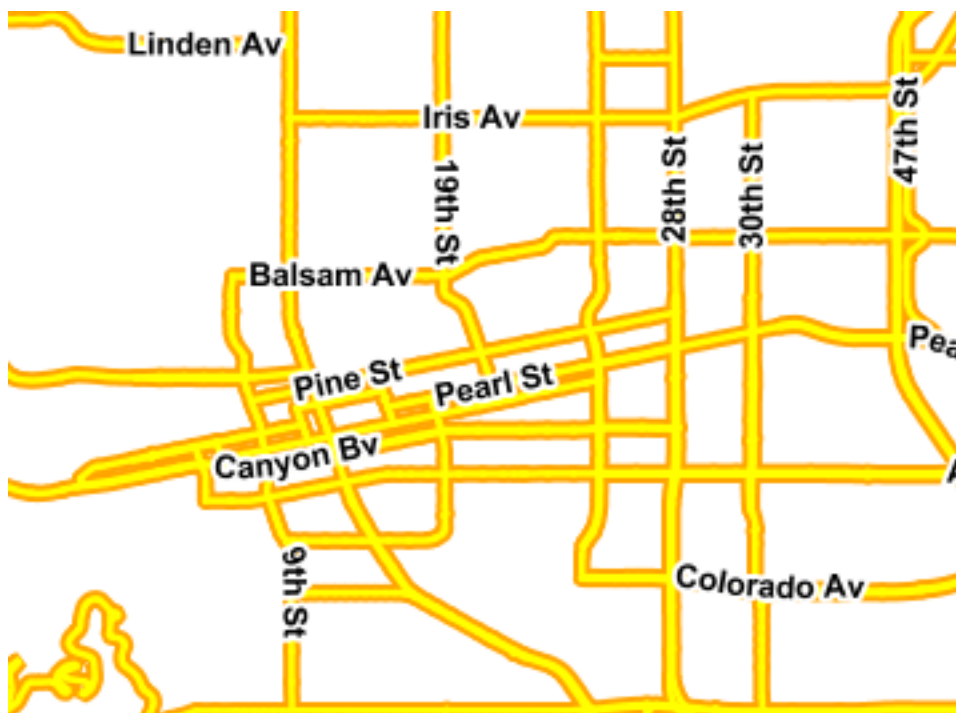


(continued from previous page)

```

font-weight: bold;
halo-color: white;
halo-radius: 2;
-gt-label-follow-line: true;
-gt-label-group: true;
-gt-label-repeat: 400;
-gt-label-max-displacement: 50;
}

```



Creating point styles

1. Similarly to the previous section, switch the map to “bptlandmarks” and create a new style called “css_bptlandmarks”
2. Insert the following in the CSS to get a basic point style, and press “Submit”:

```

* {
  mark: symbol('circle');
  mark-size: 5;
}

```

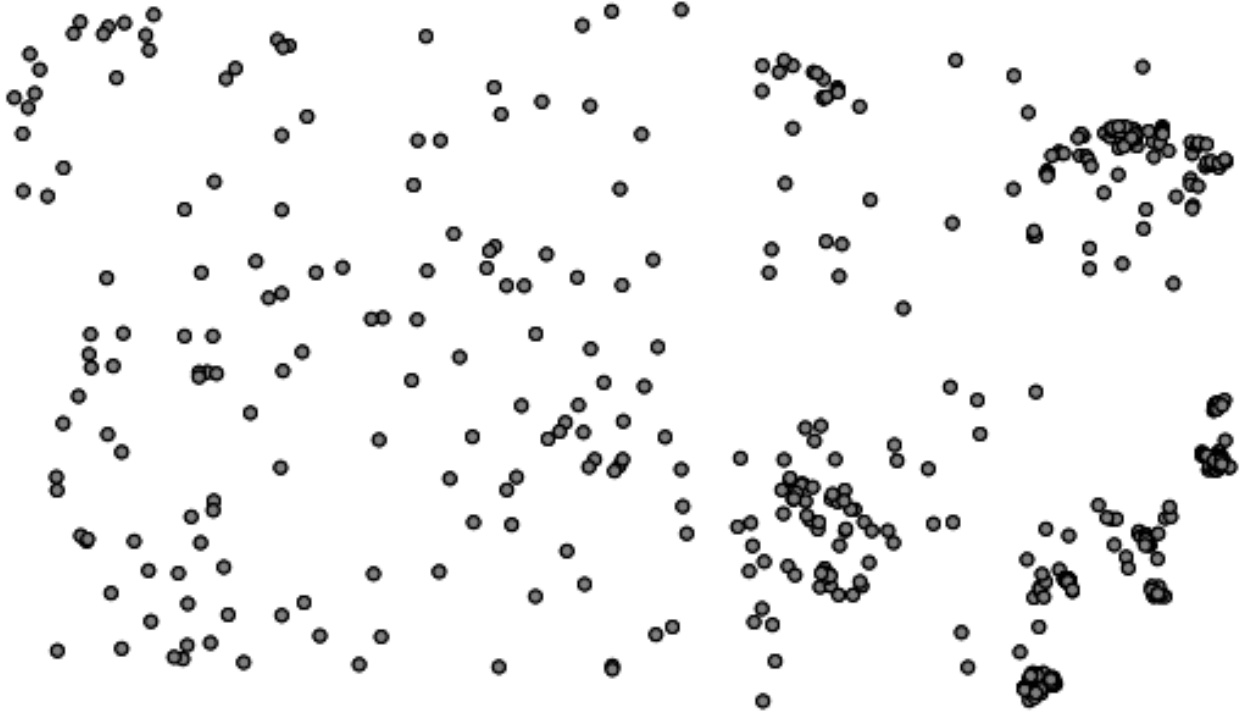
3. Let’s change the color of the points by specifying a fill. If we specified a fill in the top level rule it would be interpreted as a polygonal fill, to express that we want to fill inside the marks we have to create a new rule with the `:mark` pseudo-selector:

```

* {
  mark: symbol('circle');
  mark-size: 5;
}

```

(continues on next page)



(continued from previous page)

```
:mark {
  fill: cyan;
  stroke: darkblue;
}
```

4. Finally, let's override the default styling for all shopping centers. Shopping centers are not easy to find, they have a MTFCC category of C3081 and contain Shopping in the name

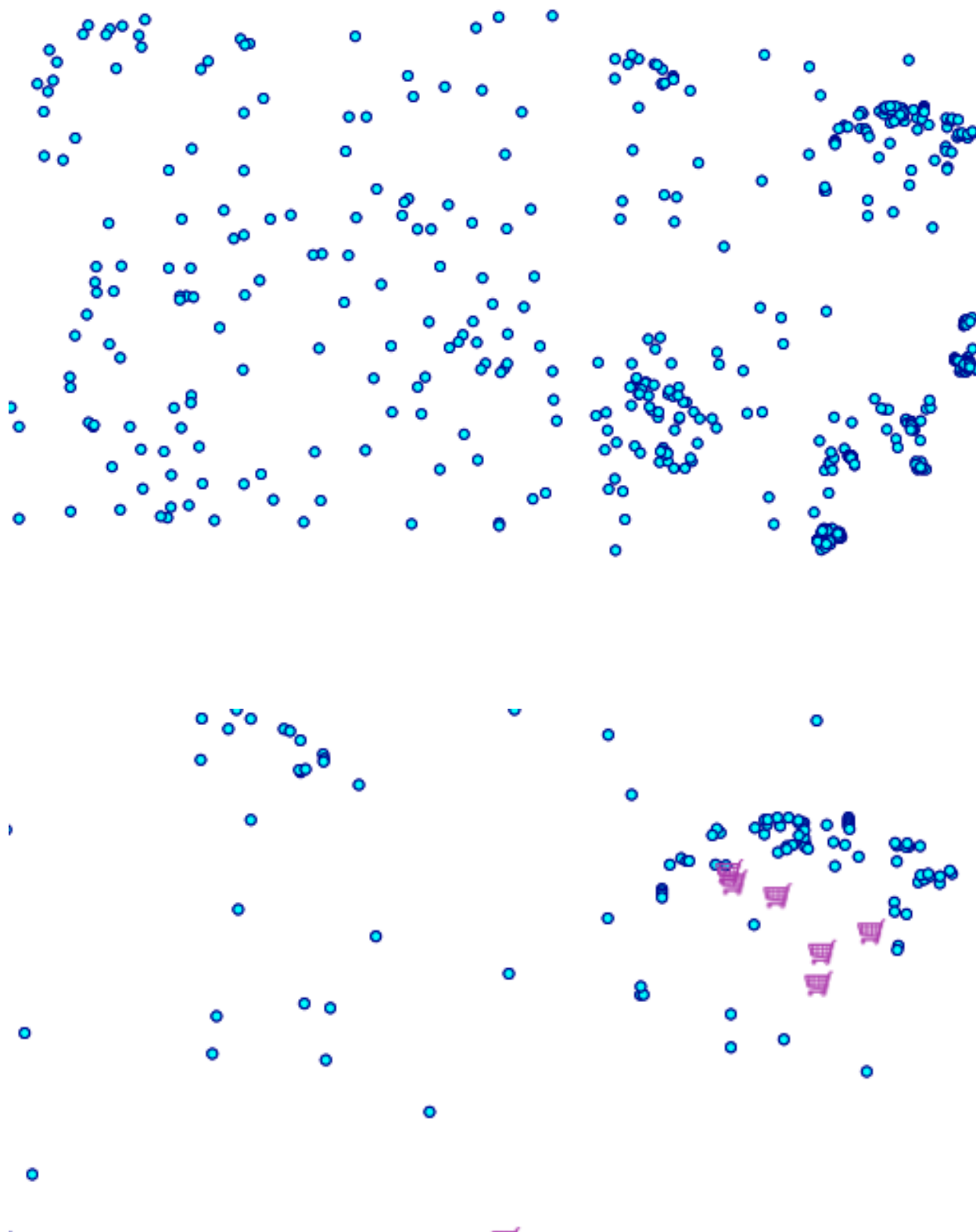
```
* {
  mark: symbol('circle');
  mark-size: 5;
}

:mark {
  fill: cyan;
  stroke: darkblue;
}

[MTFCC = 'C3081' AND FULLNAME LIKE '%Shopping%'] {
  mark: url("../img/landmarks/shop_supermarket.p.16.png");
  mark-size: ;
}
```

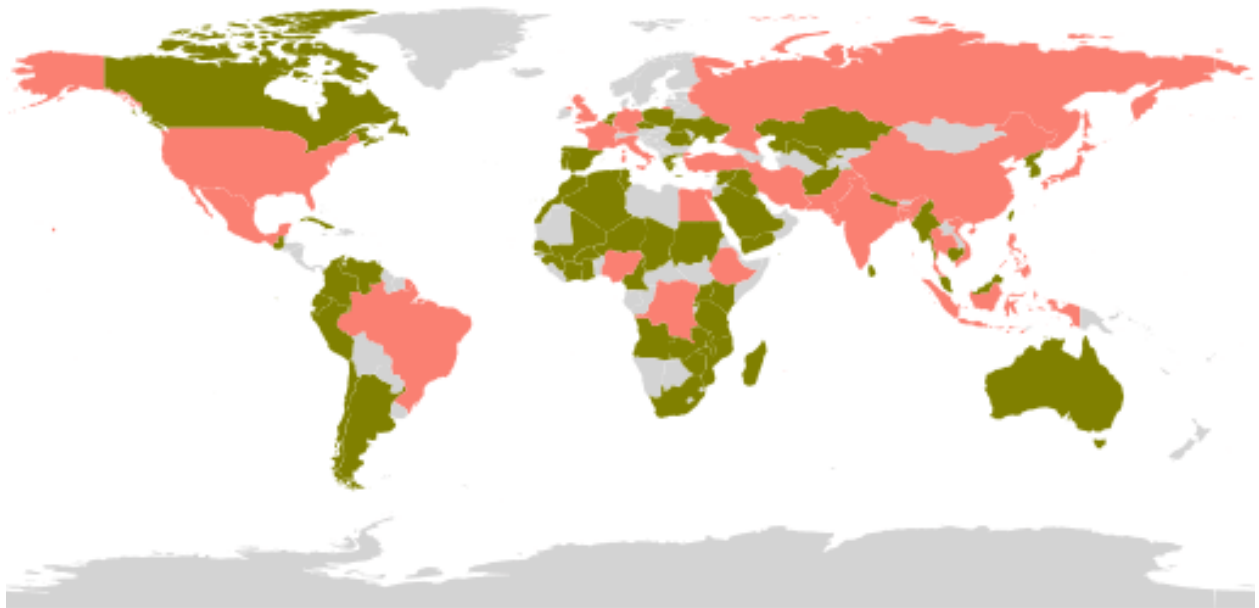
Creating polygon styles

1. For this exercise, change the current layer to “WorldCountries” and create a new style called “css_worldcountries”



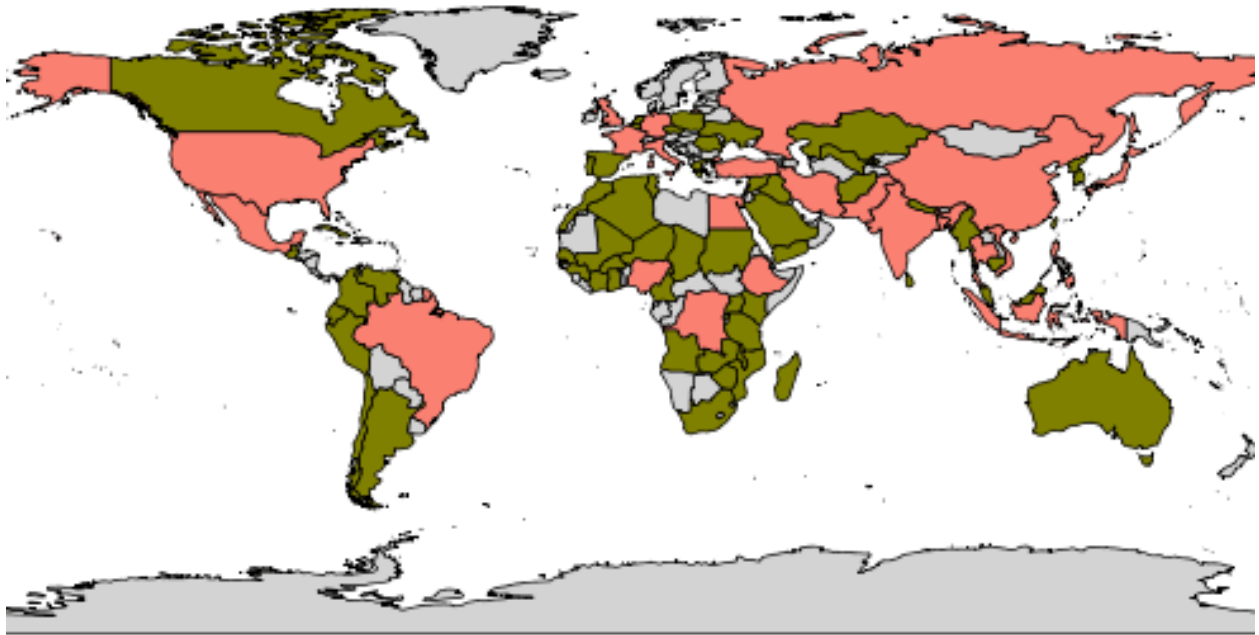
2. We want to create a simple 3 class thematic map based on the country population, stored in the POP_EST attribute

```
[POP_EST < 10000000] {  
  fill: lightgrey;  
}  
  
[POP_EST >= 10000000 AND POP_EST < 50000000] {  
  fill: olive;  
}  
  
[POP_EST > 50000000] {  
  fill: salmon  
}
```



3. Let's also add a very thin black border around all polygons, regardless of their population, using the * selector

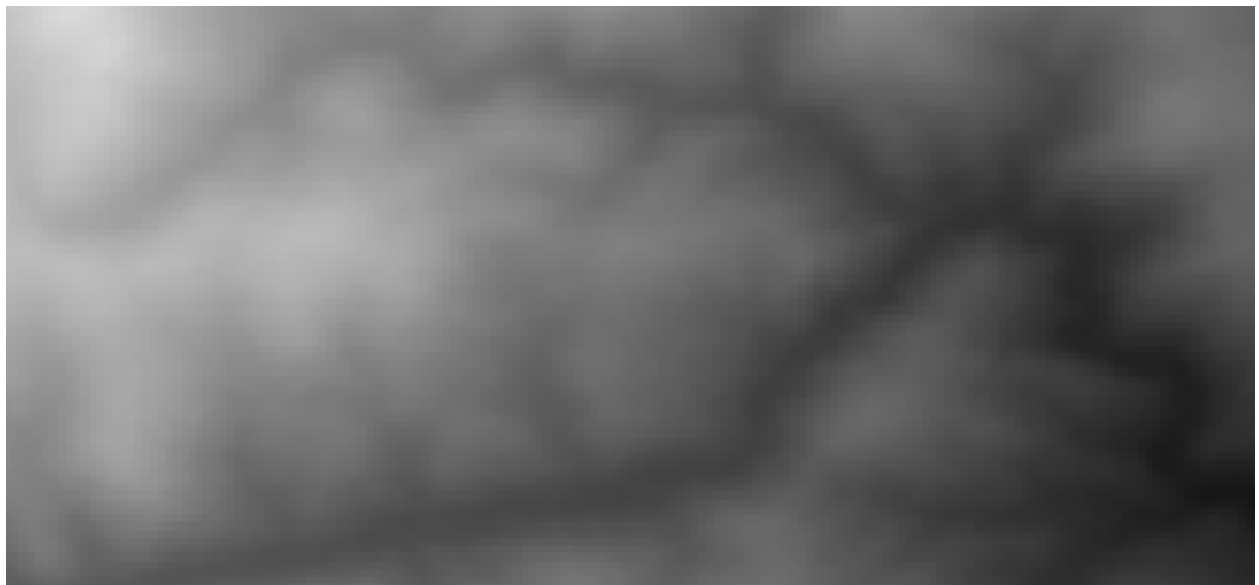
```
[POP_EST < 10000000] {  
  fill: lightgrey;  
}  
  
[POP_EST >= 10000000 AND POP_EST < 50000000] {  
  fill: olive;  
}  
  
[POP_EST > 50000000] {  
  fill: salmon  
}  
  
* {  
  stroke: black;  
  stroke-width: 0.2;  
}
```



Styling raster data

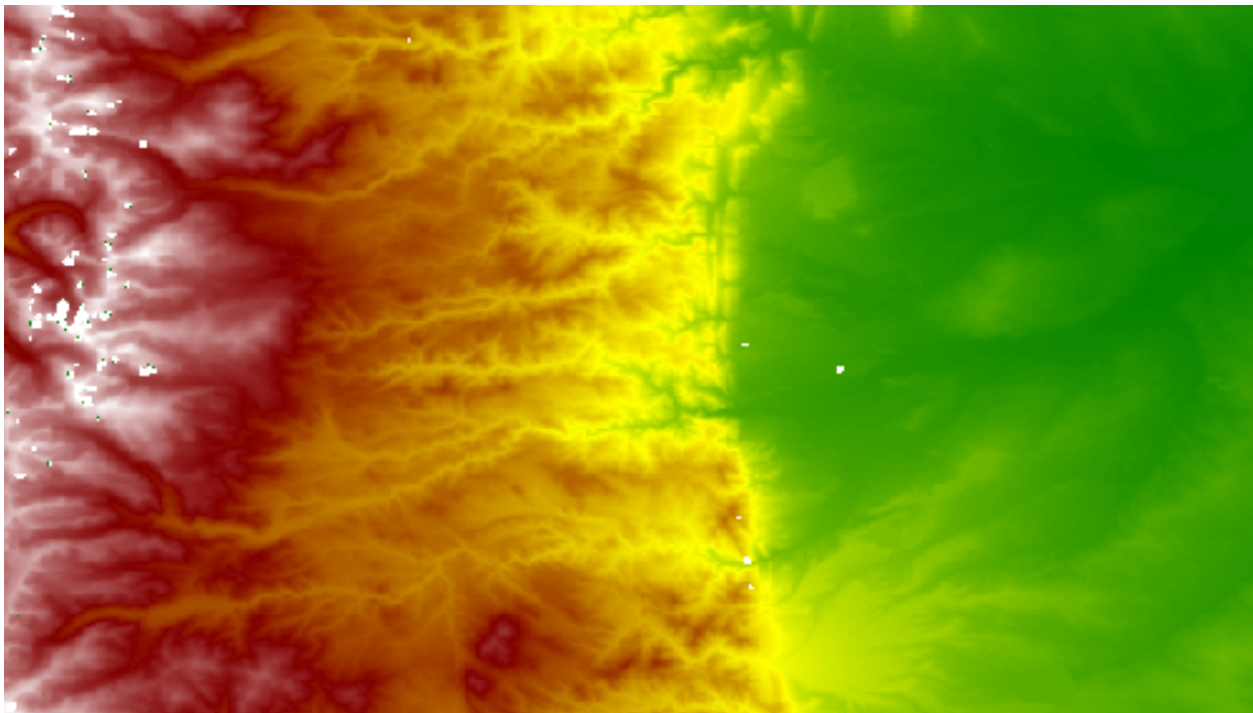
1. For this exercise we are going to switch to the “srtm” layer and create a new `css_raster` style
2. In order to activate raster styling the `raster-channels` property needs to be specified, in this case with a value of `auto` to let the engine choose automatically the bands to use:

```
* {  
  raster-channels: auto;  
}
```



3. The above map shows GeoServer automatically creating a grayscale map out of the elevation data, with automatic adaptation to the current contents of the map (the black areas you see once applied the map are “no data” areas, try to go into an area that does not have any)
4. Now let’s apply a color map to get a nicer and consistent looking map instead

```
* {  
  raster-channels: auto;  
  raster-color-map:  
    color-map-entry(black, 0, 0)  
    color-map-entry(green, 1500)  
  
    color-map-entry(yellow, 2000)  
    color-map-entry(maroon, 3000)  
    color-map-entry(white, 4000);  
}
```



Creating a Base Map with a Layer Group

The best way to easily set-up a map with more than one layer for consumption is to create a *Layer Group*, that is what we are going to do in this section.

1. Locate the *Layer Group* link and click it.
2. Click the *Add new layer group* link.
3. Name it test.
4. Click the *Add layer* link:
5. Select the “Mainrd” layer in the popup window.
6. Add also “BoulderCityLimits” and “bplandmarks”, the final list should look like this:.

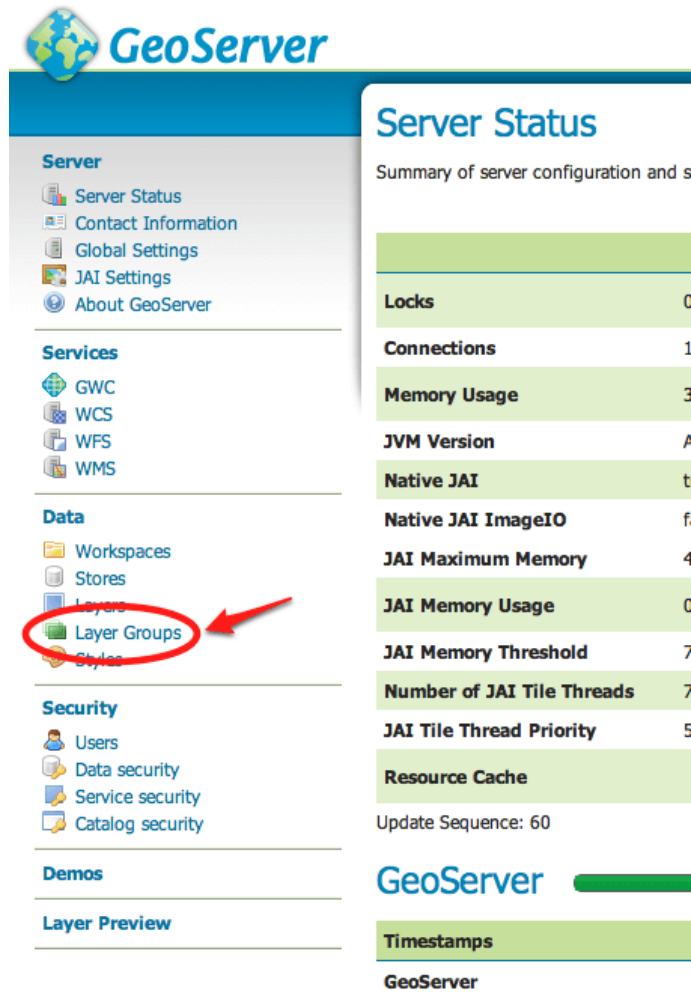


Fig. 198: Layer Group link

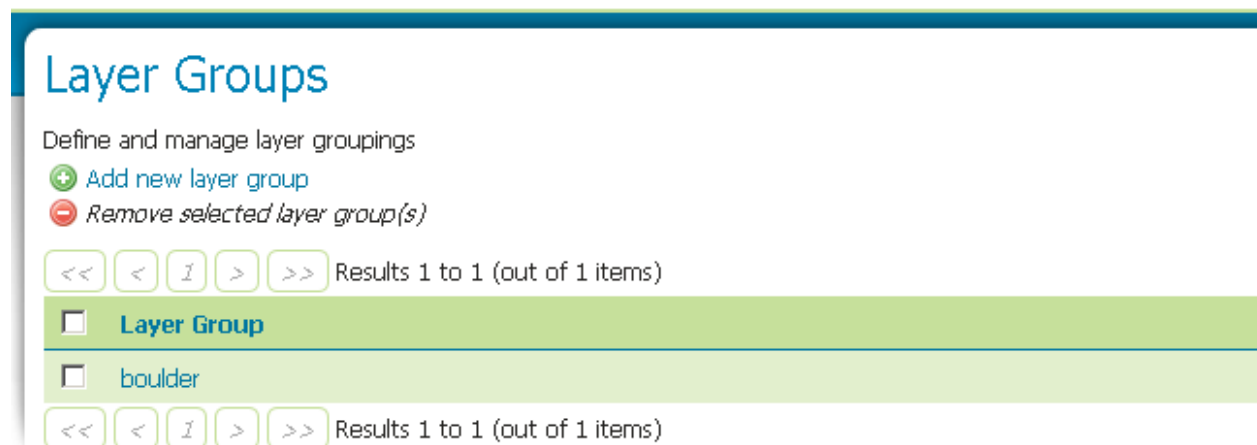


Fig. 199: Add new layer group link

New Layer Group

Add a new layer grouping

Name

Title

Abstract

Mode

Single

Layers

- ☒ Add Layer...
- ☒ Add Layer Group...

Position	Layer
----------	-------

<< < > >> Results 0 to 0 (out of 0 items)

Fig. 200: Add new layer

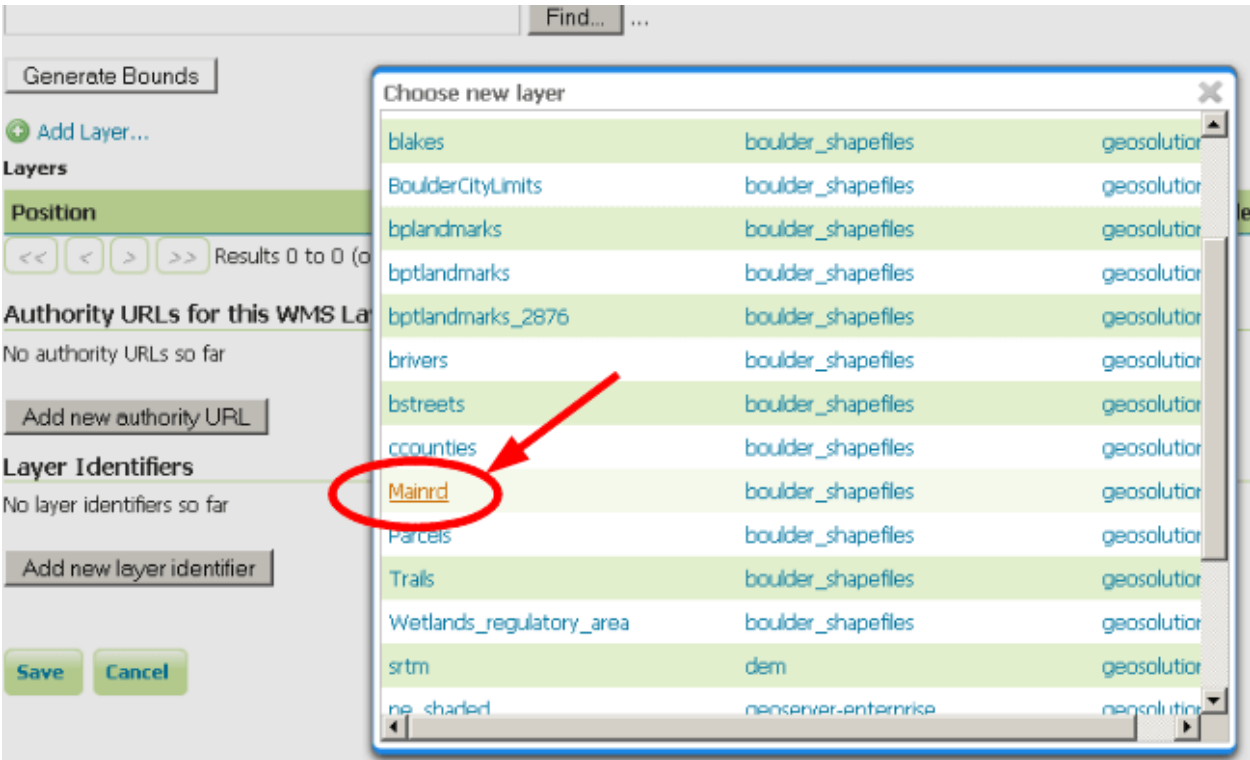


Fig. 201: Select a layer

Layers

Add Layer...

Add Layer Group...

Position	Layer	Default Style	Style	Remove
<div><div></div></div>	geosolutions:Mainrd	<div><div></div></div>	mainrd	<div><div></div></div>
<div><div></div><div></div></div>	geosolutions:BoulderCityLimits	<div><div></div></div>	citylimits	<div><div></div></div>
<div><div></div></div>	geosolutions:bplandmarks	<div><div></div></div>	arealandmarks	<div><div></div></div>

<<

<

I

>

>>

Results 0 to 0 (out of 0 items)

Fig. 202: List of layers for the group

Note: You can use the green arrows to adjust the ordering of the layers until it looks like the above figure.

7. Click the generate bounds button to have GeoServer compute the group bounds from the layers inside of it:
8. Scroll to the bottom of the page and then click *Save*.
9. If all went well, you should see something like this:

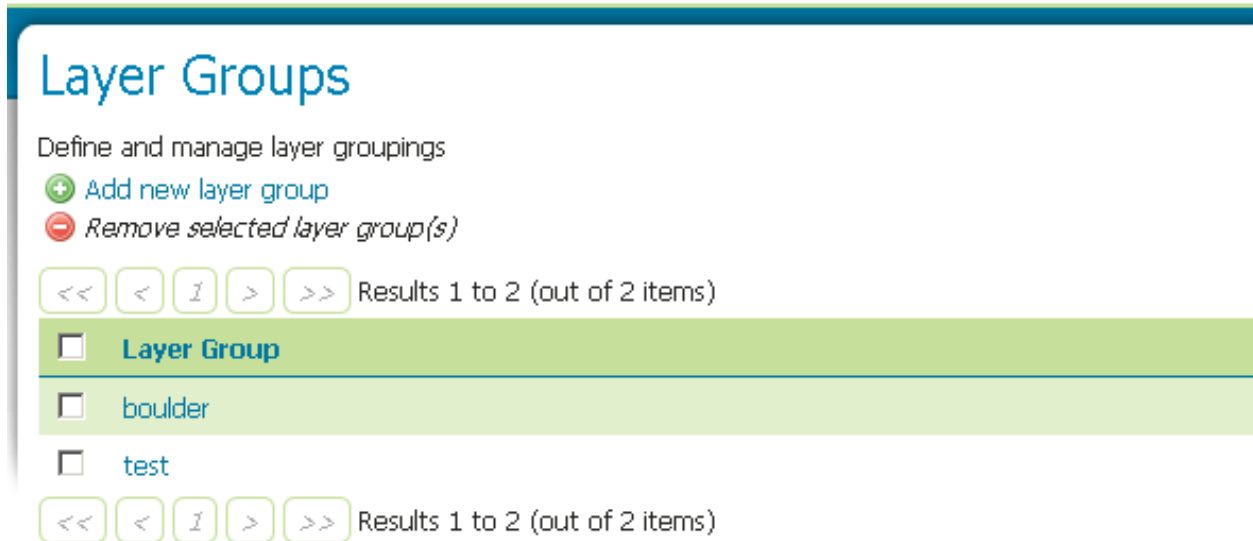


Fig. 203: After a successful save.

Note: The autogenerated bounds may be too large and you may experience a bad feeling when previewing the map. You can optionally reduce the layer group bounds by inserting manually the bbox values. Good values are the following: minx = 3.057.566,8646; maxx = 3.079.500,65246; miny = 1.241.929,35617; maxy = 1.257.467,5777

The layer group is now ready to be consumed:

1. Navigate to the GeoServer [Welcome Page](#).
2. Go to the *Layer Preview* link at the bottom of the left-hand menu.
3. Find the *test* layer group and click on the *OpenLayers* link. You will see a slippy map with all the configured layers of the Boulder district. You can control zoom by using the mouse wheel, pan by dragging, and zoom by window holding **SHIFT** pressed while dragging.

Note: Check the browser's address bar for an interesting sample WMS request for the layer.

4. As you might have noticed before, a larger, more realistic group has already been configured for you. It is named *boulder*. Have a look at its definition and add to this the *Mainrd* layer. Then using the green arrows move the layer at the following position (see the screenshot).
5. Then use the Map Preview to display it.

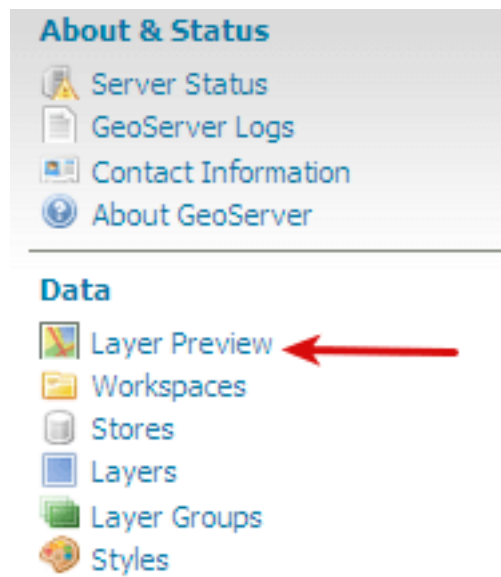


Fig. 204: Layer Preview

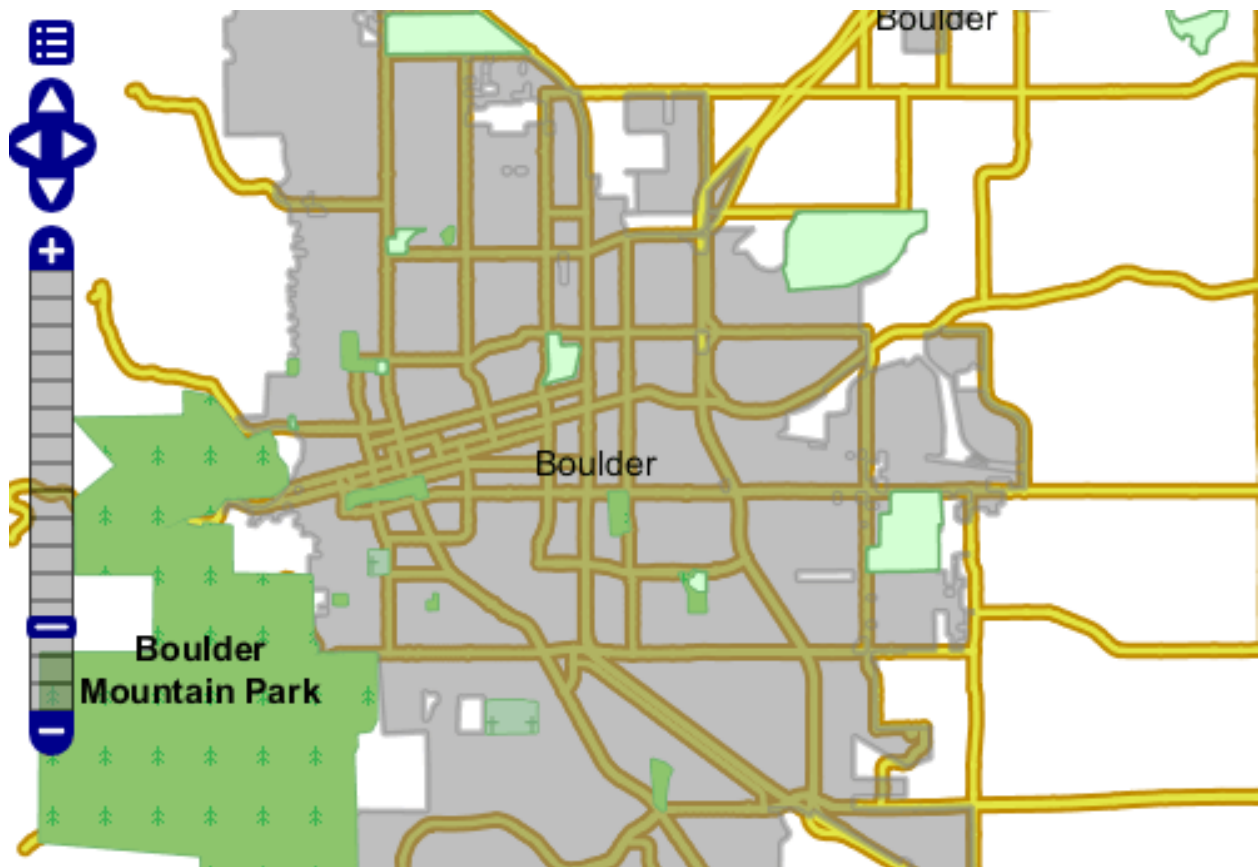


Fig. 205: OpenLayers view

Position	Layer	Default Style	Style
↓	geosolutions:srtm	<input checked="" type="checkbox"/>	<i>dem</i>
↑ ↓	geosolutions:BoulderCityLimits	<input checked="" type="checkbox"/>	<i>citylimits</i>
↑ ↓	geosolutions:bplandmarks	<input checked="" type="checkbox"/>	<i>arealandmarks</i>
↑ ↓	geosolutions:blakes	<input checked="" type="checkbox"/>	<i>lakes</i>
↑ ↓	geosolutions:brivers	<input checked="" type="checkbox"/>	<i>rivers</i>
↑ ↓	geosolutions:Trails	<input checked="" type="checkbox"/>	<i>trails</i>
↑ ↓	geosolutions:Wetlands_regulatory_area	<input checked="" type="checkbox"/>	<i>wetlands</i>
↑ ↓	geosolutions:Parcels	<input checked="" type="checkbox"/>	<i>parcels</i>
↑ ↓	geosolutions:bbuildings	<input checked="" type="checkbox"/>	<i>buildings</i>
↑ ↓	geosolutions:bstreets	<input checked="" type="checkbox"/>	<i>streets</i>
↑ ↓	geosolutions:Mainrd	<input checked="" type="checkbox"/>	<i>mainrd</i>
↑	geosolutions:bptlandmarks_2876	<input checked="" type="checkbox"/>	<i>point_landmark_u</i>

Fig. 206: A new layer inside the existing layer group.

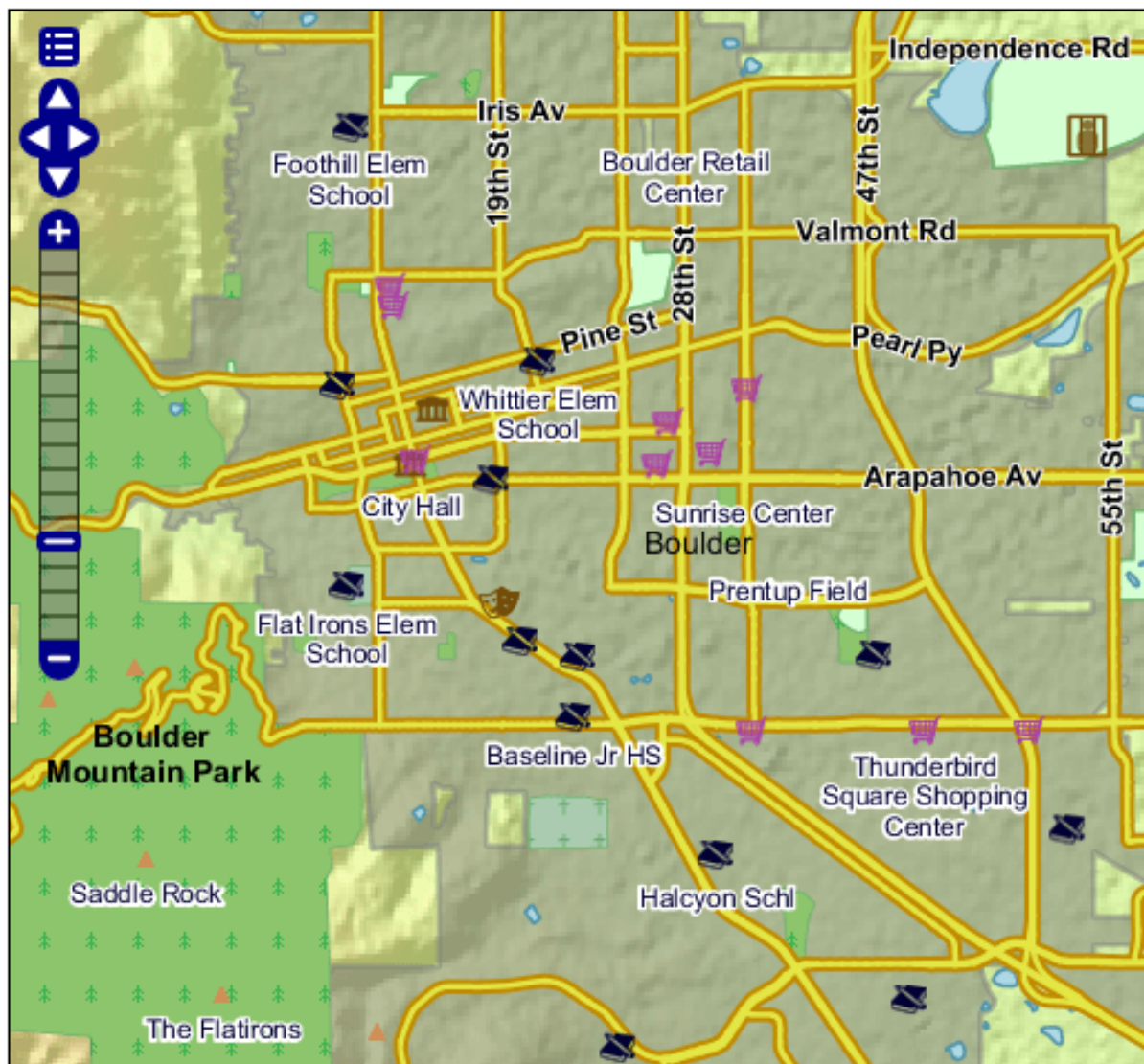
6. Try clicking in the middle of the map. A couple of tables with more information about the vector features that have been clicked should appear at the bottom.
7. Try zooming in more and more. New layers should start to appear. This is scale dependent styling.

Now let's see how desktop clients handle the layer group, and how we can change the way the see it.

1. Go to the command line, enter the workshop directory, if you haven't done that yet, run `setenv.bat` and then `udig.bat`
2. Once both GeoServer and uDig are up, organize their Windows so that you can see GeoServer and uDig ones at the same time
3. Now go to the GeoServer home page, where all the capabilities links are kept, take the WMS 1.1.1 one, and drag&drop it into uDig "Catalog" tab to import the WMS as a uDig data source:
4. Look at the layer tree. The `boulder` group is visible as a simple layer, and all the layers it contains are actually shown at the same level as the group.
5. Let's change it so that the layer group internal structure is shown. Go back to the "boulder" layer group page, and change its "mode" to "Named tree", then press the "Save" button
6. We need to make uDig aware of the change. Right click the root of the capabilities tree and choose the "Reset" command
7. Now most of the layers are contained inside the "boulder" group

Filtering Maps

This section shows the GeoServer WMS filtering capabilities.

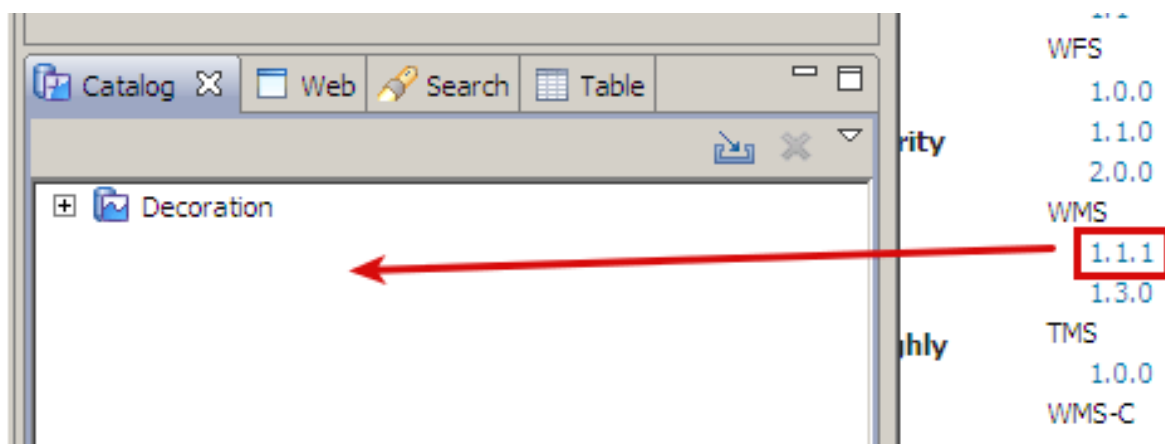


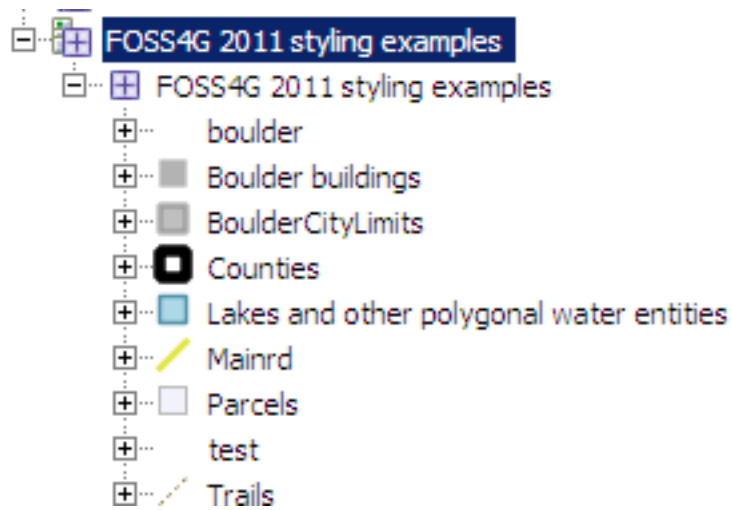
Scale = 1 : 171K

3077309.82062, 1249385.37324

Click on the map to get feature info

Fig. 207: Feature info



**Bounds**

Min X	Min Y	Max X	Max Y
2,943,771.9805851	1,170,043.152052	3,140,453.387552	1,352,572.813349

Coordinate Reference System

EPSG:2876

Find...

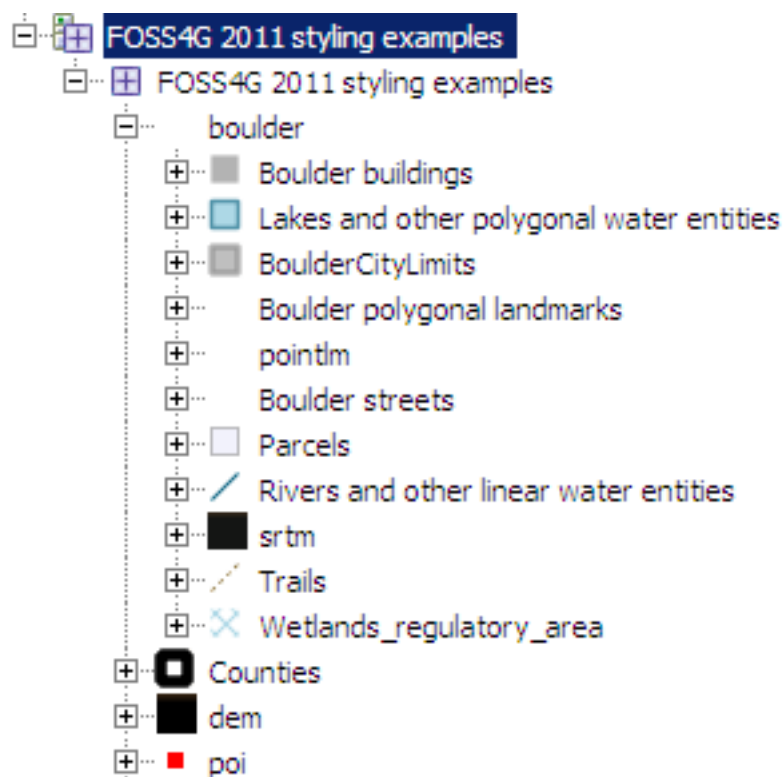
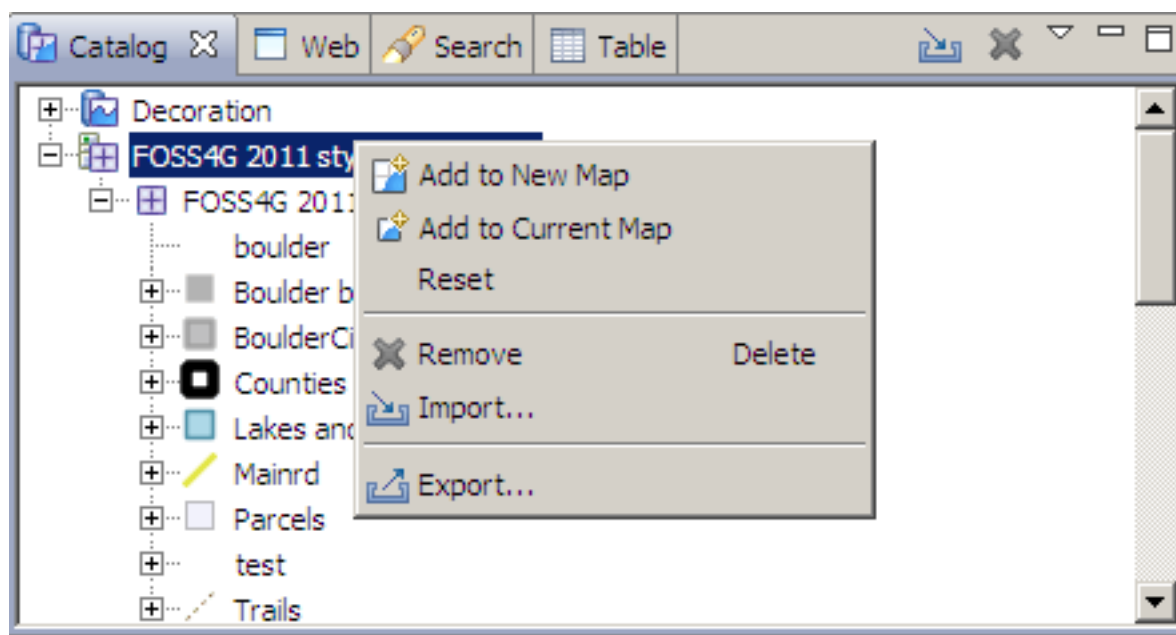
EPSG:NAD83(

Generate Bounds

Mode

- Single
- Single
- Named Tree**
- Container Tree
- Earth Observation Tree

 Add Layer Group...



1. Navigate to the GeoServer [Welcome Page](#).
2. Go to the *Layer Preview* link at the bottom of the left-hand menu and show the *geosolutions:WorldCountries* layer with OpenLayers 'Common Format'.

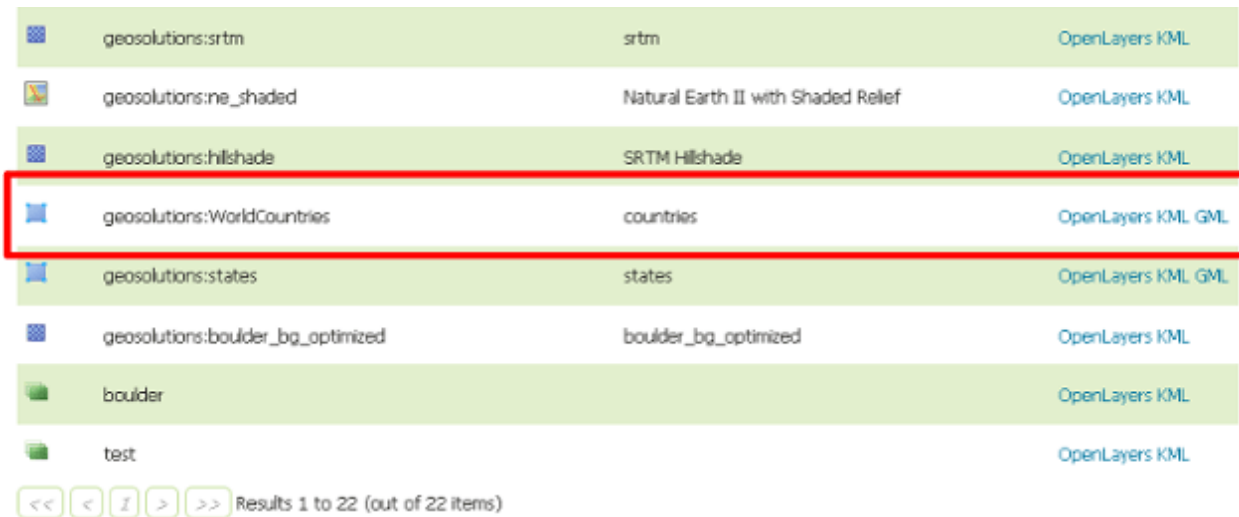


Fig. 208: Showing the GeoServer layer preview

3. From the *Filter* combo box select 'CQL' and enter the following command in the text field:

```
POP_EST <= 5000000 AND POP_EST >100000
```

4. Click 'Apply Filter' button on the right.

The corresponding WMS request is:

```
http://localhost:8083/geoserver/geosolutions/wms?service=WMS&version=1.1.0&
request=GetMap&layers=geosolutions:WorldCountries&styles=&bbox=-180.0,-89.
99889902136009,180.00000000000003,83.59960032829278&width=684&height=330&
srs=EPSG:4326&format=image/png&CQL_FILTER=POP_EST%20%3C%205000000%20AND%20POP_
EST%20%3E100000
```

5. Now enter the following command in the text field:

```
DISJOINT(the_geom, POLYGON((-90 40, -90 45, -60 45, -60 40, -90 40))) AND
strToLowerCase(NAME) LIKE '%on%'
```

6. Click 'Apply Filter' button on the right.

The corresponding WMS request is:

```
http://localhost:8083/geoserver/geosolutions/wms?service=WMS&version=1.1.0&
request=GetMap&layers=geosolutions:WorldCountries&styles=&bbox=-180.0,-89.
99889902136009,180.00000000000003,83.59960032829278&width=684&height=330&
srs=EPSG:4326&format=image/png&CQL_FILTER=DISJOINT%28the_geom%2C%20POLYGON%28
%28-90%2040%2C%20-90%2045%2C%20-60%2045%2C%20-60%2040%2C%20-90%2040%29%29%29
%20AND%20strToLowerCase%28NAME%29%20LIKE%20%27%25on%25%27
```

7. From the *Filter* combo box select 'OGC' and enter the following filter in the text field:

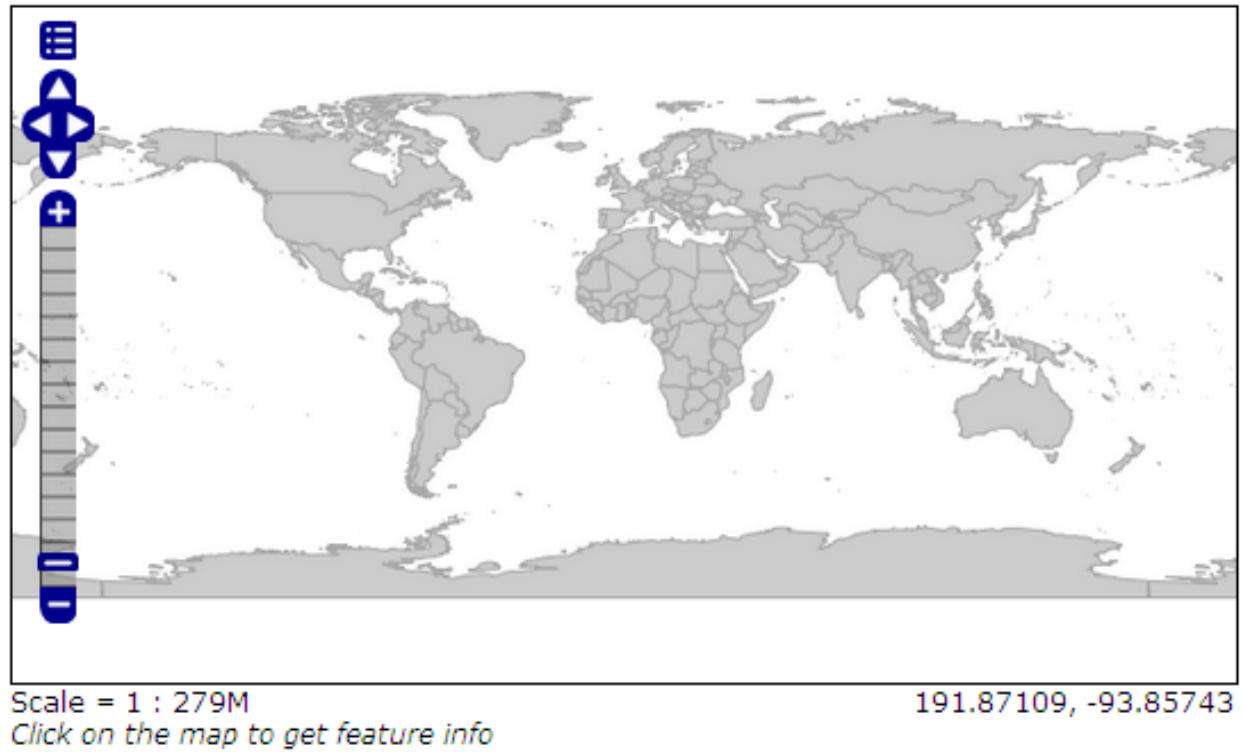


Fig. 209: Show the layer with OpenLayers



Fig. 210: Result of the CQL filter

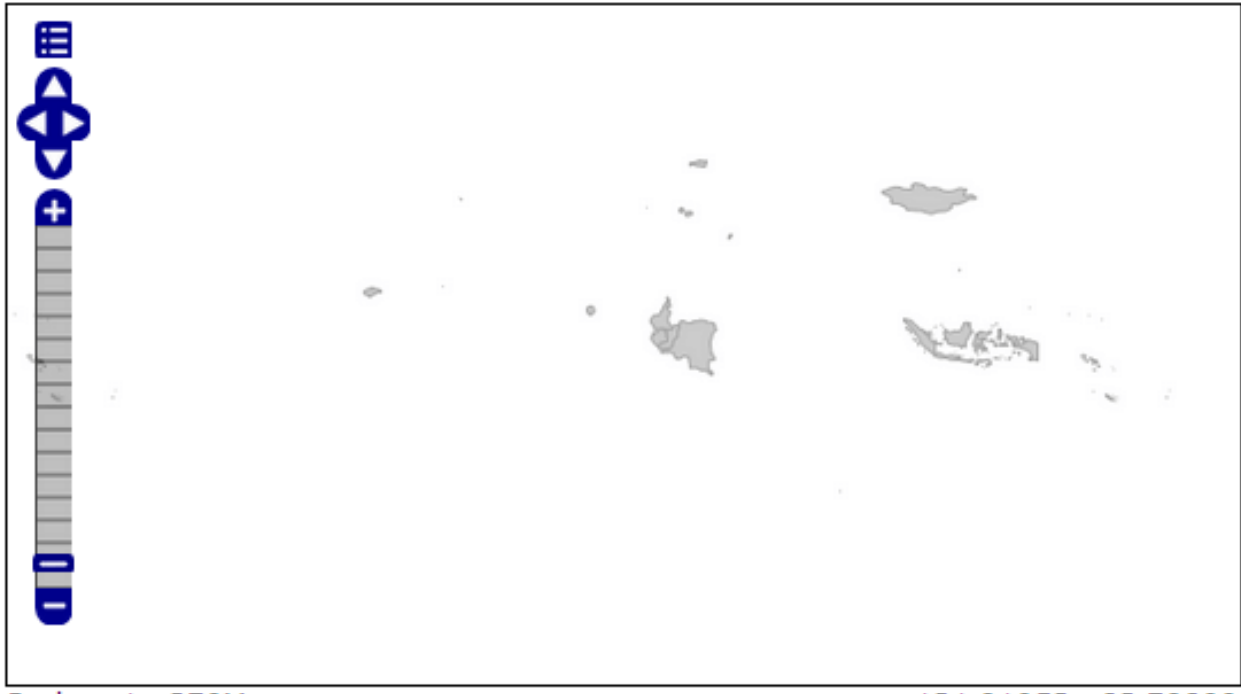


Fig. 211: Result of the CQL filter

```
<Filter><PropertyIsEqualTo><PropertyName>TYPE</PropertyName><Literal>Sovereign_
↪country</Literal></PropertyIsEqualTo></Filter>
```

8. Click 'Apply Filter' button on the right.

The corresponding WMS request is

```
http://localhost:8083/geoserver/geosolutions/wms?service=WMS&version=1.1.0&
↪request=GetMap&layers=geosolutions:WorldCountries&styles=&bbox=-180.0,-89.
↪99889902136009,180.00000000000003,83.59960032829278&width=684&height=330&
↪srs=EPSG:4326&format=image/png&CQL_FILTER=TYPE%20%3D%20Sovereign%20country%27
```

9. From the *Filter* combo box select 'FeatureID' and enter the following features ids in the text field separated by comma:

```
WorldCountries.227,WorldCountries.184,WorldCountries.33
```

10. Click 'Apply Filter' button on the right.

The corresponding WMS request is:

```
http://localhost:8083/geoserver/geosolutions/wms?service=WMS&version=1.1.0&
↪request=GetMap&layers=geosolutions:WorldCountries&styles=&bbox=-180.0,-89.
↪99889902136009,180.00000000000003,83.59960032829278&width=684&height=330&
↪srs=EPSG:4326&format=image/png&FEATUREID=WorldCountries.227,WorldCountries.184,
↪WorldCountries.33
```



Fig. 212: Result of the OGC filter



Fig. 213: Result of the FeatureID filter

Producing and Using palettes

GeoServer has the ability to output high quality 256 color images. This tutorial introduces you to the palette concepts, the various image generation options, and offers a quality/resource comparison of them in different situations. In this section the task is to use the palettes.

Note: Some image formats, such as GIF or PNG, can use a palette, which is normally a table of 256 colors use get get better compression (trading it sometimes with a lower image quality). Basically, instead of representing each pixel with its full color triplet, which takes 24bits (plus eventual 8 more for transparency), they use a 8 bit index that represent the position inside the palette, and thus the color. This allows for images that are 3-4 times smaller than the standard images, with the limitation that only 256 different colors can appear on the image itself. Depending of the actual map, this may be a very stringent limitation, visibly degrading the image quality, or it may be that the output cannot be told from a full color image. For many common vector maps one can easily find 256 representative colors that are a good fit. In the latter case, the smaller footprint of paletted images is usually a gain in both performance and costs, because more data can be served with the same internet connection, and the clients will obtain responses faster.

Options to enable paletted output

The easiest way to get a paletted image output is to ask for a 256 color output format, such as:

- image/png8: PNG output, with a 256 color palette
- image/gif: standard GIF output

These output formats, if no other parameters are provided, do compute the optimal palette on the fly. This is an expensive process (CPU bound) but, depending on the speed of the network connecting the server and the client, the extra CPU cost can be offset by a lower data transfer time (especially on slow/busy networks).

Optimal palette computation is anyway a repetitive work that can be done up front: a user can compute the optimal palette once, and tell GeoServer to use it. There are three ways to do so:

- Use the [internet safe palette](#), a standard palette built in into GeoServer, by appending palette=safe to the GetMap request. Of course, to get good results, the styling will have to be made using the colors in that palette.
- Provide a palette by example. In this case, the user will generate an 256 color images using an external program (such as Photoshop), and then will save it into the \$GEOSERVER_DATA_DIR/palettes directory. The sample file can be either in GIF or PNG format. If the file is named mypalette.gif or mypalette.png, the user will be able to refer it appending palette=mypalette to the GetMap request. GeoServer will load the palette from the file and use it.
- Provide a palette file. The process is just as before, but this time only the palette, in .PAL format, will be stored into the \$GEOSERVER_DATA_DIR/palettes directory. The PAL file in in Microsoft Palette Format, and can be generated by programs such as Paint Shop Pro and IrfanView.

An Example with Vector Data

Enough theory, let's have a look at how to deal with paletted images in practice. We'll use the `prato` basemap to gather some numbers and we'll change various parameters in order to play with formats and palettes. Here goes the sampler:

- 1) The standard PNG full color output:

```
http://localhost:8083/geoserver/wms?service=WMS&version=1.1.0&request=GetMap&
↳layers=geosolutions:BoulderCityLimits,geosolutions:blakes,
↳geosolutions:bplandmarks,geosolutions:drivers,geosolutions:Mainrd&format=png
↳bbox=3056181.93510,1237476.92868,3080671.07513,1260141.38768&width=512&
↳height=475&srs=EPSG:2876&format=image/png
```

(continued from previous page)

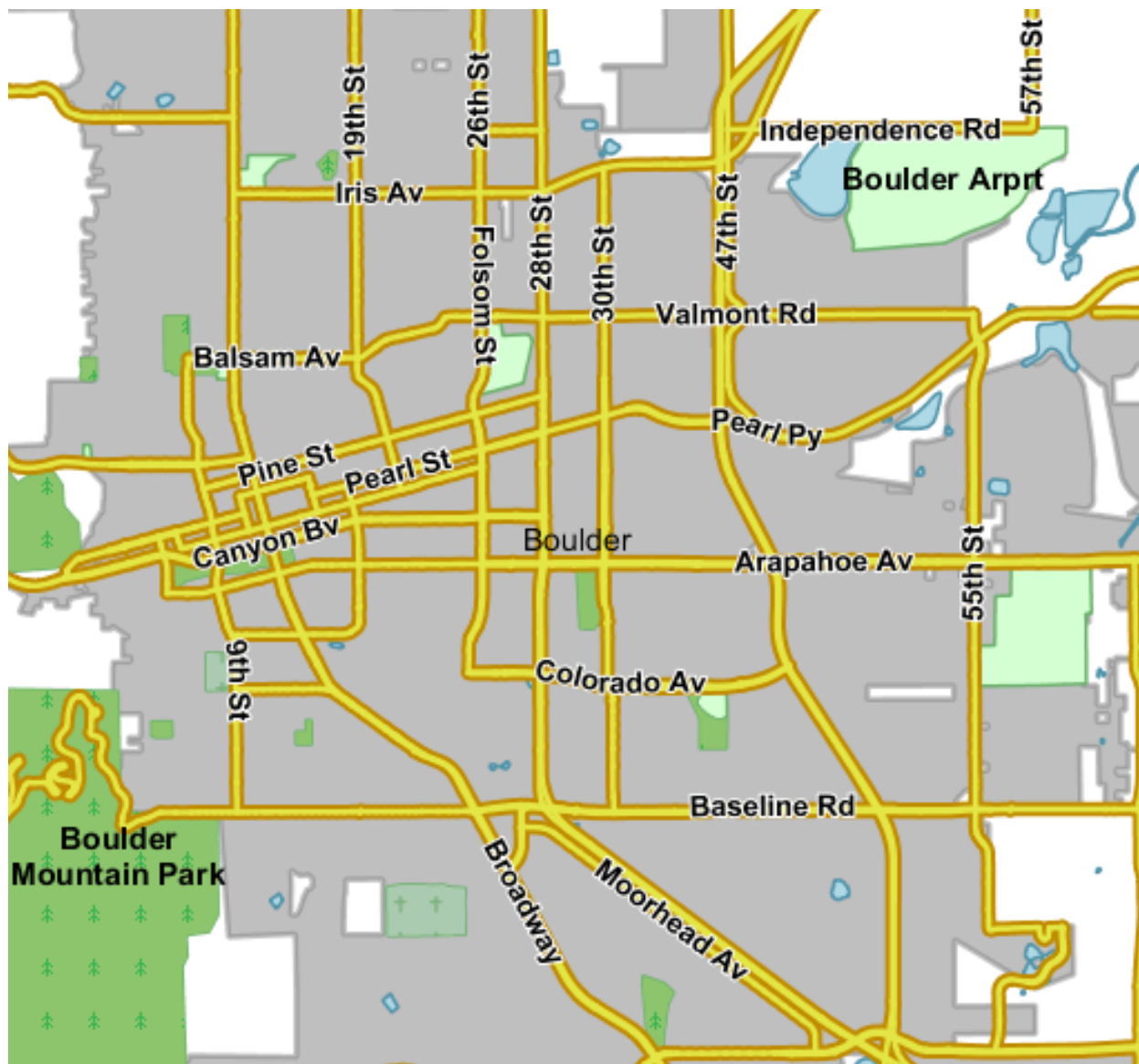


Fig. 214: The standard PNG output

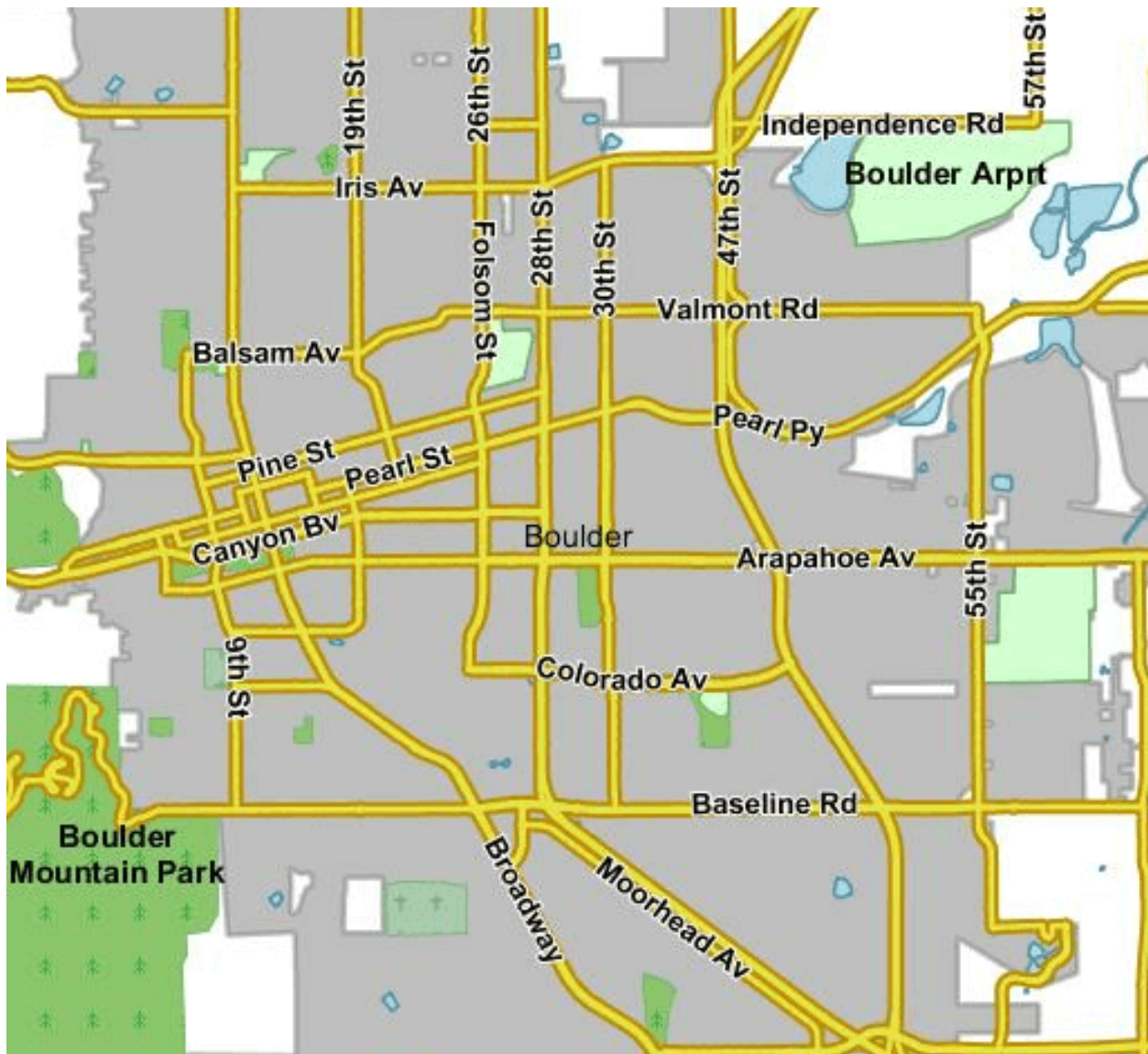
Parameters:FORMAT=image/png | Size: 105.5 KB | Map generation time: 186 ms

2) JPEG output:

```
http://localhost:8083/geoserver/wms?service=WMS&version=1.1.0&request=GetMap&
↳layers=geosolutions:BoulderCityLimits,geosolutions:blakes,
↳geosolutions:bplandmarks,geosolutions:brivers,geosolutions:Mainrd&styles=&
↳bbox=3056181.93510,1237476.92868,3080671.07513,1260141.38768&width=512&
↳height=475&srs=EPSG:2876&format=image/jpeg
```

Parameters:FORMAT=image/jpeg | Size: 43.2 KB | Map generation time: 100 ms

3) The PNG8 output:

Fig. 215: *JPEG output*

```
http://localhost:8083/geoserver/wms?service=WMS&version=1.1.0&request=GetMap&
↳layers=geosolutions:BoulderCityLimits,geosolutions:blakes,
↳geosolutions:bplandmarks,geosolutions:brivers,geosolutions:Mainrd&styles=&
↳bbox=3056181.93510,1237476.92868,3080671.07513,1260141.38768&width=512&
↳height=475&srs=EPSG:2876&format=image/png8
```

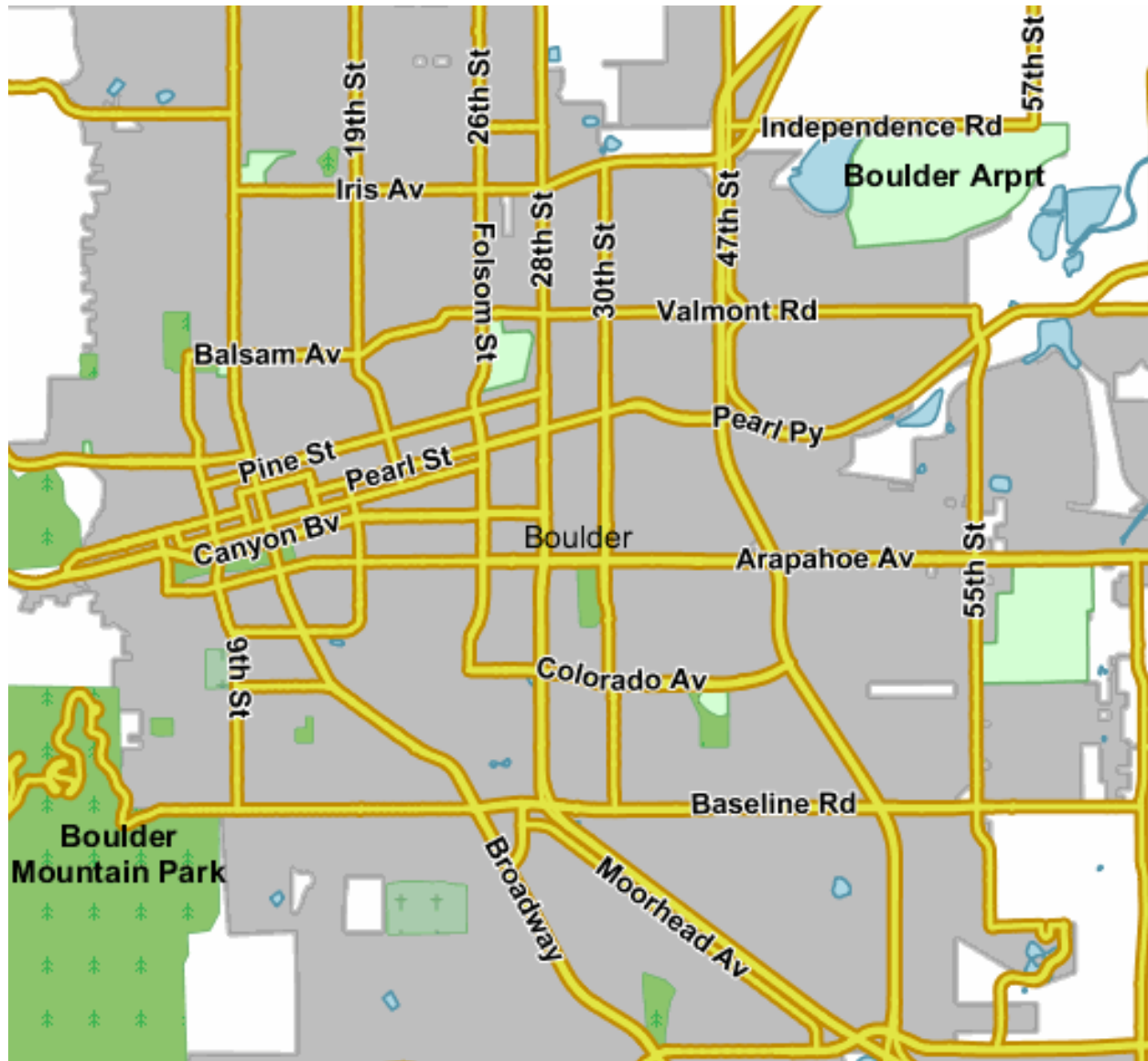


Fig. 216: The PNG8 output

Parameters:FORMAT=image/png8 | Size: 48.0 KB | Map generation time: 190 ms

4) PNG + internet safe palette:

```
http://localhost:8083/geoserver/wms?service=WMS&version=1.1.0&request=GetMap&
↳layers=geosolutions:BoulderCityLimits,geosolutions:blakes,
↳geosolutions:bplandmarks,geosolutions:brivers,geosolutions:Mainrd&styles=&
↳bbox=3056181.93510,1237476.92868,3080671.07513,1260141.38768&width=512&
↳height=475&srs=EPSG:2876&format=image/png&palette=safe
```

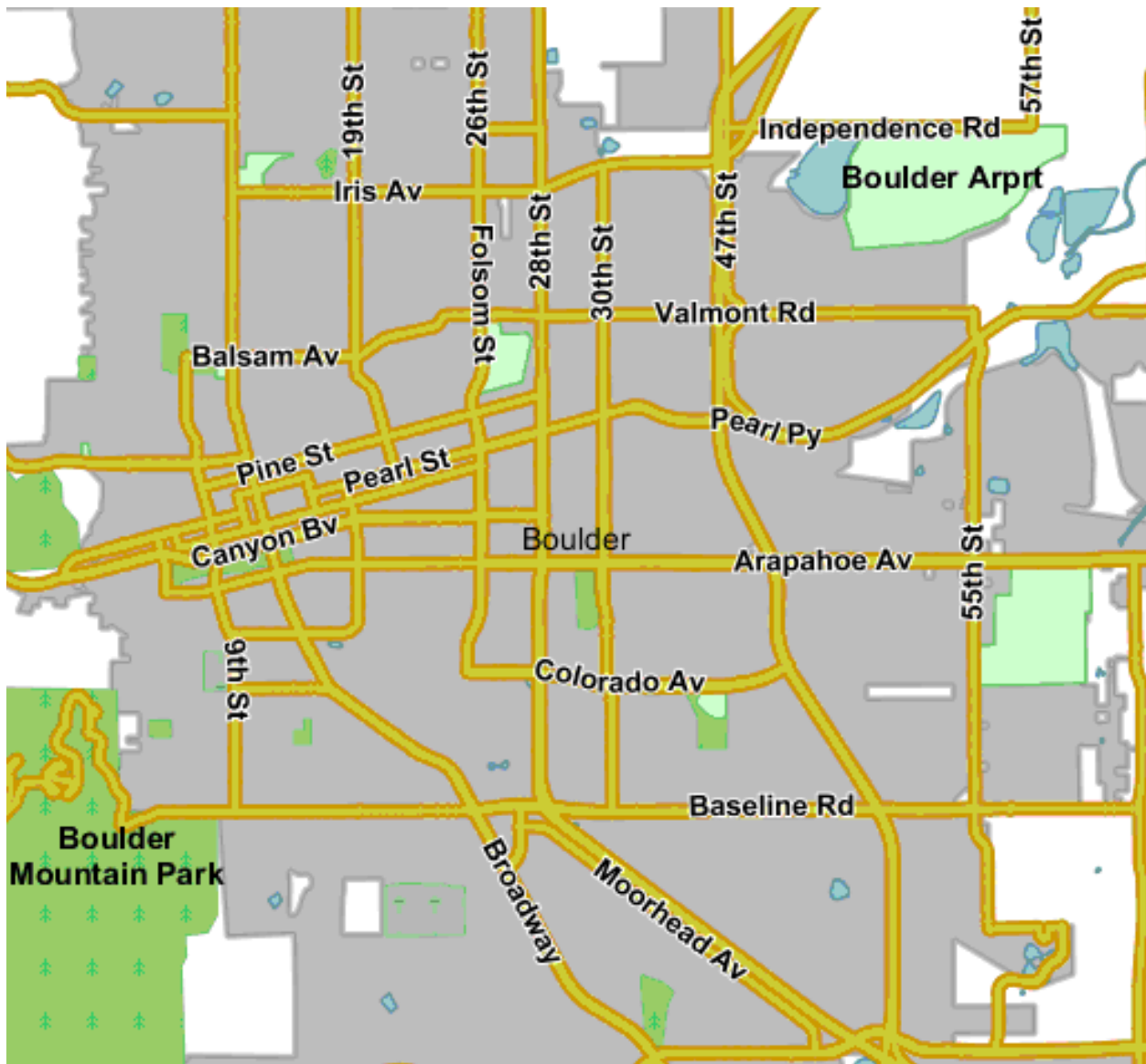


Fig. 217: The PNG output + internet safe palette

Parameters:FORMAT=image/png&palette=safe | Size: 38.8 KB | Map generation time: 161 ms

5) PNG + palette by example:

```
http://localhost:8083/geoserver/wms?service=WMS&version=1.1.0&request=GetMap&
↳layers=geosolutions:BoulderCityLimits,geosolutions:blakes,
↳geosolutions:bplandmarks,geosolutions:brivers,geosolutions:Mainrd&styles=&
↳bbox=3056181.93510,1237476.92868,3080671.07513,1260141.38768&width=512&
↳height=475&srs=EPSG:2876&format=image/png&palette=boulder
```

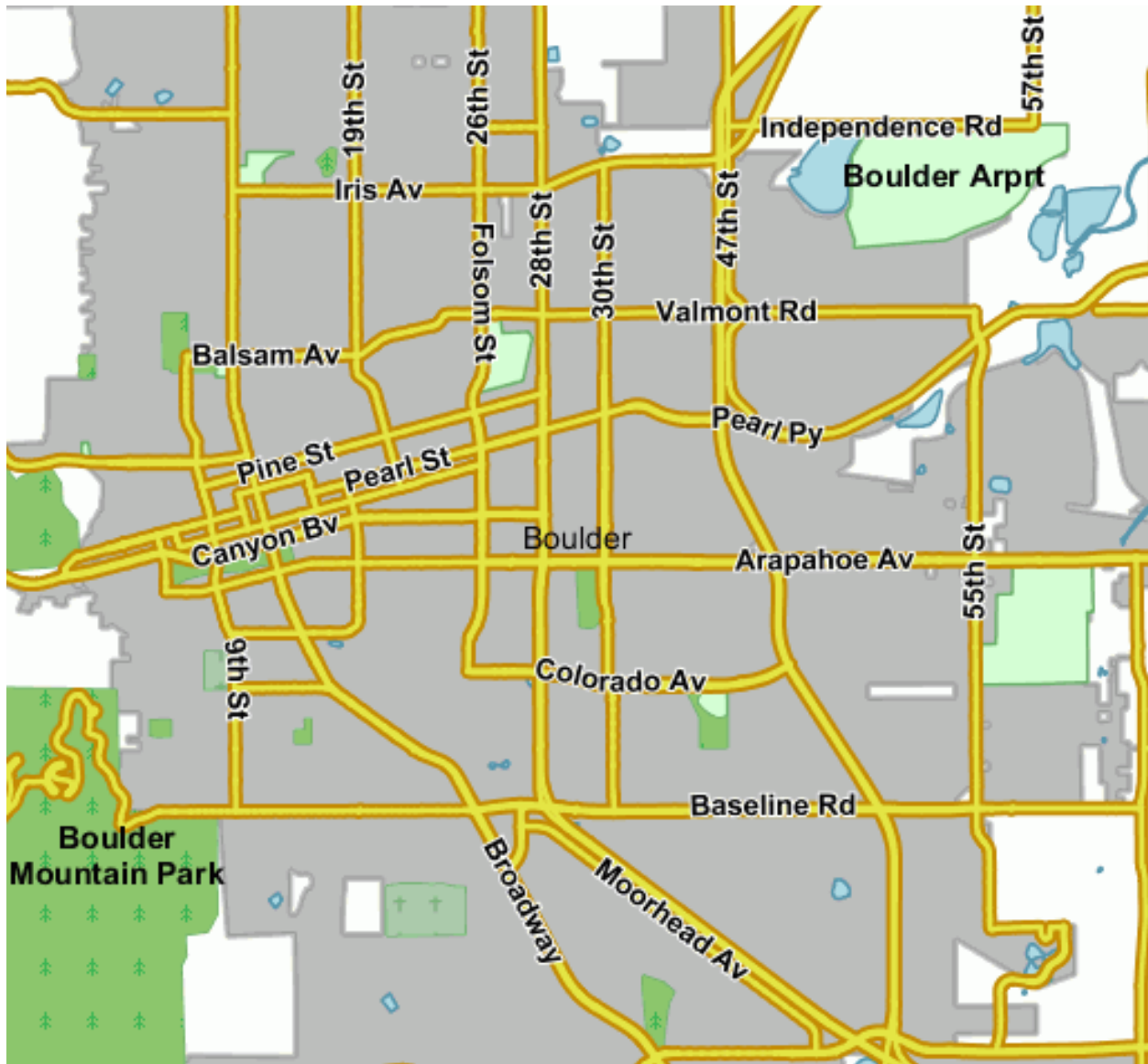


Fig. 218: The PNG output palette by example

Parameters:FORMAT=image/png&palette=boulder | Size: 54.4 KB | Map generation time: 163 ms

Generating the custom palette

To generate a custom palette you can use IrfanView for example, on Windows. The steps are simple:

- Open the 24-bit PNG version of the image
- Use Image/Decrease Color Depth and set 256 colors
- Use Image/Palette/Export to save the palette

An example with raster data

To give you an example when paletted images may not fit the bill, let's consider the geosolutions:13tde815295_200803_0x6000m_cl coverage from the sample data, and repeat the same operation as before.

- 1) The standard PNG full color output:

```
http://localhost:8083/geoserver/geosolutions/wms?LAYERS=geosolutions
↳%3A13tde815295_200803_0x6000m_cl&STYLES=&SERVICE=WMS&VERSION=1.1.1&
↳REQUEST=GetMap&SRS=EPSG%3A26913&BBOX=482574.82910157,4429949.7070313,482949.
↳82910157,4430324.7070313&WIDTH=512&HEIGHT=512&FORMAT=image%2Fpng
```

Parameters:FORMAT=image/png | Size: 528.9 KB | Map generation time:90ms

- 2) JPEG output:

```
http://localhost:8083/geoserver/geosolutions/wms?LAYERS=geosolutions
↳%3A13tde815295_200803_0x6000m_cl&STYLES=&SERVICE=WMS&VERSION=1.1.1&
↳REQUEST=GetMap&SRS=EPSG%3A26913&BBOX=482574.82910157,4429949.7070313,482949.
↳82910157,4430324.7070313&WIDTH=512&HEIGHT=512&FORMAT=image%2Fjpeg
```

Parameters:FORMAT=image/jpeg | Size: 39.5 KB | Map generation time: 35ms

- 3) PNG8 output (the output using a “palette by example would be the same”):

```
http://localhost:8083/geoserver/geosolutions/wms?LAYERS=geosolutions
↳%3A13tde815295_200803_0x6000m_cl&STYLES=&SERVICE=WMS&VERSION=1.1.1&
↳REQUEST=GetMap&SRS=EPSG%3A26913&BBOX=482574.82910157,4429949.7070313,482949.
↳82910157,4430324.7070313&WIDTH=512&HEIGHT=512&FORMAT=image%2Fpng8
```

Parameters:FORMAT=image/png8 | Size: 141.8 KB | Map generation time: 201ms

- 4) PNG output + safe palette:

```
http://localhost:8083/geoserver/geosolutions/wms?LAYERS=geosolutions
↳%3A13tde815295_200803_0x6000m_cl&STYLES=&SERVICE=WMS&VERSION=1.1.1&
↳REQUEST=GetMap&SRS=EPSG%3A26913&BBOX=482574.82910157,4429949.7070313,482949.
↳82910157,4430324.7070313&WIDTH=512&HEIGHT=512&FORMAT=image%2Fpng&palette=safe
```

Parameters:FORMAT=image/png&palette=safe | Size: 96.8 KB | Map generation time: 235ms

Note: As the sampler shows, the JPEG output has the same quality as the full color image, is generated faster and uses only a fraction of its size. At the opposite, the version using the internet safe palette is fast and smaller than the full PNG, but the output is totally ruined. Everything considered, JPEG is the clear winner, sporting good quality, fast image generation and smaller size that. PNGs are the suggested imagery raster format only in case the output needs to be used as an overlay and thus requires transparent areas, or when the raster has large areas with uniform colors, which may happen for example in land use rasters.



Fig. 219: *The standard PNG output*

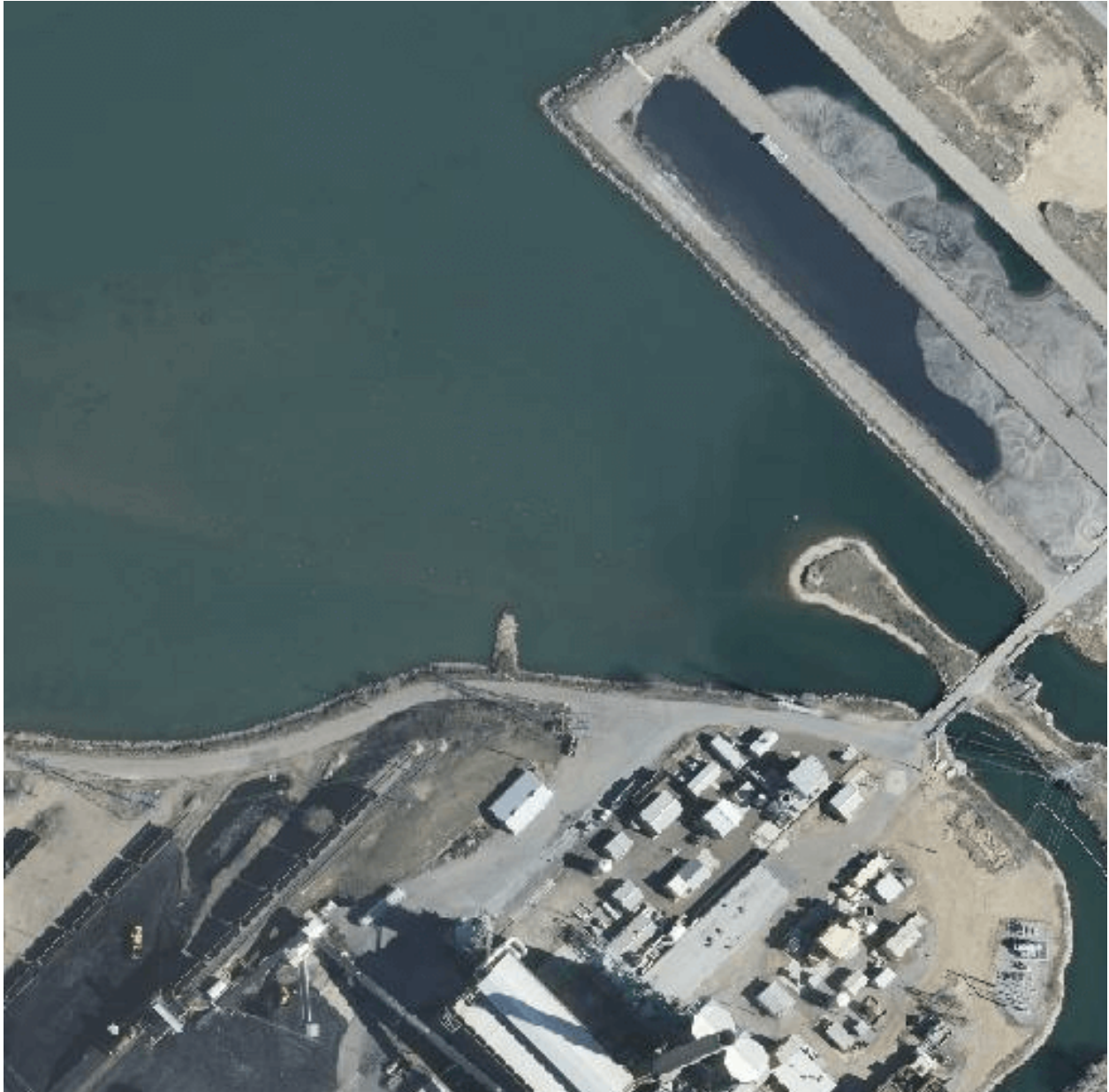


Fig. 220: *JPEG output*



Fig. 221: *PNG8 output*



Fig. 222: *PNG + sape palette output*

Decorating a Map

WMS Decorations provide a framework for visually annotating images from WMS with absolute, rather than spatial, positioning. Examples of this decoration include scale lines, legends, and image.

1. Go to `$GEOSERVER_DATA_DIR` and create a new directory named *layouts* and create a new file named *boulder_ly.xml* inside it.
2. Inside the *boulder_ly.xml* file enter the following XML (replace `${GEOSERVER_DATA_DIR}` with your actual path, e.g., `file:///C:/training/geoserver_data`):

```
<layout>

    <decoration type="image" affinity="top,left" offset="45,8"
        size="174,60">
        <option name="url"
            value="${GEOSERVER_DATA_DIR}/geosolutions-logo-tx.
↪png" />
    </decoration>

    <decoration type="text" affinity="bottom,right" offset="3,3">
        <option name="message" value="Boulder City" />
        <option name="font-size" value="14" />
        <option name="font-color" value="#FFFFFF" />
        <option name="halo-radius" value="1" />
        <option name="halo-color" value="#000000" />
    </decoration>

    <decoration type="scaleline" affinity="bottom,left" offset="3,3" /
↪>

    <decoration type="legend" affinity="top,right"
        offset="6,6" size="auto" />

</layout>
```

3. Save and close the file.
4. Go to the **Layer Preview** to preview the new map decoration on *geosolutions:Mainrd* layer. Once the layout *boulder_ly.xml* is defined, request it by adding *format_options=layout:boulder_ly* to the request parameters.

The request:

```
http://localhost:8083/geoserver/geosolutions/wms?service=WMS&version=1.1.0&
↪request=GetMap&layers=geosolutions:Mainrd&styles=&bbox=3048474.661,1226045.092,
↪3095249.0,1279080.5&width=451&height=512&srs=EPSG:2876&format=application/
↪openlayers&format_options=layout:boulder_ly
```

Note: Zoom-in until the layer and legend appears since for this layer we have *scale_denominator* based rules. Also you can apply this *format_layout* to any layer, but be careful with the overalys since you will have all the legends printed out on the right-top side of the map.

Accessing Map information

This workshop section describes how to use the GeoServer template system to create custom HTML GetFeatureInfo responses. GetFeatureInfo is a WMS standard call that allows one to retrieve information about features and coverages displayed in a map.

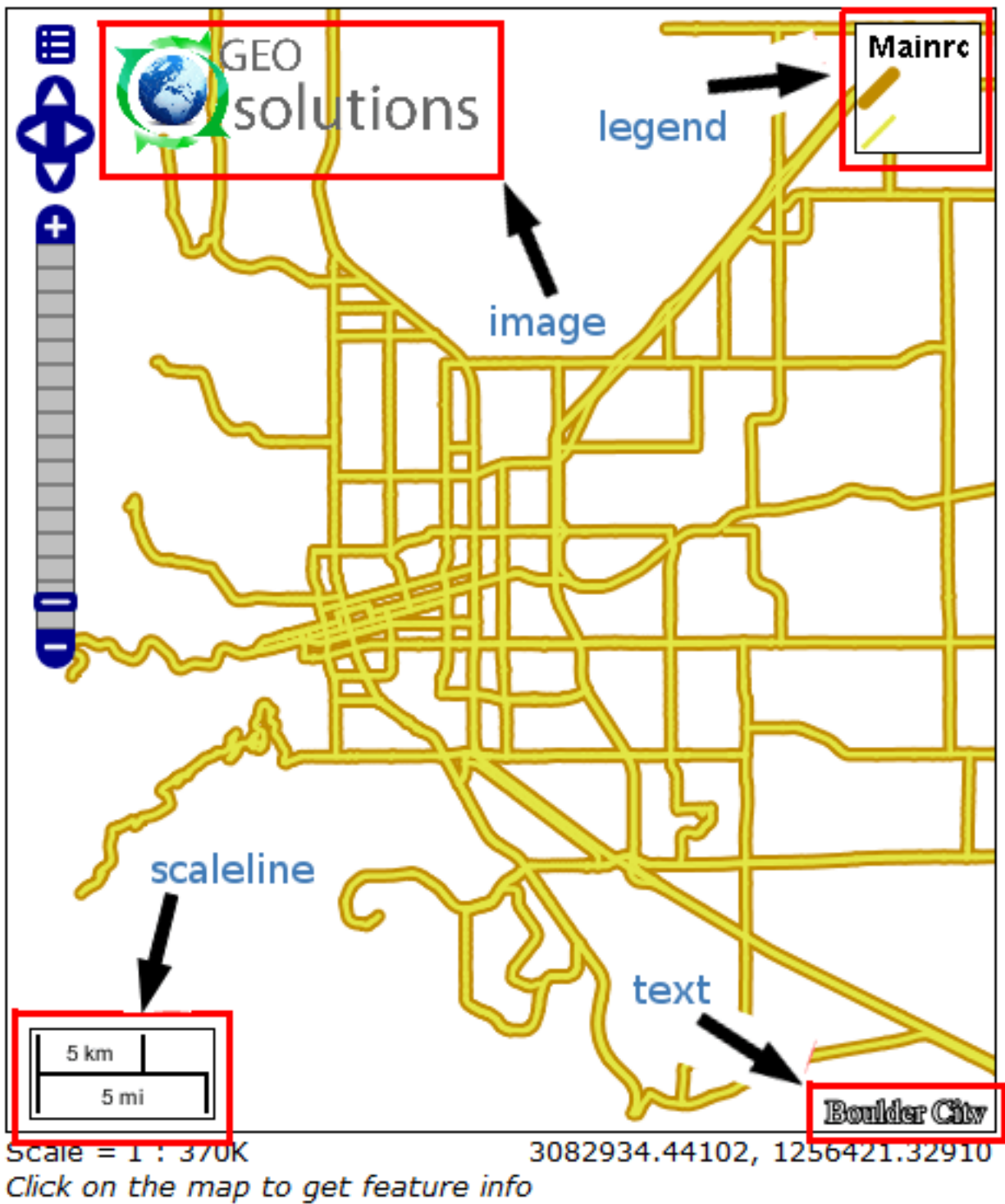


Fig. 223: Map decoration

The map can be composed of various layers, and GetFeatureInfo can be instructed to return multiple feature descriptions, which may be of different types. GetFeatureInfo can generate output in various formats: GML2, plain text and HTML.

Templating is concerned with the HTML one.

1. Go to the **Layer preview** to show *geosolutions:bplandmarks* layer.
2. Click for example on the *Rocky Mountain Natl Park* region in the OpenLayers map to show the FeatureInfo.

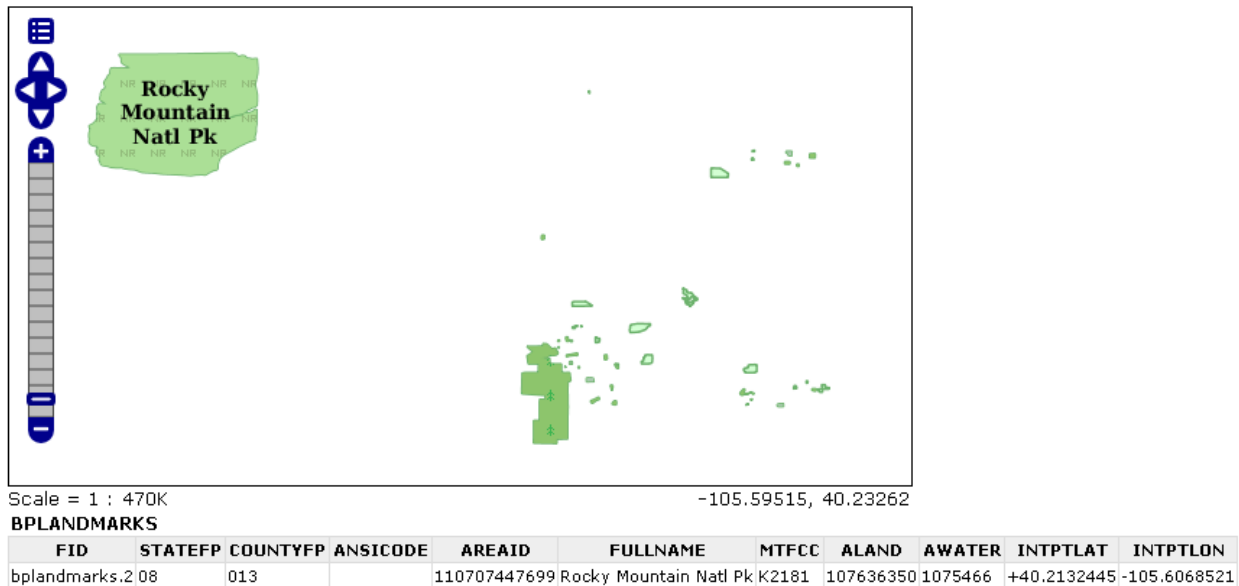


Fig. 224: Default GetFeatureInfo request

3. In order to configure a custom template of the GetFeatureInfo results create three .ftl files in *\$GEOSERVER_DATA_DIR/workspaces/geosolutions* directory named:

```
- header.ftl
- content.ftl
- footer.ftl
```

Note: The Template is managed using [Freemarker](#). This is a simple yet powerful template engine that GeoServer uses whenever developers allowed user customization of textual outputs. In particular, at the time of writing it's used to allow customization of GetFeatureInfo, GeoRSS and KML outputs.

Note: Splitting the template in three files allows the administrator to keep a consistent styling for the GetFeatureInfo result, but use different templates for different workspaces or different layers: this is done by providing a master `header.ftl` and `footer.ftl` file, but specify a different `content.ftl` for each layer.

4. In *header.ftl* file enter the following HTML:

```
<!--
Header section of the GetFeatureInfo HTML output. Should have the <head> section,
and
a starter of the <body>. It is advised that eventual CSS uses a special class for
featureInfo,
```

(continues on next page)

(continued from previous page)

```

since the generated HTML may blend with another page changing its aspect when
↳using generic classes
like td, tr, and so on.
-->
<html>
  <head>
    <title>Geoserver GetFeatureInfo output</title>
  </head>
  <style type="text/css">
    table.featureInfo, table.featureInfo td, table.featureInfo th {
      border:1px solid #ddd;
      border-collapse:collapse;
      margin:0;
      padding:0;
      font-size: 90%;
      padding:.2em .1em;
    }

    table.featureInfo th{
      padding:.2em .2em;
      text-transform:uppercase;
      font-weight:bold;
      background:#eee;
    }

    table.featureInfo td{
      background:#fff;
    }

    table.featureInfo tr.odd td{
      background:#eee;
    }

    table.featureInfo caption{
      text-align:left;
      font-size:100%;
      font-weight:bold;
      text-transform:uppercase;
      padding:.2em .2em;
    }
  </style>
</body>

```

5. In *content.ftl* file enter the following HTML:

```

<ul>
<#list features as feature>
  <li><b>Type: ${type.name}</b> (id: <em>${feature.fid}</em>):
    <ul>
    <#list feature.attributes as attribute>
      <#if !attribute.isGeometry>
        <li>${attribute.name}: ${attribute.value}</li>
      </#if>
    </#list>
    </ul>
  </li>
</#list>

```

(continues on next page)

(continued from previous page)

```
</ul>
```

6. In *footer.ftl* file enter the following HTML:

```
<!--
Footer section of the GetFeatureInfo HTML output. Should close the body and the
html tag.
-->
    </body>
</html>
```

7. Go to the Map Preview to show *geosolutions:bplandmarks* layer.

8. Click on the *Rocky Mountain Natl Park* region in the OpenLayers map to show the new FeatureInfo representation.



- **Type: bplandmarks** (id: *bplandmarks.2*):
 - ◊ STATEFP: 08
 - ◊ COUNTYFP: 013
 - ◊ ANSICODE:
 - ◊ AREAID: 110707447699
 - ◊ FULLNAME: Rocky Mountain Natl Pk
 - ◊ MTFCC: K2181
 - ◊ ALAND: 107636350
 - ◊ AWATER: 1075466
 - ◊ INTPTLAT: +40.2132445
 - ◊ INTPTLON: -105.6068521

Fig. 225: Custom GetFeatureInfo template

Cross layer filtering with GeoServer

Normal GeoServer operation allows a filter to be applied on each layer in isolation, based on its attribute and external information (geometry, values) provided by the user. Cross layer filtering is instead the ability to select features from one layer that bear some relationship with features coming from another layer. Common questions that cross layer filters can help answering are:

- find all the ice cream stores located in a public park (point vs polygon)
- find all bus stops within 100m from the National Bank subsidiaries (point vs point, with distance reference)
- find all coastal roads (line VS polygon, assuming we have a set of polygons representing the water areas)

In order to solve these questions with a vanilla GeoServer a client would have to first use WFS to gather all the geometries satisfying the base conditions (e.g., find the National Bank Subsidiaries), load and unite them, and then issue a second request to the server in order to get the data from the other layer (e.g., the bus stops within 100m from the previously loaded points).

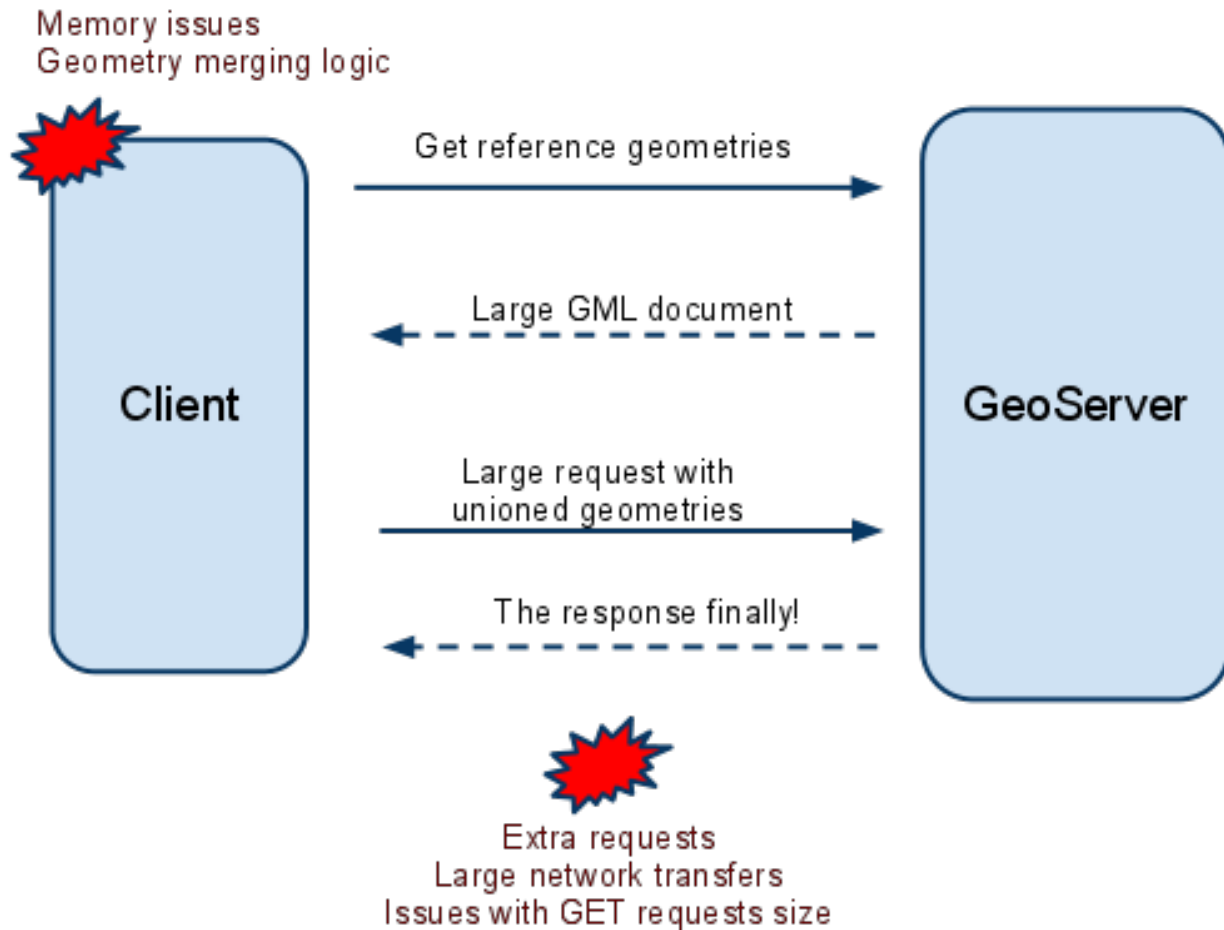


Fig. 226: Round trips without cross layer filtering

The querylayer module

The querylayer extension, already installed in the workshop GeoServer instance, provides three new filter functions that can be used to avoid the client/server extra round trips, and have the server handle the secondary geometries collection instead.

Name	Arguments	Description
querySingle	layer: String, attribute:String, filter:String	Queries the specified layer applying the specified (E)CQL filter and returns the value of attribute from the first feature in the result set. The layer name should be qualified (e.g. topp:states), the filter can be INCLUDE if no filtering is desired
queryCollection	layer: String, attribute:String, filter:String	Queries the specified layer applying the specified (E)CQL filter and returns the list of the values from attribute out of every single feature in the result set. The layer name should be qualified (e.g. topp:states), the filter can be INCLUDE if no filtering is desired. Will throw an exception if too many results are being collected (see the memory limits section for details)
collectGeometries	geometries: a list of Geometry objects	Turns the list of geometries into a single Geometry object, suitable for being used as the reference geometry in spatial filters. Will throw an exception if too many coordinates are being collected (the results of queryCollection cannot be used as is)

These filter functions can be used directly in CQL filters, OGC filters and SLD, meaning they are available both from WMS and WFS.

Finding all polygonal landmarks crossing a trail

The following map, obtained using the WMS reflector to keep the URL short, shows all polygonal landmarks and trails in Boulder (trails are visible when zooming-in due to scale dependencies):

```
http://localhost:8083/geoserver/geosolutions/wms/reflect?
↳layers=geosolutions:bplandmarks,Trails&format=application/openlayers&width=512&
↳height=512&BBOX=-105.31,39.97,-105.26,40.2
```

Now, let's assume we want to find all polygonal landmarks crossing any trail using the above filter functions. The first step would be to locate all the trails and extract their geometry attribute (the_geom):

```
queryCollection('Trails', 'the_geom', 'INCLUDE')
```

The above builds a list of geometries that we want to turn into a single MULTILINESTRING, in order to use it as a reference for a INTERSECTS filter. So we'll call collectGeometries:

```
collectGeometries(queryCollection('Trails', 'the_geom', 'INCLUDE'))
```

Now that we have all the trails in a single geometry object we can use it to build a intersection filter with the polygonal landmarks:

```
INTERSECTS(the_geom, collectGeometries(queryCollection('Trails', 'the_geom', 'INCLUDE
↳'))))
```

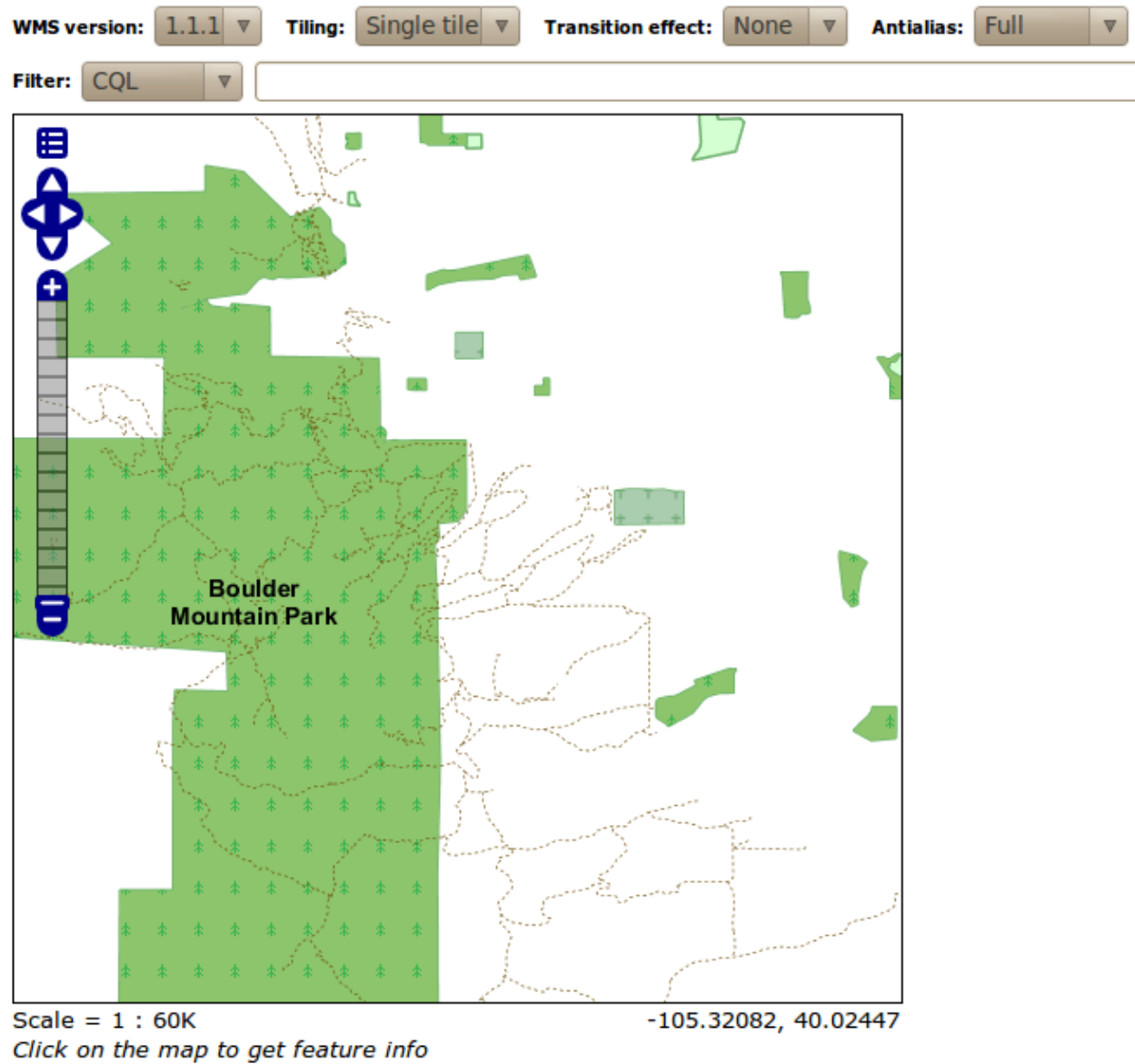


Fig. 227: Polygonal landmarks and trails in Boulder

Since the map contains two layers and we only want to filter on the first, the final CQL filter used in the GetMap request will be:

```
INTERSECTS(the_geom, collectGeometries(queryCollection('Trails', 'the_geom', 'INCLUDE
↳'))) ; INCLUDE
```

The result is that only two polygonal landmarks, the Boulder Mountain Park, and the smaller Buckingham Park, cross any trail:

Finding all buildings located inside a park

In this case we'll start with [this map](#):

```
http://localhost:8083/geoserver/geosolutions/wms/reflect?
↳layers=geosolutions:bplandmarks,bbuildings&format=application/openlayers&width=512&
↳height=512&&BBOX=-105.29,40.01,-105.28,40.02
```

The filter construction is similar to the previous case, but this time we need to collect geometries only from parks, which have a MTFCC attribute equals to K2180:

```
INCLUDE;INTERSECTS(the_geom, collectGeometries(queryCollection('bplandmarks', 'the_
↳geom', 'MTFCC = 'K2180')))
```

Finding all buildings close enough to the Boulder County Courthouse

In this case we want to find all the buildings close to the Boulder County Courthouse. The [reference map](#) this time is:

```
http://localhost:8083/geoserver/geosolutions/wms/reflect?
↳layers=geosolutions:bptlandmarks,bbuildings&format=application/openlayers&width=512&
↳height=512&&BBOX=-105.28061758059,40.016146865234,-105.27475307863,40.021151240234
```

This will extract a single geometry that we'll use as a reference, so this time we are going to use the `querySingle` function instead, and use the `DWITHIN` function to locate all buildings within 400 feet from the courthouse:

```
INCLUDE;DWITHIN(the_geom, querySingle('bptlandmarks', 'the_geom', 'FULLNAME = '
↳'Boulder County Courthouse')), 400, feet)
```

the resulting map is going to be:

Advanced Raster Data Management

Introduction To Processing With GDAL Utilities

In the [Adding a GeoTiff](#) section, a GeoTIFF file has been added to GeoServer as is. However, it's common practice to do a preliminary analysis on the available data and, if needed, optimize it since configuring big datasets without proper pre-processing, may result in low performance when accessing them. In this section, instructions about how to do data optimization will be provided by introducing some FWTools Utilities.

Note: On a *Windows* machine you can set-up a shell with all GDAL Utilities opening a terminal and running the file `setenv.bat` under the `%TRAINING_ROOT%` folder. This operation must be repeated whenever a new terminal window is open. Alternatively run directly the file `gdal.bat` under the `%TRAINING_ROOT%` folder.

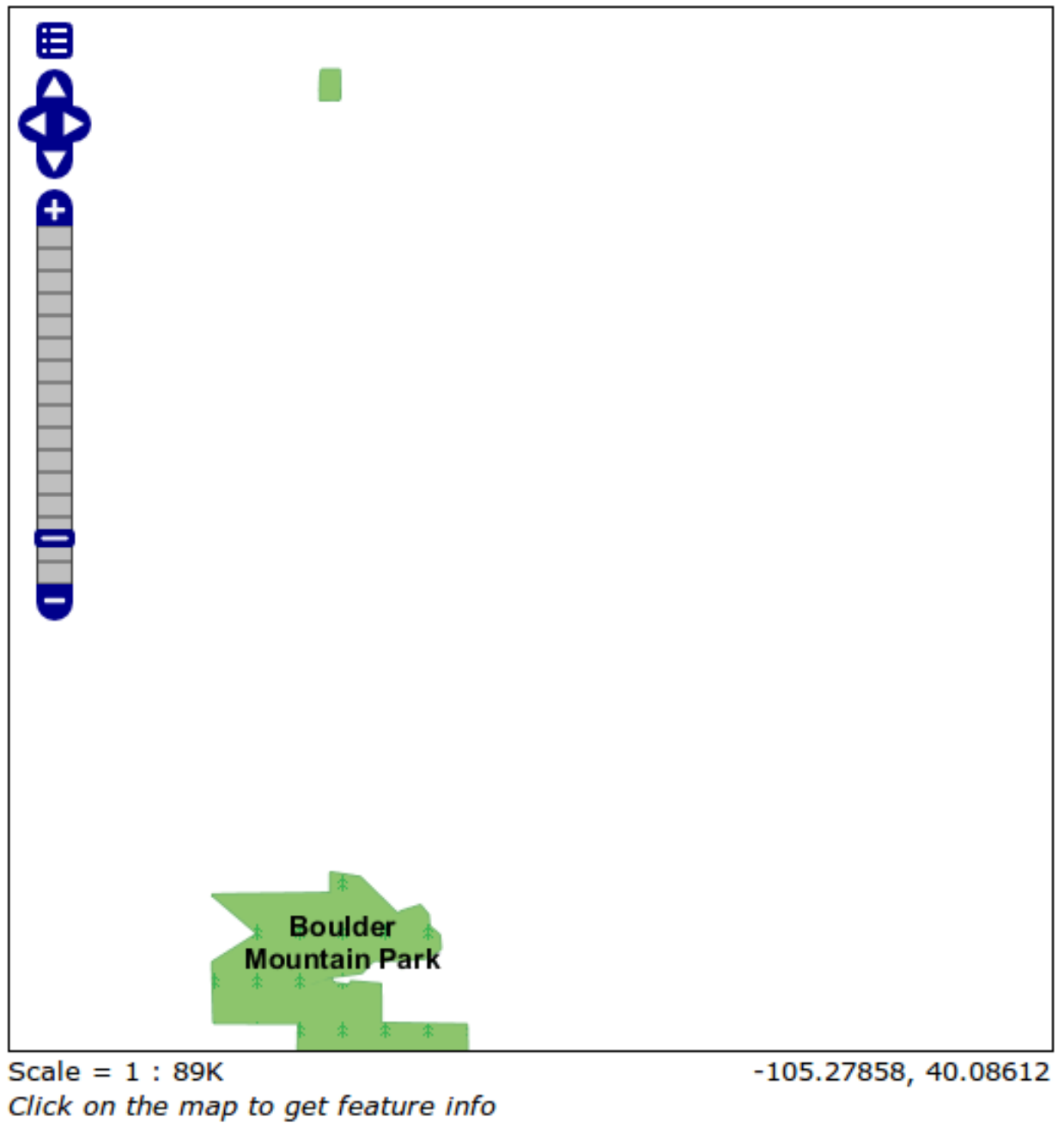


Fig. 228: Polygonal landmarks intersecting trails in Boulder



Fig. 229: Buildings and parks in Boulder

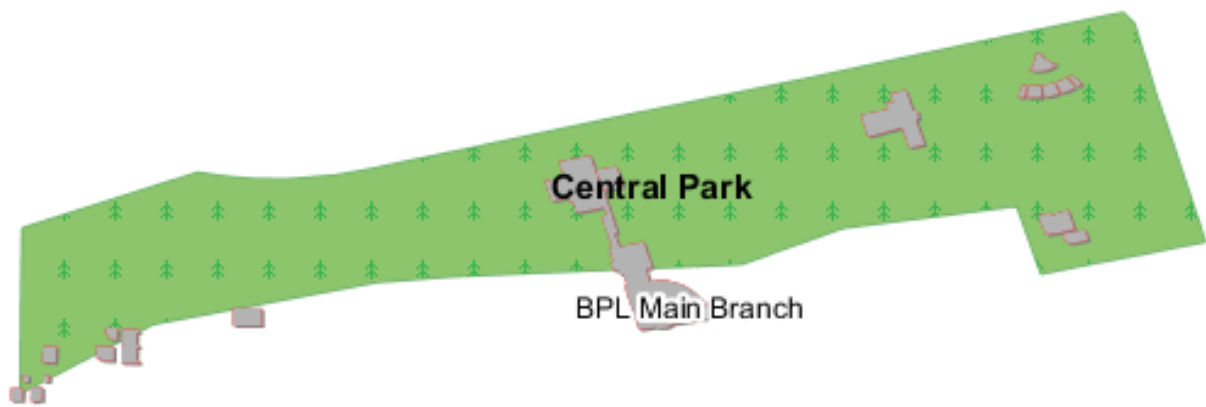


Fig. 230: Buildings inside parks in Boulder

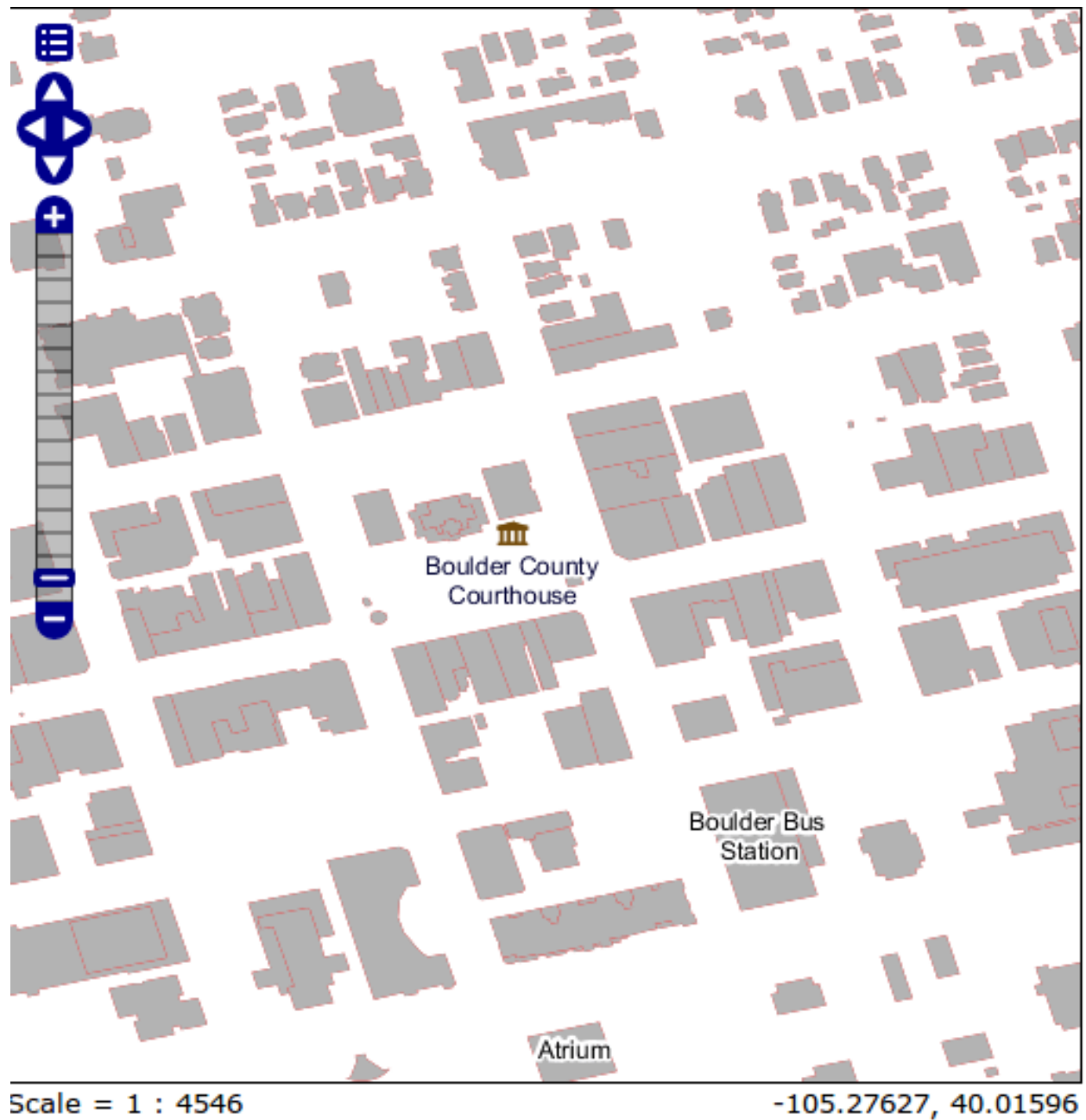


Fig. 231: Boulder County Courthouse surrounded by buildings

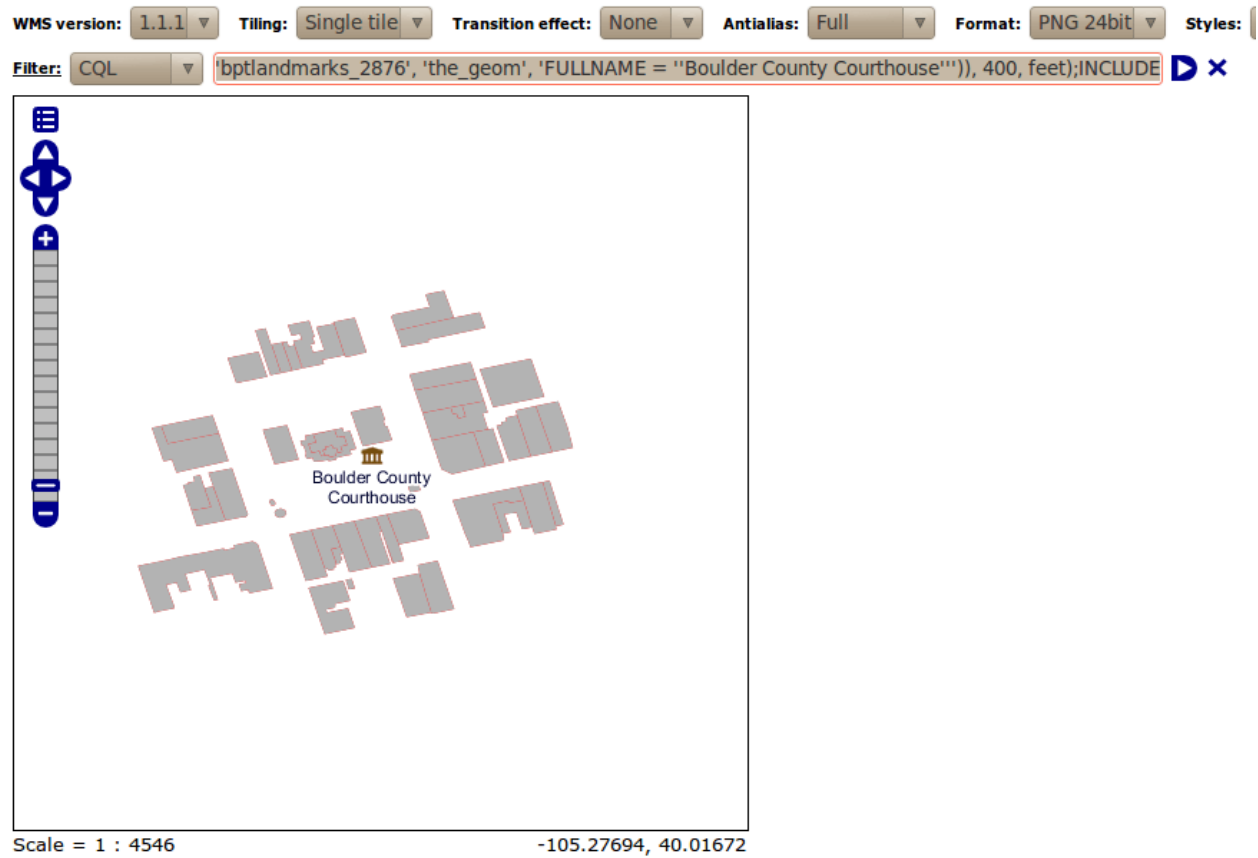


Fig. 232: Buildings close to the Boulder County Courthouse

gdalinfo

This utility allows to get several info from the GDAL library, for instance, specific Driver capabilities and input Datasets/Files properties.

gdalinfo - Getting Drivers Capabilities

Being GeoTIFF a widely adopted geospatial format, it's useful to get information about the GDAL GeoTIFF's Driver capabilities using the command:

```
gdalinfo --format GTIFF
```

This is only a trimmed down version of a typical output:

```

Format Details:
  Short Name: GTiff
  Long Name: GeoTIFF
  Extension: tif
  Mime Type: image/tiff
  Help Topic: frmt_gtiff.html
  Supports: Create() - Create writeable dataset.
  Supports: CreateCopy() - Create dataset by copying another.
  Supports: Virtual IO - eg. /vsimem/
  Creation Datatypes: Byte UInt16 Int16 UInt32 Int32 Float32 Float64 CInt16 CInt32_
↳ CFloat32 CFloat64
  <CreationOptionList>
    <Option name="COMPRESS" type="string-select">
      <Value>NONE</Value>
      <Value>LZW</Value>
      <Value>PACKBITS</Value>
      <Value>JPEG</Value>
      <Value>CCITTRLE</Value>
      <Value>CCITTFAX3</Value>
      <Value>CCITTFAX4</Value>
      <Value>DEFLATE</Value>
    </Option>
    <Option name="PREDICTOR" type="int" description="Predictor Type" />
    <Option name="JPEG_QUALITY" type="int" description="JPEG quality 1-100" default="75
↳ "/>
    <Option name="ZLEVEL" type="int" description="DEFLATE compression level 1-9"
↳ default="6" />
    <Option name="LZMA_PRESET" type="int" description="LZMA compression level 0(fast)-
↳ 9(slow)" default="6" />
    <Option name="NBITS" type="int" description="BITS for sub-byte files (1-7), sub-
↳ uint16 (9-15), sub-uint32 (17-31)" />
    <Option name="INTERLEAVE" type="string-select" default="PIXEL">
      <Value>BAND</Value>
      <Value>PIXEL</Value>
    </Option>
    <Option name="TILED" type="boolean" description="Switch to tiled format"/>
    <Option name="TFW" type="boolean" description="Write out world file"/>
    <Option name="RPB" type="boolean" description="Write out .RPB (RPC) file" />
    <Option name="BLOCKXSIZE" type="int" description="Tile Width"/>
    <Option name="BLOCKYSIZE" type="int" description="Tile/Strip Height"/>
    <Option name="PHOTOMETRIC" type="string-select">
      <Value>MINISBLACK</Value>

```

(continues on next page)

(continued from previous page)

```

    <Value>MINISWHITE</Value>
    <Value>PALETTE</Value>
    <Value>RGB</Value>
    <Value>CMYK</Value>
    <Value>YCBCR</Value>
    <Value>CIELAB</Value>
    <Value>ICCLAB</Value>
    <Value>ITULAB</Value>
  </Option>
  <Option name="SPARSE_OK" type="boolean" description="Can newly created files have_
↳missing blocks?" default="FALSE" />
  <Option name="ALPHA" type="boolean" description="Mark first extrasample as being_
↳alpha" />
  <Option name="PROFILE" type="string-select" default="GDALGeoTIFF">
    <Value>GDALGeoTIFF</Value>
    <Value>GeoTIFF</Value>
    <Value>BASELINE</Value>
  </Option>
  <Option name="PIXELTYPE" type="string-select">
    <Value>DEFAULT</Value>
    <Value>SIGNEDBYTE</Value>
  </Option>
  <Option name="BIGTIFF" type="string-select" description="Force creation of BigTIFF_
↳file">
    <Value>YES</Value>
    <Value>NO</Value>
    <Value>IF_NEEDED</Value>
    <Value>IF_SAFER</Value>
  </Option>
  <Option name="ENDIANNESS" type="string-select" default="NATIVE" description="Force_
↳endianness of created file. For DEBUG purpose mostly">
    <Value>NATIVE</Value>
    <Value>INVERTED</Value>
    <Value>LITTLE</Value>
    <Value>BIG</Value>
  </Option>
  <Option name="COPY_SRC_OVERVIEWS" type="boolean" default="NO" description="Force_
↳copy of overviews of source dataset (CreateCopy())" />
</CreationOptionList>

```

From the above list of create options it's possible to determine the main GeoTIFF Driver's writing capabilities:

- COMPRESS: customize the compression to be used when writing output data
- JPEG_QUALITY: specify a quality factor to be used by the JPEG compression
- TILED: When set to YES it allows to tile output data
- BLOCKXSZ, BLOCKYSZ: Specify the Tile dimension width and Tile dimension height
- PHOTOMETRIC: Specify the photometric interpretation of the data
- PROFILE: Specify the GeoTIFF profile to be used (some profiles only support a minimal set of TIFF Tags while some others provide a wider range of Tags)
- BIGTIFF: Specify when to write data as BigTIFF (A TIFF format which allows to break the 4GB Offset boundary)

gdalinfo - Getting Dataset/File Properties

The following instructions allow you to get information about the sample dataset previously configured in GeoServer.

1. Run:

```
* Linux::

cd ${TRAINING_ROOT}/data/user_data/aerial

gdalinfo 13tde815295_200803_0x6000m_cl.tif

* Windows::

setenv.bat

cd %TRAINING_ROOT%\data\user_data\airial\

gdalinfo 13tde815295_200803_0x6000m_cl.tif
```

```
Driver: GTiff/GeoTIFF
Files: 13tde815295_200803_0x6000m_cl.tif
Size is 2500, 2500
Band 1 Block=2500x16 Type=Byte, ColorInterp=Red
Band 2 Block=2500x16 Type=Byte, ColorInterp=Green
Band 3 Block=2500x16 Type=Byte, ColorInterp=Blue
```

Fig. 233: Part of the *gdalinfo* output on a sample dataset

2. Check the **Block** info as well as the **Overviews** info if present.

- **Block:** It represents the internal tiling. Notice that the sample dataset has tiles made of 16 rows having width equals to the full image width.
- **Overviews:** It provides information about the underlying overviews. Notice that the sample dataset doesn't have overviews since the *Overviews* property is totally missing from the *gdalinfo* output.

gdal_translate

This utility allows to convert a dataset to a different format by allowing a wide set of parameters to customize the conversion.

Running the command:

```
gdal_translate
```

allows to get the list of supported parameters as well as the supported output formats:

```
Usage: gdal_translate [--help-general]
      [-ot {Byte/Int16/UInt16/UInt32/Int32/Float32/Float64/
           CInt16/CInt32/CFloat32/CFloat64}] [-strict]
      [-of format] [-b band] [-mask band] [-expand {gray|rgb|rgba}]
      [-outsize xsize[%] ysize[%]]
```

(continues on next page)

(continued from previous page)

```

[- unscale] [-scale [src_min src_max [dst_min dst_max]]]
[-srcwin xoff yoff xsize ysize] [-projwin ulx uly lrx lry]
[-a_srs srs_def] [-a_ullr ulx uly lrx lry] [-a_nodata value]
[-gcp pixel line easting northing [elevation]]*
[-mo "META-TAG=VALUE"]* [-q] [-sds]
[-co "NAME=VALUE"]* [-stats]
src_dataset dst_dataset

```

Where the meaning of the main parameters is summarized below:

- *-ot*: allows to specify the output datatype (Make sure that the specified datatype is contained in the *Creation Datatypes* list of the Writing driver)
- *-of*: specify the desired output format (GTIFF is the default value)
- *-b*: allows to specify an input band to be written in the output file. (Use multiple *-b* option to specify more bands)
- *-mask*: allows to specify an input band to be write an output dataset mask band.
- *-expand*: allows to expose a dataset with 1 band with a color table as a dataset with 3 (rgb) or 4 (rgba) bands. The (gray) value allows to expand a dataset with a color table containing only gray levels to a gray indexed dataset.
- *-outsize*: allows to set the size of the output file in terms of pixels and lines unless the % sign is attached in which case it's as a fraction of the input image size.
- *-unscale*: allows to apply the scale/offset metadata for the bands to convert from scaled values to unscaled ones.
- *-scale*: allows to rescale the input pixels values from the range *src_min* to *src_max* to the range *dst_min* to *dst_max*. (If omitted the output range is 0 to 255. If omitted the input range is automatically computed from the source data).
- *-srcwin*: allows to select a subwindow from the source image in terms of xoffset, yoffset, width and height
- *-projwin*: allows to select a subwindow from the source image by specifying the corners given in georeferenced coordinates.
- *-a_srs*: allows to override the projection for the output file. The *srs_def* may be any of the usual GDAL/OGR forms, complete WKT, PROJ.4, EPSG:n or a file containing the WKT.
- *-a_ullr*: allows to assign/override the georeferenced bounds of the output file.
- *-a_nodata*: allows to assign a specified nodata value to output bands.
- *-co*: allows to set a creation option in the form "NAME=VALUE" to the output format driver. (Multiple *-co* options may be listed.)
- *-stats*: allows to get statistics (min, max, mean, stdDev) for each band
- *src_dataset*: is the source dataset name. It can be either file name, URL of data source or subdataset name for multi*-dataset files.
- *dst_dataset*: is the destination file name.

gdal_translate - Tiling the sample dataset

The following steps provide instructions to tile the sample dataset previously configured in GeoServer, by using the GeoTIFF driver.

1. Create a directory to store the converted data:

- Linux:

```
cd ${TRAINING_ROOT}/data/user_data  
mkdir retiled
```

- Windows:

```
cd %TRAINING_ROOT%\data\user_data  
mkdir retiled
```

2. Convert the input sample data to an output file having tiling set to 512x512. Run:

- Linux:

```
gdal_translate -co "TILED=YES" -co "BLOCKXSIZE=512" -co "BLOCKYSIZE=512" aerial/  
↳13tde815295_200803_0x6000m_cl.tif retiled/13tde815295_200803_0x6000m_cl.tif
```

- Windows:

```
gdal_translate -co "TILED=YES" -co "BLOCKXSIZE=512" -co "BLOCKYSIZE=512"   
↳aerial\13tde815295_200803_0x6000m_cl.tif retiled\13tde815295_200803_0x6000m_cl.  
↳tif
```

3. Optionally, check that the output dataset have been successfully tiled, by running the command:

- Linux:

```
gdalinfo retiled/13tde815295_200803_0x6000m_cl.tif
```

- Windows:

```
gdalinfo retiled\13tde815295_200803_0x6000m_cl.tif
```

```
Driver: GTiff/GeoTIFF  
Files: retiled/13tde815295_200803_0x6000m_cl.tif  
Size is 2500, 2500  
Band 1 Block=512x512 Type=Byte, ColorInterp=Red  
Band 2 Block=512x512 Type=Byte, ColorInterp=Green  
Band 3 Block=512x512 Type=Byte, ColorInterp=Blue
```

Fig. 234: Part of the *gdalinfo* output on the tiled dataset. Notice the **Block** value now is 512x512

gdaladdo

This utility allows to add overviews to a dataset. The following steps provide instructions to add overviews to the tiled sample dataset.

Running the command:

```
gdaladdo
```

allows to get the list of supported parameters:

```
Usage: gdaladdo [-r {nearest,average,gauss,average_mp,average_magphase,mode}]
               [-ro] [--help-general] filename levels
```

Where the meaning of the main parameters is summarized below:

- *-r*: allows to specify the resampling algorithm (Nearest is the default value)
- *-ro*: allows to open the dataset in read-only mode, in order to generate external overview (for GeoTIFF especially)
- *filename*: represents the file to build overviews for.
- *levels*: allows to specify a list of overview levels to build.

gdaladdo - Adding overviews to the sample dataset

1. Run:

- Linux:

```
cd ${TRAINING_ROOT}/data/user_data/retiled
gdaladdo -r average 13tde815295_200803_0x6000m_cl.tif 2 4 8 16 32
```

- Windows:

```
cd %TRAINING_ROOT%\data\user_data\retiled
gdaladdo -r average 13tde815295_200803_0x6000m_cl.tif 2 4 8 16 32
```

to add 5 levels of overviews having 2,4,8,16,32 subsampling factors applied to the original image resolution respectively.

1. Optionally, check that the overviews have been added to the dataset, by running the command:

```
gdalinfo 13tde815295_200803_0x6000m_cl.tif
```

```
Driver: GTiff/GeoTIFF
Files: 13tde815295_200803_0x6000m_cl.tif
Size is 2500, 2500
Band 1 Block=512x512 Type=Byte, ColorInterp=Red
  Overviews: 1250x1250, 625x625, 313x313, 157x157, 79x79
Band 2 Block=512x512 Type=Byte, ColorInterp=Green
  Overviews: 1250x1250, 625x625, 313x313, 157x157, 79x79
Band 3 Block=512x512 Type=Byte, ColorInterp=Blue
  Overviews: 1250x1250, 625x625, 313x313, 157x157, 79x79
```

Fig. 235: Part of the *gdalinfo* output on the tiled dataset with overviews. Notice the **Overviews** properties

Process in bulk

Instead of manually repeating these 2 steps (retile + add overviews) for each file, we can invoke a few commands to get it automated.

1. Run:

• Linux:

```
cd ${TRAINING_ROOT}/data/user_data

mkdir optimized

cd aerial

for i in `find *.tif`; do gdal_translate -CO "TILED=YES" -CO "BLOCKXSIZE=512" -CO
↪ "BLOCKYSIZE=512" $i ../optimized/$i; gdaladdo -r average ../optimized/$i 2 4 8
↪ 16 32; done
```

• Windows:

```
cd %TRAINING_ROOT%\data\user_data\

mkdir optimized

cd aerial

    notepad optimize.bat
```

will open a text editor. Copy the following content:

```
for %%F in (*.tif) do (
    echo Processing file %%F

    REM translate
    echo Performing gdal_translate on file %%F to file %%~nF.tiff
    gdal_translate -co "TILED=YES" -co "BLOCKXSIZE=512" -co "BLOCKYSIZE=512" -
↪ co "COMPRESS=DEFLATE" %%F ..\optimized\%%~nF.tiff

    REM add overviews
    echo Adding overviews on file %%~nF.tiff
    gdaladdo -r average --config COMPRESS_OVERVIEW DEFLATE ..\optimized\%%~nF.
↪ tiff 2 4 8 16 32
)
```

Then save the file and run the created .bat file:

```
optimize.bat
```

1. You should see a list of run like this:

```
...
Input file size is 2500, 2500
0...10...20...30...40...50...60...70...80...90...100 - done.
0...10...20...30...40...50...60...70...80...90...100 - done.
Input file size is 2500, 2500
0...10...20...30...40...50...60...70...80...90...100 - done.
```

(continues on next page)

(continued from previous page)

```
0...10...20...30...40...50...60...70...80...90...100 - done.
Input file size is 2500, 2500
0...10...20...30...40...50...60...70...80...90...100 - done.
0...10...20...30...40...50...60...70...80...90...100 - done.
...
```

Warning: This process can take some seconds.

At this point optimized datasets have been prepared and they are ready to be served by GeoServer as an ImageMosaic.

gdalwarp

This utility allows to warp and reproject a dataset. The following steps provide instructions to reproject the aerial dataset (which has “EPSG:26913” coordinate reference system) to WGS84 (“EPSG:4326”).

Running the command:

```
gdalwarp
```

allows to get the list of supported parameters:

```
Usage: gdalwarp [--help-general] [--formats]
[-s_srs srs_def] [-t_srs srs_def] [-to "NAME=VALUE"]
[-order n | -tps | -rpc | -geoloc] [-et err_threshold]
[-refine_gcps tolerance [minimum_gcps]]
[-te xmin ymin xmax ymax] [-tr xres yres] [-tap] [-ts width height]
[-wo "NAME=VALUE"] [-ot Byte/Int16/...] [-wt Byte/Int16]
[-srcnodata "value [value...]"] [-dstnodata "value [value...]"] -dstalpha
[-r resampling_method] [-wm memory_in_mb] [-multi] [-q]
[-cutline datasource] [-cl layer] [-cwhere expression]
[-csql statement] [-cblend dist_in_pixels] [-crop_to_cutline]
[-of format] [-co "NAME=VALUE"]* [-overwrite]
srcfile* dstfile
```

Where the meaning of the main parameters is summarized below:

- `-s_srs`: allows to specify the source coordinate reference system
- `-t_srs`: allows to specify the target coordinate reference system
- `-te`: allows to set georeferenced extents (expressed in target CRS) of the output
- `-tr`: allows to specify the output resolution (expressed in target georeferenced units)
- `-ts`: allows to specify the output size in pixel and lines.
- `-r`: allows to specify the resampling method (one of near, bilinear, cubic, cubicspline and lanczos)
- `-srcnodata`: allows to specify band values to be excluded from interpolation.
- `-dstnodata`: allows to specify nodata values on output file.
- `-wm`: allows to specify the amount of memory (expressed in megabytes) used by the warping API for caching.

gdalwarp - Reprojecting sample dataset to WGS84

1. Run:

- Linux:

```
cd ${TRAINING_ROOT}/data/user_data/retiled

gdalwarp -t_srs "EPSG:4326" -co "TILED=YES" 13tde815295_200803_0x6000m_
→cl.tif 13tde815295_200803_0x6000m_cl_warped.tif
```

- Windows:

```
cd %TRAINING_ROOT%/data/user_data/retiled

gdalwarp -t_srs "EPSG:4326" -co "TILED=YES" 13tde815295_200803_0x6000m_
→cl.tif 13tde815295_200803_0x6000m_cl_warped.tif
```

to reproject the specified aerial dataset to WGS84 coordinate reference system.

1. Optionally, check that reprojection has been successful, by running the command:

```
gdalinfo 13tde815295_200803_0x6000m_cl_warped.tif
```

```
Files: 13tde815295_200803_0x6000m_cl_warped.tif
Coordinate System is:
GEOGCS["WGS 84",
  DATUM["WGS_1984",
    SPHEROID["WGS 84",6378137,298.257223563,
      AUTHORITY["EPSG","7030"]],
    AUTHORITY["EPSG","6326"]],
  PRIMEM["Greenwich",0],
  UNIT["degree",0.0174532925199433],
  AUTHORITY["EPSG","4326"]]
Origin = (-105.216821614497917,40.029045462025380)
Pixel Size = (0.000006275236661,-0.000006275236661)
.....
Band 1 Block=256x256 Type=Byte, ColorInterp=Red
Band 2 Block=256x256 Type=Byte, ColorInterp=Green
Band 3 Block=256x256 Type=Byte, ColorInterp=Blue
```

Fig. 236: Part of the *gdalinfo* output on the warped dataset. Notice the updated **Coordinate System** property

In the *next* section, instructions to configure an ImageMosaic will be provided.

Advanced Mosaics and Pyramids Configuration

In this section will learn how to manage Image Mosaics and Image Pyramids in GeoServer.

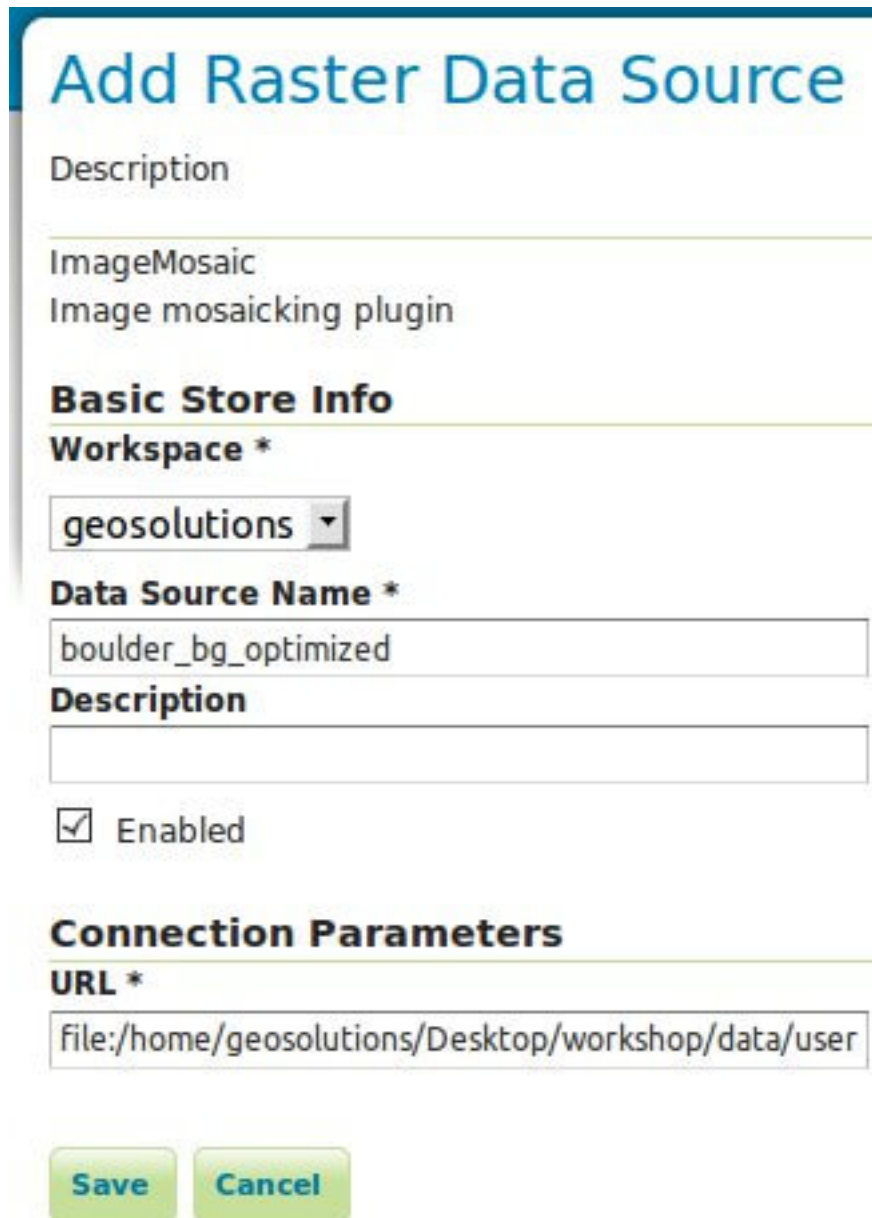
Configuring an Image Mosaic

As introduced in a previous section an Image Mosaic is composed of a set of datasets which are exposed as a single coverage. The ImageMosaic format allows to automatically build and setup a mosaic from a set of georeferenced datasets. This section provides better instructions to configure an Image Mosaic

Note: Before you start, ensure that the *Maps - Raster* section has been completed.

We will configure an ImageMosaic using the optimized dataset. As explained in the *Maps - Raster* section, follow the steps 1 to 4, then at the step 5 fill the fields as explained below.

1. Specify a proper name (as an instance, boulder_bg_optimized) in the *Data Source Name* field of the interface.
2. Specify `file:<TRAINING_ROOT>/data/user_data/optimized` as URL of the sample data in the *Connections Parameter's - URL* field.



Add Raster Data Source

Description

ImageMosaic
Image mosaicking plugin

Basic Store Info

Workspace *

geosolutions ▼

Data Source Name *

boulder_bg_optimized

Description

☒ Enabled

Connection Parameters

URL *

file:/home/geosolutions/Desktop/workshop/data/user

Save **Cancel**

3. Click *Save*.

- Publish the layer by clicking on the *publish* link.

New Layer

Add a new layer

Add layer from

Here is a list of resources contained in the store 'boulder_bg_optimized'. Click on the layer you wish to configure

<< < | > >> Results 0 to 0 (out of 0 items)

Published	Layer name	action
<input checked="" type="checkbox"/>	optimized	Publish

<< < | > >> Results 0 to 0 (out of 0 items)

- Set `boulder_bg_optimized` as name and title of the layer.

Edit Layer

Edit layer data and publishing

geosolutions:optimized

Configure the resource and publishing information for the current layer

Data **Publishing**

Basic Resource Info

Name

Title

- Check the Coordinate Reference Systems and the Bounding Boxes fields are properly set.
- Customize the ImageMosaic properties if needed. For the sample mosaic, set the `OutputTransparentColor` to the value `000000` (Which is the Black color). Click on *Save* when done.
 - AllowMultithreading:** If true, enable tiles multithreading loading. This allows to perform parallelized loading of the granules that compose the mosaic.
 - BackgroundValues:** Set the value of the mosaic background. Depending on the nature of the mosaic it is wise to set a value for the no data area (usually -9999). This value is repeated on all the mosaic bands.
 - Filter:** Filter granules based on attributes from the input coverage.

[Data](#)[Publishing](#)[Dimensions](#)[Tile Caching](#)

Basic Resource Info

Name

☒ Enabled☒ Advertised

Title

Abstract

Keywords

Current Keywords

WCS

ImageMosaic

aerial

[Remove selected](#)

New Keyword

Vocabulary

[Add Keyword](#)

Metadata links

No metadata links so far

[Add link](#)

Note only FGDC and TC211 metadata links show up in WMS 1.1.1 capabilities

Coordinate Reference Systems

Native SRS

[EPSG:NAD83 / UTM zone 13N...](#)

Declared SRS

[Find...](#)[EPSG:NAD83 / UTM zone 13N...](#)

3.2.5 Tutorial

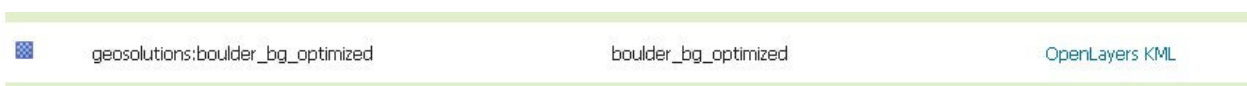
- **InputTransparentColor:** Set the transparent color for the granules prior to mosaicking them in order to control the superimposition process between them. When GeoServer composes the granules to satisfy the user request, some of them can overlap some others, therefore, setting this parameter with the opportune color avoids the overlap of no data areas between granules.
- **MaxAllowedTiles:** Set the maximum number of the tiles that can be load simultaneously for a request. In case of a large mosaic this parameter should be opportunely set to not saturating the server with too many granules loaded at the same time.
- **MergeBehavior:** Merging behaviour for the various granules of the mosaic that GeoServer will produce. This parameter controls whether we want to merge in a single mosaic or stack all the bands into the final mosaic.
- **OutputTransparentColor:** Set the transparent color for the created mosaic.
- **SORTING:** Allow to specify the time order of the obtained granules set. Valid values are DESC (descending) or ASC (ascending). Note that it works just using DBMS as indexes.
- **SUGGESTED_TILE_SIZE:** Controls the tile size of the input granules as well as the tile size of the output mosaic. It consists of two positive integers separated by a comma, like 512,512.
- **USE_JAI_IMAGEREAD:** If true, GeoServer will make use of JAI ImageRead operation and its deferred loading mechanism to load granules; if false, GeoServer will perform direct ImageIO read calls which will result in immediate loading.

At this point the ImageMosaic is being published with GeoServer. Next step is checking how the performances in accessing the datasets have been improved.

1. Click the *Layer Preview* link in the left GeoServer menu.
2. Look for a *geosolutions:boulder_bg* layer (the dataset without optimization) and click the *OpenLayers* link beside of it.



3. Play with the map preview by zooming and panning. When zooming, the response time isn't immediate due to the access to the underlying big datasets which haven't been optimized.
4. Return to the *Layer Preview* page.
5. Look for a *geosolutions:boulder_bg_optimized* layer (the optimized dataset with tiling and overviews set) and click the *OpenLayers* link beside of it.



6. Play with the map preview by zooming and panning:
 - Check how the **performances have been improved** leveraging on both overviews and tiling.
 - Note the **better image quality** of the lowest resolution views, having used an average interpolation algorithm when creating the overviews.

Configuring an Image Pyramid

GeoServer can efficiently deal with large TIFF with overviews, as long as the TIFF is below the 2GB size limit. Once the image size goes beyond such limit it's time to start considering an image pyramid instead. An image pyramid builds multiple mosaics of images, each one at a different zoom level, making it so that each tile is stored in a separate

file. This comes with a composition overhead to bring back the tiles into a single image, but can speed up image handling as each overview is tiled, and thus a sub-set of it can be accessed efficiently (as opposed to a single GeoTIFF, where the base level can be tiled, but the overviews never are).

Note: In order to build the pyramid we'll use the `gdal_retile.py` utility, part of the GDAL command line utilities and available for various operating systems.

1. Navigate to the workshop directory and create the *bmpyramid* directory into the `<TRAINING_ROOT>\data\user_data` directory
2. From the command line run

- Linux:

```
cd $TRAINING_ROOT/data/user_data
mkdir bmpyramid
gdal_retile.py -v -r bilinear -levels 4 -ps 2048 2048 -co "TILED=YES" -
→co "COMPRESS=JPEG" -targetDir bmpyramid bmreduced.tiff
```

- Windows:

```
cd %TRAINING_ROOT%
cd %TRAINING_ROOT%\data\user_data\
mkdir bmpyramid
gdal_retile -v -r bilinear -levels 4 -ps 2048 2048 -co "TILED=YES" -co
→"COMPRESS=JPEG" -targetDir bmpyramid bmreduced.tiff
```

The `gdal_retile.py` user guide provides a detailed explanation for all the possible parameters, here is a description of the ones used in the command line above:

- `-v`: verbose output, allows the user to see each file creation scroll by, thus knowing progress is being made (a big pyramid construction can take hours)
- `-r bilinear`: use bilinear interpolation when building the lower resolution levels. This is key to get good image quality without asking GeoServer to perform expensive interpolations in memory
- `-levels 4`: the number of levels in the pyramid
- `-ps 2048 2048`: each tile in the pyramid will be a 2048x2048 GeoTIFF
- `-co "TILED=YES"`: each GeoTIFF tile in the pyramid will be inner tiled
- `-co "COMPRESS=JPEG"`: each GeoTIFF tile in the pyramid will be JPEG compressed (trades small size for higher performance, try out it without this parameter too)
- `-targetDir bmpyramid`: build the pyramid in the bmpyramid directory. The target directory must exist and be empty
- `bmreduced.tiff`: the source file

This will produce a number of TIFF files in bmpyramid along with the sub-directories 1, 2, 3, and 4.

3. Go to the **Stores** section and add a new Raster Data Source clicking on **ImagePyramid**:

Warning: This assumes the GeoServer image pyramid plug-in is already installed. The pyramid is normally an extension.

Raster Data Sources

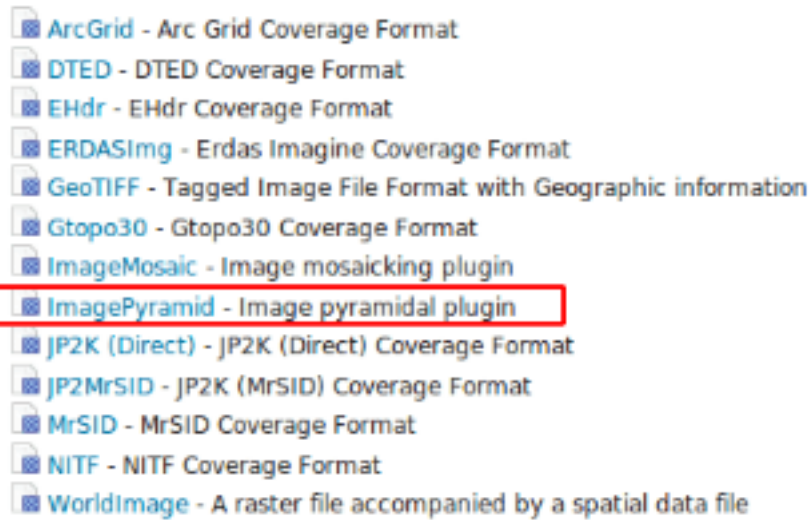


Fig. 237: Adding a ImagePyramid Data Source

If the ImagePyramid store is not available, before doing the exercise install the geoserver pyramid plugin from `%TRAINING_ROOT%/data/plugins/`. Just decompress the zip file into `%TRAINING_ROOT%/webapps/geoserver/WEB-INF/lib/` and restart GeoServer.

- Specify a proper name (bm_pyramid) in the Data Source Name field of the interface and specify a proper URL with the pyramid data directory

- Windows:

```
file:%TRAINING_ROOT%/data/user_data/bmpyramid
```

- Linux:

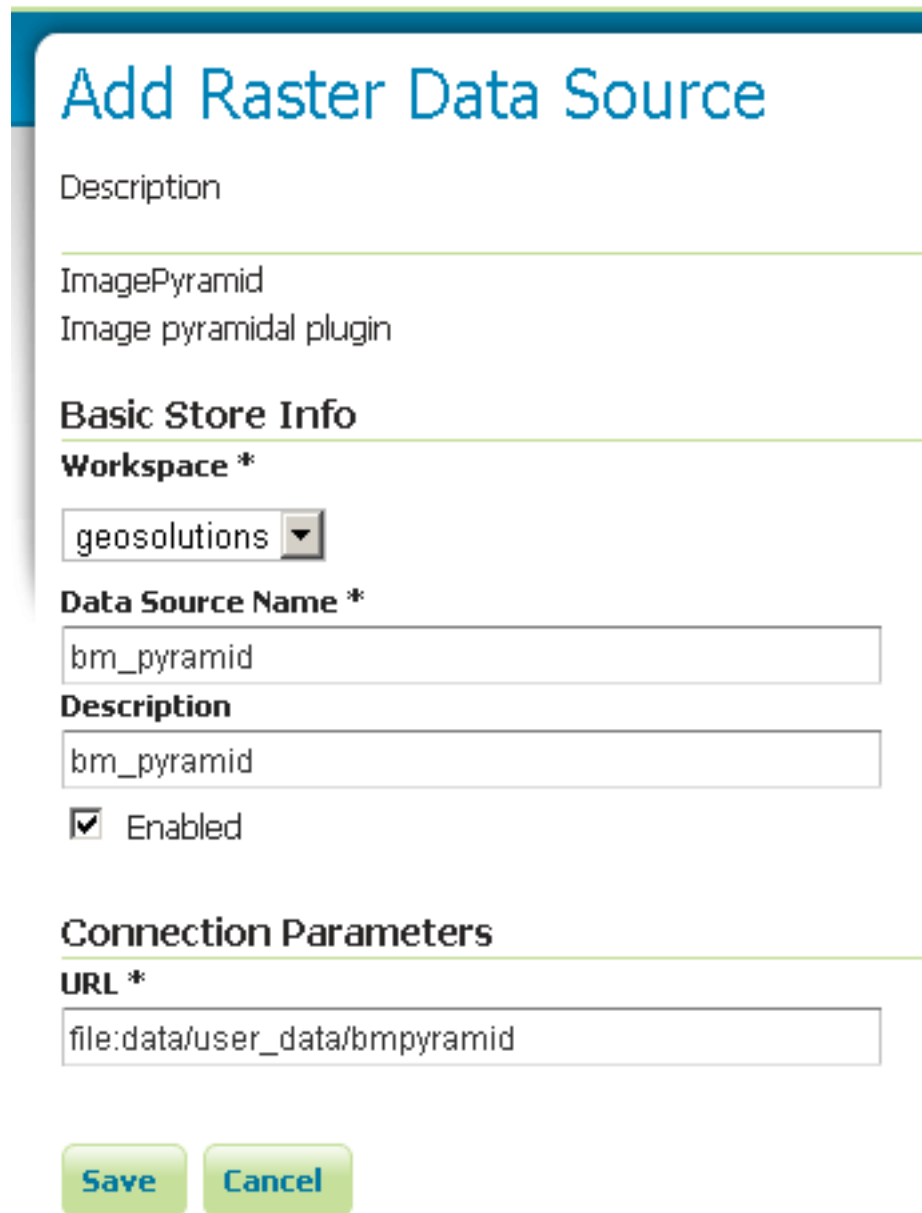
```
file:/home/geosolutions/Desktop/geoserver_training/data/user_data/  
↪bmpyramid
```

- Click the **Save** button.

Note: When clicking save the store will look into the directory, recognize a `gdal_retile` generated structure and perform some background operations:

```
- move all tiff files in the root to a newly create directory 0  
- create an image mosaic in all sub-directories (shapefile index plus property_  
↪file)  
- create the root property file describing the whole pyramid structure
```

- Publish the new pyramid created:



Add Raster Data Source

Description

ImagePyramid
Image pyramidal plugin

Basic Store Info

Workspace *

geosolutions ▼

Data Source Name *

bm_pyramid

Description

bm_pyramid

☒ Enabled

Connection Parameters

URL *

file:data/user_data/bmpyramid

Save **Cancel**

Fig. 238: *Configuring a image pyramid store*

New Layer chooser

Add layer from

Here is a list of resources contained in the store 'bm_pyramid'. Click on the layer you wish to configure

<< < 1 > >>

Results 0 to 0 (out of 0 items)

Published	Layer name	
	bmpyramid	Publish

<< < 1 > >>

Results 0 to 0 (out of 0 items)

Fig. 239: *Choosing the coverage for publishing*

7. Setup the layer parameter **USE_JAI_IMAGEREAD** to false to get better scalability: as told before the image loading using ImageRead is done using the JAI deferred mode so the data will be really loaded when are needed. This may cause many idle open ImageReaders, in case of having to deal with big pyramids (lots of granules over many levels) and it could cause performances issues.
8. Click **Submit** button and go to the GeoServer **Map Preview** to see the pyramid:

Using the ImageMosaic plugin with footprint management

Introduction

This section describes how to associate a vector footprint to a raster dataset in GeoServer using the ImageMosaic plugin.

A vector footprint is a shape used as a mask for the mosaic. Masking can be useful for hiding pixels which are meaningless, or for enhancing only few regions of the image in respect to others.

This chapter contains two sub-sections:

- The first sub-section, **Configuration**, describes the possible configurations needed to set up an ImageMosaic with footprint.
- The second sub-section, **Examples**, provides examples of configuration of an ImageMosaic with footprint.

Configuration

A vector footprint can be linked to an ImageMosaic in three different ways:

1. By using for each mosaic granule a **Sidecar File**, a Shapefile with the same filename of the granule which contains the footprint for it;
2. By using a single Shapefile called **footprints.shp** which contains all the footprints for each granule; each feature contained in the shapefile represent a footprint for an ImageMosaic granule. Each footprint is associated to a granule with the *location* attribute;
3. By using a file called **footprints.properties**, this option add more flexibility to the option number 2.

The last option allows to write the following configuration inside the **footprints.properties** file:

Coverage Parameters

AllowMultithreading

BackgroundValues

Fading

InputTransparentColor

MaxAllowedTiles

OutputTransparentColor

SUGGESTED_TILE_SIZE

USE_JAI_IMAGEREAD

Fig. 240: *Tuning the pyramid parameters*

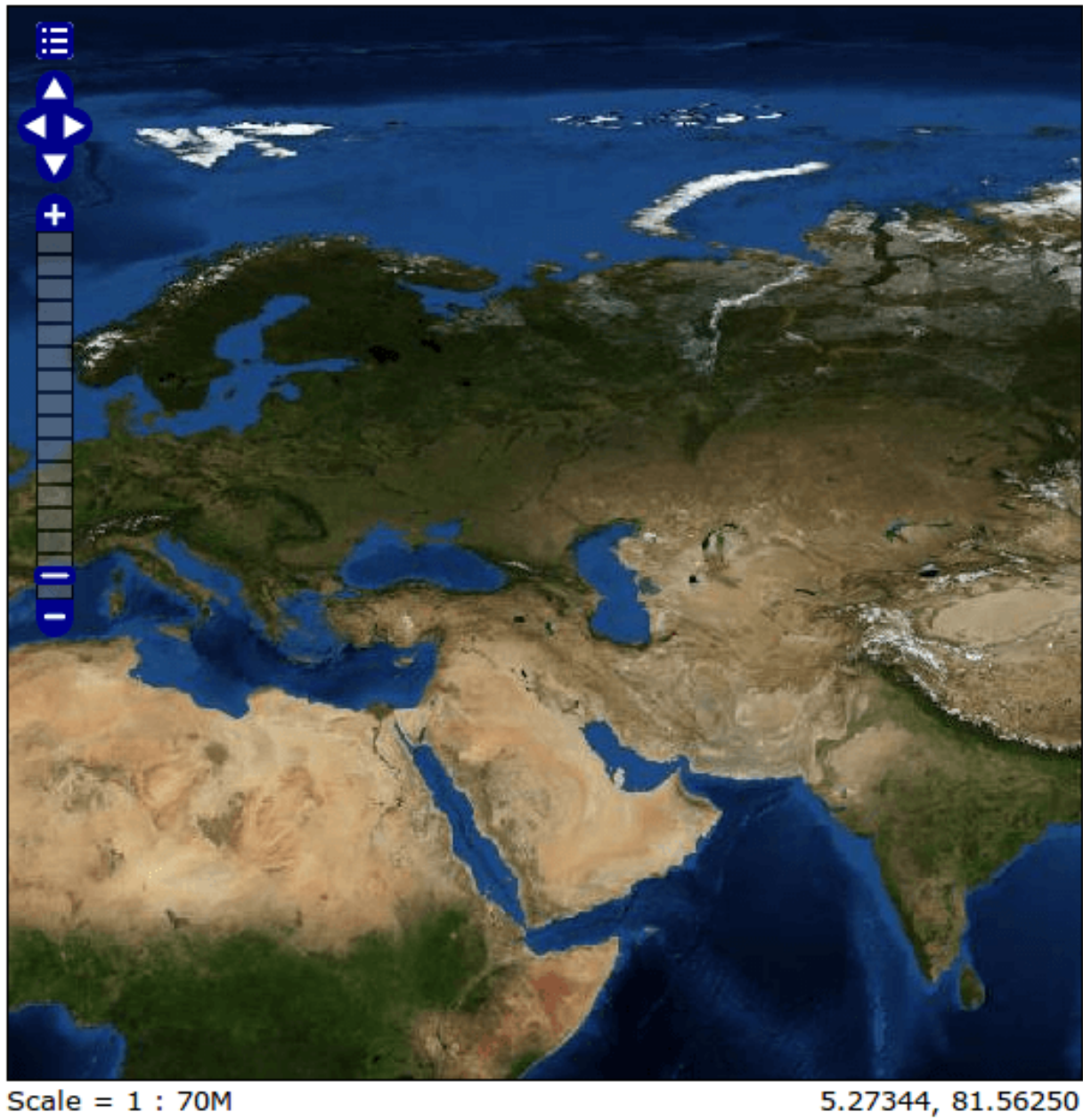


Fig. 241: *Previewing the pyramid*

```
footprint_source=*location of the Shapefile*
footprint_filter=*filter on the Shapefile searching for the attribute associated to_
↳each granule*
```

So the user is free to decide the Shapefile name to use (not only footprints.shp) and the attribute to use for the footprint granule association adding also a custom filter for the content of that attribute.

The **footprint.properties** can be used also to hold other kind of configurations, see the *Inset Support* paragraph below.

For example if a Shapefile called **fakeShapeFile** stores the various footprints in a table like this, where each *Name* attribute is referred to a granule file:

	Name ▾	Shape_Leng	Shape_Area	BUFF_DIST
0	ortho_1-1_1n_s_la087_2010_1	577885.23793900001	4133138857.73999977112	-200.000000000000
1	ortho_2-2_1n_s_la075_2010_1	294998.35735900002	3507709365.03000020981	-200.000000000000
2	ortho_1-1_1n_s_la103_2010_1	315807.69107800000	3903136899.71000003815	-200.000000000000
3	ortho_1-1_1n_s_la071_2010_1	224493.69950700001	1668022813.30999994278	-200.000000000000
4	ortho_1-2_1n_s_la075_2010_1	444899.89763299999	3136445247.32999992371	-200.000000000000
5	ortho_1-1_1n_s_la117_2010_1	235219.58861599999	2778660886.46000003815	-200.000000000000

And the associated granules are:

- ortho_1-1_1n_s_la087_2010_1.tif
- ortho_2-2_1n_s_la075_2010_1.tif
- ortho_1-1_1n_s_la103_2010_1.tif
- and so on ...

The associated **footprints.properties** file must be like this:

```
footprint_source=fakeShapeFile.shp
footprint_filter=Name=strSubstring(granule.location, 0, strLength(granule.location) -
↳4)
```

The **substring** operation is done for comparing the footprint attribute names and the granule names without the *.tif* extension. Standard GeoServer Filter Functions can be use in this expression. A complete reference for them can be found [here](#).

Footprint Behaviours

There are three possible behaviours for Footprint:

- *None*: simply doesn't use the Footprint and behaves like a standard ImageMosaic layer;
- *Transparent*: adds an alpha band of 0s on the image portions outside of the Footprint making them transparent, typically used for RGB data;
- *Cut*: set the background value on the image portions outside of the Footprint, typically used for grayscale data.

The behaviour must be set directly on the Layer configuration page.

Inset Support

Another feature of the *Footprint* is the possibility to calculate an **Inset** on the vector footprint prior to applying it to the image. With the *Inset* a shrinking of the footprint is applied, typically for removing compression artefacts or any nasty effect at the borders. The inset can be activated by adding the following code inside **footprints.properties**:

```
footprint_inset=*value in the shapefile u.o.m.*
footprint_inset_type=*full/border*

* **Full** inset type calculates the inset for each footprint side
* **Border** does the same operation but those straight lines that overlap the image_
↳ bounds are avoided; this last parameter is useful for images already cut in a_
↳ regular grid.
```

Each modification of the **footprints.properties** file requires to *Reload* GeoServer. This operation can be achieved by going to *Server Status* and clicking on the *Reload* button on the bottom-right side of the page.

Examples

The two datasets used in the following can be found into

- Linux: `$TRAINING_ROOT/data/user_data/footprint_data`
- Windows `%TRAINING_ROOT%\data\user_data\footprint_data`

The content of the `footprint_data` is:

- The first dataset, *mosaic_single_tiff*, contains a Shapefile called *srtm_boulder.shp* which represents a mask to use on the Boulder (Colorado) layer inside the `$TRAINING_ROOT/data/user_data/boulder` folder and can be used for testing footprint configuration with a *Sidecar File*.
- The second dataset, *mosaic_sample*, is a mosaic which represents Italy and is used for testing the other two configurations.

Here are presented a few steps for configuring a new ImageMosaic layer with footprint.

1. Vector Footprint configured with a sidecar file

Here the steps to load an ImageMosaic with a sidecar file as a vector footprint.

Step 1: Create a new ImageMosaic Layer

As seen in a previous chapter an ImageMosaic data store can be created by going to *Stores* → *Add New Store* → *ImageMosaic*.

Load the *mosaic_single_tiff* folder, from the `TRAINING_ROOT` folder navigate to `\data\user_data\footprint_data\mosaic_single_tiff`

Publish a Layer from that store going to *Layers* → *Add New Resource*, choosing the name of the data store created above and then clicking on the *publish* button.

Add Raster Data Source

Description

ImageMosaic

Image mosaicking plugin

Basic Store Info

Workspace *

geosolutions ▼

Data Source Name *

mosaic_single_tiff

Description

mosaic_single_tiff

☒ Enabled

Connection Parameters

URL *

file:///D:/DEMO/data/user_data/footprint_data/mosaic_si [Browse...](#)

Save

Cancel

Step 2: Configuring a new Layer for the Mosaic

Warning: fill the field **Declared CRS** with the value `EPSG:4326` if the CRS is not automatically set.

The layer will be rendered depending on the value of the *FootprintBehavior* field:

The user can set one of the three values for the Footprint behaviour as described above (*None*, *Transparent*, *Cut*).

After that, the user must confirm the modification by clicking on the *Save* button on the bottom side of the page.

Step 3: Example Results

Here are presented the results for each dataset.

This is an example of mosaic (*mosaic_single_tiff*) without applying Footprint:

And this is the result of setting **FootprintBehavior** to *Cut*:

Then navigate the filesystem in the mosaic directory, open (or create it if not exist) the file `footprints.properties` and write:

```
footprint_inset=0.01
footprint_inset_type=full
```

to add an inset.

Note: Remember that each modification on **footprints.properties** requires a GeoServer catalog and a GeoServer resource cache reloading in order to apply the changes.

If an Inset is added, the final mosaic is:

2. Vector Footprint configured with `footprints.shp`

Repeat the steps described above but loading the *mosaic_sample* folder from `%TRAINING_ROOT%\data\user_data\footprint_data\mosaic_sample`

This is another example of mosaic (*mosaic_sample*) without Footprint:

And now after setting **FootprintBehavior** to *Transparent* (no Inset is used) on the Layer:

Coverage Parameters

Accurate resolution computation

AllowMultithreading

BackgroundValues

Filter

FootprintBehavior

InputTransparentColor

MaxAllowedTiles

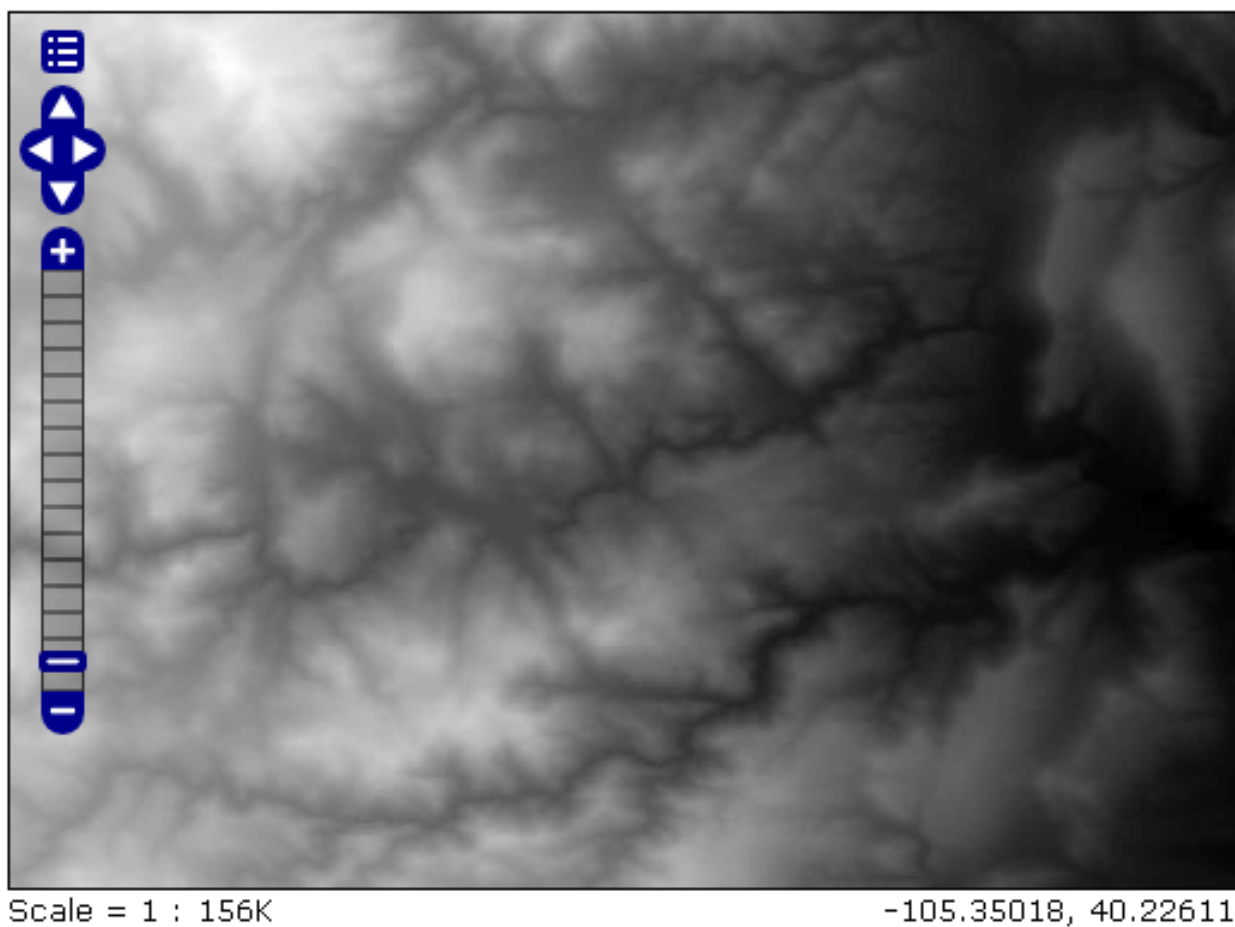
MergeBehavior

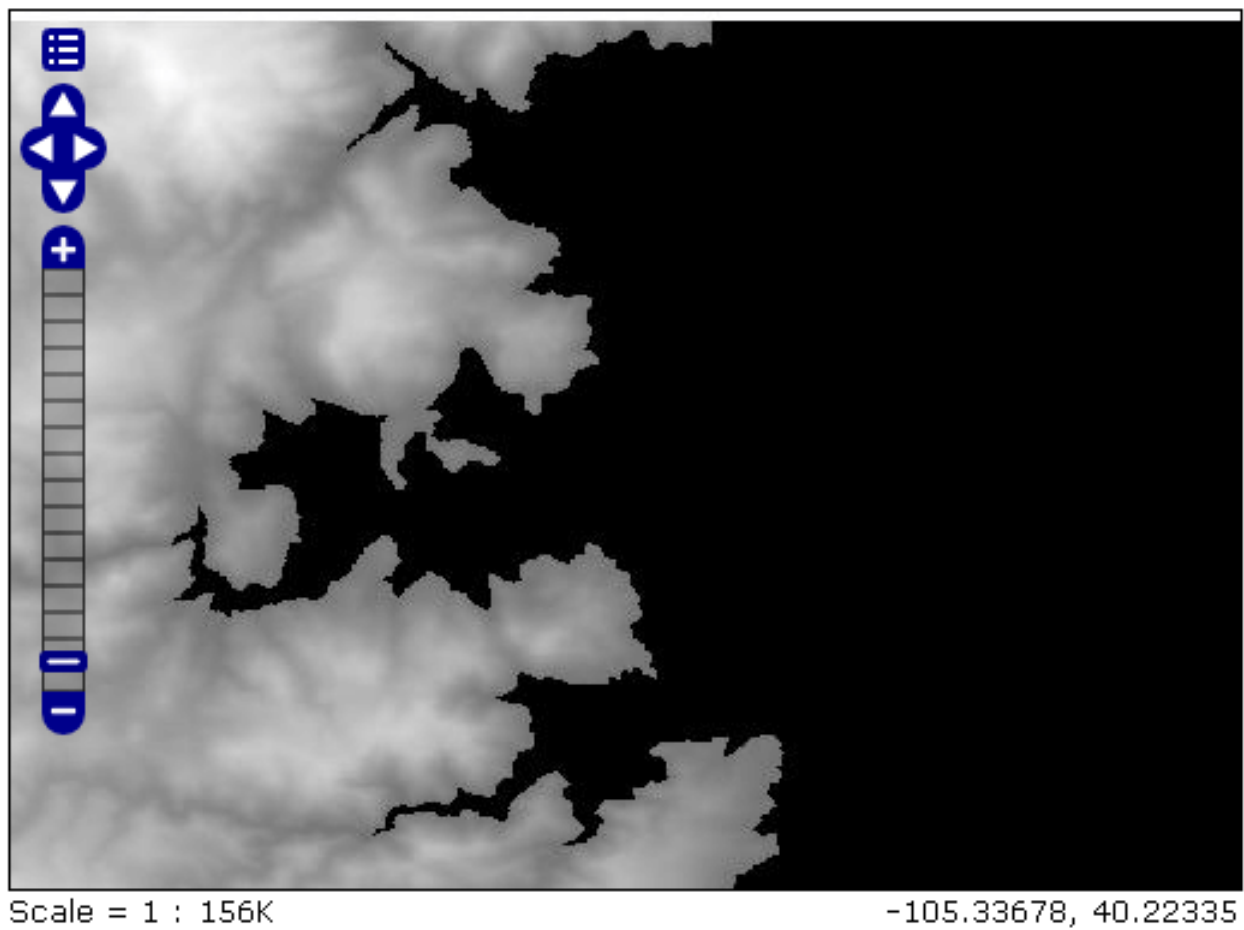
OutputTransparentColor

SORTING

SUGGESTED_TILE_SIZE

USE_JAI_IMAGEREAD











Scale = 1 : 22M

28.01486, 49.64588

Click on the map to get feature info

3. Vector Footprint configured with *footprints.properties*

For testing this functionality the user must

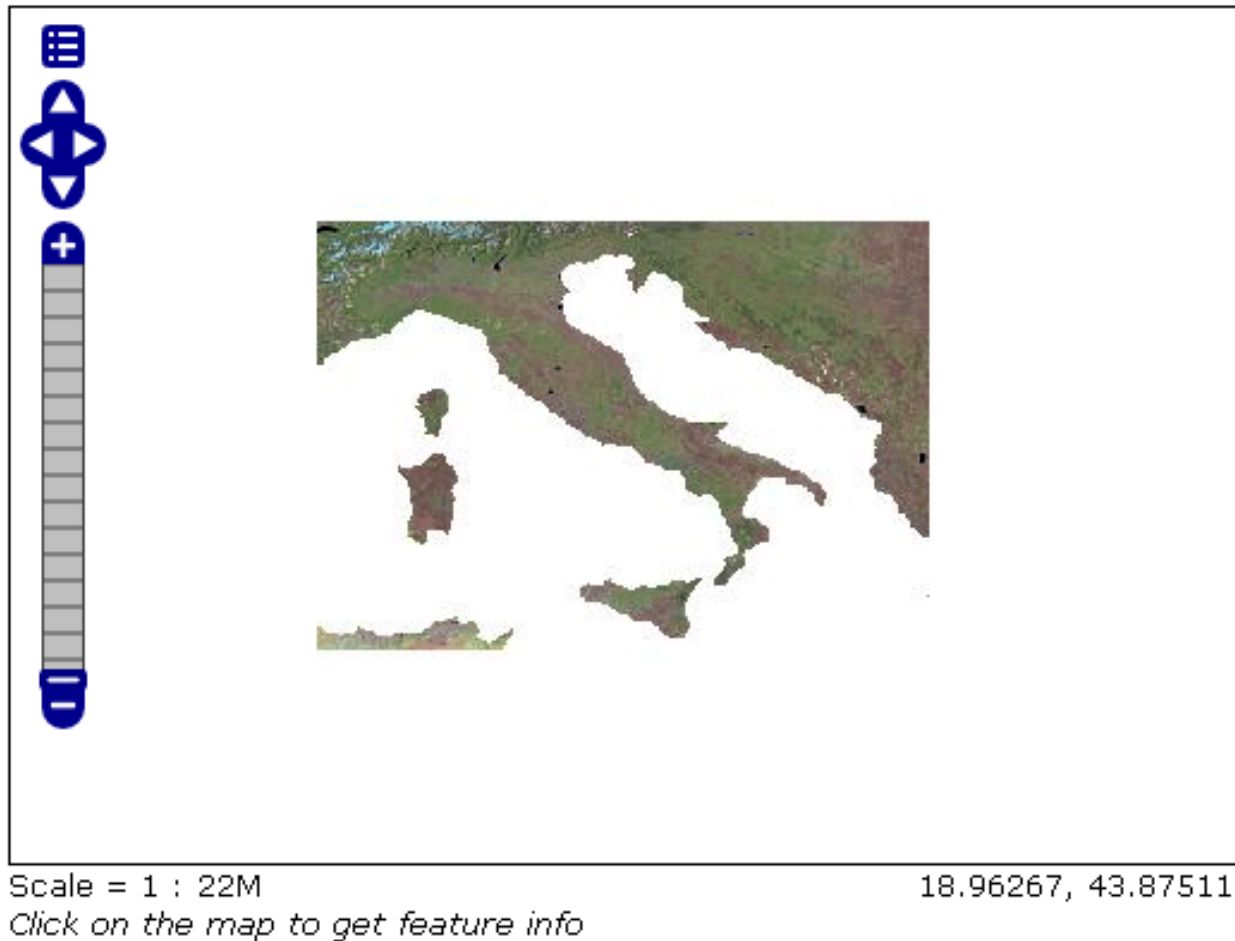
- Clone the directory %TRAINING_ROOT%\data\user_data\footprint_data\mosaic_sample and call it mosaic_sample2
- Rename all the *footprints.xxx* files that compose the shapefile to *mask.xxx* (don't rename *footprints.properties* too!) and load another ImageMosaic datastore.
- open (create if not exist) the **footprints.properties** file and write:

```
footprint_source=mask.shp
footprint_inset=0.00001
footprint_inset_type=border
```

In order to specify an inset and use a footprint shapefile with a custom name.

- Then publish the ImageMosaic located in the cloned directory called mosaic_sample2

The result of setting **FootprintBehavior** to *Transparent*, Inset type to *border* and Inset value to 0.00001 is:



Advanced Processing With GDAL Utilities

In this section we are including some advanced examples of processing of Raster Data for GeoServer serving using GDAL Utilities. Here below you can find a list of examples.

Example n° 1: Serving a large number of grayscale GeoTIFFs with palettes

In this example, there is a group of grayscale GeoTIFF images. The purpose of this section is to describe how to merge these images using GDAL. These data are taken from the [Regione Marche Cartographic Portal](#).

Note: Data have the same pixel resolution and same Coordinate Reference System *EPSG:26592*.

1. Navigate to the workshop directory `$TRAINING_ROOT/data/user_data/gdal_processing_data` (on Windows `%TRAINING_ROOT%\data\user_data\gdal_processing_data`) and find the *grayscale_data* directory.
2. Navigate inside the *grayscale_data* directory with the SDK shell.

Note: The following operations must be executed from the shell inside the selected directory. In Windows, run *setenv.bat* if not already launched.

1. Calling the **gdalinfo** command on an image for retrieving the associated information:

```
gdalinfo 32501_.tif
```

And the result is:

```
Driver: GTiff/GeoTIFF
Files: 32501_.tif
       32501_.tfw
Size is 5494, 4526
Coordinate System is ` '
Origin = (2356751.582169299000000,4762684.428062002200000)
Pixel Size = (1.269090000000000,-1.269090000000000)
Metadata:
  TIFFTAG_RESOLUTIONUNIT=2 (pixels/inch)
  TIFFTAG_XRESOLUTION=1200
  TIFFTAG_YRESOLUTION=1200
Image Structure Metadata:
  COMPRESSION=LZW
  INTERLEAVE=BAND
Corner Coordinates:
Upper Left  ( 2356751.582, 4762684.428)
Lower Left  ( 2356751.582, 4756940.527)
Upper Right  ( 2363723.963, 4762684.428)
Lower Right  ( 2363723.963, 4756940.527)
Center      ( 2360237.772, 4759812.477)
Band 1 Block=5494x1 Type=Byte, ColorInterp=Palette
  Color Table (RGB with 256 entries)
    0: 0,0,0,255
    1: 1,1,1,255
    2: 2,2,2,255
```

(continues on next page)

(continued from previous page)

~

```
254: 254,254,254,255
255: 255,255,255,255
```

From **gdalinfo** it is possible to note:

- No CRS definition. An image without CRS cannot be displayed on GeoServer.
- Tiles Striped (5494x1).
- LZW Compression.
- ColorInterpretation as a Palette.

2. Using **gdal_translate** it is possible to change the ColorInterpretation from Palette to Gray.:

```
gdal_translate -expand gray -a_srs EPSG:26592 -of vrt $f ${f:0:6}.vrt
```

The final image format is not GeoTIFF but **VRT**. This format simply creates an **XML** file containing information about the operation to perform on the image; the output image is created only when the image must be shown to the screen. The CRS is set with the **-a_srs** parameter. The color interpretation can be set to *gray* because each palette value is equal to a grayscale value (this last step is optional).

Note: The **expand gray** option does not create a multi banded image but only one band is present.

Note: In future a possible operation could be the processing of the input image with the color interpretation set to *gray* and the calculation of the optimal palette on the final image.

For executing the same operation on all the input images a script called **script.sh** (Linux) or **script.bat** (Windows) must be created and executed:

Note: In order to edit the scripts use the basic **notepad** editor on Windows or **gedit** on Linux. Remember that on Linux, after the script creation, it must be marked as *executable* with the command `chmod +x <nome_script>.sh`

Linux:

```
#!/bin/bash
FILES="*.tif"
echo start
for f in $FILES
do
    echo $f
    gdal_translate -expand gray -a_srs EPSG:26592 -of vrt $f ${f:0:6}.vrt
done
echo stop
```

Windows:

```
for /R %%f in (*.tif) do (
    gdal_translate -expand gray -a_srs EPSG:26592 -of vrt %%~f %%~f.vrt
)
```

3. Creating a list of the VRT files:

```
ls *.vrt > list.txt (Linux)

or

dir /b *.vrt > list.txt (Windows)
```

4. Merging of all the input files with the **gdalbuildvrt** command:

```
gdalbuildvrt -srcnodata 255 -vrtnodata 255 -resolution highest -input_file_list_
↪list.txt merged_vrt.vrt
```

Parameters used:

- **-srcnodata 255 -vrtnodata 255** : setting of the input and output image No Data.
- **-resolution highest** : selection of the highest image resolution.
- **-input_file_list list.txt** : definition of the input file list.

The result of calling **gdalinfo** on the output image is:

```
Driver: VRT/Virtual Raster
Files: merged_vrt.vrt
       32501_.vrt
       ~
       32507_.vrt
Size is 16342, 9157
Coordinate System is:
PROJCS["Monte Mario (Rome) / Italy zone 2 (deprecated)",
  GEOGCS["Monte Mario (Rome)",
    DATUM["Monte_Mario_Rome",
      SPHEROID["International 1924",6378388,297,
        AUTHORITY["EPSG","7022"]],
      TOWGS84[-104.1,-49.1,-9.9,0.971,-2.917,0.714,-11.68],
      AUTHORITY["EPSG","6806"]],
    PRIMEM["Rome",12.452333333333333,
      AUTHORITY["EPSG","8906"]],
    UNIT["degree",0.0174532925199433,
      AUTHORITY["EPSG","9122"]],
    AUTHORITY["EPSG","4806"]],
  PROJECTION["Transverse_Mercator"],
  PARAMETER["latitude_of_origin",0],
  PARAMETER["central_meridian",2.547666666666666],
  PARAMETER["scale_factor",0.9996],
  PARAMETER["false_easting",2520000],
  PARAMETER["false_northing",0],
  UNIT["metre",1,
    AUTHORITY["EPSG","9001"]],
  AXIS["X",EAST],
  AXIS["Y",NORTH],
  AUTHORITY["EPSG","26592"]],
Origin = (2356629.695870598300000,4762684.428062002200000)
Pixel Size = (1.267290000000000,-1.267290000000000)
Corner Coordinates:
Upper Left  ( 2356629.696, 4762684.428) ( 0d32'36.59"E, 42d59'54.65"N)
```

(continues on next page)

(continued from previous page)

```

Lower Left ( 2356629.696, 4751079.854) ( 0d32'48.78"E, 42d53'38.68"N)
Upper Right ( 2377339.749, 4762684.428) ( 0d47'50.77"E, 43d 0' 9.65"N)
Lower Right ( 2377339.749, 4751079.854) ( 0d48' 1.42"E, 42d53'53.63"N)
Center      ( 2366984.722, 4756882.141) ( 0d40'19.38"E, 42d56'54.40"N)
Band 1 Block=128x128 Type=Byte, ColorInterp=Gray
NoData Value=255

```

5. Transforming from VRT to GeoTIFF with `gdal_translate`:

```

gdal_translate -co "BLOCKXSIZE=512" -co "BLOCKYSIZE=512" -co "TILED=YES" -co
↪ "BIGTIFF=YES" -co "COMPRESS=DEFLATE" merged_vrt.vrt merged_tif.tif

```

Warning: This operation might take many minutes.

Parameters used:

- `-co "BLOCKXSIZE=512"` `-co "BLOCKYSIZE=512"` `-co "TILED=YES"` : setting tile dimensions.
- `-co "BIGTIFF=YES"` `-co "COMPRESS=DEFLATE"` : (Optional) loss-less compression of the image for reducing the disk space occupation, similar to LZW.

Note: `-co "BIGTIFF=YES"` is used because GDAL is not automatically able to convert the GeoTiff image into a BigTiff if compression is set.

From `gdalinfo`:

```

Driver: GTiff/GeoTIFF
Files: merged_tif.tif
Size is 16342, 9157
Coordinate System is:
PROJCS["Monte Mario (Rome) / Italy zone 2",
    GEOGCS["Monte Mario (Rome)",
        DATUM["Monte_Mario_Rome",
            SPHEROID["International 1924",6378388,297.0000000000014,
                AUTHORITY["EPSG","7022"]],
            TOWGS84[-104.1,-49.1,-9.9,0.971,-2.917,0.714,-11.68],
            AUTHORITY["EPSG","6806"]],
        PRIMEM["Rome",12.452333333333],
        UNIT["degree",0.017453292519943],
        AUTHORITY["EPSG","4806"]],
    PROJECTION["Transverse_Mercator"],
    PARAMETER["latitude_of_origin",0],
    PARAMETER["central_meridian",15],
    PARAMETER["scale_factor",0.9996],
    PARAMETER["false_easting",2520000],
    PARAMETER["false_northing",0],
    UNIT["metre",1,
        AUTHORITY["EPSG","9001"]],
    AUTHORITY["EPSG","26592"]]
Origin = (2356629.695870598300000,4762684.428062002200000)
Pixel Size = (1.267290000000000,-1.267290000000000)
Metadata:
  AREA_OR_POINT=Area

```

(continues on next page)

(continued from previous page)

```

Image Structure Metadata:
  COMPRESSION=DEFLATE
  INTERLEAVE=BAND
Corner Coordinates:
Upper Left  ( 2356629.696, 4762684.428) ( 12d59'44.99"E, 42d59'54.65"N)
Lower Left  ( 2356629.696, 4751079.854) ( 12d59'57.18"E, 42d53'38.68"N)
Upper Right ( 2377339.749, 4762684.428) ( 13d14'59.17"E, 43d 0' 9.65"N)
Lower Right ( 2377339.749, 4751079.854) ( 13d15' 9.82"E, 42d53'53.63"N)
Center      ( 2366984.722, 4756882.141) ( 13d 7'27.78"E, 42d56'54.40"N)
Band 1 Block=512x512 Type=Byte, ColorInterp=Gray
  NoData Value=255

```

This image can be displayed on GeoServer but a further optimization step could bring to better performances. There could be two ways for optimizing the GeoServer performances:

- building image overviews.
- building a pyramid of the image.

6. (Optional) Optimization.

- Building overview with `gdaladdo`:

```

gdaladdo -r cubicspline --config COMPRESS_OVERVIEW DEFLATE --config GDAL_
↳TIFF_OVR_BLOCKSIZE 512 merged_tif.tif 2 4 8 16 32

```

Overviews are reduced views of the input image used by GeoServer for displaying the image at a lower resolutions.

Parameters used:

- **-r cubicspline** : setting the interpolation mode to *cubicspline* (by default is *nearest-neighbour*).
- **--config COMPRESS_OVERVIEW DEFLATE** : setting DEFLATE compression on the overviews, for reducing disk space occupation.
- **--config GDAL_TIFF_OVR_BLOCKSIZE 512** : setting tile dimensions on overviews.
- **2 ~ 32** : setting overview level.

And with `gdalinfo`:

```

Band 1 Block=512x512 Type=Byte, ColorInterp=Gray
  NoData Value=255
  Overviews: 8171x4579, 4086x2290, 2043x1145, 1022x573, 511x287

```

Then the result can be displayed in GeoServer by configuring the image as a GeoTIFF (see [Adding a GeoTiff](#) section).

- Building a pyramid through several `gdalwarp` invocations, each time by reducing the image resolution:

```

gdalwarp -r cubicspline -dstnodata 255 -srcnodata 255 -multi -tr 2,53458 -2,
↳53458 -co BLOCKXSIZE=512 -co BLOCKYSIZE=512 -co TILED=YES -co_
↳COMPRESS=DEFLATE merged_tif.tif merged_tif_2.tif

```

Parameters used:

- **-r cubicspline** : definition interpolation method.
- **-dstnodata 255 -srcnodata 255** : definition of the image input and output NO DATA.
- **-multi** : forcing to use multithreading.

- **-tr 2,53458 -2,53458** : definition of the image resolutions.

Output image from **gdalinfo**:

```
Driver: GTiff/GeoTIFF
Files: merged_tif_2.tif
Size is 8171, 4578
Coordinate System is:

~

Band 1 Block=512x512 Type=Byte, ColorInterp=Gray
  NoData Value=255
```

After another **gdalwarp** on the output image:

```
gdalwarp -r cubicspline -dstnodata 255 -srcnodata 255 -multi -tr 5,06916 -5,
→06916 -co BLOCKXSIZE=512 -co BLOCKYSIZE=512 -co TILED=YES -co_
→COMPRESS=DEFLATE merged_tif_2.tif merged_tif_4.tif
```

And **gdalinfo**:

```
Driver: GTiff/GeoTIFF
Files: merged_tif_4.tif
Size is 4085, 2289
Coordinate System is:

~

Band 1 Block=512x512 Type=Byte, ColorInterp=Gray
  NoData Value=255
```

The operations must be executed on the first image, then the same operation must be repeated on the output image and so on. This cycle allows to create a pyramid of images, each one with a lower resolution.

Then the result can be displayed in GeoServer by configuring the images as a pyramid (see [Advanced Mosaic and Pyramid configuration](#) section).

7. Displaying the result on GeoServer:

Example n°2: Serving a large number of DTM ASCII Grid Files

In this example there is a group of DTM images in ASCII Grid format. The purpose of this section is to describe how the GDAL commands may be used for merging the input files provided. These data are taken from [Regione Calabria Open Data Portal](#) at the *ASCII - GRID* section.

Note: Data have the same pixel resolution and same Coordinate Reference System *EPSG:3003*.

Warning: This example requires GDAL with Python bindings.

1. Navigate to the workshop directory `$TRAINING_ROOT/data/user_data/gdal_processing_data` (on Windows `%TRAINING_ROOT%\data\user_data\gdal_processing_data`) and find the *DTM_data* directory.
2. Navigate into the *DTM_data* directory with the SDK Shell.

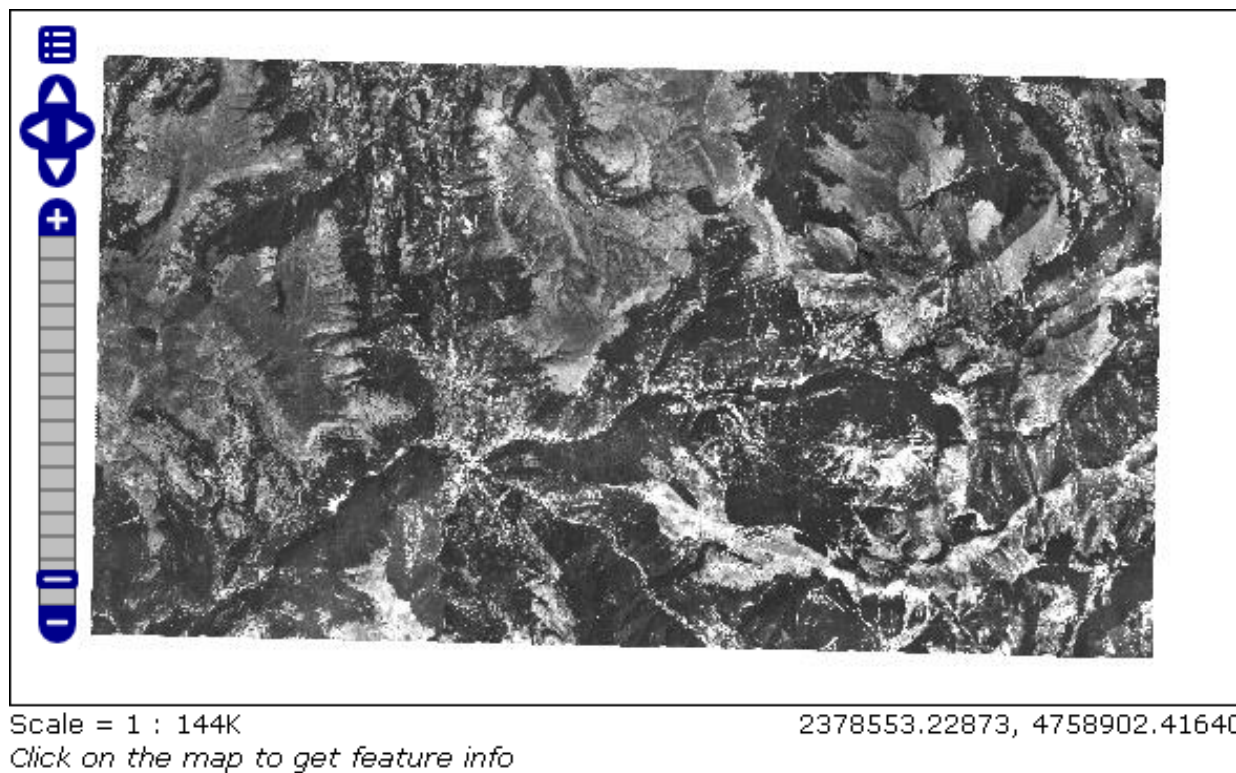


Fig. 242: Result with gdaladdo

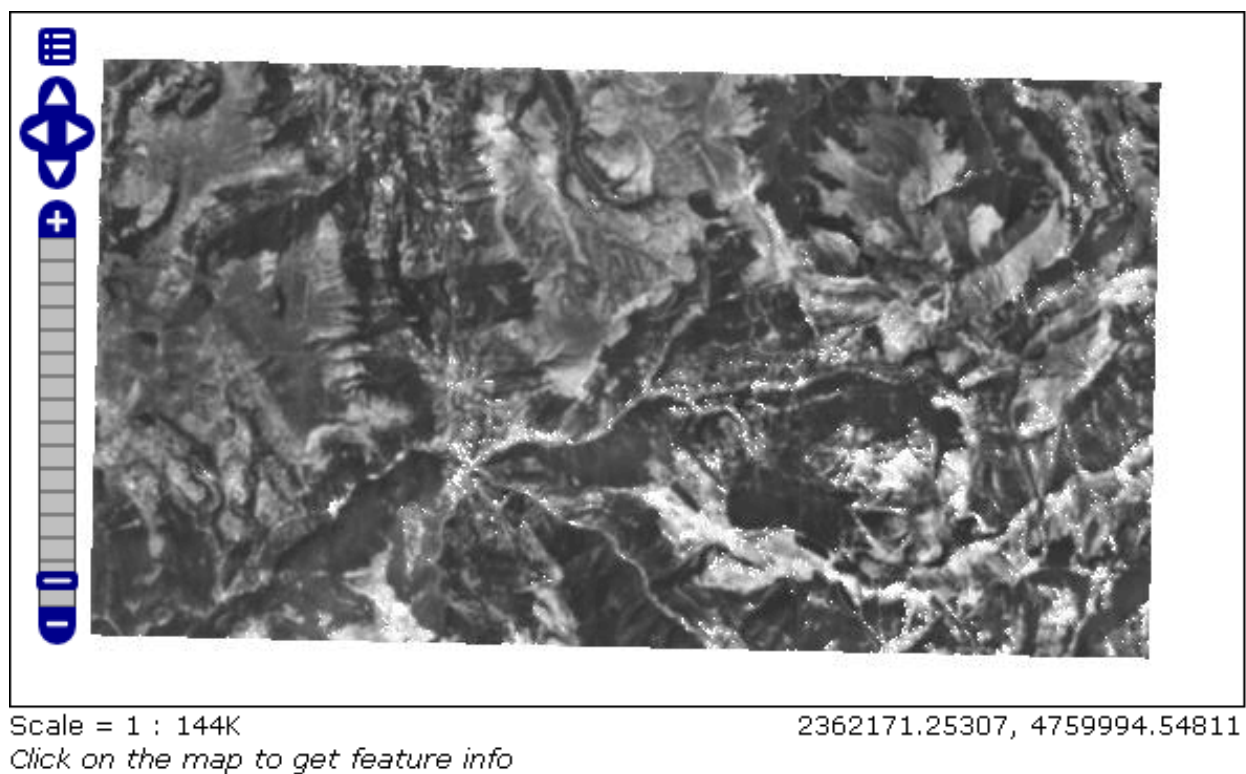


Fig. 243: Result with ImagePyramid

Note: The following operations must be executed from the shell inside the selected directory. In Windows, run *setenv.bat* if not already launched.

1. Calling the **gdalinfo** command on an image for retrieving the associated information:

```
gdalinfo 521150.asc
```

And the result is:

```
Driver: AAIGrid/Arc/Info ASCII Grid
Files: 521150.asc
Size is 193, 154
Coordinate System is ``
Origin = (2590740.000000000000000,4433860.000000000000000)
Pixel Size = (40.000000000000000,-40.000000000000000)
Metadata:
  AREA_OR_POINT=Point
Corner Coordinates:
Upper Left  ( 2590740.000, 4433860.000)
Lower Left  ( 2590740.000, 4427700.000)
Upper Right ( 2598460.000, 4433860.000)
Lower Right ( 2598460.000, 4427700.000)
Center      ( 2594600.000, 4430780.000)
Band 1 Block=193x1 Type=Float32, ColorInterp=Undefined
  NoData Value=-9999
```

From **gdalinfo** it is possible to note:

- No CRS definition. An image without CRS cannot be displayed on GeoServer.
- Tiles Striped (193x1).
- No Compression.

2. Listing of all the images into a single text list with the following command:

```
ls *.asc > list.txt (Linux)

or

dir /b *.asc > list.txt (Windows)
```

3. Merging of all the input files with the **gdal_merge.py** command:

```
gdal_merge.py -o merged.tif -co "TILED=YES" -co "BLOCKXSIZE=512" -co
↪"BLOCKYSIZE=512" -co "COMPRESS=DEFLATE" -co "ZLEVEL=9" -co "BIGTIFF=YES" -init -
↪9999 -a_nodata -9999 -n -9999 -ot Float32 --optfile list.txt
```

Note: This command must be executed with python for avoiding import errors.

Parameters used:

- **-o merged.tif** : definition of the output file name.
- **-co "TILED=YES" -co "BLOCKXSIZE=512" -co "BLOCKYSIZE=512"** : definition of tile dimensions.

- **-co "COMPRESS=DEFLATE" -co "ZLEVEL=9" -co "BIGTIFF=YES"** : definition of the compression mode.

Note: **-co "BIGTIFF=YES"** is used because GDAL is not automatically able to convert the GeoTiff image into a BigTiff if compression is set.

- **-init -9999** : initialization of the image pixels to NO DATA.
- **-a_nodata -9999** : definition of the output value for NO DATA.
- **-n -9999** : definition of the input pixel value to ignore during merging.
- **-ot Float32** : definition of the image output type.
- **-optfile list.txt** : definition of the input file list.

The **gdalinfo** output on the merged image is:

```
Driver: GTiff/GeoTIFF
Files: merged.tif
Size is 3613, 6284
Coordinate System is ``
Origin = (2570700.000000000000000,4445900.000000000000000)
Pixel Size = (40.000000000000000,-40.000000000000000)
Image Structure Metadata:
  COMPRESSION=DEFLATE
  INTERLEAVE=BAND
Corner Coordinates:
Upper Left  ( 2570700.000, 4445900.000)
Lower Left  ( 2570700.000, 4194540.000)
Upper Right ( 2715220.000, 4445900.000)
Lower Right ( 2715220.000, 4194540.000)
Center      ( 2642960.000, 4320220.000)
Band 1 Block=512x512 Type=Float32, ColorInterp=Gray
NoData Value=-9999
```

The merged image has a good tiling(512x512) and compression, but the CRS is still undefined.

4. Setting of the image CRS with **gdal_translate**:

```
gdal_translate -a_srs "EPSG:3003" -co "TILED=YES" -co "BLOCKXSIZE=512" -co
→"BLOCKYSIZE=512" -co "COMPRESS=DEFLATE" -co "ZLEVEL=9" -co "BIGTIFF=YES" merged.
→tif merged_CRS.tif
```

The various input parameters are maintained because by default GDAL do not compress the input image and set a bad tiling.

From **gdalinfo**:

```
Driver: GTiff/GeoTIFF
Files: merged_CRS.tif
Size is 3613, 6284
Coordinate System is:
PROJCS["Monte Mario / Italy zone 1",
  GEOGCS["Monte Mario",
    DATUM["Monte_Mario",
      SPHEROID["International 1924",6378388,297.0000000000014,
        AUTHORITY["EPSG","7022"]],
      TOWGS84[-104.1,-49.1,-9.9,0.971,-2.917,0.714,-11.68],
```

(continues on next page)

(continued from previous page)

```

        AUTHORITY["EPSG", "6265"]],
        PRIMEM["Greenwich", 0],
        UNIT["degree", 0.0174532925199433],
        AUTHORITY["EPSG", "4265"]],
    PROJECTION["Transverse_Mercator"],
    PARAMETER["latitude_of_origin", 0],
    PARAMETER["central_meridian", 9],
    PARAMETER["scale_factor", 0.9996],
    PARAMETER["false_easting", 1500000],
    PARAMETER["false_northing", 0],
    UNIT["metre", 1,
        AUTHORITY["EPSG", "9001"]],
    AUTHORITY["EPSG", "3003"]]
Origin = (2570700.0000000000000000, 4445900.0000000000000000)
Pixel Size = (40.0000000000000000, -40.0000000000000000)
Metadata:
  AREA_OR_POINT=Area
Image Structure Metadata:
  COMPRESSION=DEFLATE
  INTERLEAVE=BAND
Corner Coordinates:
Upper Left  ( 2570700.000,  4445900.000) ( 21d25'57.43"E, 39d29'28.80"N)
Lower Left  ( 2570700.000,  4194540.000) ( 21d 3'12.94"E, 37d16'39.68"N)
Upper Right ( 2715220.000,  4445900.000) ( 23d 3'58.08"E, 39d18' 6.80"N)
Lower Right ( 2715220.000,  4194540.000) ( 22d38'27.42"E, 37d 6' 9.29"N)
Center      ( 2642960.000,  4320220.000) ( 22d 2'40.73"E, 38d17'47.75"N)
Band 1 Block=512x512 Type=Float32, ColorInterp=Gray
  NoData Value=-9999

```

This image can be displayed on GeoServer but a further optimization step could bring to better performances.

5. (Optional) Creation of the **overviews** associated to the merged image for having better throughput:

```

gdaladdo -r nearest --config COMPRESS_OVERVIEW DEFLATE --config GDAL_TIFF_OVR_
↪BLOCKSIZE 512 merged_CRS.tif 2 4 8 16

```

Overviews are reduced views of the input image used by GeoServer for displaying the image at a lower resolutions.

Parameters used:

- **-r nearest** : definition of the interpolation method.
- **--config COMPRESS_OVERVIEW DEFLATE** : definition of the compression on overviews.
- **--config GDAL_TIFF_OVR_BLOCKSIZE 512** : definition of the tile dimensions on overviews.

And with **gdalinfo**:

```

Driver: GTiff/GeoTIFF
Files: merged_CRS.tif
Size is 3613, 6284
Coordinate System is:
PROJCS["Monte Mario / Italy zone 1",
    GEOGCS["Monte Mario",
        DATUM["Monte_Mario",
            SPHEROID["International 1924", 6378388, 297.0000000000014,
                AUTHORITY["EPSG", "7022"]],
            TOWGS84[-104.1, -49.1, -9.9, 0.971, -2.917, 0.714, -11.68],

```

(continues on next page)

(continued from previous page)

```

        AUTHORITY["EPSG", "6265"]],
        PRIMEM["Greenwich", 0],
        UNIT["degree", 0.0174532925199433],
        AUTHORITY["EPSG", "4265"]],
        PROJECTION["Transverse_Mercator"],
        PARAMETER["latitude_of_origin", 0],
        PARAMETER["central_meridian", 9],
        PARAMETER["scale_factor", 0.9996],
        PARAMETER["false_easting", 1500000],
        PARAMETER["false_northing", 0],
        UNIT["metre", 1,
            AUTHORITY["EPSG", "9001"]],
        AUTHORITY["EPSG", "3003"]]
Origin = (2570700.0000000000000000, 4445900.0000000000000000)
Pixel Size = (40.0000000000000000, -40.0000000000000000)
Metadata:
  AREA_OR_POINT=Area
Image Structure Metadata:
  COMPRESSION=DEFLATE
  INTERLEAVE=BAND
Corner Coordinates:
Upper Left  ( 2570700.000, 4445900.000) ( 21d25'57.43"E, 39d29'28.80"N)
Lower Left  ( 2570700.000, 4194540.000) ( 21d 3'12.94"E, 37d16'39.68"N)
Upper Right ( 2715220.000, 4445900.000) ( 23d 3'58.08"E, 39d18' 6.80"N)
Lower Right ( 2715220.000, 4194540.000) ( 22d38'27.42"E, 37d 6' 9.29"N)
Center      ( 2642960.000, 4320220.000) ( 22d 2'40.73"E, 38d17'47.75"N)
Band 1 Block=512x512 Type=Float32, ColorInterp=Gray
  NoData Value=-9999
  Overviews: 1807x3142, 904x1571, 452x786, 226x393

```

Then the result can be displayed in GeoServer by configuring the image as a GeoTIFF (see [Adding a GeoTiff](#) section).

6. Displaying the result on GeoServer:

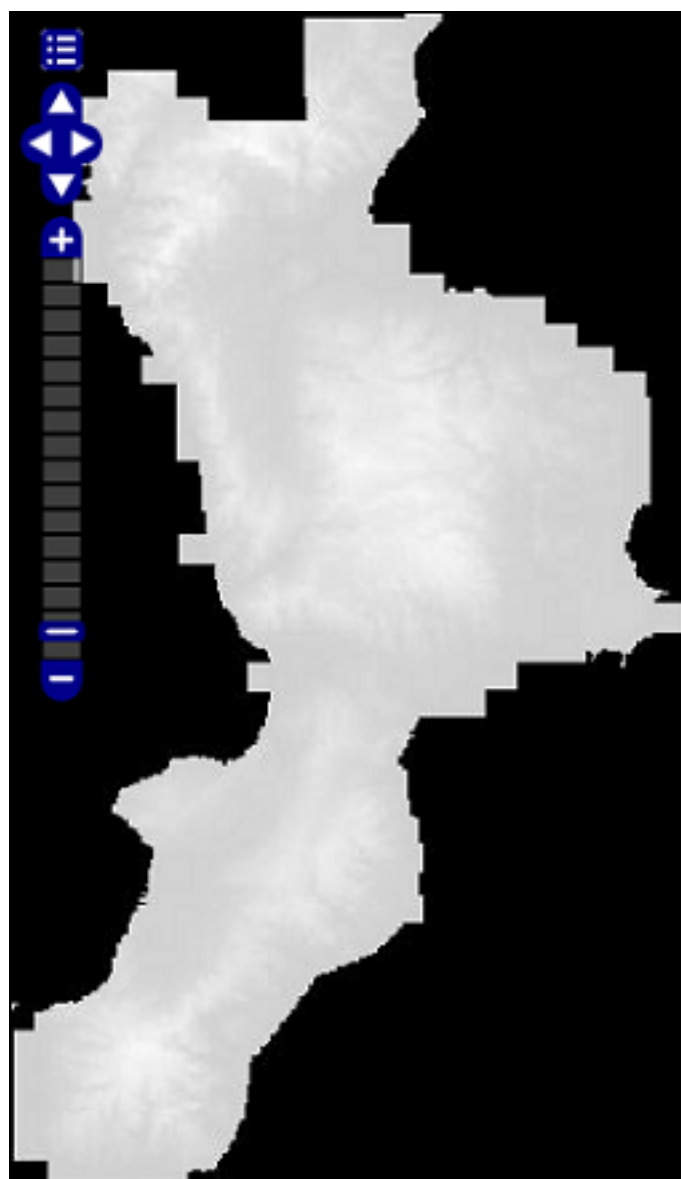
Example n° 3: Serving a large number of Cartographic Black/White GeoTiff with Palette

In this example there is a group of Cartographic Black/White images from “CARTA TECNICA DELLA REGIONE TOSCANA”. The purpose of this example is to describe how the GDAL commands may be used for merging the input files provided.

Note: Data have the same pixel resolution and same Coordinate Reference System *EPSG:25832*. Also each pixel is stored into single bit.

1. Navigate to the workshop directory `$TRAINING_ROOT/data/user_data/gdal_processing_data` (on Windows `%TRAINING_ROOT%\data\user_data\gdal_processing_data`) and find the *CTR_data* directory.
2. Navigate into the *CTR_data* directory with the SDK Shell.

Note: The following operations must be executed from the shell inside the selected directory. In Windows, run *setenv.bat* if not already launched.



Scale = 1 : 2711813.43750, 4290763.75000
2M

Click on the map to get feature info

1. Calling the **gdalinfo** command on an image for retrieving the associated information:

```
gdalinfo 20E27_1994.TIF
```

And the result is:

```
Driver: GTiff/GeoTIFF
Files: 20E27_1994.TIF
       20E27_1994.TFW
Size is 16050, 14050
Coordinate System is ``
GeoTransform =
  600769.026848671, 0.1, 7.3789937e-007
  4863785.940434861, -8.172141e-008, -0.1
Metadata:
  TIFFTAG_RESOLUTIONUNIT=2 (pixels/inch)
  TIFFTAG_SOFTWARE=IrfanView
  TIFFTAG_XRESOLUTION=72
  TIFFTAG_YRESOLUTION=72
Image Structure Metadata:
  COMPRESSION=PACKBITS
  INTERLEAVE=BAND
  MINISWHITE=YES
Corner Coordinates:
Upper Left  ( 600769.027, 4863785.940)
Lower Left  ( 600769.037, 4862380.940)
Upper Right  ( 602374.027, 4863785.939)
Lower Right  ( 602374.037, 4862380.939)
Center      ( 601571.532, 4863083.440)
Band 1 Block=16050x4 Type=Byte, ColorInterp=Palette
  Image Structure Metadata:
    NBITS=1
  Color Table (RGB with 2 entries)
    0: 255,255,255,255
    1: 0,0,0,255
```

From **gdalinfo** it is possible to note:

- No CRS definition. An image without CRS cannot be displayed on GeoServer.
- Color Interpretation as palette.
- A GeoTransformation matrix is associated.
- Tiles Striped (16050x4).
- Packbits Compression.

2. Executing the following commands on the tiff images:

```
gdalwarp -t_srs EPSG:25832 -of vrt $f ${f:0:10}_temp.vrt
gdal_translate -expand gray -of vrt ${f:0:10}_temp.vrt ${f:0:10}.vrt
```

The first operation sets the CRS to each image and creates a **VRT** file, for reducing space occupation. Also the use of **gdalwarp** internally performs the GeoTransformation associated to the image.

The second operation is used for changing the color interpretation from *palette* to *gray*. This operation is done because in the final steps other grey levels will be added with the interpolation. The expansion to the *gray* color interpretation leads to an expansion of the pixel dimension from 1 to 8 bits.

Note: The **expand gray** option does not create a multi banded image but only a single banded one.

Note: If the user wants to keep the palette, then can go directly to the *Optional elaboration without expanding the Palette* paragraph.

These operations must be executed inside a script:

Linux:

```
#!/bin/bash
FILES="*.TIF"
echo start
for f in $FILES
do
    echo $f
    gdalwarp -t_srs EPSG:25832 -of vrt $f ${f:0:10}_temp.vrt
    gdal_translate -expand gray -of vrt ${f:0:10}_temp.vrt ${f:0:10}.vrt
done
echo stop
```

Windows:

```
for /R %%f in (*.tif) do (
    gdalwarp -t_srs EPSG:25832 -of vrt %%~f %%~f_temp.vrt
    gdal_translate -expand gray -of vrt %%~f_temp.vrt %%~f.vrt
)
```

3. Listing of all the **VRT** files into a single text list with the following command:

```
ls *.vrt > list.txt (Linux)

or

dir /b *.vrt > list.txt (Windows)
```

Warning: Delete the **_temp.vrt** files from the list because they overlap with the final vrt files created.

4. Merging of all the input files with the **gdalbuildvrt** command:

```
gdalbuildvrt -srcnodata 255 -vrtnodata 255 -input_file_list list.txt merged_vrt.
↪vrt
```

Parameters used:

- **-srcnodata 255 -vrtnodata 255** : definition of the No Data associated with the file.
- **-input_file_list list.txt** : definition of input files to elaborate.

The **gdalinfo** output on the merged image is:

```
Driver: VRT/Virtual Raster
Files: merged_vrt_nodata.vrt
      20E27_1994.vrt
```

(continues on next page)

(continued from previous page)

```

~
20E60_1995.vrt
Size is 50052, 62047
Coordinate System is:
PROJCS["ETRS89 / UTM zone 32N",
    GEOGCS["ETRS89",
        DATUM["European_Terrestrial_Reference_System_1989",
            SPHEROID["GRS 1980", 6378137, 298.257222101,
                AUTHORITY["EPSG", "7019"]],
            TOWGS84[0,0,0,0,0,0,0],
            AUTHORITY["EPSG", "6258"]],
        PRIMEM["Greenwich", 0,
            AUTHORITY["EPSG", "8901"]],
        UNIT["degree", 0.0174532925199433,
            AUTHORITY["EPSG", "9122"]],
        AUTHORITY["EPSG", "4258"]],
    PROJECTION["Transverse_Mercator"],
    PARAMETER["latitude_of_origin", 0],
    PARAMETER["central_meridian", 9],
    PARAMETER["scale_factor", 0.9996],
    PARAMETER["false_easting", 500000],
    PARAMETER["false_northing", 0],
    UNIT["metre", 1,
        AUTHORITY["EPSG", "9001"]],
    AXIS["Easting", EAST],
    AXIS["Northing", NORTH],
    AUTHORITY["EPSG", "25832"]]
Origin = (600768.734234663190000, 4863785.940434861000000)
Pixel Size = (0.100000372821407, -0.100000372821407)
Corner Coordinates:
Upper Left ( 600768.734, 4863785.940) ( 10d15'18.69"E, 43d55'13.06"N)
Lower Left ( 600768.734, 4857581.217) ( 10d15'14.46"E, 43d51'51.99"N)
Upper Right ( 605773.953, 4863785.940) ( 10d19' 3.07"E, 43d55'10.54"N)
Lower Right ( 605773.953, 4857581.217) ( 10d18'58.64"E, 43d51'49.47"N)
Center ( 603271.344, 4860683.579) ( 10d17' 8.72"E, 43d53'31.28"N)
Band 1 Block=128x128 Type=Byte, ColorInterp=Gray
NoData Value=255

```

5. Transforming from VRT to GeoTIFF with `gdal_translate`:

```

gdal_translate -a_nodata none -co "BLOCKXSIZE=512" -co "BLOCKYSIZE=512" -co
↪ "TILED=YES" -co "BIGTIFF=YES" -co "COMPRESS=DEFLATE" merged_vrt.vrt merged_tif.
↪ tif

```

The various input parameters are:

- **-a_nodata none** : avoid setting 255 as No Data for a better image optimization.
- **-co "BLOCKXSIZE=512" -co "BLOCKYSIZE=512" -co "TILED=YES"** : definition of the tile dimensions.
- **-co "BIGTIFF=YES" -co "COMPRESS=DEFLATE"** : definition of the compression method.

Note: BIGTIFF=YES must be set for big images because when compression is used, by

default `gdal_translate` is not able to check if the final image is a BigTiff or not.

From `gdalinfo`:

```
Driver: GTiff/GeoTIFF
Files: merged_tif.tif
Size is 50052, 62047
Coordinate System is:
PROJCS["ETRS89 / UTM zone 32N",
    GEOGCS["ETRS89",
        DATUM["European_Terrestrial_Reference_System_1989",
            SPHEROID["GRS_1980",6378137,298.2572221010002,
                AUTHORITY["EPSG","7019"]],
            TOWGS84[0,0,0,0,0,0],
            AUTHORITY["EPSG","6258"]],
        PRIMEM["Greenwich",0],
        UNIT["degree",0.0174532925199433],
        AUTHORITY["EPSG","4258"]],
    PROJECTION["Transverse_Mercator"],
    PARAMETER["latitude_of_origin",0],
    PARAMETER["central_meridian",9],
    PARAMETER["scale_factor",0.9996],
    PARAMETER["false_easting",500000],
    PARAMETER["false_northing",0],
    UNIT["metre",1,
        AUTHORITY["EPSG","9001"]],
    AUTHORITY["EPSG","25832"]]
Origin = (600768.734234663190000,4863785.940434861000000)
Pixel Size = (0.100000372821407,-0.100000372821407)
Metadata:
  AREA_OR_POINT=Area
Image Structure Metadata:
  COMPRESSION=DEFLATE
  INTERLEAVE=BAND
Corner Coordinates:
Upper Left  ( 600768.734, 4863785.940) ( 10d15'18.69"E, 43d55'13.06"N)
Lower Left  ( 600768.734, 4857581.217) ( 10d15'14.46"E, 43d51'51.99"N)
Upper Right ( 605773.953, 4863785.940) ( 10d19' 3.07"E, 43d55'10.54"N)
Lower Right ( 605773.953, 4857581.217) ( 10d18'58.64"E, 43d51'49.47"N)
Center      ( 603271.344, 4860683.579) ( 10d17' 8.72"E, 43d53'31.28"N)
Band 1 Block=512x512 Type=Byte, ColorInterp=Gray
```

This image can be displayed on GeoServer but a further optimization step could bring to better performances. There could be two ways for optimizing the GeoServer performances:

- building image overviews.
- building a pyramid of the image.

6. **(Optional)** Optimization methods. Here are described two possible optimizations each of them using a different interpolation type:

- Creation of the **overviews** associated to the merged image for having better throughput:

```
gdaladdo -r average --config COMPRESS_OVERVIEW DEFLATE --config GDAL_TIFF_
↳OVR_BLOCKSIZE 512 merged_tif.tif 2 4 8 16 32 64 128
```

Overviews are reduced views of the input image used by GeoServer for displaying the image at a lower resolutions.

Parameters used:

- **-r average** : definition of the interpolation method.
- **-config COMPRESS_OVERVIEW DEFLATE** : definition of the compression on overviews.
- **-config GDAL_TIFF_OVR_BLOCKSIZE 512** : definition of the tile dimensions on overviews.
- **2 ~ 128** : definition of the overviews level

And with **gdalinfo**:

```
Driver: GTiff/GeoTIFF
Files: merged_tif.tif
Size is 50052, 62047

~

Band 1 Block=512x512 Type=Byte, ColorInterp=Gray
  Overviews: 25026x31024, 12513x15512, 6257x7756, 3129x3878, 1565x1939,
↳ 783x970, 392x485
```

Then the result can be displayed in GeoServer by configuring the image as a GeoTIFF (see [Adding a GeoTiff](#) section).

- **(Optional)** Creation of a **pyramid** associated to the merged image and displaying the image on GeoServer with the ImagePyramid plugin (see [Advanced Mosaic and Pyramid configuration](#) section).

For building a pyramid the **gdalwarp** command must be called several times. The operation to execute on the first image is:

```
gdalwarp -r cubic -multi -tr 0,200000745642814 -0,200000745642814 -co
↳ BLOCKXSIZE=512 -co BLOCKYSIZE=512 -co TILED=YES -co COMPRESS=DEFLATE
↳ merged_tif.tif merged_tif_2.tif
```

The parameters are:

- **-r cubic** : definition of the interpolation method (**average** interpolation can be used only with GDAL 1.10).
- **-multi** : forcing to use multithreading.
- **-tr 0,200000745642814 -0,200000745642814** : definition of the image resolution.

From **gdalinfo** on the result image:

```
Driver: GTiff/GeoTIFF
Files: merged_tif_2.tif
Size is 25026, 31024

~

Band 1 Block=512x512 Type=Byte, ColorInterp=Gray
```

Then the same operation, with another value for the resolution must be executed on the result image:

```
gdalwarp -r cubic -multi -tr 0,400001491285628 -0,400001491285628 -co
↳ BLOCKXSIZE=512 -co BLOCKYSIZE=512 -co TILED=YES -co COMPRESS=DEFLATE
↳ merged_tif_2.tif merged_tif_4.tif
```

These operation must be repeated until the final image has a resolution 128 times lower than that of the initial image.

Note: Each call of **gdalwarp** reduces by half the image resolution.

After creating the various rasters, they must be saved inside a new directory. This directory must be internally divided into sub-directories numbered from 1 to 7, each of them containing a raster of smaller dimension (going from 1 to 7) and leaving the original raster in the super-directory.

Then the user can configure the following structure with the ImagePyramid plugin.

7. Displaying the result on GeoServer:

Result as a pyramid (Zoom on the image for seeing the result).

Result with overviews (Zoom on the image for seeing the result).

Optional elaboration without expanding the Palette

If the user wants to keep the palette the steps to achieve are quite similar.

1. Executing the following commands on the tiff images:

```
gdalwarp -t_srs EPSG:25832 -of vrt $f ${f:0:10}_temp.vrt
gdal_translate -of vrt ${f:0:10}_temp.vrt ${f:0:10}.vrt
```

These operations must be executed inside a script:

Linux:

```
#!/bin/bash
FILES="*.TIF"
echo start
for f in $FILES
do
    echo $f
    gdalwarp -t_srs EPSG:25832 -of vrt $f ${f:0:10}_temp.vrt
    gdal_translate -of vrt ${f:0:10}_temp.vrt ${f:0:10}.vrt
done
echo stop
```

Windows:

```
for /R %%f in (*.tif) do (
    gdalwarp -t_srs EPSG:25832 -of vrt %%~f %%~f_temp.vrt
    gdal_translate -of vrt %%~f_temp.vrt %%~f.vrt
)
```

2. Listing of all the **VRT** files into a single text list with the following command:

```
ls *.vrt > list.txt (Linux)

or

dir /b *.vrt > list.txt (Windows)
```

Warning: Delete the **_temp.vrt** files from the list because they overlap with the final vrt files created.



Scale = 1 : 5410

601837.72684, 4860725.99397

Click on the map to get feature info



Scale = 1 : 5410

601986.17969, 4860648.73789

Click on the map to get feature info

3. Merging of all the input files with the **gdalbuildvrt** command:

```
gdalbuildvrt -srcnodata 0 -vrtnodata 0 -input_file_list list.txt merged_vrt.vrt
```

Parameters used:

- **-srcnodata 0 -vrtnodata 0** : definition of the No Data associated with the file.
- **-input_file_list list.txt** : definition of input files to elaborate.

The **gdalinfo** output on the merged image is:

```
Driver: VRT/Virtual Raster
Files: merged_vrt_0.vrt
20E27_1994.TIF.vrt

~

20E60_1995.TIF.vrt
Size is 50052, 62047
Coordinate System is:
PROJCS["ETRS89 / UTM zone 32N",
    GEOGCS["ETRS89",
        DATUM["European_Terrestrial_Reference_System_1989",
            SPHEROID["GRS 1980", 6378137, 298.257222101,
                AUTHORITY["EPSG", "7019"]],
            TOWGS84[0, 0, 0, 0, 0, 0],
            AUTHORITY["EPSG", "6258"]],
        PRIMEM["Greenwich", 0,
            AUTHORITY["EPSG", "8901"]],
        UNIT["degree", 0.0174532925199433,
            AUTHORITY["EPSG", "9122"]],
        AUTHORITY["EPSG", "4258"]],
    PROJECTION["Transverse_Mercator"],
    PARAMETER["latitude_of_origin", 0],
    PARAMETER["central_meridian", 9],
    PARAMETER["scale_factor", 0.9996],
    PARAMETER["false_easting", 500000],
    PARAMETER["false_northing", 0],
    UNIT["metre", 1,
        AUTHORITY["EPSG", "9001"]],
    AXIS["Easting", EAST],
    AXIS["Northing", NORTH],
    AUTHORITY["EPSG", "25832"]]
Origin = (600768.734234663190000, 4863785.940434861000000)
Pixel Size = (0.100000372821407, -0.100000372821407)
Corner Coordinates:
Upper Left ( 600768.734, 4863785.940) ( 10d15'18.69"E, 43d55'13.06"N)
Lower Left ( 600768.734, 4857581.217) ( 10d15'14.46"E, 43d51'51.99"N)
Upper Right ( 605773.953, 4863785.940) ( 10d19' 3.07"E, 43d55'10.54"N)
Lower Right ( 605773.953, 4857581.217) ( 10d18'58.64"E, 43d51'49.47"N)
Center ( 603271.344, 4860683.579) ( 10d17' 8.72"E, 43d53'31.28"N)
Band 1 Block=128x128 Type=Byte, ColorInterp=Palette
  NoData Value=0
  Color Table (RGB with 2 entries)
    0: 255,255,255,255
    1: 0,0,0,255
```

4. Transforming from VRT to GeoTIFF with **gdal_translate**:

```
gdal_translate -co "NBITS=1" -co "BLOCKXSIZE=512" -co "BLOCKYSIZE=512" -co
↪ "TILED=YES" -co "BIGTIFF=YES" -co "COMPRESS=DEFLATE" merged_vrt.vrt merged_tif.
↪ tif
```

The various input parameters are:

- **-co “NBITS=1”** : sets the bits per pixel to 1, because the Palette contains only 0 or 1.
- **-co “BLOCKXSIZE=512” -co “BLOCKYSIZE=512” -co “TILED=YES”** : definition of the tile dimensions.
- **-co “BIGTIFF=YES” -co “COMPRESS=DEFLATE”** : definition of the compression method.

Note: BIGTIFF=YES must be set for big images because when compression is used, by default **gdal_translate** is not able to check if the final image is a BigTiff or not.

From **gdalinfo**:

```
Size is 50052, 62047
Coordinate System is:
PROJCS["ETRS89 / UTM zone 32N",
  GEOGCS["ETRS89",
    DATUM["European_Terrestrial_Reference_System_1989",
      SPHEROID["GRS 1980",6378137,298.2572221010002,
        AUTHORITY["EPSG","7019"]],
      TOWGS84[0,0,0,0,0,0,0],
      AUTHORITY["EPSG","6258"]],
    PRIMEM["Greenwich",0],
    UNIT["degree",0.0174532925199433],
    AUTHORITY["EPSG","4258"]],
  PROJECTION["Transverse_Mercator"],
  PARAMETER["latitude_of_origin",0],
  PARAMETER["central_meridian",9],
  PARAMETER["scale_factor",0.9996],
  PARAMETER["false_easting",500000],
  PARAMETER["false_northing",0],
  UNIT["metre",1,
    AUTHORITY["EPSG","9001"]],
  AUTHORITY["EPSG","25832"]]
Origin = (600768.734234663190000,4863785.940434861000000)
Pixel Size = (0.100000372821407,-0.100000372821407)
Metadata:
  AREA_OR_POINT=Area
Image Structure Metadata:
  COMPRESSION=DEFLATE
  INTERLEAVE=BAND
Corner Coordinates:
Upper Left  ( 600768.734, 4863785.940) ( 10d15'18.69"E, 43d55'13.06"N)
Lower Left  ( 600768.734, 4857581.217) ( 10d15'14.46"E, 43d51'51.99"N)
Upper Right ( 605773.953, 4863785.940) ( 10d19' 3.07"E, 43d55'10.54"N)
Lower Right ( 605773.953, 4857581.217) ( 10d18'58.64"E, 43d51'49.47"N)
Center      ( 603271.344, 4860683.579) ( 10d17' 8.72"E, 43d53'31.28"N)
Band 1 Block=512x512 Type=Byte, ColorInterp=Palette
  NoData Value=0
Image Structure Metadata:
  NBITS=1
Color Table (RGB with 2 entries)
```

(continues on next page)

(continued from previous page)

```
0: 255,255,255,255
1: 0,0,0,255
```

5. **(Optional)** Optimization methods described here are similar to that described above:
 - The overview creation method is equal to that described above.
 - For creating the pyramid the commands to use are the same as described above with the addition of the **-co “NBITS=1”** command.
6. Displaying the result on GeoServer:
 - Result as a pyramid (Zoom on the image for seeing the result).
 - Result with overviews (Zoom on the image for seeing the result).

Advanced Vectorial Data Management

This module presents working with vector data, how to obtain vector data information, filter, extract and update.

In this module you will learn how to:

Retrieving vector data and metadata

In this section we will learn how to deal with vector data using WFS. First we will learn how to deal with metadata and then how to retrieve the features. We will be using the layer named `Counties` in the workshop namespace.

Note: The Open Geospatial Consortium Web Feature Service Interface Standard (WFS) provides an interface allowing requests for geographical features across the web using platform-independent calls. One can think of geographical features as the “source code” behind a map, whereas the WMS interface or online mapping portals like Google Maps return only an image, which end-users cannot edit or spatially analyze.

1. Navigate to the GeoServer [Welcome Page](#).
2. On the Welcome page locate the *Layer Preview* link (no need to login).
3. Navigate to the WFS GML output of the *Counties* layer.

Depending on the browser, the output may be unformatted or recognized as XML. Here is what Firefox 3 shows: <http://localhost:8083/geoserver/ows?service=WFS&version=1.0.0&request=GetFeature&typeName=geosolutions:Counties&maxFeatures=50&outputFormat=GML2>

Note: We recommend the Mozilla Firefox web browser for navigating WFS response documents.

4. Now that we know the quick and easy way to get WFS data, let’s go back and do it the way a standard WFS client works. First, the only thing expected to be known is the WFS server URL: <http://localhost:8083/geoserver/ows?service=WFS&version=1.0.0>

Using that URL, we can issue a `GetCapabilities` request in order to know which layer it contains and what operations are supported: <http://localhost:8083/geoserver/ows?service=WFS&version=1.0.0&request=GetCapabilities>

If we scroll below, we will find the `Counties` feature type described:



Scale = 1 : 5410

601912.51909, 4860232.72909

Click on the map to get feature info



Scale = 1 : 5410

601707.45189, 4860324.56535

Click on the map to get feature info

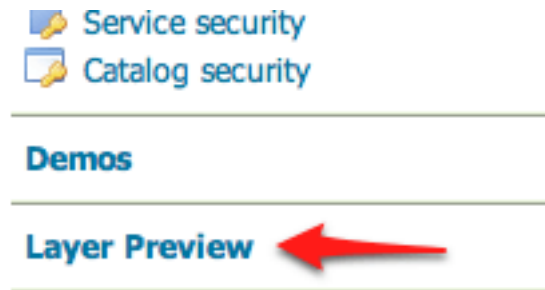


Fig. 244: Layer Preview

<< < 1 > >> Results 1 to 18 (out of 18 items)

Type	Name	Title	Common Formats	All Formats
	geosolutions:srtm	srtm	OpenLayers KML	Select one
	geosolutions:states	states	OpenLayers KML GML	Select one
	geosolutions:bptlandmarks	Point landmarks	OpenLayers KML GML	Select one
	geosolutions:bstreets	Boulder streets	OpenLayers KML GML	Select one
	geosolutions:Counties	Counties of Colorado	OpenLayers KML GML	Select one
	geosolutions:Parcels	Parcels	OpenLayers KML GML	Select one
	geosolutions:BoulderCityLimits	BoulderCityLimits	OpenLayers KML GML	Select one
	geosolutions:Trails	Trails	OpenLayers KML GML	Select one

Fig. 245: WFS GML output

```
-105.038697,40.349251 -105.038739,40.349251 -105.039041,40.349252 -105.040029,40.349256
-105.040274,40.349257 -105.040417,40.349257 -105.040832,40.349258 -105.042632,40.349261
-105.042923,40.349262 -105.044341,40.34926 -105.045742,40.349268 -105.047122,40.349267
-105.048508,40.34927 -105.049967,40.349274 -105.051793,40.349273 -105.052039,40.349273
-105.052679,40.349272 -105.052942,40.349272 -105.054221,40.349271 -105.056557,40.34928
-105.056628,40.34928 -105.05672,40.349281
</gml:coordinates>
</gml:LinearRing>
</gml:outerBoundaryIs>
</gml:Polygon>
</gml:polygonMember>
</gml:MultiPolygon>
</geosolutions:the_geom>
<geosolutions:STATEFP10>08</geosolutions:STATEFP10>
<geosolutions:COUNTYFP10>069</geosolutions:COUNTYFP10>
<geosolutions:COUNTYNS10>00198150</geosolutions:COUNTYNS10>
<geosolutions:GEOID10>08069</geosolutions:GEOID10>
<geosolutions:NAME10>Larimer</geosolutions:NAME10>
<geosolutions:NAMELSAD10>Larimer County</geosolutions:NAMELSAD10>
<geosolutions:LSAD10>06</geosolutions:LSAD10>
<geosolutions:CLASSFP10>H1</geosolutions:CLASSFP10>
<geosolutions:MTFCC10>G4020</geosolutions:MTFCC10>
<geosolutions:CSAFP10/>
<geosolutions:CBSAFP10>22660</geosolutions:CBSAFP10>
<geosolutions:METDIVFP10/>
<geosolutions:FUNCSTAT10>A</geosolutions:FUNCSTAT10>
<geosolutions:ALAND10>6723613486</geosolutions:ALAND10>
<geosolutions:AWATER10>98295250</geosolutions:AWATER10>
<geosolutions:INTPTLAT10>+40.6630912</geosolutions:INTPTLAT10>
<geosolutions:INTPTLON10>-105.4821309</geosolutions:INTPTLON10>
</geosolutions:Counties>
</gml:featureMember>
- <gml:featureMember>
- <geosolutions:Counties fid="Counties.2">
- <geosolutions:the_geom>
- <gml:MultiPolygon srsName="http://www.opengis.net/gml/srs/epsg.xml#4269">
- <gml:polygonMember>
- <gml:Polygon>
- <gml:outerBoundaryIs>
- <gml:LinearRing>
- <gml:coordinates decimal=" " srs=" " text=" >
```

Fig. 246: Default WFS layer preview.

```

- <WFS_Capabilities version="1.0.0" xsi:schemaLocation="http://www.opengis.net/wfs
  http://localhost:8080/geoserver/schemas/wfs/1.0.0/WFS-capabilities.xsd">
- <Service>
  <Name>My GeoServer WFS</Name>
  <Title>GeoSolutions GeoServer workshop WFS</Title>
- <Abstract>
  This is the reference implementation of WFS 1.0.0 and WFS 1.1.0, supports all WFS operations
  including transactions.
</Abstract>
<Keywords>WFS, WMS, GEOSERVER</Keywords>
<OnlineResource>http://localhost:8080/geoserver/wfs</OnlineResource>
<Fees>NONE</Fees>
<AccessConstraints>NONE</AccessConstraints>
</Service>
- <Capability>
- <Request>
  - <GetCapabilities>
    - <DCPType>
      - <HTTP>
        <Get onlineResource="http://localhost:8080/geoserver/wfs?request=GetCapabilities"/>
      </HTTP>
    </DCPType>
  - <DCPType>
    - <HTTP>
      <Post onlineResource="http://localhost:8080/geoserver/wfs"/>
    </HTTP>
    </DCPType>
  </GetCapabilities>
- <DescribeFeatureType>
  - <SchemaDescriptionLanguage>
    <XMLSCHEMA/>
  </SchemaDescriptionLanguage>
  - <DCPType>
    - <HTTP>
      <Get onlineResource="http://localhost:8080/geoserver/wfs?request=DescribeFeatureType"/>
    </HTTP>
  </DCPType>
</DescribeFeatureType>
</Request>
</Capability>
</WFS_Capabilities>

```

Fig. 247: GetCapabilities response

```

- <FeatureType>
  <Name>geosolutions:Counties</Name>
  <Title>Counties of Colorado</Title>
  <Abstract/>
  <Keywords/>
  <SRS>EPSG:4269</SRS>
  <LatLongBoundingBox minx="-109.06" miny="36.992" maxx="-102.041"
    maxy="41.003"/>
</FeatureType>
- <FeatureType>
  <Name>geosolutions:blakes</Name>
  <Title>Lakes and other polygonal water entities</Title>
  <Abstract/>
  <Keywords/>
  <SRS>EPSG:4269</SRS>
  <LatLongBoundingBox minx="-105.692" miny="39.915" maxx="-105.055"
    maxy="40.262"/>
</FeatureType>

```

Annotations in the image:

- A red box highlights the `<Name>` and `<Title>` elements, with the text "Name and title" to its right.
- A red arrow points to the `<SRS>` element, with the text "SRS" above it.
- A red arrow points to the `<LatLongBoundingBox>` element, with the text "Bounding box" to its right.

Fig. 248: GetCapabilities response (Counties feature type)

5. Now let's request more information for the **Counties** layer using a **DescribeFeatureType** request:

```
http://localhost:8083/geoserver/ows?service=WFS&version=1.0.0&request=DescribeFeatureType&
typename=geosolutions:Counties
```

Which gives us information about the fields names and types as well as the geometry type, in this case `MultiPolygon`.

6. After that, we can issue a basic `GetFeature` request, that looks like this:

```
http://localhost:8083/geoserver/ows?service=WFS&version=1.0.0&request=GetFeature&
type=geosolutions:Counties&featureId=Counties.1
```

Note: Notice it's almost the same as the one that Geoserver generated, but it's requestin a single feature specifying its identifier via `featureId=Counties.1`

In the *next* section we will see how to filter the WFS output based on various attributes.

Filtering and Extracting vector data

WFS also defines mechanisms to only retrieve a subset of the data that matches some specified constraints.

1. Create a new `request.xml` file in the training root and past the following request into it:

```
<wfs:GetFeature xmlns:wfs='http://www.opengis.net/wfs'
  xmlns:ogc='http://www.opengis.net/ogc' service='WFS' version='1.0.0'>
  <Query typeName='geosolutions:WorldCountries'>
    <ogc:Filter>
      <ogc:FeatureId fid='WorldCountries.137' />
    </ogc:Filter>
  </Query>
</wfs:GetFeature>
```

(continues on next page)

```

- <xsd:schema elementFormDefault="qualified" targetNamespace="http://www.geo-solutions.it/workshop">
  <xsd:import namespace="http://www.opengis.net/gml" schemaLocation="http://localhost:8080/geoserver/schemas/gml/2.1.2/feature.xsd"/>
- <xsd:complexType name="CountiesType">
  - <xsd:complexContent>
    - <xsd:extension base="gml:AbstractFeatureType">
      - <xsd:sequence>
        <xsd:element maxOccurs="1" minOccurs="0" name="the_geom" nillable="true"
          type="gml:MultiPolygonPropertyType"/>
        <xsd:element maxOccurs="1" minOccurs="0" name="STATEFP10" nillable="true"
          type="xsd:string"/>
        <xsd:element maxOccurs="1" minOccurs="0" name="COUNTYFP10" nillable="true"
          type="xsd:string"/>
        <xsd:element maxOccurs="1" minOccurs="0" name="COUNTYNS10" nillable="true"
          type="xsd:string"/>
        <xsd:element maxOccurs="1" minOccurs="0" name="GEOID10" nillable="true"
          type="xsd:string"/>
        <xsd:element maxOccurs="1" minOccurs="0" name="NAME10" nillable="true"
          type="xsd:string"/>
        <xsd:element maxOccurs="1" minOccurs="0" name="NAMELSAD10" nillable="true"
          type="xsd:string"/>
        <xsd:element maxOccurs="1" minOccurs="0" name="LSAD10" nillable="true"
          type="xsd:string"/>
        <xsd:element maxOccurs="1" minOccurs="0" name="CLASSFP10" nillable="true"
          type="xsd:string"/>
        <xsd:element maxOccurs="1" minOccurs="0" name="MTFCC10" nillable="true"
          type="xsd:string"/>

```

Fig. 249: DescribeFeatureType response for Counties feature type

(continued from previous page)

```
</Query>
</wfs:GetFeature>
```

2. Go on the command line, move to the training folder root, and execute the request using CURL:

```
curl -XPOST -d @request.xml -H "Content-type: application/xml" "http://
↳localhost:8083/geoserver/ows"
```

3. Now, let's write an equivalent request - using the name of the state instead of the `id` - issuing a GET and encoding the filter in a language called CQL. Copy the following URL in your browser's navigation bar:

```
http://localhost:8083/geoserver/wfs?request=GetFeature&service=WFS&version=1.0.0&
↳typeName=geosolutions:WorldCountries&outputFormat=GML2&CQL_FILTER=NAME=%27Monaco
↳%27
```

Fig. 250: The CQL filter in the Firefox URL bar

That's how a feature set is filtered with either the OGC encoding or the CQL notation.

In the *next* section we will see how to edit the features via a protocol called WFS Transactional (WFS-T).

Modifying Feature Types

GeoServer provides a fully Transactional Web Feature Service (WFS-T) which enables users to insert/delete/modify the available FeatureTypes. This section shows a few of the GeoServer WFS-T capabilities and interactions with GIS clients.

Modifying Feature Types using GeoNode

1. Open your instance of **GeoNode** and log in as a *superuser* or a user having write rights on at least some *Layers*
2. Select a *Layer* on which when you have right to edit *data*

Warning: You can edit **only** *Layers* which have been stored on a JDBC DataStore, like a DataBase. On GeoNode this is only possible if the DB `datastore` has been enabled from the `settings`.

3. Click on *Edit Layer* and then, from the pop-up window, click on *Edit data*

Warning: The *Edit data* button will be available **only** for writable *Layers* (see above).

4. When the *Map* shows up along with your *Layer*, zoom in to a region you want to update or create.
5. Identify the *Edit* button on the map top toolbar, click on the **small arrow** on the left in order to show up the context menu.
6. Lets first *Modify* a FeatureType. Click on **Modify**.
7. Select a geometry and click over it. From the small info dialog window, select **Edit**


```


- <wfs:FeatureCollection xsi:schemaLocation="http://www.geo-solutions.it/workshop
http://localhost:8080/geoserver/wfs?service=WFS&version=1.0.0&request=DescribeFeatureType&
typeName=geosolutions%3AWorldCountries http://www.opengis.net/wfs http://localhost:8080/geoserver
/schemas/wfs/1.0.0/WFS-basic.xsd">
- <gml:boundedBy>
  <gml:null>unknown</gml:null>
</gml:boundedBy>
- <gml:featureMember>
- <geosolutions:WorldCountries fid="WorldCountries.137">
  - <geosolutions:the_geom>
    - <gml:MultiPolygon srsName="http://www.opengis.net/gml/srs/epsg.xml#4326">
      - <gml:polygonMember>
        - <gml:Polygon>
          - <gml:outerBoundaryIs>
            - <gml:LinearRing>
              - <gml:coordinates decimal="." cs="," ts=" ">
                7.43869876,43.75045645 7.37772057,43.73174958 7.38009769,43.75324698
                7.3949805,43.76533926 7.41441084,43.77092032 7.43694177,43.76146353
                7.43869876,43.75045645
              </gml:coordinates>
            </gml:LinearRing>
          </gml:outerBoundaryIs>
        </gml:Polygon>
      </gml:polygonMember>
    </gml:MultiPolygon>
  </geosolutions:the_geom>
  <geosolutions:ScaleRank>6</geosolutions:ScaleRank>
  <geosolutions:LabelRank>6</geosolutions:LabelRank>
  <geosolutions:FeatureCla>Admin-0 countries</geosolutions:FeatureCla>
  <geosolutions:SOVEREIGNT>Monaco</geosolutions:SOVEREIGNT>
  <geosolutions:SOV_A3>MCO</geosolutions:SOV_A3>
  <geosolutions:ADM0_DIF>0.0</geosolutions:ADM0_DIF>
  <geosolutions:LEVEL>2.0</geosolutions:LEVEL>
  <geosolutions:TYPE>Sovereign country</geosolutions:TYPE>


```

Fig. 251: The results of the CQL filter

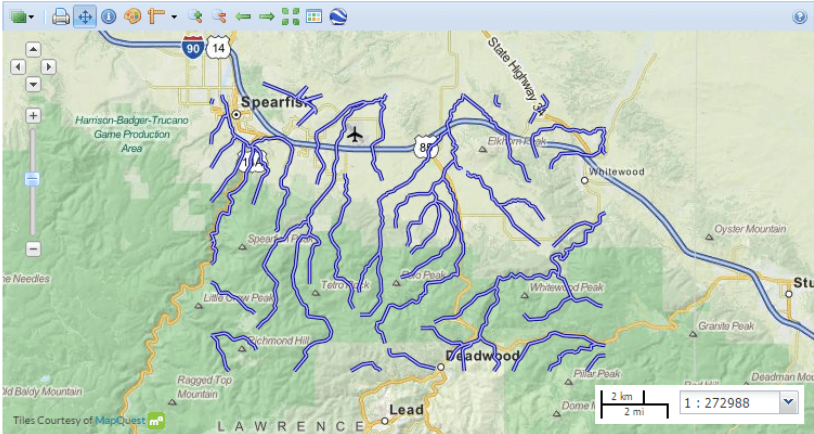
The screenshot shows the GeoNode 'Explore Layers' page. At the top is a dark navigation bar with the GeoNode logo, tabs for 'Layers', 'Maps', 'Documents', 'People', and 'Groups', a search bar with the placeholder 'Type your search here ...', and a user profile icon labeled 'admin'. Below the navigation bar is the 'Explore Layers' section. On the left is a sidebar with filters: 'Your selections' (with a 'Clear all filters' link), 'TEXT' (expanded), a search input 'Enter your text here ...', 'TYPE' (with 'Raster' (2) and 'Vector' (24) options), 'CATEGORIES', 'KEYWORDS', 'OWNERS', 'DATE', 'REGIONS', and 'EXTENT'. The main content area shows a list of layers. At the top of this area is a 'Total: 26' count and sorting options: 'Most recent' (selected), 'Less recent', 'A - Z', 'Z - A', and 'Most popular'. There is also a 'Set permissions' button. The list of layers includes: 1. '_2014_cc_omega_2' by u0899xga, No abstract provided, 9 Oct 2015, 0 views, 0 shares, 0 stars. 2. '_2014_cc_omega_1' by u0899xga, No abstract provided, 9 Oct 2015, 2 views, 0 shares, 0 stars. 3. '_2014_cc_omega' by u0899pfe, No abstract provided. Each layer entry has a thumbnail map showing a spatial distribution of points (purple squares, yellow crosses, and teal triangles respectively) and a 'Select' button.

Fig. 252: GeoNode Layers


[Layers](#)
[Maps](#)
[Documents](#)
[People](#)
[Groups](#)




streams_1



[Download Layer](#)
[Edit Layer](#)
[Download Metadata](#)

Legend



Maps using this layer

This layer is not currently used in any maps.


Create a map using this layer

Click the button below to generate a new map based on this layer.

[Create a Map](#)

Styles

The following styles are associated with this layer. Choose a style to view it in the preview map.

-  streams_1

[Info](#)
[Attributes](#)
[Share](#)
[Ratings](#)
[Comments](#)

Title	streams_1
Abstract	No abstract provided
Publication Date	Oct. 8, 2015, 6:59 a.m.
Type	Vector Data
Owner	admin
More info	-

Fig. 253: GeoNode Layer Select

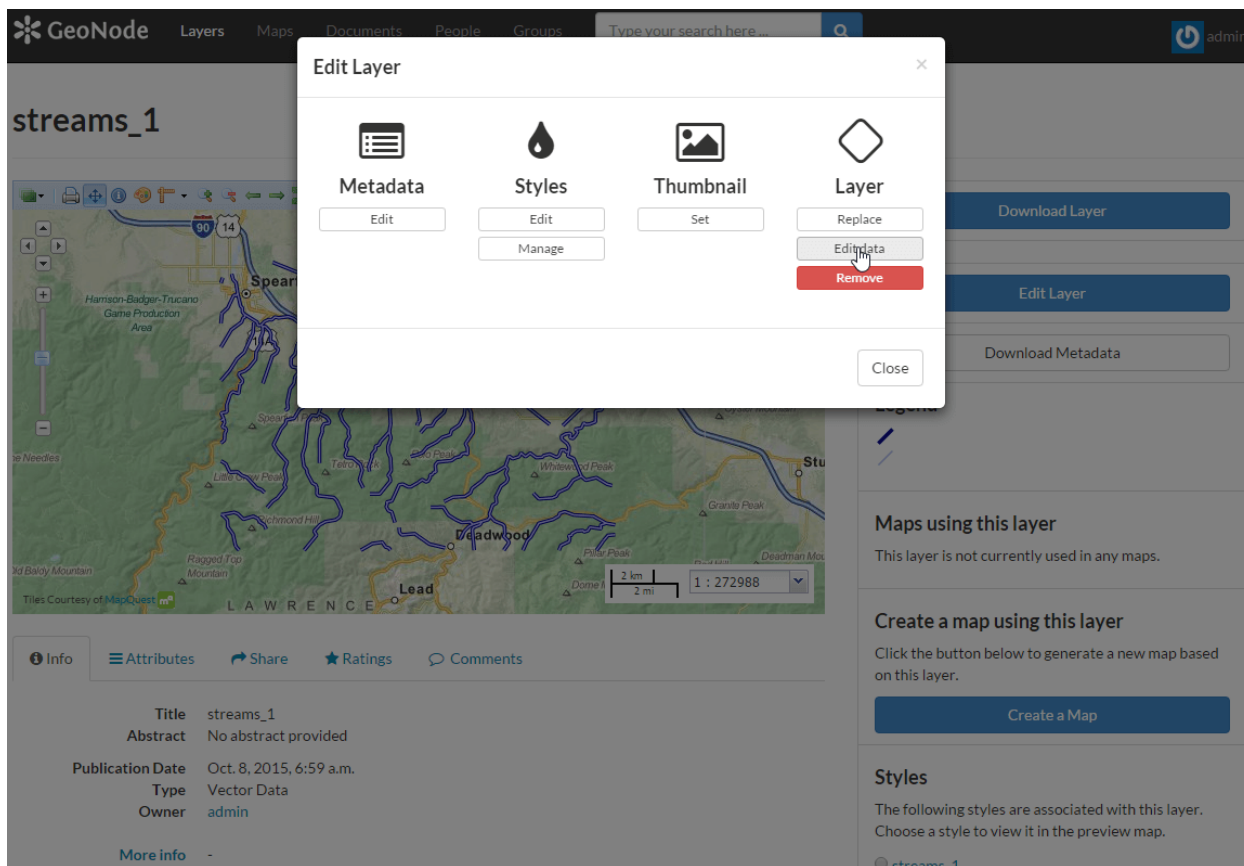


Fig. 254: GeoNode Edit Layer

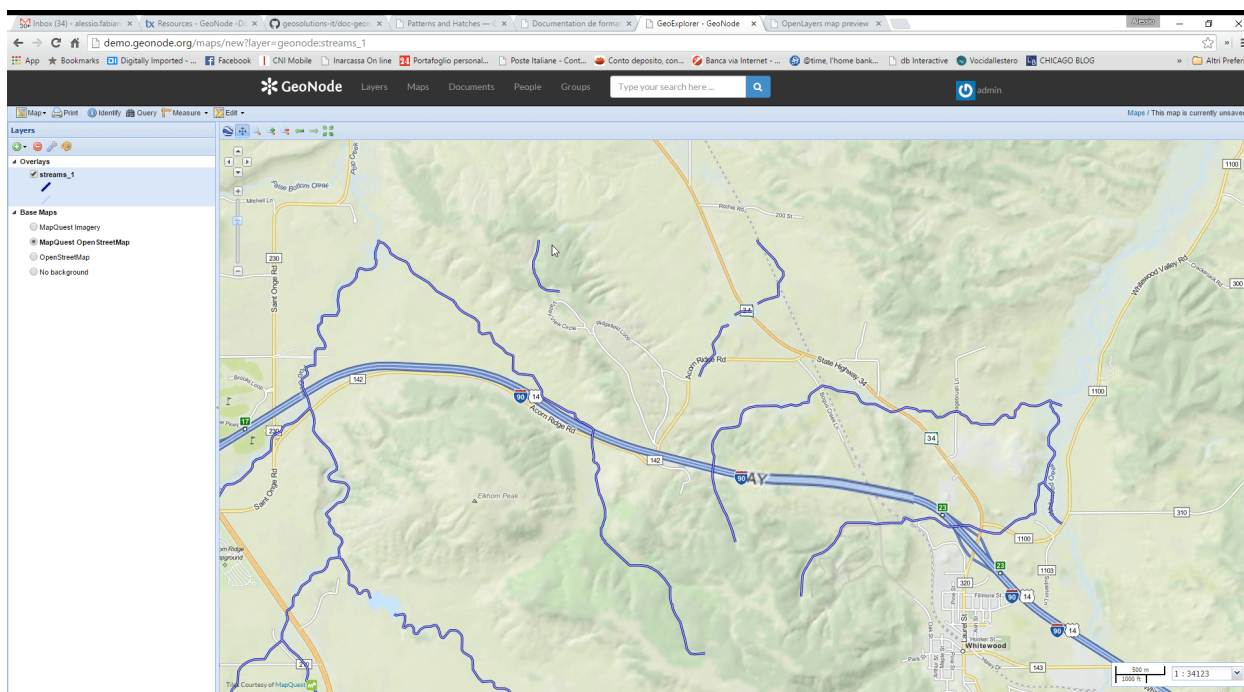


Fig. 255: GeoNode Navigate Layer

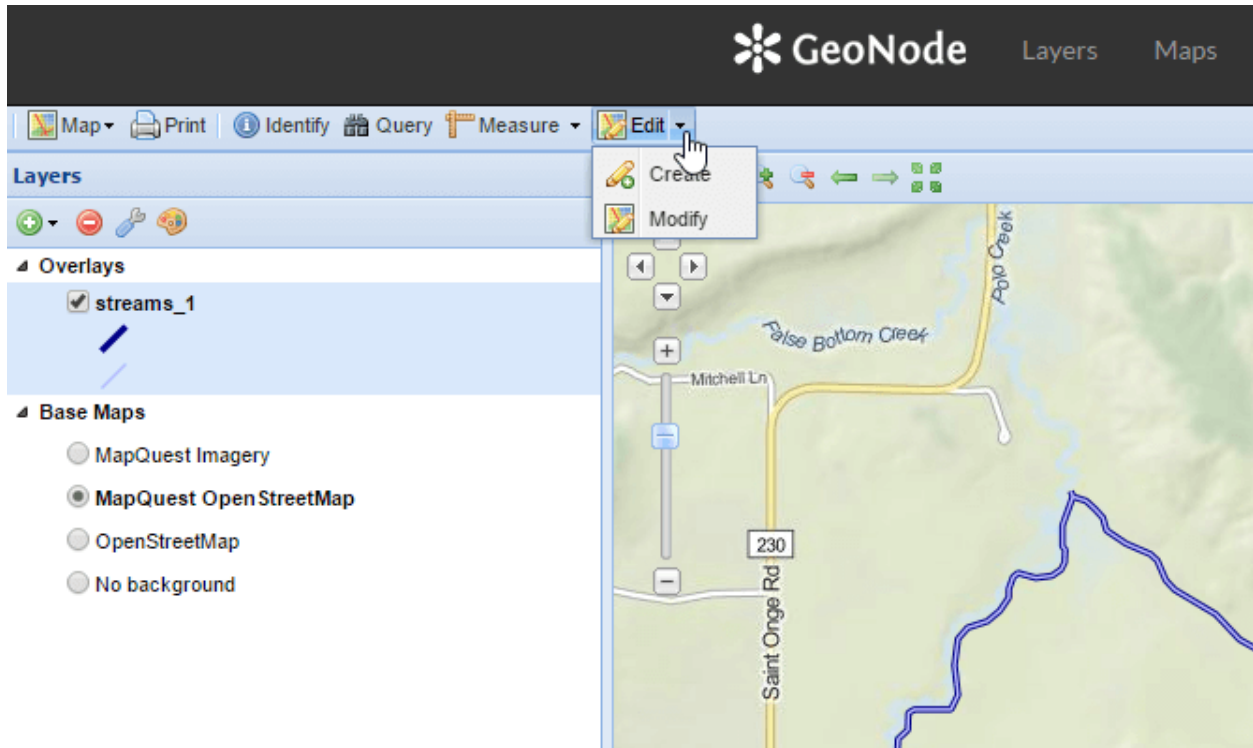


Fig. 256: GeoNode Edit Button

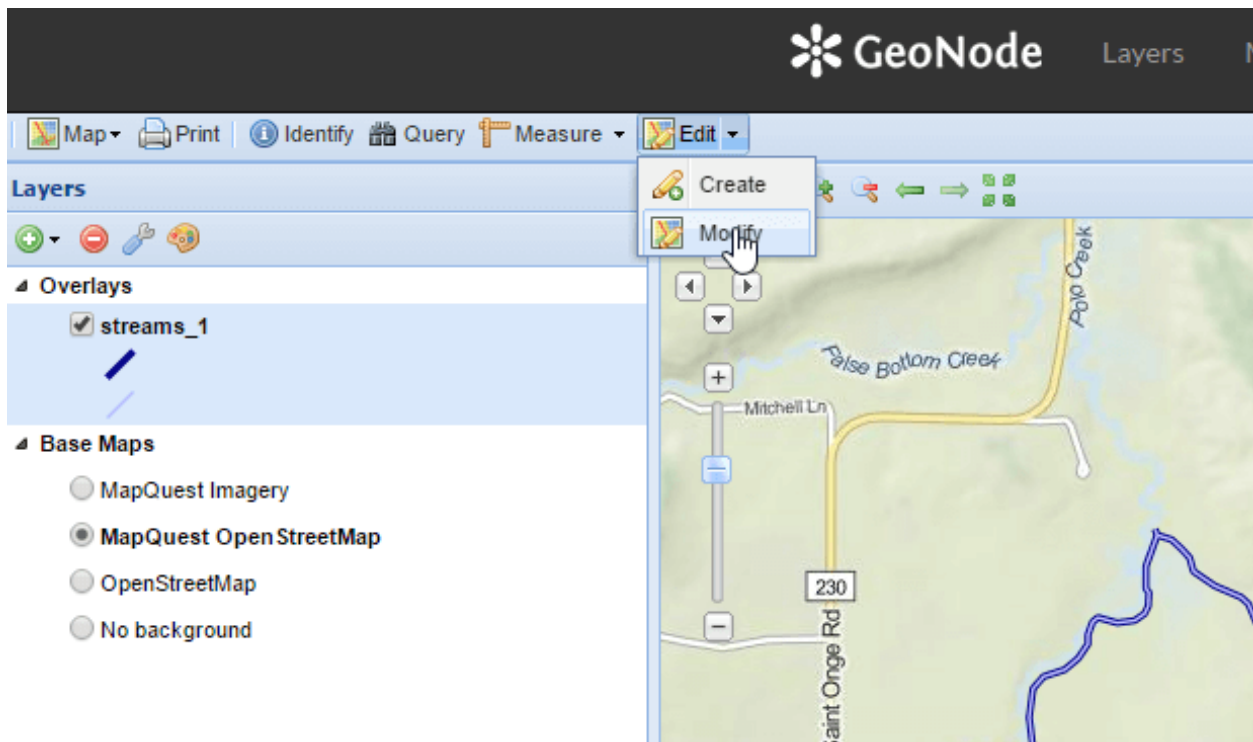


Fig. 257: GeoNode Modify FeatureType

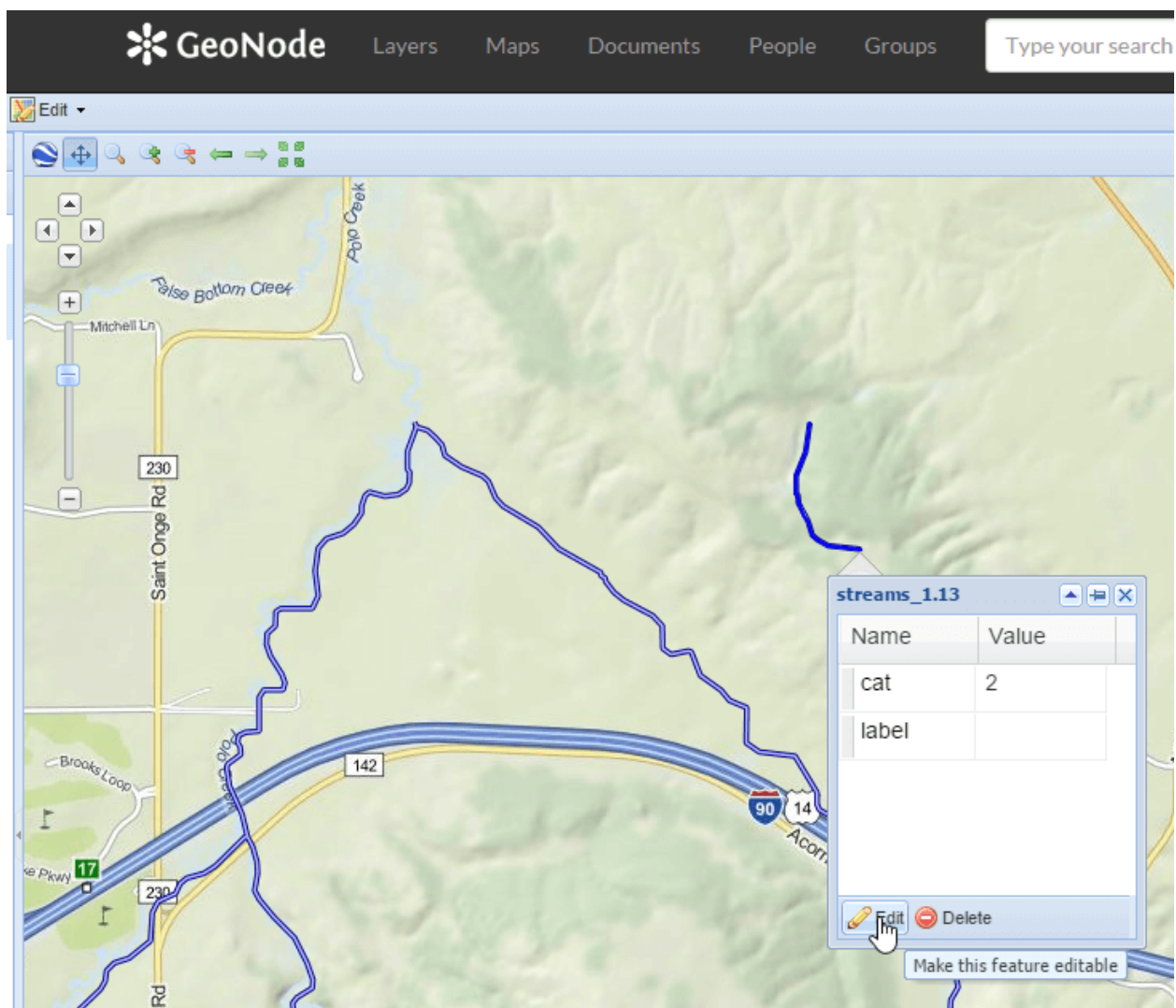


Fig. 258: GeoNode Editing a FeatureType

8. Modify the geometry and/or the values of the field as you wish, and then click on **Save**.

Hint: If you want you can also completely delete the FeatureType by clicking on the **Delete** button from the same info dialog window.

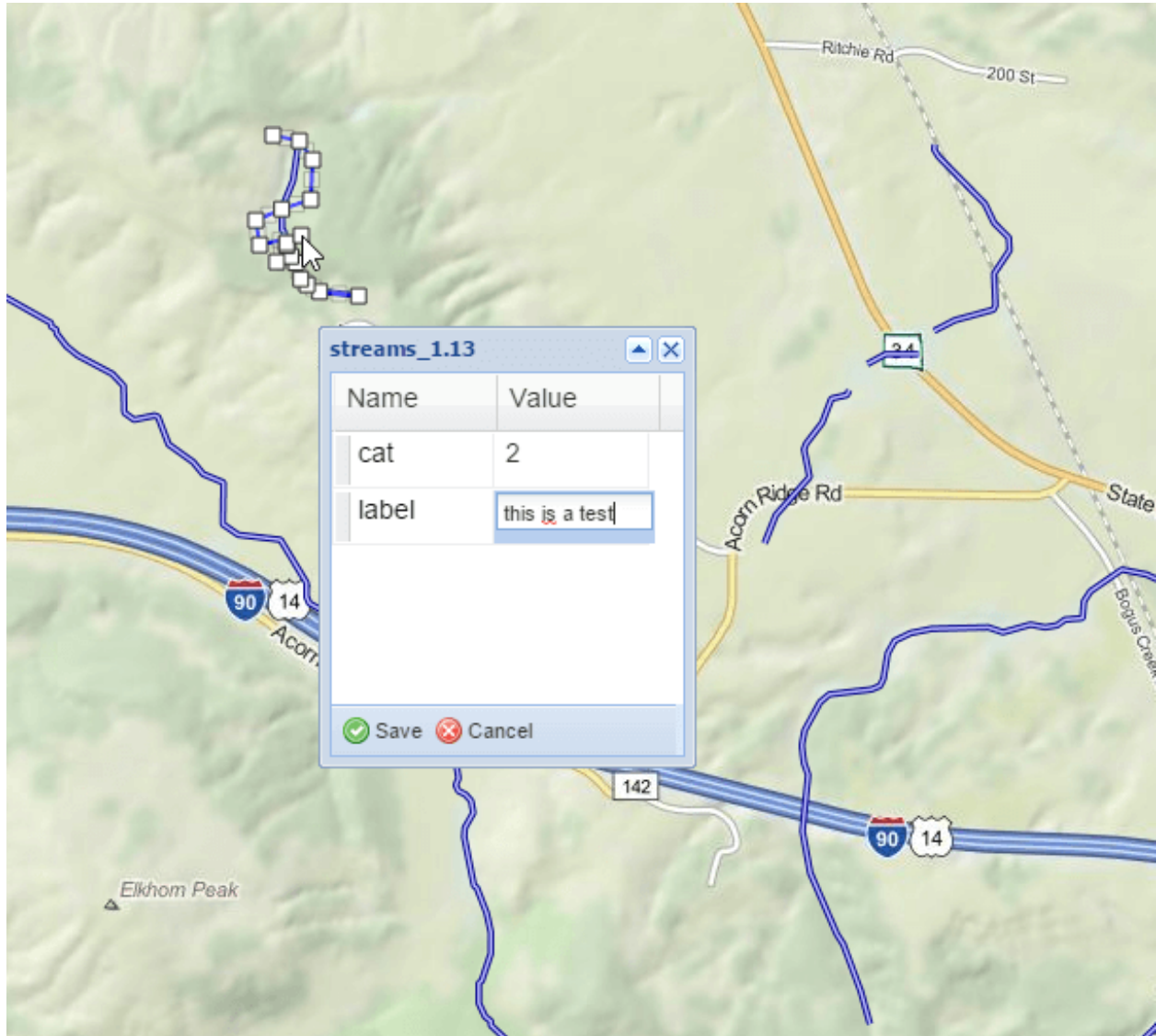


Fig. 259: GeoNode Updating a FeatureType

9. Verify that the changes have been stored on GeoServer.

Replace the URL

```
http://your_host/maps/new?layer=geonode:streams_1
```

with

```
http://your_host/geoserver/wms/reflect?layers=geonode:streams_1
```

Warning: Pay attention to the parameter: `layer` becomes `layers`, plural. If you want you can also add an **output format** parameter, like `format=openlayers`. In that case the complete URL becomes:

http://your_host/geoserver/wms/reflect?layers=geonode:streams_1&format=openlayers

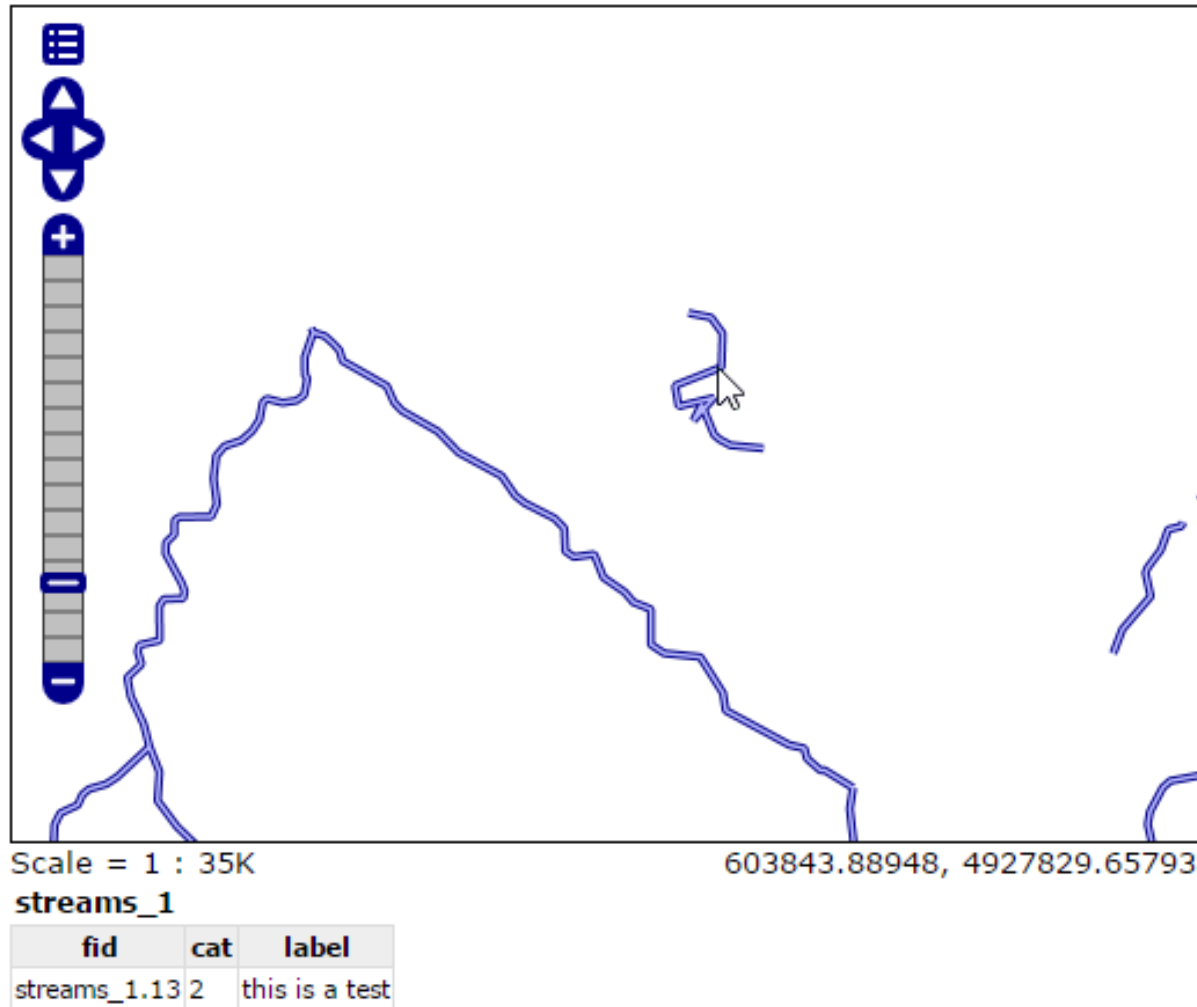


Fig. 260: GeoServer Displaying the Updated Layer

Click over the FeatureType in order to display the updates values too.

10. Repeat the FeatureType editing but this time click on **Create** (or simply click over the *Edit* button and **not** on its right small arrow).

Modifying Feature Types using a Desktop GIS client

1. Open **uDig** GIS desktop client by going on the command line, changing directory in the training root if necessary, and running the `udig` command.

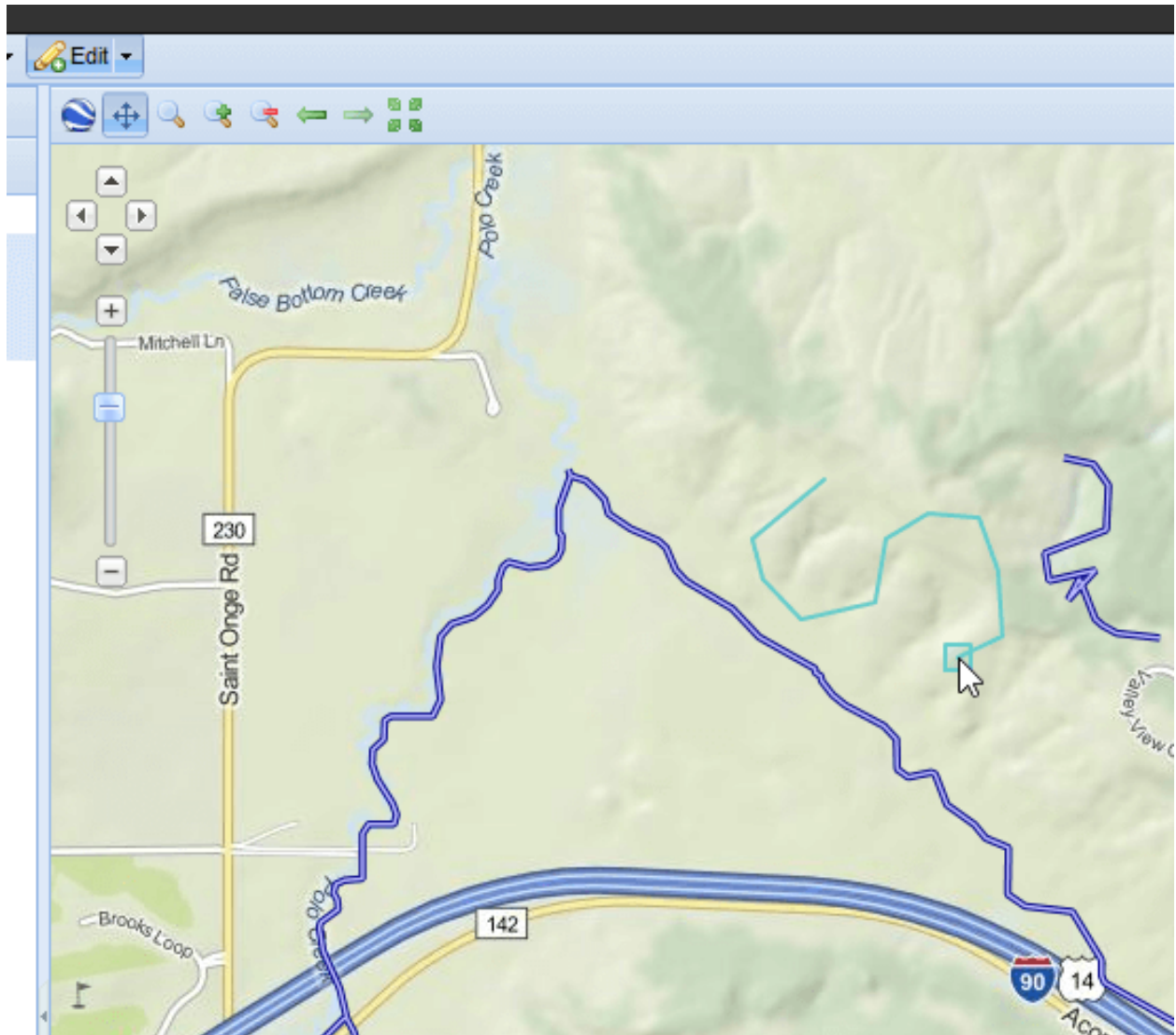


Fig. 261: GeoNode Creating a FeatureType

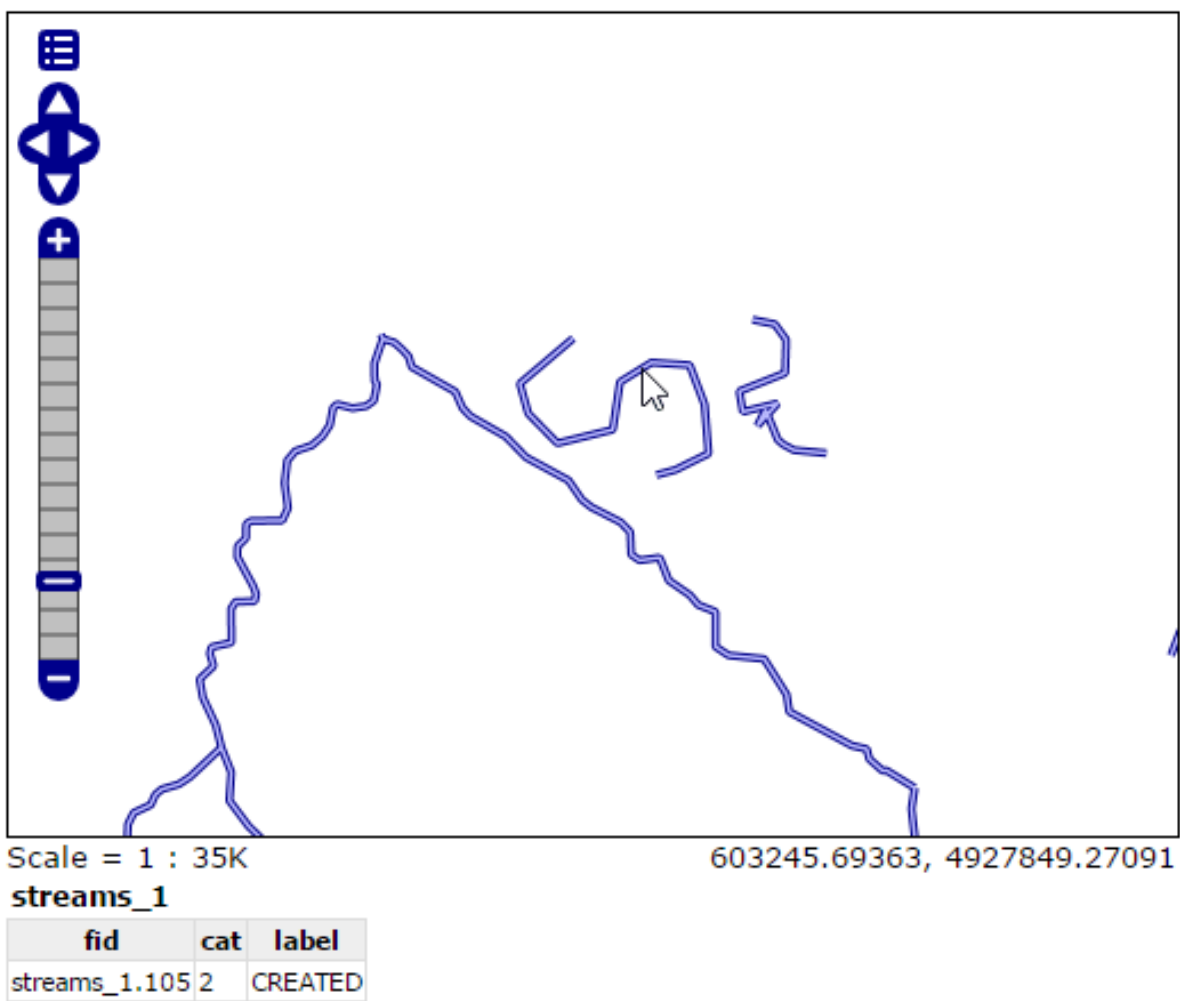
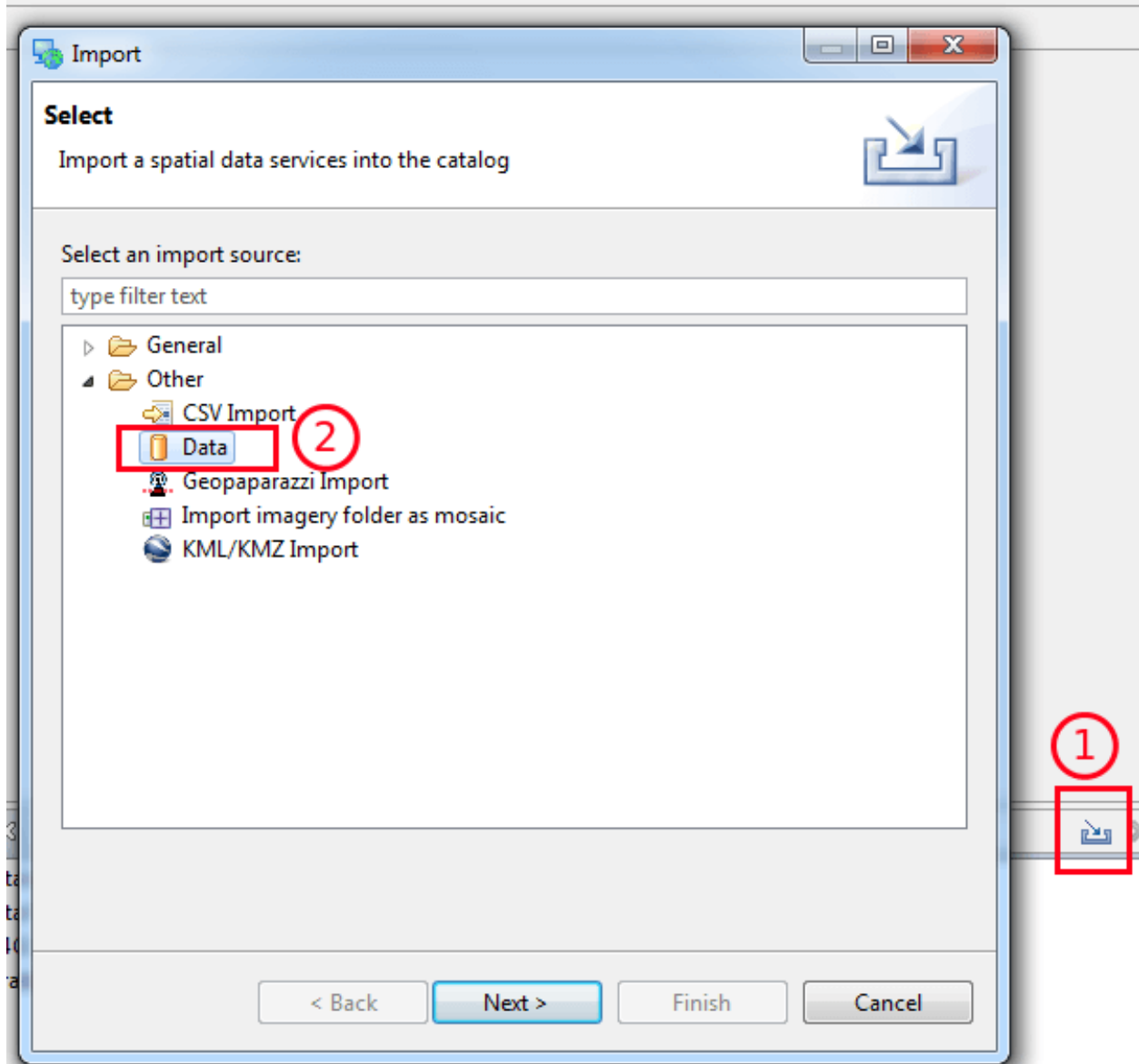


Fig. 262: GeoServer Displaying the New Feature

2. Add GeoServer WFS to the catalog.



Use the import button in the catalog tab, and select “data” in the first page of the wizard

Insert into the URL text box the following address:

```
http://localhost:8083/geoserver/wfs?request=GetCapabilities&service=WFS
```

Select the *Mainrd* from the list

3. Load the *Mainrd* Feature Type using *drag-n-drop*.
4. Perform a zoom operation on the upper-right part of the layer.
5. By using the *Select and Edit Geometry* tool try to move/add/remove some vertex to the small line at the center of the screen.
6. Once finished use the *Commit* tool to persist the changes on GeoServer.

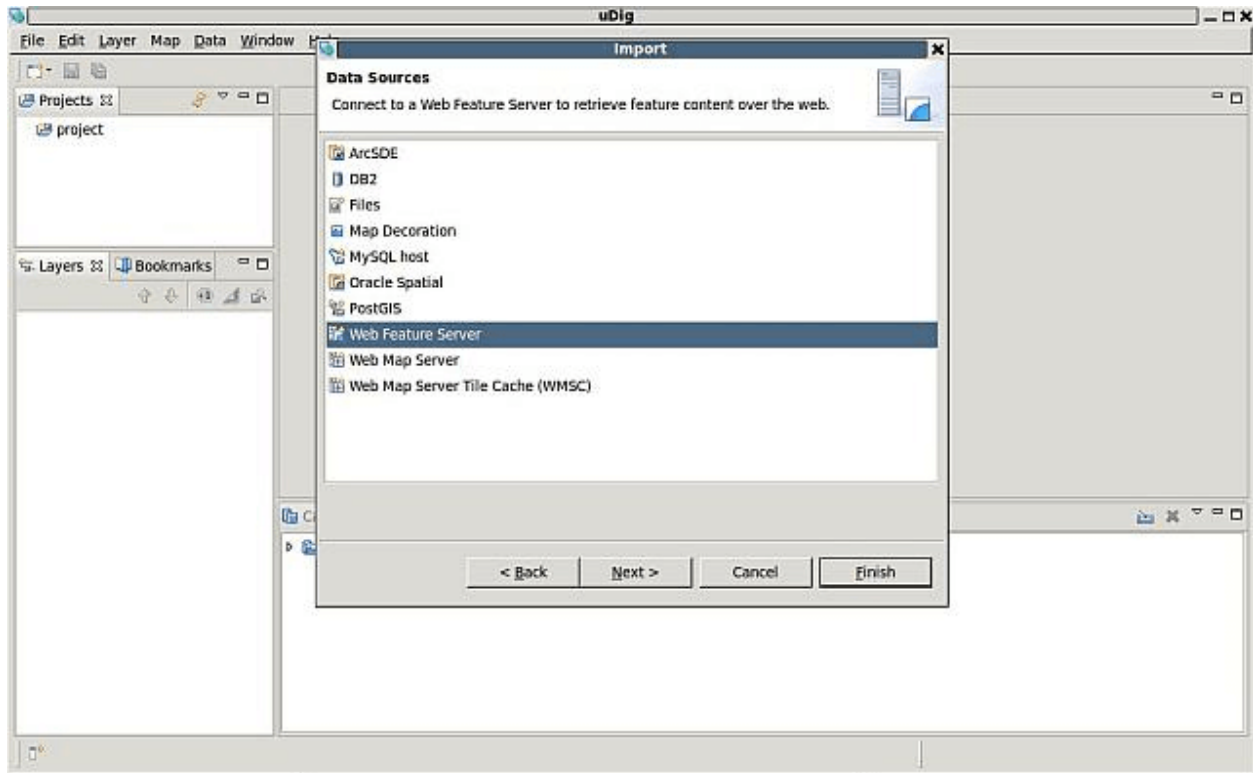


Fig. 263: Selection of Web Feature Service data

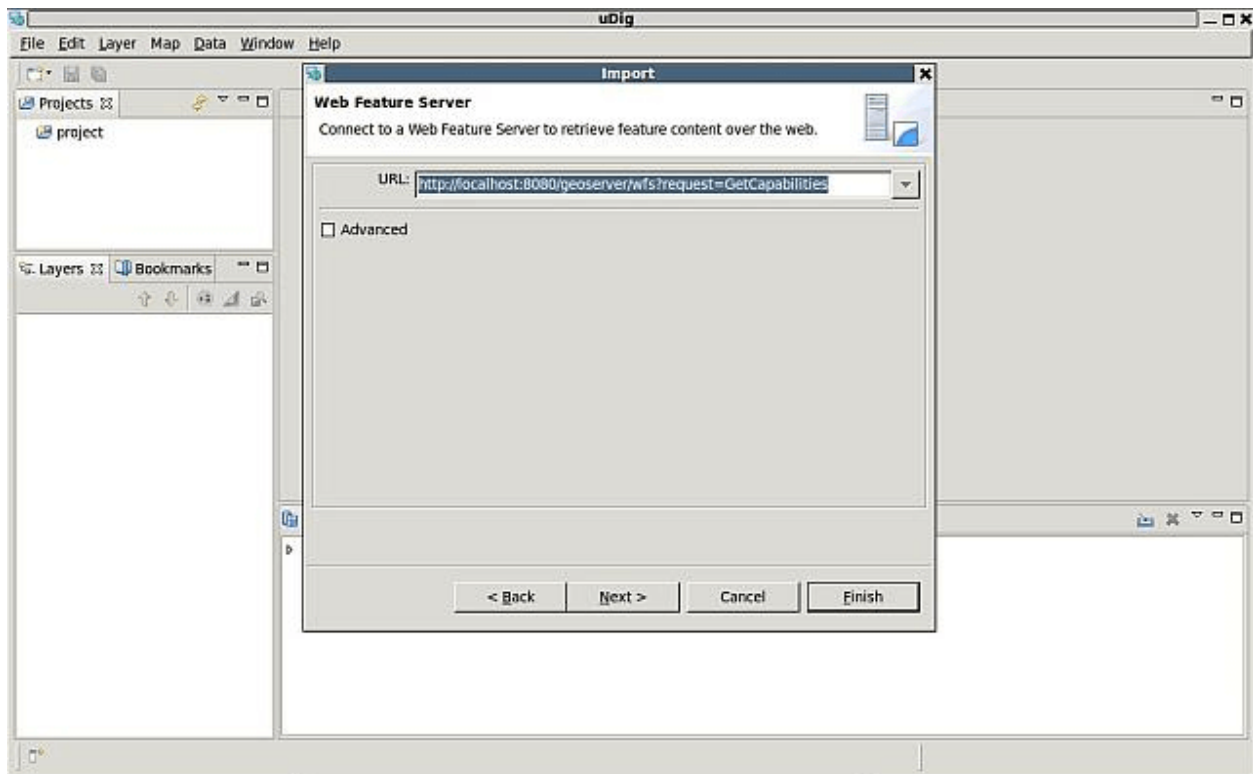


Fig. 264: The WFS URL

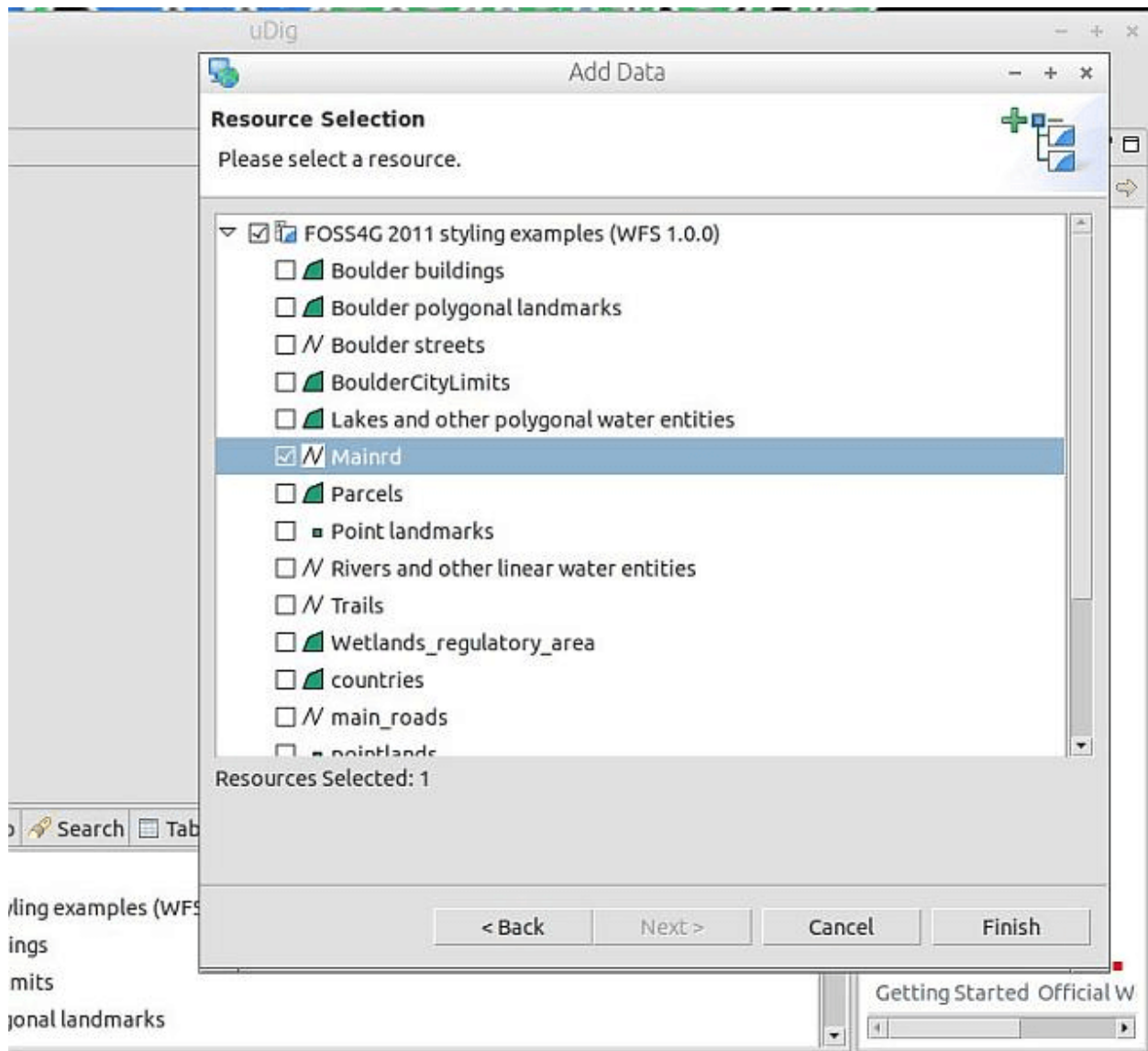
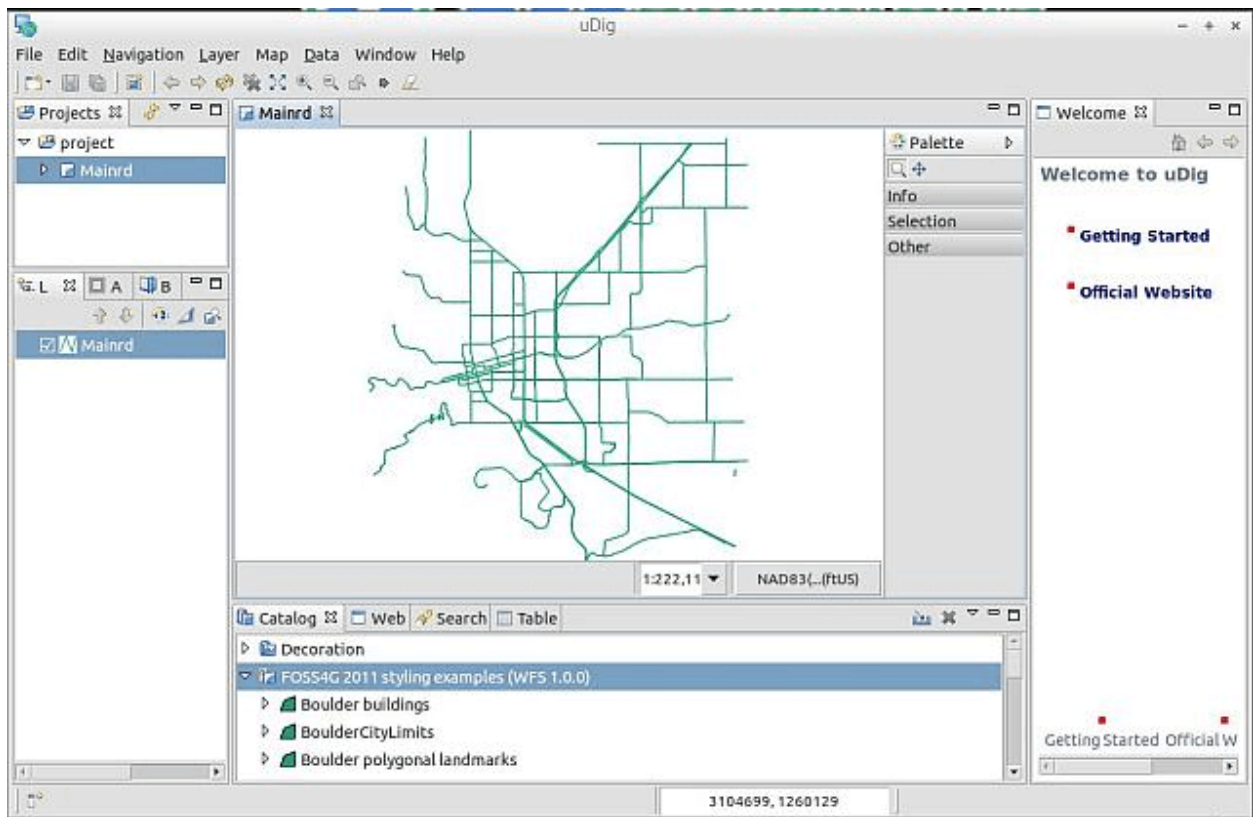


Fig. 265: WFS Datasets shown into the uDig catalog

Fig. 266: Importing *Mainrd* into the map

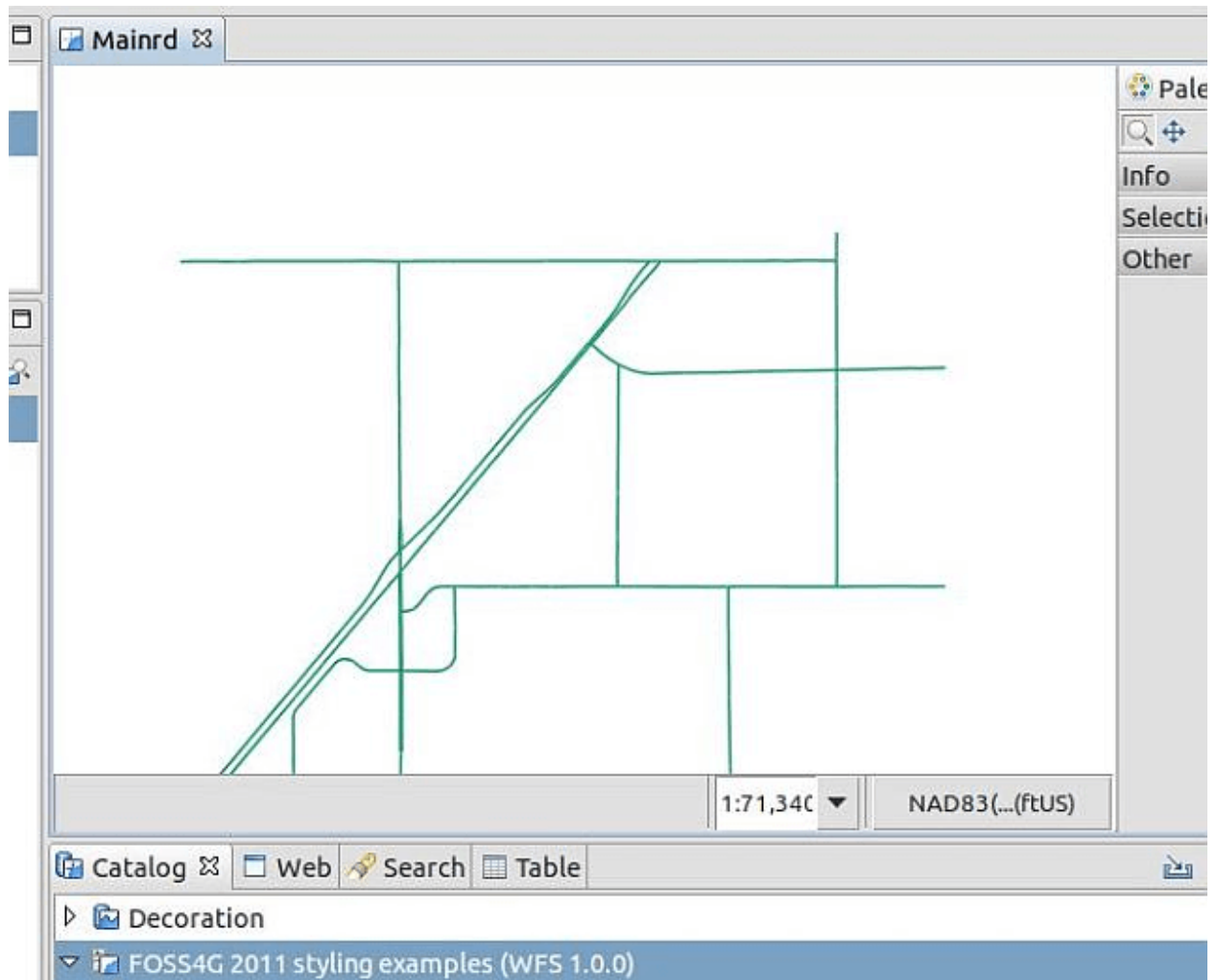


Fig. 267: Zooming in ...

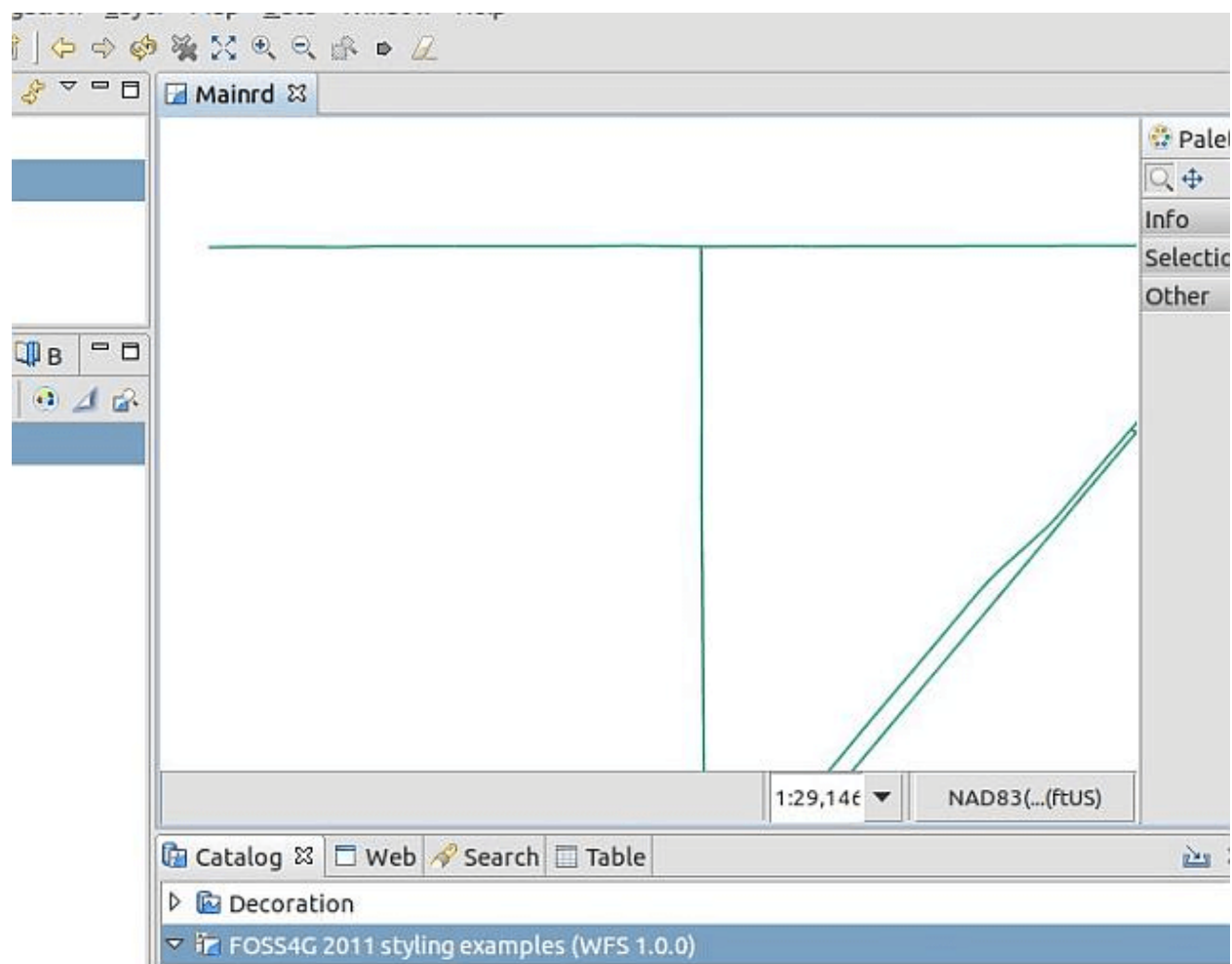


Fig. 268: Zooming in ...

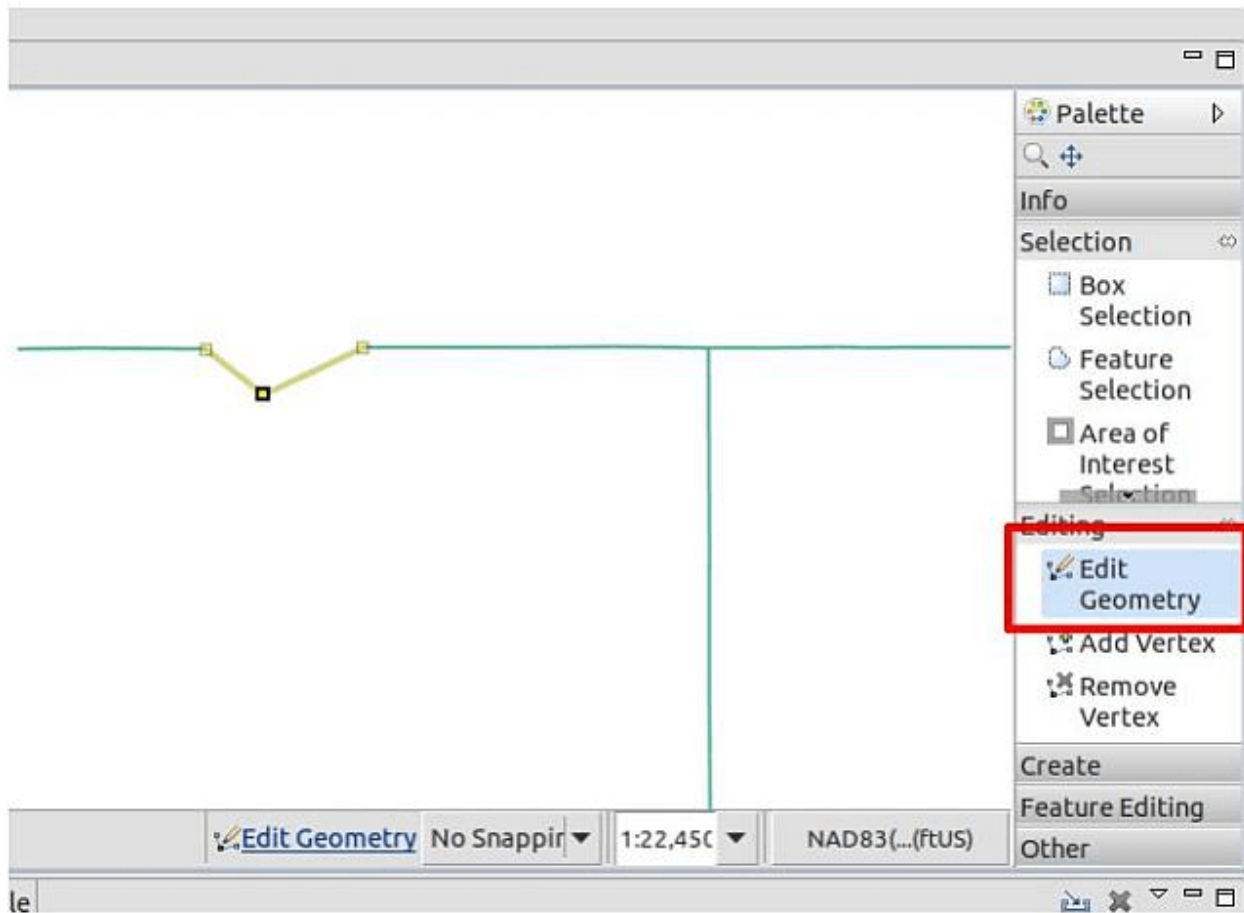


Fig. 269: Playing with the Geometry

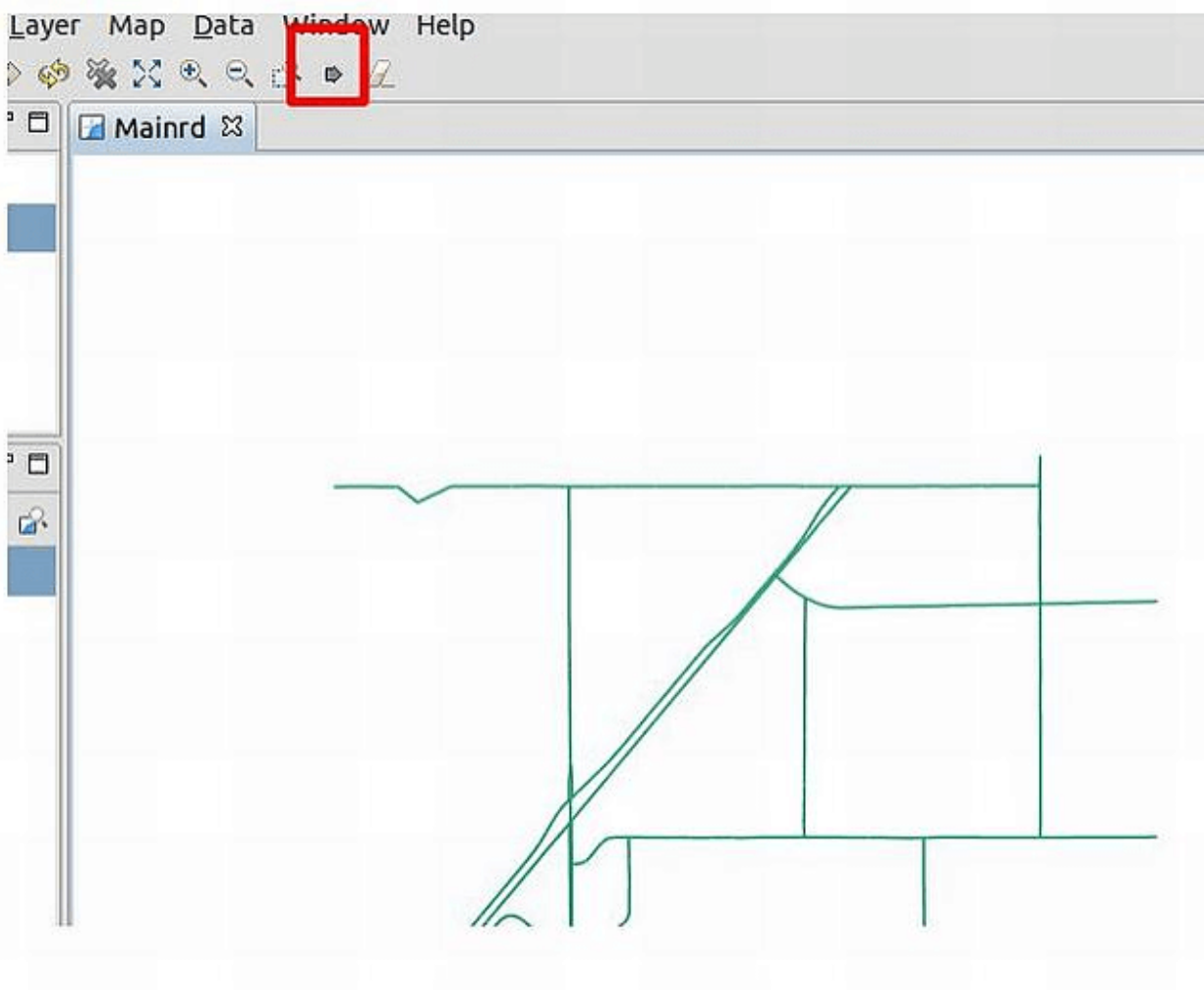


Fig. 270: Committing changes through the WFS-T protocol

7. Use GeoServer **Layer Preview** to view the changes on the *Mainrd* layer.

Warning: In order to view the streets lines you have to specify the *line* style on the GetMap request.

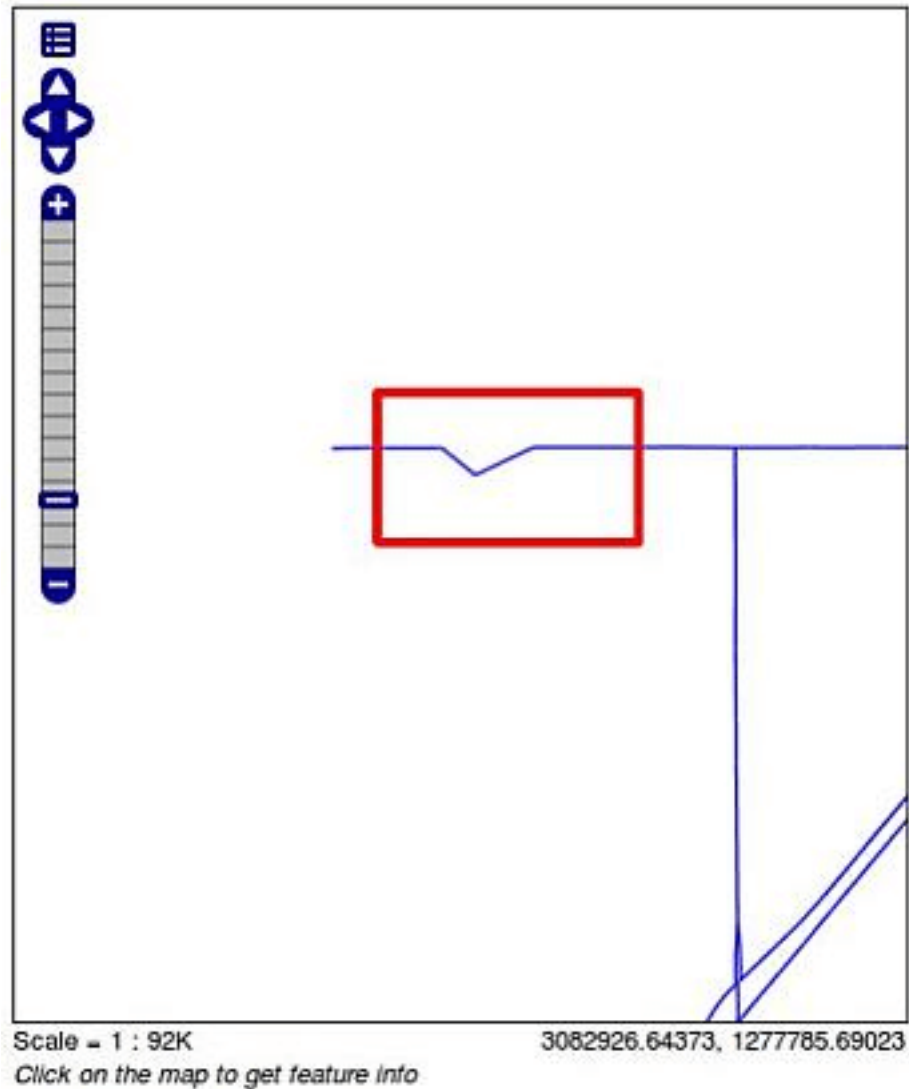


Fig. 271: Showing the changes to the *Mainrd* Feature Type

8. On uDig look the Feature attribute values using the *Info* tool.
9. Now open/create the `request.xml` file in the training root directory and set in the following request, which will be used to issue an **Update** Feature type request to the WFS-T updating all roads labelled as Monarch Rd to Monarch Road

```
<wfs:Transaction xmlns:topp="http://www.openplans.org/topp" xmlns:ogc="http://www.
↪opengis.net/ogc" xmlns:wfs="http://www.opengis.net/wfs" service="WFS" version=
↪"1.0.0">
  <wfs:Update typeName="geosolutions:Mainrd">
    <wfs:Property>
```

(continues on next page)

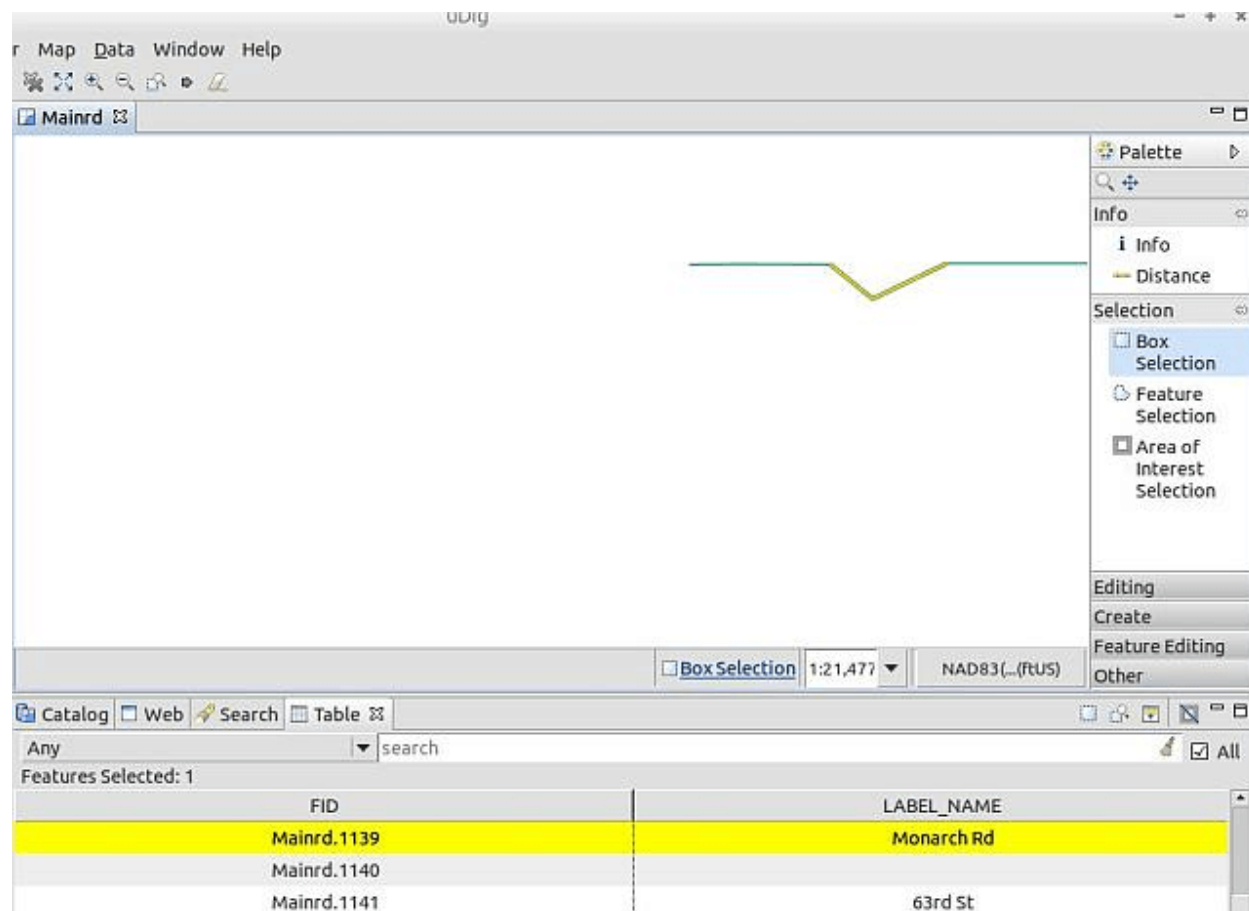


Fig. 272: Retrieving Feature Type info from uDig interface

(continued from previous page)

```

    <wfs:Name>LABEL_NAME</wfs:Name>
    <wfs:Value>Monarch Road</wfs:Value>
  </wfs:Property>
  <ogc:Filter>
    <ogc:PropertyIsEqualTo>
      <ogc:PropertyName>LABEL_NAME</ogc:PropertyName>
      <ogc:Literal>Monarch Rd</ogc:Literal>
    </ogc:PropertyIsEqualTo>
  </ogc:Filter>
</wfs:Update>
</wfs:Transaction>

```

10. Issue the WFS-T request towards GeoServer using curl on the command line:

```

curl -XPOST -d @request.xml -H "Content-type: application/xml" "http://
localhost:8083/geoserver/ows"

```

11. The response should be a TransactionResponse XML document containing a `wfs:SUCCESS` element
12. Ask the info again using the uDig *Info* tool ...

Note: In order to issue a GetFeatureInfo request from the OpenLayers MapPreview tool, just left-click over the line.

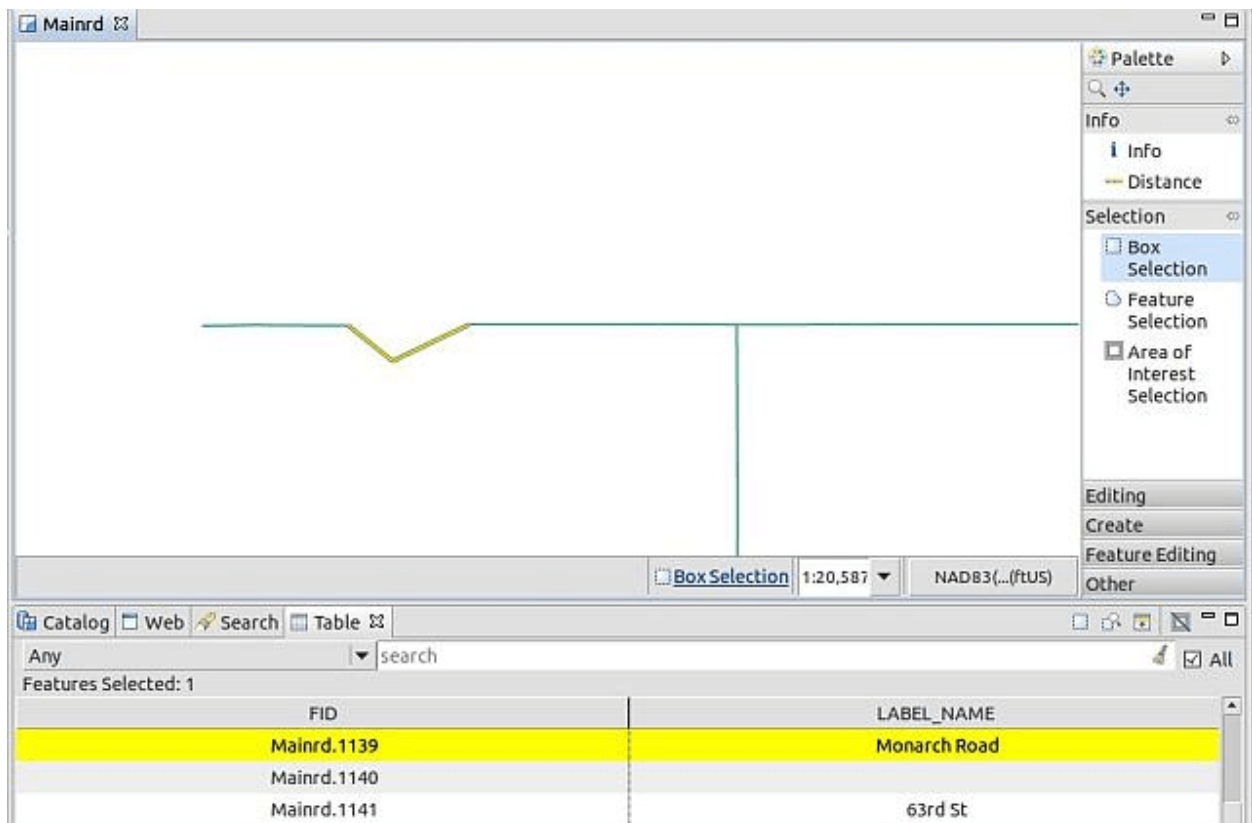


Fig. 273: Obtaining the updated Feature Type info from uDig interface

13. Finally, obtain the Feature type info using the GetFeatureInfo operation issued directly by the [Map Preview](#) .

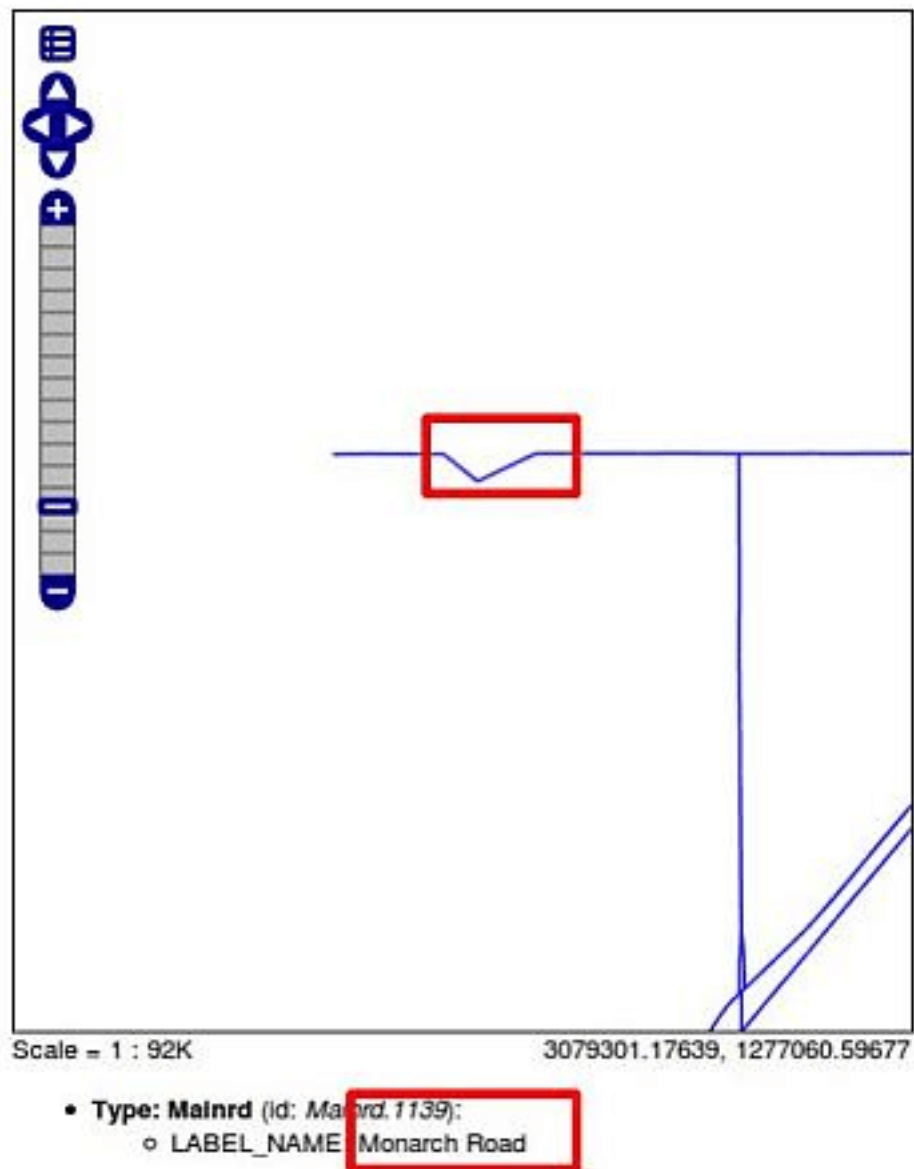


Fig. 274: Obtaining the updated Feature Type info from OpenLayers MapPreview GetFeatureInfo

Loading OSM Data into GeoNode Learn how to load OSM Data into GeoNode

Adding Data to GeoServer Learn how to add data to GeoServer.

Pretty maps with GeoServer Learn how to create pretty styles for the Maps in GeoServer.

Advanced Raster Data Management Learn advanced techniques for the delivery of Raster Data with GeoServer.

Advanced Vectorial Data Management Learn advanced techniques for the delivery of Vectorial Data with GeoServer.

spatial_processing Learn how to do Spatial Processing using external tools.

3.2.1.6.2 GeoNode Advanced Configuration

Here you will find information about every component of Geonode, such as GeoServer, geonode settings, security, etc.

Settings

GeoNode Django Apps

Make a GeoNode release

Settings

Here's a list of settings available in GeoNode and their default values. This includes settings for some external applications that GeoNode depends on.

Documents settings

Here's a list of settings available for the Documents app in GeoNode.

ALLOWED_DOCUMENT_TYPES

Default: `['doc', 'docx', 'xls', 'xlsx', 'pdf', 'zip', 'jpg', 'jpeg', 'tif', 'tiff', 'png', 'gif', 'txt']`

A list of acceptable file extensions that can be uploaded to the Documents app.

MAX_DOCUMENT_SIZE

Default: 2

Metadata settings

CATALOGUE

A dict with the following keys:

- **ENGINE:** The CSW backend (default is `geonode.catalogue.backends.pycsw_local`)
- **URL:** The FULLY QUALIFIED base URL to the CSW instance for this GeoNode
- **USERNAME:** login credentials (if required)

- **PASSWORD:** login credentials (if required)

pycsw is the default CSW enabled in GeoNode. pycsw configuration directives are managed in the PYCSW entry.

PYCSW

A dict with pycsw's configuration. Of note are the sections `metadata:main` to set CSW server metadata and `metadata:inspire` to set INSPIRE options. Setting `metadata:inspire['enabled']` to `true` will enable INSPIRE support. Server level configurations can be overridden in the `server` section. See <http://docs.pycsw.org/en/latest/configuration.html> for full pycsw configuration details.

MODIFY_TOPICCATEGORY

Default: `False`

Metadata Topic Categories list should not be modified, as it is strictly defined by ISO (See: <http://www.isotc211.org/2005/resources/CodeList/gmxCodeLists.xml> and check the `<CodeListDictionary gml:id="MD_MD_TopicCategoryCode">` element).

Some customisation it is still possible changing the `is_choice` and the GeoNode description fields.

In case it is absolutely necessary to add/delete/update categories, it is possible to set the `MODIFY_TOPICCATEGORY` setting to `True`.

Maps settings

DEFAULT_MAP_BASE_LAYER

The name of the background layer to include in newly created maps.

DEFAULT_MAP_CENTER

Default: `(0, 0)`

A 2-tuple with the latitude/longitude coordinates of the center-point to use in newly created maps.

DEFAULT_MAP_ZOOM

Default: `0`

The zoom-level to use in newly created maps. This works like the OpenLayers zoom level setting; 0 is at the world extent and each additional level cuts the viewport in half in each direction.

MAP_BASELAYERS

Default:

```

MAP_BASELAYERS = [{
  "source": {
    "ptype": "gxp_wmssource",
    "url": OGC_SERVER['default']['PUBLIC_LOCATION'] + "wms",
    "restUrl": "/gs/rest"
  }
}, {
  "source": {"ptype": "gxp_olsource"},
  "type": "OpenLayers.Layer",
  "args": ["No background"],
  "visibility": False,
  "fixed": True,
  "group": "background"
}, {
  "source": {"ptype": "gxp_osmsource"},
  "type": "OpenLayers.Layer.OSM",
  "name": "mapnik",
  "visibility": False,
  "fixed": True,
  "group": "background"
}, {
  "source": {"ptype": "gxp_mapquestsource"},
  "name": "osm",
  "group": "background",
  "visibility": True
}, {
  "source": {"ptype": "gxp_mapquestsource"},
  "name": "naip",
  "group": "background",
  "visibility": False
}, {
  "source": {"ptype": "gxp_bingsource"},
  "name": "AerialWithLabels",
  "fixed": True,
  "visibility": False,
  "group": "background"
}, {
  "source": {"ptype": "gxp_mapboxsource"},
}, {
  "source": {"ptype": "gxp_olsource"},
  "type": "OpenLayers.Layer.WMS",
  "group": "background",
  "visibility": False,
  "fixed": True,
  "args": [
    "bluemarble",
    "http://maps.opengeo.org/geowebcache/service/wms",
    {
      "layers": ["bluemarble"],
      "format": "image/png",
      "tilled": True,
      "tilesOrigin": [-20037508.34, -20037508.34]
    },
    {"buffer": 0}
  ]
}]

```

A list of dictionaries that specify the default map layers.

GEONODE_CLIENT_LAYER_PREVIEW_LIBRARY

Default: "geoext "

The library to use for display preview images of layers. The library choices are:

- "leaflet "
- "geoext "

OGC_SERVER

Default: {} (Empty dictionary)

A dictionary of OGC servers and their options. The main server should be listed in the 'default' key. If there is no 'default' key or if the OGC_SERVER setting does not exist Geonode will raise an Improperly Configured exception. Below is an example of the OGC_SERVER setting:

```
OGC_SERVER = {
    'default' : {
        'LOCATION' : 'http://localhost:8080/geoserver/',
        'USER' : 'admin',
        'PASSWORD' : 'geoserver',
    }
}
```

BACKEND

Default: "geonode.geoserver"

The OGC server backend to use. The backend choices are:

- 'geonode.geoserver'

BACKEND_WRITE_ENABLED

Default: True

Specifies whether the OGC server can be written to. If False, actions that modify data on the OGC server will not execute.

DATASTORE

Default: '' (Empty string)

An optional string that represents the name of a vector datastore that Geonode uploads are imported into. In order to support vector datastore imports there also needs to be an entry for the datastore in the DATABASES dictionary with the same name. Example:


```
OGC_SERVER = {
    'default' : {
        'LOCATION' : 'http://localhost:8080/geoserver/',
        'USER' : 'admin',
        'PASSWORD' : 'geoserver',
        'DATASTORE' : 'geonode_imports'
    }
}

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': 'development.db',
    },
    'geonode_imports' : {
        'ENGINE': 'django.contrib.gis.db.backends.postgis',
        'NAME': 'geonode_imports',
        'USER' : 'geonode_user',
        'PASSWORD' : 'a_password',
        'HOST' : 'localhost',
        'PORT' : '5432',
    }
}
```

GEOGIG_ENABLED

Default: False

A boolean that represents whether the OGC server supports GeoGig datastores.

GEONODE_SECURITY_ENABLED

Default: True

A boolean that represents whether GeoNode's security application is enabled.

LOCATION

Default: "http://localhost:8080/geoserver/"

A base URL from which GeoNode can construct OGC service URLs. If using GeoServer you can determine this by visiting the GeoServer administration home page without the /web/ at the end. For example, if your GeoServer administration app is at <http://example.com/geoserver/web/>, your server's location is <http://example.com/geoserver>.

MAPFISH_PRINT_ENABLED

Default: True

A boolean that represents whether the MapFish printing extension is enabled on the server.

PASSWORD

Default: 'geoserver'

The administrative password for the OGC server as a string.

PRINT_NG_ENABLED

Default: True

A boolean that represents whether printing of maps and layers is enabled.

PUBLIC_LOCATION

Default: "http://localhost:8080/geoserver/"

The URL used to in most public requests from Geonode. This settings allows a user to write to one OGC server (the LOCATION setting) and read from a separate server or the PUBLIC_LOCATION.

USER

Default: 'admin'

The administrative username for the OGC server as a string.

WMST_ENABLED

Default: False

Not implemented.

WPS_ENABLED

Default: False

Not implemented.

TIMEOUT

Default: 10

The maximum time, in seconds, to wait for the server to respond.

SITEURL

Default: 'http://localhost:8000/'

A base URL for use in creating absolute links to Django views and generating links in metadata.

Proxy settings

PROXY_ALLOWED_HOSTS

Default: `()` (Empty tuple)

A tuple of strings representing the host/domain names that GeoNode can proxy requests to. This is a security measure to prevent an attacker from using the GeoNode proxy to render malicious code or access internal sites.

Values in this tuple can be fully qualified names (e.g. `'www.geonode.org'`), in which case they will be matched against the request's Host header exactly (case-insensitive, not including port). A value beginning with a period can be used as a subdomain wildcard: `.geonode.org` will match `geonode.org`, `www.geonode.org`, and any other subdomain of `geonode.org`. A value of `'*'` will match anything and is not recommended for production deployments.

PROXY_URL

Default `/proxy/?url=`

The URL to a proxy that will be used when making client-side requests in GeoNode. By default, the internal GeoNode proxy is used but administrators may favor using their own, less restrictive proxies.

Search settings

DEFAULT_SEARCH_SIZE

Default: `10`

An integer that specifies the default search size when using `geonode.search` for querying data.

Security settings

AUTH_EXEMPT_URLS

Default: `()` (Empty tuple)

A tuple of URL patterns that the user can visit without being authenticated. This setting has no effect if `LOCKDOWN_GEONODE` is not `True`. For example, `AUTH_EXEMPT_URLS = ('/maps',)` will allow unauthenticated users to browse maps.

LOCKDOWN_GEONODE

Default: `False`

By default, the GeoNode application allows visitors to view most pages without being authenticated. If this is set to `True` users must be authenticated before accessing URL routes not included in `AUTH_EXEMPT_URLS`.

RESOURCE_PUBLISHING

Default: `True`

By default, the GeoNode application allows GeoNode staff members to publish/unpublish resources. By default resources are published when created. When this settings is set to True the staff members will be able to unpublish a resource (and eventually publish it back).

Social settings

SOCIAL_BUTTONS

Default: True

A boolean which specifies whether the social media icons and JavaScript should be rendered in GeoNode.

SOCIAL_ORIGINS

Default:

```
SOCIAL_ORIGINS = [{
    "label": "Email",
    "url": "mailto:?subject={name}&body={url}",
    "css_class": "email"
}, {
    "label": "Facebook",
    "url": "http://www.facebook.com/sharer.php?u={url}",
    "css_class": "fb"
}, {
    "label": "Twitter",
    "url": "https://twitter.com/share?url={url}",
    "css_class": "tw"
}, {
    "label": "Google +",
    "url": "https://plus.google.com/share?url={url}",
    "css_class": "gp"
}]
```

A list of dictionaries that is used to generate the social links displayed in the Share tab. For each origin, the name and URL format parameters are replaced by the actual values of the ResourceBase object (layer, map, document).

SHOW_PROFILE_EMAIL

Default: False

A boolean which specifies whether to display the email in user's profile.

CKAN_ORIGINS

Default:

```
CKAN_ORIGINS = [{
    "label": "Humanitarian Data Exchange (HDX)",
    "url": "https://data.hdx.rwlab.org/dataset/new?title={name}&notes={abstract}",
    "css_class": "hdx"
}]
```

A list of dictionaries that is used to generate the links to CKAN instances displayed in the Share tab. For each origin, the name and abstract format parameters are replaced by the actual values of the ResourceBase object (layer, map, document). This is not enabled by default. To enable, uncomment the following line: `SOCIAL_ORIGINS.extend(CKAN_ORIGINS)`.

TWITTER_CARD

Default:: `True`

A boolean that specifies whether Twitter cards are enabled.

TWITTER_SITE

Default:: `'@GeoNode'`

A string that specifies the site to for the twitter:site meta tag for Twitter Cards.

TWITTER_HASHTAGS

Default:: `['geonode']`

A list that specifies the hashtags to use when sharing a resource when clicking on a social link.

OPENGRAPH_ENABLED

Default:: `True`

A boolean that specifies whether Open Graph is enabled. Open Graph is used by Facebook and Slack.

Upload settings

GEOGIG_DATASTORE_NAME

Default: `None`

A string with the default GeoGig datastore name. This value is only used if no GeoGig datastore name is provided when data is uploaded but it must be populated if your deployment supports GeoGig.

UPLOADER

Default:

```
{
  'BACKEND' : 'geonode.rest',
  'OPTIONS' : {
    'TIME_ENABLED': False,
    'GEOGIG_ENABLED': False,
  }
}
```

A dictionary of Uploader settings and their values.

BACKEND

Default: `'geonode.rest'`

The uploader backend to use. The backend choices are:

- `'geonode.importer'`
- `'geonode.rest'`

The importer backend requires the GeoServer importer extension to be enabled and is required for uploading data into GeoGig datastores.

OPTIONS

Default:

```
'OPTIONS' : {  
  'TIME_ENABLED': False,  
  'GEOGIG_ENABLED': False,  
}
```

TIME_ENABLED

Default: `False`

A boolean that specifies whether the upload should allow the user to enable time support when uploading data.

GEOGIG_ENABLED

Default: `False`

A boolean that specifies whether the uploader should allow the user to upload data into a GeoGig datastore.

User Account settings

REGISTRATION_OPEN

Default: `False`

A boolean that specifies whether users can self-register for an account on your site.

THEME_ACCOUNT_CONTACT_EMAIL

Default: `'admin@example.com'`

This email address is added to the bottom of the password reset page in case users have trouble unlocking their account.

Download settings

DOWNLOAD_FORMATS_METADATA

Specifies which metadata formats are available for users to download.

Default:

```
DOWNLOAD_FORMATS_METADATA = [  
    'Atom', 'DIF', 'Dublin Core', 'ebRIM', 'FGDC', 'ISO',  
]
```

DOWNLOAD_FORMATS_VECTOR

Specifies which formats for vector data are available for users to download.

Default:

```
DOWNLOAD_FORMATS_VECTOR = [  
    'JPEG', 'PDF', 'PNG', 'Zipped Shapefile', 'GML 2.0', 'GML 3.1.1', 'CSV',  
    'Excel', 'GeoJSON', 'KML', 'View in Google Earth', 'Tiles',  
]
```

DOWNLOAD_FORMATS_RASTER

Specifies which formats for raster data are available for users to download.

Default:

```
DOWNLOAD_FORMATS_RASTER = [  
    'JPEG', 'PDF', 'PNG', 'Tiles',  
]
```

Contrib settings

EXIF_ENABLED

Default: False

A boolean that specifies whether the Exif contrib app is enabled. If enabled, metadata is generated from Exif tags when documents are uploaded.

NLP_ENABLED

Default: False

A boolean that specifies whether the NLP (Natural Language Processing) contrib app is enabled. If enabled, NLP (specifically MITIE) is used to infer additional metadata from uploaded documents to help fill metadata gaps.

NLP_LOCATION_THRESHOLD

Default: 1.0

A float that specifies the threshold for location matches.

NLP_LIBRARY_PATH

Default:: '/opt/MITIE/mitielib'

A string that specifies the location of the MITIE library

NLP_MODEL_PATH

Default:: '/opt/MITIE/MITIE-models/english/ner_model.dat'

A string that specifies the location of the NER (Named Entity Resolver). MITIE comes with English and Spanish NER models. Other models can be trained.

SLACK_ENABLED

Default: False

A boolean that specifies whether the Slack contrib app is enabled. If enabled, GeoNode will send messages to the slack channels specified in SLACK_WEBHOOK_URLS when a document is uploaded, metadata is updated, etc. Coverage of events is still incomplete.

SLACK_WEBHOOK_URLS

A list that specifies the URLs to post Slack messages to. Each URL is for a different channel. The default URL should be replaced when slack integration is enabled.

Default:

```
SLACK_WEBHOOK_URLS = [  
    "https://hooks.slack.com/services/T000/B000/XX"  
]
```

MONITORING_ENABLED

Default: False

Enable internal monitoring application (*geonode.contrib.monitoring*). If set to *True*, add following code to your local settings:

```
MONITORING_ENABLED = True  
# add following lines to your local settings to enable monitoring  
if MONITORING_ENABLED:  
    INSTALLED_APPS + ('geonode.contrib.monitoring',)  
    MIDDLEWARE_CLASSES + ('geonode.contrib.monitoring.middleware.MonitoringMiddleware'  
→',)
```

See *GeoNode Monitoring* for details.

GeoNode Django Apps

The user interface of a GeoNode site is built on top of the Django web framework. GeoNode includes a few “apps” (reusable Django modules) to support development of those user interfaces. While these apps have reasonable default configurations, for customized GeoNode sites you will probably want to adjust these apps for your specific needs.

`geonode.base` - GeoNode core functionality

Stores core functionality used throughout the GeoNode application.

Template Tags

`num_ratings` <object>

Returns the number of ratings an object has. Example usage:

```
{% load base_tags %}
{% num_ratings map as num_votes %}

<p>Map votes: {{num_votes}}.</p>
```

`categories`

Returns topic categories and the count of objects in each category.

`geonode.documents` - Document creation and management

Manages uploaded files that can be related to maps. Documents can be any type of file that is included in the `ALLOWED_DOCUMENTS_TYPES` setting.

`settings.py` Entries

`ALLOWED_DOCUMENT_TYPES` Default: `['doc', 'docx', 'xls', 'xlsx', 'pdf', 'zip', 'jpg', 'jpeg', 'tif', 'tiff', 'png', 'gif', 'txt']`

A list of acceptable file extensions that can be uploaded to the Documents app.

`MAX_DOCUMENT_SIZE` Default: 2

The maximum size (in megabytes) that an upload will be before it gets rejected.

`geonode.layers` - Layer creation and geospatial data management

This Django app provides support for managing and manipulating single geospatial datasets known as layers.

Models

- Attribute - Feature attributes for a layer managed by the GeoNode.
- Layer - A data layer managed by the GeoNode
- Style - A data layer’s style managed by the GeoNode

Views

- Creating, viewing, browsing, editing, and deleting layers and their metadata

Template Tags

featured_layers Returns the 7 newest layers.

layer_thumbnail <layer> Returns the layer's thumbnail.

manage.py Commands

importlayers `python manage.py importlayers`

Brings a data file or a directory full of data files into a GeoNode site. Layers are added to the Django database, the GeoServer configuration, and the GeoNetwork metadata index.

updatelayers `python manage.py updatelayers`

Scan GeoServer for data that has not been added to GeoNode.

geonode.maps - Map creation and geospatial data management

This Django app provides some support for managing and manipulating geospatial datasets. In particular, it provides tools for editing, viewing, and searching metadata for data layers, and for editing, viewing, and searching maps that aggregate data layers to display data about a particular topic.

Models

- Map - A collection of data layers composed in a particular order to form a map
- MapLayer - A model that maintains some map-specific information related to a layer, such as the z-indexing order.

Views

The maps app provides views for:

- Creating, viewing, browsing, editing, and deleting Maps

These operations require the use of GeoServer to manage map rendering, as well as GeoExt to provide interactive editing and previewing of maps and data layers.

There are also some URL mappings in the `geonode.maps.urls` module for easy inclusion in GeoNode sites.

settings.py Entries

OGC_SERVER Default: `{ }` (Empty dictionary)

A dictionary of OGC servers and their options. The main server should be listed in the 'default' key. If there is no 'default' key or if the `OGC_SERVER` setting does not exist GeoNode will raise an Improperly Configured exception. Below is an example of the `OGC_SERVER` setting:

```
OGC_SERVER = {
  'default' : {
    'LOCATION' : 'http://localhost:8080/geoserver/',
    'USER' : 'admin',
    'PASSWORD' : 'geoserver',
  }
}
```

BACKEND Default: "geonode.geoserver"

The OGC server backend to use. The backend choices are:

- 'geonode.geoserver'

BACKEND_WRITE_ENABLED Default: True

Specifies whether the OGC server can be written to. If False, actions that modify data on the OGC server will not execute.

LOCATION Default: "http://localhost:8080/geoserver/"

A base URL from which GeoNode can construct OGC service URLs. If using GeoServer you can determine this by visiting the GeoServer administration home page without the /web/ at the end. For example, if your GeoServer administration app is at <http://example.com/geoserver/web/>, your server's location is <http://example.com/geoserver>.

PUBLIC_LOCATION Default: "http://localhost:8080/geoserver/"

The URL used to in most public requests from GeoNode. This settings allows a user to write to one OGC server (the LOCATION setting) and read from a separate server or the PUBLIC_LOCATION.

USER Default: 'admin'

The administrative username for the OGC server as a string.

PASSWORD Default: 'geoserver'

The administrative password for the OGC server as a string.

MAPFISH_PRINT_ENABLED Default: True

A boolean that represents whether the MapFish printing extension is enabled on the server.

PRINT_NG_ENABLED Default: True

A boolean that represents whether printing of maps and layers is enabled.

GEONODE_SECURITY_ENABLED Default: True

A boolean that represents whether GeoNode's security application is enabled.

GEOGIG_ENABLED Default: False

A boolean that represents whether the OGC server supports GeoGig datastores.

WMST_ENABLED Default: False

Not implemented.

WPS_ENABLED Default: False

Not implemented.

DATASTORE Default: '' (Empty string)

An optional string that represents the name of a vector datastore that GeoNode uploads are imported into. In order to support vector datastore imports there also needs to be an entry for the datastore in the DATABASES dictionary with the same name. Example:

```
OGC_SERVER = {
    'default' : {
        'LOCATION' : 'http://localhost:8080/geoserver/',
        'USER' : 'admin',
        'PASSWORD' : 'geoserver',
        'DATASTORE': 'geonode_imports'
    }
}

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': 'development.db',
    },
    'geonode_imports' : {
        'ENGINE': 'django.contrib.gis.db.backends.postgis',
        'NAME': 'geonode_imports',
        'USER' : 'geonode_user',
        'PASSWORD' : 'a_password',
        'HOST' : 'localhost',
        'PORT' : '5432',
    }
}
```

GEOSERVER_CREDENTIALS Removed in GeoNode 2.0, this value is now specified in the OGC_SERVER settings.

GEOSERVER_BASE_URL Removed in GeoNode 2.0, this value is now specified in the OGC_SERVER settings.

CATALOGUE A dict with the following keys:

- **ENGINE**: The CSW backend (default is `geonode.catalogue.backends.pycsw_local`)
- **URL**: The FULLY QUALIFIED base URL to the CSW instance for this GeoNode
- **USERNAME**: login credentials (if required)
- **PASSWORD**: login credentials (if required)

pycsw is the default CSW enabled in GeoNode. pycsw configuration directives are managed in the PYCSW entry.

PYCSW A dict with pycsw's configuration. Of note are the sections `metadata:main` to set CSW server metadata and `metadata:inspire` to set INSPIRE options. Setting `metadata:inspire['enabled']` to `true` will enable INSPIRE support. Server level configurations can be overridden in the `server` section. See <http://pycsw.org/docs/configuration.html> for full pycsw configuration details.

SITEURL Default: `'http://localhost:8000/'`

A base URL for use in creating absolute links to Django views.

DEFAULT_MAP_BASE_LAYER The name of the background layer to include in newly created maps.

DEFAULT_MAP_CENTER Default: `(0, 0)`

A 2-tuple with the latitude/longitude coordinates of the center-point to use in newly created maps.

DEFAULT_MAP_ZOOM Default: `0`

The zoom-level to use in newly created maps. This works like the OpenLayers zoom level setting; 0 is at the world extent and each additional level cuts the viewport in half in each direction.

ASYNC_SIGNALS_BROKER_URL Default: ``memory://``

Sets broker url for async signals handling. Asynchronous signals are part of scalable architecture of GeoNode.

Internally, this will be parsed and used by *kombu* library. Default is *memory* transport, meaning all signals send with this method will be handled in-process, synchronously. When using *memory://* you do not need to set up external AMQP broker.

See *Asynchronous signals handling* for details.

geonode.proxy - Assist JavaScript applications in accessing remote servers

This Django app provides some HTTP proxies for accessing data from remote servers, to overcome restrictions imposed by the same-origin policy used by browsers. This helps the GeoExt applications in a GeoNode site to access various XML documents from OGC-compliant data services.

Views

geonode.proxy.views.proxy This view forwards requests without authentication to a URL provided in the request, similar to the proxy.cgi script provided by the OpenLayers project.

geonode.proxy.views.geoserver This view proxies requests to GeoServer. Instead of a URL-encoded URL parameter, the path component of the request is expected to be a path component for GeoServer. Requests to this URL require valid authentication against the Django site, and will use the default OGC_SERVER USER, PASSWORD and LOCATION settings as defined in the maps application.

geonode.search - Provides the GeoNode search functionality.

This Django app provides a fast search functionality to GeoNode.

Views

- **search_api**- Builds and executes a search query based on url parameters and returns matching results in requested format.

geonode.security - GeoNode granular Auth Backend

This app provides an authentication backend for use in assigning permissions to individual objects (maps and layers).

settings.py Entries

LOCKDOWN_GEONODE Default: `False`

By default, the GeoNode application allows visitors to view most pages without being authenticated. Set `LOCKDOWN_GEONODE = True` to require a user to be authenticated before viewing the application.

AUTH_EXEMPT_URLS Default: () (Empty tuple)

A tuple of URL patterns that the user can visit without being authenticated. This setting has no effect if LOCKDOWN_GEOCODE is not True. For example, AUTH_EXEMPT_URLS = ('/maps',) will allow unauthenticated users to browse maps.

Template Tags

geonode_media <media_name> Accesses entries in MEDIA_LOCATIONS without requiring the view to explicitly add it to the template context. Example usage:

```
{% include geonode_media %}
{% geonode_media "ext_base" %}
```

has_obj_perm <user> <obj> <permission> Checks whether the user has the specified permission on an object.

```
{% has_obj_perm user obj "app.view_thing" as can_view_thing %}
```

Django's error templates

GeoNode customizes some of Django's default error templates.

500.html

If no custom handler for 500 errors is set up in urls.py, Django will call django.views.defaults.server_error which expects a 500.html file in the root of the templates directory. In GeoNode, we have put a template that does not inherit from anything as 500.html and because most of Django's machinery is down when an INTERNAL ERROR (500 code) is encountered the use of template tags should be avoided.

3rd party apps

pinax.notifications

This application enables users to receive notifications from specific events within GeoNode. For user-specific configuration, see *Setting notification preferences*. For settings, see *User notifications settings*.

Make a GeoNode release

Making a GeoNode release requires a quite complex preparation of the environment while once everything is set up is a really easy and quick task. As said the complex part is the preparation of the environment since it involves, the generation of a password key to be uploaded to the Ubuntu servers and imported in launchpad.

If you have already prepared the environment then jump to the last paragraph.

Before start, make sure to have a [pypi](#) and a [launchpad](#) account.

Before doing the release, a GeoNode team member who can already make release has to add you as a launchpad GeoNode team member.

Creating and importing a gpg key

A GPG key is needed to push and publish the package. There is a complete guide on how to [create and import](#) a GPG key

Preparing the environment

Make sure to have a Ubuntu 12.04 machine. Install the following packages in addition to the python virtualenv tools:

```
$ sudo apt-get install git-core git-buildpackage debhelper devscripts
```

Get the GeoNode code (from master) in a virtualenv:

```
$ mkvirtualenv geonode
$ git clone https://github.com/GeoNode/geonode.git
$ cd geonode
```

Edit the .bashrc file and add the following lines (the key ID can be found in “your personal keys” tab:

```
export GPG_KEY_GEONODE="your_gpg_key_id"
export DEBEMAIL=yourmail@mail.com
export EDITOR=vim
export DEBFULLNAME="Your Full Name"
```

then close and:

```
$ source .bashrc
```

Type “env” to make sure all the variables are correctly exported

Set the correct git email:

```
$ git config --global user.email "yourmail@mail.com"
```

Register on Pypi with your Pypi credentials:

```
$ python setup.py register
```

Make the release

The followings are the only commands needed if the environment and the various registrations have already been done.

Make sure to have pulled the master to the desired commit. Edit the file geonode/__init__.py at line 21 and set the correct version.

Install GeoNode in the virtualenv (make sure to have the virtualenv activated and be in the geonode folder):

```
$ pip install -e geonode
```

Publish the package:

```
$ cd geonode
$ paver publish
```

The last command will:

- Tag the release and push it to GitHub
- Create the debian package and push it at ppa:geonode/testing in launchpad
- Create the .tar.gz sources and push them to Pypi
- Update the changelog and commit it to master

3.2.1.6.3 GeoNode on Production

Configuring GeoNode for Production

Advanced GeoServer Configuration

Running GeoNode under SSL

GeoSites: GeoNode Multi-Tenancy

Configuring GeoNode for Production

This page documents recommendations for configuring GeoNode in production environments. The steps mentioned in the first section are required to run GeoNode, the ones in the second section are either optional or ways to get more performance.

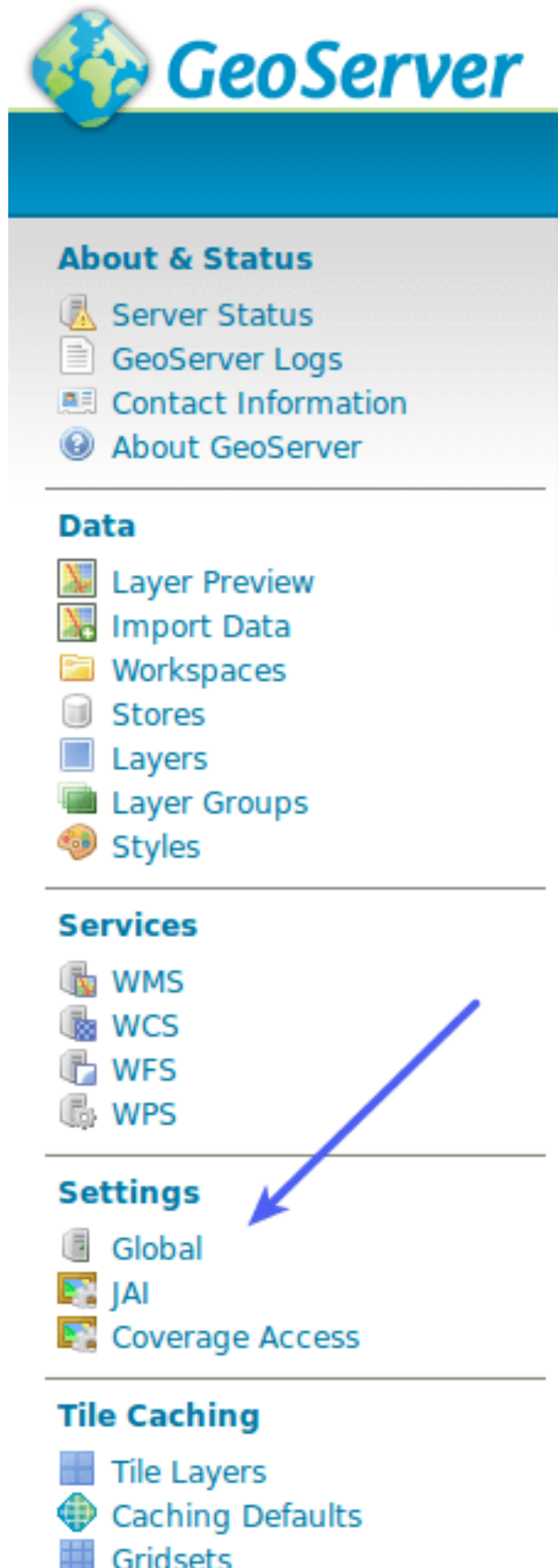
Note: This document makes numerous references to the **<host>** variable, please replace it with the IP Address of your GeoNode or the DNS entry.

For example: instead of `http://<host>/geoserver`, write down: `http://mygeonode.com/geoserver` or `http://127.0.0.1/geoserver`

Set the correct GeoServer Proxy URL value

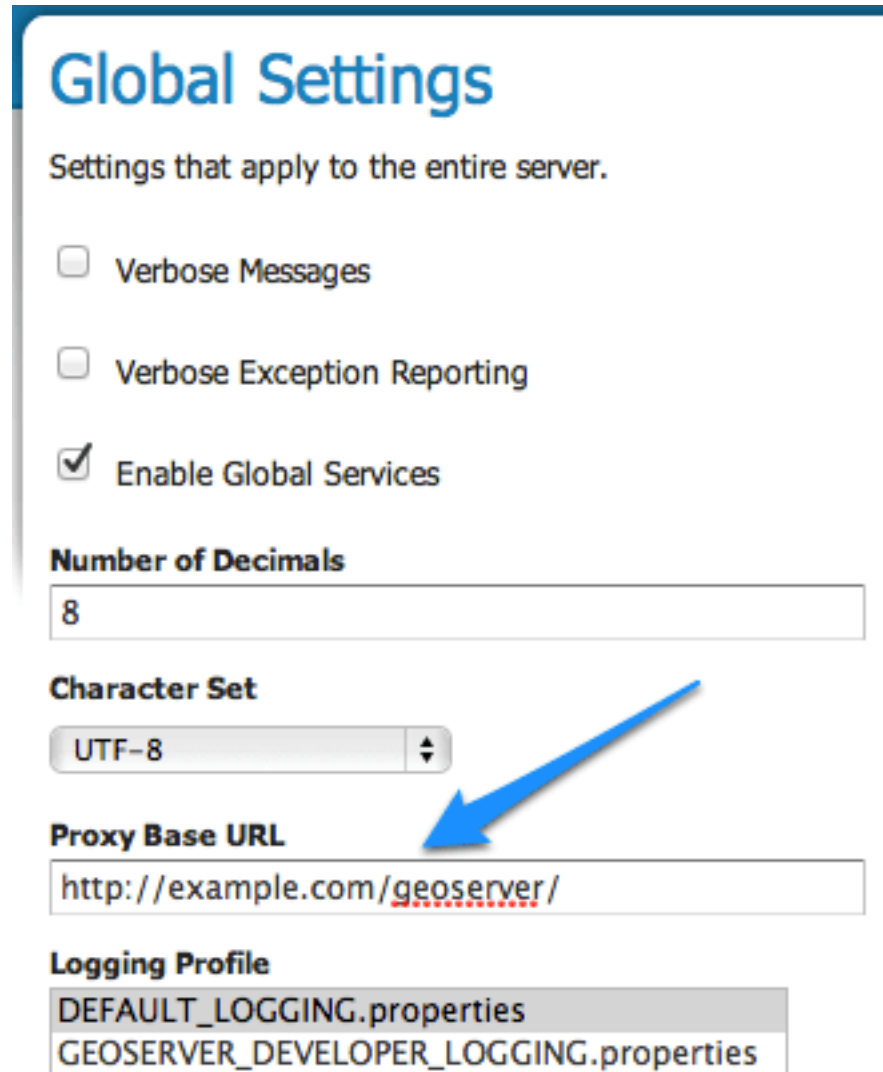
Navigate to `http://localhost/geoserver`, log in and click on the Global link in the Settings section.

Note: The Geoserver default username is **admin** with **geoserver** as the password. Change this ASAP and update `local_settings.py`.



Find the Proxy Base URL text field, put the complete address there:

```
http://<host>/geoserver/
```



Global Settings

Settings that apply to the entire server.

- ☐ Verbose Messages
- ☐ Verbose Exception Reporting
- ☒ Enable Global Services

Number of Decimals

Character Set

UTF-8

Proxy Base URL

Logging Profile

DEFAULT_LOGGING.properties
GEOSERVER_DEVELOPER_LOGGING.properties

Configure the Printing Module

This lives in the GeoServer Data directory `/usr/share/geoserver/data/printing/config.yaml`, add your server's IP address or domain name to the list of exceptions. Please refer to <http://docs.geoserver.org/2.4.x/en/user/datadirectory/index.html> for additional information on managing the GeoServer data directory:

```
hosts:
- !dnsMatch
  host: YOUR_IP_ADDRESS
  port: 80
```

Recommended Steps (optional)

Adding layers from Google, Bing and other providers

Bing

Get an API key from Microsoft at <http://www.bingmapsportal.com/> and place it in `local_settings.py`:

```
BING_API_KEY="zxcxzcXAWdsdfkjsdfWWsdfjpowxcxz"
```

Copy the `MAP_BASELAYERS` dictionary from `settings.py` into `local_settings.py` and add the following snippet:

```
{, {
"source": {
    "ptype": "gxp_bingsource",
    "apiKey": BING_API_KEY
},
"group": "background",
"name": "Aerial",
"visibility": False,
"fixed": True,
```

Note: The approach described above only works with versions of django-geoexplorer up to 4.0.34. If you have a newer version installed, you can revert to this older version using `pip install django-geoexplor==4.0.34`.

Google

Get an API key from Google at <https://developers.google.com/maps/faq?cs=1#using-google-maps-apis> and place it in `local_settings.py`, for example:

```
GOOGLE_API_KEY="zxcxzcXAWdqwdQWWQEDzxcxz"
```

Copy the `MAP_BASELAYERS` dictionary from `settings.py` into `local_settings.py` (or edit the previously copied snippet) and add the following snippet:

```
{, {
"source": {
    "ptype": "gxp_googlesource",
    "otherParams": "sensor=false&key={0}".format(GOOGLE_API_KEY)
},
"group": "background",
"name": "SATELLITE",
"visibility": False,
"fixed": True,
```

Sitemaps Configuration

GeoNode can automatically generate a sitemap suitable for submission to search engines which can help them to index your GeoNode site more efficiently and effectively.

In order to generate the sitemap properly, the sites domain name must be set within the sites framework. This requires that an superuser login to the admin interface and navigate to the sites module and change `example.com` to the actual

domain name (and port if applicable). The admin interface can be accessed at <http://<host>/admin/sites/site/>. Click on the `example.com` link, and change both the `Domain name` and `Display name` entries to match your system.

It is possible to ‘inform’ google of changes to your sitemap. This is accomplished using the `ping_google` management command. More information can be found here https://docs.djangoproject.com/en/dev/ref/contrib/sitemaps/#django.contrib.sitemaps.ping_google. It is recommended to put this call into a cron (scheduled) job to update google periodically.

Configuring User Registration

You can optionally configure GeoNode to allow new users to register through the web. New registrants will be sent an email inviting them to activate their account.

To allow new user registration:

1. Set up the email backend for Django (see [Django documentation](#)) and add the appropriate settings to `./src/GeoNodePy/geonode/local_settings.py`. For example:

```
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_HOST_USER = 'foo@gmail.com'
EMAIL_HOST_PASSWORD = 'bar'
EMAIL_PORT = 587
EMAIL_USE_TLS = True
```

2. In the same settings file set:

```
REGISTRATION_OPEN=True
```

3. With the Django application running, set the domain name of the service properly through the admin interface as specified above in the Sitemaps section. (This domain name is used in the account activation emails.).

5. Restart Apache:

```
$ sudo service apache2 restart
```

6. (Optional) Disable automatic approval of new users. Administrators would receive an email and need to manually approve new accounts. For this option to work, an email backed has to be defined in order to email the users with Staff status the notification to approve the new account:

```
ACCOUNT_APPROVAL_REQUIRED = True
```

To register as a new user, click the “Register” link in the GeoNode index header.

Additional Configuration

Some other things that require tweaking:

- Web-accessible uploads directory for user profile photos

Robot Exclusion File

GeoNode has several views that require considerable resources to properly respond - for example, the download links on layer detail pages require GeoServer to dynamically generate output in PDF, PNG, etc. format.

Crawlers for web search engines such as Google may cause problems by quickly following many such links in succession.

In order to request that “robots” do not make requests directly to GeoServer, you can ensure that requests to `/robots.txt` return a text file with the following content:

```
User-agent: *
Disallow: /geoserver/
```

This will only affect automated web agents; web browsers in particular are unaffected.

Memory Management

At the time of the GeoNode 1.0 release, the GeoNode manual recommended at least 4GB RAM for servers running GeoNode.

While 4GB *physical* RAM is sufficient, it is recommended that machines be configured with at least 6GB total *virtual* memory.

For example, a machine with 4GB physical RAM and 2GB swap space should be able to run GeoNode, but if you would like to run without a swapfile then you should configure the machine with at least 6GB RAM.

On Linux and Mac OS X hosts, you can check the available RAM with the following command:

```
$ free -m
              total            used             free      shared    buffers     cached
Mem:          6096             3863             2232          0           0           0
-/+ buffers/cache:          3863             2232
Swap:           0               0               0
```

The “total” column lists the available physical memory and swap space in megabytes; adding them together yields the amount of virtual memory available to the system.

In this example, there is no Swap space so that field is 0 and the available RAM on the system is 6096MB (6 GB).

Security Integration Optimization

GeoServer delegates authentication and authorization to GeoNode. The default configuration uses an HTTP endpoint in GeoNode to discover the current user and the layers that are accessible. For production, it is advisable to use a database-level connection.

Installing the Stored Procedure

The SQL for the stored procedure is distributed with the GeoServer web application archive and can be found at `WEB-INF/classes/org/geonode/security/geonode_authorize_layer.sql` in the webapps directory. It can be loaded using the `psql` command by following these steps (if not using tomcat6 or Ubuntu, locate the webapps directory for your configuration):

```
$ cd /var/lib/tomcat6/webapps
$ sudo su - postgres
$ psql -d YOUR_DATABASE < geoserver/WEB-INF/classes/org/geonode/security/geonode_
↪authorize_layer.sql
```

Configuring GeoServer to Use the Database Security Client

If a context configuration XML file does not already exist, create one for GeoServer. If using Tomcat 6 on Ubuntu, this file resides at `/etc/tomcat6/Catalina/localhost/geoserver.xml`. If creating a new file, the following XML should be added (replace ALLCAPS with your specific values):

```
<Context path="/geoserver"
  antiResourceLocking="false" >
  <Parameter name="org.geonode.security.databaseSecurityClient.url"
    value="jdbc:postgresql://localhost:5432/DATABASE?user=USER&password=PASSWORD" /
  </>
</Context>
```

If the file exists already, just add the *Parameter* element from above.

Verification of Database Security Client

To verify the settings change, look in the GeoServer logs for a line that notes: “using geonode database security client”. If any issues arise, check your connection configuration as specified in the context file above.

Configuring the Servlet Container

GeoServer is the most resource intensive component of GeoNode.

There are some general notes on setting up GeoServer for production environments in the [GeoServer manual](#).

However, the following are some GeoServer recommendations with GeoNode’s specific needs in mind.

JVM Options

The JRE used with GeoNode should be that distributed by Oracle.

Others such as OpenJDK may work but Oracle’s JRE is recommended for higher performance rendering.

Startup options should include the following:

```
-Xmx1024M -Xms1024M -XX:MaxPermSize=256M \
-XX:CompileCommand=exclude,net/sf/saxon/event/ReceivingContentHandler.startEvent
```

These can be specified using the `CATALINA_OPTS` variable in Tomcat’s `bin/catalina.sh` file, or the `JETTY_OPTS` in Jetty’s `bin/jetty.sh` file.

Constrain GeoServer Worker Threads

While the JVM provides memory management for most operations in Java applications, the memory used for rendering (in GeoServer’s case, responding to WMS GetMap requests) is not managed this way, so it is allocated in addition to the memory permitted by the JVM options above.

If GeoServer receives many concurrent requests, it may increase the memory usage significantly, so it is recommended to constrain the number of concurrent requests at the servlet container (ie, Jetty or Tomcat).

For Tomcat, you can edit `conf/server.xml`. By default, this file contains an entry defining a `ContextHandler`:

```
<Connector port="8080" protocol="HTTP/1.1"
  connectionTimeout="20000"
  redirectPort="8443"/>
```

Add a `maxThreads` attribute to limit the number of threads (concurrent requests) to 50 (the default in Tomcat is 200):

```
<Connector port="8080" protocol="HTTP/1.1"
  connectionTimeout="20000"
  redirectPort="8443" maxThreads="50"/>
```

Note: This configuration is possible in Jetty as well but not yet documented in this manual.

Native JAI and JAI ImageIO

Using the native-code implementation of JAI and JAI ImageIO speeds up GeoServer, thereby requiring less concurrency at the same level of throughput.

The GeoServer manual contains [platform-specific instructions](#) for configuring JAI and JAI ImageIO.

GeoServer Configuration

There are a few controls to be set in the GeoServer configuration itself as well.

On the JAI Settings page

On the WMS Service page

Printing with the Mapfish Print Service

The GeoNode map composer can “print” maps to PDF documents using the [Mapfish print service](#). The recommended way to run this service is by using the printing extension to GeoServer (if you are using the pre-built GeoNode package, this extension is already installed for you). However, the print service includes restrictions on the servers that can provide map tiles for printed maps. These restrictions have a fairly strict default, so you may want to loosen these constraints.

Adding servers by hostname

The MapFish printing module is configured through a [YAML](#) configuration file, usually named `print.yaml`. If you are using the GeoServer plugin to run the printing module, this configuration file can be found at `GEOSERVER_DATA_DIR/printing/config.yaml`. The default configuration should contain an entry like so:

```
hosts:
- !dnsMatch
  host: labs.metacarta.com
  port: 80
- !dnsMatch
```

(continues on next page)

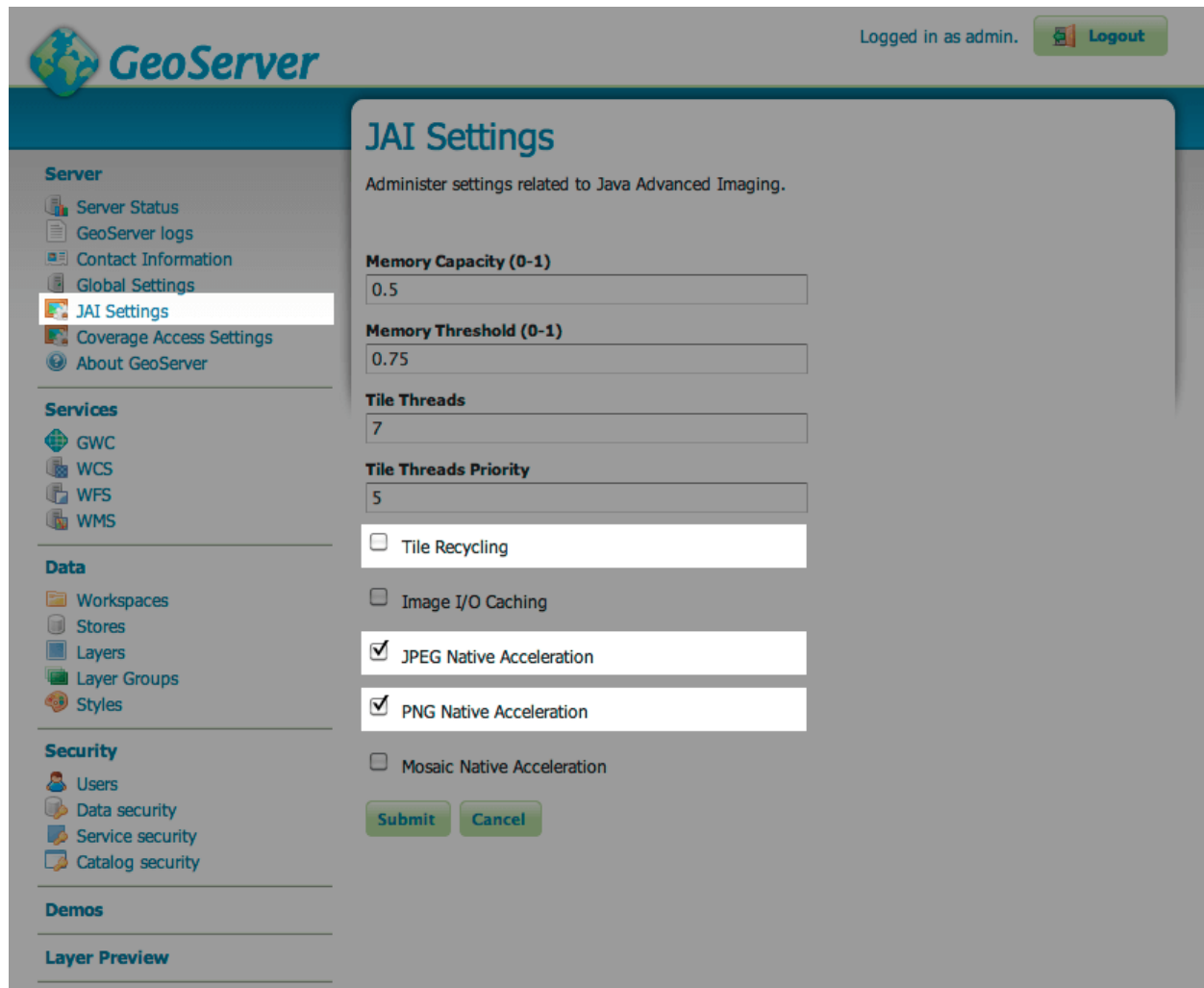


Fig. 275: There are two considerations for the JAI settings.

- Enable JPEG and PNG Native Acceleration to speed up the performance of WMS requests
- Disable Tile Recycling as this optimization is less relevant on recent JVM implementations and has some overhead itself.

GeoServer Logged in as admin. [Logout](#)

Server

- Server Status
- GeoServer logs
- Contact Information
- Global Settings
- JAI Settings
- Coverage Access Settings
- About GeoServer

Services

- GWC
- WCS
- WFS
- WMS**

Data

- Workspaces
- Stores
- Layers
- Layer Groups
- Styles

Security

- Users
- Data security
- Service security
- Catalog security

Demos

Layer Preview

Web Map Service

Manage map publishing

Service Metadata

☒ Enable WMS

☐ Strict CITE compliance

Maintainer

Online resource

Title

Abstract

Resource consumption limits

Max rendering memory (KB)

Max rendering time (s)

Max rendering errors (count)

[Submit](#) [Cancel](#)

Fig. 276: There is only one consideration for the Web Map Service page

- Don't set the "Resource Consumption Limits." This sounds a bit counter intuitive, but these limits are implemented in an inefficient way such that unless resource-intensive requests are common on your server it is more efficient to avoid the limits. A better implementation of this feature is available for GeoServer 2.1 and will be incorporated in GeoNode 1.1.

(continued from previous page)

```
host: terraservice.net
port: 80
```

You can add host/port entries to this list to allow other servers.

See also:

The [Mapfish documentation](#) on configuring the print module.

The [GeoServer documentation](#) on configuring the print module.

Advanced GeoServer Configuration

In this module we are going to cover some advanced configurations and :

Configuring GeoServer for robustness

In a production environment may be necessary to properly configure the WMS service in order to give a limit to resources associated with a request. The **Resource Limits** options allow the administrator to limit the resources consumed by each WMS GetMap request.

GeoServer provides a user interface for these options:

Resource consumption limits

Max rendering memory (KB)**Max rendering time (s)****Max rendering errors (count)**

Fig. 277: *Setting the Resource consumption limits*

The following table shows each option name, a description, and the minimum GeoServer version at which the option is available (old versions will just ignore it if set).

Option	Description	Version
Max rendering memory	Sets the maximum amount of memory, in kilobytes, a single GetMap request is allowed to use. Each output format will make a best effort attempt to respect the maximum using the highest consuming portion of the request processing as a reference. For example, the PNG output format will take into consideration the memory used to prepare the image rendering surface in memory, usually proportional to the image size multiplied by the number of bytes per pixel	1.7.5
Max rendering time	Sets the maximum amount of time, in seconds, GeoServer will use to process the request. This time limits the “blind processing” portion of the request serving, that is, the part in which GeoServer is computing the results before writing them out to the client. The portion that is writing results back to the client is not under the control of this parameter, since this time is also controlled by how fast the network between the server and the client is. So, for example, in the case of PNG/JPEG image generation, this option will control the pure rendering time, but not the time used to write the results back.	1.7.5
Max rendering errors	Sets the maximum amount of rendering errors tolerated by a GetMap. Usually GetMap skips over faulty features, reprojection errors and the like in an attempt to serve the results anyways. This makes for a best effort rendering, but also makes it harder to spot issues, and consumes CPU cycles as each error is handled and logged	1.7.5

Out of the box GeoServer uses 65MB, 60 seconds and 1000 errors respectively. All limits can be disabled by setting their value to 0.

Once any of the set limits is exceeded, the GetMap operation will stop and a `ServiceException` will be returned to the client.

It is suggested that the administrator sets all of the above limits taking into consideration peak conditions. For example, while a GetMap request under normal circumstance may take less than a second, under high load it is acceptable for it to take longer, but usually, it's not sane that a request goes on for 30 minutes straight. The following table shows some example values for the configuration options above, with explanations of how each is computed:

Option	Value	Rationale
maxRequestMemory	65000	65MB are sufficient to render a 4078x4078 image at 4 bytes per pixel (full color and transparency), or a 8x8 meta-tile if you are using GeoWebCache or TileCache. Mind the rendering process will use an extra in memory buffer for each subsequent FeatureTypeStyle in your SLD, so this will also limit the size of the image. For example, if the SLD contains two FeatureTypeStyle element in order to draw cased lines for an highway the maximum image size will be limited to 2884x2884 (the memory goes like the square of the image size, so halving the memory does not halve the image size)
maxRenderingTime	60	A request that processes for one minute straight is probably drawing a lot of features independent of the current load. It might be the result of a client making a GetMap against a big layer using a custom style that does not have the proper scale dependencies
maxRenderingErrors	1000	Encountering 1000 errors is probably the result of a request that is trying to reproject a big data set into a projection that is not suited to area it covers, resulting in many reprojection failures.

Advanced Production GeoServer configuration

Most of the GeoServer downloads are geared towards quickly showing off the capabilities, with an array of demos, sample layers, and an embedded servlet container. If you are using GeoServer in a production environment, there are a few things we'd like to recommend. In this section the task is to configure your system to use it in production.

Note: Before you start, ensure that the *Web Administrator Interface - Server* section has been completed.

Configuring your container for production

Note: Most open source Java web containers, such as Tomcat, ship with development mode configurations that allow for quick startup but don't deliver the best performance.

Make sure that in the 'setenv.sh', or 'setenv.bat' on Windows machines, file exists the following configuration to set up the Java virtual machine options in your container. Open the 'setenv.sh/.bat' file located in '<TRAINING_ROOT>' directory and look at the options:

```
1 set ROOT=%~dp0
2
3 REM Variable used in documentation
4 set TRAINING_ROOT=%ROOT%
5 REM clear the user PATH
6 set PATH=C:\Windows\System32;C:\Windows;C:\Windows\SysWow64;
7
8 REM Setting Java
9 set JAVA_HOME=%ROOT%\jdk
10 REM set JAVA_HOME=C:\Java\jdk
11 set PATH=%JAVA_HOME%\bin;%PATH%
12
13 REM GeoServer and GeoWebCache Options
14 set GEOSERVER_DATA_DIR=%ROOT%\geoserver_data\
15 REM set GEOSERVER_DATA_DIR=%ROOT%\tomcat-6.0.36\instances\instance1\webapps\geoserver\data
16 set GEOSERVER_LOG_LOCATION=%CATALINA_HOME%\logs\%INSTANCE_NAME%.log
17 set GEOWEBCACHE_CACHE_DIR=%ROOT%\data\gwc
18
19 REM Tomcat Options for the JVM
20 set JAVA_OPTS=%JAVA_OPTS% -Xms512m -Xmx512m -XX:MaxPermSize=128m -XX:PermSize=128m
21 set JAVA_OPTS=%JAVA_OPTS% -XX:+UseConcMarkSweepGC -XX:+UseParNewGC -XX:ParallelGCThreads=4
22 set JAVA_OPTS=%JAVA_OPTS% -DGEOSERVER_DATA_DIR=%GEOSERVER_DATA_DIR%
23 set JAVA_OPTS=%JAVA_OPTS% -DGEOSERVER_LOG_LOCATION=%GEOSERVER_LOG_LOCATION%
24 set JAVA_OPTS=%JAVA_OPTS% -DGEOWEBCACHE_CACHE_DIR=%GEOWEBCACHE_CACHE_DIR%
25 set JAVA_OPTS=%JAVA_OPTS% -Djavax.servlet.request.encoding=UTF-8
26 set JAVA_OPTS=%JAVA_OPTS% -Djavax.servlet.response.encoding=UTF-8 -Dfile.encoding=UTF-8
27 set CATALINA_BASE=%CATALINA_HOME%\instances\%INSTANCE_NAME%
```

Fig. 278: Setting the JAVA_OPTS for Tomcat container

- **-server:** Not present among the training options, this option enables the server JVM, which JIT compiles byte-code much earlier, and with stronger optimizations. Startup and first calls will be slower due to JIT compilation taking more time, but subsequent ones will be faster (to give you some numbers, on the same machine a vanilla VM returns GML at 7MB/s speed, a -server one runs at 10MB/s). This option is required only if the JVM does not already get into server mode, which happens on a server operating system (Linux, Windows server) with at least 2 cores and 2 GB of memory.

Note: This parameter is necessary only for Windows environments of class workstation

- **-Xms512m -Xmx512M:** give your server memory. By default JVM will use only 64MB of heap. If you're serving just vector data, you'll be full streaming, so having much memory won't help a lot, but if you're serving

coverages JAI will use a cache to avoid hitting the disk often. In this case, give GeoServer at least 256MB of memory, or more if you have plenty of RAM, and go configure the JAI title cache size in the GeoServer configuration panel so that it uses 75% of the heap (0.75). If you have plenty of memory it is suggested to set `-Xms` to the same value as `-Xmx`, this will make heap management more stable during heavy load serving. Generally speaking, don't allocate more than 2GB for the GeoServer heap.

- **-XX:MaxPermSize=128m** (or more): the permanent generation is the heap portion where the class bytecode is stored. GeoServer uses lots of classes, and it may exhaust that space quickly leading to out of memory errors. If you're deploying GeoServer along with other applications in the same container or if you need to deploy multiple GeoServer instances inside the same container (e.g., different instances for different customers or similar needs) you better raise up the `MaxPermSize` to 128m or more.

Warning: In order to obtain best performance, install the native **JAI** version in your JDK. In particular, installing the native JAI is important for all raster processing, which is used heavily in both WMS and WCS to rescale, cut and reproject rasters. Installing the native JAI is also important for all raster reading and writing, which affects both WMS and WCS. Finally, native JAI is very useful even if there is no raster data involved, as WMS output encoding requires writing PNG/GIF/JPEG images, which are themselves rasters. For more information how to install JAI and ImageIO see the Installing the native JAI and ImageIO section

Setting up logging for production

Note: Logging may visibly affect the performance of your server. High logging levels are often necessary to track down issues, but by default you should run with low ones (and you can switch the logging levels at runtime, so don't worry about having to stop the server to gather more information). You can change the logging level by going to the GeoServer configuration panel, Server section.

- 1) Go to <http://localhost:8083/geoserver> and click on the 'Global' link in the 'Settings' section.
- 2) Select 'PRODUCTION_LOGGING.properties' in *Logging Profile* and click submit.

Choosing a service strategy

Note: A service strategy is the way we serve the output to the client. Basically, you have to choose between being absolutely sure of reporting errors with the proper OGC codes and serving output quickly.

You can configure the service strategy modifying the `web.xml` file located in '`<TOMCAT_HOME>/instances/instance1/webapps/geoserver/WEB-INF`' directory of your GeoServer install:

- 1) Set the 'serviceStrategy' param-name with 'SPEED'.

All the possible strategies are:

- **SPEED:** serve outputs right away. The fastest strategy, make it unlikely to be able to report proper OGC errors in WFS though (they are reported only if the error occurs before the GML encoding starts).
- **BUFFER:** stores the whole result in memory, and then serves it after the output is complete. This ensures proper OGC error reporting, but delays the response quite a bit and will exhaust memory if the response is big.
- **FILE:** same as buffer, but uses a file storage for buffering. Slower than BUFFER, ensures there won't be memory issues.

Global Settings

Settings that apply to the entire server.

☐ Verbose Messages

☐ Verbose Exception Reporting

Number of Decimals

Character Set

Proxy Base URL

Logging Profile

DEFAULT_LOGGING.properties

VERBOSE_LOGGING.properties

PRODUCTION_LOGGING.properties

GEOTOOLS_DEVELOPER_LOGGING.properties

GEOSERVER_DEVELOPER_LOGGING.properties

☒ Log to StdOut

Log Location

Fig. 279: Set up logging for production

- PARTIAL-BUFFER2: a balance between the two, tries to buffer in memory a few kilobytes of response, then behaves like SPEED.

Configuring all data and metadata to your instance

Note: It may be tempting to just skip some of the configuration steps, leave in the same keywords and abstract as the sample. Please do not, as this will only confuse potential users. They will have a list of GeoServers called ‘My GeoServer’.

- Completely fill out the WFS and WMS Contents sections.
- Put in your own URI (such as the name of your website) for the Namespace (Data -> Workspace) and remove the defaults.
- Make sure your datastores all use your URI.
- Remove the sample layers (states, spearfish, New York roads and the like, the easiest way to is go and remove the demo workspaces, everything contained in them will be removed as a result)

Change the administrator password

GeoServer ships by default with “admin” and “geoserver” as the default administrator user name and password. Before putting the GeoServer on-line it is imperative to change at least the administrator password.

Making use of an external Data Directory to store your configurations

Note: The configuration data resides within the GEOSERVER_DATA_DIR. To increase the portability of their data and to facilitate updates GeoServer, you should place this directory outside of GeoServer editing the web.xml file with the path that you prefer

See the ‘GEOSERVER_DATA_DIR’ context parameter in ‘<TRAINING_ROOT>/tomcat-6.0.36/instances/instance1/webapps/geoserver/WEB-INF’:

```
<context-param>
  <param-name>GEOSERVER_DATA_DIR</param-name>
  <param-value>$GEOSERVER_DATA_DIR</param-value>
</context-param>
```

Note: The external data dir can be also configured through the environment variables on the ‘setenv.sh/.bat’ file.

Using a Spatial Database

We make shapefiles available as a datastore, as they are such a common format. But if you are running GeoServer in a production environment and if you need to manage the spatial indexes, transactions or if you have specific requirements involving the use of a database, setting up a spatial database and converting your shapefiles is highly recommended. If you’re doing transactions against GeoServer this is essential. Even though we have a very nice transaction framework, doubling up with the native transaction support of relational databases ensures your data integrity. Most all the major

spatial DBs provide support to easily turn shapefiles into their native format. We recommend PostGIS, open source extensions to the PostgreSQL DB, most of our testing has been performed against it. Oracle, DB2, SQL Server and ArcSDE are also well supported. At the moment we don't recommend MySQL, as it has trouble with rollbacks on geometry tables, and lacks advanced spatial functionality, but it is an option.

Setting security

GeoServer by default includes WFS-T, which lets users modify your backend database. If you don't want that to happen, you can turn off transactions in the web admin tool, Service Panel -> WFS and set Service Level to Basic. If you'd like some users to be able to modify it, but not all, you'll have to set up data access level security. For extra security when operating in read only mode, make sure that the connection to the datastore that is open to all is with a user who has read only permissions. That will make it so it's completely impossible to do a SQL injection (though GeoServer is generally designed well enough that it's not vulnerable).

Dealing with a locked down environment

GeoServer code, and the libraries it uses (Geotools, JAI) are not designed to be run in a security locked down environment. They need free access to environment variables, temp directory, user preferences and the like. In operating systems like Ubuntu the default Tomcat is locked down so that most of the above is not authorized. So far, the only way to run GeoServer in that environment is to grant all permissions to it.

Caching

Server-side caching of WMS tiles is the best way to get performance. Essentially how the caching works is the server will recognize a request and quickly return a pre-rendered result. This is how you can optimize for tile-based WMS clients and it works the best for them. There are several ways to set up caching for GeoServer like GeoWebCache.

Advanced Coordinate Reference System Handling

This section describes how coordinate reference systems (CRS) are handled in GeoServer, as well as what can be done to extend GeoServer's CRS handling abilities.

Coordinate Reference System Configuration

When adding data, GeoServer tries to inspect data headers looking for an EPSG code: if the data has a CRS with an explicit EPSG code and the full CRS definition behind the code matches the one in GeoServer, the CRS will be already set for the data. If the data has a CRS but no EPSG code, you'll have to manually guess the EPSG code. Browsing to <http://www.spatialreference.org> might be a good option to find the exact EPSG code for your data.

If an EPSG code cannot be found, then either the data has no CRS or it is unknown to GeoServer. In this case, there are a few options:

- Force the declared CRS, ignoring the native one. This is the best solution if the native CRS is known to be wrong.
- Reproject from the native to the declared CRS. This is the best solution if the native CRS is correct, but cannot be matched to an EPSG number. An alternative is to add a custom EPSG code that matches exactly the native SRS.

If your data has no native CRS information, the only option is to specify/force an EPSG code.

Custom CRS Definitions

Add a custom CRS

This example shows how to add a custom projection in GeoServer.

1. The projection parameters need to be provided as a WKT (well known text) definition. The code sample below is just an example:

```
PROJCS["NAD83 / Austin",
  GEOGCS["NAD83",
    DATUM["North_American_Datum_1983",
      SPHEROID["GRS 1980", 6378137.0, 298.257222101],
      TOWGS84[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
      PRIMEM["Greenwich", 0.0],
      UNIT["degree", 0.017453292519943295],
      AXIS["Lon", EAST],
      AXIS["Lat", NORTH]],
    PROJECTION["Lambert_Conformal_Conic_2SP"],
    PARAMETER["central_meridian", -100.333333333333],
    PARAMETER["latitude_of_origin", 29.666666666667],
    PARAMETER["standard_parallel_1", 31.883333333333297],
    PARAMETER["false_easting", 2296583.333333],
    PARAMETER["false_northing", 9842500.0],
    PARAMETER["standard_parallel_2", 30.116666666667],
    UNIT["m", 1.0],
    AXIS["x", EAST],
    AXIS["y", NORTH],
    AUTHORITY["EPSG", "100002"]]
```

Note: This code sample has been formatted for readability. The information will need to be provided on a single line instead, or with backslash characters at the end of every line (except the last one).

2. Go into the `user_projections` directory inside your data directory, and open the `epsg.properties` file. If this file doesn't exist, you can create it.
3. Insert the code WKT for the projection at the end of the file (on a single line or with backslash characters):

```
100002=PROJCS["NAD83 / Austin", \
  GEOGCS["NAD83", \
    DATUM["North_American_Datum_1983", \
      SPHEROID["GRS 1980", 6378137.0, 298.257222101], \
      TOWGS84[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], \
      PRIMEM["Greenwich", 0.0], \
      UNIT["degree", 0.017453292519943295], \
      AXIS["Lon", EAST], \
      AXIS["Lat", NORTH]], \
    PROJECTION["Lambert_Conformal_Conic_2SP"], \
    PARAMETER["central_meridian", -100.333333333333], \
    PARAMETER["latitude_of_origin", 29.666666666667], \
    PARAMETER["standard_parallel_1", 31.883333333333297], \
    PARAMETER["false_easting", 2296583.333333], \
    PARAMETER["false_northing", 9842500.0], \
    PARAMETER["standard_parallel_2", 30.116666666667], \
    UNIT["m", 1.0], \
    AXIS["x", EAST], \
```

(continues on next page)

(continued from previous page)

```
AXIS["y", NORTH], \
AUTHORITY["EPSG", "100002"]]
```

Note: Note the number that precedes the WKT. This will determine the EPSG code. So in this example, the EPSG code is 100002.

1. Save the file.
2. Restart GeoServer.
3. Verify that the CRS has been properly parsed by navigating to the *srs_list* page in the *web_admin*.
4. If the projection wasn't listed, examine the logs for any errors.

Override an official EPSG code

In some situations it is necessary to override an official EPSG code with a custom definition. A common case is the need to change the TOWGS84 parameters in order to get better reprojection accuracy in specific areas.

The GeoServer referencing subsystem checks the existence of another property file, `epsg_overrides.properties`, whose format is the same as `epsg.properties`. Any definition contained in `epsg_overrides.properties` will **override** the EPSG code, while definitions stored in `epsg.properties` can only **add** to the database.

Special care must be taken when overriding the Datum parameters, in particular the **TOWGS84** parameters. To make sure the override parameters are actually used the code of the Datum must be removed, otherwise the referencing subsystem will keep on reading the official database in search of the best Datum shift method (grid, 7 or 5 parameters transformation, plain affine transform).

For example, if you need to override the official **TOWGS84** parameters of EPSG:3003 to better match the peninsular area of Italy:

```
PROJCS["Monte Mario / Italy zone 1",
GEOGCS["Monte Mario",
  DATUM["Monte Mario",
    SPHEROID["International 1924", 6378388.0, 297.0, AUTHORITY["EPSG","7022"]],
    TOWGS84[-50.2, -50.4, 84.8, -0.69, -2.012, 0.459, -5.791915759418465],
    AUTHORITY["EPSG","6265"]],
  PRIMEM["Greenwich", 0.0, AUTHORITY["EPSG","8901"]],
  UNIT["degree", 0.017453292519943295],
  AXIS["Geodetic longitude", EAST],
  AXIS["Geodetic latitude", NORTH],
  AUTHORITY["EPSG","4265"]],
PROJECTION["Transverse Mercator", AUTHORITY["EPSG","9807"]],
PARAMETER["central_meridian", 9.0],
PARAMETER["latitude_of_origin", 0.0],
PARAMETER["scale_factor", 0.9996],
PARAMETER["false_easting", 1500000.0],
PARAMETER["false_northing", 0.0],
UNIT["m", 1.0],
AXIS["Easting", EAST],
AXIS["Northing", NORTH],
AUTHORITY["EPSG","3003"]]
```

You should write the following (in a single line, here it's reported formatted over multiple lines for readability):

```

3003 =
PROJCS["Monte Mario / Italy zone 1",
GEOGCS["Monte Mario",
    DATUM["Monte Mario",
        SPHEROID["International 1924", 6378388.0, 297.0, AUTHORITY["EPSG","7022"]],
        TOWGS84[-104.1, -49.1, -9.9, 0.971, -2.917, 0.714, -11.68],
        AUTHORITY["EPSG","6265"]],
        PRIMEM["Greenwich", 0.0, AUTHORITY["EPSG","8901"]],
        UNIT["degree", 0.017453292519943295],
        AXIS["Geodetic longitude", EAST],
        AXIS["Geodetic latitude", NORTH]],
    PROJECTION["Transverse_Mercator"],
    PARAMETER["central_meridian", 9.0],
    PARAMETER["latitude_of_origin", 0.0],
    PARAMETER["scale_factor", 0.9996],
    PARAMETER["false_easting", 1500000.0],
    PARAMETER["false_northing", 0.0],
    UNIT["m", 1.0],
    AXIS["Easting", EAST],
    AXIS["Northing", NORTH],
    AUTHORITY["EPSG","3003"]]

```

The definition has been changed in two places: the **TOWGS84** parameters, and the Datum code, `AUTHORITY["EPSG","4265"]`, have been removed.

Advanced Database Connection Pooling Configuration

Database connections are valuable resources and as such shall be managed with care:

- they are heavy to create and maintain for the database server itself since they are usually child processes of the DBMS server process
- being processes that means that creating a connection is not a zero-cost process therefore we should avoid creating connections as we need to connect to a DB but we should tend to create them in advance in order to minimize the impact of the time needed to create them on the time needed to serve a request.
- as a consequence of the fact that a connection require non negligible resources on the server DBMS DBAs tend to
 - limit the number of connections globally available (e.g. PostgreSQL by default has a limit set to 100)
 - limit the lifetime of connections created in order to discourage clients from retaining connections for a really long time

The purpose served by a connection pool is to maintain connections to an underlying database between requests. The benefit is that connection set-up only need to occur once on the first request while subsequent requests use existing connections and achieve a performance benefit as a result.

Ok, now let's go into GeoServer specifics. In most GeoServer datastores you have the possibility to use the JNDI¹ or the standard store which basically means you can have GeoServer manage the connection pool for you or you can configure it externally (from within the Application Server of choice) and then have GeoServer lean onto it to get connections. Baseline is, one way or the other you'll always end-up using a connection pool in GeoServer.

¹ https://en.wikipedia.org/wiki/Java_Naming_and_Directory_Interface

GeoServer Internal Connection Pool Parameters

Whenever a data store backed by a database is added to GeoServer an internal connection pool, for which relies on Apache Commons DBCP², is created by default. This connection pool is configurable, however let me say this upfront, the number of pool configuration parameters that we expose is a subset of the possible ones, although the most interesting are there. Namely there are a few that you might want to customize. Here below you can find some more details on the available connection parameters.

² <http://commons.apache.org/proper/commons-dbcp/>

max connections	The maximum number of connections the pool can hold. When the maximum number of connections is exceeded, additional requests that require a database connection will be halted until a connection from the pool becomes available and eventually times out if that's not possible within the time specified in the connection time-out. <i>The maximum number of connections limits the number of concurrent requests that can be made against the database.</i>
min connections	<p>The minimum number of connections the pool will hold. This number of connections is held even when there are no active connections.</p> <p>-1- If it is very far from the max connections this might limit the ability of the GeoServer to respond quickly to unexpected or random heavy load situations due to the fact that it takes a non negligible time to create a new connections. However this set up is very good when the DBMS is quite loaded since it tends to use as less connections as possible at all times.</p> <p>-2- If it is very close to the max connections value the GeoServer will be very fast to respond to random load situation. However in this case the GeoServer would put a big burden on DBMS shoulders as the the pool will try to hold all needed connections at all times.</p>
validate connections	Flag indicating whether connections from the pool should be validated before they are used. A connection in the pool can become invalid for a number of reasons including network breakdown, database server timeout, etc.. The benefit of setting this flag is that an invalid connection will never be used which can prevent client errors. The downside of setting the flag is that a small performance penalty is paid in order to validate connections when the connection is borrowed from the pool since the validation is done by sending small query to the server. However the cost of this query is usually small, as an instance on PostGis the validation query is <i>Select 1</i> .
fetch size	The number of records read from the database in each network exchange. If set too low (<50) network latency will affect negatively performance, if set too high it might consume a significant portion of GeoServer memory and push it towards an <i>Out Of Memory Error</i> . Defaults to 1000, it might be beneficial to push it to a higher number if the typical database query reads much more data than this, and there is enough heap memory to hold the results
connection time-out	Time, in seconds , the connection pool will wait before giving up its attempt to get a new connection from the database. Defaults to 20 seconds. This timeout kicks in during heavy load conditions when the number of requests needing a connection to a certain DB outnumber greatly the number of available connections in the pool, therefore some requests might get error messages due to the timeouts while acquiring a connection. This condition is not per se problematic since usually a request does not use a DB connection for its entire lifecycle hence we do not need 100 connections at hand to respond to 100 requests; however we should strive to limit this condition since it would queue threads on the connection pool after they might have allocated memory (e.g. for rendering). We will get back to this later on.
max open prepared statements	Maximum number of prepared statements kept open and cached for each connection in the pool.
max wait	number of seconds the connection pool will wait before timing out attempting to get a new connection (default, 20 seconds)
validate connection	It forces GeoServer to check that the connections borrowed from the pool are valid (i.e. not closed on the DBMS side) before using them.
Test while idle	Periodically test if the connections are still valid also while idle in the pool. Sometimes performing a test query using an idle connection can make the datastore unavailable for a while. Often the cause of this troublesome behaviour is related to a network firewall placed between Geoserver and the Database that force the closing of the underlying idle TCP connections.
3.2. Tutorials	
Evictor run periodicity	Number of seconds between idle object evictor runs.
Max connection idle time	Number of seconds a connection needs to stay idle before the evictor starts to consider closing it.

Prepared statements consideration

Prepared statements are used by databases to avoid re-planning the data access every time, the plan is done only once up-front, and as long as the statement is cached, the plan does not need to be re-computed.

In business applications fetching a small amount of data at a time this is beneficial for performance, however, in spatial ones, where we typically fetch thousands of rows, the benefit is limited, and sometimes, turns into a performance problem. This is the case with PostGIS, that is able to tune the access plan by inspecting the requested bounding box, and deciding if a sequential scan is preferable (the BBOX really accesses most of the data) or using the spatial index is best instead. So, as a rule of thumb, when working with PostGIS, it's better not to enable prepared statements.

With other databases there are no choices, Oracle currently works only with prepared statements, SQL server only without them (this is often related to implementation limitations than database specific issues).

There is an upside of using prepared statement though: no SQL injection attacks are possible when using them. GeoServer code tries hard to avoid this kind of attack when working without prepared statements, but enabling them makes the attack via filter parameters basically impossible.

Final Thoughts

To summarize, when creating standard datastores for serving vector data from DBMS in GeoServer you need to remember that internally a connection pool will be created. This approach is the simplest to implement but might lead to an inefficient distribution of the connections between different stores in the following cases:

- if we tend to separate tables into different schemas this will lead to the need for creating multiple stores to serve them out since GeoServer works best if the “schema” parameter is specified, this leading to the creation of (mostly unnecessary) connection pools
- if we want to create stores in different workspaces connecting to the same database this again will lead to unnecessary duplication of connection pools in different store leading to inefficient usage of connections

Long story short the fact that the pool is internal with respect to the stores may lead to inefficient usage of connections to the underlying DBMS since they will be split between multiple stores limiting the scalability of each of them: in fact having 100 connections shared between N normal datastore will impose limits to the maximum number that each can use, otherwise if we managed to keep the connections into a single pool shared, in turn, with the various datastore we would achieve a much more efficient sharing between the store as, as an instance, a single store under high load could scale to use all the connections available.

Configuration of a JNDI connection pool with Tomcat

Many datastores in GeoServer provide the option of exploiting the Java Naming and Directory Interface or **JNDI**, for managing the connections pools. JNDI allows for components in a Java system to look up other objects and data by a predefined name. A common use of JNDI is to set-up a connection pool in order to improve the database resource management.

In order to set-up a connection pool, Tomcat needs to be provided with a JDBC driver for the database used and the necessary pool configurations. Usually the JDBC driver can be found in the website of the DBMS provider or can be available in the database installation directory. This is important to know since we are not usually allowed to redistribute them.

The JDBC driver for creating connection pool to be shared via JNDI shall be placed in the `$TOMCAT_HOME/lib` directory, where `$TOMCAT_HOME` is the directory on which Tomcat is installed.

Note: Make sure to remove the JDBC driver from the Geoserver `WEB-INF/lib` folder when copied to the Tomcat shared libs, to avoid issues in JNDI DataStores usage.

The configuration is very similar between different databases. Here below some typical examples will be described.

PostgreSQL JNDI Configuration

For configuring a PostgreSQL JNDI pool you have to remove the Postgres JDBC driver (it should be named `postgresql-X.X-XXX.jdbc3.jar`) from the GeoServer `WEB-INF/lib` folder and put it into the `TOMCAT_HOME/lib` folder.

Tomcat Set-up

The first step to perform for creating a JNDI datasource (connection pool) is to edit the **context.xml** file inside `$TOMCAT_HOME/conf` directory. This file contains the different JNDI resources configured for Tomcat. In this case, we are going to configure a JNDI datasource for a PostgreSQL database. If the file is not present you should create it and add a content similar to the following:

```
<Context>
  <Resource
    name="jdbc/postgres" auth="Container" type="javax.sql.DataSource"
    driverClassName="org.postgresql.Driver"
    url="jdbc:postgresql://localhost:5432/testdb"
    username="admin"
    password="admin"
    maxActive="20"
    maxIdle="10"
    maxWait="10000"
    minEvictableIdleTimeMillis="300000"
    timeBetweenEvictionRunsMillis="300000"
    validationQuery="SELECT 1"/>
</Context>
```

Note: If the file is already present, do not add the `<Context></Context>` labels.

In the previous XML snippet, we configured a connection to a PostgreSQL database called **testdb** which have the host name as *localhost* and port number equal to 5432.

Note: Note that the user shall set proper *username* and *password* for the database.

Some of the parameters that can be configured for the JNDI connection pool are as follows:

- **maxActive** : The number of maximum active connections to use.
- **maxIdle** : The number of maximum unused connections.
- **maxWait** : The maximum number of **milliseconds** that the pool will wait.
- **poolPreparedStatements** : Enable the prepared statement pooling (very important for good performance).
- **maxOpenPreparedStatements** : The maximum number of prepared statements in pool.

- **validationQuery** : (default null) A validation query that double checks the connection is still alive before actually using it.
- **timeBetweenEvictionRunsMillis** : (default -1) The number of **milliseconds** to sleep between runs of the idle object evictor thread. When non-positive, no idle object evictor thread will be run.
- **numTestsPerEvictionRun** : (default 3) The number of objects to examine during each run of the idle object evictor thread (if any).
- **minEvictableIdleTimeMillis** : : (default 1000 * 60 * 30) The minimum amount of time, in **milliseconds**, an object may sit idle in the pool before it is eligible for eviction by the idle object evictor (if any).
- **removeAbandoned** : (default false) Flag to remove abandoned connections if they exceed the `removeAbandonedTimeout`. If set to true a connection is considered abandoned and eligible for removal if it has been idle longer than the `removeAbandonedTimeout`. Setting this to true can recover db connections from poorly written applications which fail to close a connection.
- **removeAbandonedTimeout** : (default 300) Timeout in **seconds** before an abandoned connection can be removed.
- **logAbandoned** : (default false) Flag to log stack traces for application code which abandoned a `Statement` or `Connection`.
- **testWhileIdle** : (default false) Flag used to test connections when idle.

Warning: The previous settings should be modified only by experienced users. Using wrong low values for `removeAbandonedTimeout` and `minEvictableIdleTimeMillis` may result in connection failures; if so try it is important to set-up **logAbandoned** to true and check your `catalina.out` log file.

More informations about the parameters can be found at the [DBCP documentation](#).

GeoServer Set-up

Launch GeoServer and navigate to the *Stores* → *Add new Store* section.

First, choose the *PostGIS (JNDI)* datastore and give it a name:

New data source

Choose the type of data source you wish to configure

Vector Data Sources

- ☐ **Directory of spatial files (shapefiles)** - Takes a directory of shapefiles and exposes it as a data store
- ☐ **PostGIS** - PostGIS Database
- ☒ **PostGIS (JNDI)** - PostGIS Database (JNDI)
- ☐ **Properties** - Allows access to Java Property files containing Feature information
- ☐ **Shapefile** - ESRI(tm) Shapefiles (*.shp)

Fig. 280: PostGIS JNDI Store Configuration

And then you can configure the connection pool:

Edit Vector Data Source

Edit an existing vector data source

PostGIS (JNDI)

PostGIS Database (JNDI)

Basic Store Info

Workspace *

test ▼

Data Source Name *

test

Description

☐ Enabled

Connection Parameters

jndiReferenceName *

java:comp/env/jdbc/postgres

schema

Namespace *

http://test

fetch size

1000

☐ Expose primary keys

Primary key metadata table

Session startup SQL

Session close-up SQL

☒ Loose bbox

☒ Estimated extends

When you are doing this, make sure the schema is properly configured, or the datastore will list all the tables it can find in the schema it is given access to.

Microsoft SQLServer JNDI Configuration

Before configuring a SQLServer connection pool you must follow these [Guidelines](#).

Warning: You must remove the `sqljdbc.jar` file from the `WEB-INF/lib` folder and put it inside the `$TOMCAT_HOME/lib` folder.

Tomcat Set-up

In this case, we are going to configure a JNDI datasource for a SQLServer database. You shall create/edit the **context.xml** file inside `$TOMCAT_HOME/conf` directory with the following lines:

```
<Context>
  <Resource
    name="jdbc/sqlserver"
    auth="Container"
    type="javax.sql.DataSource"
    url="jdbc:sqlserver://localhost:1433;databaseName=test;user=admin;
    ↪password=admin;"
    driverClassName="com.microsoft.sqlserver.jdbc.SQLServerDriver"
    username="admin"
    password="admin"
    maxActive="20"
    maxIdle="10"
    maxWait="10000"
    minEvictableIdleTimeMillis="300000"
    timeBetweenEvictionRunsMillis="300000"
    validationQuery="SELECT 1"/>
</Context>
```

Note: Note that **database name**, **username** and **password** must be defined directly in the URL.

GeoServer Set-up

Launch GeoServer and navigate to the *Stores* → *Add new Store* section.

Then choose the *Microsoft SQL Server (JNDI)* datastore and give it a name:

After, you can configure the connection pool:

Oracle JNDI Configuration

Before configuring an Oracle connection pool you should download the Oracle plugin from the [GeoServer Download Page](#) and then put the `ojdbc14.jar` file into the `$TOMCAT_HOME/lib` folder.

New data source

Choose the type of data source you wish to configure

Vector Data Sources








-  [Directory of spatial files \(shapefiles\)](#) - Takes a directory of shapefiles and exposes it as a data store
-  [Microsoft SQL Server - Microsoft SQL Server](#)
-  [Microsoft SQL Server \(JNDI\) - Microsoft SQL Server \(JNDI\)](#)
-  [PostGIS - PostGIS Database](#)
-  [PostGIS \(JNDI\) - PostGIS Database \(JNDI\)](#)
-  [Properties](#) - Allows access to Java Property files containing Feature information
-  [Shapefile](#) - ESRI(tm) Shapefiles (*.shp)

Fig. 282: Microsoft SQLServer JNDI Store Configuration

New Vector Data Source

Add a new vector data source

Microsoft SQL Server (JNDI)

Microsoft SQL Server (JNDI)

Basic Store Info

Workspace *

test ▼

Data Source Name *

test

Description

☒ Enabled

Connection Parameters

jndiReferenceName *

java:comp/env/jdbc/sqlserver

Fig. 283: Microsoft SQLServer JNDI Store Configuration

Warning: You must remove the `ojdbc14.jar` file from the `WEB-INF/lib` folder and put it inside the `$TOMCAT_HOME/lib` folder.

Tomcat Set-up

First you must create/edit the **context.xml** file inside `$TOMCAT_HOME/conf` directory with the following lines:

```
<Context>
    <Resource
        name="jdbc/oralocal"
        auth="Container" type="javax.sql.DataSource"
        url="jdbc:oracle:thin:@localhost:1521:xe"
        driverClassName="oracle.jdbc.driver.OracleDriver"
        username="dbuser"
        password="dbpasswd"
        maxActive="20"
        maxIdle="3"
        maxWait="10000"
        minEvictableIdleTimeMillis="300000"
        timeBetweenEvictionRunsMillis="300000"
        poolPreparedStatements="true"
        maxOpenPreparedStatements="100"
        validationQuery="SELECT SYSDATE FROM DUAL" />
</Context>
```

GeoServer Set-up

Launch GeoServer and navigate to the *Stores* → *Add new Store* section.

Then choose the *Oracle NG (JNDI)* datastore and give it a name:

New data source

Choose the type of data source you wish to configure

Vector Data Sources







-  Directory of spatial files (shapefiles) - Takes a directory of shapefiles and exposes it as a data store
-  Microsoft SQL Server - Microsoft SQL Server
-  Microsoft SQL Server (JNDI) - Microsoft SQL Server (JNDI)
-  Oracle NG - Oracle Database
-  **Oracle NG (JNDI) - Oracle Database (JNDI)**
-  Oracle NG (OCI) - Oracle Database (OCI)

Fig. 284: Oracle JNDI Store Configuration

After, you can configure the connection pool:

New Vector Data Source

Add a new vector data source

Oracle NG (JNDI)
Oracle Database (JNDI)

Basic Store Info

Workspace *

test ▼

Data Source Name *

XE

Description

Shows how to retrieve a connection pool from JNDI

☒ Enabled

Connection Parameters

jndiReferenceName *

java:comp/env/jdbc/oralocal

schema

DBUSER

Fig. 285: Oracle JNDI Store Configuration

Note: In Oracle the schema is usually the user name, upper cased.

Configuring Connection Pools for production usage

Connection waiting time and relation with other parameters

In general it is important to set the connection waiting time in a way that the connection pool does not become a place where to queue threads executing requests under big load. It is indeed possible that under big load threads executing requests for a vector layer will outnumber the available connections in the pool hence such threads will be blocked trying to acquire a new connection; if the number of connections is much smaller than the number of incoming requests and the max wait time is quite big (e.g. 60 seconds) we will find ourselves in the condition to have many threads waiting for a long time to acquire a connection after most of the resources they need will be allocated, especially the memory back buffer if these are WMS requests.

The *max wait time* in general shall be set accordingly to the expected maximum execution time for a requests, end-to-end. This include things like, accessing the file system, loading the data. As an instance if we take into account WMS requests we are allowed to specify a maximum response time, therefore if set this to N seconds the max wait time should be set to a value smaller than that since we don't want to waste resources having threads blocked unnecessarily waiting for a connection. In this case it shall be preferable to fail fast to release resources that might be used unnecessarily otherwise.

Maximizing sharing of Connection Pools

How the data is organized between database, schemas and table impact the degree of flexibility we have when trying to best share

- Splitting tables into many schemas makes it hard for GeoServer to access tables belonging to different schemas unless we switch to JNDI, since the schema must be specified as part of the connection parameters when using internal pools
- Using different users for different schemas prevent JNDI from working efficiently across schemas. It's best to use when possible a single dedicated account across schemas
- Generally speaking having a complex combination of users and schema can lead to inefficient split of available connections in multiple pools

Long story short, whenever it's possible strive to make use of a small number of users and if not using JNDI to a small number of schema, although JNDI is a must for organization willing to create a complex set up where different workspaces (i.e Virtual Services) serve the same content differently.

Query Validation

Regardless of how we configure the validation query it is extremely important that we always remember to validate connections before using them in GeoServer; not doing this might lead to spurious errors due to stale connections sitting the pool. This can be achieved with the internal connection pool (via the validate connections box) as well as with the pools declared in JNDI (via the validation query mechanism); it is worth to remind that the latter will account for finer grain configurability.

Installing and Configuring the Monitoring plugin

The monitoring extension provides a request monitor for GeoServer. It captures information about each request a GeoServer instance handles and produces reports based on the request data.

Installing the Monitoring Extension

Monitoring is an official extension, as such it can be found alongside any GeoServer release. The extension is split into two modules, “core” and “hibernate”, where core provides the basic underpinnings of the module and allows to monitor “live” requests, while the hibernate extension provides database storage of the requests.

1. Get the monitoring zip files, already downloaded for you, from the training material `data\plugins` folder (search for zip files containing the `monitoring` word, there will be two)
2. Extract the contents of the archives into the `<TRAINING_ROOT>/tomcat-6.0.36/instances/instance1/webapps/geoserver/WEB-INF/lib` directory of the GeoServer installation.

Verifying the Installation

There are two ways to verify that the monitoring extension has been properly installed.

- Start GeoServer and open the [Web Administration interface](#). Log in using the administration account. If successfully installed, there will be a *Monitor* section on the left column of the home page.

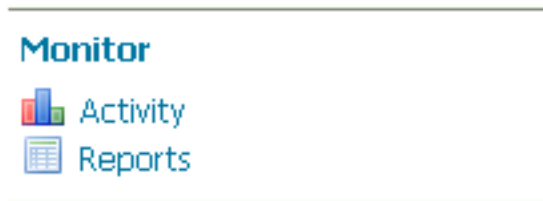


Fig. 286: *Monitoring section in the web admin interface*

- Start GeoServer and navigate to the current data directory. If successfully installed, a new directory named `monitoring` will be created in the data directory.

Basic Configuration of the Monitor Extension

Many aspects of the monitor extension are configurable. All configuration files are stored in the data directory under the `monitoring` directory:

```
<data_directory>
  monitoring/
    db.properties
    filter.properties
    hibernate.properties
    monitor.properties
```

The function of these files will be discussed below.

Monitor Mode

The monitoring extension supports different “monitoring modes” that control how request data is captured and stored. Currently three modes are supported:

- **history** (*Default*) - Only historical request information is available. No live information is maintained.
- **live** - Only information about live requests is maintained.
- **mixed** - A combination of live and history. This mode is experimental.

The mode is set in the `monitor.properties` file.

Note: For the Virtual Machine GeoServer instance we are “live” mode.

History Mode

History mode persists information about all requests in an external database. It does not provide any real time information. This mode is appropriate in cases where a user is most interested in analyzing request history over a given time period.

Live Mode

Live mode only maintains short lived information about requests that are currently executing. It also maintains a small buffer of recent requests. No external database is used with this mode and no information is persisted for the long term.

This mode is most appropriate in cases where a user only cares about what a server is doing in real time and is not interested about request history.

Mixed Mode

Mixed mode combines both live and history mode in that it maintains both real time information and persists all request data to the monitoring database. This mode however is experimental and comes with more overhead than the other two modes. This is because mixed mode must perform numerous database transactions over the life cycle of a single request (in order to maintain live information), whereas history mode only has to perform a single database transaction for a request.

This mode is most appropriate when both real time request information and request history are desired. This mode is also most appropriate in a clustered server environment in which a user is interested in viewing real time request information about multiple nodes in a cluster.

Monitor Database

By default monitored request data is stored in an embedded H2 database located in the `monitoring` directory. This can be changed by editing the `db.properties` file:

```
# default configuration is for h2
driver=org.h2.Driver
url=jdbc:h2:file:${GEOSERVER_DATA_DIR}/monitoring/monitoring
```

For example to store request data in an external PostgreSQL database:


```
driver=org.postgresql.Driver
url=jdbc:postgresql://localhost:5432/monitoring
username=bob
password=foobar
```

Warning: The above is just an example. Does not match the training users and environment.

Request Filters

By default not all requests are monitored. Those requests excluded include any web admin requests or any monitor HTTP API requests. These exclusions are configured in the `filter.properties` file:

```
/rest/monitor/**
/web
/web/**
```

These default filters can be changed or extended to filter more types of requests. For example to filter out all WFS requests:

```
/wfs
```

How to determine the filter path

The contents of `filter.properties` are a series of ant-style patterns that are applied to the *path* of the request. Consider the following request:

```
http://localhost:8083/geoserver/wms?request=getcapabilities
```

The path of the above request is `/wms`. In the following request:

```
http://localhost:8083/geoserver/rest/workspaces/topp/datastores.xml
```

The path is `/rest/workspaces/topp/datastores.xml`.

In general, the path used in filters is comprised of the portion of the URL after `/geoserver` (including the preceding `/`) and before the query string `?`:

```
http://<host>:<port>/geoserver/<path>?<queryString>
```

Note: For more information about ant-style pattern matching, see the [Apache Ant manual](#).

1. Go to the Map [Map Preview](#) and open the *geosolutions:Counties* layer clicking on the OpenLayer link.
2. Perform a few times zoom the map.
3. Use also the GML preview for said layer
4. Navigate to the [Monitor/Reports section](#)
5. Click on OWS Request Summary to show a detailed chart like the following:

OWS Request Summary

Categorized Overview of OWS Request Activity

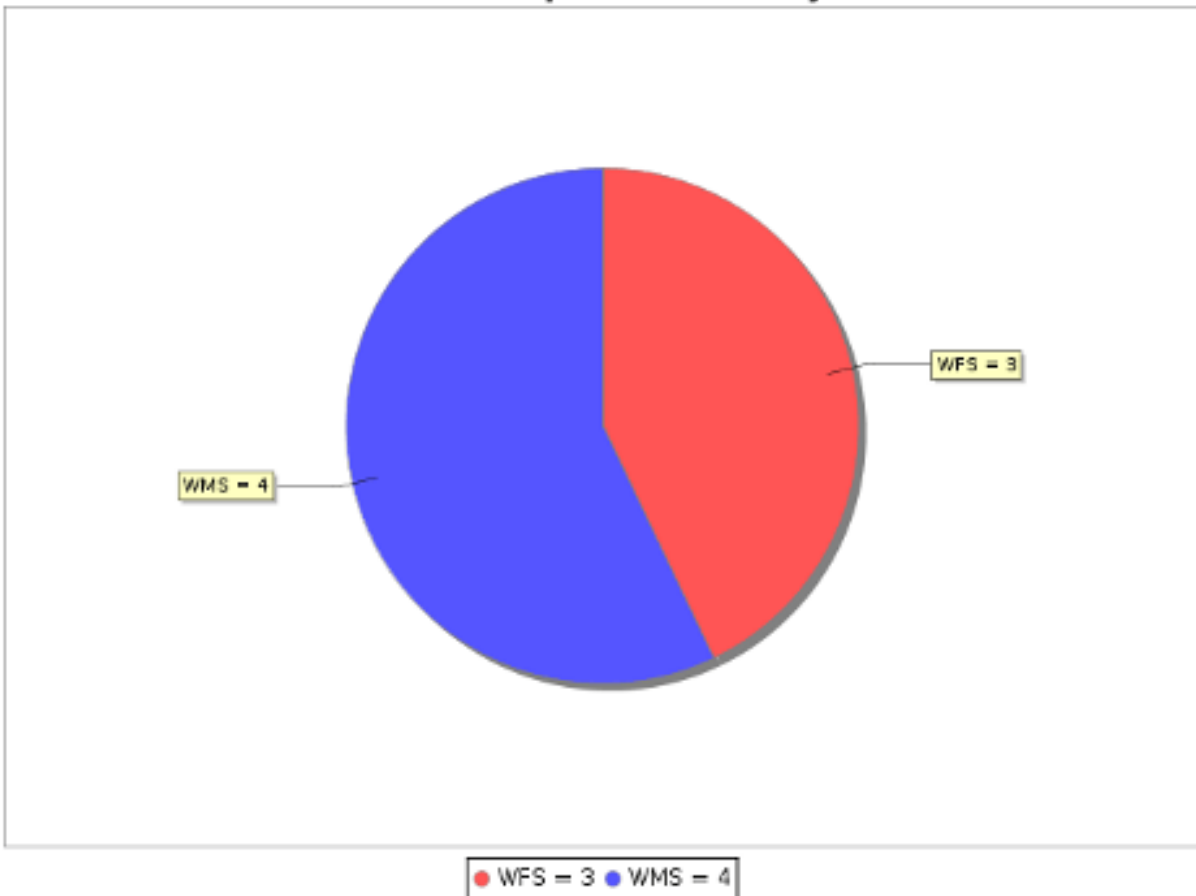
Overview

WFS

WMS

WCS

OWS Request Summary



Logging all requests on the file system

The history mode logs all requests into a database. This can put a very significant strain on the database and can lead to insertion issues as the request table begins to host millions of records.

As an alternative to the history mode it's possible to enable the auditing logger, which will log the details of each request in a file, which is periodically rolled. Secondary applications can then process these log files and built ad-hoc summaries off line.

Configuration

The `monitor.properties` file can contain the following items to enable and configure file auditing:

1. Go to the `${GEOSERVER_DATA_DIR}/monitoring` and open the `monitor.properties` then append the following configuration:

```
audit.enabled=true
audit.path=${TRAINING_ROOT}
audit.roll_limit=20
```

2. Replace `${TRAINING_ROOT}` with the full path to the workshop root folder, and remember to always use forward slashes, `/`, in the path, regardless of the operating system. For example, on Windows the path might look like `c:/data/Training_2.5.X-32`
3. Go to the Map [Map Preview](#) and open the `geosolutions:states` layer clicking on the `OpenLayer` link.
4. Perform a few times zoom the map.
5. Open the new created log file (named like `geoserver_audit_yyyymmdd_nn.log`) located at `${TRAINING_ROOT}`.

Note:

- **audit.enable:** is used to turn on the logger (it is off by default).
 - **audit.path:** is the directory where the log files will be created.
 - **audit.roll_limit:** is the number of requests logged into a file before rolling happens.
-

Note: The files are also automatically rolled at the beginning of each day.

Outputs and contents

The log directory will contain a number of log files following the `geoserver_audit_yyyymmdd_nn.log` pattern. The `nn` is increased at each roll of the file. The contents of the log directory will look like:

```
geoserver_audit_20110811_2.log
geoserver_audit_20110811_3.log
geoserver_audit_20110811_4.log
geoserver_audit_20110811_5.log
geoserver_audit_20110811_6.log
geoserver_audit_20110811_7.log
geoserver_audit_20110811_8.log
```

Customizing the log contents

The log contents are driven by three FreeMarker templates. We can customize them to have the log file be a CSV file for example.

1. On the file system navigate to the GeoServer data directory located at `$GEOSERVER_DATA_DIR`.
2. In the monitoring directory create a new file named `header.ftl` (is used once when a new log file is created to form the first few lines of the file).
3. Open `header.ftl` in the text editor of your choice and enter the following content:

```
# start time,services,version,operation,url,response content type,total time,  
↪response length,error flag
```

4. Create another file named `content.ftl`.
5. Open `content.ftl` in the text editor of your choice and enter the following content:

```
${startTime?datetime?iso_utc_ms},${service!""},${owsVersion!""},${operation!""}, "${  
↪{path!""}${queryString!""},${responseContentType!""},${totalTime},${  
↪{responseLength?c}},<#if error??>failed<#else>success</#if>
```

6. Create a last file named `footer.ftl`, and leave its contents empty
7. Run again a few requests, the log files should contain something like the following now:

```
# start time,services,version,operation,url,response content type,total time,  
↪response lenght,error flag  
2012-06-07T10:37:09.725Z,WMS,1.1.1,GetMap,"/geosolutions/  
↪wmsLAYERS=geosolutions:ccounties&STYLES=&FORMAT=image/png&SERVICE=WMS&VERSION=1.  
↪1.1&REQUEST=GetMap&SRS=EPSG:4269&BBOX=-106.17254516602,39.489453002927,-105.  
↪18378466798,40.054948608395&WIDTH=577&HEIGHT=330",image/png,59,30420,success  
2012-06-07T10:37:10.075Z,WMS,1.1.1,GetMap,"/geosolutions/  
↪wmsLAYERS=geosolutions:ccounties&STYLES=&FORMAT=image/png&SERVICE=WMS&VERSION=1.  
↪1.1&REQUEST=GetMap&SRS=EPSG:4269&BBOX=-105.84010229493,39.543136352537,-105.  
↪34572204591,39.825884155271&WIDTH=577&HEIGHT=330",image/png,45,18692,success
```

How to measure performances with JMeter

In this submodule we are going to describe how to use the JMeter tool :

Configuring JMeter for a simple test

Apache JMeter is an open source Java desktop application, built to verify functional behavior, perform load tests, and measure performance.

This section explains how to run performance tests using JMeter in order to evaluate the GeoServer performances when serving WMS requests. The performance test aim to stress the server and evaluate the response time and throughput with an increasing number of simulated users sending concurrent request to the server.

Note: Ideally, to avoid adding extra load to the server JMeter should run on a different machine.

Warning: If you have performed the exercises in the security section, please go back to the layer and service security pages and open access to everybody, on all data and all services, before performing the exercises in this section

1. From the training root, on the command line, run `jmeter.bat` (or `jmeter.sh` if you're on Linux) to start JMeter:

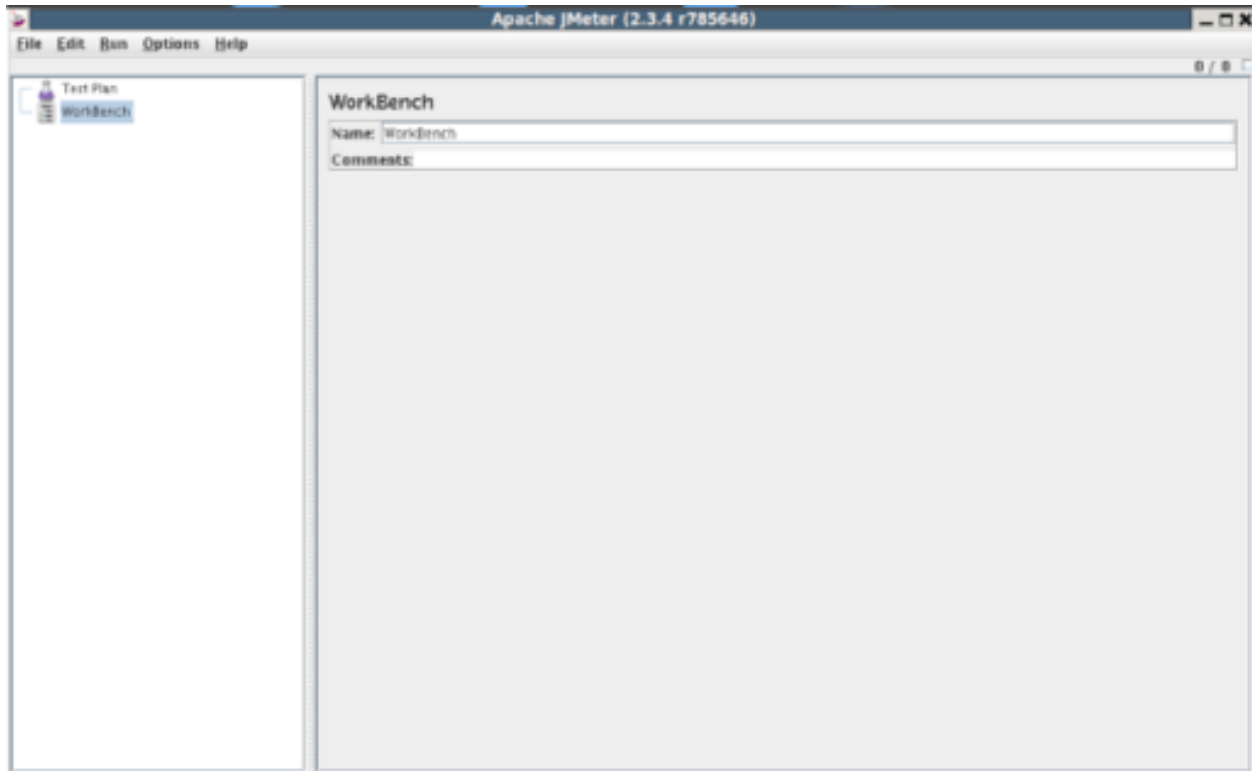


Fig. 287: *jMeter interface*

2. Add a new **Thread Group** with the mouse right click on `Test Plan` tree node:
3. Add a new **Loop Controller** with the mouse right click on `Thread Group` tree node:
4. In the `Thread Group` panel set the number of thread for the test to 4 (this represents the number of simultaneous requests that are made to GeoServer) and the ramp-up period to 60. Also, check `Forever` on the **Loop Count** field.
5. Right click on the `Loop Controller` tree node and add a new **HTTP Request** element:
6. Add the following listeners by right clicking on `Test Plan` tree node: `View results Tree`, `Summary Report`, `Graph results`
7. In the **HTTP Request** enter the following basic configuration:

Field	Value
Server Name or IP	localhost
Port Number	8083
Path	geoserver/ows

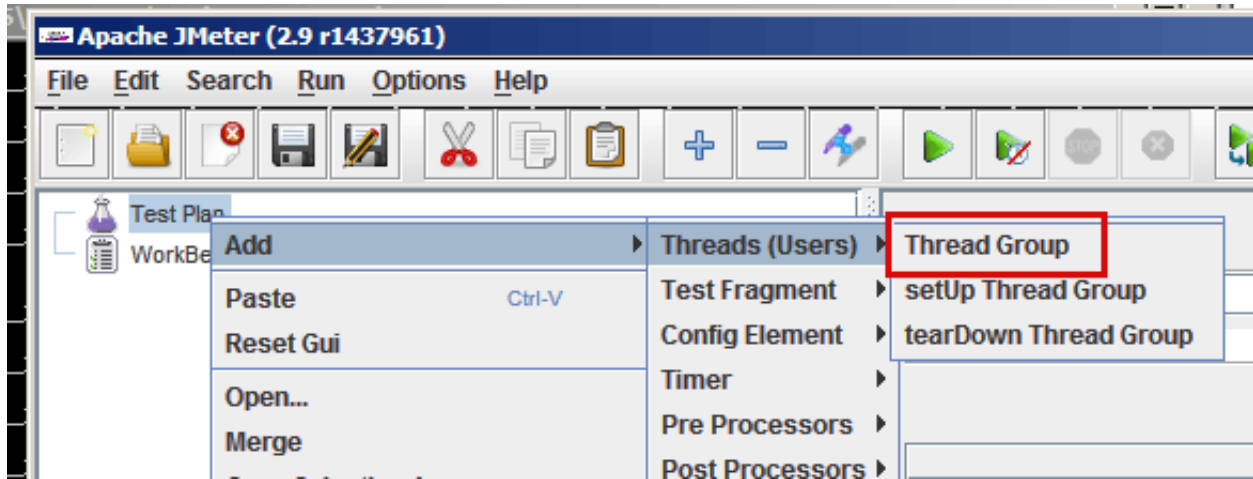


Fig. 288: Adding a new Thread Group

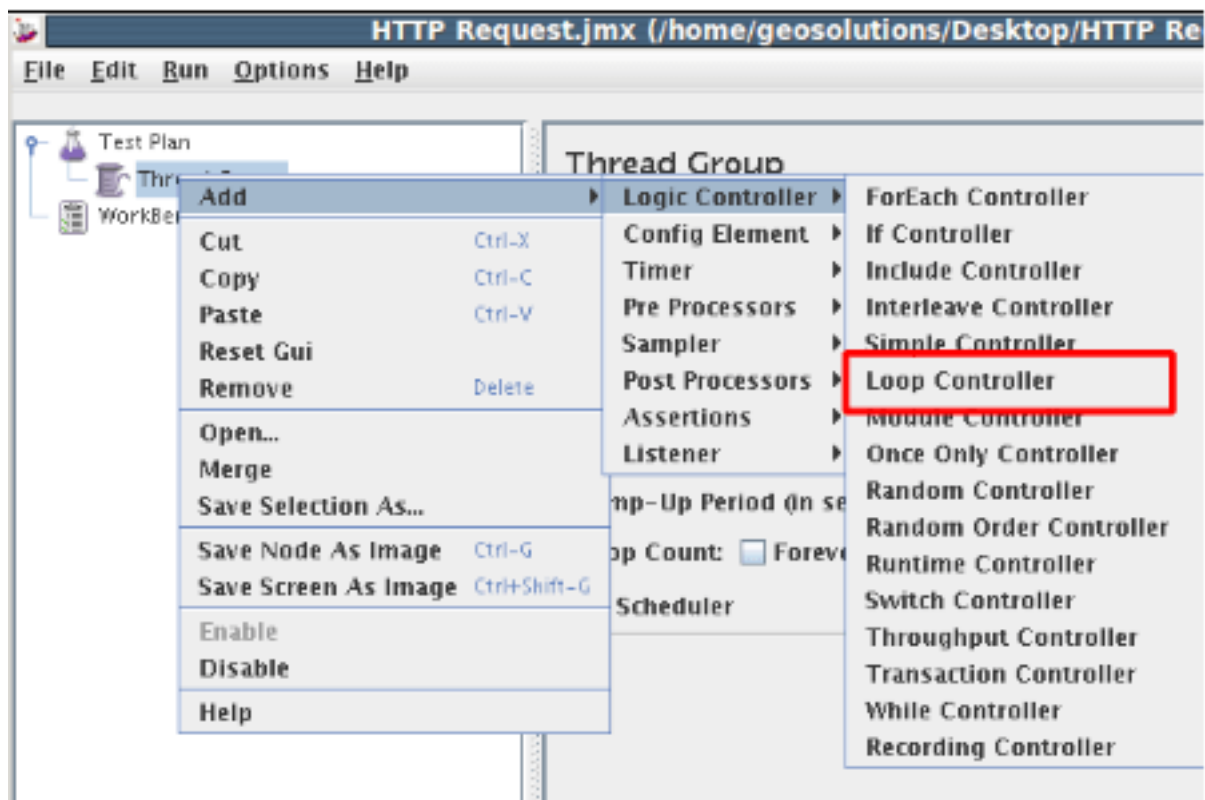


Fig. 289: Adding a new Loop Controller

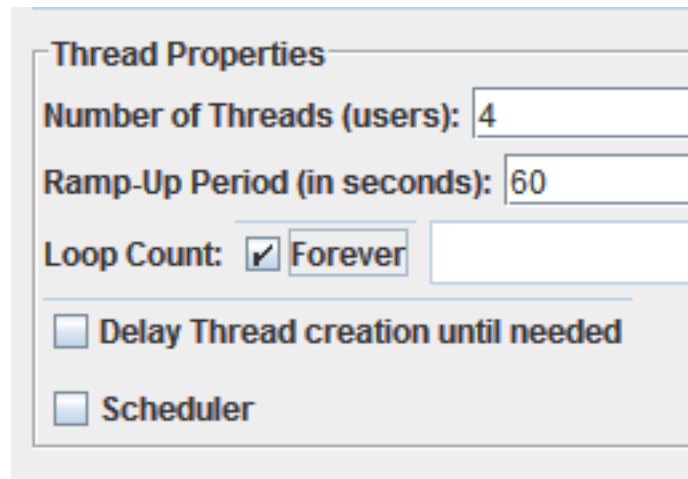


Fig. 290: Setting the Thread Group

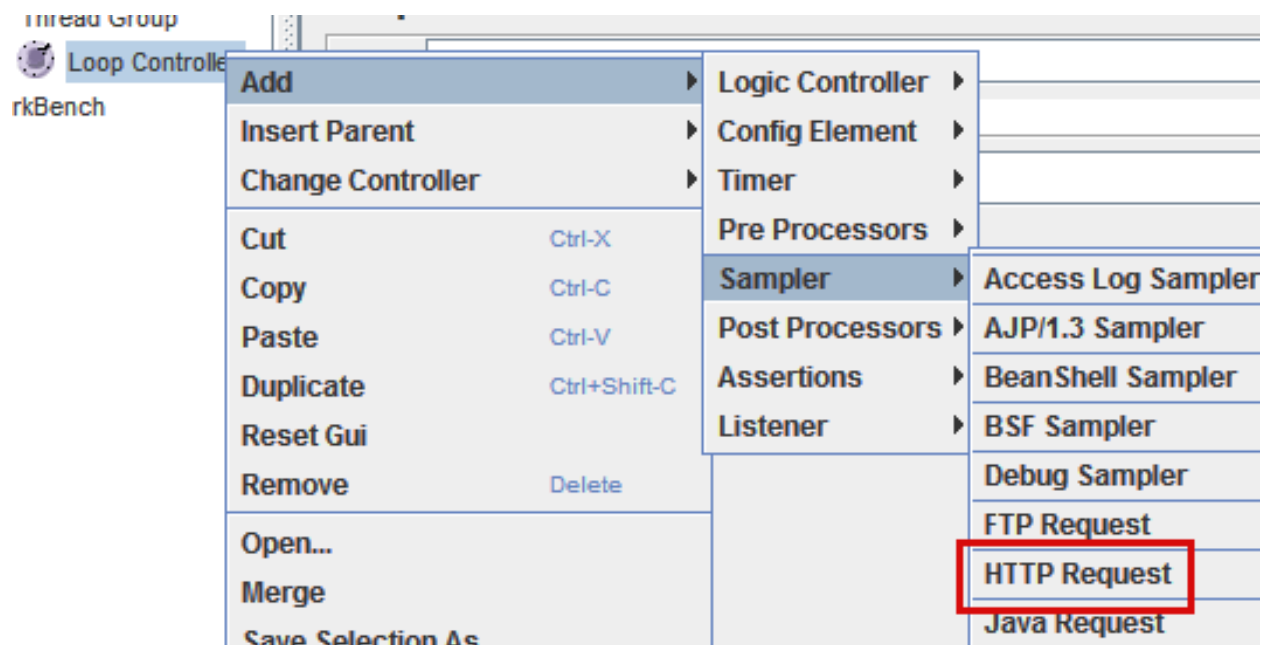


Fig. 291: Adding a new HTTP Request

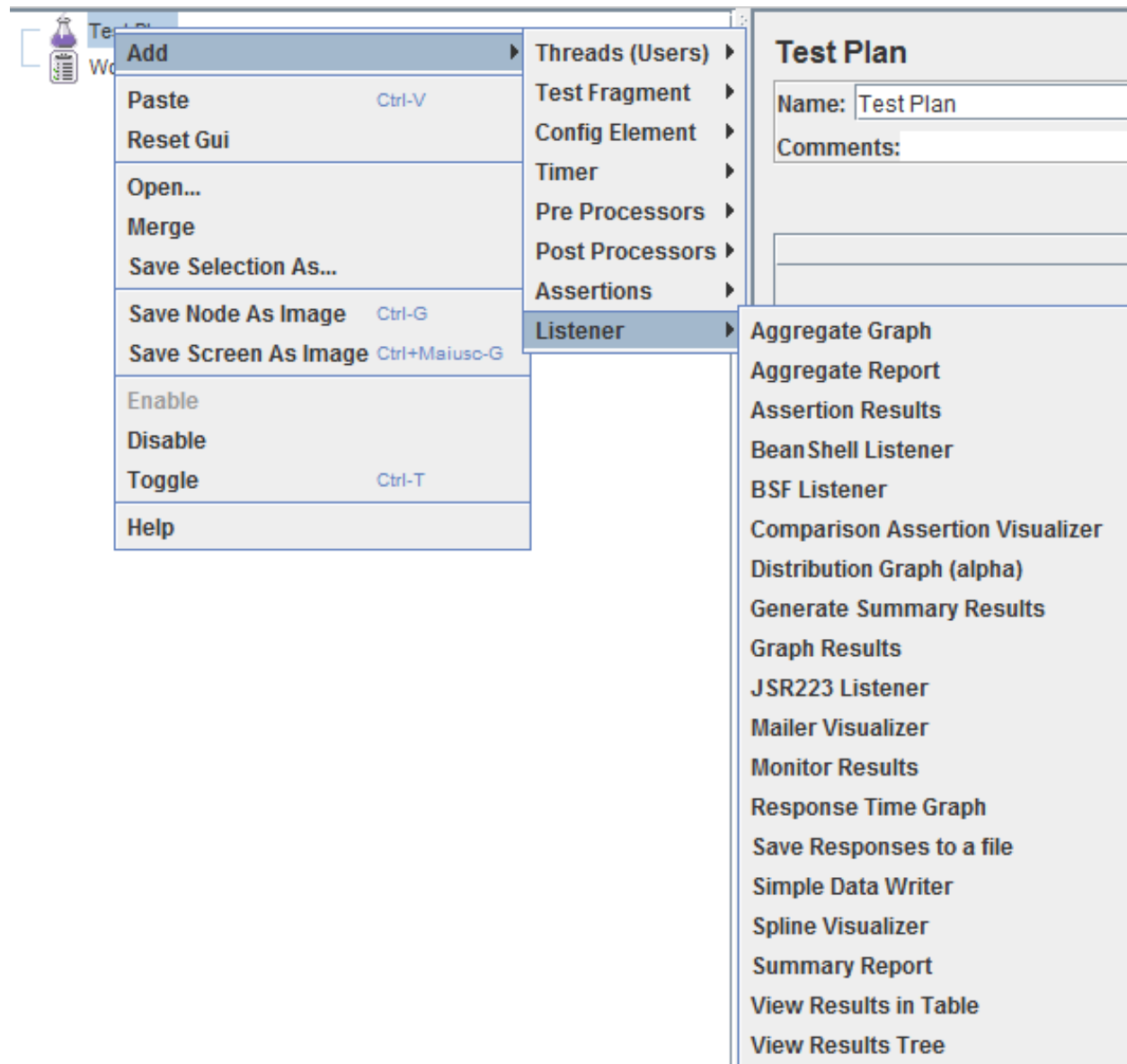


Fig. 292: Adding a Listeners

HTTP Request

Name: HTTP Request

Comments:

Web Server

Server Name or IP: localhost Port Number: 8083

HTTP Request

Implementation: Protocol [http:]: Method: GET Content encoding:

Path: geoserver/ows

Fig. 293: HTTP Request host/port/path configuration

- From the training data root directory, open the `data/jmeter_data/jmeter_request_params.txt`, select and copy its contents in the clipboard, then click “Add from Clipboard” in the “HTTP request” panel to setup a sample GetMap request:

Parameters **Post Body**

Send Parameters With the Request:

Name:	Value
request	GetMap
service	WMS
version	1.1.1
layers	geosolutions:bbuildings
styles	polygon
sr	EPSG:2876
bbox	3062098.0062867,1252158.6903078,3065695.2047731...
width	750
height	500
format	image/png

Detail Add **Add from Clipboard** Delete Up Down

Fig. 294: HTTP parameters configuration

At this point JMeter is configured to run a GeoServer performance test:

- Select on **Thread Group** tree node and then click on Run tool and select **Start** to starting the JMeter test.
- Select View Results Tree to directly see the request informations produced and the request result:
- Select Summary report to view the statistical information about the requests:
- Select Graph Results to analyze the technical trend of the requests:

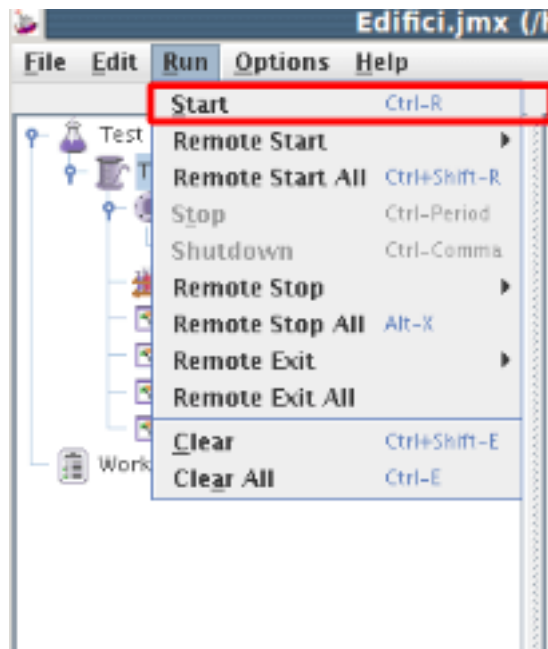


Fig. 295: starting jMeter test

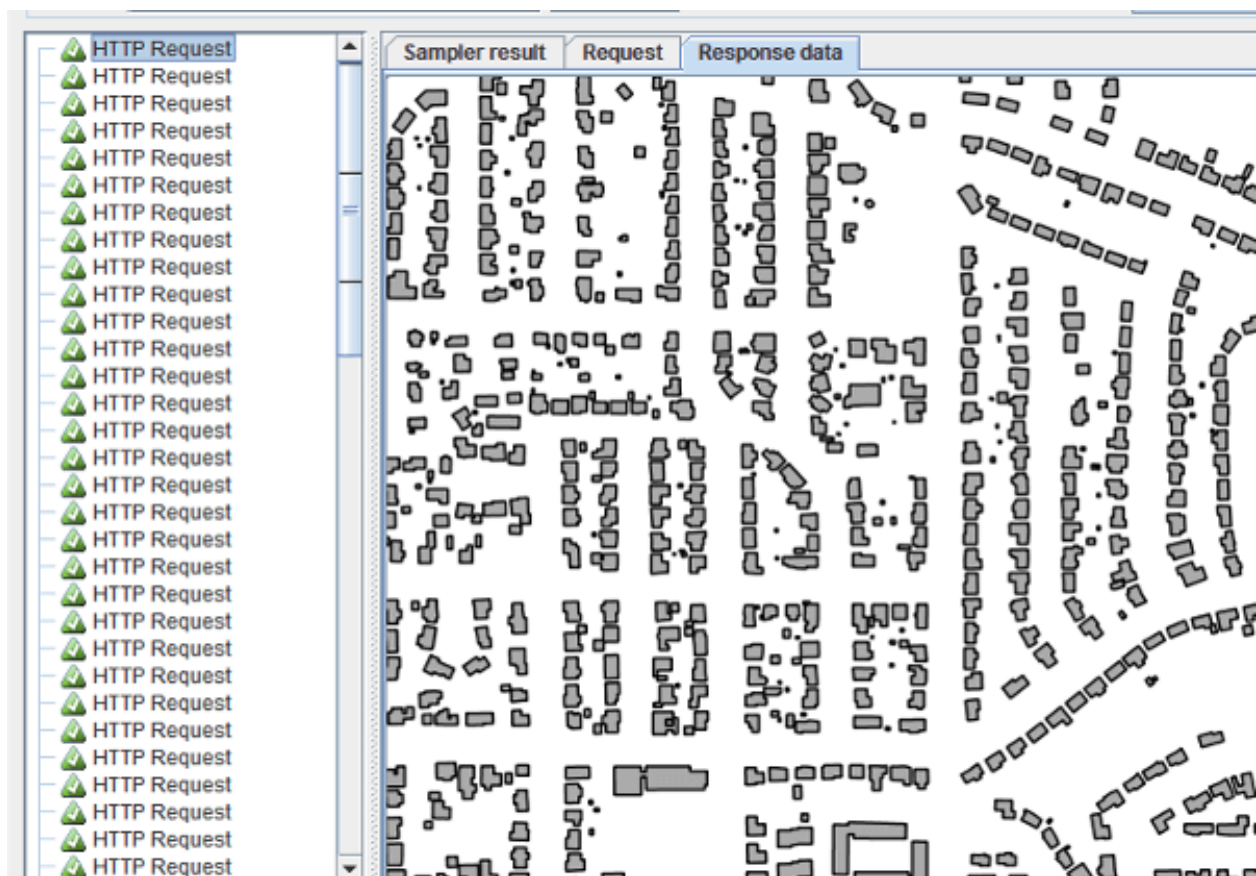


Fig. 296: The View Results Tree panel

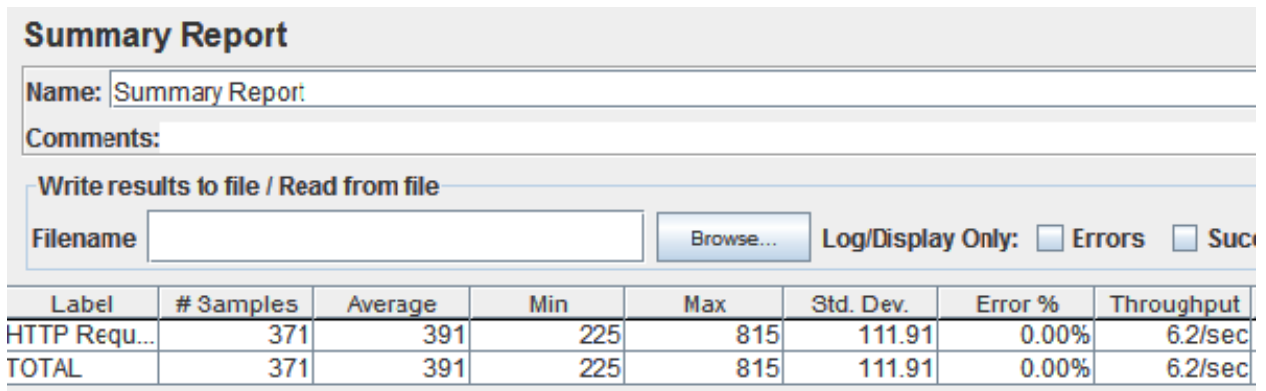


Fig. 297: The Aggregate Graph panel

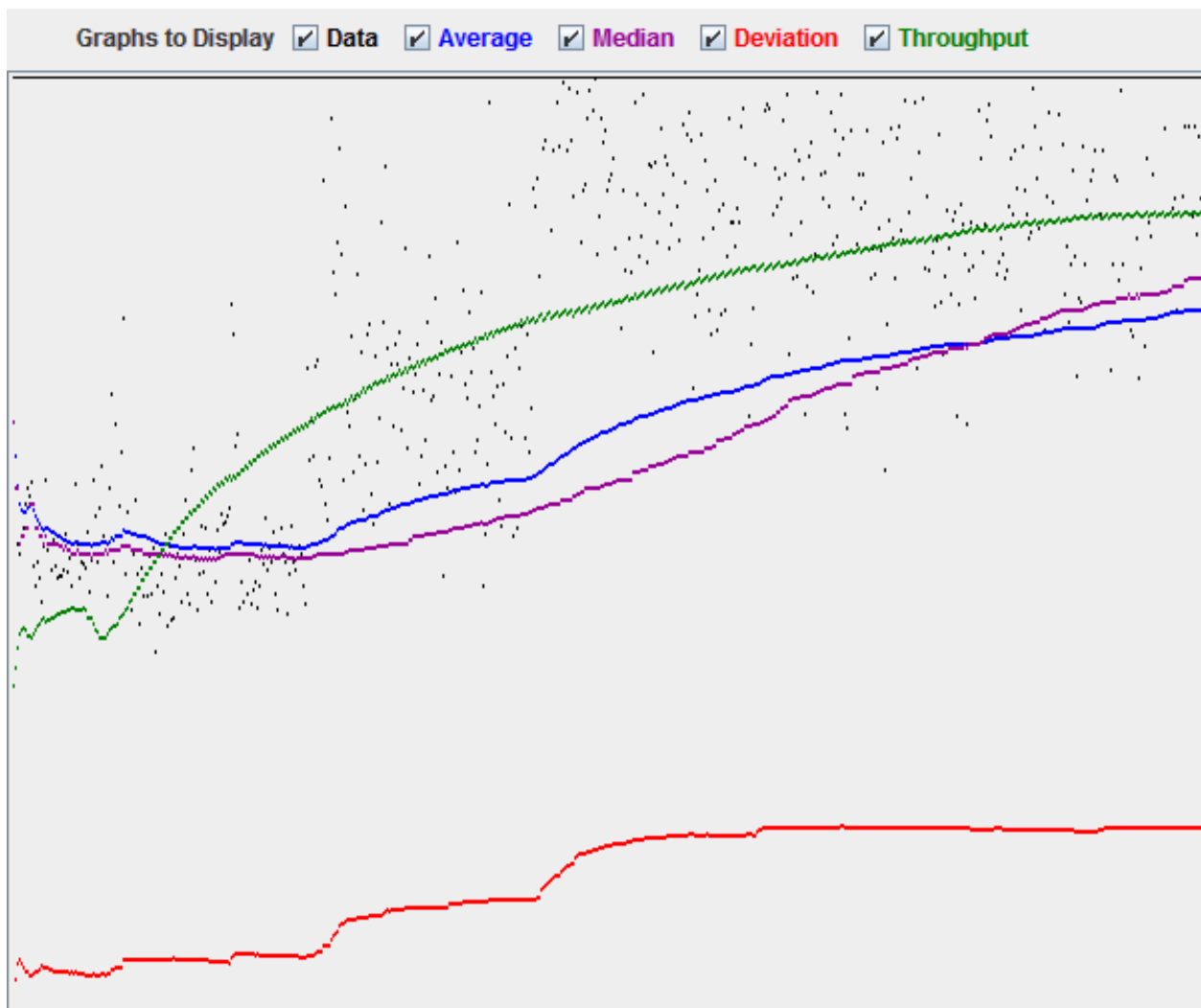


Fig. 298: The Spline Visualizer panel

Configuring JMeter for a Multiscale test

This chapter explains how to create a custom randomized Multiscale test with a set of multiple concurrent threads.

In the first paragraph is described how to generate a CSV file for randomized requests at different scales. In the second one is shown how to configure a new JMeter test with multiple simultaneous threads.

Create CSV file

1. Open the file `gdal.bat` under `$TRAINING_ROOT` folder inside the training home folder.
2. Run:

```
cd %TRAINING_ROOT%\geoserver_data\data\boulder
gdalinfo srtm_boulder.tiff
```

3. The output of the command will be something like this:

```
Driver: GTiff/GeoTIFF
Files: srtm_boulder.tiff
Size is 2520, 1800
Coordinate System is:
GEOGCS["WGS 84",
    DATUM["WGS_1984",
        SPHEROID["WGS 84",6378137,298.257223563,
            AUTHORITY["EPSG","7030"]],
        AUTHORITY["EPSG","6326"]],
    PRIMEM["Greenwich",0],
    UNIT["degree",0.0174532925199433],
    AUTHORITY["EPSG","4326"]]
Origin = (-105.70013888888890,40.300138888888888)
Pixel Size = (0.00027777777777778,-0.00027777777777778)
Metadata:
  AREA_OR_POINT=Area
Image Structure Metadata:
  INTERLEAVE=BAND
Corner Coordinates:
Upper Left  (-105.7001389,  40.3001389) (105d42' 0.50"W, 40d18' 0.50"N)
Lower Left  (-105.7001389,  39.8001389) (105d42' 0.50"W, 39d48' 0.50"N)
Upper Right (-105.0001389,  40.3001389) (105d 0' 0.50"W, 40d18' 0.50"N)
Lower Right (-105.0001389,  39.8001389) (105d 0' 0.50"W, 39d48' 0.50"N)
Center      (-105.3501389,  40.0501389) (105d21' 0.50"W, 40d 3' 0.50"N)
Band 1 Block=256x256 Type=Int16, ColorInterp=Gray
  Overviews: 1260x900, 630x450, 315x225, 158x113, 79x57, 40x29
```

4. The information needed for create a Multiscale CSV file are:

Tile Size	256 x 256
Pixel Size	0.00027777777777778
Bounding Box	((-105.7001389, -105.0001389), (39.8001389, 40.3001389))

5. Run:

```
cd %TRAINING_ROOT%\data\jmeter_data

wms_request.py -count 100 -region -105.7 39.8 -105.0 40.3 -minres 0.
↪00028 -maxres 0.00224 -minsize 256 256 -maxsize 1024 1024 > multiscale.
↪CSV
```

wms_request.py is a python script which generates randomized request at different bounding box and resolutions. The parameters are described in the following table:

Parameter	Description
<i>count</i>	Indicates the number of requests to generate
<i>region</i>	Indicates the maximum bounding box of each request
<i>minres/maxres</i>	Indicates the minimum and maximum value for the Pixel Size to request (Typically it should be at least the minimum resolution)
<i>minsize/maxsize</i>	Indicates the minimum and maximum dimensions of the requested image (Typically it should be at least as big as the tile size)

The CSV file is structured following the rule `$width;$height;$bbox`.

For example `290;444;-105.5904,39.910198,-105.48776,40.067338` indicates a request of size **290x444** and Bounding box **[-105.5904,39.910198,-105.48776,40.067338]**.

JMeter must be configured for parsing the CSV file correctly by using the CSV Data Set Config element.

Configure JMeter

1. From the training root, on the command line, run `jmeter.bat` (or `jmeter.sh` if you're on Linux) to start JMeter:
2. Add 3 new **Thread Group** called *1, 2, 4*
3. For each **Thread Group** set the Number of Thread(users) field equal to the **Thread Group** name, the ramp-up period and Loop Count fields to 1.
4. In the **Test Plan** section, check the *Run Thread Groups consecutively* checkbox
5. Add a new **Loop Controller** for each **Thread Group** object:
6. Each **Loop Controller** should be configured following this schema:

Thread Group • 1	Loop Controller → Loop Count • 100
Thread Group • 2	Loop Controller → Loop Count • 50
Thread Group • 4	Loop Controller → Loop Count • 50

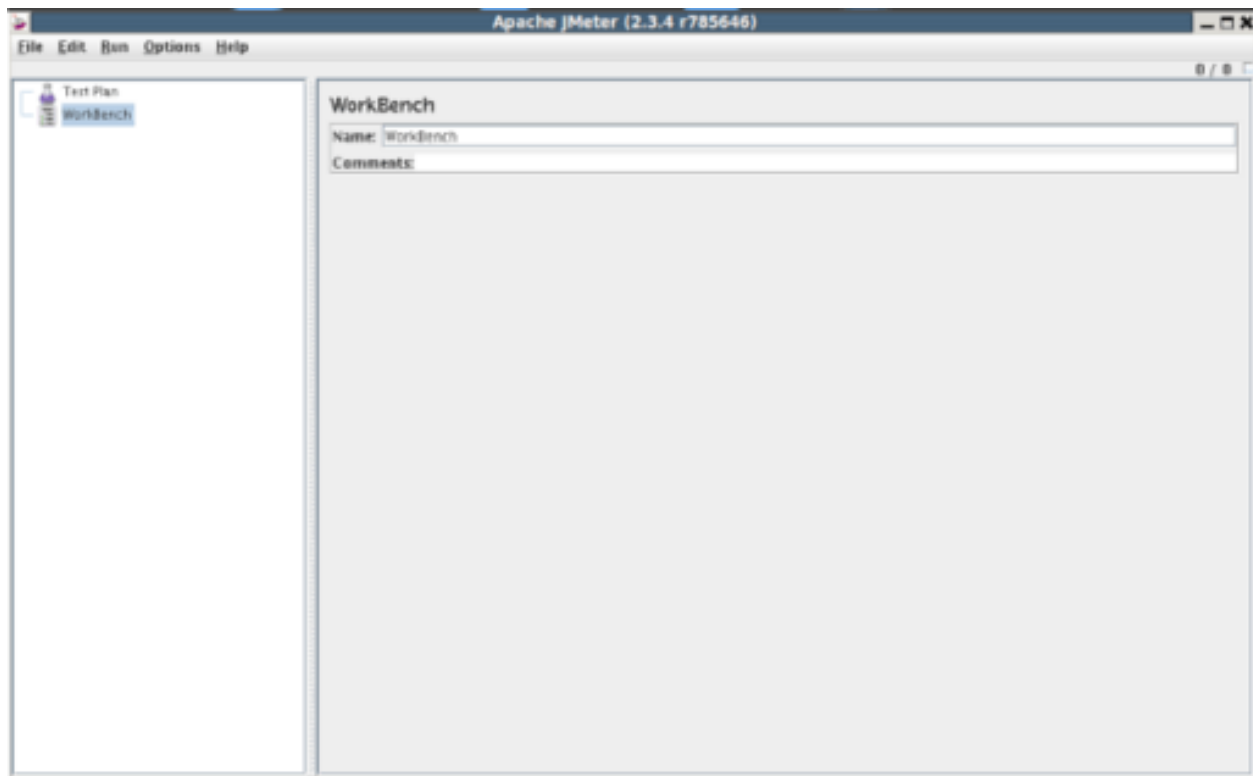


Fig. 299: *jMeter* interface

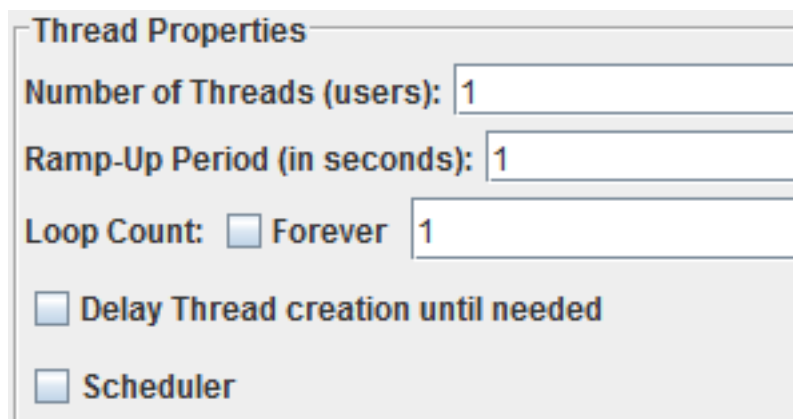


Fig. 300: *Setting the Thread Group*

- Right click on each `Loop Controller` tree node and add a new **HTTP Request** element with the same name of the **Thread Group**:

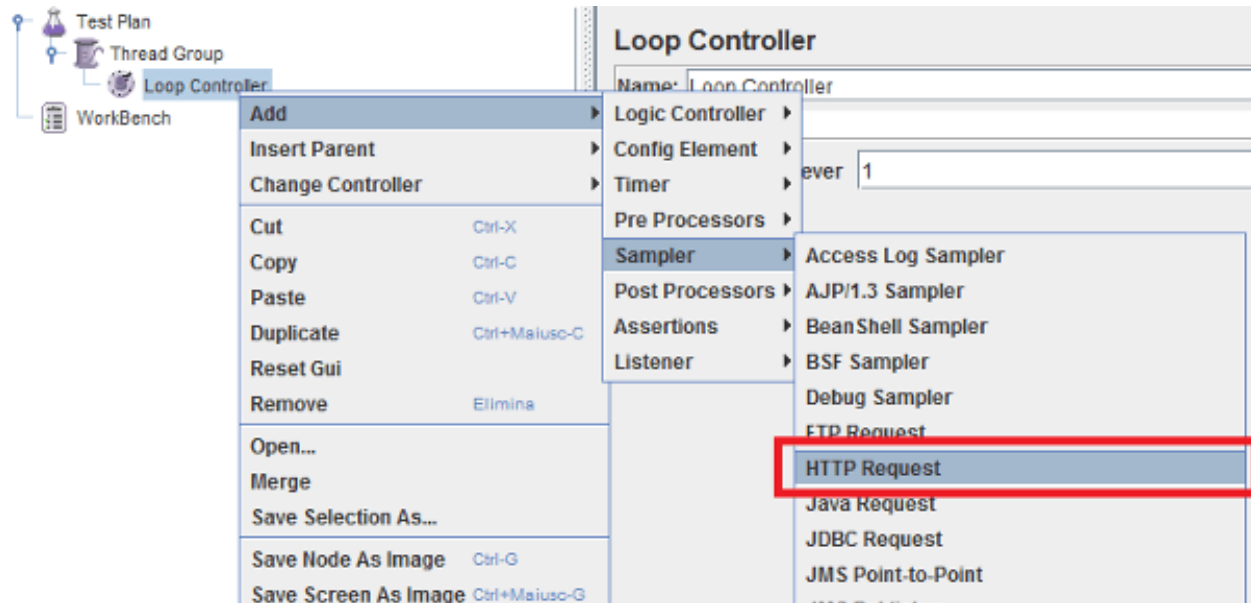


Fig. 301: Setting the HTTP Request

- In each **HTTP Request** add the following fields to the panel:

Name	Value	Encode?	Include Equals?
bbox	\${bbox}	unchecked	checked
height	\${height}	unchecked	checked
width	\${width}	unchecked	checked

Which should look like in the picture

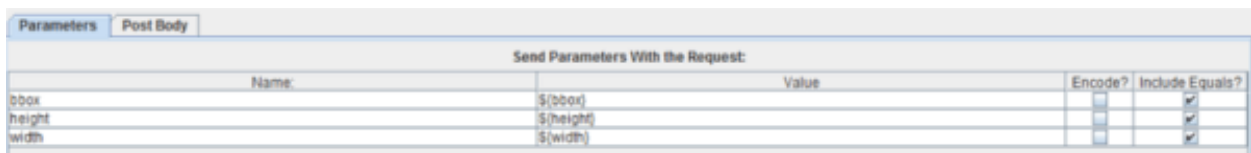


Fig. 302: HTTP Request panel configuration

- Uncheck the `Follow Redirects` and `Use KeepAlive` checkbox
- Right click on each `Loop Controller` tree node and add a new **CSV Data Set Config** element:
- Configure the `CSV Data Set Config` element by adding the **path** of the CSV file created before and setting the variable definitions:
- From the `Test Plan` tree node add an **HTTP Request Default** element and enter the following basic configuration:

Field	Value
Server Name or IP	localhost
Port Number	8083
Path	geoserver/ows

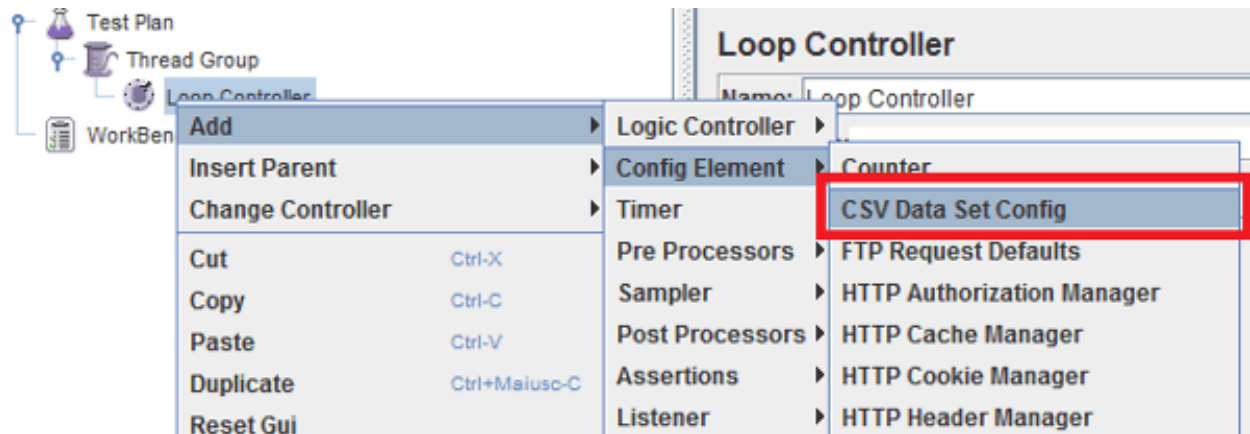


Fig. 303: Setting the CSV Data Set Config

CSV Data Set Config

Name: CSV Data Set Config

Comments:

Configure the CSV Data Source

Filename:

File encoding:

Variable Names (comma-delimited): width,height,bbox

Delimiter (use 't' for tab): ;

Allow quoted data?: False

Recycle on EOF?: True

Stop thread on EOF?: False

Sharing mode: All threads

Fig. 304: Configuring the CSV Data Set Config

It should look like this:

HTTP Request

Name: HTTP Request

Comments:

Web Server

Server Name or IP: localhost Port Number: 8083

HTTP Request

Implementation: [dropdown] Protocol [http]: [text] Method: GET [dropdown] Content encoding: [text]

Path: geoserver/ows

Fig. 305: *HTTP Default Request host/port/path configuration*

13. From the training data root directory, open the `data/jmeter_data/jmeter_request_params_2.txt`, select and copy its contents in the clipboard, then click “Add from Clipboard” in the “HTTP request” panel to setup a sample GetMap request:
14. Add the following listeners by right clicking on `Test Plan` tree node: “View results Tree”, “Summary Report”
15. Add the following assertions by right clicking on `Test Plan` tree node: “Response Assertion”

Note: Using `Assertions` is helpful because it avoids to continuously do a visual check on the results.

16. Configure the “Response Assertion” following this table:

Field	Value
Apply to	Main sample only
Response field to test	Response Headers
Pattern Matching Rules	Contains

In the `Pattern to test` panel add:

Content-Type: image/png

The final result should look like in the picture:

At this point JMeter is configured to run a GeoServer performance test:

1. Select the **Test Plan** tree node and select `Run - Start` from the top menu to start the JMeter test.
2. Select `View Results Tree` to directly see the request information produced and the requests results:
3. Select `Summary report` to view the statistical information about the requests:

Configuring JMeter for testing Raster optimization

The following section explains how the GeoServer performances improves with the optimization of raster files.

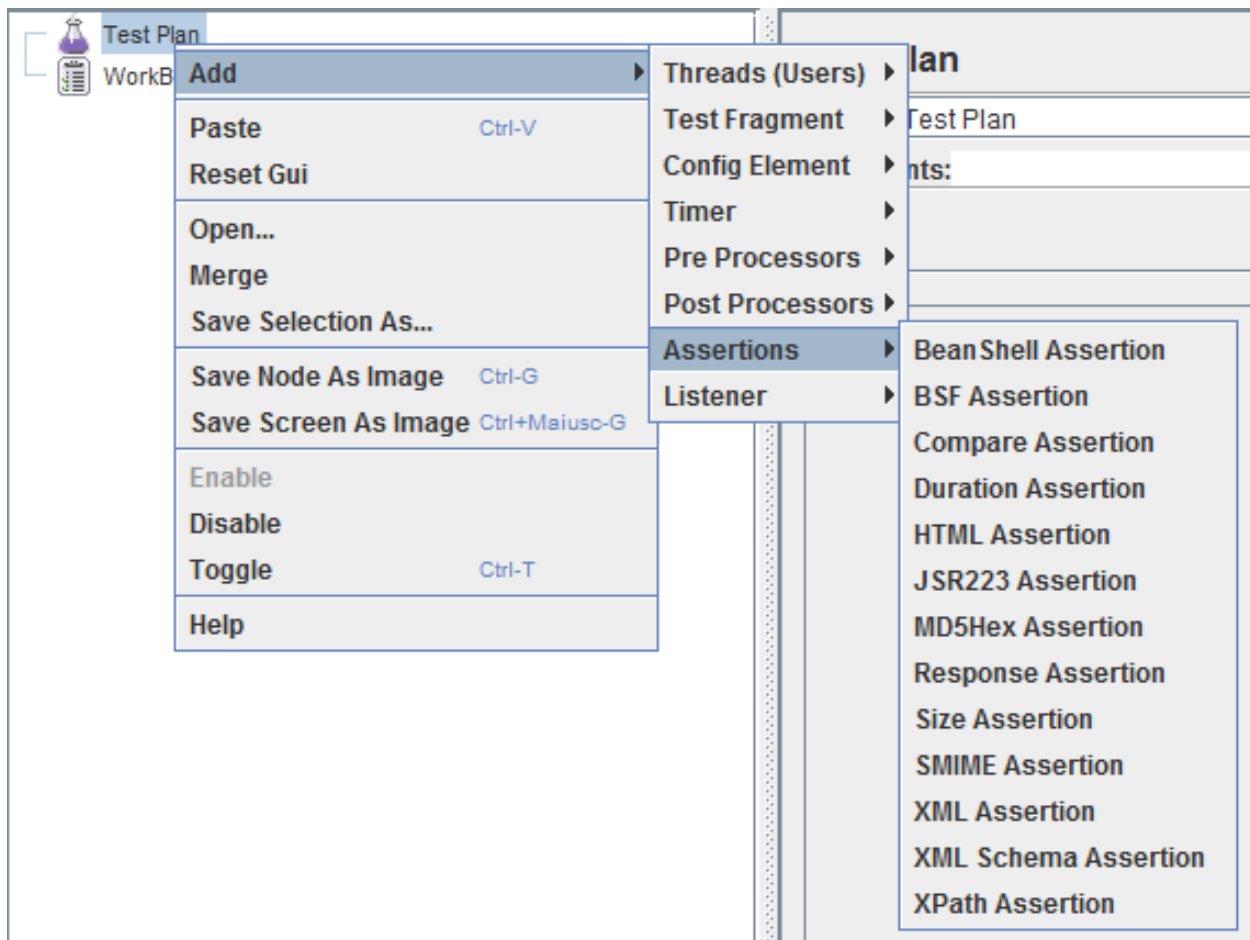


Fig. 306: Adding Assertions

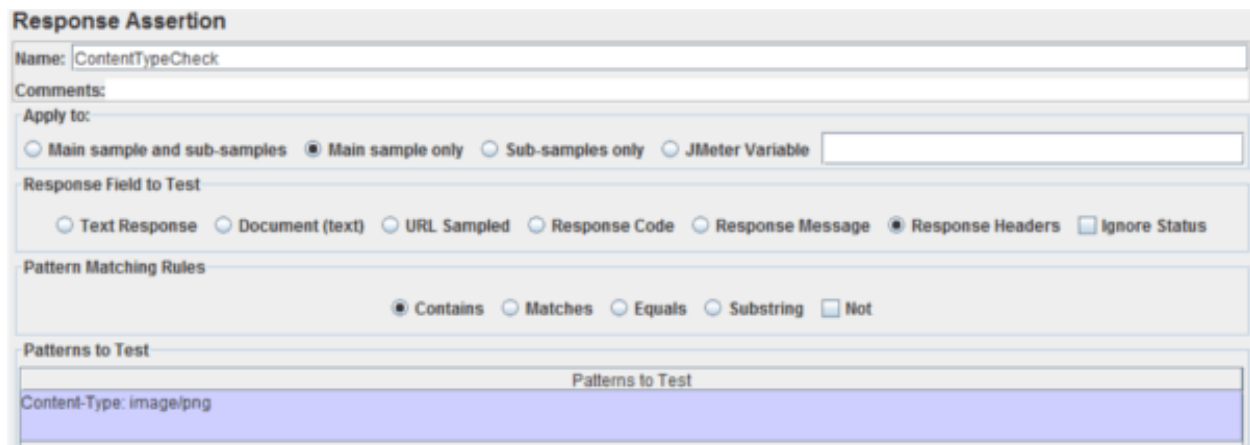


Fig. 307: Configuring Response Assertion

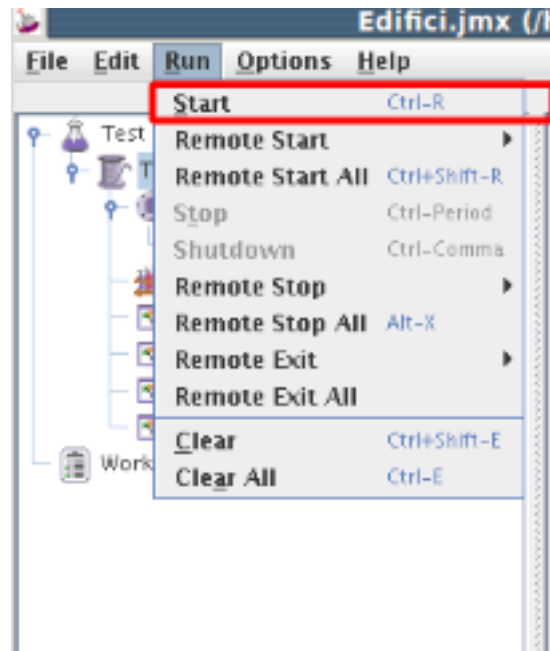


Fig. 308: starting jMeter test

Optimization has already been discussed in the *Introduction To Processing With GDAL* sections, describing the most common techniques used.

Note: This section requires the layers published in the *Adding an Image Mosaic*, *Introduction To Processing With GDAL* and *Advanced Mosaics Configuration* sections.

Test the Unoptimized Mosaic

1. Go to `$TRAINING_ROOT/data/jmeter_data` and copy the file `template.jmx` file, creating a `mosaic.jmx` file
2. From the training root, on the command line, run `jmeter.bat` (or `jmeter.sh` if you're on Linux) to start JMeter
3. On the top left go to *File* → *Open* and search for the new *jmx* file copied
4. Disable the Thread Groups **8**, **16**, **32**, **64** by right-clicking on them and selecting *Disable*.
5. In the *CSV Data Set Config* element of the remaining thread groups, modify the **path** of the CSV file by setting the path for the file `optimized.csv` in the `$TRAINING_ROOT/data/jmeter_data` directory
6. In the **HTTP Request Default** element modify the following parameters:

Name	Value
layers	geosolutions:boulder_bg
srs	EPSG:26913

At this point JMeter is configured to run a GeoServer performance test:

1. Run the test

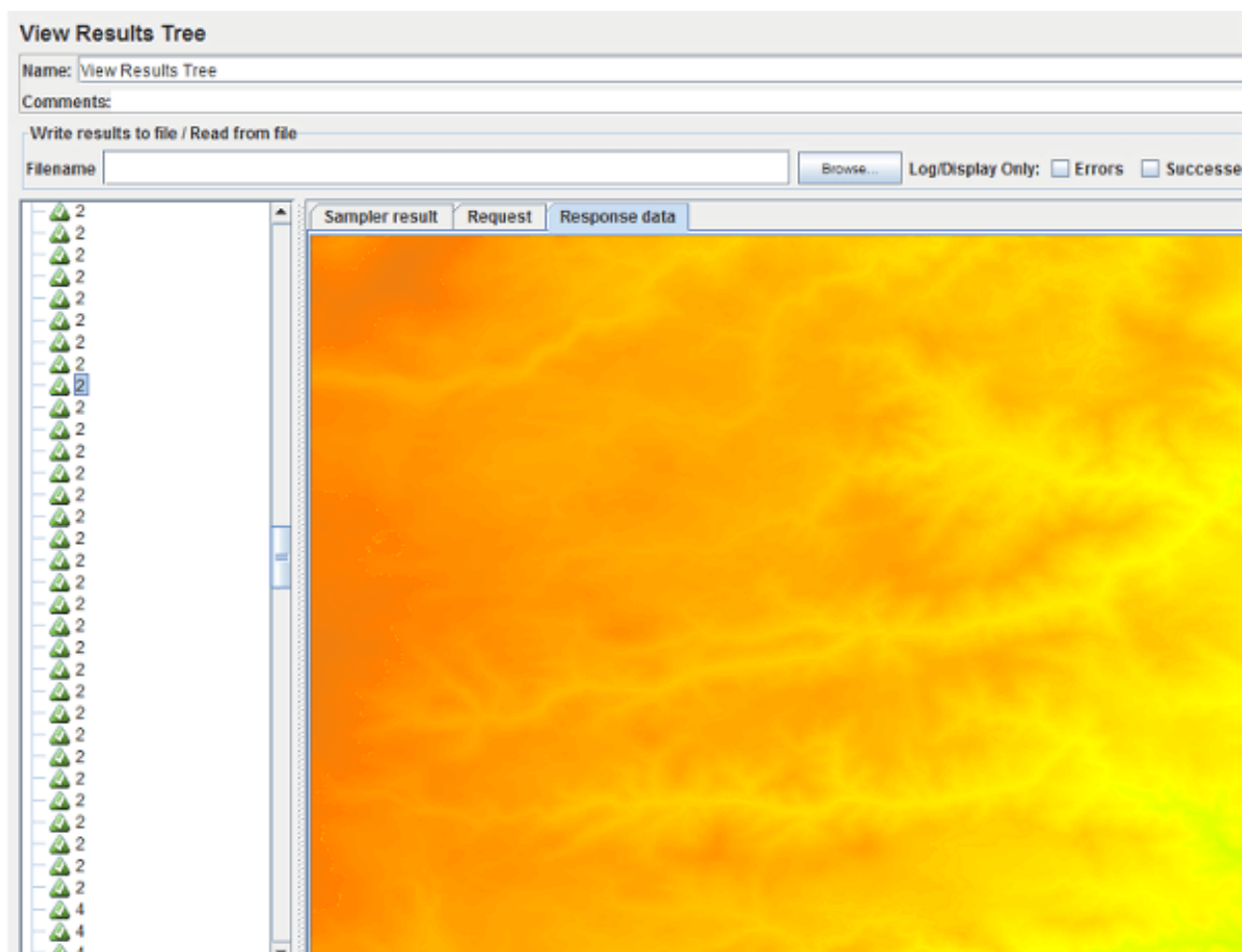


Fig. 309: The View Results Tree panel with a sample request

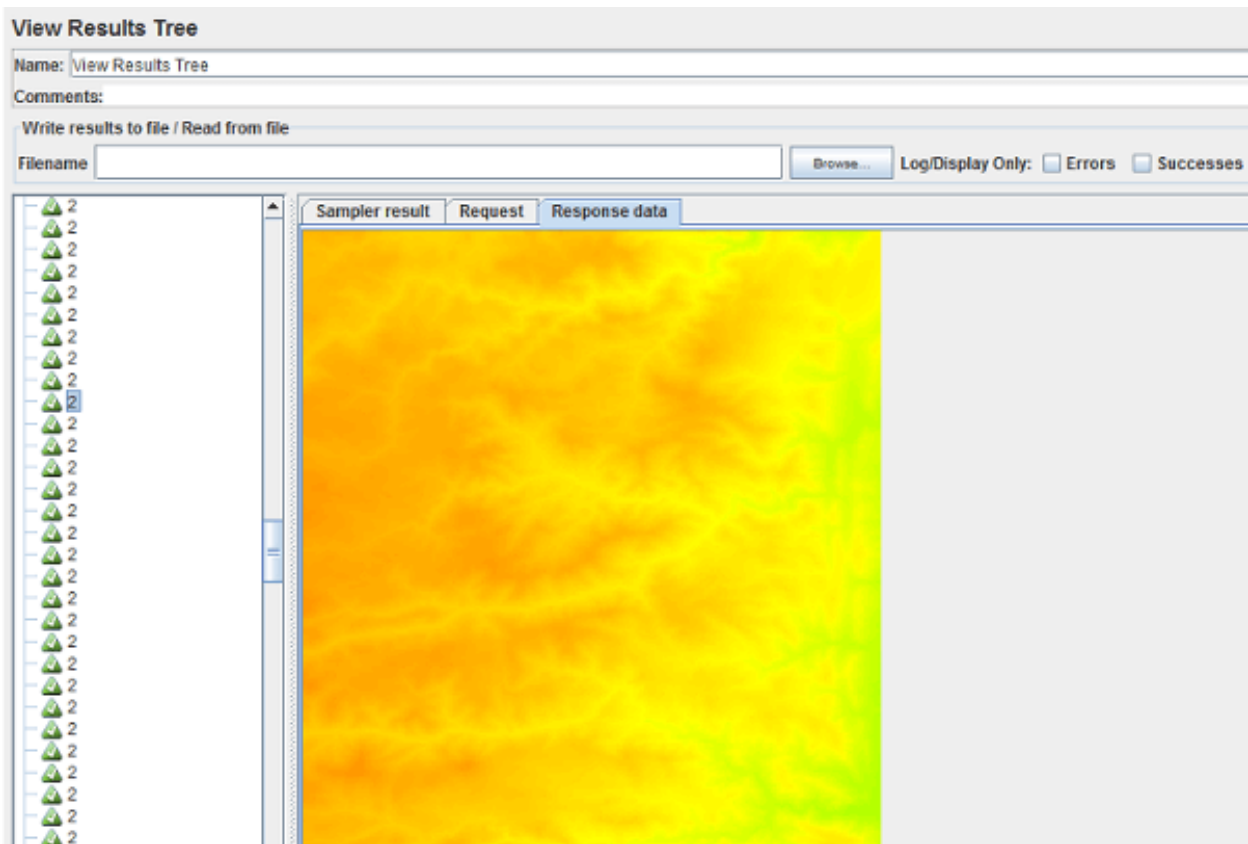


Fig. 310: Another request with different resolution and bounding box

Label	# Samples	Average	Min	
1	100	126	35	
2	100	102	33	
4	200	140	35	
TOTAL	400	127	33	

Fig. 311: Summary report panel

Note: Remember to run and stop the test a few times for having stable results

2. You should see something like this:

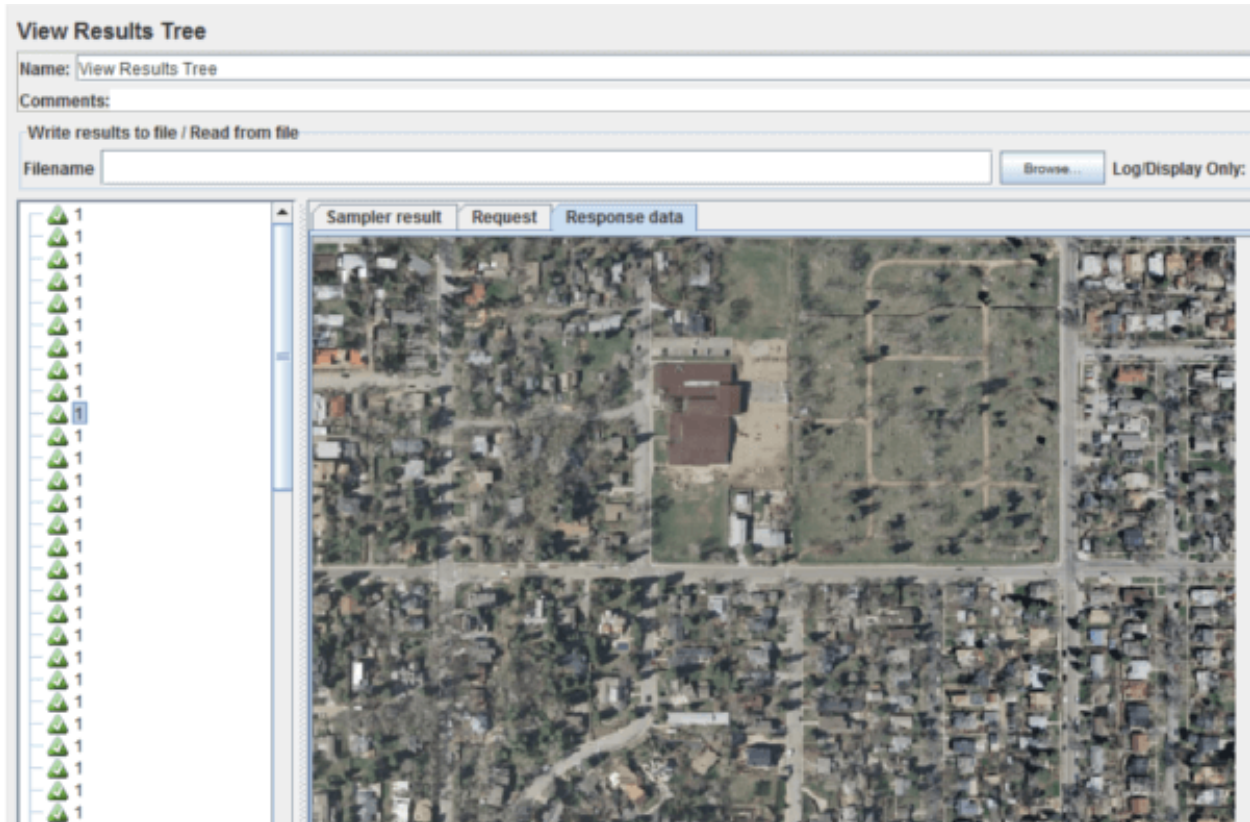


Fig. 312: ‘View Results Tree’ panel

3. When the test is completed, Save the results in a text file and remove them from the console by clicking on *Run* → *Clear All* on the menu

Test the Optimized Mosaic

1. In the **HTTP Request Default** section modify the following parameter:

Name	Value
layers	geosolutions:boulder_bg_optimized

2. Run the test again
3. Compare the results of this test with the ones saved before. **You should see that throughput is increased with the optimized Mosaic**

Configuring JMeter for testing Vector data

The following section compares vector data preparation using Shapefile and PostGIS. For this example a Shapefile containing primary or secondary roads is used.

The purpose is to test the throughput between the shapefile and an optimized database containing the same data. The result will demonstrate that database optimization can provide a better throughput than the one of the shapefile

Configuring the database

1. Open the terminal and go to the %TRAINING_ROOT%
2. Load the shapefile `tl_2014_01_prisecroads` located in `%TRAINING_ROOT%\data\user_data` into PostGIS with the following commands:

```
setenv.bat

createdb -U postgres -T postgis20 shape2

shp2pgsql -k -I "data\user_data\tl_2014_01_prisecroads\tl_2014_01_
→prisecroads.shp" public.pgroads | psql -U postgres -d shape2
```

Note: More information can be found at [Loading a Shapefile into PostGIS](#)

3. On the %TRAINING_ROOT% run **pgAdmin.bat**
4. Go to the table `pgroads` inside database `shape2` and execute the following SQL script for creating an index on the *MTFCC* column:

```
CREATE INDEX mtfcc_idx ON pgroads ("MTFCC");
```

The following index optimizes the access to the database when filtering on the *MTFCC* column.

Configuring GeoServer

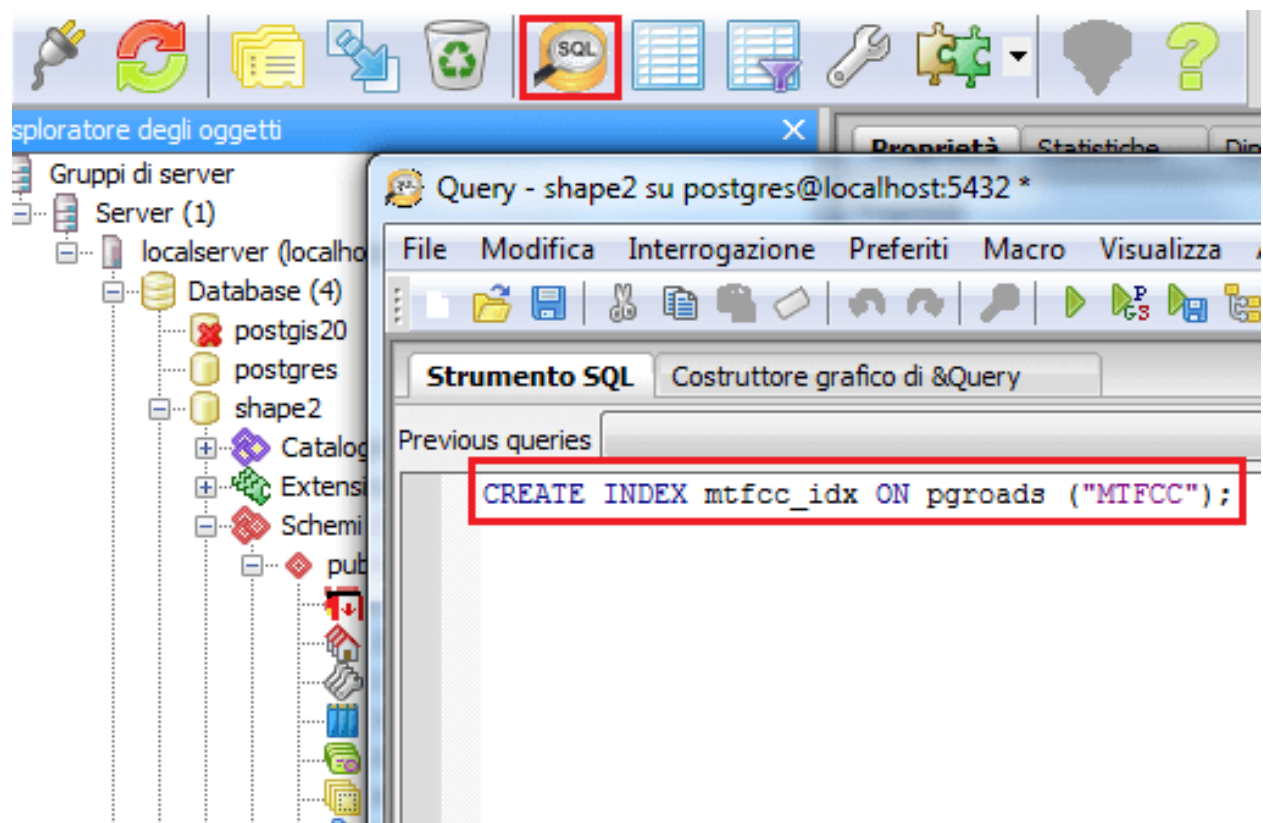
1. On your Web browser, navigate to the GeoServer [Welcome Page](#).
2. Following the instructions on [Adding a Postgis layer](#), configure the database `shape2` in GeoServer and call it **pgroads**

Note: Note that the database *Coordinate Reference System* is EPSG:4269

3. Configure the shapefile `tl_2014_01_prisecroads` used before in GeoServer following the instructions in [Adding a Shapefile](#) and call it **allroads**

Note: Note that the shapefile *Coordinate Reference System* is EPSG:4269

4. Go to *Styles* and click on *Add new Style*
5. On the bottom of the page, click on *Choose File* and select the SLD file called `shproads` in the `%TRAINING_ROOT%/data/jmeter_data` directory

Fig. 313: *Create a new index*

- Click on *Upload* and then on *Submit*. This new style supports scale dependency which is used as filter on the roads to display.

Configuring JMeter

- Go to `$TRAINING_ROOT/data/jmeter_data` and copy the file `template.jmx` file creating a `vector.jmx` file.
- From the training root, on the command line, run `jmeter.bat` (or `jmeter.sh` if you're on Linux) to start JMeter
- On the top left go to *File* → *Open* and search for the new *jmx* file copied
- In the *CSV Data Set Config* element, modify the **path** of the CSV file by setting the path for the file `shp2pg.csv` in the `$TRAINING_ROOT/data/jmeter_data` directory
- In the **HTTP Request Default** element modify the following parameters:

Name	Value
layers	geosolutions:allroads
srs	EPSG:4269
styles	shproads

Test the Shapefile

- Run the test. You should see something like this:

Note: Remember to run and stop the test a few times for having stable results

- When the test is completed, Save the results in a text file.
- Remove the result from JMeter by clicking on *Run* → *Clear All* on the menu

Test the Database

- In the **HTTP Request Default** element modify the following parameter:

Name	Value
layers	geosolutions:pgroads

- Run the test again. It should be noted that database throughput is greater than that of the Shapefile, because the new index created provides a faster access on the database, improving GeoServer performances

Configuring JMeter for testing Style optimization

The following section explains how GeoServer performances are improved when using optimized styles. Styling is an important feature for GeoServer, but requires some attention in order to avoid slowing down the performances.

This tutorial is aimed to show how GeoServer performances change by choosing a different style for the same data set using JMeter.

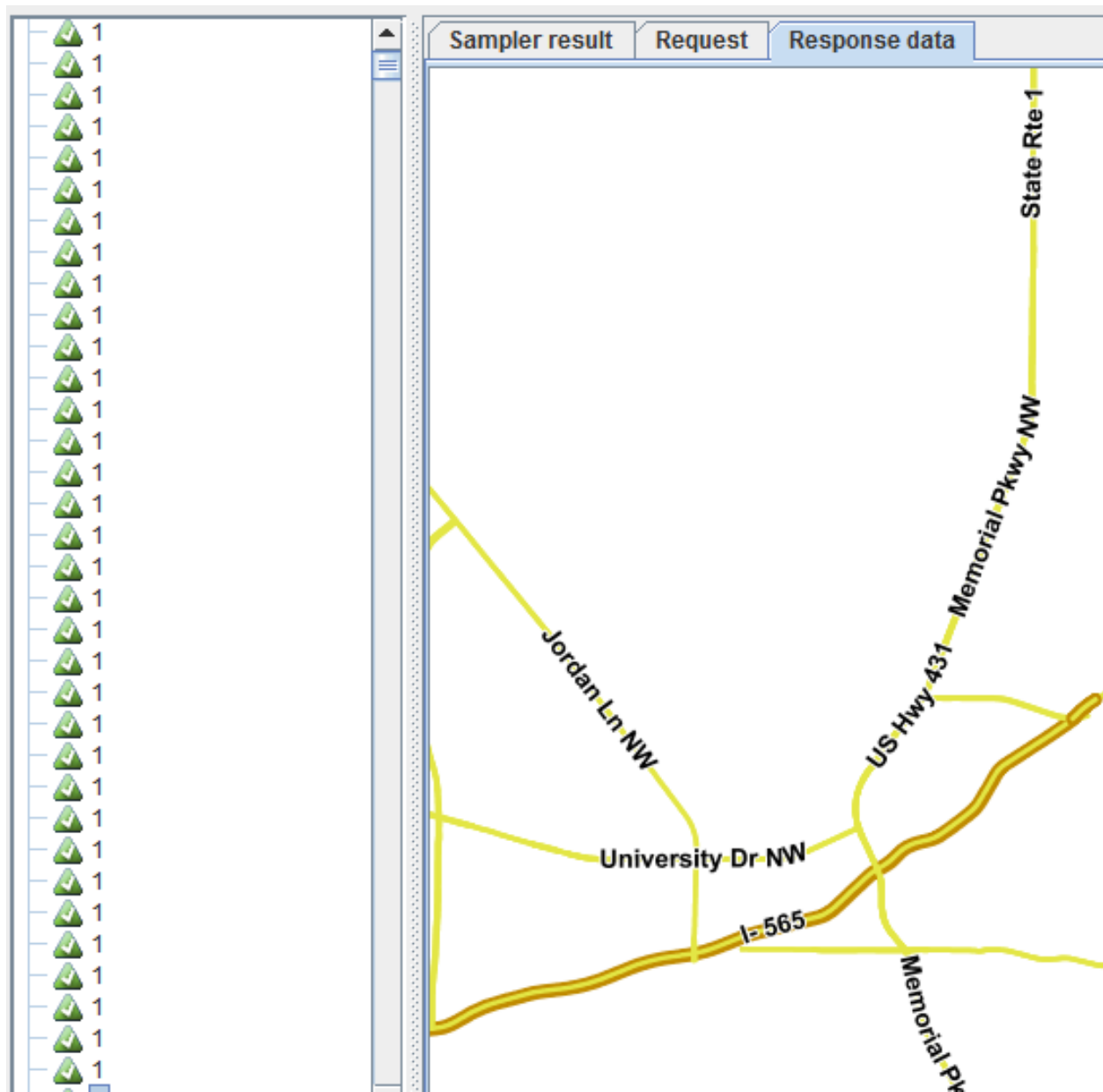


Fig. 314: Sample request on the Shapefile

Note: This example requires to have already completed the first 9 steps of the *Creating a Base Map with a Layer Group* section, *Adding a Shapefile* and *Adding a Style* sections .

Configuring GeoServer

1. On your Web browser, navigate to the GeoServer [Welcome Page](#).
2. Go to *Styles* and click on *Add new Style*
3. On the bottom of the page, click on *Choose File* and select the SLD file called `line_label` in the `$TRAINING_ROOT/data/jmeter_data` directory
4. Click on *Upload* and then on *Submit*. Now we have a style which supports labeling but has no control on the label conflicts and overlapping
5. Return to the GeoServer [Welcome Page](#).
6. Go to *Layer Groups* and click on `test`
7. Add a new Layer to the Layer Group called **bbuildings**

Layers

+ Add Layer...
 + Add Layer Group...

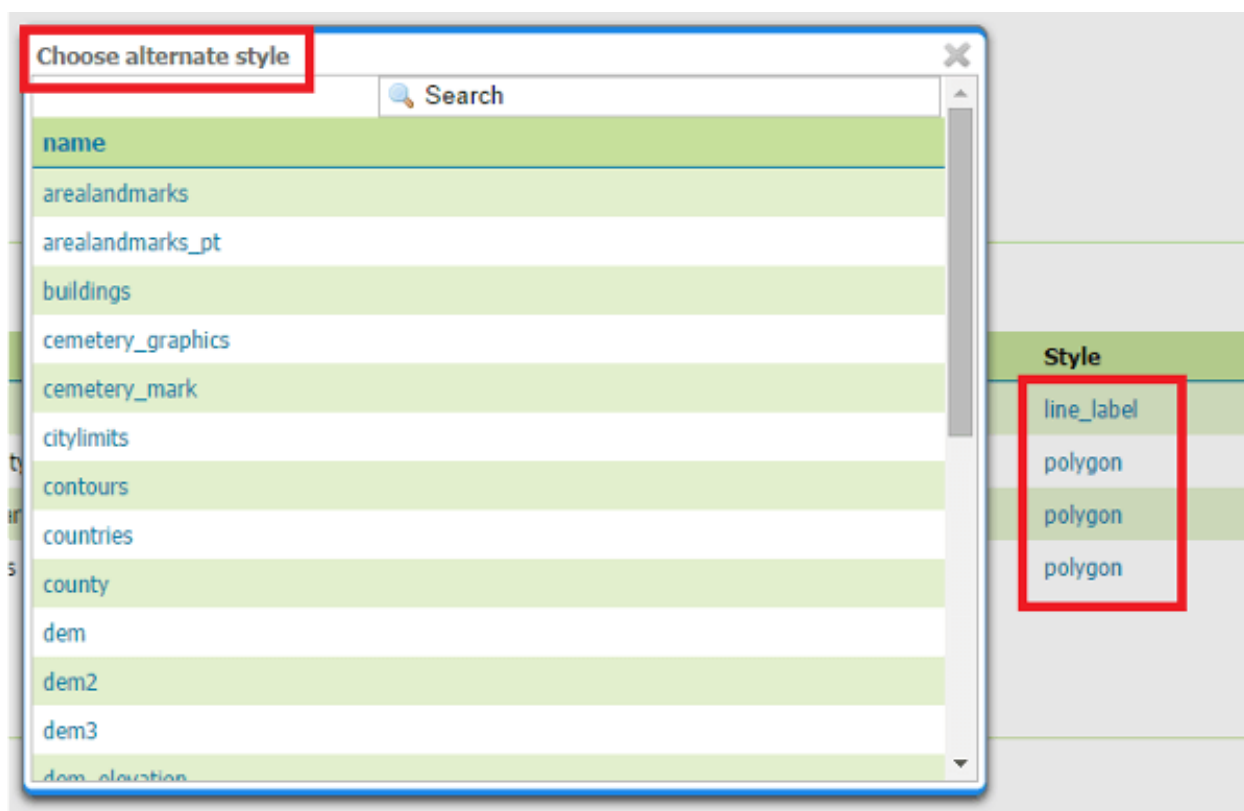
Drawing order	Layer
1 ↓	geosolutions:Mainrd
2 ↑ ↓	geosolutions:BoulderCityLimits
3 ↑ ↓	geosolutions:bplandmarks
4 ↑	geosolutions:bbuildings

<< < 1 > >> Results 1 to 4 (out of 4 items)

Fig. 315: Add a new Layer to the Layer Group

8. Change the associated styles by clicking on each style and choosing another one on the list. Use the following styles:

Layer	Style
geosolutions:Mainrd	line_label
geosolutions:BoulderCityLimits	polygon
geosolutions:bplandmarks	polygon
geosolutions:bbuildings	polygon

Fig. 316: *Styles configuration*

- Click on *Save*. With this configuration we have a Layer Group composed by 4 Layers with 4 bad styles associated. This will result in a low throughput, if compared to that of the test with optimized styles.

Configuring JMeter

- Go to `$TRAINING_ROOT/data/jmeter_data` and copy the file `template.jmx` file and create a `styles.jmx` one
- From the training root, on the command line, run `jmeter.bat` (or `jmeter.sh` if you're on Linux) to start JMeter
- On the top left go to *File* → *Open* and search for the new *jmx* file copied
- Disable **Thread Group 8, 16, 32 and 64**
- In the **CSV Data Set Config** element, modify the **path** of the CSV file by setting the path for the file `style.csv` in the `$TRAINING_ROOT/data/jmeter_data` directory
- In the **HTTP Request Default** element modify the following parameters:

Name	Value
layers	test
srs	EPSG:2876

Test with unoptimized styles

- Run the test. You should see something like this:

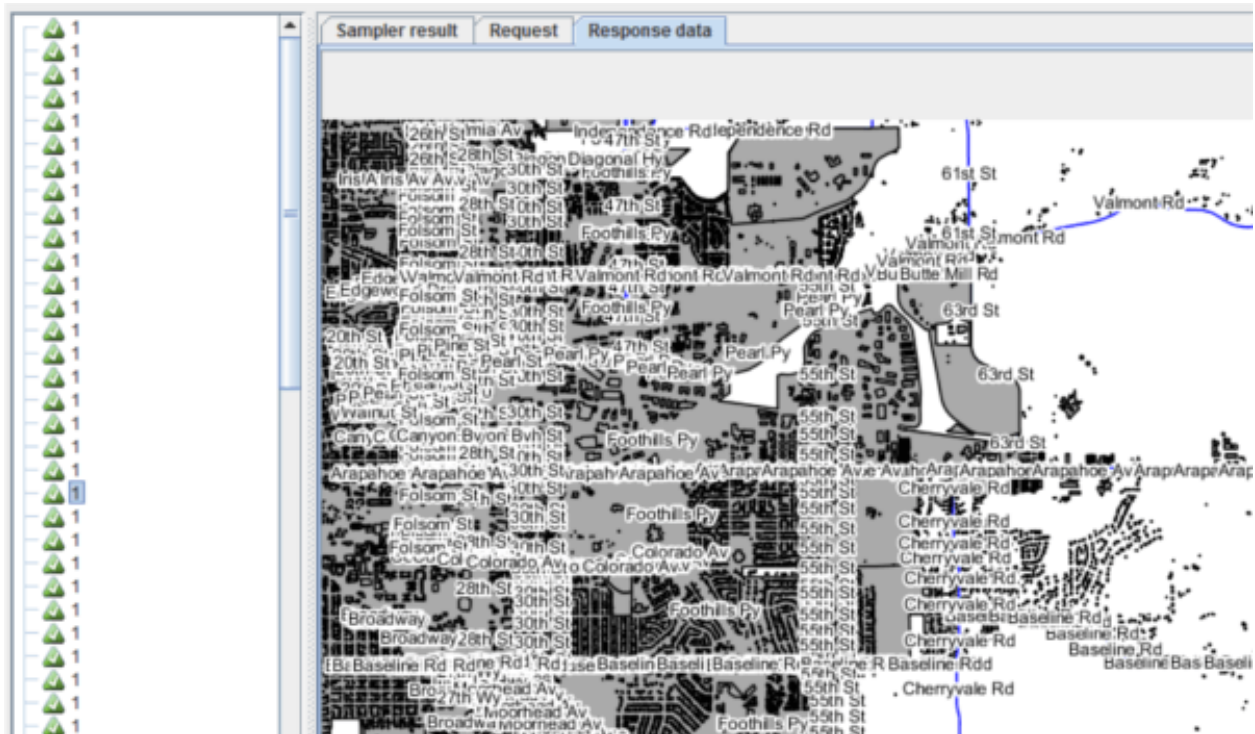


Fig. 317: View Results Tree panel with a bad styling

Note: Remember to run and stop the test a few times for having stable results

2. When the test is completed, Save the results in a text file.
3. Remove the result from JMeter by clicking on *Run* -> *Clear All* on the menu

Setting optimized styles

1. Go to *Layer Groups* and click on `test`
2. Change the associated styles by clicking on each style and choosing another one on the list. Use the following styles:

Layer	Style
geosolutions:Mainrd	mainrd
geosolutions:BoulderCityLimits	citylimits
geosolutions:bplandmarks	arealandmarks
geosolutions:bbuildings	buildings

Layer	Default Style	Style
geosolutions:Mainrd	<input type="checkbox"/>	mainrd
geosolutions:BoulderCityLimits	<input type="checkbox"/>	citylimits
geosolutions:bplandmarks	<input type="checkbox"/>	arealandmarks
geosolutions:bbuildings	<input type="checkbox"/>	buildings

Fig. 318: *Styles configuration*

3. Click on *Save*. The new styles contain scale dependencies and label optimization, which will result in a better throughput.

Test with optimized styles

1. Run again the test.

You may see that the throughput is greater than that of the first test. The use of scale dependencies reduces the layers to see at lower zoom levels while conflict resolution avoids to show multiple overlapping label at each zoom level.

Configuring JMeter for testing GeoWebCache fullWMS support

The following section compare GeoServer WMS with GeoWebCache `fullWMS` support. `FullWMS` is a new feature which allows GeoWebCache to act as a WMS endpoint, like GeoServer. Using GeoWebCache, the server is able to cache the requested tiles in order to return them faster then GeoServer.

This example will show how to configure GeoWebCache with full WMS support and how to improve performance.

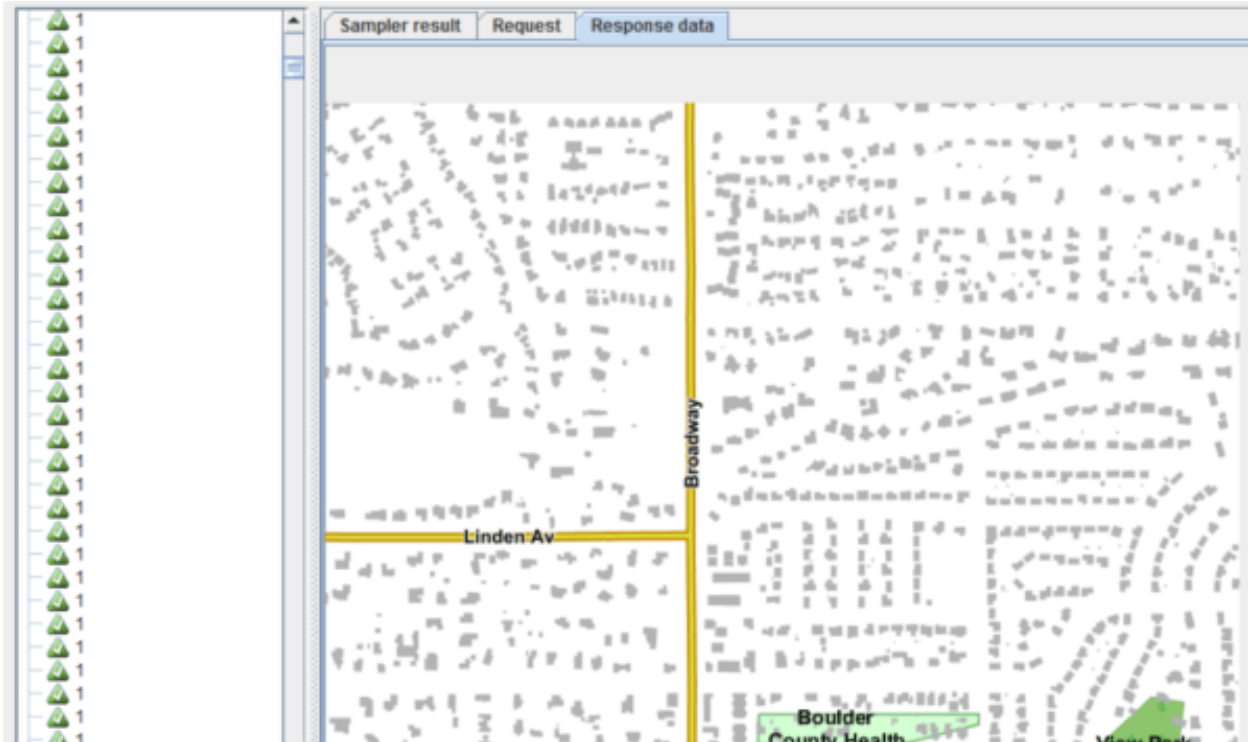


Fig. 319: View Results Tree panel with good styling

Configuring GeoServer/GeoWebCache

1. On your Web browser, navigate to the GeoServer [Welcome Page](#).
2. Go to *Gridsets* and click on *Create a new gridset*
3. Call it `EPSG_2876` and set `EPSG:2876` as *Coordinate Reference System*
4. Click on *Compute from maximum extent of CRS* and add 15 new **Zoom Levels** (from 0 to 14) by clicking on *Add zoom level*. It should look like this picture:
5. Click on *Save*. Now this *GridSet* can be added to the Layer Group *boulder* for caching it with GeoWebCache
6. Go to *Layer Groups* and click on *boulder*
7. On the *Available gridsets* panel add the gridset `EPSG_2876` from the list. Then click on *Save*.

Note: Remember to set *Published zoom levels* to Min and Max

Configuring JMeter

1. Go to `$TRAINING_ROOT/data/jmeter_data` and copy the file `template.jmx` into `gwc.jmx`
2. From the training root, on the command line, run `jmeter.bat` (or `jmeter.sh` if you're on Linux) to start JMeter
3. On the top left go to *File* → *Open* and search for the new *jmx* file copied

Name *

Description

Coordinate Reference System
 Find... [EPSG:NAD83\(HARN\) / Colorado North \(ftUS\)...](#)
 Units: foot_survey_us
 Meters per unit: 0.30480060960121924

Gridset bounds

Min X	Min Y	Max X	Max Y
1.999.305,528579	1.082.572,159097	3.972.520,326056	1.626.770,085995

[Compute from maximum extent of CRS](#)

Tile width in pixels *

Tile height in pixels *

Tile Matrix Set

Define grids based on: ☒ Resolutions ☐ Scale denominators

Level	Pixel Size	Scale
0	<input type="text" value="2.125,7731519436893"/>	1: <input type="text" value="2.314.060,5449512205"/>
1	<input type="text" value="1.062,8865759718446"/>	1: <input type="text" value="1.157.030,2724756103"/>
2	<input type="text" value="531,4432879859223"/>	1: <input type="text" value="578.515,1362378051"/>
3	<input type="text" value="265,72164399296116"/>	1: <input type="text" value="289.257,56811890256"/>
4	<input type="text" value="132,86082199648058"/>	1: <input type="text" value="144.628,78405945128"/>
5	<input type="text" value="66,43041099824029"/>	1: <input type="text" value="72.314,39202972564"/>
6	<input type="text" value="33,215205499120145"/>	1: <input type="text" value="36.157,19601486282"/>

Fig. 320: Create a new Gridset

Available gridsets

Gridset	Published zoom levels
EPSG:4326	Min ▼ / Max ▼
EPSG:900913	Min ▼ / Max ▼
EPSG_2876	Min ▼ / Max ▼


Add grid subset: Scegliere uno ▼ 

Fig. 321: Add the new Gridset

4. Disable all the **Thread Groups** except for **8**
5. Disable the **Content-Type Check**
6. In the **CSV Data Set Config** element, modify the **path** of the CSV file by setting the path for the file `gwc.csv` in the `$TRAINING_ROOT/data/jmeter_data` directory
7. In the **HTTP Request Default** element modify the following parameters:

Name	Value
layers	boulder
srs	EPSG:2876

Test GeoServer WMS

1. Run the test

Note: Remember to run and stop the test a few times for having stable results

2. When the test is completed, Save the results in a text file.
3. Remove the result from JMeter by clicking on *Run -> Clear All* on the menu
4. Stop GeoServer

Test GeoWebCache fullWMS

1. Go to `$TRAINING_ROOT/data/gwc/geowebcache.xml` and add the following snippet:

```
<gwcConfiguration>
...
  <fullWMS>true</fullWMS>
</gwcConfiguration>
```

Setting `fullWMS` to `true` allows GeoWebCache to use fullWMS support

2. Restart GeoServer
3. On the JMeter **HTTP Request Default** panel, change the *Path* from `geoserver/ows` to `geoserver/gwc/service/wms` in order to execute WMS requests directly to GeoWebCache, without passing from GeoServer
4. Add a new parameter called **hints** which can have 3 values `speed`, `default` and `quality`. The first one should be used for having a faster response without concerning about image quality; the last one, instead, is slower but with a better quality; the second one is a good trade off between them. For the first test set **hints** to `speed`.
5. Run the test

Note: At the first run, the throughput should be lower than that of GeoServer, because GeoWebCache has spent much time on generating the cached tiles.

6. Remove the result from JMeter by clicking on *Run -> Clear All* on the menu
7. Run the same test again.

Now the throughput should be improved, because GeoWebCache have already generated the tiles to cache and can reuse them. Image quality should be very poor because of the `hints=speed` configuration.

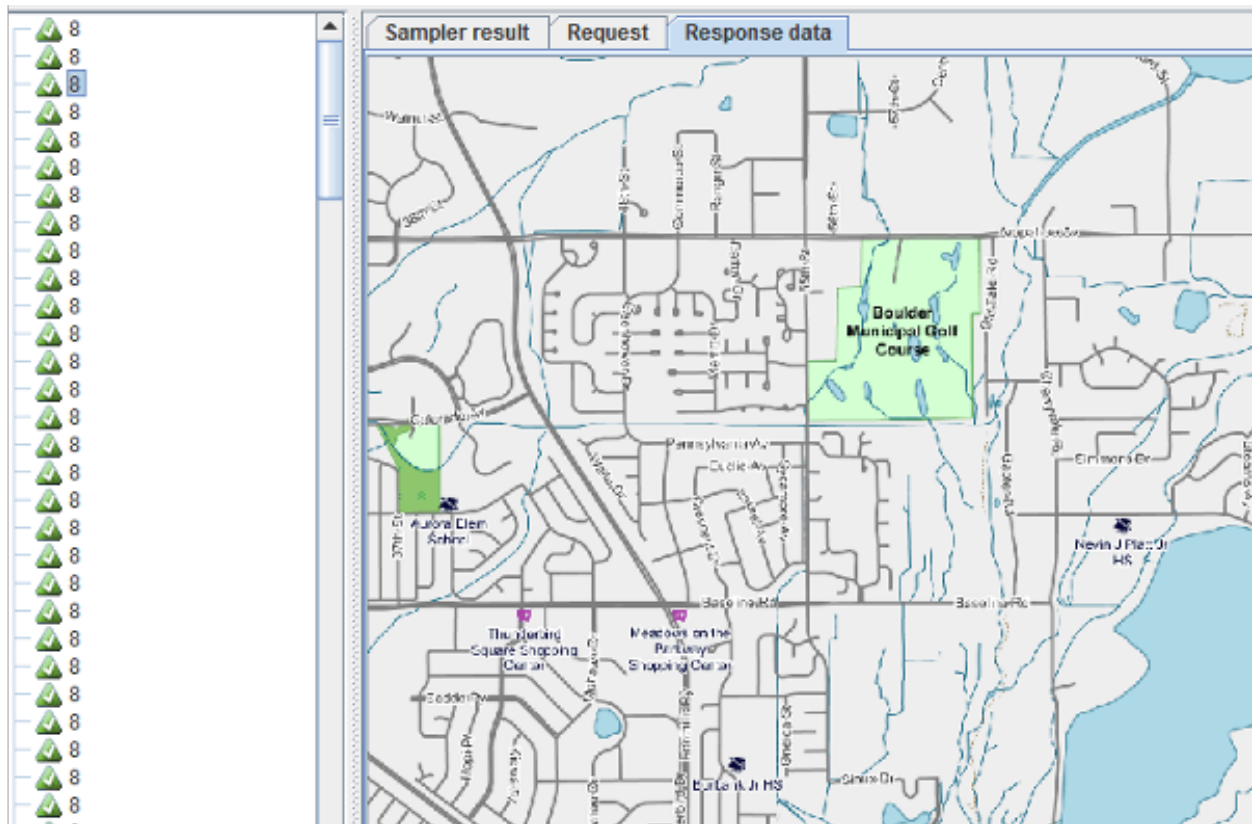


Fig. 322: Result from GeoWebCache fullWMS with `hints=speed`

8. Run the same test with `hints=default`

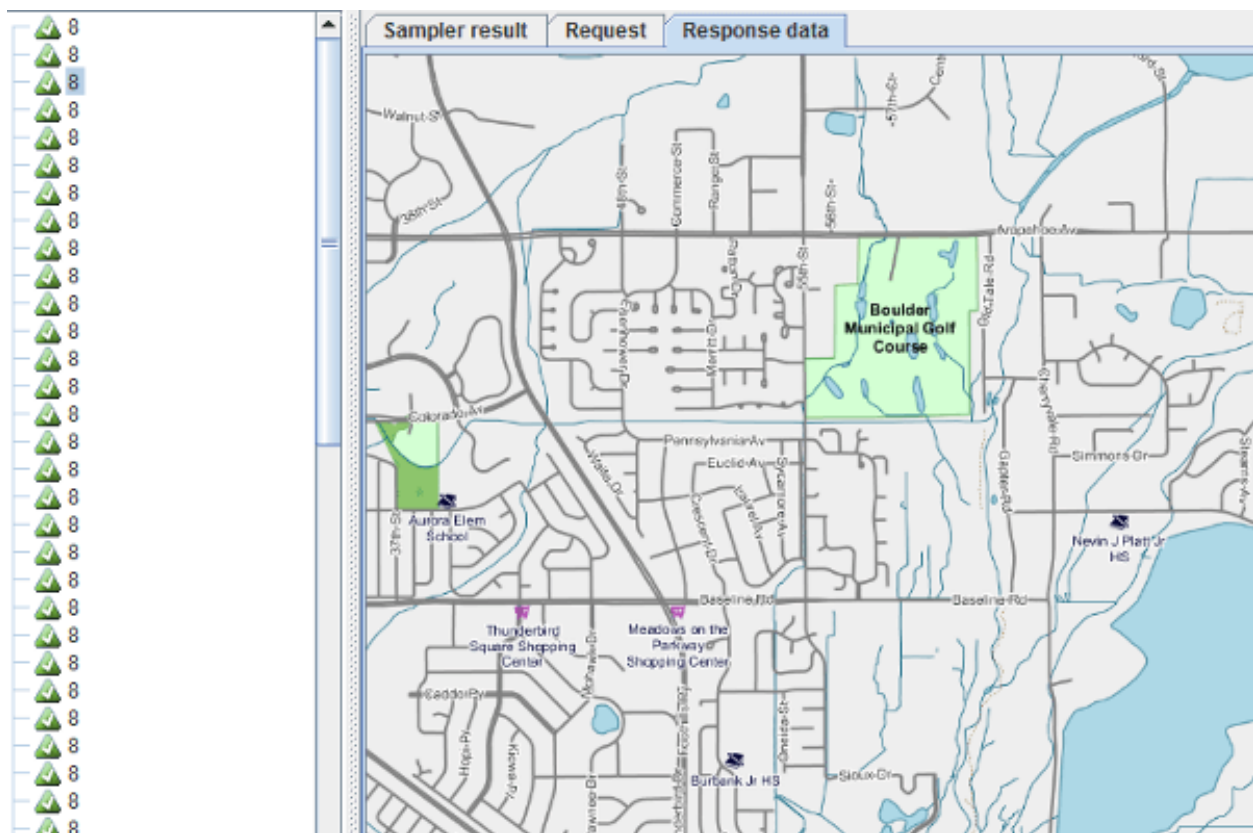


Fig. 323: Result from GeoWebCache fullWMS with hints=default

- Run the same test with `hints=quality`

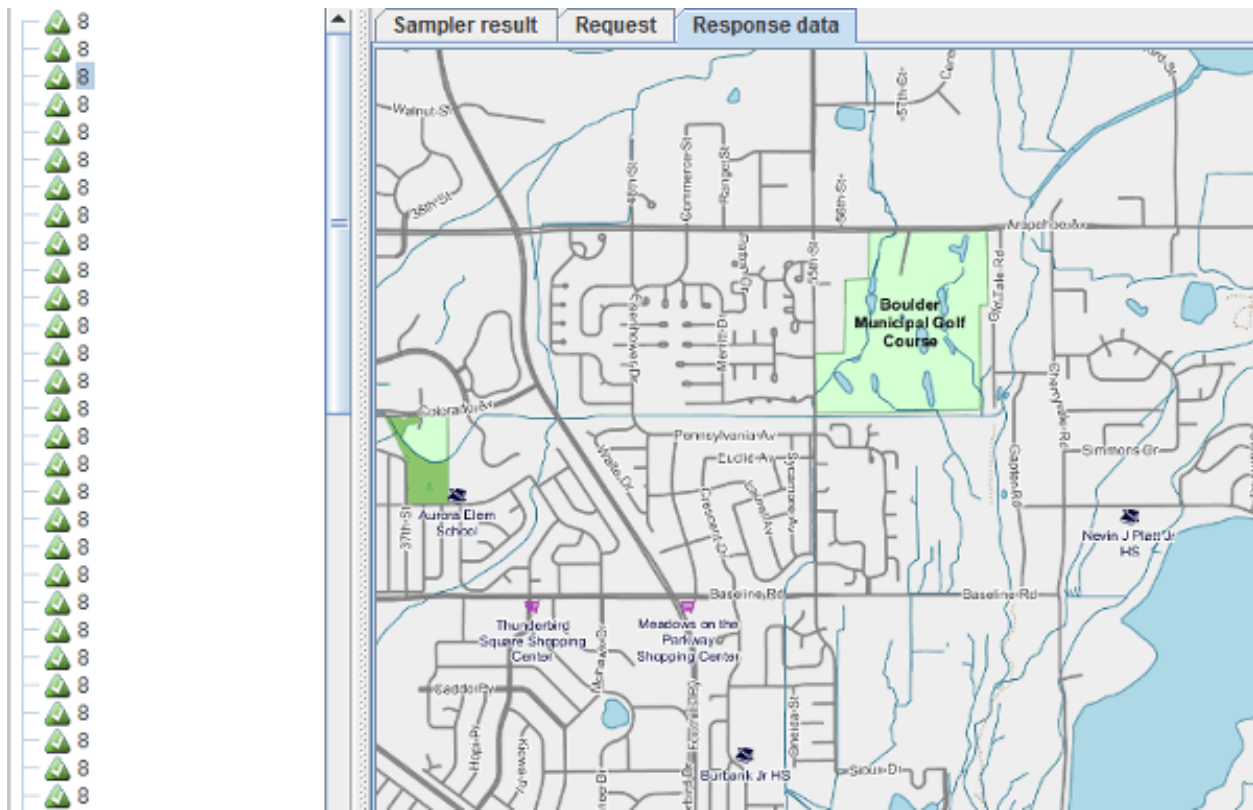


Fig. 324: Result from GeoWebCache fullWMS with `hints=quality`

It should be noted that changing the `hints` parameter changes the image quality, but the throughput is always greater than that of GeoServer WMS

Configuring JMeter for testing WMS Resource Limits

The following section explains how GeoServer performances are improved when setting the resource limits for WMS.

Preliminary Steps

- Open your Web browser and navigate to the GeoServer [Welcome Page](#).
- Go to *Stores* and select the `storms` datastore
- Change the following parameters:

Name	Value
max connections	1
Connection timeout	20000000

It should appear something like this:

Connection Parameters

host *

port *

database

schema

user *

passwd

Namespace *

<http://www.geo-solutions.it/workshop>

☐ Expose primary keys

max connections

min connections

fetch size

Connection timeout

Fig. 325: Change 'storms' parameters

Now you have configured this store to enqueue all the requests in a single queue until they are not timed out.

Configure JMeter

1. Go to `$TRAINING_ROOT/data/jmeter_data` and copy the file `template.jmx` file and create `limit.jmx`
2. From the training root, on the command line, run `jmeter.bat` (or `jmeter.sh` if you're on Linux) to start JMeter
3. On the top left go to *File* → *Open* and search for the new *jmx* file copied
4. Disable all the **Thread Groups** except for the **64** one in order to create a test environment with multiple concurrent requests to be enqueued.
5. In the **CSV Data Set Config** element, modify the **path** of the CSV file by setting the path for the file `limits.csv` in the `$TRAINING_ROOT/data/jmeter_data` directory
6. In the **HTTP Request Default** element modify the following parameters:

Name	Value
layers	geosolutions:storm_obs
srs	EPSG:4326

Test without WMS Limits

1. Run the test

Note: Remember to run and stop the test a few times for having stable results

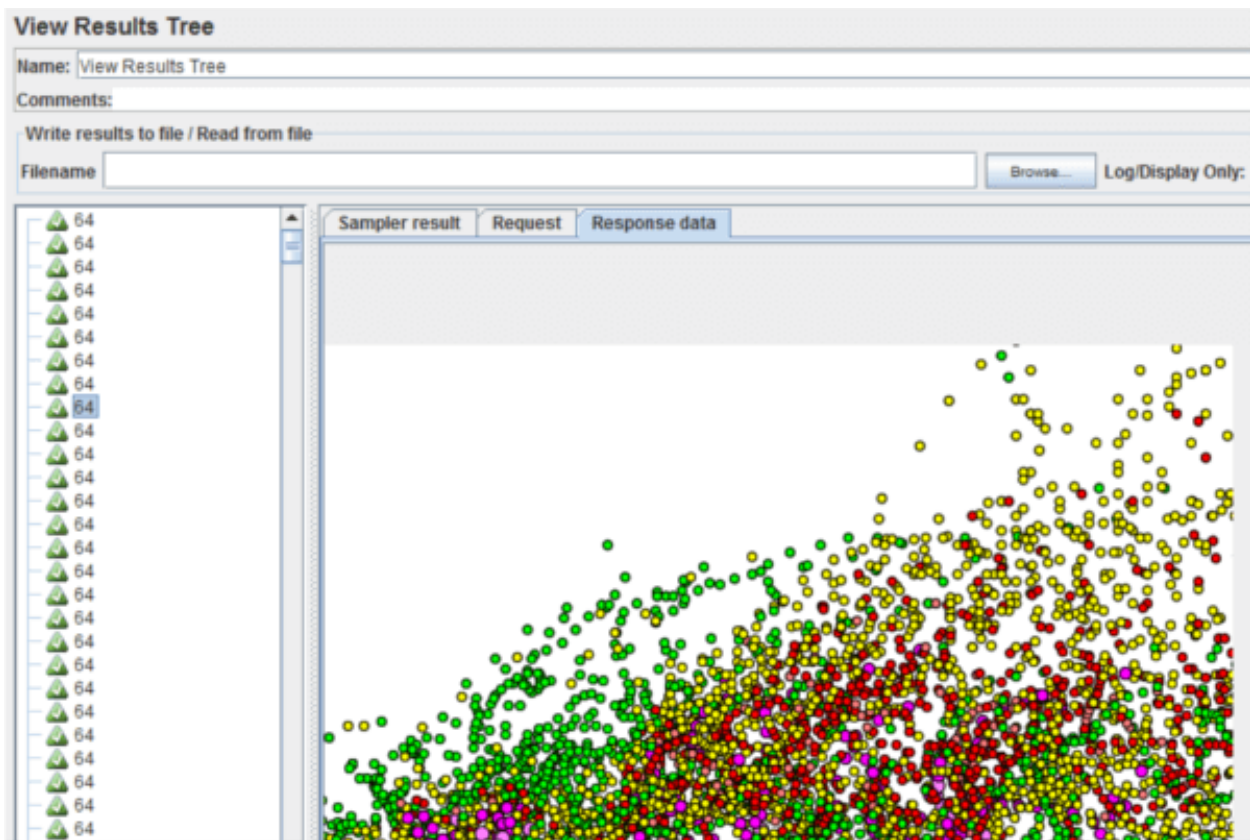
2. You should see something like this:
3. When the test is completed, Save the results in a text file and remove them from the console by clicking on *Run* → *Clear All* on the menu

Configure WMS Limits

1. On your Web browser, navigate to the GeoServer [Welcome Page](#).
2. Go to *WMS* and edit the *Raster Rendering Options* section:

Name	Value
Max rendering time	10

With this option, GeoServer will cut off all the requests that need more than 10s to be rendered, making GeoServer more responsive. Note that this will result in various error returned by GeoServer for those operations which are cut off. You can choose another value to set. For having a good result you should take a value minor than the average response time of the first test.

Fig. 326: *View Results Tree*

Raster Rendering Options

Default Interpolation

Nearest neighbor ▼

KML Options

Default Reflector Mode

refresh ▼

Default Superoverlay Mode

auto ▼

- ☒ Generate vector placemarks (KMATTR)
- ☐ Generate raster placemarks (kmlplacemark)

Raster/vector threshold (0-100, default 40)

40

Resource consumption limits

Max rendering memory (KB)

0

Max rendering time (s)

0

Max rendering errors (count)

0

Fig. 327: Changing WMS limit configuration

Test with WMS Limits

1. Run again the test. You should see multiple errors like this:

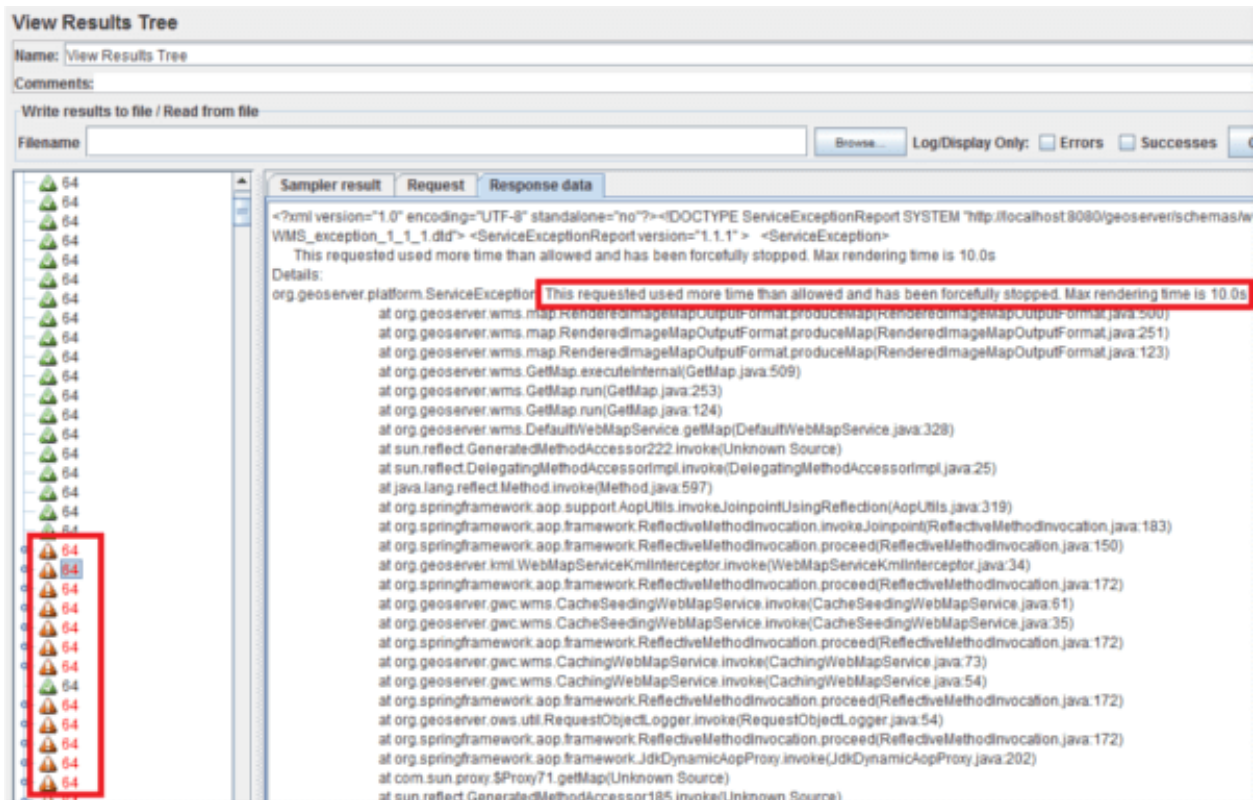


Fig. 328: Exceptions caused by maximum rendering limit exceeded

You may see that the throughput is increased because most of the timed out requests have been removed. With this kind of configuration you can control the responsiveness of your GeoServer by removing stale requests instead of waiting for them.

Note: At the end of the test remove the limits and restore the previous configuration of the `storms` DataStore

Configuring JMeter for testing Control Flow plugin

This section explains how GeoServer performances are improved when using Control-Flow plugin.

This plugin avoid GeoServer to execute too many requests together, which could lead to bad performances, by reducing the number of concurrent operations to execute and appending the others to a queue. This behaviour improves GeoServer scalability.

Note: This example requires to have already completed [Adding a ShapeFile](#) and [Adding a Style](#) sections.

Configure JMeter

1. Go to `$TRAINING_ROOT/data/jmeter_data` and copy the file `template.jmx` file into `controlflow.jmx`
2. From the training root, on the command line, run `jmeter.bat` (or `jmeter.sh` if you're on Linux) to start JMeter
3. On the top left go to *File* → *Open* and search for the new *jmx* file copied
4. Disable **View Results Tree** section
5. In the CSV Data Set Config element, modify the **path** of the CSV file by setting the path for the file `controlflow.csv` in the `$TRAINING_ROOT/data/jmeter_data` directory
6. In the **HTTP Request Default** element modify the following parameters:

Name	Value
layers	geosolutions:Mainrd
srs	EPSG:2876

Test without Control Flow

1. Run the test

Note: Remember to run and stop the test a few times for having stable results

2. When the test is completed, Save the results in a text file.

You should notice that the throughput initially increases and then starts to decrease. This is associated to a bad scalability of the input requests. Remember which number of threads provides better throughput (it should be 8). This value indicates the maximum number of concurrent requests that the server can execute simultaneously.

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec
1	100	22	7	87	14.88	0.00%	43.1/sec	1298.85
2	100	25	7	106	16.82	0.00%	51.3/sec	1625.25
4	200	36	7	179	30.57	0.00%	79.9/sec	2776.33
8	200	55	8	686	84.33	0.00%	86.0/sec	2595.12
16	400	140	8	1739	229.49	0.00%	81.7/sec	2744.13
32	384	234	7	2409	365.38	0.00%	85.3/sec	2785.63
64	768	663	6	7659	1034.06	0.00%	73.1/sec	2357.59
TOTAL	2152	315	6	7659	698.16	0.00%	74.2/sec	2410.14

Fig. 329: Decreased throughput (Note the results may be different in other machines)

3. Remove the result from JMeter by clicking on *Run* → *Clear All* on the menu
4. Stop GeoServer

Configure Control Flow

1. Go to `$TRAINING_ROOT/data/plugins/not_installed` and copy `geoserver-2.6-SNAPSHOT-control-flow-plugin.zip` zip file inside `$TRAINING_ROOT/tomcat-6.0.36/instances/instance1/webapps/geoserver/WEB-INF/lib`

2. Unzip the content of `geoserver-2.6-SNAPSHOT-control-flow-plugin.zip` inside the same folder
3. Go to `$TRAINING_ROOT/geoserver_data` and create a new file called `controlflow.properties` and add the following snippet

```
# don't allow more than 8 WMS GetMap in parallel
ows.wms.getmap=8
```

This code snippet indicates that no more than 8 *GetMap* request can be executed simultaneously by the WMS service. Other informations about the configuration can be found in the next section

Note: If during your test you have found another number for the maximum throughput, you should set that value instead of 8

Test with Control Flow

1. Restart GeoServer
2. Run again the test.

You may see that the throughput is no more reduced after the control-flow configuration, because the input requests are scheduled by the control-flow plugin, improving GeoServer scalability.

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec
1	100	22	8	78	14.56	0.00%	42.9/sec	1292.72
2	100	26	7	113	17.05	0.00%	54.3/sec	1717.85
4	200	36	7	185	32.52	0.00%	75.4/sec	2619.41
8	200	56	8	566	85.81	0.00%	82.6/sec	2494.32
16	400	133	8	767	115.25	0.00%	88.2/sec	2960.10
32	384	249	9	962	147.60	0.00%	90.0/sec	2941.04
64	768	594	8	2400	292.81	0.00%	88.1/sec	2842.03
TOTAL	2152	292	7	2400	305.38	0.00%	88.1/sec	2611.34

Fig. 330: Stable throughput (Note the results may be different in other machines)

Configuring JMeter for testing the Marlin renderer

This section explains how GeoServer performances are improved when using the **Marlin** renderer.

The Oracle JDK and OpenJDK come with two different anti-aliased renderers:

- Oracle JDK uses **Ductus**, a fast native renderer that has scalability issues (good for desktop use, less so on the server side)
- OpenJDK uses **Pisces**, a pure java renderer that is not as fast as “Ductus”, but has good scalability (anecdotally, it becomes faster than Ductus above the 4 concurrent requests)

The **Marlin** renderer is an improved version of Pisces that is as fast, if not faster, than Ductus, and scales up just as well as Pisces.

Configure JMeter

1. Go to `$TRAINING_ROOT/data/jmeter_data` and copy the file `template.jmx` file creating a `marlin.jmx` file

2. From the training root, on the command line, run `jmeter.bat` (or `jmeter.sh` if you're on Linux) to start JMeter
3. On the top left go to *File* → *Open* and search for the new *jmx* file copied
4. Disable **View Results Tree** section
5. In the **CSV Data Set Config** element, modify the **path** of the CSV file by setting the path for the file `controlflow.csv` in the `$TRAINING_ROOT/data/jmeter_data` directory
6. In the **HTTP Request Default** element modify the following parameters:

Name	Value
layers	boulder
srs	EPSG:2876

Test without Marlin

1. Go and remove the contro
2. Run the test

Note: Remember to run and stop the test a few times for having stable results

3. When the test is completed, Save the results in a text file.

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
1	100	93	45	278	41.16	0.00%	10.7/sec	860.89	82689.5
2	100	102	45	233	42.76	0.00%	17.0/sec	1587.18	95468.8
4	200	156	56	458	81.37	0.00%	23.0/sec	2416.47	107540.4
8	200	231	50	881	150.49	0.00%	28.2/sec	2590.41	94206.5
16	400	467	55	2759	410.00	0.00%	29.4/sec	2689.13	93542.2
32	384	1169	76	8800	1210.31	0.00%	22.3/sec	2126.48	97636.6
64	768	2560	77	16308	2754.93	0.00%	21.5/sec	1951.11	92792.3
TOTAL	2152	1254	45	16308	2015.61	0.00%	22.1/sec	2045.80	94953.1

Fig. 331: *Throughput without Marlin (Note the results may be different in other machines)*

4. Remove the result from JMeter by clicking on *Run* → *Clear All* on the menu
5. Stop GeoServer

Setup Marlin

1. Stop GeoServer
2. Download the Marlin rasterizer library at <https://github.com/bourgesl/marlin-renderer/releases/download/v0.4.4/marlin-0.4.4.jar> and save it in `%TRAINING_ROOT%\data`
3. Open `%TRAINING_ROOT%\setenv.bat` and add the following lines to enable the Marlin renderer, right before the “Tomcat options for the JVM” section:

```
REM Marlin support
set JAVA_OPTS=%JAVA_OPTS% -Xbootclasspath/p:"%ROOT%\data\marlin-0.4.4.jar"
set JAVA_OPTS=%JAVA_OPTS% -Dsun.java2d.renderer=org.marlin.pisces.
↪PiscesRenderingEngine
```

4. Start GeoServer again

5. Go to the map preview and open the boulder layer, you should see the following in the Tomcat console:

```
INFO: ↪
↪=====
INFO: Marlin software rasterizer           = ENABLED
INFO: Version                             = [marlin-0.4.4]
INFO: sun.java2d.renderer                  = org.marlin.pisces.
↪PiscesRenderingEngine
INFO: sun.java2d.renderer.useThreadLocal   = true
INFO: sun.java2d.renderer.useRef           = soft
INFO: sun.java2d.renderer.pixelSize        = 2048
INFO: sun.java2d.renderer.subPixel_log2_X  = 3
INFO: sun.java2d.renderer.subPixel_log2_Y  = 3
INFO: sun.java2d.renderer.tileSize_log2    = 5
INFO: sun.java2d.renderer.useFastMath      = true
INFO: sun.java2d.renderer.useSimplifier     = false
INFO: sun.java2d.renderer.doStats           = false
INFO: sun.java2d.renderer.doMonitors        = false
INFO: sun.java2d.renderer.doChecks          = false
INFO: sun.java2d.renderer.useJul            = false
INFO: sun.java2d.renderer.logCreateContext  = false
INFO: sun.java2d.renderer.logUnsafeMalloc   = false
INFO: ↪
↪=====
```

Test with Marlin renderer

1. Run again the test.

You may see that the throughput got significantly higher, especially at mid-high thread counts

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput
1	100	95	43	480	57.06	0.00%	10.4/sec
2	100	98	42	247	41.45	0.00%	16.7/sec
4	200	129	50	424	60.50	0.00%	26.1/sec
8	200	176	42	450	77.44	0.00%	38.2/sec
16	400	401	51	1117	191.52	0.00%	36.0/sec
32	384	790	79	1990	355.59	0.00%	34.1/sec
64	768	1538	95	3596	617.92	0.00%	37.3/sec
TOTAL	2152	802	42	3596	717.43	0.00%	30.1/sec

Fig. 332: Throughput with Marlin (Note the results may be different in other machines)

Configuring the Control flow plugin

The `control-flow` module for GeoServer allows the administrator to control the amount of concurrent requests actually executing inside the server. This kind of control is useful for a number of reasons:

- *Performance*: tests show that, with local data sources, the maximum throughput in *GetMap* requests is achieved when allowing at most 2 times the number of CPU cores requests to run in parallel.
- *Resource control*: requests such as *GetMap* can use a significant amount of memory. The *WMS request limits* allow to control the amount of memory used per request, but an `OutOfMemoryError` is still possible if too many requests run in parallel. By controlling also the amount of requests executing it's possible to limit the total amount of memory used below the memory that was actually given to the Java Virtual Machine.
- *Fairness*: a single user should not be able to overwhelm the server with a lot of requests, leaving other users with tiny slices of the overall processing power.

The control flow method does not normally reject requests, it just queues up those in excess and executes them late. However, it's possible to configure the module to reject requests that have been waited in queue for too long.

Rule syntax reference

The current implementation of the control flow module reads its rules from a `controlflow.properties` property file located in the *GeoServer data directory*.

Total OWS request count

The global number of OWS requests executing in parallel can be specified with:

```
ows.global=<count>
```

Every request in excess will be queued and executed when other requests complete leaving some free execution slot.

Per request control

A per request type control can be demanded using the following syntax:

```
ows.<service>[.<request>[.<outputFormat>]]=<count>
```

Where:

- `<service>` is the OWS service in question (at the time of writing can be `wms`, `wfs`, `wcs`)
- `<request>`, optional, is the request type. For example, for the `wms` service it can be `GetMap`, `GetFeatureInfo`, `DescribeLayer`, `GetLegendGraphics`, `GetCapabilities`
- `<outputFormat>`, optional, is the output format of the request. For example, for the `wms` `GetMap` request it could be `image/png`, `image/gif` and so on

A few examples:

```
# don't allow more than 16 WCS requests in parallel
ows.wcs=16
# don't allow more than 8 GetMap requests in parallel
ows.wms.getmap=8
# don't allow more than 2 WFS GetFeature requests with Excel output format
ows.wfs.getfeature.application/msexcel=2
```

Per user control

This avoid a single user to make too many requests in parallel:

```
user=<count>
```

Where `<count>` is the maximum number of parallel requests a single user can execute in parallel. The user tracking mechanism is cookie based, so it will work fine for browsers but not as much for other kinds of clients. An IP based mechanism is not provided at the time, but it would have its own fallacies as well, as it would limit all the users sitting behind a single router to `<count>` requests (imagine the effect on a big public administration).

Timeout

A request timeout is specified with the following syntax:

```
timeout=<seconds>
```

where `<seconds>` is the number of seconds a request can stay queued waiting for execution. If the request does not enter execution before the timeout expires it will be rejected.

A complete example

Assuming the server we want to protect has 4 cores a sample configuration could be:

```
# if a request waits in queue for more than 60 seconds it's not worth executing,
# the client will likely have given up by then
timeout=60
# don't allow the execution of more than 100 requests total in parallel
ows.global=100
# don't allow more than 10 GetMap in parallel
ows.wms.getmap=10
# don't allow more than 4 outputs with Excel output as it's memory bound
ows.wfs.getfeature.application/msexcel=4
# don't allow a single user to perform more than 6 requests in parallel
# (6 being the Firefox default concurrency level at the time of writing)
user=6
```

Running GeoNode under SSL

Enabling SSL will encrypt traffic between your GeoNode server and client browsers. This approach involves re-configuring Apache to serve on port 443, instead of port 80. Other approaches exist and should be added to this document.

Generate SSL Key & Certificate

The first step is to generate a DES key.:

```
# for CommonName use GeoNode domain name or ip address as specified in GeoNode's
↪ SITEURL
openssl genrsa -des3 -out server.key 1024
openssl req -new -key server.key -out server.csr
```

(continues on next page)

(continued from previous page)

```
# generate new server.key without challenge password, or Apache will ask for password
↳at startup
mv server.key server.key.tmp
openssl rsa -in server.key.tmp -out server.key

# generate certificate
openssl x509 -req -days 365 -in server.csr -signkey server.key -out server.crt
```

Copy the key and certificate to the standard locations:

```
sudo cp server.crt /etc/ssl/certs/geonode.crt
sudo cp server.key /etc/ssl/private/geonode.key
```

It seems not necessary in standard GeoNode installations (via ppa) but if you plan to use other python or java applications, it could be useful also add the certificate to the cacerts file for python and java:

```
sudo -s "cat server.crt >> /usr/lib/python2.7/dist-packages/httplib2/cacerts.txt"
sudo keytool -import -alias geonodessl -keystore /etc/ssl/certs/java/cacerts -file_
↳server.crt
```

Note: keytool will ask for a password and the standard password for the java cacerts file is changeit.

Apache Configuration

Enable the SSL module in Apache with the command:

```
sudo a2enmod ssl
```

Next as root edit the Apache geonode config file `/etc/apache2/sites-available/geonode.conf`. At the beginning of the file replace:

```
<VirtualHost *:80>
```

with:

```
<IfModule mod_ssl.c>
<VirtualHost _default_:443>
```

At the bottom of the file, replace:

```
</VirtualHost>
```

with:

```
SSLEngine on
SSLCertificateFile /etc/ssl/certs/geonode.crt
SSLCertificateKeyFile /etc/ssl/private/geonode.key
BrowserMatch "MSIE [2-6]" \
    nokeepalive ssl-unclean-shutdown \
    downgrade-1.0 force-response-1.0
# MSIE 7 and newer should be able to use keepalive
BrowserMatch "MSIE [17-9]" ssl-unclean-shutdown
</VirtualHost>
</IfModule>
```

(continues on next page)

(continued from previous page)

```
<VirtualHost *:80>
  Redirect permanent / https://<ipaddressOrDomainName>/
</VirtualHost>
```

Replace `<ipaddressOrDomainName>` with current value.

This tells Apache where to find the key and certificate. There are also some additional lines to handle MSIE, taken from Apache's default-ssl file.

Tomcat Configuration

As root edit the Tomcat server config file `/etc/tomcat7/server.xml`, and replace:

```
<Connector port="8080" protocol="HTTP/1.1"
  connectionTimeout="20000"
  URIEncoding="UTF-8"
  redirectPort="8443"
/>
```

with:

```
<Connector port="8080" protocol="HTTP/1.1"
  connectionTimeout="20000"
  URIEncoding="UTF-8"
  scheme="https"
  proxyName="<yourServersIPorDomainName>"
  proxyPort="443"
/>
```

This tells Tomcat that it is running behind an https proxy. If this is omitted Tomcat will try to redirect to http.

GeoNode Configuration

As root edit the geonode config file `/etc/geonode/local_settings.py` and change the `SITEURL` protocol to https:

```
SITEURL = 'https://<ipaddressOrDomainName>/'
```

GeoServer Configuration

As root edit the file `/usr/share/geoserver/WEB-INF/web.xml` and ensure the `GEONODE_BASE_URL` is specified as follows:

```
<context-param>
  <param-name>GEONODE_BASE_URL</param-name>
  <param-value>https://localhost</param-value>
</context-param>
```

Also update `proxyBaseUrl` in the Geoserver global settings file `/var/lib/geoserver/geonode-data/global.xml`:

```
<proxyBaseUrl>https://XXX.XXX.XXX.XXX/geoserver/</proxyBaseUrl>
```

Replace XXX.XXX.XXX.XXX with your server internal address.

Restart

Finally restart Apache and Tomcat with:

```
sudo /etc/init.d/apache2 restart
sudo /etc/init.d/tomcat6 restart
```

This information was compiled from a number of sources. The main links are listed below. Please contact the GeoNode list with any updates or corrections.

- <https://confluence.atlassian.com/jira/connecting-to-ssl-services-117455.html>
- <https://confluence.atlassian.com/adminjiraserver072/integrating-jira-with-apache-using-ssl-828788158.html>
- http://www.akadia.com/services/ssh_test_certificate.html
- <https://help.ubuntu.com/lts/serverguide/httpd.html>
- <https://help.ubuntu.com/lts/serverguide/certificates-and-security.html>

GeoSites: GeoNode Multi-Tenancy

GeoSites is a way to run multiple websites with a single instance of GeoNode. Each GeoSite can have different templates, apps, and data permissions but share a single database (useful for sharing users and data layers), GeoServer, and CSW. This is useful when multiple websites are desired to support different sets of users, but with a similar set of data and overall look and feel of the sites. Users can be given permission to access multiple sites if needed, which also allows administrative groups can be set up to support all sites with one account.

Master Website

A GeoSites installation uses a ‘master’ GeoNode website that has some additional administrative pages for doing data management. Layers, Maps, Documents, Users, and Groups can all be added and removed from different sites. Users can be given access to any number of sites, and data may appear on only a single site, or all of them. Additionally, if desired, any or all of the Django apps installed on the other sites can be added to the master site to provide a single administrative interface that gives full access to all apps. The master site need not be accessible from the outside so that it can be used as an internal tool to the organization.

Users created on a particular site are created with access to just that site. Data uploaded to a particular site is given permission on that site as well as the master site. Any further adjustments to site-based permissions must be done from the master site.

Database

The master site, and all of the individual GeoSites, share a single database. Objects, including users, groups, and data layers, all appear within the database but an additional sites table indicates which objects have access to which sites. The geospatial data served by GeoServer (e.g., from PostGIS) can exist in the database like normal, since GeoServer will authenticate against GeoNode, which will use it’s database to determine permissions based on the object, current user, and site.

GeoServer

A single GeoServer instance is used to serve data to all of the GeoSites. Data that is common to all sites can be added to the master site which will appear in the generic ‘geonode’ workspace.

Settings Files and Templates

A key component in managing multiple sites is keeping data organized and using a structured series of settings files so that common settings can be shared and only site specific settings are separated out. It is also best to import the default GeoNode settings from the GeoNode installation. This prevents the settings from having to be manually upgraded if there is any default change the GeoNode settings.

Settings which are common to all GeoSites, but differ from the default GeoNode, are separated into a contrib/geosites/settings.py file. Then, each individual site has settings file which imports from the master site and will then only need to specify a small selection that make that site unique, such as:

- SITE_ID: Each one is unique, the master site should have a SITE_ID of 1.
- SITENAME
- SITEURL
- ROOT_URLCONF: This may be optional. The master site url.conf can be configured to automatically import the urls.py of all SITE_APPS, so a different ROOT_URLCONF is only needed if there are further differences.
- SITE_APPS: Containing the site specific apps
- App settings: Any further settings required for the above sites
- Other site specific settings, such as REGISTRATION_OPEN

A GeoSite therefore has three layers of imports, which is used for settings as well as the search path for templates. First it uses the individual site files, then the master GeoSite, then default GeoNode. These are specified via variables defined in settings:

- SITE_ROOT: The directory where the site specific settings and files are located (templates, static)
- PROJECT_ROOT: The top-level directory of all the GeoSites which should include the global settings file as well as template and static files
- GEONODE_ROOT: The GeoNode directory.

The TEMPLATE_DIRS, and STATICFILES_DIRS will then include all three directories as shown:

```
TEMPLATE_DIRS = (
    os.path.join(SITE_ROOT, 'templates/'),
    os.path.join(PROJECT_ROOT, 'templates/'), # files common to all sites
    os.path.join(GEONODE_ROOT, 'templates/')
)

STATICFILES_DIRS = (
    os.path.join(SITE_ROOT, 'static/'),
    os.path.join(PROJECT_ROOT, 'static/'),
    os.path.join(GEONODE_ROOT, 'static/')
)
```

At the end of the post_settings.py the following variables will be set based on site specific settings:

```
STATIC_URL = os.path.join(SITEURL, 'static/')
GEONODE_CLIENT_LOCATION = os.path.join(STATIC_URL, 'geonode/')
GEOSERVER_BASE_URL = SITEURL + 'geoserver/'
if SITE_APPS:
    INSTALLED_APPS += SITE_APPS
```

Templates and Static Files

As mentioned above for each website there will be three directories used for template and static files. The first template file found will be the one used so templates in the SITE_ROOT/templates directory will override those in PROJECT_ROOT/templates, which will override those in GEONODE_ROOT/templates.

Static files work differently because (at least on a production server) they are collected and stored in a single location. Because of this care must be taken to avoid clobbering of files between sites, so each site directory should contain all static files in a subdirectory with the name of the site (e.g., static_root/siteA/logo.png)

The location of the proper static directory can then be found in the templates syntax such as:

```
{{ STATIC_URL }}{{ SITENAME|lower }}/logo.png
```

Permissions by Site

By default GeoNode is publicly available. In the case of GeoSites, new data will be publicly available, but only for the site it was added to, and the master site (all data is added to the master site).

Activate geosites app

In order to use geonode in Multi-tenancy mode, follow these steps: 1. check in settings.py if 'geonode.contrib.geosites' in GEONODE_CONTRIB_APPS is rightly uncommented 2. add in settings.py 'geonode.contrib.geosites' in INSTALLED_APPS 3. run python manage.py syncdb

Adding New Sites

To add a new site follow the following steps:

1. copy the directory site_template in your geonode-project folder and give it a name
2. from geonode administration pannel add 'new site'
3. create a virtualhost in the webserver related to the new created site. Remember to setup the WSGIDeamon-Process with the name you gave to the folder created at point 1. and the path to the geosites directory. WSGIProcessGroup have to be pointed to the name you choose for the folder you created at point 1. Eventually, WSGIScriptAlias have to be set to the wsgi.py you have in your site folder.
4. **check the configuration files: local_settings.py, pre_settings.py, post_settings.py in /geonode-project as well as local_settings.py**
 - in /geonode-project/local_settings.py set the variable SERVE_PATH. It has to point geosites folder.
 - **in the local_setting of the site folder insert the values to the following variables:**
 - SITE_ID
 - SITE_NAME

– SITE_URL

5. create static_root directory where you usually let the webserver serve webpages (e.g., /var/www) and give it grants to be accessed by the user www-data
6. **create an uploaded/layers and uploaded/thumbs folder in your geonode-project folder and give them grants as follow:**
`sudo mkdir -p geonode-project/uploaded/thumbs sudo mkdir -p geonode-project/uploaded/layers sudo
chmod -Rf 755 geonode-project/uploaded/thumbs sudo chmod -Rf 755 geonode-project/uploaded/layers`
7. run `python manage.py collectstatics` - Pay attention on the folder where you are running the command and the folder structure of your geonode/geosites project, in case pass to the path the settings file by using `--settings` to the python command
8. you can customize the look and feel of your site working on the css and html file you find in your template site directory. After a change, run again `collectstatics` command.

Advanced Data Management and Processing Advanced Data Management and Processing techniques.

GeoNode Advanced Configuration Learn how to deal with advanced GeoNode configuration settings and external Django Apps.

GeoNode on Production Concepts and techniques for the deployment of GeoNode and GeoServer on a Production system.

GeoNode Overview & Reference This module guides the user to an overview of GeoNode and its main components.

At the end of this section you will have a clear view of what GeoNode is and can do. You will be able also to use the GeoNode main functionalities and understand some of the basic concepts of the system infrastructure.

Installation & Admin This module is more oriented to users having some System Administrator background.

At the end of this section you will be able to setup from scratch the whole GeoNode infrastructure and understand how to the different pieces are interconnected and which are their dependencies.

Prerequisites Before proceeding with the reading, it is strongly recommended to be sure having clear the following concepts:

1. GeoNode and Django framework basic concepts
2. What is Python
3. What is a DBMS
4. What is a Java Virtual Machine and the JDK
5. Linux OS basic shell and maintenance commands
6. Basic TCP/IP and networking concepts
7. Apache HTTPD Server and WSGI Python bindings

Users Workshop This workshop will teach how to use the GeoNode going in depth into what we can do with software application. At the end of this section you will master all the GeoNode sections and entities from a user perspective.

You will know how to:

1. Manage users accounts and how to modify them.
2. Use and manage the different GeoNode basic resources.
3. Use the GeoNode searching tools to find your resources.
4. Manage Layers and Maps, update the styles and publish them.
5. Load datasets into GeoNode and keep them synchronized with GeoServer.

Prerequisites Before proceeding with the reading, it is strongly recommended to be sure having clear the following concepts:

1. GeoNode and Django framework basic concepts
2. What is Python
3. What is a geospatial server and a basic knowledge of the geospatial web services.
4. What is a metadata and a catalog.
5. What is a map and a legend.

Administrators Workshop This workshop will teach how to install and manage a deployment of the [GeoNode](#) software application. At the end of this section you will master all the GeoNode sections and entities from an administrator perspective.

You will know how to:

1. Use the GeoNode's Django Administration Panel.
2. Use the console Management Commands for GeoNode.
3. Configure and customize your GeoNode installation.

Prerequisites Before proceeding with the reading, it is strongly recommended to be sure having clear the following concepts:

1. GeoNode and Django framework concepts
2. Good knowledge of Python
3. Good knowledge of what is a geospatial server and geospatial web services.
4. Good knowledge of what is metadata and catalog.
5. Good knowledge of HTML and CSS.

Developers Workshop This workshop will teach how to develop with and for the [GeoNode](#) software application. This module will introduce you to the components that GeoNode is built with, the standards that it supports and the services it provides based on those standards, and an overview its architecture.

Prerequisites GeoNode is a web based GIS tool, and as such, in order to do development on GeoNode itself or to integrate it into your own application, you should be familiar with basic web development concepts as well as with general GIS concepts.

Advanced Workshop This module introduces advanced techniques and methodologies for the management of the geospatial data and the maintenance and tuning of the servers on *Production Environments*.

The last sections of the module will teach also you how to add brand new classes and functionalities to your GeoNode installation.

Prerequisites You should be familiar with GeoNode, GeoServer, Python framework and development concepts other than with system administrator and caching concepts and techniques.

3.3 Reference

The [Reference](#) section provides details about the internals of the GeoNode project. It has background information about components that make up GeoNode, the security system, APIs and much more.

3.3.1 Reference documentation

In this section, you will find information about every component of GeoNode, such as GeoServer, GeoNode settings, security, etc.

Security and Permissions

GeoNode ad-hoc API

Localization

GeoNode Django Apps

JavaScript in GeoNode

Settings

Supported Browsers

GeoSites: GeoNode Multi-Tenancy

3.3.1.1 Security and Permissions

GeoNode has the ability to restrict the access on your layers, maps, and documents to other users or group of users. This section should help you to understand which restrictions are possible and what to take care of while using them. Generally permissions can be set on all your uploaded data. Here's an overview:

1. Users
 - Superuser permissions
 - Django admin interface permissions
2. Groups
 - Public
 - Public (invite only)
 - Private
3. Layers
 - View a layer
 - Download a layer
 - Change metadata for a layer
 - Edit layer's features
 - Edit styles for a layer
 - Manage a layer (update, delete, change permissions, publish/unpublish it)
4. Maps
 - View a map
 - Download a map
 - Change metadata for a map
 - Manage a map (update, delete, change permissions, publish/unpublish it)
5. Documents

- View a document
- Download a document
- Change metadata for a document
- Manage a document (update, delete, change permissions, publish/unpublish it)

To understand how this permissions can be set, you first have to know about the different kinds of users.

3.3.1.1.1 Permissions and GeoNode objects

Users

GeoNode has two types of users:

- Unregistered users (anonymous)
- Registered users

An unregistered user is someone who is just visiting the site, but doesn't have any data uploaded yet. A registered user has already done that. But there are even more kinds of registered users! A registered user can have one or more of those three status:

- Superuser
- Staff
- Active

A superuser is usually generated directly after the installation of GeoNode via the terminal. When creating a *superuser* through the terminal it always has the status *active* and the status *staff* as well. It is also important to know that a superuser is a user that has all permissions without explicitly assigning them! That means that he is able to upload and edit layers, create maps etc. and can not be restricted from that! So the superuser is basically the administrator, who knows and has access on everything.

The status *staff* only implies that a user with this status is able to attend the *Django Admin Interface*. *Active* has no special meaning, it only says that there is a user and it is available. Instead of deleting this user, you could just unset the status *active*, your user will still be remaining, but it won't show up.

There are several options to create a user:

- From the terminal: Here you can only create a *superuser*
- From the GeoNode interface (when GeoNode registration are open): A *normal* user will be created by signing up to GeoNode. It only has the status *active* so far!
- From the GeoNode interface (when GeoNode registration are closed): A superuser will be able to invite a user
- From the Django administrative interface: a new user can be created as well as the status of an already existing user can be changed, e.g make a generic user a superuser.

Groups

In GeoNode you can assign permissions to groups, all the users that belong to the group will inherit its permissions.

If you are an administrator you can create a group in the dedicated tab and invite or assign users to it. The group will be available in the permissions widget in geonode and you will be able to assign object permissions to it.

Layers

As mentioned above, a superuser or the layer owner or a user with management permissions on the layer should be able to restrict other users from the layer itself.

Generally there are the following types of permissions:

These are permissions that is possible to assign to a GeoNode layer:

- Who can view the layer
- Who can download the layer
- Who can change the metadata of the layer
- Who can edit data of the layer
- Who can edit styles of the layer
- Who can manage the layer (update, delete, change permissions, publish/unpublish it)

Each of these permissions can be assigned to:

- Anyone (only for who can view and download)
- One or more users
- One or more groups

A user with all of these permissions in the layer detail page will have a button to download the layer, a button to download its metadata, a button to change the layer permissions and an edit button that will display links to:

- Edit metadata
- Edit styles
- Manage styles
- Replace the layer
- Remove the layer

This can also be seen here:

If the layer is vectorial the user will be able also to edit the layer's features in a GeoNode map (the "Edit" tool should be enabled).

Now take a closer look on to the section *Edit Metadata*. All the following things can be edited in the metadata section:

- Owner
- Title
- Date
- Data type
- Edition
- Abstract
- Purpose
- Maintenance frequency
- Keywords region
- Restrictions
- Restrictions other

Set permissions for this layer



Who can view it?

☐ Anyone

The following users:

× bobby

The following groups:

× analysts

× mappers

Who can download it?

☐ Anyone

The following users:

× admin

× bobby

The following groups:

× analysts

Who can change metadata for it?

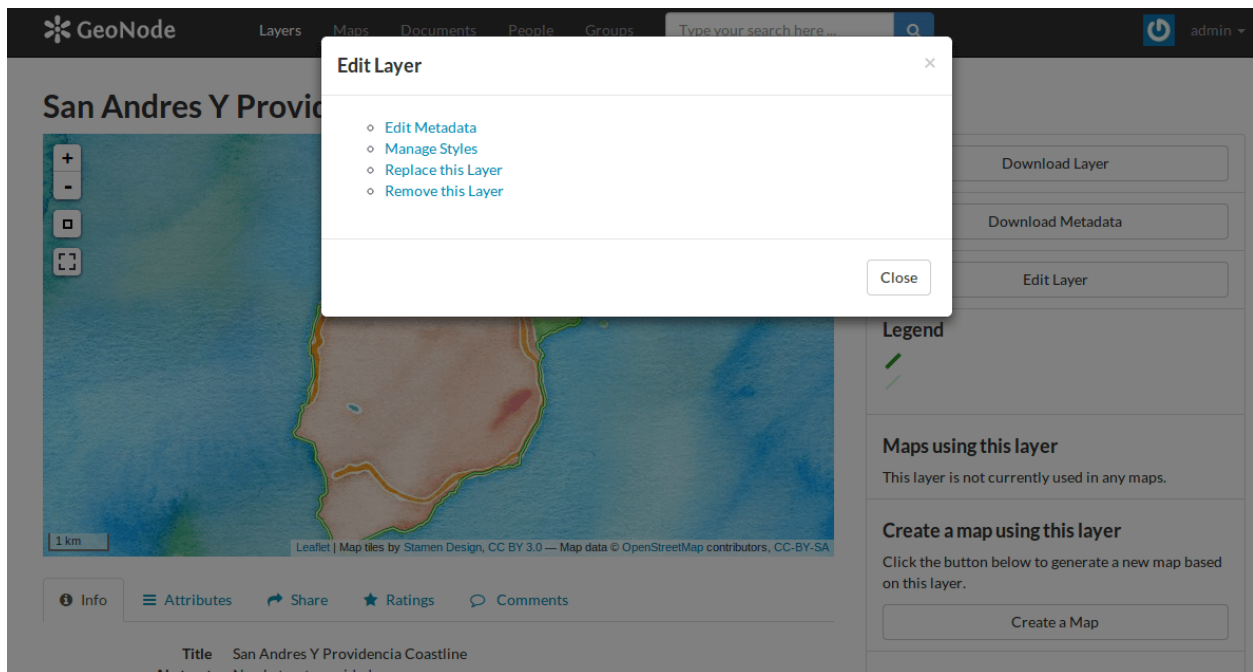
Who can edit data for this layer?

Who can edit styles for this layer?

Who can manage it? (update, delete, change permissions, publish/unpublish it?)

Cancel

Apply Changes



- Language
- Category
- Spatial representation type
- Temporal extent start
- Temporal extent end
- Supplemental information
- Distribution URL
- Distribution description
- Data quality statement
- Keywords
- Point of contact
- Metadata author
- Attributes (those can though not be changed!)

Maps

Generally all the same applies to maps, but with fewer options:

- Who can view the map
- Who can download the map
- Who can change the metadata of the map
- Who can manage (delete, change permissions, publish/unpublish it, set map thumbnail)

The section *Edit metadata* is almost the same as for layers, with two more options:

- Metadata XML
- Thumbnail

In *Set map thumbnail* the thumbnail of the map can be set.

Documents

The same permissions that can be used on layers can be used on the documents, with the exception of the edit data and edit styles permissions.

Require authentication to access GeoNode

By default, unregistered users cannot view maps, layers, and documents on your site without being authenticated. GeoNode comes a security option that requires users to authenticate before accessing any page. To enable this option, set the `LOCKDOWN_GEOCODE` setting to true in your `settings.py` file. You can fine-tune which URL routes are white-listed (accessible by unregistered users) by listing the routes in the `AUTH_EXEMPT_URLS` tuple. See the *GeoNode Django Apps* documentation for more information.

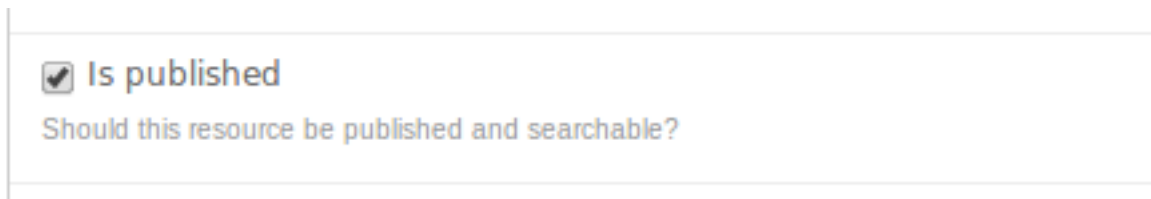
3.3.1.1.2 Publishing and unpublishing objects

By default GeoNode does not implement any kind of mechanism to publish/unpublish resources such as layer, maps and documents.

Setting the `RESOURCE_PUBLISHING` to True such a workflow is used, and by default new uploaded resources are unpublished.

It is possible for any GeoNode staff member that has permissions on the base/ResourceBase model to decide to publish/unpublish a layer, map or document.

The staff member can go to the resource base Django admin page, and publish or unpublish the resource by checking or unchecking the `is_published` field:



The image shows a snippet of a web form from the Django admin interface. It features a checkbox with a checkmark inside, followed by the text 'Is published' in a bold, blue font. Below this, there is a line of text in a lighter blue font that asks 'Should this resource be published and searchable?'. The entire snippet is enclosed in a light gray border.

When the resource is unpublished, it will be not available to any user, including administrators, in the GeoNode site. If the unpublished resource is a layer it will be considered in the `GetCapabilities` generated by GeoServer.

The unpublished resource will not be reachable by anyone using GeoNode search features. The only way to access to it is by the Django admin site, from where it will be eventually possible to publish again the resource by a staff member, or from the layer details page, accessible by any user with the `publish_resourcebase` permission on that layer.

3.3.1.2 GeoNode ad-hoc API

GeoNode provides a JSON API which currently supports the GET method. The API is also used as main search engine.

3.3.1.2.1 API endpoints

GeoNode provides some endpoints and filtering.

- `"/api/base"` query on the ResourceBase table and returns combined results of Maps, Layers Documents and Services
- `"/api/layers"` query the Layer table
- `"/api/maps"` query the Map table
- `"/api/documents"` query the Document table
- `"/api/groups"` query the GroupProfile table (which contains the Groups)
- `"/api/profiles"` query the Profile table (which is the geonode authentication table)
- `"/api/categories"` query the Category table
- `"/api/keywords"` query the Tag table
- `"/api/featured"` query the ResourceBase table by limiting the items to the ones flagged as "featured" (listed in home page)

3.3.1.2.2 API filtering

The API allow filtering by adding Django style model filters to the URL.

As an example, filtering by title corresponds to a URL like `"/api/layers?title__contains=grid"` It's also possible to filter by related tables like `"/api/layers?keywords__slug__exact=the-keyword"`

There are many possible filter, refer to the [django filters guide](#).

3.3.1.2.3 API limit and pagination

It's possible to limit the number of the results returned by the API by adding a limit parameter like `"/api/layers?limit=10"` It's also possible to specify an offset so that the first results will not be returned (together with the limit this makes a pagination logic), `"/api/layers?offset=5"`

So a query like `"/api/layers?offset=5&limit=10"` will return 10 results starting from the 6th found in the database.

3.3.1.2.4 API settings

You can configure how many results will be lists per page on the client (in the list pages and search page) by changing this line <https://github.com/GeoNode/geonode/blob/master/geonode/settings.py#L643>

And you can set the amount of data returned by default from the API (if the limit parameter is not set), the default is 0 which means no limit <https://github.com/GeoNode/geonode/blob/master/geonode/settings.py#L646>

3.3.1.2.5 Searching with Haystack

GeoNode is ready to use a complete full text search engine. Note that haystack will be used only on the base, layers, maps and documents API.

Once activated the full text API is reachable by appending "search" to the URL, for example `"/api/base/search?limit=0&offset=0"`

Although the backend type is not mandatory, we suggest (for simplicity) to use [Elasticsearch](#):

To activate the search backend make sure that you have a running instance of Elasticsearch, then uncomment the following line in the geonode settings:

<https://github.com/GeoNode/geonode/blob/master/geonode/settings.py#L219>

And activate the search through the settings at the line:

<https://github.com/GeoNode/geonode/blob/master/geonode/settings.py#L607>

Also uncomment and correct the address of Elasticsearch if needed: <https://github.com/GeoNode/geonode/blob/master/geonode/settings.py#L612> <https://github.com/GeoNode/geonode/blob/master/geonode/settings.py#L619>

You can do some more customizations like:

- should the search skip the permissions filtering? <https://github.com/GeoNode/geonode/blob/master/geonode/settings.py#L609>
- should the search update the facets counts on every search you make or keep the standard behavior? <https://github.com/GeoNode/geonode/blob/master/geonode/settings.py#L611>
- How many results should the backend return by default? <https://github.com/GeoNode/geonode/blob/master/geonode/settings.py#L620>

3.3.1.3 Localization

To enable a new language in GeoNode you have to do the following:

1. Install gettext:

```
sudo apt-get install gettext
```

2. Create a directory named locale in the root of your project:

```
mkdir locale
```

3. In the root of your project, run:

```
python manage.py makemessages -l fr
```

4. Navigate to the GeoNode directory and do:

```
cd src/GeoNodePy/geonode/maps; django-admin.py makemessages -l fr  
cd src/GeoNodePy/geonode; django-admin.py makemessages -l fr
```

Optional steps:

1. Install django-rosetta:

```
http://code.google.com/p/django-rosetta/
```

2. Install django-modeltranslation

3. If you want to enable metadata in the other format too, make sure you have model translation installed and create a translations.py file like this:

```
from modeltranslation.translator import translator, TranslationOptions  
from geonode.maps.models import Layer
```

(continues on next page)

(continued from previous page)

```

class LayerTO(TranslationOptions):
    fields = (
        'title',
        'edition',
        'abstract',
        'purpose',
        'constraints_other',
        'data_quality_statement',
        'supplemental_information',
    )

translator.register(FlatBlock, FlatBlockTO)
translator.register(Layer, LayerTO)

```

3.3.1.4 Developers Reference

Here you will find information about each and every component of GeoNode, for example GeoServer, GeoNode settings, security, etc.

GeoNode Django Apps

JavaScript in GeoNode

Settings

GeoSites: GeoNode Multi-Tenancy

3.3.1.4.1 GeoNode Django Apps

The user interface of a GeoNode site is built on top of the Django web framework. GeoNode includes a few “apps” (reusable Django modules) to support development of those user interfaces. While these apps have reasonable default configurations, for customized GeoNode sites you will probably want to adjust these apps for your specific needs.

geonode.base - GeoNode core functionality

Stores core functionality used throughout the GeoNode application.

Template Tags

num_ratings <object>

Returns the number of ratings an object has. Example usage:

```

{% load base_tags %}
{% num_ratings map as num_votes %}

<p>Map votes: {{num_votes}}.</p>

```

categories

Returns topic categories and the count of objects in each category.

geonode.documents - Document creation and management

Manages uploaded files that can be related to maps. Documents can be any type of file that is included in the `ALLOWED_DOCUMENTS_TYPES` setting.

settings.py Entries

ALLOWED_DOCUMENT_TYPES Default: `['doc', 'docx', 'xls', 'xlsx', 'pdf', 'zip', 'jpg', 'jpeg', 'tif', 'tiff', 'png', 'gif', 'txt']`

A list of acceptable file extensions that can be uploaded to the Documents app.

MAX_DOCUMENT_SIZE Default: 2

The maximum size (in megabytes) that an upload will be before it gets rejected.

geonode.layers - Layer creation and geospatial data management

This Django app provides support for managing and manipulating single geospatial datasets known as layers.

Models

- Attribute - Feature attributes for a layer managed by the GeoNode.
- Layer - A data layer managed by the GeoNode
- Style - A data layer's style managed by the GeoNode

Views

- Creating, viewing, browsing, editing, and deleting layers and their metadata

Template Tags

featured_layers Returns the 7 newest layers.

layer_thumbnail <layer> Returns the layer's thumbnail.

manage.py Commands

importlayers `python manage.py importlayers`

Brings a data file or a directory full of data files into a GeoNode site. Layers are added to the Django database, the GeoServer configuration, and the GeoNetwork metadata index.

updatelayers `python manage.py updatelayers`

Scan GeoServer for data that has not been added to GeoNode.

geonode.maps - Map creation and geospatial data management

This Django app provides some support for managing and manipulating geospatial datasets. In particular, it provides tools for editing, viewing, and searching metadata for data layers, and for editing, viewing, and searching maps that aggregate data layers to display data about a particular topic.

Models

- Map - A collection of data layers composed in a particular order to form a map
- MapLayer - A model that maintains some map-specific information related to a layer, such as the z-indexing order.

Views

The maps app provides views for:

- Creating, viewing, browsing, editing, and deleting Maps

These operations require the use of GeoServer to manage map rendering, as well as GeoExt to provide interactive editing and previewing of maps and data layers.

There are also some URL mappings in the `geonode.maps.urls` module for easy inclusion in GeoNode sites.

settings.py Entries

OGC_SERVER Default: `{}` (Empty dictionary)

A dictionary of OGC servers and their options. The main server should be listed in the 'default' key. If there is no 'default' key or if the `OGC_SERVER` setting does not exist GeoNode will raise an Improperly Configured exception. Below is an example of the `OGC_SERVER` setting:

```
OGC_SERVER = {
    'default' : {
        'LOCATION' : 'http://localhost:8080/geoserver/',
        'USER' : 'admin',
        'PASSWORD' : 'geoserver',
    }
}
```

BACKEND Default: `"geonode.geoserver"`

The OGC server backend to use. The backend choices are:

- `'geonode.geoserver'`

BACKEND_WRITE_ENABLED Default: `True`

Specifies whether the OGC server can be written to. If False, actions that modify data on the OGC server will not execute.

LOCATION Default: `"http://localhost:8080/geoserver/"`

A base URL from which GeoNode can construct OGC service URLs. If using GeoServer you can determine this by visiting the GeoServer administration home page without the `/web/` at the end. For example, if your GeoServer administration app is at <http://example.com/geoserver/web/>, your server's location is <http://example.com/geoserver>.

PUBLIC_LOCATION Default: "http://localhost:8080/geoserver/"

The URL used to in most public requests from GeoNode. This settings allows a user to write to one OGC server (the LOCATION setting) and read from a separate server or the PUBLIC_LOCATION.

USER Default: 'admin'

The administrative username for the OGC server as a string.

PASSWORD Default: 'geoserver'

The administrative password for the OGC server as a string.

MAPFISH_PRINT_ENABLED Default: True

A boolean that represents whether the MapFish printing extension is enabled on the server.

PRINT_NG_ENABLED Default: True

A boolean that represents whether printing of maps and layers is enabled.

GEONODE_SECURITY_ENABLED Default: True

A boolean that represents whether GeoNode's security application is enabled.

GEOGIG_ENABLED Default: False

A boolean that represents whether the OGC server supports GeoGig datastores.

WMST_ENABLED Default: False

Not implemented.

WPS_ENABLED Default: False

Not implemented.

DATASTORE Default: '' (Empty string)

An optional string that represents the name of a vector datastore that GeoNode uploads are imported into. In order to support vector datastore imports there also needs to be an entry for the datastore in the DATABASES dictionary with the same name. Example:

```
OGC_SERVER = {
    'default' : {
        'LOCATION' : 'http://localhost:8080/geoserver/',
        'USER' : 'admin',
        'PASSWORD' : 'geoserver',
        'DATASTORE': 'geonode_imports'
    }
}

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': 'development.db',
    },
    'geonode_imports' : {
        'ENGINE': 'django.contrib.gis.db.backends.postgis',
        'NAME': 'geonode_imports',
        'USER' : 'geonode_user',
        'PASSWORD' : 'a_password',
        'HOST' : 'localhost',
        'PORT' : '5432',
    },
}
```

(continues on next page)

(continued from previous page)

```
}
}
```

GEOSERVER_CREDENTIALS Removed in GeoNode 2.0, this value is now specified in the `OGC_SERVER` settings.

GEOSERVER_BASE_URL Removed in GeoNode 2.0, this value is now specified in the `OGC_SERVER` settings.

CATALOGUE A dict with the following keys:

- **ENGINE:** The CSW backend (default is `geonode.catalogue.backends.pycsw_local`)
- **URL:** The FULLY QUALIFIED base URL to the CSW instance for this GeoNode
- **USERNAME:** login credentials (if required)
- **PASSWORD:** login credentials (if required)

`pycsw` is the default CSW enabled in GeoNode. `pycsw` configuration directives are managed in the `PYCSW` entry.

PYCSW A dict with `pycsw`'s configuration. Of note are the sections `metadata:main` to set CSW server metadata and `metadata:inspire` to set INSPIRE options. Setting `metadata:inspire['enabled']` to `true` will enable INSPIRE support. Server level configurations can be overridden in the `server` section. See <http://docs.pycsw.org/en/stable/configuration.html> for full `pycsw` configuration details.

SITEURL Default: `'http://localhost:8000/'`

A base URL for use in creating absolute links to Django views.

DEFAULT_MAP_BASE_LAYER The name of the background layer to include in newly created maps.

DEFAULT_MAP_CENTER Default: `(0, 0)`

A 2-tuple with the latitude/longitude coordinates of the center-point to use in newly created maps.

DEFAULT_MAP_ZOOM Default: `0`

The zoom-level to use in newly created maps. This works like the OpenLayers zoom level setting; 0 is at the world extent and each additional level cuts the viewport in half in each direction.

`geonode.proxy` - Assist JavaScript applications in accessing remote servers

This Django app provides some HTTP proxies for accessing data from remote servers, to overcome restrictions imposed by the same-origin policy used by browsers. This helps the GeoExt applications in a GeoNode site to access various XML documents from OGC-compliant data services.

Views

`geonode.proxy.views.proxy` This view forwards requests without authentication to a URL provided in the request, similar to the `proxy.cgi` script provided by the OpenLayers project.

`geonode.proxy.views.geoserver` This view proxies requests to GeoServer. Instead of a URL-encoded URL parameter, the path component of the request is expected to be a path component for GeoServer. Requests to this URL require valid authentication against the Django site, and will use the default `OGC_SERVER` `USER`, `PASSWORD` and `LOCATION` settings as defined in the maps application.

`geonode.search` - Provides the GeoNode search functionality.

This Django app provides a fast search functionality to GeoNode.

Views

- `search_api`- Builds and executes a search query based on URL parameters and returns matching results in requested format.

`geonode.security` - GeoNode granular Auth Backend

This app provides an authentication backend for use in assigning permissions to individual objects (maps and layers).

`settings.py` Entries

LOCKDOWN_GEONODE Default: `False`

By default, the GeoNode application allows visitors to view most pages without being authenticated. Set `LOCKDOWN_GEONODE = True` to require a user to be authenticated before viewing the application.

AUTH_EXEMPT_URLS Default: `()` (Empty tuple)

A tuple of URL patterns that the user can visit without being authenticated. This setting has no effect if `LOCKDOWN_GEONODE` is not `True`. For example, `AUTH_EXEMPT_URLS = ('/maps',)` will allow unauthenticated users to browse maps.

Template Tags

`geonode_media` <media_name> Accesses entries in `MEDIA_LOCATIONS` without requiring the view to explicitly add it to the template context. Example usage:

```
{% include geonode_media %}
{% geonode_media "ext_base" %}
```

`has_obj_perm` <user> <obj> <permission> Checks whether the user has the specified permission on an object.

```
{% has_obj_perm user obj "app.view_thing" as can_view_thing %}
```

`geonode.people` - User account registration and profile management

This app provides functionality related to user accounts and user profiles.

User authentication is based on `django-allauth`, with some geonode-specific customizations. Users can register to a geonode instance by either creating a new account or by using one of their social providers, such as linkedin, facebook, etc.

User invitations are leveraged via **‘django-invitations’**.

Adding authentication with a social provider

Configuring geonode user authentication with a social provider is a multi-step process:

- Create an application on the provider's developer site. For example, for

linkedin:

- Go to <https://developer.linkedin.com/apps> and select `Create Application`
- Proceed to create an application, referring to [django-allauth's documentation](#) where needed. Be sure to select the `r_basicprofile` and `r_emailaddress` application permissions. These will enable geonode to access some information regarding the user's linkedin public profile. Once created, take note of the `Client ID` and `Client Secret` parameters which linkedin assigns to your new application
- Follow the [post-install](#) instructions on django-allauth's documentation. These basically instruct you to:
 - include the relevant app in geonode's `INSTALLED_APPS` setting for the social provider that you are using. For linkedin this means adding `allauth.socialaccount.providers.linkedin_oauth2`.
 - Access geonode's admin interface and add a new `Social App` for the provider. Add the previously gathered `Client id` and `Secret key` parameters that were obtained when creating a linkedin application.
 - Optionally add relevant value to geonode's `SOCIALACCOUNT_PROVIDERS` setting. Consult django-allauth's provider-specific documentation for more information on this setting
- Optionally add a profile extractor entry to geonode's `SOCIALACCOUNT_PROFILE_EXTRACTORS` setting. This setting is specific to geonode and is not part of django-allauth. The expected value is a dictionary with the provider's id as key and the python path to a data extractor class as the value. Example:

```
# settings.py
SOCIALACCOUNT_PROFILE_EXTRACTORS = {
    "linkedin_oauth2": "geonode.people.profileextractors.LinkedInExtractor"
}
```

A profile extractor class can retrieve information from each social account provider and use that to enhance a user's profile information on geonode. Look into the `geonode.people.profileextractors` module on geonode's source code for more information.

settings.py entries

ACCOUNT_OPEN_SIGNUP Default: `True`

Whether the geonode site allows new users to register for an account. When set to `False` new users cannot register, they must be created by a staff member

ACCOUNT_APPROVAL_REQUIRED Default: `False`

Whether new user registrations must be manually approved. When set to `True`, the site's staff must manually approve every new account. The approval flow goes like this:

- User registers in the site (either with a local account or by using a social login);
- Staff users receive an email notification that a new user is requesting access;
- Staff users manually approve the user by using geonode's administration interface;
- User receives an e-mail notification when the account has been approved;

- User can now login to the geonode site.

ACCOUNT_ADAPTER Default: `geonode.people.adapters.LocalAccountAdapter`

This is a [django-allauth setting](#). It allows specifying a custom class to handle authentication for local accounts.

ACCOUNT_CONFIRM_EMAIL_ON_GET Default: `True`

This is a [django-allauth setting](#). It allows specifying the HTTP method used when confirming e-mail addresses.

ACCOUNT_EMAIL_REQUIRED Default: `True`

This is a [django-allauth setting](#). Controls whether the user is required to provide an e-mail address upon registration

ACCOUNT_EMAIL_VERIFICATION Default: `optional`

This is a [django-allauth setting](#).

SOCIALACCOUNT_ADAPTER Default: `geonode.people.adapters.SocialAccountAdapter`

This is a [django-allauth setting](#). It allows specifying a custom class to handle authentication for social accounts.

SOCIALACCOUNT_PROVIDERS Default: `None`

This is a [django-allauth setting](#). It should be a dictionary with provider specific settings

SOCIALACCOUNT_PROFILE_EXTRACTORS Default: `None`

A dictionary with provider ids as keys and path to custom profile extractor classes as values.

Django's error templates

GeoNode customizes some of Django's default error templates.

500.html

If no custom handler for 500 errors is set up in `urls.py`, Django will call `django.views.defaults.server_error` which expects a `500.html` file in the root of the templates directory. In GeoNode, we have put a template that does not inherit from anything as `500.html` and because most of Django's machinery is down when an INTERNAL ERROR (500 code) is encountered the use of template tags should be avoided.

3rd party apps

pinax.notifications

This application enables users to receive notifications from specific events within GeoNode. For user-specific configuration, see [Setting notification preferences](#). For settings, see [User notifications settings](#).

3.3.1.4.2 JavaScript in GeoNode

GeoNode provides a number of facilities for interactivity in the web browser built on top of several high-quality JavaScript frameworks:

- [Bootstrap](#) for GeoNode's front-end user interface and common user interaction.
- [Bower](#) for GeoNode's front-end package management.

- [ExtJS](#) for component-based UI construction and data access
- [OpenLayers](#) for interactive mapping and other geospatial operations
- [GeoExt](#) for integrating ExtJS with OpenLayers
- [Grunt](#) for front-end task automation.
- [GXP](#) for providing some higher-level application building facilities on top of GeoExt, as well as improving integration with GeoServer.
- [jQuery](#) to abstract JavaScript manipulation, event handling, animation and XMLHttpRequest.

GeoNode uses application-specific modules to handle pages and services that are unique to GeoNode. This framework includes:

- A [GeoNode mixin](#) class that provides GeoExplorer with the methods needed to properly function in GeoNode. The class is responsible for checking permissions, retrieving and submitting the [CSRF token](#), and user authentication.
- A [search module](#) responsible for the GeoNode's site-wide search functionality.
- An [upload and status module](#) to support file uploads.
- [Template files](#) for generating commonly used HTML sections.
- A [front-end testing module](#) to test GeoNode JavaScript.

The following concepts are particularly important for developing on top of the GeoNode's JavaScript framework.

- **Components** - Ext components handle most interactive functionality in "regular" web pages. For example, the scrollable/sortable/filterable table on the default Search page is a Grid component. While GeoNode does use some custom components, familiarity with the idea of Components used by ExtJS is applicable in GeoNode development.
- **Viewers** - Viewers display interactive maps in web pages, optionally decorated with Ext controls for toolbars, layer selection, etc. Viewers in GeoNode use the GeoExplorer base class, which builds on top of GXP's Viewer to provide some common functionality such as respecting site-wide settings for background layers. Viewers can be used as components embedded in pages, or they can be full-page JavaScript applications.
- **Controls** - Controls are tools for use in OpenLayers maps (such as a freehand control for drawing new geometries onto a map, or an identify control for getting information about individual features on a map.) GeoExt provides tools for using these controls as ExtJS "Actions" - operations that can be invoked as buttons or menu options or associated with other events.

3.3.1.4.3 Settings

Here's a list of settings available in GeoNode and their default values. This includes settings for some external applications that GeoNode depends on.

Documents settings

Here's a list of settings available for the Documents app in GeoNode.

ALLOWED_DOCUMENT_TYPES

Default: `['doc', 'docx', 'xls', 'xlsx', 'pdf', 'zip', 'jpg', 'jpeg', 'tif', 'tiff', 'png', 'gif', 'txt']`

A list of acceptable file extensions that can be uploaded to the Documents app.

MAX_DOCUMENT_SIZE

Default: 2

Metadata settings

CATALOGUE

A dict with the following keys:

- ENGINE: The CSW backend (default is `geonode.catalogue.backends.pycsw_local`)
- URL: The FULLY QUALIFIED base URL to the CSW instance for this GeoNode
- USERNAME: login credentials (if required)
- PASSWORD: login credentials (if required)

`pycsw` is the default CSW enabled in GeoNode. `pycsw` configuration directives are managed in the `PYCSW` entry.

PYCSW

A dict with `pycsw`'s configuration. Of note are the sections `metadata:main` to set CSW server metadata and `metadata:inspire` to set INSPIRE options. Setting `metadata:inspire['enabled']` to `true` will enable INSPIRE support. Server level configurations can be overridden in the `server` section. See <http://docs.pycsw.org/en/latest/configuration.html> for full `pycsw` configuration details.

MODIFY_TOPICCATEGORY

Default: `False`

Metadata Topic Categories list should not be modified, as it is strictly defined by ISO (See: <http://www.isotc211.org/2005/resources/Codelist/gmxCodellists.xml> and check the `<CodeListDictionary gml:id="MD_MD_TopicCategoryCode">` element).

Some customization is still possible by changing the `is_choice` and the GeoNode description fields.

In case it is absolutely necessary to add/delete/update categories, it is possible to set the `MODIFY_TOPICCATEGORY` setting to `True`.

Maps settings

DEFAULT_MAP_BASE_LAYER

The name of the background layer to include in newly created maps.

DEFAULT_MAP_CENTER

Default: (0, 0)

A 2-tuple with the latitude/longitude coordinates of the center-point to use in newly created maps.

DEFAULT_MAP_ZOOM

Default: 0

The zoom-level to use in newly created maps. This works like the OpenLayers zoom level setting; 0 is at the world extent and each additional level cuts the viewport in half in each direction.

MAP_BASELAYERS

Default:

```

MAP_BASELAYERS = [{
  "source": {
    "ptype": "gxp_wmssource",
    "url": OGC_SERVER['default']['PUBLIC_LOCATION'] + "wms",
    "restUrl": "/gs/rest"
  }, {
    "source": {"ptype": "gxp_olsource"},
    "type": "OpenLayers.Layer",
    "args": ["No background"],
    "visibility": False,
    "fixed": True,
    "group": "background"
  }, {
    "source": {"ptype": "gxp_osmsource"},
    "type": "OpenLayers.Layer.OSM",
    "name": "mapnik",
    "visibility": False,
    "fixed": True,
    "group": "background"
  }, {
    "source": {"ptype": "gxp_mapquestsource"},
    "name": "osm",
    "group": "background",
    "visibility": True
  }, {
    "source": {"ptype": "gxp_mapquestsource"},
    "name": "naip",
    "group": "background",
    "visibility": False
  }, {
    "source": {"ptype": "gxp_mapboxsource"},
  }, {
    "source": {"ptype": "gxp_olsource"},
    "type": "OpenLayers.Layer.WMS",
    "group": "background",
    "visibility": False,
    "fixed": True,

```

(continues on next page)

(continued from previous page)

```

    "args": [
      "bluemarble",
      "http://maps.opengeo.org/geowebcache/service/wms",
      {
        "layers": ["bluemarble"],
        "format": "image/png",
        "tiled": True,
        "tilesOrigin": [-20037508.34, -20037508.34]
      },
      {"buffer": 0}
    ]
  }
}

```

A list of dictionaries that specify the default map layers.

Specific settings for map API providers (if they are not set those base maps will not be available):

- ALT_OSM_BASEMAPS set this variable to True if you want additional OSM basemaps
- CARTODB_BASEMAPS set this variable to True if you want CartoDB basemaps
- STAMEN_BASEMAPS set this variable to True if you want stamen basemaps
- THUNDERFOREST_BASEMAPS set this variable to True if you want Thunderforest basemaps
- MAPBOX_ACCESS_TOKEN set this variable to your Mapbox public token
- BING_API_KEY set this variable to your BING Map Key value

GEONODE_CLIENT_LAYER_PREVIEW_LIBRARY

Default: "geoext"

The library to use for display preview images of layers. The library choices are:

- "leaflet"
- "geoext"
- "react"

More instructions can be found *here* <<https://github.com/GeoNode/geonode-client/blob/master/README.md>>

OGC_SERVER

Default: {} (Empty dictionary)

A dictionary of OGC servers and their options. The main server should be listed in the 'default' key. If there is no 'default' key or if the OGC_SERVER setting does not exist GeoNode will raise an Improperly Configured exception. Below is an example of the OGC_SERVER setting:

```

OGC_SERVER = {
  'default' : {
    'LOCATION' : 'http://localhost:8080/geoserver/',
    'USER' : 'admin',
    'PASSWORD' : 'geoserver',
  }
}

```

BACKEND

Default: "geonode.geoserver"

The OGC server backend to use. The backend choices are:

- 'geonode.geoserver'

BACKEND_WRITE_ENABLED

Default: True

Specifies whether the OGC server can be written to. If False, actions that modify data on the OGC server will not execute.

DATASTORE

Default: '' (Empty string)

An optional string that represents the name of a vector datastore that GeoNode uploads are imported into. In order to support vector datastore imports there also needs to be an entry for the datastore in the `DATABASES` dictionary with the same name. Example:

```
OGC_SERVER = {
    'default' : {
        'LOCATION' : 'http://localhost:8080/geoserver/',
        'USER' : 'admin',
        'PASSWORD' : 'geoserver',
        'DATASTORE': 'geonode_imports'
    }
}

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': 'development.db',
    },
    'geonode_imports' : {
        'ENGINE': 'django.contrib.gis.db.backends.postgis',
        'NAME': 'geonode_imports',
        'USER' : 'geonode_user',
        'PASSWORD' : 'a_password',
        'HOST' : 'localhost',
        'PORT' : '5432',
    }
}
```

GEOGIG_ENABLED

Default: False

A boolean that represents whether the OGC server supports GeoGig datastores.

GEONODE_SECURITY_ENABLED

Default: True

A boolean that represents whether GeoNode's security application is enabled.

LOCATION

Default: "http://localhost:8080/geoserver/"

A base URL from which GeoNode can construct OGC service URLs. If using GeoServer you can determine this by visiting the GeoServer administration home page without the /web/ at the end. For example, if your GeoServer administration app is at <http://example.com/geoserver/web/>, your server's location is <http://example.com/geoserver>.

MAPFISH_PRINT_ENABLED

Default: True

A boolean that represents whether the MapFish printing extension is enabled on the server.

PASSWORD

Default: 'geoserver'

The administrative password for the OGC server as a string.

PG_GEOGIG

Default: False

A boolean that represents whether GeoNode will use a Postgres database as a backend for GeoGig stores. When set to true, the DATASTORE field must be set to a Postgres database which will be used.

PRINT_NG_ENABLED

Default: True

A boolean that represents whether printing of maps and layers is enabled.

PUBLIC_LOCATION

Default: "http://localhost:8080/geoserver/"

The URL used to in most public requests from GeoNode. This settings allows a user to write to one OGC server (the LOCATION setting) and read from a separate server or the PUBLIC_LOCATION.

USER

Default: 'admin'

The administrative username for the OGC server as a string.

WMST_ENABLED

Default: `False`

Not implemented.

WPS_ENABLED

Default: `False`

Not implemented.

TIMEOUT

Default: `10`

The maximum time, in seconds, to wait for the server to respond.

SITEURL

Default: `'http://localhost:8000/'`

A base URL for use in creating absolute links to Django views and generating links in metadata.

Proxy settings

PROXY_ALLOWED_HOSTS

Default: `()` (Empty tuple)

A tuple of strings representing the host/domain names that GeoNode can proxy requests to. This is a security measure to prevent an attacker from using the GeoNode proxy to render malicious code or access internal sites.

Values in this tuple can be fully qualified names (e.g. `'www.geonode.org'`), in which case they will be matched against the request's Host header exactly (case-insensitive, not including port). A value beginning with a period can be used as a subdomain wildcard: `.geonode.org` will match `geonode.org`, `www.geonode.org`, and any other subdomain of `geonode.org`. A value of `'*'` will match anything and is not recommended for production deployments.

PROXY_URL

Default `/proxy/?url=`

The URL to a proxy that will be used when making client-side requests in GeoNode. By default, the internal GeoNode proxy is used but administrators may favor using their own, less restrictive proxies.

Search settings

DEFAULT_SEARCH_SIZE

Default: `10`

An integer that specifies the default search size when using `geonode.search` for querying data.

API settings

API_LIMIT_PER_PAGE

Default: 20

Number of items returned by the API. 0 equals no limit

API_INCLUDE_REGIONS_COUNT

Default: False

Specifies if to include facets count for regions.

Security settings

AUTH_EXEMPT_URLS

Default: () (Empty tuple)

A tuple of URL patterns that the user can visit without being authenticated. This setting has no effect if `LOCKDOWN_GEONODE` is not True. For example, `AUTH_EXEMPT_URLS = ('/maps',)` will allow unauthenticated users to browse maps.

LOCKDOWN_GEONODE

Default: False

By default, the GeoNode application allows visitors to view most pages without being authenticated. If this is set to True users must be authenticated before accessing URL routes not included in `AUTH_EXEMPT_URLS`.

RESOURCE_PUBLISHING

Default: True

By default, the GeoNode application allows GeoNode staff members to publish/unpublish resources. By default resources are published when created. When this settings is set to True the staff members will be able to unpublish a resource (and eventually publish it back).

Social settings

SOCIAL_BUTTONS

Default: True

A boolean which specifies whether the social media icons and JavaScript should be rendered in GeoNode.

SOCIAL_ORIGINS

Default:

```
SOCIAL_ORIGINS = [{
    "label": "Email",
    "url": "mailto:?subject={name}&body={url}",
    "css_class": "email"
}, {
    "label": "Facebook",
    "url": "http://www.facebook.com/sharer.php?u={url}",
    "css_class": "fb"
}, {
    "label": "Twitter",
    "url": "https://twitter.com/share?url={url}",
    "css_class": "tw"
}, {
    "label": "Google +",
    "url": "https://plus.google.com/share?url={url}",
    "css_class": "gp"
}]
```

A list of dictionaries that is used to generate the social links displayed in the Share tab. For each origin, the name and URL format parameters are replaced by the actual values of the ResourceBase object (layer, map, document).

CKAN_ORIGINS

Default:

```
CKAN_ORIGINS = [{
    "label": "Humanitarian Data Exchange (HDX)",
    "url": "https://data.hdx.rwlab.org/dataset/new?title={name}&notes={abstract}",
    "css_class": "hdx"
}]
```

A list of dictionaries that is used to generate the links to CKAN instances displayed in the Share tab. For each origin, the name and abstract format parameters are replaced by the actual values of the ResourceBase object (layer, map, document). This is not enabled by default. To enable, uncomment the following line: `SOCIAL_ORIGINS.extend(CKAN_ORIGINS)`.

TWITTER_CARD

Default:: True

A boolean that specifies whether Twitter cards are enabled.

TWITTER_SITE

Default:: '@GeoNode'

A string that specifies the site to for the twitter:site meta tag for Twitter Cards.

TWITTER_HASHTAGS

Default:: ['geonode']

A list that specifies the hashtags to use when sharing a resource when clicking on a social link.

OPENGRAPH_ENABLED

Default:: True

A boolean that specifies whether Open Graph is enabled. Open Graph is used by Facebook and Slack.

Upload settings

GEOGIG_DATASTORE_NAME

Default: None

A string with the default GeoGig datastore name. This value is only used if no GeoGig datastore name is provided when data is uploaded but it must be populated if your deployment supports GeoGig.

UPLOADER

Default:

```
{
  'BACKEND' : 'geonode.rest',
  'OPTIONS' : {
    'TIME_ENABLED': False,
    'GEOGIG_ENABLED': False,
  }
}
```

A dictionary of Uploader settings and their values.

BACKEND

Default: 'geonode.rest'

The uploader backend to use. The backend choices are:

- 'geonode.importer'
- 'geonode.rest'

The importer backend requires the GeoServer importer extension to be enabled and is required for uploading data into GeoGig datastores.

OPTIONS

Default:


```
'OPTIONS' : {  
  'TIME_ENABLED': False,  
  'GEOGIG_ENABLED': False,  
}
```

TIME_ENABLED

Default: `False`

A boolean that specifies whether the upload should allow the user to enable time support when uploading data.

GEOGIG_ENABLED

Default: `False`

A boolean that specifies whether the uploader should allow the user to upload data into a GeoGig datastore.

User Account settings

THEME_ACCOUNT_CONTACT_EMAIL

Default: `'admin@example.com'`

This email address is added to the bottom of the password reset page in case users have trouble unlocking-locking their account.

ACCOUNT_OPEN_SIGNUP

Default: `True`

Whether the geonode site allows new users to register for an account. When set to `False` new users cannot register, they must be created by a staff member

ACCOUNT_APPROVAL_REQUIRED

Default: `False`

Whether new user registrations must be manually approved. When set to `True`, the site's staff must manually approve every new account. The approval flow goes like this:

- User registers in the site (either with a local account or by using a social login);
- Staff users receive an email notification that a new user is requesting access;
- Staff users manually approve the user by using geonode's administration interface;
- User receives an e-mail notification when the account has been approved;
- User can now login to the geonode site.

ACCOUNT_ADAPTER

Default: `geonode.people.adapters.LocalAccountAdapter`

This is a [django-allauth setting](#). It allows specifying a custom class to handle authentication for local accounts.

ACCOUNT_CONFIRM_EMAIL_ON_GET

Default: `True`

This is a [django-allauth setting](#). It allows specifying the HTTP method used when confirming e-mail addresses.

ACCOUNT_EMAIL_REQUIRED

Default: `True`

This is a [django-allauth setting](#). Controls whether the user is required to provide an e-mail address upon registration

ACCOUNT_EMAIL_VERIFICATION

Default: `optional`

This is a [django-allauth setting](#).

SOCIALACCOUNT_ADAPTER

Default: `geonode.people.adapters.SocialAccountAdapter`

This is a [django-allauth setting](#). It allows specifying a custom class to handle authentication for social accounts.

SOCIALACCOUNT_PROVIDERS

Default: `None`

This is a [django-allauth setting](#). It should be a dictionary with provider specific settings

SOCIALACCOUNT_PROFILE_EXTRACTORS

Default: `None`

A dictionary with lowercase provider names as keys and path to custom profile extractor classes as values.

User notifications settings

Note: `PINAX_*` settings are relevant if you have `pinax.notifications` app added to `INSTALLED_APPS`.

Note: user configuration is described [Setting notification preferences](#).

Note: Full list of Pinax-notifications app settings is available in [Pinax notifications docs](#).

PINAX_NOTIFICATIONS_QUEUE_ALL

Default: *False*

A boolean that specifies if user notifications should be queued and send outside main web application process. If set to *True*, additional sending script, *python manage.py emit_notices* should be called periodically.

PINAX_NOTIFICATIONS_BACKENDS

Default:

```
[
("email", "pinax.notifications.backends.email.EmailBackend"),
]
```

List of backends (channels) that delivers messages to recipients.

Download settings

DOWNLOAD_FORMATS_METADATA

Specifies which metadata formats are available for users to download.

Default:

```
DOWNLOAD_FORMATS_METADATA = [
    'Atom', 'DIF', 'Dublin Core', 'ebRIM', 'FGDC', 'ISO',
]
```

DOWNLOAD_FORMATS_VECTOR

Specifies which formats for vector data are available for users to download.

Default:

```
DOWNLOAD_FORMATS_VECTOR = [
    'JPEG', 'PDF', 'PNG', 'Zipped Shapefile', 'GML 2.0', 'GML 3.1.1', 'CSV',
    'Excel', 'GeoJSON', 'KML', 'View in Google Earth', 'Tiles',
]
```

DOWNLOAD_FORMATS_RASTER

Specifies which formats for raster data are available for users to download.

Default:

```
DOWNLOAD_FORMATS_RASTER = [  
    'JPEG', 'PDF', 'PNG', 'Tiles',  
]
```

Contrib settings

EXIF_ENABLED

Default: `False`

A boolean that specifies whether the Exif contrib app is enabled. If enabled, metadata is generated from Exif tags when documents are uploaded.

GEOTIFF_IO_ENABLED

Default: `False`

A boolean that specifies whether the GeoTIFF.io contrib feature is enabled. If enabled, an ‘Analyze with GeoTIFF.io’ button is added to the `layer_detail` page.

GEOTIFF_IO_BASE_URL

Default: `https://app.geotiff.io`

A string that specifies what instance of GeoTIFF.io should be opened when the ‘Analyze with GeoTIFF.io’ button is clicked.

NLP_ENABLED

Default: `False`

A boolean that specifies whether the NLP (Natural Language Processing) contrib app is enabled. If enabled, NLP (specifically MITIE) is used to infer additional metadata from uploaded documents to help fill metadata gaps.

NLP_LOCATION_THRESHOLD

Default: `1.0`

A float that specifies the threshold for location matches.

NLP_LIBRARY_PATH

Default: `'/opt/MITIE/mitielib'`

A string that specifies the location of the MITIE library

NLP_MODEL_PATH

Default:: `'/opt/MITIE/MITIE-models/english/ner_model.dat'`

A string that specifies the location of the NER (Named Entity Resolver). MITIE comes with English and Spanish NER models. Other models can be trained.

SLACK_ENABLED

Default: `False`

A boolean that specifies whether the Slack contrib app is enabled. If enabled, GeoNode will send messages to the slack channels specified in `SLACK_WEBHOOK_URLS` when a document is uploaded, metadata is updated, etc. Coverage of events is still incomplete.

SLACK_WEBHOOK_URLS

A list that specifies the URLs to post Slack messages to. Each URL is for a different channel. The default URL should be replaced when slack integration is enabled.

Default:

```
SLACK_WEBHOOK_URLS = [  
    "https://hooks.slack.com/services/T000/B000/XX"  
]
```

Amazon Web Services Settings

S3_STATIC_ENABLED

Default: `False`

A boolean that specifies whether GeoNode's static files will be served through an S3 bucket. Set through the environment variable `S3_STATIC_ENABLED`.

S3_MEDIA_ENABLED

Default: `False`

A boolean that specifies whether GeoNode's media files will be served through an S3 bucket. Set through the environment variable `S3_MEDIA_ENABLED`.

AWS_BUCKET_NAME

The name of the S3 bucket GeoNode will pull static and/or media files from. Set through the environment variable `S3_BUCKET_NAME`.

AWS_STORAGE_BUCKET_NAME

The name of the S3 bucket GeoNode will pull static and/or media files from. Set through the environment variable `S3_BUCKET_NAME`.

AWS_ACCESS_KEY_ID

The access key for the S3 bucket GeoNode will pull static and/or media files from. Set through the environment variable `AWS_ACCESS_KEY_ID`.

AWS_SECRET_ACCESS_KEY

The secret access key for the S3 bucket GeoNode will pull static and/or media files from. Set through the environment variable `AWS_SECRET_ACCESS_KEY`.

AWS_QUERYSTRING_AUTH

Default: `False`

Generate an S3 auth querystring.

3.3.1.4.4 GeoSites: GeoNode Multi-Tenancy

GeoSites is a way to run multiple websites with a single instance of GeoNode. Each GeoSite can have different templates, apps, and data permissions but share a single database (useful for sharing users and data layers), GeoServer, and CSW. This is useful when multiple websites are desired to support different sets of users, but with a similar set of data and overall look and feel of the sites. Users can be given permission to access multiple sites if needed, which also allows administrative groups can be set up to support all sites with one account.

Master Website

A GeoSites installation uses a ‘master’ GeoNode website that has some additional administrative pages for doing data management. Layers, Maps, Documents, Users, and Groups can all be added and removed from different sites. Users can be given access to any number of sites, and data may appear on only a single site, or all of them. Additionally, if desired, any or all of the Django apps installed on the other sites can be added to the master site to provide a single administrative interface that gives full access to all apps. The master site need not be accessible from the outside so that it can be used as an internal tool to the organization.

Users created on a particular site are created with access to just that site. Data uploaded to a particular site is given permission on that site as well as the master site. Any further adjustments to site-based permissions must be done from the master site.

Database

The master site, and all of the individual GeoSites, share a single database. Objects, including users, groups, and data layers, all appear within the database but an additional sites table indicates which objects have access to which sites. The geospatial data served by GeoServer (e.g., from PostGIS) can exist in the database like normal, since GeoServer will authenticate against GeoNode, which will use it’s database to determine permissions based on the object, current user, and site.

GeoServer

A single GeoServer instance is used to serve data to all of the GeoSites. To keep data organized each site specifies a default workspace (DEFAULT_WORKSPACE) that GeoServer will use to partition the data depending on which site uploaded the data. The workspaces themselves don't have any impact on permissions, since data can be added and removed from different sites, however it provides at least some organization of the data based on the initial site.

Data that is common to all sites can be added to the master site which will appear in the generic 'geonode' workspace.

Settings Files and Templates

A key component in managing multiple sites is keeping data organized and using a structured series of settings files so that common settings can be shared and only site specific settings are separated out. It is also best to import the default GeoNode settings from the GeoNode installation. This prevents the settings from having to be manually upgraded if there is any default change the GeoNode settings.

Settings which are common to all GeoSites, but differ from the default GeoNode, are separated into a master_settings.py file. Then, each individual site has settings file which imports from the master site and will then only need to specify a small selection that make that site unique, such as:

- SITE_ID: Each one is unique, the master site should have a SITE_ID of 1.
- SITENAME
- SITEURL
- ROOT_URLCONF: This may be optional. The master site url.conf can be configured to automatically import the urls.py of all SITE_APPS, so a different ROOT_URLCONF is only needed if there are further differences.
- SITE_APPS: Containing the site specific apps
- App settings: Any further settings required for the above sites
- Other site specific settings, such as REGISTRATION_OPEN

A GeoSite therefore has three layers of imports, which is used for settings as well as the search path for templates. First it uses the individual site files, then the master GeoSite, then default GeoNode. These are specified via variables defined in settings:

- SITE_ROOT: The directory where the site specific settings and files are located (templates, static)
- PROJECT_ROOT: The top-level directory of all the GeoSites which should include the global settings file as well as template and static files
- GEONODE_ROOT: The GeoNode directory.

The TEMPLATE_DIRS, and STATICFILES_DIRS will then include all three directories as shown:

```
TEMPLATE_DIRS = (
    os.path.join(SITE_ROOT, 'templates/'),
    os.path.join(PROJECT_ROOT, 'templates/'), # files common to all sites
    os.path.join(GEONODE_ROOT, 'templates/')
)

STATICFILES_DIRS = (
    os.path.join(SITE_ROOT, 'static/'),
    os.path.join(PROJECT_ROOT, 'static/'),
    os.path.join(GEONODE_ROOT, 'static/')
)
```

At the end of the settings_global.py the following variables will be set based on site specific settings:

```
STATIC_URL = os.path.join(SITEURL, 'static/')
GEONODE_CLIENT_LOCATION = os.path.join(STATIC_URL, 'geonode/')
GEOSERVER_BASE_URL = SITEURL + 'geoserver/'
if SITE_APPS:
    INSTALLED_APPS += SITE_APPS
```

Templates and Static Files

As mentioned above for each website there will be three directories used for template and static files. The first template file found will be the one used so templates in the `SITE_ROOT/templates` directory will override those in `PROJECT_ROOT/templates`, which will override those in `GEONODE_ROOT/templates`.

Static files work differently because (at least on a production server) they are collected and stored in a single location. Because of this care must be taken to avoid clobbering of files between sites, so each site directory should contain all static files in a subdirectory with the name of the site (e.g., `static/siteA/logo.png`)

The location of the proper static directory can then be found in the templates syntax such as:

```
{{ STATIC_URL }}{{ SITENAME|lower }}/logo.png
```

Permissions by Site

By default GeoNode is publicly available. In the case of GeoSites, new data will be publicly available, but only for the site it was added to, and the master site (all data is added to the master site).

Adding New Sites

A management command exists to easily create a new site. This will create all the needed directories, as well as a site specific settings file. The command may also create a website configuration file.

3.3.1.5 Supported Browsers

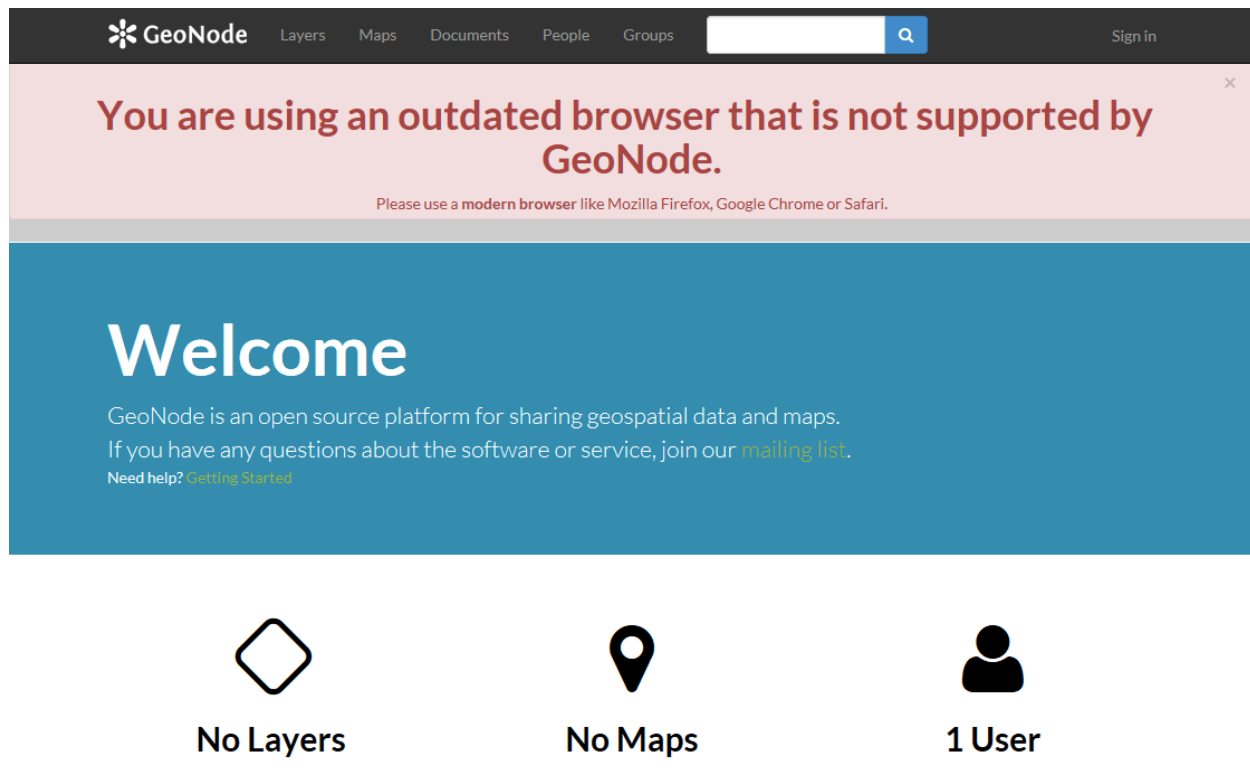
GeoNode is known to work on all modern web browsers. This list includes (but is not limited to):

- Google Chrome.
- Apple Safari.
- Mozilla Firefox.
- Microsoft Edge.
- Microsoft Internet Explorer.

The vast majority of GeoNode developers do their work with Google Chrome.

3.3.1.5.1 Internet Explorer

Versions of Microsoft Internet Explorer older than 10 exhibit known issues when used to browse a GeoNode site. As such a message is displayed warning the user that they should upgrade their browser.



3.3.1.5.2 Testing on Internet Explorer

When working on front end code, developers should take care to test carefully with Microsoft Internet Explorer to ensure that the features they are working on do indeed work correctly and on this browser. It is good practice to test on all browsers available, but the use of modern front end libraries like bootstrap and jQuery make it much more likely code will work across browsers seamlessly.

In order to test on Internet Explorer, developers can use the [Modern IE](#) site to download virtual machines for use in [Oracle VM Virtual Box](#).

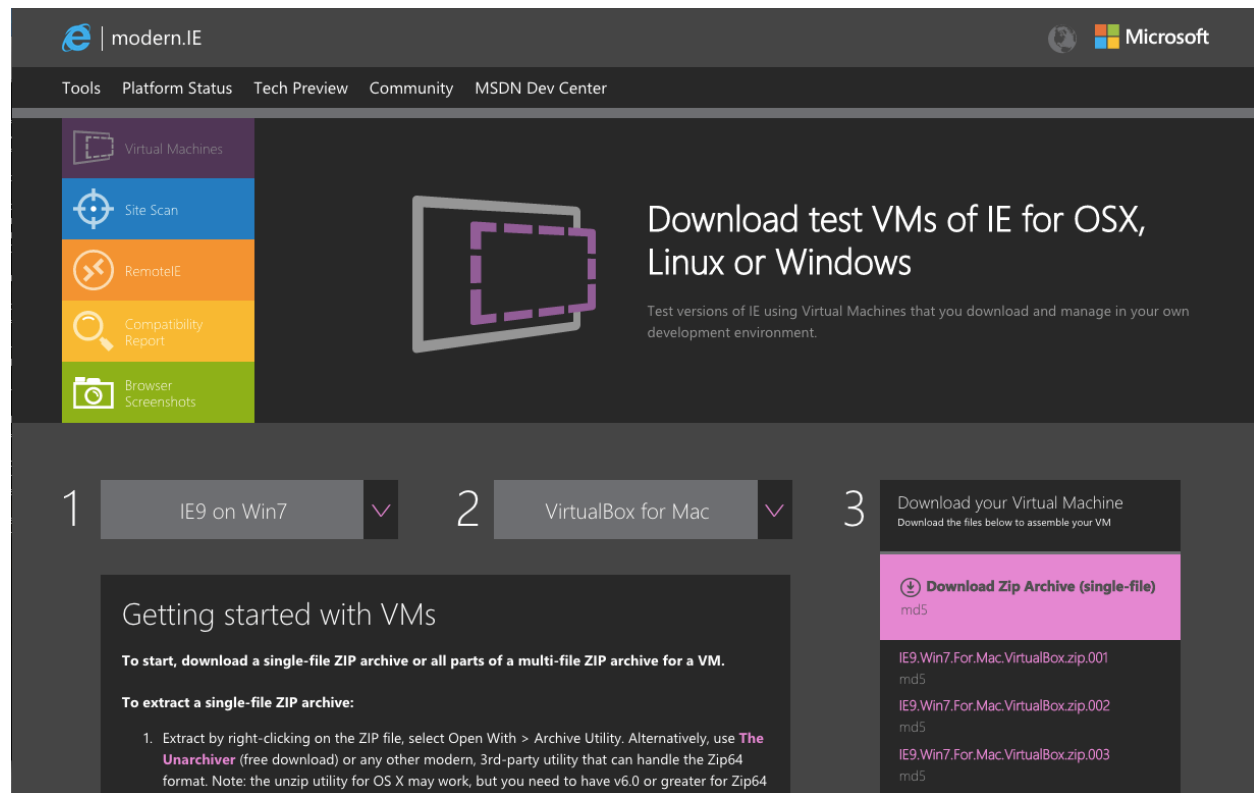
Once the VM is downloaded, follow the instructions to configure it in your VirtualBox setup.

After the VM is setup, you can access your development instance of GeoNode by visiting the IP address of your host machine or on the bridged interface (usually 10.0.2.2) and begin your testing.

3.3.1.6 WorldMap

By using the WorldMap optional application, GeoNode is extended with the following additional features:

- **a customized GeoExplorer viewer**
 - the table of contents is hierarchical with layer categories. When a layer is added a new category containing the layer is added to the table of contents. If the category is already in the table of contents, then the layer is added to it. By default the category is the same as the layer's topic category, but that can be renamed by right clicking on it
 - the "Add Layers" dialog comes with a "Search" tab which uses Hypermap Registry (Hypermap) as a catalogue of remote and local layers. Hypermap is a requirement when using the WorldMap contrib application



- a gazetteer application: it is possible to add a given layer to a gazetteer. The gazetteer can be checked using the map client. When a layer is part of the gazetteer it is possible to include it in a general gazetteer or in a specific project one. It is possible to search place names in the gazetteer by date range, in which case it is necessary to specify the layer attributes for the start and end depict dates

3.3.1.6.1 Installation

Requirements

We are assuming a Ubuntu 16.04.1 LTS development environment, but these instructions can be adapted to any recent Linux distributions:

```
# Install Ubuntu dependencies
sudo apt-get update
sudo apt-get install python-virtualenv python-dev libxml2 libxml2-dev libxslt1-dev
↪zlib1g-dev libjpeg-dev libpq-dev libgdal-dev git default-jdk postgresql postgis

# Install Java 8 (needed by latest GeoServer 2.13)
sudo apt-add-repository ppa:webupd8team/java
sudo apt-get update
sudo apt-get install oracle-java8-installer
```

Virtual environment creation and installation of Python packages

Create and activate the virtual environment:



```
cd ~
virtualenv --no-site-packages env
. env/bin/activate
```

Now install GeoNode from source code:

```
git clone -b https://github.com/geonode/geonode.git
cd geonode
pip install -r requirements.txt
pip install pygdal==1.11.3.3
pip install -e .
paver setup
paver sync
```

Set the following environment variables as needed (change `SITE_NAME` and `SERVER_IP` s needed. Also `HYPERMAP_REGISTRY_URL` and `SOLR_URL` may be different). Even better, create a file and source it:

```
export USE_WORLDMAP=True
export SITE_NAME=worldmap
export SERVER_IP=128.31.22.73
export PG_USERNAME=worldmap
export PG_PASSWORD=worldmap
export PG_WORLDMAP_DJANGO_DB=worldmap
export PG_WORLDMAP_UPLOADS_DB=wmdata
export OWNER=$PG_USERNAME
export ALLOWED_HOSTS="localhost, $SERVER_IP, "
export GEOSERVER_LOCATION=http://localhost:8080/geoserver/
export GEOSERVER_PUBLIC_LOCATION=http://$SERVER_IP/geoserver/
export SOLR_URL =http://localhost:8983/solr/hypermap/select/
export HYPERMAP_REGISTRY_URL =http://localhost:8001
export MAPPROXY_URL=http://localhost:8001
```

You can install GeoNode WorldMap in two different ways:

- 1) By installing GeoNode itself
- 2) By using the recommended way of a geonode-project

GeoNode/WorldMap without a geonode-project

Copy the included `local_settings.py` file and customize it to your needs:

```
cp local_settings.py.worldmap.sample local_settings.py
```

GeoNode/WorldMap with a geonode-project

You will use a geonode-project in order to separate the customization of your instance from GeoNode.

Create your geonode project by using the WorldMap geonode-project as a template (<https://github.com/cga-harvard/geonode-project>). Rename it to something meaningful (for example your web site name):

```
cd ~
django-admin startproject $SITE_NAME --template=https://github.com/cga-harvard/
↪geonode-project/archive/master.zip -epy,rst
cd $SITE_NAME
```

Create a `local_settings.py` by copying the included template:

```
cp $SITE_NAME/local_settings.py.sample $SITE_NAME/local_settings.py
make build
paver setup
```

Start the Server

Start GeoNode with Worldmap using pavement:

```
python manage.py runserver 0.0.0.0:8000
paver start_geoserver
```

To upload layers you can login with the default GeoNode administrative account:

user: admin password: admin

Configuring instance for production

Please follow best practices suggested by GeoNode documentation:

http://docs.geonode.org/en/master/tutorials/advanced/geonode_production/

Remember to add the ip of your server in `ALLOWED_HOSTS` in the `local_settings.py` file:

```
ALLOWED_HOSTS = ['localhost', '128.31.22.73', ]
```

3.3.1.6.2 Hypermap Registry

GeoNode with the WorldMap contribute module requires a Hypermap Registry (Hypermap) running instance.

You can install Hypermap by following these instructions (use the “Manual Installation” section): https://github.com/cga-harvard/HHypermap/blob/master/_docs/developers.md

Note that you can bypass Java 8 installation as it was installed previously. As a search engine you should install Solr, as we haven’t tested Elasticsearch with WorldMap so far. Create a specific virtual environment for Hypermap in order not to interfere with the GeoNode/WorldMap virtual environment.

After installing Hypermap, start it on a different port than 8000, for example:

```
python manage.py runserver 0.0.0.0:8001
```

In another shell start the Celery process as well:

```
cd HHypermap
celery -A hypermap worker --beat --scheduler django -l info
```

3.3.1.6.3 Test the stack

Now that GeoNode/WorldMap and Hypermap are both running, test the stack by uploading a layer.

Login in GeoNode (admin/admin) and upload a shapefile from this page: <http://localhost:8000/layers/upload>

Make sure the shapefile is correctly displayed in GeoNode by going to the layer page.

Now login in Hypermap (admin/admin) and go to the admin services page: <http://localhost:8001/admin/aggregator/service/> Add a service like this:

- Title: My GeoNode WorldMap SDI
- Url: <http://localhost:8000/>
- Type: GeoNode WorldMap

Go to the Hypermap service page and check if the service and the layer is there: <http://localhost:8001/registry/>

In order to have layers in the search engine (Solr) there are two options:

- 1) from task runner press the “Index cached layers” button
- 2) schedule a task in celery

We recommend the second option, which can be configured in the next section.

3.3.1.6.4 Schedule Celery tasks

Go to the Periodic Task administrative interface: http://localhost:8001/admin/django_celery_beat/periodictask/

Create the following two tasks:

Index Cached Layer Task

This task will sync the layers from the cache to the search engine. Layers are sent in the cache every time they are saved:

- Name: Index Cached Layer
- Task (registered): `hypermap.aggregator.tasks.index_cached_layers`
- Interval: every 1 minute (or as needed)

Check Worldmap Service

This task will do a check of all of WorldMap service:

- Name: Check WorldMap Service
- Task (registered): `hypermap.aggregator.tasks.check_service`
- Interval: every 1 minute (or as needed)
- Arguments: `[1]` # 1 is the id of the service. Change it as is needed

Now upload a new layer in GeoNode/WorldMap and check if it appears in Hypermap and in Solr (you may need to wait for the tasks to be executed)

Update Last GeoNode WorldMap Layers

If your GeoNode/WorldMap instance has many layers, it is preferable to run the `check_service` not so often, as it can be time consuming, and rather use the `update_last_wm_layers`.

As a first thing, change the interval for the `check_service` task you created for GeoNode/WorldMap to a value such as “one day” or “one week”.

Then create the following periodic task:

- Name: Sync last layers in WorldMap Service
- Task (registered): hypermap.aggregator.update_last_wm_layers
- Interval: every 1 minute
- Arguments: [1] # 1 is the id of the service. Change it as is needed

3.4 Organizational

The *Organizational* section contains information about the GeoNode project itself, how to contribute, learn about the community, helpful links, about the patch review process, the project road map and other administrative items.

3.4.1 Organizational

3.4.1.1 Project Information

This is information on existing projects, contributing to GeoNode (code, documentation, translation, ...) and the community itself.

About GeoNode - What is GeoNode, the big picture.

Roadmap Process - How GeoNode can move ahead into the future.

Community Resources - Lots of links, think of it like your personal GeoNode bookmarks.

Community Bylaws - Some rules to keeps us stronger.

GeoNode Projects - Who else is doing cool stuff with GeoNode.

GNIPS GeoNode Improvement Proposals.

3.4.1.2 Contributing

Helping out the GeoNode project is great and by contributing we all benefit and here is how:

Contributing to GeoNode is the best way to help out and here we show you how.

GeoNode Patch Review Process is where code review happens, explained for developers.

Patch Review criteria for extending GeoNode.

How to contribute to GeoNode's translation and update an existing language or add a new one.

How to contribute to GeoNode's Documentation is outlined how to get started writing documentation.

How to write Documentation a work in progress outlining well ..how to write documentation.

How to Translate the Documentation describes how to translate the documentation.

3.4.1.3 Table of Content

3.4.1.3.1 Roadmap Process

The GeoNode Roadmap Process is designed to complement the more technical *GeoNode Improvement Proposals (GNIPS)* and strives to make it easier for the various organizations invested in GeoNode to collaborate on features of common interest.

It is based on the roadmap items developed at the GeoNode Summit held in May 2011.

Overall, the process for adding items to the collective roadmap is as follows:

1. Organizational partner has an intent to add a feature to the roadmap.
2. Organizational partner communicates with the [organizational partners list](#) about the change to gauge interest and determine who else is committed to making it happen.
3. Organizational partner creates a feature specification on the wiki to further flesh out the idea.
4. Organizational partner finds a committer on the [developer list](#) to shepherd the roadmap item through the GeoNode Improvement Proposals (GNIPS).

Each roadmap item will go through four stages:

1. Descriptive Stage (under discussion/"Active")
2. Technical Stage
3. Development Stage
4. Released

After communicating on the [organizational partners list](#) the roadmap items enters the *Descriptive Stage* and must have a wiki page that lays out the description, user stories, and other interested parties. Optionally, the roadmap item will also include an idea of the difficulty and goals as well as any wireframes, technical diagrams, or prior art.

A roadmap item enters the *Technical Stage* once a committer has been found to shepherd the roadmap item through the [GNIPS](#) process, then the wiki page must contain a clear sense of the technical assumptions, requirements or dependencies, and suggested implementation. Some roadmap items may need to be divided into multiple independent GNIP proposals.

Once it passes through the Improvement Proposals process, a roadmap item enters the *Development Stage* on its way to *Release*.

[[RoadMap-Items]]

3.4.1.3.2 Community Resources

Here you will find many links to resources on GitHub, external sites using GeoNode. Think of like your GeoNode Bookmarks.

Main Links

These 3 top links are the GeoNode landing pages and will take you to all other information on GeoNode.

- Main home page and blog <http://geonode.org>
- Documentation <http://docs.geonode.org>
- GitHub Code <https://github.com/GeoNode>

Community Contact

Contact members and ask questions

- **Mailing Lists**
 - Users List <http://lists.osgeo.org/cgi-bin/mailman/listinfo/geonode-users>

- Developers List <http://lists.osgeo.org/cgi-bin/mailman/listinfo/geonode-devel>
- IRC <irc://irc.freenode.net/geonode>
- Webchat: <http://webchat.freenode.net/?channels=#geonode>
- **Social Media**
 - Blog <http://geonode.org/blog/>
 - Twitter <https://twitter.com/geonode>
 - Google+ <https://plus.google.com/u/0/100587124776656797019>

Github Project Links

- Main GitHub page <https://github.com/GeoNode/geonode>
- Master Branch <https://github.com/GeoNode/geonode/tree/master>
- Issue Tracker <https://github.com/GeoNode/geonode/issues>
- Pull Requests <https://github.com/GeoNode/geonode/pulls>
- geonode.org Homepage Code <https://github.com/geonode/geonode.github.com> (published to <http://geonode.org>)
- Localisation using Transifex <https://www.transifex.com/geonode/geonode/>

Demo Sites

- Demo GeoNode Site <http://demo.geonode.org/>
- UI/UX Review Site <https://sites.google.com/a/opengeo.org/geonode-ui/>

Testing and Packaging

- **Testing**
 - Travis-CI <https://travis-ci.org/#!/GeoNode/geonode>
 - Openhub <https://www.openhub.net/p/geonode>
- **Packages**
 - PyPI <https://pypi.python.org/pypi/GeoNode>
 - **Launchpad**
 - * <https://launchpad.net/~geonode>
 - * <https://launchpad.net/~geonode/+archive/release>
 - * <https://launchpad.net/~geonode/+archive/testing>
 - * <https://launchpad.net/~geonode/+archive/snapshots>

Important Forks

Full List here <https://github.com/GeoNode/geonode/network/members>

Hint: Look at the branches in these Forks

- <https://github.com/cga-harvard/cga-worldmap>
- <https://github.com/jj0hns0n/geonode>
- <https://github.com/ingenieroariel/geonode>
- <https://github.com/gfdrr/geonode>

Downstream Github Projects

- <https://github.com/aifdr/tsudat2>
- <https://github.com/aifdr/riab>
- <https://github.com/gem/>
- <https://github.com/MapStory/mapstory>
- <https://github.com/CIGNo-project/CIGNo>
- https://github.com/ROGUE-JCTD/rogue_geonode
- <https://github.com/boundlessgeo/exchange>
- Many More ...

Additional Modules

- <https://github.com/simod/geonode-documents>
- <https://github.com/GFDRR/geonode-registry>

Public Sites

- CIGNo network: CNR - ISMAR Node <http://cigno.ve.ismar.cnr.it/>
- ... Many Many More

3.4.1.3.3 Community Bylaws

Committers

The GeoNode community is divided into two groups - users and committers. There are no requirements or responsibilities to be a GeoNode user. To be a committer, you must be voted in by the existing committers (2 +1's and no -1's; a committer must initiate the vote.) Non-committers are encouraged to engage in discussions on the mailing lists, code review, and issue reports to qualify them to be voted in as committers. Committers (or PRIMARY AUTHORS) can be found in the [AUTHORS file](<https://github.com/GeoNode/geonode/blob/master/AUTHORS>)

Committers must:

- Make useful contributions to the project in the form of commits at least once in a 6-month period, else they fall back to “committer emeritus” status. A committer emeritus has no special involvement in the project, but may request committer privileges from the current body of committers.
- Review code contributions, which may come from other committers or from users. Users must submit code externally to the main GeoNode repository (ie as a patch or a github pull request); committers can do this as well if they see review as particularly important (for example, a patch might affect a particularly crucial component of GeoNode, or a committer might be working in a part of the code that he is relatively unfamiliar with.) A review should result in either (a) instructions on how to bring the code to a more acceptable condition or (b) merging the changes in and notifying the submitter that this has been done.
- Committers also have the option to “self-review” and commit changes directly. It is at the discretion of individual committers when this is appropriate, but it should be rare - we encourage committers to only use this option when they deem a change extremely safe.

GeoNode Improvement Proposals (GNIPS)

GNIPS If a committer thinks a proposed change to the software is particularly destabilizing or far-reaching, that committer can upgrade the ticket for that change to a GeoNode Improvement Proposal (GNIP). GNIP tickets are an opportunity for committers and users alike to provide feedback about the design of a proposed feature or architectural change. The proposal should be iteratively edited in response to community feedback.

To upgrade an issue to a GNIP, an active committer should give the ticket the ‘GNIP’ label in the issue tracker, and announce the issue on the developer mailing list.

If a ticket has a GNIP label, its patch can’t be committed unless it also has the ‘Approved’ label. To be approved, it must pass community vote (see below).

When the GNIP is announced, other committers should review and provide feedback in the issue comments. Feedback should take the form of:

- +1 (with optional comment)
- -1, with mandatory rationale and suggestion for a better approach. The suggestion may be omitted if the objection doesn’t have a straightforward solution - we don’t want to withhold feedback just because problems with a proposal are hard to solve.

After receiving feedback, the proposal’s author should discuss the feedback on the list if necessary and adjust the proposal in response.

A proposal can be Approved when there are 3 +1 responses (including the author’s implicit approval) and no -1 responses from committers; and no feedback is offered in 3 days. If a proposal fails to receive multiple +1 responses within 5 days of the request for feedback it is rejected and the issue should be **closed** (but the author is free to draft similar proposals in the future.) Any committer may reverse or withdraw votes on a proposal until the proposal is closed.

If a user would like to submit a GNIP, they are welcome to write it as a ticket but should find an active committer willing to promote it to GNIP status.

Project Steering Committee

In the event that a revision to these bylaws becomes necessary, authority for that decision lies with the currently presiding Project Steering Committee (PSC). The PSC at any time is made up of the top 7 committers over the past 365 days, by number of commits.

GNIPS: <https://github.com/GeoNode/geonode/wiki/GeoNode-Improvement-Proposals>

3.4.1.3.4 Contributing to GeoNode

Warning: This section is freely adapted from the official [GitHub guides](#).

If you are interested in helping us to make GeoNode, there are many ways to do so.

Participate in the Discussion

GeoNode has a mailing list (<http://geonode.org/communication/#mailing-lists>) where users can ask and answer questions about the software. There are also IRC chats on Gitter where users (<https://gitter.im/GeoNode/general>) and developers (<https://gitter.im/GeoNode>) can discuss GeoNode in real time. Sometimes users also post interesting tips for managing sites running GeoNode. If you want to help out with GeoNode, one easy way is to sign up for the mailing list and help answer questions.

Report Problems on the Issue Tracking System

Informative bug reports are a key part of the bug fixing process, so if you do run into a problem with GeoNode, please don't hesitate to report it on our bug tracker, available online at <https://github.com/GeoNode/geonode/issues>. Useful information for bug reports includes:

- What were you doing when the bug occurred? Does the problem occur every time you do that, or only occasionally?
- What were you expecting to happen? What happened instead?
- What version of the software are you using? Please also note any changes from the default configuration.
- If there is a data file involved in the bug (such as a Shapefile that doesn't render properly), please consider including it in the bug report. Be aware that not all data files are freely distributable.

To help GeoNode address the issue, you can tag the ticket with one or more labels that you can find on the side column.

Write Documentation

GeoNode's documentation can always use improvement - there are always more questions to be answered. For managing contributions to the manual, GeoNode uses a process similar to that used for managing the code itself. The documentation is generated from source files in the *docs/* directory within the GeoNode source repository. See <http://www.sphinx-doc.org> for more information on the documentation system GeoNode uses.

If you want to learn more about contributing to the documentation, please go ahead to the “[How to contribute to GeoNode's Documentation](#)”. GeoNode also have some guidelines to help with writing once you are set up “[How to write Documentation](#)”.

Provide Translations

If GeoNode doesn't provide a user interface in your native language, consider contributing a new translation. To get started here are the instructions “[How to contribute to GeoNode's Translation](#)”.

Write Code

Of course since GeoNode is an open source project which encourages contributions of source code as well. If you are interested in making small changes, you can find an open ticket on <https://github.com/GeoNode/geonode/issues>, hack away, and get started on the “[Patch Review Process](#)”.

Further Reading

Contributing to Open Source on GitHub

Work With GitHub Issues and Pull Requests

Roadmap Process

Contributing to Open Source on GitHub

Warning: This section is freely adapted from the official [GitHub guides](#).

A great way to get involved in open source is to contribute to the existing projects you’re using.

A Typical GitHub Project Structure

The Community

Projects often have a community around them, made up of other users in different (formal or informal) roles:

- **Owner** is the user or organization that created the project has the project on their account.
- **Maintainers** and Collaborators are the users primarily doing the work on a project and driving the direction. Oftentimes the owner and the maintainer are the same. They have write access to the repository.
- **Contributors** is everyone who has had a pull request merged into a project.
- **Community Members** are the users who often use and care deeply about the project and are active in discussions for features and pull requests.

Readme

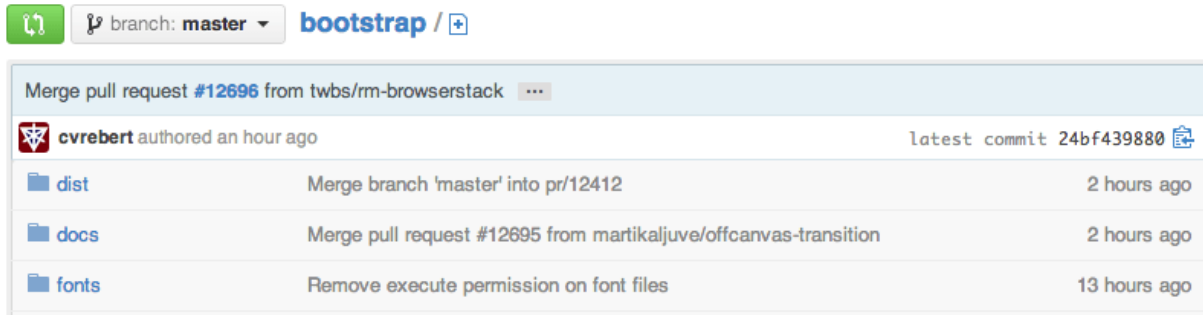
Nearly all GitHub projects include a README.md file. The readme provides a lay of the land for a project with details on how to use, build and sometimes contribute to a project.

License

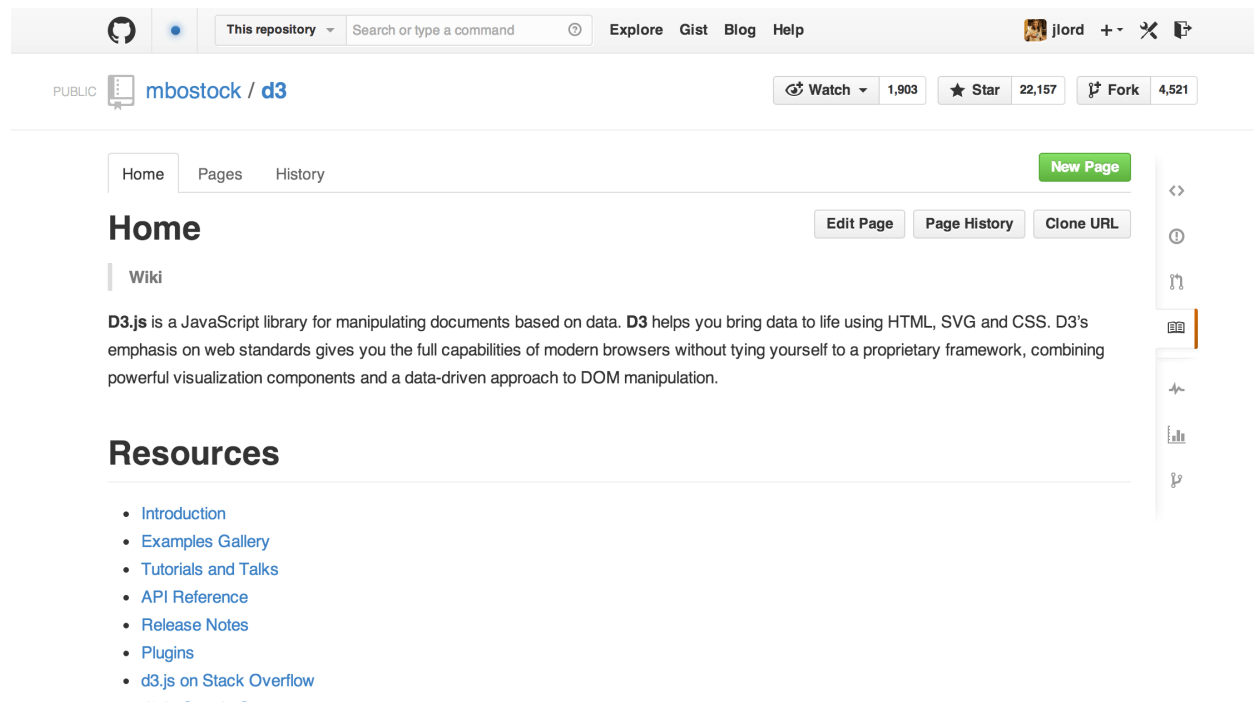
A *LICENSE* file, well, is the license for the project. An open source project’s license informs users what they can and can’t do (e.g., use, modify, redistribute), and contributors, what they are allowing others to do.

Documentation and Wikis

Many larger projects go beyond a readme to give instructions for how people can use their project. In such cases you'll often find a link to another file or a folder named *docs* in the repository.



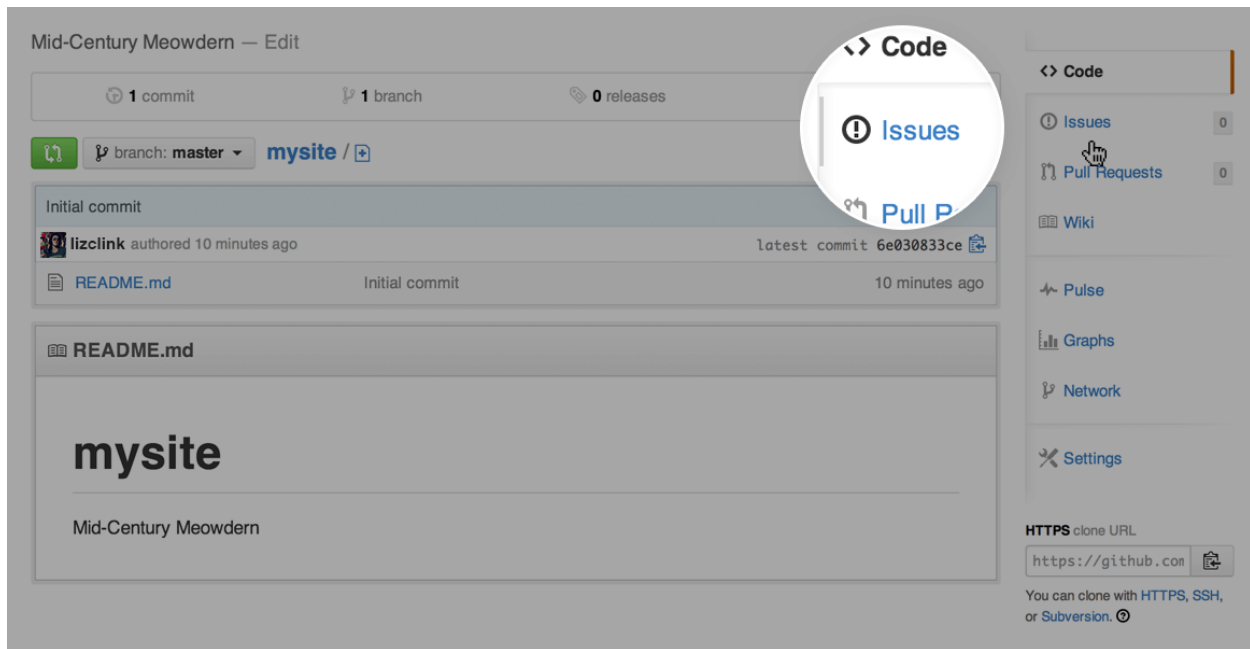
Alternatively, the repository may instead use the GitHub wiki to break down documentation.



Issues

Issues are a great way to keep track of tasks, enhancements, and bugs for your projects. They're kind of like email—except they can be shared and discussed with the rest of your team. Most software projects have a bug tracker of some kind. GitHub's tracker is called Issues, and has its own section in every repository.

For more information on how Issues work, see the section “*Work With GitHub Issues and Pull Requests*”



Pull Requests

If you're able to patch the bug or add the feature yourself, make a pull request with the code. Be sure you've read any documents on contributing, understand the license and have signed a CLA if required.

Once you've submitted a pull request, the maintainer(s) can compare your branch to the existing one and decide whether or not to incorporate (pull) your changes.

For more information on how Pull Requests work, see the section "[Work With GitHub Issues and Pull Requests](#)"

Work With GitHub Issues and Pull Requests

Warning: This section is freely adapted from the official [GitHub guides](#).

Issues

An Issue is a note on a repository about something that needs attention. It could be a bug, a feature request, a question or lots of other things. On GitHub you can label, search and assign Issues, making managing an active project easier.

For example, let's take a look at [Bootstrap's Issues section](#):

GitHub's issue tracking is special because of our focus on collaboration, references, and excellent text formatting. A typical issue on GitHub looks a bit like this:

- A **title** and **description** describe what the issue is all about.
- Color-coded **labels** help you categorize and filter your issues (just like labels in email).
- A **milestone** acts like a container for issues. This is useful for associating issues with specific features or project phases (e.g. *Weekly Sprint 9/5-9/16* or *Shipping 1.0*).
- One **assignee** is responsible for working on the issue at any given time.

Issues

Pull requests

Labels

Milestones

Filters

is:open is:issue

New Issue

104 Open

9,660 Closed

Author

Labels

Milestones

Assignee

Sort

.form-group-sm .form-group-lg shrink textarea

confirmed

css

#13989 opened 11 hours ago by limitstudios

v3.2.1

4

Tooltip unnecessarily breaks into multiple lines when positioned to the right

confirmed

js

#13987 opened 15 hours ago by hnrch02

v3.2.1

0

Tooltip Arrows in Modal example facing wrong way

css

#13981 opened a day ago by SDCore

6

Table improvement

css

#13978 opened a day ago by Tjoosten

0

docs/dist files

docs

#13977 opened 2 days ago by XhmikosR

v3.2.1

7

Potential solution to #4647

js

#13976 opened 2 days ago by julioarmandof

4

Bootstrap site: right-hand navigation text becomes rasterized after scrolling

css

docs

#13974 opened 2 days ago by mg1075

v3.2.1

4

Dropdown toggle requires two clicks

js

#13972 opened 2 days ago by Kizmar

1

◀ The no-conflict mode should be the default behaviour #12395

Edit New Issue

Open thewebdreamer opened this issue 3 days ago · 10 comments

thewebdreamer commented 3 days ago

The no-conflict mode should be the default behaviour. Why would a Bootstrap client need to implement this?

cvrebert commented 3 days ago

I believe no-conflict-is-not-the-default is the norm for jQuery plugins?

thewebdreamer commented 3 days ago

It is true that it is the norm for jQuery plugins.

Couldn't there be a clash with other jQuery plugins with the current implementation of Bootstrap though?

Labels

js

Milestone

No milestone

Assignee

No one assigned

Notifications

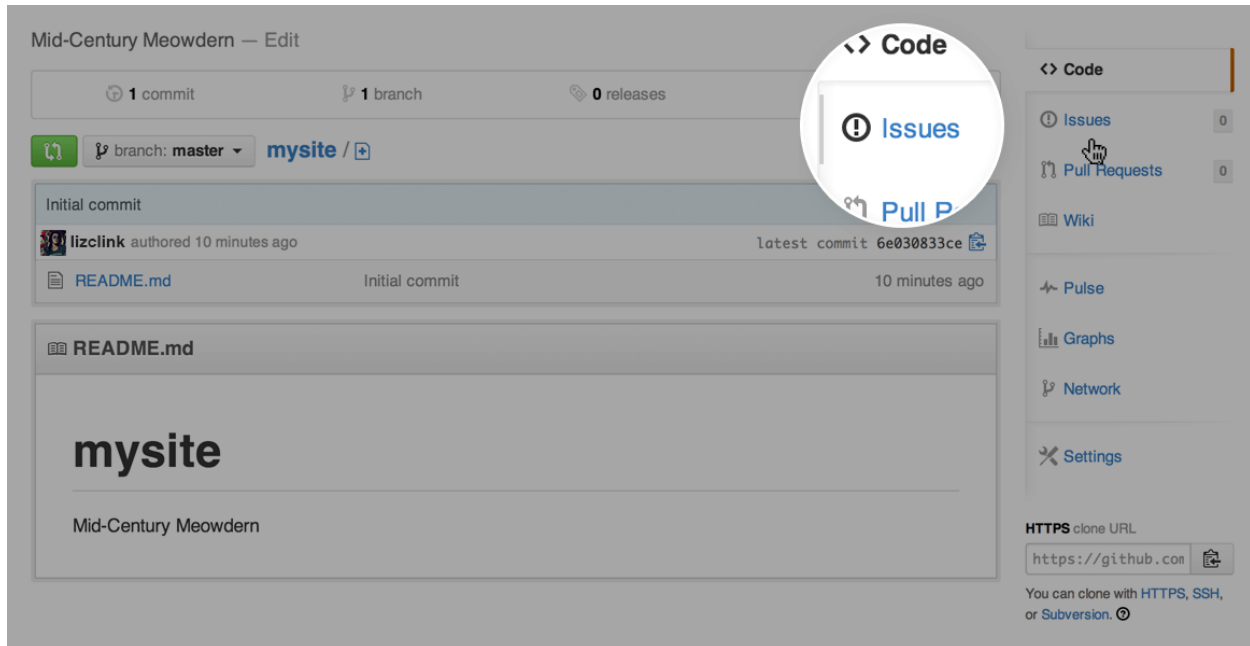
Subscribe

3 participants

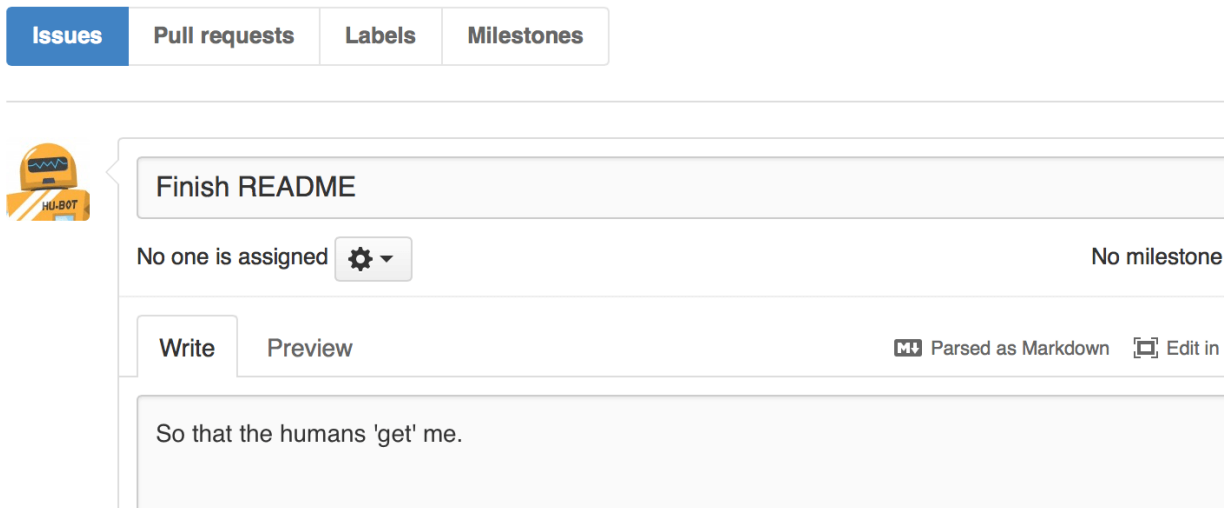
- **Comments** allow anyone with access to the repository to provide feedback.

Open an Issue

1. Click the Issues tab from the sidebar.



2. Click New Issue.
3. Give your Issue a title and description: *Add a new Logo to GeoNode custom.*



Click **Submit new Issue** when you're done. Now this issue has a permanent home (URL) that you can reference even after it is closed.

Issues Pro Tips

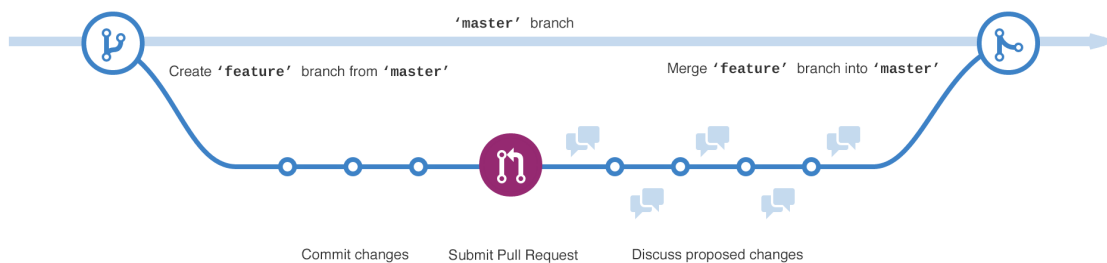
- **Check existing issues** for your issue. Duplicating an issue is slower for both parties so search through open and closed issues to see if what you're running into has been addressed already.
- **Be clear** about what your problem is: what was the expected outcome, what happened instead? Detail how someone else can recreate the problem.
- **Link to demos** recreating the problem on things like JSFiddle or CodePen.
- **Include system details** like what the browser, library or operating system you're using and its version.
- **Paste error output** or logs in your issue or in a Gist. If pasting them in the issue, wrap it in three backticks: ````` so that it renders nicely.

Branching

Branching is the way to work on different parts of a repository at one time.

When you create a repository, by default it has one branch with the name `master`. You could keep working on this branch and have only one, that's fine. But if you have another feature or idea you want to work on, you can create another branch, starting from `master`, so that you can leave `master` in its working state.

When you create a branch, you're making a **copy** of the original branch as it was at that point in time (*like a photo snapshot*). If the original branch changes while you're working on your new branch, no worries, you can always pull in those updates.



At GeoNode developers use branches for keeping bug fixes and feature work separate from `master` (**production**) branch. When a feature or fix is ready, the branch is **merged** into `master` through a **Pull Request**.

To create a new branch

- Go to the project folder and create a new branch

```
$ cd /home/geonode/geonode_custom/  
$ sudo git branch add_logo  
$ sudo git checkout add_logo
```

- Check that you are working on the correct branch: `add_logo`.

```
$ cd /home/geonode/geonode_custom/  
$ git branch
```

- Push the new branch to GitHub.

```
geo@geonode:/home/geonode/geonode_custom$ sudo git branch add_logo
geo@geonode:/home/geonode/geonode_custom$ sudo git checkout add_logo
Switched to branch 'add_logo'
geo@geonode:/home/geonode/geonode_custom$
```

No Layers

No Maps

ck to search for geospatial data published by other
, organizations and public sources. Download data in

Data is available for browsing, aggregating ar
generate maps which can be shared publicly c

```
geo@geonode:/home/geonode/geonode_custom$ git branch
* add_logo
  master
geo@geonode:/home/geonode/geonode_custom$
```

No Layers

No

```
$ cd /home/geonode/geonode_custom/
$ sudo git push origin add_logo
```

```
geo@geonode:/home/geonode/geonode_custom$ sudo git push origin add_logo
Username for 'https://github.com': afabiani
Password for 'https://afabiani@github.com':
Total 0 (delta 0), reused 0 (delta 0)
To https://github.com/afabiani/geonode_custom.git
 * [new branch]      add_logo -> add_logo
geo@geonode:/home/geonode/geonode_custom$
```

Make a commit

On GitHub, saved changes are called **commits**.

Each commit has an associated **commit message**, which is a description explaining why a particular change was made. Thanks to these messages, you and others can read through commits and understand what you've done and why.

- Add a new logo to your custom GeoNode as described in the section *Theming your GeoNode project*
- Stash the new files into the working project using `git add`

```
$ cd /home/geonode/geonode_custom/
$ sudo git add geonode_custom/static
$ git status
```

- **Commit** the changes providing a **commit messages** and push them into your branch : `add_logo`.

```
$ cd /home/geonode/geonode_custom/
$ sudo git commit -m "Adding a new logo to the custom GeoNode"
$ sudo git push origin add_logo
```

```
geo@geonode:/home/geonode/geonode_custom$ git status
On branch add_logo
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   geonode_custom/static/css/site_base.css
        new file:    geonode_custom/static/img/UWI-logo.JPG

geo@geonode:/home/geonode/geonode_custom$
```

```
geo@geonode:/home/geonode/geonode_custom$ sudo git commit -m "Adding a new Logo to the custom GeoNode"
[add_logo a9a1da1] Adding a new logo to the custom GeoNode
 2 files changed, 6 insertions(+)
 create mode 100644 geonode_custom/static/img/UWI-logo.JPG
geo@geonode:/home/geonode/geonode_custom$ sudo git push origin add_logo
Username for 'https://github.com': afabiani
Password for 'https://afabiani@github.com':
Counting objects: 14, done.
Compressing objects: 100% (7/7), done.
Writing objects: 100% (8/8), 9.28 KiB | 0 bytes/s, done.
Total 8 (delta 2), reused 0 (delta 0)
To https://github.com/afabiani/geonode_custom.git
 4fb59e2..a9a1da1  add_logo -> add_logo
geo@geonode:/home/geonode/geonode_custom$
```

Pull Requests

Pull Requests are the heart of collaboration on GitHub. When you make a pull request, you're proposing your changes and requesting that someone pull in your contribution - aka merge them into their branch. GitHub's Pull Request feature allows you to compare the content on two branches. The changes, additions and subtractions, are shown in green and red and called diffs (differences).

As soon as you make a change, you can open a Pull Request. People use Pull Requests to start a discussion about commits (code review) even before the code is finished. This way you can get feedback as you go or help when you're stuck.

By using GitHub's @mention system in your Pull Request message, you can ask for feedback from specific people or teams.

Create a Pull Request for changes to the Logo

- Click the Pull Request icon on the sidebar, then from the Pull Request page, click the green **New pull request** button.
- Select the branch you made, `add_logo`, to compare with `master` (the original).
- Look over your changes in the diffs on the Compare page, make sure they're what you want to submit.
- When you're satisfied that these are the changes you want to submit, click the big green Create Pull Request button.



Compare and review just about anything

Branches, tags, commit ranges, and time ranges. In the same repository and across forks.

EXAMPLE COMPARISONS

add_logo	7 minutes ago
master@{1day}...master	24 hours ago

Showing 2 changed files with 6 additions and 0 deletions.

6 geonode_custom/static/css/site_base.css

... @@ -0,0 +1,6 @@

```
1 +.navbar-brand {
2 +  width: 373px;
3 +  height: 79px;
4 +  background: transparent url("../img/UII-logo.
5 +}
6 +
```

BIN geonode_custom/static/img/UII-logo.JPG



Choose two branches to see what's changed or to start a new pull request. If you need to, you can also [compare across forks](#).



base: **master** ▼

...

compare: **add_logo** ▼

✓ **Able to merge.** These branches can be automatically merged.



Create pull request

Discuss and review the changes in this comparison with others.

1 commit


2 files changed




Commits on Dec 17, 2015

- Give your pull request a title and since it relates directly to an open issue, include “fixes #” and the issue number in the title. Write a brief description of your changes.

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across fork](#)


 base: **master** ... compare: **add_logo** ✓ Able to merge. These branches can be automatically merged.



Adding a new logo to the custom GeoNode

Write

Preview

 Styling with Markdown is supported

Leave a comment

Attach files by dragging & dropping, [selecting them](#), or pasting from the clipboard.

Create pull request

When you're done with your message, click **Create pull request!**

Merge your Pull Request

It's time to bring your changes together – merge your `add_logo` branch into the `master` (the original) branch.

Click the green button to merge the changes into master. Click Confirm merge. Go ahead and delete the branch, since its changes have been incorporated, with the Delete branch button in the purple box.



This pull request can be automatically merged.

You can also merge branches on the [command line](#).

 Merge pull request

If you revisit the issue you opened, it's now closed! Because you included “fixes #1” in your Pull Request title, GitHub took care of closing that issue when the Pull Request was merged!

Roadmap Process

The GeoNode Roadmap Process is designed to complement the more technical **GeoNode Improvement Proposals** and strives to make it easier for the various organizations invested in GeoNode to collaborate on features of common interest.

It is based on the roadmap items developed at the GeoNode Summit held in May 2011.

Overall, the process for adding items to the collective roadmap is as follows:

1. Organizational partner has an intent to add a feature to the roadmap.
2. Organizational partner communicates with the [organizational partners list](#) about the change to gauge interest and determine who else is committed to making it happen.
3. Organizational partner creates a feature specification on the wiki to further flesh out the idea.
4. Organizational partner finds a committer on the [developer list](#) to shepherd the roadmap item through the **GeoNode Improvement Proposals**.

Each roadmap item will go through four stages:

1. Descriptive Stage (under discussion/"Active")
2. Technical Stage
3. Development Stage
4. Released

After communicating on the [organizational partners list](#) the roadmap items enters the *Descriptive Stage* and must have a wiki page that lays out the description, user stories, and other interested parties. Optionally, the roadmap item will also include an idea of the difficulty and goals as well as any wireframes, technical diagrams, or prior art.

A roadmap item enters the *Technical Stage* once a committer has been found to shepherd the roadmap item through the **GeoNode Improvement Proposals** process, then the wiki page must contain a clear sense of the technical assumptions, requirements or dependencies, and suggested implementation. Some roadmap items may need to be divided into multiple independent GNIP proposals.

Once it passes through the **GeoNode Improvement Proposals** process, a roadmap item enters the *Development Stage* on its way to *Release*.

3.4.1.3.5 GeoNode Patch Review Process

This document outlines the code review process for GeoNode code. Each commit proposed for inclusion in GeoNode should be reviewed by at least one developer other than the author. For pragmatic reasons, some developers, referred to in this document as *core committers*, may commit directly to the GeoNode repository without waiting for review. Such changes are still subject to review, and may be reverted if they fail any of the [\[\[Review Criteria\]\]](#).

A related process is [\[\[Improvement Proposals\]\]](#). While patch review protects code quality in the GeoNode project at a small granularity, the Improvement Proposal process is intended to promote coordinated design and feedback for larger modifications such as new features or architectural changes.

Goals

By requiring a review of code coming into GeoNode, we aim to maintain code quality within the GeoNode project while still allowing contributions from the GeoNode community.

Review Criteria

Patch reviews should adhere to the standards set in the [[Review Criteria]], a [[Project Steering Committee]] approved set of criteria for the inclusion of new code.

Process

For contributors who do not have commit access to the GeoNode repository, the review process is as follows:

0. Find or create a ticket describing the feature or bug to be resolved.
1. Create changes to GeoNode code addressing the ticket.
2. Publish those changes and request a review from the GeoNode committers. The recommended format is a GitHub [pull request](<https://help.github.com/articles/about-pull-requests/>). If you are unable or unwilling to provide a change as a branch on GitHub, please consult the developer's list for advice.
3. At least one GeoNode committer should review the submitted changes. If he finds the patch acceptable, the changes will be pulled into GeoNode. If problems are found, then he should list them clearly in order to allow the original author to update the submission (at which point we return to point 2 in this process.) In the case of a feature idea which is simply not suitable for inclusion as core GeoNode functionality, the patch may be rejected outright.

Note: If after a few days your patch has not been reviewed by any GeoNode committer, please feel free to bring it up either in the mailing list or the IRC channel. The GeoNode community (and it's committers) try to respond quickly and give adequate feedback to maintain the interest of new potential members. However, sometimes other responsibilities prevent us from responding quickly.

Core Committers

It is assumed that core committers are familiar with the quality guidelines and capable of producing acceptable patches without the need for waiting on review. Therefore, core committers may make changes without requesting review first (although they are welcome to request review for code if they feel it is appropriate.) For commits made without prior review, committers should review the changes and revert them if they are in violation of the project quality guidelines.

Becoming a Core Committer

In order for a developer to become a core committer, he must demonstrate familiarity with the quality guidelines for the GeoNode project by producing at least two patches which successfully pass review and are merged without requiring modification. A candidate for core committer-ship must be nominated by a member of the [[Project Steering Committee]], and approved via Apache consensus voting among PSC members.

3.4.1.3.6 Patch Review criteria

When a patch is rejected in the Patch Review Process, it should be given a valid review.

This review should point out issues with the patch where it fails to meet one or more of the following criteria.

- Major new features must be approved by the Improvement Process

GeoNode needs to be coherent software despite the diverse interests driving its development. Therefore, major new features need to first be approved according to the [[Improvement Process]].

If a patch fails by this criterion, then its developer is welcome to go through the improvement process to get approval. Otherwise, they can refactor their patch into a GeoNode extension.

- Patches need sufficient documentation

We strive to keep GeoNode well-documented. If a patch contributes significant functionality to GeoNode that requires documentation to be understood, the patch review is an opportunity to hold the developer accountable for providing the adequate documentation.

- New functionality needs to be internationalized

We strive to build GeoNode in a way that can be used in many different localities, by all languages. While there is no localization requirement for GeoNode besides providing default English text, new user-facing features need to be sufficiently internationalized so that others can write translations.

- Design consistency

We strive to keep the default user interface for GeoNode appealing for new users and developer's approaching the project. If a patch significantly diminishes the user experience of the software, then a patch may be rejected with a review of how to improve it.

Note: Good design can sometimes be in the eye of the beholder. Developer's are encouraged to consult the community and/or a designer about the user interface design of their patches, and to be humble in their design criticisms of others.

- Code should be covered by automated tests

To make development easier for others and guarantee software quality, we strive to have good automated test coverage in GeoNode. Patches may fail a review for having insufficient unit and/or integration tests.

Reviews saying that a patch has insufficient tests should offer actionable advice on how to improve those tests. This advice could be to improve code coverage. It may also be a list of possible cases that currently lack tests.

- Patches should not have known bugs

A patch may be rejected for having a known bug, (e.g.) one discovered by reading the code or testing it at the time of review.

- Patches should meet GeoNode's code style guidelines

New patches should meet GeoNode's code style guidelines. We follow different conventions per language:

- In Java we use the GeoTools/GeoServer convention, essentially the [conventions recommended by Oracle](<http://www.oracle.com/technetwork/java/codeconvtoc-136057.html>) modified to make the recommended line length 100 columns instead of 80 to accommodate the long identifiers commonly used in GeoTools code. The GeoServer project provides an [Eclipse configuration](<http://docs.geoserver.org/stable/en/developer/eclipse-guide/index.html#eclipse-preferences>) which helps to stick to this convention.
- In Python we use the conventions enumerated in [PEP8](<https://www.python.org/dev/peps/pep-0008/>). Many editors have plugins available to assist with conformance to this convention.
- In JavaScript we use the OpenLayers conventions, described on the [OpenLayers wiki](<http://trac.osgeo.org/openlayers/wiki/CodingStandards>).

3.4.1.3.7 How to contribute to GeoNode's translation

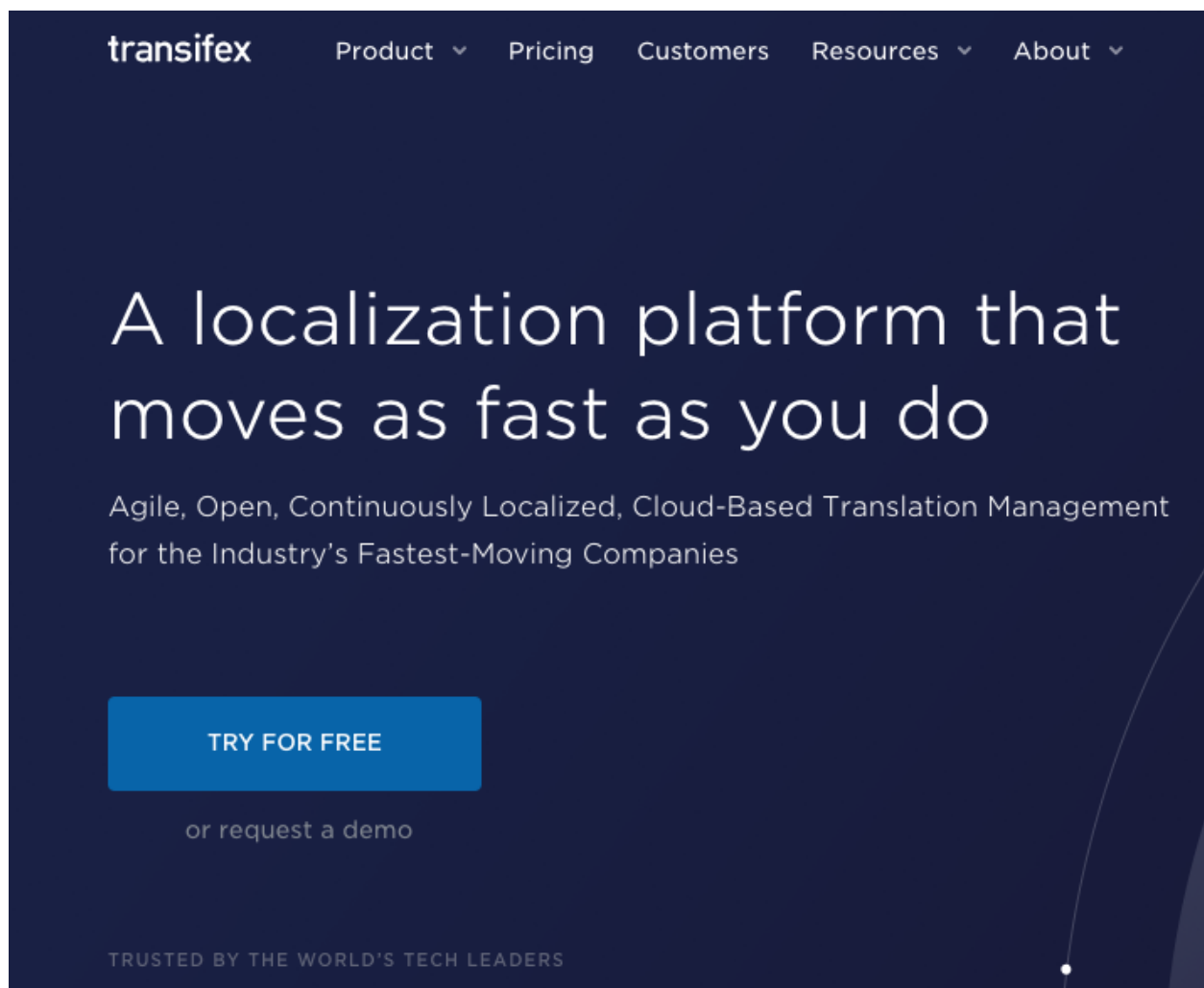
Everyone is welcome to contribute to the GeoNode's translation. There are two different ways to translate the GeoNode user interface: you can use the [Transifex](#) web application or you can edit the translation files on your local machine using Git and send Pull Requests (PR) to the GeoNode repository.

Edit translations using Transifex

The first workflow for contributing to GeoNode's translation is by using Transifex. In this step you will see how to update the translations directly on the Transifex website.

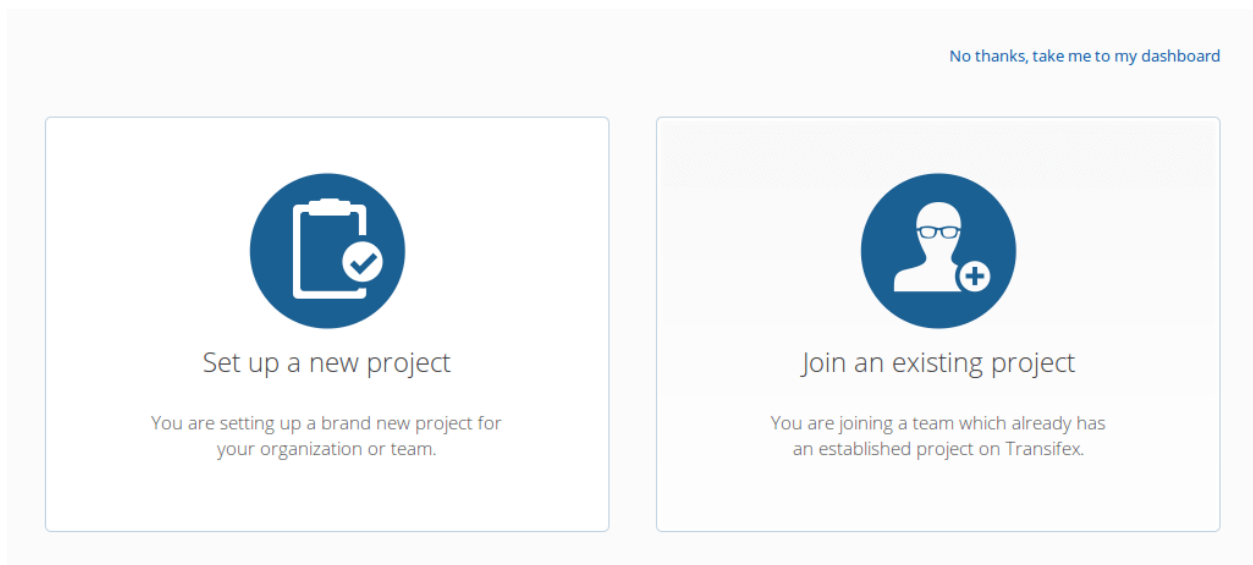
1. Create account

Go to <https://www.transifex.com> and, click *Try it for free* and enter the needed information to create your free account



2. Join our project

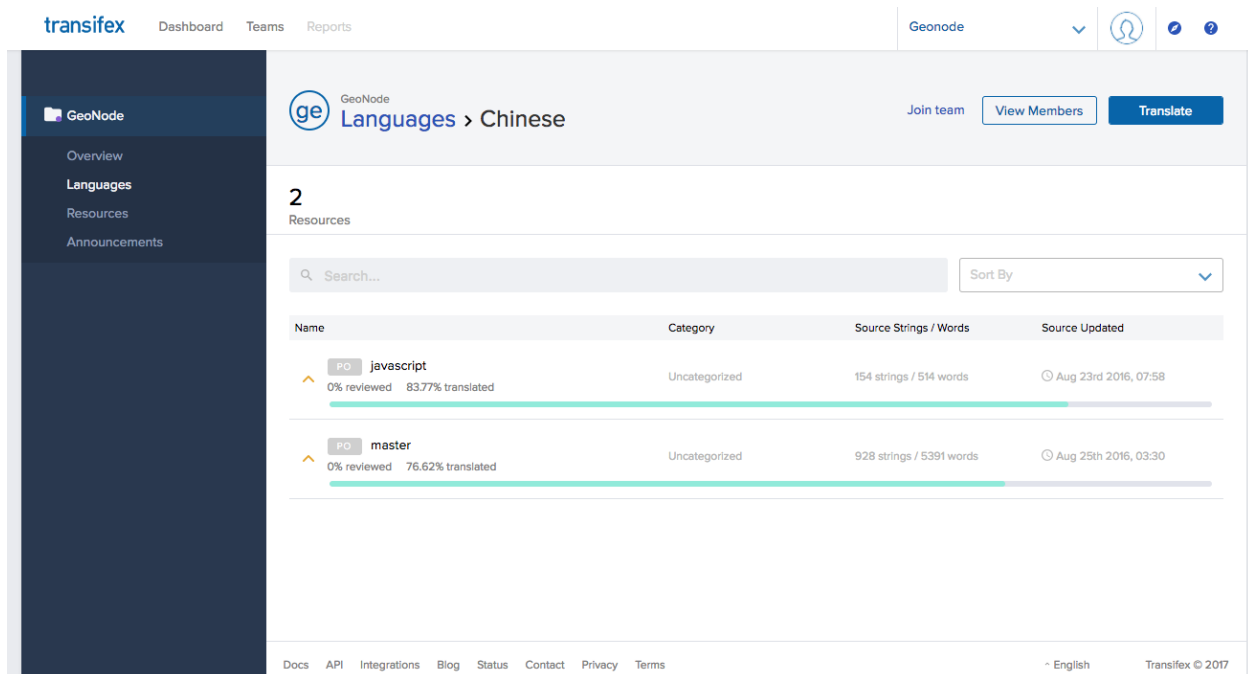
After activating the link you've got in your email, you will be asked whether you want to start a new project or to join an existing project



Click *join an existing project* and type *geonode* into the search bar. Select the GeoNode project from the GeoNode organization (not the other ones!). You will be directed to the GeoNode project site on transifex. To join the team, click on the *Join team* button, then click on the language you want to add a translation in.

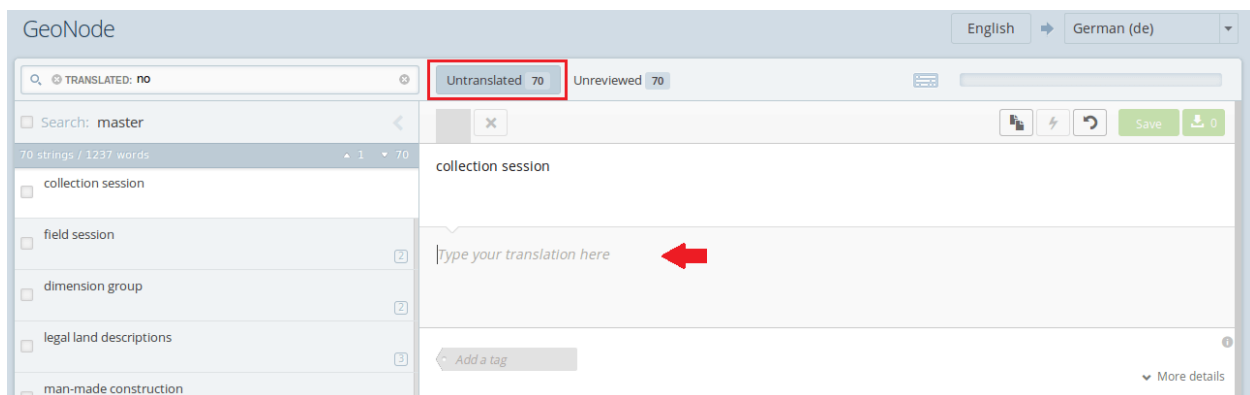
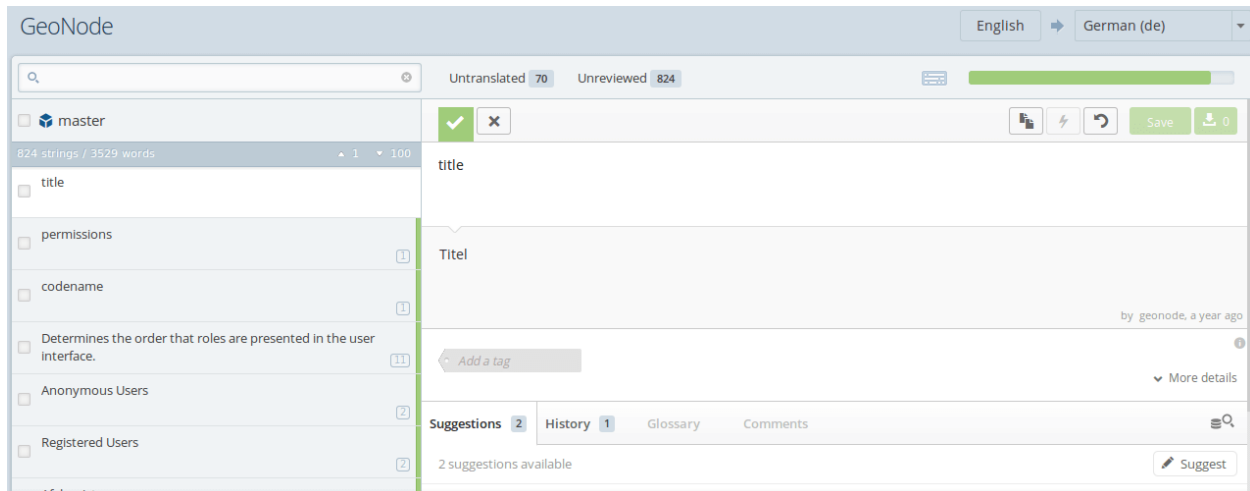
3. Wait for permission to update translations from translation leader (email)
4. Start a translation

Click on the *Translate* button, then select the language where you want to add a translation



You'll see two main translation areas: *javascript* and *master*. Choose the one you want to add a translation and you'll see an interface like this

Click *untranslated* and add your translation like shown below



When you stop translating, do not forget to hit the green *save* button at the top right!

5. See the strings translated in GeoNode

In order to see the strings translated in your GeoNode instance, you will need to synchronize your GeoNode instance with what it is in Transifex. For this purpose the best workflow it is to wait that a Transifex maintainer (see below section) will pull the files from Transifex, run the *makemessages* and *compilemessages* and finally send a PR to the GeoNode official repository. Or you can follow the following step, which uses github and does not require the help from a Transifex maintainer.

Translate on local machine from github

The second workflow for contributing to GeoNode's translation is by translating the files on your local machine and send the translation to the GeoNode official repository in GitHub with pull requests.

Using this option it is assumed that you have a local Geonode GitHub repository forked.

Note: It is recommended to first create a new branch e.g. *translation* in your repository for your translations.

1. As a first step, generate all of the needed .pot files (any time the master documentation changes):

```
$ cd docs
$ sphinx-build -b gettext . i18n/pot
```

2. Run the `pre_translate.sh` script to generate/merge (update) all of the .po files for each language:

```
$ sh i18n/scripts/pre_translate.sh
```

3. Do a pull from Transifex to get latest translations:

```
$ tx pull -a
```

4. Now edit the .po files you need, make the translations and then run the `post_translate.sh` script:

```
$ vi i18n/it/LC_MESSAGES/index.po
$ sh i18n/scripts/post_translate.sh
```

5. Now you have to push the changed .po files and the appropriate .pot file (can be found in `geonode/docs/i18n/pot`) to your remote repository using:

```
$ git commit
$ git push
```

6. Now make a pull request and GeoNode will push your changes to Transifex and include them in its official repository.

Only for transifex maintainers

Note: This section is only for the maintainers of a transifex group!

For main Geonode localization files

These files are generated using standard django-admin tools : makemessages and compilemessages command line tools.

At first make sure to install Transifex client and configure your .transifexrc with our credentials as:

```
[https://www.transifex.com]
username = YOURUSERNAME
token =
password = PUTYOURPW
hostname = https://www.transifex.com
```

To prepare the PR in your branch, execute following steps. First go to the [yourpath]/geonode/geonode/ subdirectory and get all translations from transifex.com and force their replacement:

```
$ tx pull -a -s -f
```

Then update the messages in the po and compile the corresponding mo:

```
$ geonode makemessages --all
$ geonode makemessages -d djangojs --all
$ geonode compilemessages
```

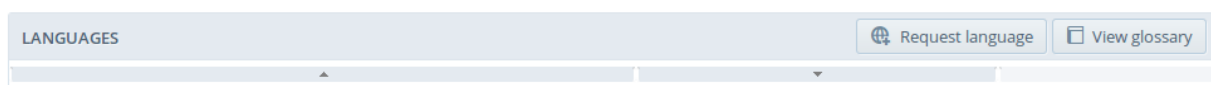
Now you can send the PR to GeoNode GitHub repository.

Finally update resources on transifex.com:

```
$ tx push -s -t
```

How to add a new language

To add a new language, click on *Request language* on the right top of the Transifex webpage.



3.4.1.3.8 How to contribute to GeoNode's Documentation

If you feel like adding or changing something in the GeoNode documentation you are very welcome to do so. The documentation always needs improvement as the development of the software is going quite fast.

In order to contribute to the GeoNode documentation you should:

- Create an account on GitHub
- Fork the GeoNode repository
- Edit the files in the */docs* directory
- Submit pull requests

All these things can generally be done on the web, you won't need to download anything. But if you want to add images to the documentation you will have to do some more initial steps, because this can't be done on the web. To

learn about how images can be added to your documentation and which additional steps have to be done, read the section *Add images*.

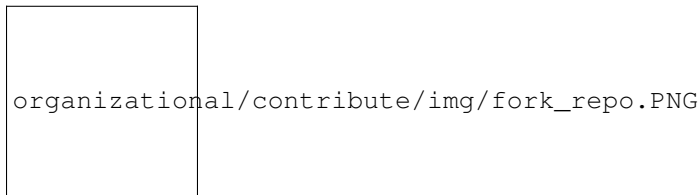
The general steps are explained in more detail below.

Create an account on GitHub

The first step is to create an account on GitHub. Just go to [Github](#), find a username that suits you, enter your email and a password and hit *Sign up for GitHub*. Now you've signed in, you can type *geonode* into the searching area at the top of the website. On top of the search results you will find the repository *GeoNode/geonode*. By clicking on it you will be entering the repository and will be able to see all the folders and files that are needed for GeoNode. The files needed for the documentation can be found in */docs*.

Fork a repository

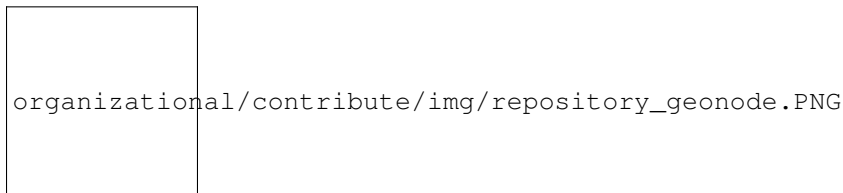
In order to make changes to these files, you first have to fork the repository. On the top of the website you can see the following buttons:



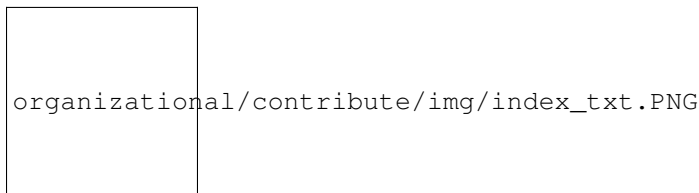
Click on the button *Fork* at the top right and the *geonode* repository will be forked. You should now be able to see your repository *your_name/geonode*. If you want to read more about how to fork a repository go to <https://help.github.com/articles/fork-a-repo>.

Edit files

To make some changes to already existing files or to create new files, go to your GitHub account. Under *repositories* you will find the geonode repository that you have forked. Click on it and you will again see all the folders and files GeoNode needs.



Click on the folder *docs* and search for the file that you want to edit. If you found it, click on it and you will be able to see the content of this file.



To make changes to this file, hit the button *edit* on the right top. You can now make your changes or add something to the existing content.



organizational/contribute/img/index_edit.PNG

As you can see now, the documentation is written in *reStructuredText*, a lightweight markup language. To learn how to use it you should read the documentation that can be found here <http://docutils.sourceforge.net/docs/user/rst/quickref.html>. By hitting the *preview* button you will be able to see how your text it is going to look like on the web. To save your changes, click on *Commit Changes* at the bottom of the site. Now you've saved the changes in your repository, but the original geonode repository still doesn't know anything about that! In order to tell them that you have made some changes you have to send a *pull request* (as described below).

To see your modifications for validation purpose just make sure you installed sphinx tools as:


```
pip install sphinx
pip install sphinx_rtd_theme
```

Just go to the docs subdirectory and use the make command with the html option, after you can open the result in your browser as:

```
cd [yourpath]/geonode/docs
make html
#you can open the index.html in _build subdirectory
```

Create a new branch

If you are planning bigger changes on the structure of the documentation it is recommended to create a new branch and make your edits here. A new branch can be created by clicking on the button *branch: master* as shown here.



organizational/contribute/img/create_branch_example.PNG

Just type the name of your new branch, hit enter and your branch will be created. To learn more about branches it is recommended to take a look here <https://git-scm.com/book/en/Git-Branching-What-a-Branch-Is>.

Note: Before you start editing make sure that you are in the right branch!

Create a new folder/file

If you want to add a completely new issue to the documentation, you have to create a new file (and maybe even folder). As you will see there is no possibility to create an empty folder. You always have to create a new file as well! This can be done here



If you click on *create new file here* you can first change into another folder by typing the *foldername* followed by */*. If this folder doesn't exist until now, one will be created. To create a new file in this folder just type *filename.txt* into the box and hit enter. A short example on how to manage this is given here <http://i.stack.imgur.com/n3Wg3.gif>.



```
organizational/contribute/img/create_file_test.PNG
```

Now a black box will appear where you can add your comments. To save the file, hit the green *Commit New File* button at the bottom.

Add images

This section is about adding images to your documentation. Providing that you've read and done the steps described above you can now follow those further steps.

Install and set up Git

To add images to your documentation you have to get your repository onto your local machine. So far you only had your repository on the web. To be able to work on your local machine as well, you have to install *git*. To do so, type:

```
sudo apt-get install git
```

(Usually git has already been installed during geonode installation)

Before you go further you should do some setup steps (can be found here: <https://help.github.com/articles/set-up-git>).

Clone repository

We assume you have already forked the geonode repository. If not, please do so following [_link](#) and return back if ready.

Until now your repository only exists on the web! To get your forked repository on to your machine, you have to *clone* it. To do so, open a terminal, go to the folder where you want the project to be and type:

```
git clone https://github.com/your_username/geonode.git my_geonode
```

Now change the active directory to the newly cloned geonode directory using:

```
cd my_geonode
```

To keep track of the original repository (the geonode repository where you forked from), you need to add a remote named *upstream*. Therefore type:

```
git remote add upstream https://github.com/GeoNode/geonode.git
```

By typing:

```
git fetch upstream
```

Changes not present in your local repository will be pulled in without modifying your files.

Add folder with images

Warning: If you've already made some changes and commits to your repository on the web (during cloning the repository and now), you have to update your repository on the local machine!

Therefore you have to run the following commands:

```
git fetch origin  
git merge
```

Or instead you could use:

```
git pull
```

Your repository should now be up to date! For more information on those commands go to <https://git-scm.com/docs>.

Note: If you’ve created a new branch, and you want to add the new folder to this branch, make sure you are working on this branch!

Typing:

```
git status
```

will show you the current branch. To change this you have to run this command (*your_branch* is the name of the branch you want to change in):

```
git checkout your_branch
```

Now you can easily add a new folder containing images to your repository. Go to the repository on your local machine and decide where you want your new folder containing the images to be (e.g in *docs_example*). There create a new folder (e.g. *images*) and add the images manually. Once you’ve done this, open a terminal and direct to the folder *docs_example*. To add the folder *images* including all content to the repository, type:

```
git add images
```

If this command doesn’t work, check your path, maybe it is incorrect!

Remark: In order to commit and push the folder, it must not be empty!

The next step is to commit the folder/files:

```
git commit -m 'Message'
```

Instead of ‘Message’ write something like ‘add images’. To push the files to the repository type:

```
git push
```

Now you are able to see the folder on the web as well!

Include images

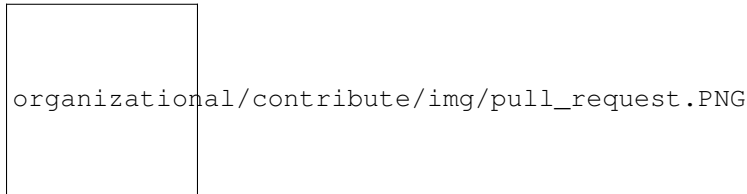
To include the images in to your documentation, you have to add the following lines to your file:

```
.. image:: images/test_img.png
```

Note: Be aware that everytime you commit something on the web, you have to make *git pull* on your machine, to keep it up to date!

Pull Request

If you are done with your changes, you can send a pull request. This means, that you let the core developers know that you have done some changes and you would like them to review. They can hit accept and your changes will go in to the main line. The *pull request* can be found [here](#).



3.4.1.3.9 How to write Documentation

GeoNode uses reStructuredText with [Sphinx](#) . Writing style should follow the same policies as GeoServer does in their [Documentation Style Guide](#)

Sphinx Syntax

This page contains syntax rules, tips, and tricks for using Sphinx and reStructuredText. For more information, please see:

- [RST Quick Reference](#)
- [Comprehensive guide to reStructuredText](#)
- [Sphinx reStructuredText Primer](#)

Basic markup

A reStructuredText document is written in plain text. Without the need for complex formatting, one can be composed simply, just like one would any plain text document. For basic formatting, see this table:

Format	Syntax	Output
Italics	<i>*italics*</i> (single asterisk)	<i>italics</i>
Bold	**bold** (double asterisk)	bold
Monospace	<code>“ monospace “</code> (double back quote)	monospace

Sections, subtitles and titles

Use sections to break up long pages and to help Sphinx generate tables of contents.

The top of the page (i.e. the title) should have an equals sign (=) above and below:

```
=====
Camel Spotting
=====
```

Level 2 section headers should have an equals sign (=) below the section name with same length as name:

```
I am a level 2 header
=====
```

Level 3 sections should have a single minus symbol (-):

```
I am a level 3 header
-----
```

Level 4 sections should have a single dot, period symbol (.):

```
I am a level 4 header
.....
```

Level 5 sections should have a single dot, period symbol (.):

```
I am a level 5 header
+++++
```

Page labels

Ensure every page has a label. For example if the page is named `foo_bar.txt` then the page should have the label at the top of the file (line1)

```
.. _foo_bar:
```

Other pages can then link to that page by using the following code:

```
:ref:`foo_bar`
```

Linking

Links to other pages should never be titled as “here”. Sphinx makes this easy by automatically inserting the title of the linked document.

Using the following code:

```
:ref:`linking`
```

And here is the link in action [Linking](#) to use the link place this code some where in your open file:

```
.. _linking:
```

To insert a link to an external website:

```
`Text of the link <http://docs.geoserver.org/latest/en/docguide/style.html>`_
```

The resulting link would look like this: [Text of the link](http://docs.geoserver.org/latest/en/docguide/style.html)

Lists

There are two types of lists, bulleted lists and numbered lists.

A **bulleted list** looks like this:

- An item
- Another item
- Yet another item

This is accomplished with the following code:

```
* An item
* Another item
* Yet another item
```

A **numbered list** looks like this:

1. First item
2. Second item
3. Third item

This is accomplished with the following code:

```
#. First item
#. Second item
#. Third item
```

Note that numbers are automatically generated, making it easy to add/remove items.

List-tables

Bulleted lists can sometimes be cumbersome and hard to follow. When dealing with a long list of items, use list-tables. For example, to talk about a list of options, create a table that looks like this:

Shapes	Description
Square	Four sides of equal length, 90 degree angles
Rectangle	Four sides, 90 degree angles

This is done with the following code:

```
.. list-table::
   :widths: 20 80
   :header-rows: 1

   * - Shapes
     - Description
   * - Square
     - Four sides of equal length, 90 degree angles
   * - Rectangle
     - Four sides, 90 degree angles
```

Notes and warnings

To emphasize something Sphinx has two ways, a note and a warning. They work the same, and only differ in their coloring. You should use notes and warnings sparingly, however, as adding emphasis to everything makes the emphasis less effective.

Here is an example of a note:

Note: This is a note.

This note is generated with the following code:

```
.. note:: This is a note.
```

Similarly, here is an example of a warning:

Warning: Beware of dragons.

This warning is generated by the following code:

```
.. warning:: Beware of dragons.
```

Images

Add images to your documentation when possible. Images, such as screen shots, are a very helpful way of making documentation understandable. When making screenshots, try to crop out unnecessary content (browser window, desktop, etc). Avoid scaling the images, as the Sphinx theme automatically resizes large images. It is also helpful to include a caption underneath the image.



Fig. 333: *The GeoNode logo as shown on the homepage.*

This image is generated by the following code:

```
.. figure:: img/logo.png
   :align: center

   *The GeoNode logo as shown on the homepage.*
```

In this example, the image file exists in the same directory as the source page. If this is not the case, you can insert path information in the above command.

External files

Text snippets, large blocks of downloadable code, and even zip files or other binary sources can all be included as part of the documentation. To include files as part of the build process, use the following syntax:

```
:download:`An external file <README>`
```

The result of this code will generate a standard link to an `external file`

Reference files and paths

Use the following syntax to reference files and paths:

```
:file:`myfile.txt`
```

This will output: `myfile.txt`.

You can reference paths in the same way:

```
:file:`path/to/myfile.txt`
```

This will output: `path/to/myfile.txt`.

For Windows paths, use double backslashes:

```
:file:`C:\\myfile.txt`
```

This will output: `C:\\myfile.txt`.

If you want to reference a non-specific path or file name:

```
:file:`{your/own/path/to}/myfile.txt`
```

This will output: `your/own/path/to/myfile.txt`

Reference commands

Reference commands (such as **make**) with the following syntax:

```
:command:`make`
```

Reference an element in a GUI

Use the following syntax to direct a user to click a link or look to a certain area of the GUI:

```
:guilabel:`Main Menu`
```

This will output: *Main Menu*.

Menu traversal

Direct a user through a menu with the following syntax:

```
:menuselection:`Start Menu --> Programs --> Geonode`
```

This will output *Start Menu → Programs → Geonode*.

Show Source

Every page in the GeoNode documentation has a link for `Show Source` under the Table of Contents on the right side of the page. This allows for easy reverse engineering of unfamiliar markup. When in doubt, look at the source!

3.4.1.3.10 How to Translate the Documentation

All documentation is written in English as the master language but can be translated into your native language. This write up will show you how to translate the documentation. There are different methods to translate the documentation and here we list a few.

Translate using Github Locally

Here we assume you have GitHub running locally and that you are familiar with the command line.

1. Get latest version from GitHub:

```
$ git pull
```

2. Get latest version from Transifex (it should be synced with Github, but just in case):

```
$ tx pull -a
```

3. Edit your files for example (if using vi):

```
$ vi i18n/it/LC_MESSAGES/index.po
```

4. Build the documentation now, before committing and pushing, build the doc to see if everything worked smoothly:

```
$ make html LANG=it
```

5. Push changes if everything worked well, push the changes:

```
$ tx push -s -t
```


CHAPTER 4

Need Help?

Having trouble? Cant find what you are looking for? We'd like to help!

- Search for information in the [archives of the GeoNode mailing list](#), or [subscribe and post a question](#).
- Join the GeoNode chat in [gitter.im/GeoNode](#).
- Ask a question in the [#geonode IRC channel](#) using [Freenode's web based client](#).
- Report bugs with GeoNode in our [issue tracker](#).

—
[_geoserver_adv_config](#), 870

g

[geoserver.add_geotiff](#), 599
[geoserver.add_shp](#), 588
[geoserver.add_sqllayers](#), 613
[geoserver.add_style](#), 646
[geoserver.add_wfscascade](#), 610
[geoserver.add_wmscascade](#), 599
[geoserver.adding_base_types](#), 587
[geoserver.adding_data](#), 585
[geoserver.advanced_gdal](#), 787
[geoserver.creating_setting](#), 585
[geoserver.crs_handling](#), 876
[geoserver.data_format](#), 626
[geoserver.db_pooling](#), 879
[geoserver.example1](#), 788
[geoserver.example2](#), 793
[geoserver.example3](#), 798
[geoserver.gs_data_dir](#), 585
[geoserver.gsproduction](#), 871
[geoserver.imagemosaic_footprint](#), 774
[geoserver.introducing_rest](#), 493
[geoserver.jmeter](#), 896
[geoserver.mosaic_pyramid](#), 766
[geoserver.parameters](#), 870
[geoserver.postgis_lay](#), 596
[geoserver.pretty_maps](#), 685
[geoserver.processing](#), 752
[geoserver.raster_data](#), 752
[geoserver.rest](#), 493
[geoserver.shp_postgis](#), 591
[geoserver.structure](#), 586
[geoserver.using_rest](#), 494
[geoserver.vector_data](#), 810

Symbols

`_geoserver_adv_config (module)`, 870

G

`geoserver.add_geotiff (module)`, 599
`geoserver.add_shp (module)`, 588
`geoserver.add_sqllayers (module)`, 613
`geoserver.add_style (module)`, 646
`geoserver.add_wfscascade (module)`, 610
`geoserver.add_wmascascade (module)`, 599
`geoserver.adding_base_types (module)`, 587
`geoserver.adding_data (module)`, 585
`geoserver.advanced_gdal (module)`, 787
`geoserver.creating_setting (module)`, 585
`geoserver.crs_handling (module)`, 876
`geoserver.data_format (module)`, 626
`geoserver.db_pooling (module)`, 879
`geoserver.example1 (module)`, 788
`geoserver.example2 (module)`, 793
`geoserver.example3 (module)`, 798
`geoserver.gs_data_dir (module)`, 585
`geoserver.gsproduction (module)`, 871
`geoserver.imagemosaic_footprint (module)`,
774
`geoserver.introducing_rest (module)`, 493
`geoserver.jmeter (module)`, 896
`geoserver.mosaic_pyramid (module)`, 766
`geoserver.parameters (module)`, 870
`geoserver.postgis_lay (module)`, 596
`geoserver.pretty_maps (module)`, 646, 677, 685,
748
`geoserver.processing (module)`, 752
`geoserver.raster_data (module)`, 752
`geoserver.rest (module)`, 493
`geoserver.shp_postgis (module)`, 591
`geoserver.structure (module)`, 586
`geoserver.using_rest (module)`, 494
`geoserver.vector_data (module)`, 810