# Table of contents

1. **What is GeoNode** ................................. 2
   1.1.1 **Showcase** ................................. 2
   1.1.2 **Most useful links** ......................... 2

2. **Licensing** ................................... 3

3. **Current Version and Features** .................. 3

4. **Get in touch with the community** ............... 3

5. **Roadmap** .................................. 3

6. **GeoNode Basics** ................................ 4
   1.6.1 *With GeoNode, non-specialized users can share data and create interactive maps.* ........................................... 4
   1.6.2 **Geospatial data storage** ...................... 4
   1.6.3 **Data mixing, maps creation** .................. 6
   1.6.4 **GeoNode as a building block** .................. 6
   1.6.5 **Convinced! Where do I sign?** ................. 8

7. **Supported Browsers** ............................ 8

8. **Online Demo** .................................. 9

9. **Quick Installation Guide** ....................... 10
   1.9.1 **Introduction** ................................ 10
   1.9.2 **Recommended Minimum System Requirements** .... 10
   1.9.3 **Install via Docker** ......................... 10

10. **GeoNode Users Guide** ......................... 12
    1.10.1 **Accounts and User Profile** .................. 12
    1.10.2 **Interacting with Users and Groups** .......... 22
    1.10.3 **Data** .................................. 34
    1.10.4 **Managing Documents** ....................... 40
    1.10.5 **Managing Datasets** .......................... 61
    1.10.6 **Managing Maps** ............................. 97
    1.10.7 **Dashboard** ................................ 157
    1.10.8 **GeoStory** ................................ 159
    1.10.9 **Publishing Data** ............................ 163
    1.10.10 **Using GeoNode with Other Applications** .... 163
    1.10.11 **Dynamic Extra Metadata** ................... 177
    1.10.12 **Download Handlers** ........................ 183
    1.10.13 **Settings** ................................ 183
    1.10.14 **CODE** .................................. 183

11. **GeoNode Basic Installation** .................... 184
    1.11.1 **Overview** ................................ 184
    1.11.2 **First Step: Deploy GeoNode on a local server** .... 184
    1.11.3 **Second Step: Deploy GeoNode on a production server** .... 187
    1.11.4 **Customize .env to match your needs** .......... 190
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.12 GeoNode Advanced Installation</td>
<td>191</td>
</tr>
<tr>
<td>1.12.1 GeoNode Vanilla</td>
<td>191</td>
</tr>
<tr>
<td>1.12.2 GeoNode Project</td>
<td>222</td>
</tr>
<tr>
<td>1.12.3 Further Production Enhancements</td>
<td>231</td>
</tr>
<tr>
<td>1.13 GeoNode Settings</td>
<td>240</td>
</tr>
<tr>
<td>1.13.1 Settings</td>
<td>240</td>
</tr>
<tr>
<td>1.14 Docker Environmental variables</td>
<td>289</td>
</tr>
<tr>
<td>1.14.1 Docker Environmental variables</td>
<td>289</td>
</tr>
<tr>
<td>1.15 Customize the Look and Feel</td>
<td>293</td>
</tr>
<tr>
<td>1.15.1 GeoNode Themes</td>
<td>293</td>
</tr>
<tr>
<td>1.15.2 Theming your GeoNode Project</td>
<td>293</td>
</tr>
<tr>
<td>1.16 GeoNode permissions</td>
<td>301</td>
</tr>
<tr>
<td>1.16.1 Permissions</td>
<td>301</td>
</tr>
<tr>
<td>1.17 Read-Only and Maintenance Mode</td>
<td>304</td>
</tr>
<tr>
<td>1.17.1 Read-Only and Maintenance Modes</td>
<td>304</td>
</tr>
<tr>
<td>1.18 Harvesting resources from remote services</td>
<td>306</td>
</tr>
<tr>
<td>1.18.1 Harvesting resources from remote services</td>
<td>306</td>
</tr>
<tr>
<td>1.19 GeoNode Backup and Restore</td>
<td>314</td>
</tr>
<tr>
<td>1.19.1 Full GeoNode Backup &amp; Restore</td>
<td>314</td>
</tr>
<tr>
<td>1.20 GeoNode Components and Architecture</td>
<td>343</td>
</tr>
<tr>
<td>1.20.1 OAuth2 Security: Authentication and Authorization</td>
<td>343</td>
</tr>
<tr>
<td>1.21 Hardening GeoNode</td>
<td>388</td>
</tr>
<tr>
<td>1.21.1 Publish on other than HTTP port (for e.g. 8082)</td>
<td>388</td>
</tr>
<tr>
<td>1.22 Social Login</td>
<td>389</td>
</tr>
<tr>
<td>1.22.1 GeoNode Social Accounts</td>
<td>389</td>
</tr>
<tr>
<td>1.23 GeoNode Django Contrib Apps</td>
<td>391</td>
</tr>
<tr>
<td>1.23.1 Geonode auth via LDAP</td>
<td>391</td>
</tr>
<tr>
<td>1.24 GeoNode Admins Guide</td>
<td>397</td>
</tr>
<tr>
<td>1.24.1 Accessing the panel</td>
<td>397</td>
</tr>
<tr>
<td>1.24.2 Reset or Change the admin password</td>
<td>398</td>
</tr>
<tr>
<td>1.24.3 Simple Theming</td>
<td>400</td>
</tr>
<tr>
<td>1.24.4 Add a new user</td>
<td>405</td>
</tr>
<tr>
<td>1.24.5 Activate/Disable a User</td>
<td>408</td>
</tr>
<tr>
<td>1.24.6 Change a User password</td>
<td>408</td>
</tr>
<tr>
<td>1.24.7 Promoting a User to Staff member or superuser</td>
<td>411</td>
</tr>
<tr>
<td>1.24.8 Creating a Group</td>
<td>411</td>
</tr>
<tr>
<td>1.24.9 Managing a Group</td>
<td>418</td>
</tr>
<tr>
<td>1.24.10 Group based advanced data workflow</td>
<td>425</td>
</tr>
<tr>
<td>1.24.11 Manage profiles using the admin panel</td>
<td>427</td>
</tr>
<tr>
<td>1.24.12 Manage datasets using the admin panel</td>
<td>428</td>
</tr>
<tr>
<td>1.24.13 Manage the maps using the admin panel</td>
<td>430</td>
</tr>
<tr>
<td>1.24.14 Manage the documents using the admin panel</td>
<td>430</td>
</tr>
<tr>
<td>1.24.15 Manage the base metadata choices using the admin panel</td>
<td>432</td>
</tr>
<tr>
<td>1.24.16 Announcements</td>
<td>438</td>
</tr>
<tr>
<td>1.24.17 Menus, Items and Placeholders</td>
<td>443</td>
</tr>
<tr>
<td>1.24.18 OAuth2 Access Tokens</td>
<td>446</td>
</tr>
<tr>
<td>1.25 GeoNode Management Commands</td>
<td>451</td>
</tr>
<tr>
<td>1.25.1 Migrate GeoNode Base URL</td>
<td>451</td>
</tr>
<tr>
<td>1.25.2 Update Permissions, Metadata, Legends and Download Links</td>
<td>453</td>
</tr>
<tr>
<td>1.25.3 Loading Data into GeoNode</td>
<td>459</td>
</tr>
<tr>
<td>1.25.4 Thesaurus Import and Export</td>
<td>489</td>
</tr>
<tr>
<td>1.25.5 Create Users and Super Users</td>
<td>489</td>
</tr>
<tr>
<td>1.25.6 Batch Sync Permissions</td>
<td>493</td>
</tr>
<tr>
<td>1.25.7 Delete Certain GeoNode Resources</td>
<td>494</td>
</tr>
</tbody>
</table>
1.25.8 Async execution over http ........................................ 498
1.26 Changing the default Languages .................................................. 505
  1.26.1 Changing the Default Language ........................................... 505
  1.26.2 GeoNode Configuration ...................................................... 506
  1.26.3 Additional Steps ............................................................ 506
  1.26.4 Restart ................................................................. 507
1.27 GeoNode Upgrade from older versions .......................................... 507
  1.27.1 Upgrade from 3.2.x / 3.3.x .............................................. 507
1.28 GeoNode Async Signals ............................................................ 509
  1.28.1 Supervisord and Systemd .................................................. 509
  1.28.2 Celery ............................................................... 509
  1.28.3 Rabbitmq and Redis ...................................................... 509
  1.28.4 How to: Async Upload via API ......................................... 509
1.29 GeoNode Add a thesaurus .......................................................... 510
  1.29.1 Introduction ........................................................... 510
  1.29.2 Upload via the Admin panel ............................................. 511
  1.29.3 Import via the load_thesaurus command ............................ 511
  1.29.4 Configure a thesaurus in GeoNode ................................. 512
  1.29.5 Apply a thesaurus to a resource .................................... 514
  1.29.6 Exporting a thesaurus as RDF via the dump_thesaurus command .................................................. 514
1.30 Participate in the Discussion ....................................................... 515
  1.30.1 Join the community, ask for help or report bugs ................ 515
1.31 Write Documentation ............................................................. 515
  1.31.1 How to contribute to GeoNode’s Documentation .................. 515
1.32 Provide Translations .............................................................. 518
  1.32.1 Contribute to Translations ............................................. 518
1.33 Write Code .............................................................................. 524
1.34 Frontend Development ............................................................... 524
  1.34.1 Frontend development .................................................... 524
1.35 GeoNode API ........................................................................... 525
  1.35.1 API v2 - Schema .......................................................... 525
  1.35.2 API usage examples ..................................................... 565
1.36 How to Develop ................................................................. 579
  1.36.1 Start to develop with Docker ........................................... 579
  1.36.2 How to Install GeoNode-Core for development ............... 581
  1.36.3 How to run GeoNode Core for development .................... 589
  1.36.4 How to run GeoNode Project for development ............... 589
  1.36.5 Start MapStore2 client in development mode ................ 589
  1.36.6 Workshops ........................................................... 591

HTTP Routing Table ................................................................. 608
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).
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 visualization. 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.

1.1.1 Showcase

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 Showcase.

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

For a live demo see also Online Demo

1.1.2 Most useful links

General

- Project homepage: https://geonode.org
- Repository: https://github.com/GeoNode/geonode
- Official Demo: http://master.demo.geonode.org
- GeoNode Wiki: https://github.com/GeoNode/geonode/wiki
- Issue tracker: https://github.com/GeoNode/geonode-project/issues
In case of sensitive bugs like security vulnerabilities, please contact a GeoNode Core Developer directly instead of using issue tracker. We value your effort to improve the security and privacy of this project!

Related projects

- GeoNode Project: https://github.com/GeoNode/geonode-project
- GeoNode at Docker: https://hub.docker.com/u/geonode

1.2 Licensing

GeoNode is Copyright 2018 Open Source Geospatial Foundation (OSGeo).

GeoNode 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. GeoNode 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.

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

1.3 Current Version and Features

GeoNode current version: 3.2.0

Main Features: State of GeoNode

1.4 Get in touch with the community

GeoNode is an open source project and contributors are needed to keep this project moving forward. Learn more on how to contribute on our Community Bylaws.

- User Mailing List: https://lists.osgeo.org/cgi-bin/mailman/listinfo/geonode-users
- Developer Mailing List: https://lists.osgeo.org/cgi-bin/mailman/listinfo/geonode-devel
- Gitter Chat: https://gitter.im/GeoNode/general

1.5 Roadmap

GeoNode’s development roadmap is documented in a series of GeoNode Improvement Projects (GNIPS). They are documented at GeoNode Wiki.

GNIPS are considered to be large undertakings which will add a large amount of features to the project. As such they are the topic of community discussion and guidance.

The community discusses these on the developer mailing list: http://lists.osgeo.org/pipermail/geonode-devel/
1.6 GeoNode Basics

is a platform for the management and publication of geospatial data. It brings together mature open-source software projects under an easy to use interface.

1.6.1 With GeoNode, non-specialized users can share data and create interactive maps.

1.6.2 Geospatial data storage

GeoNode allows users to upload vector data (currently shapefiles, json, csv, kml and kmz) and raster data in their original projections using a web form.

Vector data is converted into geospatial tables on a DB, satellite imagery and other kinds of raster data are retained as GeoTIFFs.

Special importance is given to standard metadata formats like ISO 19139:2007 / ISO 19115 metadata standards.

As soon as the upload is finished, the user can fill the resource metadata in order to make it suddenly available through the CSW (OGC Catalogue Service) endpoints and APIs.

Users may also upload a metadata XML document (ISO, FGDC, and Dublin Core format) to fill key GeoNode metadata elements automatically.

Similarly, GeoNode provides a web based styler that lets the users to change the data portrayals and preview the changes at real time.
Fig. 1: GeoNode simplified architecture
1.6.3 Data mixing, maps creation

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

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

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

1.6.4 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.
1.6.5 Convinced! Where do I sign?

The next steps are:

1. Make a ride on the Online Demo
2. Follow the Quick Installation Guide in order to play with your own local instance and access all the admin functionalities
3. Read the documentation starting from the user guide to the admin guide
4. Subscribe to the geonode-users and/or geonode-devel mailing lists to join the community. See also the section Get in touch with the community for more info.

Thanks for your interest!

1.7 Supported Browsers

GeoNode is known to be working on all modern web browsers.

This list includes (but is not limited to):

- Google Chrome.
- Apple Safari.
- Mozilla Firefox.
- Microsoft Edge.
Note: The vast majority of GeoNode developers prefer using Google Chrome.

1.8 Online Demo

Note: Disclaimer we do not guarantee for any data published on this Demo Site. Publish the data at your own risk. Every dataset will be removed automatically every Sunday. If you find some dataset that shouldn’t be there, please write suddenly to developers and maintainers.

See the section Get in touch with the community for details.

A live demo of the latest stable build is available at http://master.demo.geonode.org/.

![Online Demo @ master.demo.geonode.org](image)

Fig. 2: Online Demo @ master.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, every sunday all the datasets will be wiped out. Users, passwords and groups will be preserved. It should hopefully allow you to easily and quickly make a tour of the main capabilities of GeoNode.

Warning: This GeoNode instance is configured with standards settings and a very low security level. This is a demo only not to be considered a really production ready system. For a complete list of settings, refer to the section: Settings
1.9 Quick Installation Guide

1.9.1 Introduction

The following is a quick guide to get started with GeoNode in most common operating systems.

**Note:** For a full setup and deployment, please refer to the complete installation guides

This is meant to be run on a fresh machine with no previously installed packages or GeoNode versions.

**Warning:** The methods presented here are meant to be used for a limited internal demo only. Before exposing your GeoNode instance to a public server, please read carefully the hardening guide

1.9.2 Recommended Minimum System Requirements

A definite specification of technical requirements is difficult to recommend. Accepted performance is highly subjective. Furthermore, the performance depends on factors such as concurrent users, records in the database or the network connectivity of your infrastructure.

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

- 8GB of RAM (16GB or more preferred for a production deployment).
- 2.2GHz processor with 4 cores. (Additional processing power may be required for multiple concurrent styling renderings)
- 30 GB software disk usage (Reserved to OS and source code only).
- Additional disk space for any data hosted with GeoNode, data stored on the DataBase and tiles cached with GeoWebCache. For db, spatial data, cached tiles, and “scratch space” useful for administration, a decent baseline size for GeoNode deployments is between 50GB and 100GB.
- 64-bit hardware **strongly** recommended.

1.9.3 Install via Docker

Docker is a free software platform used for packaging software into standardized units for development, shipment and deployment.

**Note:** credits to Docker

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 documentation will refer to the usage of Docker on a Ubuntu host, but you can of course run Docker also on other Linux distributions, Windows and Mac.

For the steps to setup Docker on Ubuntu you can jump ahead to Ubuntu 22.04

1.10 GeoNode Users Guide

1.10.1 Accounts and User Profile

In GeoNode many contents are public so unregistered users have read-only access to public maps, datasets, documents, geostory and dashboard. In order to create maps, dashboard or geostory and add datasets or documents, edit the data and share these resources with other users, you need to sign in.

GeoNode is primarily a social platform, 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 preferences, connections with social networks and email addresses.

Creating a New Account

To take full advantage of all the GeoNode features you need a user account. Follow these step to create a new one.

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.

   ![Register Link](image)

   Fig. 3: Register Link

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

3. You will be automatically logged in and redirected to the Profile page. An email will be sent confirming that you have signed up. If no errors occur during the registration, the following alerts will appear on the screen:

To log out click on the Log out link of the user menu.

You have to confirm this action as described in the picture below.
Sign up

Create a new local account

**E-mail**

john.smith@gmail.com

**Username**

johnsmith

**Captcha**

I'm not a robot

**Password**

******

**Password (again)**

******

Sign up

---

**Fig. 4:** Registering for a new account

Confirmation e-mail sent to john.smith@mail.com.

Successfully signed in as johnsmith.

**Fig. 5:** Alerts
Fig. 6: *Logout link*

Fig. 7: *Confirm Log out*
Updating the Profile

Once having an account you can enrich your profile with useful information, you can also edit or delete the existing ones. You can connect the account with your social network, associate many e-mail addresses to it and manage many options such as preferences about notifications.

You can update these information anytime from your Profile page which is accessible from the user menu.

So, click on your profile picture in the top right of the screen. A drop-down list will show. Click on Profile to enter the Profile settings page.

Fig. 8: Link to your profile

The Profile page looks like the one shown in the picture below.

Your personal information is shown under the username. At the bottom of the page are listed all the resources associated to your Profile, you can decide to view only datasets, maps, documents, dashboard or geostory by clicking on the corresponding tab.

Through the link User datasets WMS GetCapabilities document you can retrieve an XML document with the list of the available datasets.

On the right side of the page there are many useful links to edit personal information to update your Profile settings and to get in touch with other GeoNode users.

The My Activities link allows to see all your recent activities on GeoNode such as datasets uploading and maps creation. All other links and their functionalities will be described in depth in the following sections.

Editing Profile Information

Your Profile contains personal information such as your address, your telephone number, your organization and so on but it is empty by default at the beginning.

Through the Edit profile button of the Profile page (see Updating the Profile) you can set your details, including your avatar.

When finished, click Update profile. You will be redirected to the Profile page. A message will confirm the profile has been correctly updated.
Fig. 9: User profile page

Fig. 10: Recent activities
Edit Your Profile

Fig. 11: Updating Profile information

Profile johnsmith updated.

Fig. 12: Updating Profile correctly finalized
Associating your Account with an e-mail

Your account is automatically associated with the e-mail that you used to register yourself on the platform.

By clicking on Associated e-mails of the Profile page (see Updating the Profile), you will have the possibility to fill up a new e-mail address. Type it in the E-mail input filed then click on Add E-mail to perform a new association.

You can make it primary if necessary, in order to receive the notification on this address. To do that, select the e-mail that you want, then click on Make Primary.

Managing the Password

To change your password, click on the Set/Change password link of the Profile page (see Updating the Profile). You will be asked to enter your current password and the new one (two times). Click on Change my password to perform the change.

If no errors occur you will see a confirmation message.

Next time you signin, you will have to use the new password.
E-mail Addresses
The following e-mail addresses are associated with your account:

- john.smith@gmail.com Unverified Primary
- john.smith.primary@gmail.com Unverified

Make Primary | Re-send Verification | Remove

Add E-mail Address

E-mail:  john.smith.primary@gmail.cc
Add E-mail

Fig. 14: New e-mail association
E-mail Addresses
The following e-mail addresses are associated with your account:

- john.smith@gmail.com Unverified
- john.smith.primary@gmail.com Unverified Primary

- Make Primary
- Re-send Verification
- Remove

Add E-mail Address

E-mail: ____________________________

Add E-mail

Primary e-mail address set.

Fig. 15: Primary e-mail address
Password Change

Change your password here

**Current Password**

```
........
```

**New Password**

```
........
```

**New Password (again)**

```
........
```

[Change my password]

---

Fig. 16: *Change your password*

---

Password successfully changed.

---

Fig. 17: *Change password confirmation*
Setting Notification Preferences

By default GeoNode sends notifications to the users for events that they could be subscribed to such as a new dataset uploaded or a new rate added to a map. You can adjust your notification settings by clicking on the Notifications link of the Profile page (see Updating the Profile).

*Note:* Make sure to have a verified email address to which notices can be sent. If not see Associating your Account with an e-mail.

Now check/uncheck the notification types you wish to receive or not receive. It is possible to be notified for the events shown in the picture below.

1.10.2 Interacting with Users and Groups

The GeoNode platform allows you to communicate by message with other GeoNode users and groups of users.

You can also invite external users to join your GeoNode. In order to do that, click on Invite Users in the Profile page (see Updating the Profile) or in the About menu in the Home page.

You can invite your contacts typing their email addresses in the input field as shown in the picture below. Click on Submit to perform the action.

A message will confirm that invitations have been correctly sent.

*Note:* You can invite more than one user at the same time by typing the email addresses inline with a semi-colon separator.

The next sections will show you how to view information about other users and how to contact them.

Viewing other users information

Once your account is created, you can view other accounts on the system.

To see information about other users on the system, click the People link of the About menu in Home page.

You will see a list of users registered on the system.

The Search tool is very useful in case of many registered users, type the name of the user you are looking for in the input text field to filter the users list.

Select a user and click on its username to access to the user details page.

In this page the main information about the user are shown: personal information (organization) and the resources the user owns (datasets, maps, documents and other apps).

Through the User Activities link, in right side of the page, it is possible to visualize all the activities the user has been done.

The Message User link lets you to contact other users, see the next section to read more about that.
## Notification Settings

<table>
<thead>
<tr>
<th>Notification Type</th>
<th>Email</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request to download a resource</td>
<td></td>
</tr>
<tr>
<td>A request for downloading a resource was sent</td>
<td>✔</td>
</tr>
<tr>
<td>Request resource change</td>
<td></td>
</tr>
<tr>
<td>Owner has requested permissions to modify a resource</td>
<td>✔</td>
</tr>
<tr>
<td>Dataset Created</td>
<td></td>
</tr>
<tr>
<td>A Dataset was created</td>
<td>✔</td>
</tr>
<tr>
<td>Dataset Updated</td>
<td></td>
</tr>
<tr>
<td>A Dataset was updated</td>
<td>✔</td>
</tr>
<tr>
<td>Dataset Approved</td>
<td></td>
</tr>
<tr>
<td>A Dataset was approved by a Manager</td>
<td>✔</td>
</tr>
<tr>
<td>Dataset Published</td>
<td></td>
</tr>
<tr>
<td>A Dataset was published</td>
<td>✔</td>
</tr>
<tr>
<td>Dataset Deleted</td>
<td></td>
</tr>
<tr>
<td>A Dataset was deleted</td>
<td>✔</td>
</tr>
<tr>
<td>Comment on Dataset</td>
<td></td>
</tr>
<tr>
<td>A layer was commented on</td>
<td>✔</td>
</tr>
<tr>
<td>Rating for Dataset</td>
<td></td>
</tr>
<tr>
<td>A rating was given to a layer</td>
<td>✔</td>
</tr>
<tr>
<td>Map Created</td>
<td></td>
</tr>
<tr>
<td>A Map was created</td>
<td>✔</td>
</tr>
<tr>
<td>Map Updated</td>
<td></td>
</tr>
<tr>
<td>A Map was updated</td>
<td>✔</td>
</tr>
<tr>
<td>Map Approved</td>
<td></td>
</tr>
<tr>
<td>A Map was approved by a Manager</td>
<td>✔</td>
</tr>
<tr>
<td>Map Published</td>
<td></td>
</tr>
<tr>
<td>A Map was published</td>
<td>✔</td>
</tr>
<tr>
<td>Map Deleted</td>
<td></td>
</tr>
<tr>
<td>A Map was deleted</td>
<td>✔</td>
</tr>
<tr>
<td>Comment on Map</td>
<td></td>
</tr>
<tr>
<td>A map was commented on</td>
<td>✔</td>
</tr>
<tr>
<td>Rating for Map</td>
<td></td>
</tr>
<tr>
<td>A rating was given to a map</td>
<td>✔</td>
</tr>
<tr>
<td>App Created</td>
<td></td>
</tr>
<tr>
<td>A App was created</td>
<td>✔</td>
</tr>
<tr>
<td>App Updated</td>
<td></td>
</tr>
<tr>
<td>A App was updated</td>
<td>✔</td>
</tr>
<tr>
<td>Event Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>App Approved</td>
<td>A App was approved by a Manager</td>
</tr>
<tr>
<td>App Published</td>
<td>A App was published</td>
</tr>
<tr>
<td>App Deleted</td>
<td>A App was deleted</td>
</tr>
<tr>
<td>Comment on App</td>
<td>An App was commented on</td>
</tr>
<tr>
<td>Rating for App</td>
<td>A rating was given to an App</td>
</tr>
<tr>
<td>Document Created</td>
<td>A Document was created</td>
</tr>
<tr>
<td>Document Updated</td>
<td>A Document was updated</td>
</tr>
<tr>
<td>Document Approved</td>
<td>A Document was approved by a Manager</td>
</tr>
<tr>
<td>Document Published</td>
<td>A Document was published</td>
</tr>
<tr>
<td>Document Deleted</td>
<td>A Document was deleted</td>
</tr>
<tr>
<td>Comment on Document</td>
<td>A Document was commented on</td>
</tr>
<tr>
<td>Rating for Document</td>
<td>A rating was given to a document</td>
</tr>
<tr>
<td>User following you</td>
<td>Another user has started following you</td>
</tr>
<tr>
<td>User requested access</td>
<td>A new user has requested access to the site</td>
</tr>
<tr>
<td>Account activated</td>
<td>This account is now active and can log in the site</td>
</tr>
<tr>
<td>Dataset Uploaded</td>
<td>A layer was uploaded</td>
</tr>
<tr>
<td>Message received</td>
<td>New message received in one of your threads</td>
</tr>
</tbody>
</table>

Fig. 18: Notifications settings
Invite Users

Email

john.friend1@mail.com, john.friend2@mail.com, john.friend3@mail.com

Submit

Fig. 19: *Invite users to join GeoNode*

Fig. 20: *Invitations confirm message*
Fig. 21: About menu - People link
Fig. 22: List of the registered users
Fig. 23: User details

Activity Feed for julia254

- No actions yet

Fig. 24: User activities
It is also possible, in GeoNode, to see the recent activities of all users through the Recent Activities link of the user menu.

![Recent Activities link](image)

Fig. 25: Recent Activities link

In the picture below an example.

As you can see, you can decide whether to see only the activities related to datasets or those related to maps or comments by switching the tabs.

**Contacting other users**

GeoNode allows you to communicate by message with other registered users and groups.

To send a message to some user and/or groups you can follow the link Message User from your Profile page (see Updating the Profile) or from the Profile details page (see the previous section Viewing other users information) of that user.

Insert your content, type a subject and click on Send message to send the message to the users and groups you have selected.

You will be redirected to the Conversation details page related to the subject.

**The Inbox page**

You can view your conversations in your Inbox page, reachable through the Back to inbox button (see the picture above) or from the Inbox link of the user menu.

The picture below shows how your Inbox page should look like.

In Inbox all the unread messages are listed. You haven’t received any message yet so your Inbox is empty. If you switch to the All tab you can see all the conversations you are involved in.

When some user send a reply to your message your Inbox shows it, see the picture below for an example.
Fig. 26: Recent Activities
Fig. 27: Send message to users and groups
Greetings from John Smith

Sent to: julia254,

Nov. 17, 2021, 9:52 a.m. by me

Hello, I'm John Smith, I'm now registered on GeoNode!

Send Reply

Fig. 28: Your message

Fig. 29: Inbox link
Fig. 30: Inbox page

Fig. 31: All your conversations
You can open the Conversation details by clicking on the Subject link.

As you can see in the picture above, in the Conversation page you have the ability to write a quick reply. Type your message in the text box and click on Send Reply to do that.

In the Inbox page there is also the Create Message button that provides you a quick link to the message creation form.

**1.10.3 Data**

Data management tools built into GeoNode allow integrated creation of resources eg datasets, documents, link to external documents, map visualizations and other configured geonode apps. Each resource 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.

The following sections will explain more in depth what data can be managed in GeoNode and how to easily find that data.

**Data Types**

GeoNode welcome page shows a variety of information about the current GeoNode instance.

You can explore the existing data using many search tools and filters (see Finding Data) or through the links of the navigation bar at the top of the page.

There are five main types of resources that GeoNode can manage:

1. Datasets
2. Maps
3. Documents
Fig. 33: Conversation details
4. GeoStories

5. Dashboards

Each resource type has its own menu and can be reached through Datasets, Maps, Documents, GeoStories and Dashboards buttons on the navigation bar.

**Note:** GeoStories and Dashboards In the screenshot below are GeoApp resource types which are added by the client.

---

**Datasets**

Datasets are a primary component of GeoNode. Datasets are publishable resources representing a raster or vector spatial data source. Datasets also can be associated with metadata, ratings, and comments.

By clicking the Datasets link you will get a list of all published datasets. If logged in as an administrator, you will also see the unpublished datasets in the same list.

GeoNode allows the user to upload vector and raster data in their original projections using a web form.

Vector data can be uploaded in many different formats (ESRI Shapefile, KML and so on…). Satellite imagery and other kinds of raster data can be uploaded as GeoTIFFs.
Maps

Maps are a primary component of GeoNode. Maps are comprised of various datasets and their styles. Datasets can be both local datasets in GeoNode as well as remote datasets either served from other WMS servers or by web service datasets such as Google or MapQuest.

GeoNode maps also contain other information such as map zoom and extent, dataset ordering, and style.
You can create a map based on uploaded datasets, combine them with some existing datasets and a remote web service dataset, 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 datasets are automatically reprojected to web mercator for maps display, making it possible to use popular base maps such as OpenStreetMap.

Documents

GeoNode allows to publish tabular and text data and to manage metadata and associated documents. Documents can be uploaded directly from your disk (see Upload/Add Documents for further information). The following documents types are allowed: txt, .log, .doc, .docx, .ods, .odt, .slid, .qml, .xls, .xlsx, .xml, .bmp, .dxf, .gif, .jpg, .jpeg, .png, .tif, .tiff, .xml, .bmp, .dvf, .mp3, .mpga, .wav, .aif, .aiff, .aifc, .au, .3gp, .flv, .vdo, .glb, .pcd, .gltf.
Through the document detailed page is possible to view, download and manage a document.

GeoStories

GeoStory is a MapStore tool integrated in GeoNode that provides the user a way to create inspiring and immersive stories by combining text, interactive maps, and other multimedia content like images and video or other third party contents. Through this tool you can simply tell your stories on the web and then publish and share them with different groups of GeoNode users or make them public to everyone around the world.

Dashboard

Dashboard is a MapStore tool integrated in GeoNode that provides the user with a space to add many Widgets, such as charts, maps, tables, texts and counters, and can create connections between them in order to:
- Provide an overview to better visualize a specific data context
- Interact spatially and analytically with the data by creating connections between widgets
- Perform analysis on involved data/layers

Finding Data

This section will guide you to navigate GeoNode to find datasets, maps and documents and other resource types by using different routes, filters and search functions.
On every page you can find some quick search tool.

The Search box in the navigation bar (see the picture below) let you type a text and find all the resource which have to deal with that text.
When you trigger a search you are brought to the Search page which shows you the search result through all data types.
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 and filter data by Text, Types, Categories, Keywords, Owners, Regions, Group, Limitations on public access, Date and Extent.

Try to set some filter and see how the resulting data list changes accordingly. An interesting type of filter is **EXTENT**: you can apply a spatial filter by moving or zooming a map within a box as shown in the picture below.

Data can be ordered by Most recent, Less recent, Name and Popularity.

For **Users** see *Viewing other users information*.
Fig. 37: Search filter by EXTENT
1.10.4 Managing Documents

In this section all the aspects concerning Datasets will be discussed. You will learn how to upload a document and how to inspect its metadata and details. All the editing tools will be also explained.

Upload/Add Documents

GeoNode allows to share reports, conceptual notes, posters, spreadsheets, etc. A wide range of documents files can be hosted on the platform, including text files (.doc, .docx, .txt, .odt), spreadsheets (.xls, .xlsx, .ods), presentations (.ppt, .pptx, .odp), images (.gif, .jpg, .png, .tif, .tiff), PDF, zip files (.rar, .zip, .gz), SLD, XML, QML files or as External URL.

**Warning:** Only authenticated users can upload data into GeoNode.

It is possible to upload a Documents in two ways:

- From the All Resources page, by clicking Add Resource which displays a list including Upload document link:
- From the Documents page, by clicking on New button.

The Document Upload page looks like the one shown in the picture below.
Fig. 39: Document Upload from All Resources page

Fig. 40: Document Upload from Documents page
Fig. 41: Document Upload page
On *GeoNode* documents can be:

- Upload from the **Local file**
- Created with reference to **External URL**

In order to upload a document from the **Local file**, you need to:

1. Click on *Select files* button
2. Select a file from your disk.
3. Click the *Upload* button

![Select files and Upload](image)

*Fig. 42: Upload document from the select files option*

A document may refer to a remote document, without making a local copy of the remote resource. To add a document that references an **External URL** you need to:
1. Click on Add URL button
2. Select an URL
3. Select an extension from the drop-down menu
4. Click the Upload button

![Add document from the add URL option](image)

Fig. 43: Add document from the add URL option

At the end of the uploading process, by clicking on the View button, you will be driven to the document page with the Info panel open. Here it is possible to view more info, edit metadata, share, download, and delete the document. See the next section to know more about Metadata.

**Note:** If you get the following error message:

Total upload size exceeds 100.0 MB. Please try again with smaller files.

This means that there is an upload size limit of 100 MB. An user with administrative access can change the upload size limit.
limits at the admin panel.

Filling the Document Metadata

You can open the Metadata form by clicking on the Edit link in the document details menu and then Edit Metadata Link.

![Edit Metadata Link](image)

Fig. 44: Edit Metadata Link

Metadata contains all the information related to the document: they are its ID card. They provide essential information for its identification and its comprehension. Metadata also make the document more easily retrievable through search by other users.

Editing a document’s metadata is done in three steps (Basic Metadata, Location and Licenses, Optional Metadata). The first two steps are mandatory (no documents will be published if the required information are not provided) whereas the last one is optional.

1. On the Basic Metadata page, the essential information that has to be filled is:
   - **Title** of the document, which should be clear and understandable;
   - **Abstract** of the document;
   - **Creation/Publication/Revision** dates which define the time period that is covered by the document;
   - **Keywords**, which should be chosen within the available list. The contributor search for available keywords by clicking on the searching bar, or on the folder logo representing, or by entering the first letters of the desired word. Key-words should be relevant to the imported document;
   - **Category** in which the document belongs;
   - **Group** to which the document is linked.

   Once all the fields are filled, click on the blue button Next >> in the bottom right corner of the page.

2. On the Location and Licenses page, the following information should be filled:
   - **Language** of the document;
   - **Regions**, which informs on the spatial extent covered by the document. Proposed extents cover the following scales: global, continental, regional, national;
   - **Data Quality statement** (general explanation of the data producer’s knowledge about the lineage of a dataset);
   - Potential Restrictions to sharing the document should be provided in the Restrictions box.

   Click on the blue button Next >> to go ahead to the next step.

3. On the Optional Metadata page, complementary information can be added:
Fig. 45: Document Basic Metadata
Fig. 46: *Document Location and Licenses*
• Edition to indicate the reference or the source of the document;
• Purpose of the document and its objectives;
• Supplemental information that can provide a better understanding of the uploaded document;
• Maintenance frequency of the document;
• Spatial representation type used.
• Related resources to link one or multiple resources to the document. These will be visible inside the Document Information panel.

Responsible Parties, Owner and Share options are listed on the right side of the page, you can edit them.

Fig. 47: Document Optional Metadata

If all the mandatory information is filled out the document can be published, if not the Completeness progress bar warns you that something is missing.
Click on the blue button Update to save information on the system.
Document Information

From the Documents Search Page (see Finding Data) you can select the document you are interested in and click on its name to see an overview about it.

The information panel reports:

- The Info tab is active by default. This tab section shows some document metadata such as its title, the abstract, date of publication etc. The metadata also indicates the document owner, what are the topic categories the document belongs to and which regions are affected.

- The Location tab shows the spacial extent of the document.

By clicking on the copy icons you have a copy of the current Bounding Box or the Center in the clipboard which once pasted will be a WKT text.

Warning: From the Document Information the Location tab is in read only mode, to edit it see the Document Editing

- The Linked Resources tab shows the resources, that can be Datasets, Maps, GeoStories and Dashboards, linked to the document.

Note: When a resource is linked from the document the link is under the Linked by list while when a resource is linked to the document the link is under the Used by list
Fig. 48: Document Information tab
Fig. 49: Document Location tab

Fig. 50: Bounding Box and Center
Warning: From the document metadata it is possible edit the document Linked Resources

From the right toolbar on the Info panel it is possible:

• Add document in your Favorites (see Updating the Profile), clicking on star icon.
• Directly Download the document oh ShapeFile format
• Share the document url
• Change the document Title by clicking on it

You can access the document details page by clicking on View document in the overview panel. That page looks like the one shown in the picture below.

On the page of a document, the resource is either directly displayed on the page or accessible by clicking on the link provided under the title.
Exploring the Document detail menu Sections

As soon as a document is opened, the **Info** panel is shown. It reports the document metadata such as its title, abstract, date of publication etc. The metadata also indicates the user who is responsible for uploading and managing this content, as well as the group to which it is linked.

Selecting **View Metadata** from the **View** button it is possible to visualize the metadata of the document.

Under the **Resource** button, it is possible to select the **Share** option to provides the document link to share.

If you want to download the document, click on the **Download** button and the document will be downloaded automatically.

Document Editing

The **Document Information** page makes available useful menu for document editing. Click on the **Edit** link then **Edit Metadata** to see editing forms.

Setting the Document Thumbnail

From the **Metadata Form**, it is possible to **Set the Thumbnail** of the document. Click on **Choose file** to open the **Thumbnail Uploading page** and chose the image that will illustrate your document. Once this is done, click on the ok button and save. If the thumbnail has been successfully uploaded you can see it by coming back to the document list.

If no errors occur, the new selected thumbnail will be shown.
Fig. 55: *Document Sharing*

Fig. 56: *Document Metadata download*
Fig. 57: *Upload Document’s Thumbnail*
Editing the Document Metadata

You can edit the metadata of your document as described in the Filling the Document Metadata section or by using the Advanced Metadata option below.

The Advanced Metadata button takes you to a big form where all the available metadata of the document can be edited. Some information are mandatory such as the Title or the Category the document belongs to, some others are optional.

In the example shown in the picture above, the information inside the red rectangles have been changed. To save the changes click on Update, you will be redirected to the document page.
Fig. 59: Editing Metadata
Editing the Document Info

You can edit the information of your document by clicking on the Info button. From the Info panel, it is possible to upload the Thumbnail of the document.

![Upload Thumbnail](image)

Fig. 60: Upload Thumbnail

If you want this document in your Favorites (see Updating the Profile), click on the start icon.

![Favorite document](image)

Fig. 61: Favorite document

You can directly Download the document and Share the document url by clicking the buttons on the top-right of the panel.

By clicking the document Title, you can change it.

From the Locations tab, you can manage the spacial extent of the document.
Fig. 62: *Share and Download document*

Fig. 63: *Document Title*

Fig. 64: *Document Title*
Here numerical fields and a map show the spatial extent (WGS 84) for the document and you can edit it by Shift+click and drag to draw the new extent on the map and change its location by clicking the marker and drag.

Note: The Location tab is also available for Dashboard and GeoStory

**Share Options**

GeoNode encourages to publicly, share and make available for download information uploaded on the platform. By default, anyone can see and download a document. However, the document responsible can choose to limit access to the document to some contributors and/or groups.

Through the Share Link shown in the menu it is possible to manage the document share options. it opens a form where set up who can:

- None (the document is not visible).
- View Metadata (allows to view the document).
- View and Download (allows to view and download the document).
- Edit (allows to change the document metadata and attributes).
- Manage it (allows to update, delete, change share options, publish/unpublish).
See an example in the picture below.

**Note:** After making changes, always save them with the *Save* Link in the menu

Usually those with options of editing metadata and the management of a document are in charge and responsible of the document, i.e. the contributor who uploaded it has those options by default.

Once the share options are set, click *Save* link in the menu to save them.

### 1.10.5 Managing Datasets

Datasets are published resources representing raster or vector spatial data sources. Datasets can also be associated with metadata, ratings, and comments.

In this section, you will learn how to create a new dataset by uploading a local data set, add dataset info, change the style of the dataset, and share the results.

#### Datasets Uploading

The most important resource type in GeoNode is the *Dataset*. A dataset represents spatial information so it can be displayed inside a map.

To better understand what we are talking about lets upload your first dataset.

It is possible to upload a *Datasets* in two ways:

- From the *All Resources* page, by clicking *Add Resource* which displays a list including *Upload dataset* link:
- From the *Datasets* page, by clicking on *New* which displays a list including *Upload dataset* link:

The *Datasets Uploading* page looks like the one in the picture below.
Fig. 66: Changing the Document share options

Fig. 67: Dataset Upload from All Resources page
Fig. 68: Dataset Upload from Datasets page

Fig. 69: The Datasets Uploading page
Through the *Select files* button you can select files from your disk, make sure they are valid raster or vector spatial data, then you can click to *Upload* button.

A progress bar shows the operation made during the dataset upload and alerts you when the process is over.

![Upload dataset](image)

**Fig. 70: Upload dataset**

In this example the `ne_10m_airports2` ESRI Shapefile, with all its mandatory files (*shp*, *shx*, *dbf* and *prj*), has been chosen. When the process ends click on *View* button

**Note:** There are lot of free spatial dataset available in the Internet. In this example, an extract of the Berlin city center roads map from the BBBike extracts OpenStreetMap dataset has been used.

In the next paragraphs you will learn how to create a dataset from scratch, how to set share options, how to explore the dataset properties and how to edit them.

**Note:** If you get the following error message:

*Total upload size exceeds 100.0 MB. Please try again with smaller files.*

This means that there is an upload size limit of 100 MB. An user with administrative access can change the upload size limits at the *admin panel* for size limits.

Similarly, for the following error message:

*The number of active parallel uploads exceeds 5. Wait for the pending ones to finish.*

You can modify the upload parallelism limits at the *admin panel* for parallelism limits.
An interesting tool that GeoNode makes available to you is the Create dataset. It allows you to create a new vector dataset from scratch. The Dataset Creation Form is reachable through the Create dataset link shown in the picture below.

In order to create the new Dataset you have to fill out the required fields:

- **Name**
- **Title**
- **Geometry type**
Usually the datasets features should have some attributes that enrich the amount of information associated with each of them. Through the Add Attribute button you can add new attributes.

Once the form has been filled out, click on Create. You will be redirected to the Dataset Page (see Dataset Information). Now your Dataset is created but is still empty, no features have been added yet. See the Dataset Editing section to learn how to add new features.
Using Remote Services

In GeoNode you can add new datasets not only by loading them from your disk but also using Remote Services. In this section you will learn how to add a new service and how to load resources in GeoNode through that.

Let’s try it!

Click on the Remote Services link of the New menu in the navigation bar.

![Remote Services link]

The page that opens will contain the list of the available services.

To configure a new service:

- click on Add Remote Service
- type the Service URL
- select the Service Type
- click on Create

GeoNode supports four types of remote services:

- **Web Map Service**
  
  Generic Web Map Service (WMS) based on a standard protocol for serving georeferenced map images over the Internet. These images are typically produced by a map server (like GeoServer) from data provided by one or more distributed geospatial databases. Common operations performed by a WMS service are: GetCapabilities (to retrieves metadata about the service, including supported operations and parameters, and a list of the available datasets) and GetMap (to retrieves a map image for a specified area and content).

  **Note:** Lots of WMS services are available on the internet, in this example we used the `https://demo.geo-solutions.it/geoserver/wms`.

- **GeoNode Web Map Service**
  
  Generally a WMS is not directly invoked; client applications such as GIS-Desktop or WEB-GIS are used that provide the user with interactive controls. A GeoNode WMS automatically performs some operations and lets you to immediately retrieve resources.
## Remote Services

<table>
<thead>
<tr>
<th>Title</th>
<th>URL</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>GeoServer Web Map Service</td>
<td><a href="https://portal.ric.gob.gt/geoserver/wms">https://portal.ric.gob.gt/geoserver/wms</a></td>
<td>Web Map Service</td>
</tr>
<tr>
<td>GeoServer Web Map Service</td>
<td><a href="https://demo.geo-solutions.it/geoserver/wms">https://demo.geo-solutions.it/geoserver/wms</a></td>
<td>Web Map Service</td>
</tr>
<tr>
<td>Mapa Base de España</td>
<td><a href="https://www.ign.es/wms-inspire/ign-base">https://www.ign.es/wms-inspire/ign-base</a></td>
<td>Web Map Service</td>
</tr>
<tr>
<td>World Imagery</td>
<td><a href="https://server.arcgis.com/arcgis/rest/services/World_Imagery/MapServer">https://server.arcgis.com/arcgis/rest/services/World_Imagery/MapServer</a></td>
<td>ArcGIS REST MapServer</td>
</tr>
<tr>
<td>USA Topo Maps</td>
<td><a href="http://services.arcgis.com/arcgis/rest/services/USA_Topo_Maps/MapServer">http://services.arcgis.com/arcgis/rest/services/USA_Topo_Maps/MapServer</a></td>
<td>ArcGIS REST MapServer</td>
</tr>
<tr>
<td>USA Topo Maps</td>
<td><a href="http://services.arcgis.com/arcgis/rest/services/USA_Topo_Maps/MapServer/0">http://services.arcgis.com/arcgis/rest/services/USA_Topo_Maps/MapServer/0</a></td>
<td>ArcGIS REST MapServer</td>
</tr>
</tbody>
</table>

Fig. 76: Remote Services

### Service Type

- Web Map Service
- GeoNode (Web Map Service)
- ArcGIS REST MapServer
- ArcGIS REST ImageServer

Fig. 77: Service Types
Note: An example of GeoNode WMS is available at http://dev.geonode.geo-solutions.it/geoserver/wms.

- **ArcGIS REST MapServer**

  This map service provides basic information about the map, including the datasets that it contains, whether the map is cached or not, its spatial reference, initial and full extents, whether the service is allowed to export tiles and max tiles export count, etc. A set of operations that manage the state and contents of the service are allowed: Edit Service, Refresh, Update Tiles. The URL should follow this pattern: https://<servicecatalog-url>/services/<serviceName>/MapServer.

  Note: Try the following service to better understand how it works: https://sampleserver6.arcgisonline.com/arcgis/rest/services/USA/MapServer.

- **ArcGIS REST ImageServer**

  This Image Server allows you to assemble, process, analyze, and manage large collections of overlapping, multiresolution imagery and raster data from different sensors, sources, and time periods. You can also publish dynamic image services that transform source imagery and other raster data into multiple image products on demand—without needing to preprocess the data or store intermediate results—saving time and computer resources. In addition, ArcGIS Image Server uses parallel processing across multiple machines and instances, and distributed storage to speed up large raster analysis tasks. The URL should follow this pattern: https://<servicecatalog-url>/services/<serviceName>/ImageServer.

  Note: Try the following service to better understand how it works: https://sampleserver6.arcgisonline.com/arcgis/rest/services/Toronto/ImageServer.

Once the service has been configured, you can load the resources you are interested in through the Import Resources page where you will be automatically redirected to.

From the page where the services are listed, it is possible to click on the Title of a service. It opens the Service Details page.

Each service has its own metadata such as the Service Type, the URL, an Abstract, some Keywords and the Contact user.

You can edit those metadata through the form available from the Edit Service Metadata link of the Service Details page (see the picture below).

**Share Options**

In order to modify the Dataset Share options settings, on the detail page of the dataset click the Share link in the Resource menu. By default only owners can edit and manage datasets, anyone can view them.

Through the Share options Settings Panel you can add or remove options for users and groups. The picture below shows an example.

You can set the following share options:

- **View** (allows to view the dataset).
- **Download** (allows to view and download the dataset).
USA

**Type:** ArcGIS REST MapServer

**URL:** https://sampleserver6.arcgisonline.com/arcgis/rest/services/USA/MapServer

**Abstract:** This map contains sample data of the United States. This is a sample service hosted by ESRI, powered by ArcGIS Server. ESRI reserves the right to change or remove this service at any time and without notice.

**Keywords:** Data, Map, Query

**Contact:** johnsmith

**Service Resources**

No resources have been imported yet.

---

**Fig. 78: Remote Service metadata**

---

**Fig. 79: Change Dataset Share options**
Fig. 80: Dataset Share options settings for users and groups
• *Edit* (allows to change the dataset metadata, change attributes and properties of the datasets features and change the dataset style).

• *Manage* (allows to update, delete, change share options, publish and unpublish the dataset).

**Warning:** When assigning options to a group, all the group members will have those options. Be careful in case of editing them.

Click on *Save* link in the menu to save these settings.

**Dataset Information**

From the *Dataset Search Page* (see *Finding Data*) you can select the dataset you are interested in and click on its name to see an overview about it.

The information panel reports:

• The *Info* tab is active by default. This tab section shows some dataset metadata such as its title, the abstract, date of publication etc. The metadata also indicates the dataset owner, what are the topic categories the dataset belongs to and which regions are affected.

• The *Location* tab shows the spacial extent of the dataset.

By clicking on the copy icons you have a copy of the current *Bounding Box* or the *Center* in the clipboard which once pasted will be a WKT text.

• The *Attributes* tab shows the data structure behind the dataset. All the attributes are listed and for each of them some statistics (e.g. the range of values) are estimated (if possible).
### states

This is a dataset from **admin** / September 29th 2023

<table>
<thead>
<tr>
<th>Info</th>
<th>Location</th>
<th>Attributes</th>
<th>Linked Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td></td>
<td>states</td>
<td></td>
</tr>
<tr>
<td>Owner</td>
<td></td>
<td>admin</td>
<td></td>
</tr>
<tr>
<td>Publication</td>
<td></td>
<td>2023-09-29 12:18</td>
<td></td>
</tr>
<tr>
<td>Added to catalog</td>
<td></td>
<td>2023-09-29 12:18</td>
<td></td>
</tr>
<tr>
<td>Last catalog modification</td>
<td></td>
<td>2023-09-29 12:18</td>
<td></td>
</tr>
<tr>
<td>Resource type</td>
<td></td>
<td>dataset</td>
<td></td>
</tr>
<tr>
<td>Source</td>
<td></td>
<td>local</td>
<td></td>
</tr>
<tr>
<td>Point of contact</td>
<td></td>
<td>admin</td>
<td></td>
</tr>
<tr>
<td>Language</td>
<td></td>
<td>eng</td>
<td></td>
</tr>
<tr>
<td>Supplemental information</td>
<td></td>
<td>No information provided</td>
<td></td>
</tr>
</tbody>
</table>

Fig. 81: *Dataset Info tab*
Fig. 82: Dataset Location tab
Fig. 83: *Bounding Box and Center*

<table>
<thead>
<tr>
<th>Bounding Box (WGS84)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Min Lat</td>
<td>24.955967</td>
</tr>
<tr>
<td>Min Lon</td>
<td>-124.731422</td>
</tr>
<tr>
<td>Max Lat</td>
<td>49.371735</td>
</tr>
<tr>
<td>Max Lon</td>
<td>-66.969849</td>
</tr>
<tr>
<td>Center (WGS84)</td>
<td></td>
</tr>
<tr>
<td>Lat</td>
<td>37.163851</td>
</tr>
<tr>
<td>Lon</td>
<td>-95.850636</td>
</tr>
</tbody>
</table>

Fig. 84: *Dataset Attributes tab*

<table>
<thead>
<tr>
<th>Name</th>
<th>Label</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ogrc_fid</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>name</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>description</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>geometry</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>
• The *Linked Resources* tab shows the resources, that can be *Documents, Maps, GeoStories* and *Dashboards*, linked to the dataset.

![Fig. 85: Dataset Linked Resources tab](image)

**Note:** When a resource is *linked from* the dataset the link is shown under the *Linked by* list while when a resource is *linked to* the dataset the link is shown under the *Used by* list.

From the right toolbar on the *Info* panel it is possible:

![Fig. 86: Dataset Info toolbar](image)

• Add dataset in your *Favorites* (see *Updating the Profile*), clicking on star icon.
• Directly *Download* the dataset oh *ShapeFile* format
• *Share* the dataset url
• *Copy the WMS Capabilities url*
• Change the dataset *Title* by clicking on it

You can access the dataset details page by clicking on *View dataset* in the overview panel. That page looks like the one shown in the picture below.
Downloading Datasets

At the top of the Dataset Menu there is a Download link where it is possible to download:

- Dataset
- ISO Metadata
- Dublin Core Metadata

Clicking the ISO Metadata or the Dublin Core Metadata exports the layer metadata in xml format.

Open the Dataset option you will be able to select from a list of options of the supported export file formats.
As shown in the image above, GeoNode allows you to download a subset of data. Click on Download filtered dataset to download filtered data.

On clicking Export, the file is prepared and a notification is showed when file is ready

To download the file to your machine, click on the export dataset icon. This opens the prepared export files and you can save the files on your by clicking on the save icon on each item.

**Printing**

The MapStore based map viewer of GeoNode allows you to print the current view with a customizable layout. Click the Print option from the Menu, the Printing Window will open.

From this window you can:

- enter Title and Description;
- choose the Resolution in dpi;
- select the format
- select the coordinate
- add the scale
- add grid with label
- customize the Layout
  - the Sheet size (A3, A4);
Fig. 90: *Export Ready*

Fig. 91: *Export Results Icon*
Fig. 92: The Printing Window
– if include the legend or not;
– if to put the legend in a separate page;
– the page Orientation (Landscape or Portrait);

• customize the Legend
  – the Label Font;
  – the Font Size;
  – the Font Emphasis (bold, italic);
  – if Force Labels;
  – if use Anti Aliasing Font;
  – the Icon Size;
  – the Legend Resolution in dpi.

To print the view click on Print.

Performing Measurements

Click on the Measure option of the Menu to perform a measurement. As you can see in the picture below, this tool allows you to measure Distances, Areas and the Bearing of lines.

![The Measure Tool](image)

Fig. 93: The Measure Tool
Dataset Editing

The *Edit* link in the menu of the *Dataset Page* opens a list of options like ones shown in the picture below.

![Dataset Editing Link](image)

Fig. 94: *Dataset Editing Link*

In that options list, you can see three options listed as:

1. *Edit Data*
2. *Edit Styles*
3. *Edit Metadata*
4. *Upload Style*
5. *Upload Metadata*

In this section you will learn how to edit a *Dataset*, and its data. See *Datasets Metadata* to learn how to explore the dataset *Metadata*, how to upload and edit them. The *Styles* will be covered in a dedicated section, see *Dataset Styling*.

**Editing the Dataset Data**

The *Edit data* link of the *Dataset Editing* options opens the *Dataset* within a *Map*.

The *Attribute Table* panel of the *Dataset* will automatically appear at the bottom of the *Map*. In that panel all the features are listed. For each feature you can zoom to its extent by clicking on the corresponding *magnifying glass* icon at the beginning of the row, you can also observe which values the feature assumes for each attribute.

Click the *Edit Mode* button to start an editing session.

Now you can:

- *Add new Features*

  Through the *Add New Feature* button, it is possible to set up a new feature for your dataset. Fill the attributes fields and click to save your change. Your new feature doesn’t have a shape yet, click on to draw its shape directly on the *Map* then click on to save it.

  **Note:** When your new feature has a multi-vertex shape you have to double-click the last vertex to finish the drawing.
Fig. 95: *Editing the Dataset Data*
Fig. 96: Create New Feature
• **Delete Features**

If you want to delete a feature you have to select it on the Attribute Table and click on the delete button.

• **Change the Feature Shape**

You can edit the shape of an existing geometry dragging its vertices with the mouse. A blue circle lets you know what vertex you are moving.

Features can have multipart shapes. You can add parts to the shape when editing it.

• **Change the Feature Attributes**

When you are in Edit Mode you can also edit the attributes values changing them directly in the corresponding text fields. You can achieve this by going into the edit mode and double click in the values.

Once you have finished you can end the Editing Session by clicking on the end button.

By default the GeoNode map viewer is MapStore based, see the MapStore Documentation for further information.

**Datasets Metadata**

In GeoNode special importance is given to Metadata and their standard formats.

**Editing Metadata**

Metadata contains all the information related to the dataset. They provide essential information for its identification and its comprehension. Metadata also make the dataset more easily retrievable through search by other users.

The Metadata of a dataset can be changed through a Edit Metadata form which involves four steps, one for each type of metadata considered:

You can open the Metadata form of a Dataset by clicking the Edit Metadata link from the Edit options on the Dataset Page.

• **Basic Metadata**

The first two steps are mandatory (no datasets will be published if the required information are not provided) whereas the last two are optional.

In the first step the system asks you to insert the following metadata:

– **Thumbnail** of the dataset (click Edit to change it);
– **Title** of the dataset, which should be clear and understandable;
– **Abstract**; brief narrative summary of the content of the dataset

**Note:** The Abstract panel allows you to insert HTML code through a wysiwyg text editor

– **Creation/Publication/Revision Dates** which define the time period that is covered by the dataset;
– **Keywords**, which should be chosen within the available list. The contributor search for available keywords by clicking on the searching bar, or on the folder logo representing, or by entering the first letters of the desired word;
– **Category** which the dataset belongs to;
Fig. 97: Basic Dataset Metadata
– **Group** which the dataset is linked to.

**Location and Licenses**

![Image of Metadata for ne_10m_airports_vufl8s3](image)

**Metadata for ne_10m_airports_vufl8s3**

The following list shows what kinds of metadata you are required to enter (see also the picture below):

– **Language** of the dataset;

– **License** of the dataset;

– **DOI** of the dataset; if available, this represents the Digital Object Identifier of the resource

– **Attribution** of the dataset; authority or function assigned, as to a ruler, legislative assembly, delegate, or the like

– **Regions**, which informs on the spatial extent covered by the dataset. Proposed extents cover the following scales: global, continental, regional, national;

– **Data Quality statement** (general explanation of the data producer’s knowledge about the lineage of a dataset);

– Potential **Restrictions** on dataset sharing.

**Note:** The Data Quality statement and Restrictions panels allow you to insert HTML code through a wysiwyg text editor

**Optional Metadata**

Complementary information are:
Fig. 99: Optional Dataset Metadata
– **Edition** to indicate the reference or the source of the dataset;
– **Purpose** of the dataset and its objectives;
– **Supplemental information** that can provide a better understanding of the uploaded dataset;
– **Maintenance frequency** of the dataset;
– users who are **Responsible** for the dataset, its **Owner**, and the **Author** of its metadata;
– **Spatial representation** type used.

– **Related resources** to link one or multiple resources to the document. These will be visible inside the **Dataset Information** panel

---

**Note:** The **Purpose** and **Supplemental information** panels allow you to insert HTML code through a **wysiwyg** text editor

---

### Dataset Attributes

**Fig. 100:** Dataset Attributes Metadata for Dataset

At this step you can enrich the dataset attributes with useful information like the following:

– The **Label** displayed
– A detailed **Description**
– The **Display Order**
The Display Type; the default value is Label, which means that the value of the attribute will be rendered as a plain text. There’s the possibility to instruct GeoNode to threat the values as different media-types. As an instance, if the values of the selected attribute will contain image urls, by selecting the IMAGE Display Type you will allow GeoNode to render the image directly when querying the dataset from the maps. The same for VIDEO, AUDIO or IFRAME mime types.

The Visible flag; allows you to instruct GeoNode whether or not hiding an attribute from the Get Feature Type outcomes

It is possible to define a completely custom HTML template for the Get Feature Type outcome. That is possible by enabling the Use a custom template flag as shown in the figure below.

Fig. 101: Use a custom template

By using the keyword ${properties.<attribute_name>}, you can tell to GeoNode to render the actual value of the attribute on the map.

As an instance, the example below

Will render an HTML Table along with values as shown here below

Use next >> or << back to navigate through those steps. Once you have finished click on Update.

Some metadata are mandatory, if you miss any of that metadata the Completeness bar shows you a red message like the one in the picture below.
1.10. GeoNode Users Guide

Fig. 102: Use a custom template: HTML

Fig. 103: Use a custom template: Get Feature Info outcome
Metadata Advanced Editing

In the Edit Metadata page the Advanced Metadata button is also available.

Click on it to display the Metadata Advanced Editing Page. That page allows you to edit all the dataset metadata described in the previous paragraph. Once you have finished to edit them click on Update to save your changes.

Dataset Styling

Maps are helpful because they allow you gain a deeper understanding of your data by allowing you to visualize it in many different ways. So you can tell different stories depending on how the data is presented. For any given data or dataset, you should explore different styling options and choose the best style for that.

In GeoNode each dataset has one style refered to as a Default Style which is determined by the nature of the data you're mapping. When uploading a new dataset (see Datasets Uploading) a new default style will be associated to it.
Editing the Dataset Style

In order to edit a dataset style, open the Dataset Page (see Dataset Information) and click on Edit. Then click the Edit Style link in the options (see the picture below).

![Edit Styles button](image)

The Dataset will open in a new Map. The Styles Panel will show you all the default style for the dataset and some useful tools. By default, the style will be shown in a text editor form.

![The Styles Panel in the Map](image)

You could continue to change the style with the text editor or switch to a visualized editor using the Visual editor above the text editor.

The visual editor looks like this:

You can then edit the style by clicking on each attribute of the style.
Creating Style Rules

In order to create a new rule, Three options are provided.

1. Mark rule
2. Icon rule
3. text rule

Click on any of the buttons below to create a type of style you want.

The rule you have chosen is now added on the top with default attributes which you can edit to your preference*. If the rule has errors, the following will be shown.

You can switch the rule ordering by dragging it to the top or bottom depending on your preference.

It would be nice to change the style in order to decrease the opacity of the filling color as well as to reduce the lines width. The embedded MapStore makes available a powerful Style Editor to accomplish that tasks. In the next paragraph we will explain how.

Each rule has a delete icon on the top right which you can use to remove the rule on the style.

Click on Save on the top to save your changes.

Fig. 108: Visual Styles Panel in the Map
Fig. 109: Create new rule buttons

Fig. 110: New style
Advanced Dataset Management with MapStore

GeoNode provides the user with some advanced features for dataset manipulation such as dataset filtering, attribute edition and dataset export in different formats. In a nutshell, these feature are provided via MapStore and we will redirect the user to MapStore specific documentation for further details.

Filtering Datasets

With GeoNode you can filter a dataset via it’s attributes, direct map filter by drawing an area of interest over the map canvas and via cross-dataset filter, allowing intersection, contained and contains overlay methods. For more detail please check the MapStore documentation here.
1.10.6 Managing Maps

Maps are sets of datasets displayed together on an interactive web map. Maps can be composed in the map composer and saved as GeoNode resources. Maps can also be associated with metadata, ratings, and comments.

In this section, you will learn how to create a new map and share it.

Creating Maps

In this section, we’ll create a Map using some uploaded datasets, combine them with some other datasets from remote web services, and then share the resulting map for public viewing.

In order to create new maps you can use:

- The Create map listed after clicking the Add Resource button on the All Resources list page.

- The New button after clicking the Maps button on the menu bar.

- The Create map link in the Dataset Page (it creates a map using a specific dataset)
The new Map will open in a Map Viewer like the one in the picture below.

Using the Add dataset link, you can add a layer by clicking on one of the layers listed in the catalog. In the upper left corner the TOC button button opens the Table of Contents (TOC) of the Map. It allows to manage all the datasets associated with the map and to add new ones from the Add dataset. The TOC component makes possible to manage datasets overlap on the map by shifting their relative positions in the list (drag and drop them up or down in the list). It also allows to hide/show datasets (and ), to zoom to datasets extents ( ) and to manage their properties ( ). Once the map datasets have been settled it is possible to save the Map by clicking on the Save as under the Resources link in the map toolbar.

If you followed the steps above, you have just created your first Map. Now you should see it in the Explore Maps page, see Map Information for further details.

We will take a closer look at the Map Viewer tools in the Map tools and configuration section.
Map tools and configuration

In this section, we are going to explore all tools provided on the Map View page. From the list of available maps, you can select the map you are interested in and click View map, the map will look like this.

![The Map View](image)

**Fig. 116: The Map View**

The Map View (based on MapStore) provides the following tools:

- the **Table of Contents (TOC)** to manage the map contents;
- the **Basemap Switcher** to change the basemap (see the next paragraphs);
- the **Search Bar** to search by location, name and coordinates (see the paragraph below);
- the **Other Menu Tools** which contains the link to the datasets **Catalog**;
- the **Sidebar** which contains, by default, the link to the **Print** tool and to the **Measure** tool;
- the **Navigation Bar** and its tools such as the **Zoom** tools, the **3D Navigation** tool and the **Get Features Info** tool;
- the **Footer Tools** to manage the scale of the map, to track the mouse coordinates and change the CRS (Coordinates Reference System).

A map can be configured to use a custom **Map Viewer**, with which the list of tools available in the map can be customized.

Map Viewer

GeoNode allows for the administrator of the map to configure a custom viewer by choosing the set of plugins available for the viewer.

From the **Add viewer** option under the **Edit options** of the **Menu** a page opens and the user can:

- Select an existing viewer from the list of viewers by clicking on the **Link to viewer**
- Create a new viewer by clicking on the **Create new viewer**
Once the Create new viewer is selected an Edit Plugins page opens and, through the central vertical bar, the user can select the plugins to include in the context viewer by moving them from the Available Plugins list to the Enabled Plugins list.

To save and enable the map viewer, the user can click Save as option under the Resource options of the Menu.

The Map Viewer will be visible to all users who have permission to view the map and can be reused or modified by the user who has edit permissions on it. It will be available on the list of the resources on Home Page.

See the MapStore Documentation for more information.

Table of Contents (TOC)

In the upper left corner, click on to open the Table Of Contents, briefly TOC from now on, of the map. The TOC shows all the datasets involved with the Map and allows to manage their properties and representations on the map.

From the TOC you can:

- manage the TOC Settings (click on ) (See the MapStore Documentation for more information.)
- manage the datasets Overlap;
- filter the datasets list by typing text in the Filter Datasets field;
- manage the datasets properties such as Opacity (scroll the opacity cursor), Visibility (click on to make the dataset not visible, click on to show it on map);
- add and manage Annotations (click on )
- manage the Dataset Settings, see the next paragraph.
Fig. 118: Enable plugins for the viewer

Fig. 119: The Table Of Contents (TOC)
Fig. 120: *Scrolling the Dataset Opacity*

Fig. 121: *The Dataset Toolbar*
Select a Dataset from the list and click on it, the Dataset Toolbar should appear in the TOC.

The Toolbar shows you many buttons:

- the Zoom to dataset extent button allows you to zoom to the dataset extent;
- the Filter layer button that acts directly on a layer with WFS available and filter its content;
- the Attribute Table button to explore the features of the dataset and their attributes (more information at Attributes Table);
- the Delete button to delete datasets (click on Delete Dataset to confirm your choice);
- the Widgets button to create Widgets (see Creating Widgets).
- the Export data button;
- the Settings button drives you through the dataset settings customization (see the next paragraph);
- the Compare tool button to Swipe or Spy the selected layer
- the Edit Style button;

Managing Dataset Settings

The Dataset Settings panel looks like the one below.

The Dataset Settings are divided in three groups:

1. General settings
2. Visibility settings
3. Style settings
4. Tiling settings

In the General tab of the Settings Panel you can customize the dataset Title, insert a Description, change/add the Dataset Group and change the Tooltip content and the Tooltip placement.

The Visibility tab where you can Change the Opacity of the layer and add the Visibility limits to display the layer only within certain scale limits

The Style tab allows you to select the style from the available layer styles and change the Width and the Height of the Legend.

Click on the Tiling tab to change the output Format of the WMS requests, the Tile Size and enable/disable the Transparent, the Use cache options and the Single Tile.

Add an Annotation

Click on the button from the TOC Toolbar to enrich the map with special features which expose additional information, mark particular position on the map and so on. From here the editor can insert a Title and a Description.

To begin, from the annotation panel, the editor add new annotation by selecting the Geometries tab.

Here the user can choose between five different types of Geometries:

1. Marker
2. Line
3. Polygon
Fig. 122: *The Dataset Settings Panel*
Fig. 123: *The Visibility tab on Settings Panel*
Fig. 124: *The Style tab on Settings Panel*
Fig. 125: The Tiling tab on Settings Panel
Fig. 126: Annotations panel
4. **Text**

5. **Circle**

See the [MapStore Documentation](#) for more information.

**Attributes Table**

When clicking on the button of the *Table of Contents (TOC)*, the *Attributes Table* panel opens at the bottom of the *Map* page.

In that panel you can navigate through the features of the dataset, zoom to their geometries by clicking on the icon and explore their attributes.

The *Attribute Tables* has a row for each feature belonging to the dataset and a column for each attribute that describes the feature.

Each column has a *Filter* input field through which you can filter the features basing on some value or expression (depending on the data type of the field).

The *Attributes Table* panel contains a *Toolbar* which makes you available some useful functionalities.

Those functionalities are:

- *Edit Mode*
<table>
<thead>
<tr>
<th>fid</th>
<th>scale_rank</th>
<th>featureclass</th>
<th>type</th>
<th>name</th>
<th>abbrev</th>
<th>location</th>
<th>gse_code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9</td>
<td>Airport</td>
<td>small</td>
<td>Sahnewal</td>
<td>LLM</td>
<td>terminal</td>
<td>VLIQ</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>Airport</td>
<td>mid</td>
<td>Solapur</td>
<td>SSE</td>
<td>terminal</td>
<td>VSLQ</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>Airport</td>
<td>mid</td>
<td>Brane-Mundra</td>
<td>IRR</td>
<td>terminal</td>
<td>VBRC</td>
</tr>
<tr>
<td>4</td>
<td>9</td>
<td>Airport</td>
<td>mid</td>
<td>Ahwaz</td>
<td>AWZ</td>
<td>terminal</td>
<td>OAWQ</td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>Airport</td>
<td>mid and military</td>
<td>Guwaltor</td>
<td>CWL</td>
<td>terminal</td>
<td>VCB</td>
</tr>
<tr>
<td>6</td>
<td>9</td>
<td>Airport</td>
<td>mid</td>
<td>Hodeshek Int'l</td>
<td>HCD</td>
<td>terminal</td>
<td>OHVD</td>
</tr>
</tbody>
</table>

Fig. 128: *The Attributes Table Panel*
Fig. 129: Filtering Features by Attribute

Fig. 130: The Attributes Table Toolbar
By clicking on you can start an editing session. It permits you to add new features, to delete or modify the existing ones, to edit geometries. See the Editing the Dataset Data section for further information.

- **Advanced Search**
  Click on a new panel opens. That panel allows you to filter features in many different ways. This functionality will be explained in depth in the Advanced Search section.

- **Zoom to page extent**
  Click on to zoom to the page extent.

- **Export Data**
  Click on to open the export/download data form.

- **Hide/show columns**
  When clicking on another panel opens inside the Attributes Table. Through that panel you can choose what columns you want to see.

- **Create a chart**
  Through the button you can open the Chart Widgets panel where many functionalities to describe and visualize the dataset data are available (see Creating Widgets).

- **Sync map with filter**
  Click on the icon to synchronize the map with the filter.

### Advanced Search

As mentioned before, GeoNode allows both an attribute based and spatial filtering. When clicking on from the dataset Attributes Table the Advanced Search panel opens and shows you three different filtering functionalities:

- In the **Attribute Filter** section you can compose a series of conditions about the attributes of the dataset. Click on to insert a new empty condition. Select the attribute you are interested in, select an operator and type a comparison value. You can group conditions through the Add Group button. Click on to perform the search.
  
  You can also decide if All the conditions have to be met, if only Any or None of them (see the red arrow in the picture above).

- The **Area of interest** filtering allows you to filter features that have some relationship with a spatial region that you draw on the map.
  Select the Filter Type (Circle, Viewport, Polygon or Rectangle), draw the spatial region of interest on the map, select a Geometric Operation (Intersects, Bounding Box, Contains or Is contained) and then click on.

- Through the **Dataset Filter** you can select only those features which comply with some conditions on other datasets of the map. You can also add conditions on attributes for those datasets.

You can read more about the Attributes Table and the Advanced Search on the MapStore Documentation.
### Attribute filter

Match **any** of the following conditions:

### Area of interest

- **Filter type**: Select...
- **Geometric operation**: Intersects

### Layer filter

- **Target layer**: Select layer

---

**Fig. 131: Advanced Search**
Fig. 132: Filtering by Attributes
1.10. GeoNode Users Guide

Fig. 133: Filtering by Area Of Interest

Fig. 134: Dataset Filtering
Creating Widgets

Widgets are graphical elements that describe the datasets data. They can be of different types such as Charts, Texts, Tables and Counters. Through the button of the Table of Contents (TOC) you can open the Widgets panel.

![Select the widget type](image)

- **Chart**: add a chart
- **Text**: add a text area
- **Table**: add a table
- **Counter**: add a counter

![Fig. 135: Creating Widgets](image)

**Chart Widgets**

Chart Widgets are graphical representations of the dataset data. They can be Bar Chart, Pie Chart or Line Chart as shown in the picture below.

Let's create a new Bar Chart.

Click on Bar Chart then select the X Attribute, the Y Attribute, the Operation and the Color you prefer. You can also display the Legend, Hide the Y axis, Hide the grid and decide what Label display into the legend.

Now you can filter the data to be considered for the chart by clicking on . We don’t need any filter so click to configure other widget options. Insert a Title and a Description and click on Save.

1.10. GeoNode Users Guide 116
Select the Chart type

**Bar Chart**
Create a bar chart to add to the map

**Pie Chart**
Create a pie chart to add to the map

**Line Chart**
Create a line chart to add to the map

Fig. 136: *Chart Widgets*
The green icon means that the chart is connected to the viewport.

At the top of the bar chat, there is the options menu of the widget where you can Download graph as png, Zoom the widgets and Reset axes.

**Text Widgets**

If you select Text on the Widgets panel you can create Text Widgets. Add a Title and the desired descriptive text, then click on .

The resulting widget looks like the following.

**Table Widgets**

Through the Table Widgets you can add the Attributes Table of the dataset to the map. You can decide to show a subset of the features, through filters, and you can select one or more columns/attributes.

You can also enter a Title for each column to be displayed as the table header in place of the Name of the layer field and enter a Description for each field to be displayed as a tooltip, visible moving the mouse on the column header. | So, choose what attributes you are interested in and click on .

Insert Title and Description (optional) and click on . The example below shows the Table Widget on the map.

**Counter Widgets**

Counter Widgets are numeric representations of some attributes. For example you can represent the average speed limit on a road network.

Click on , insert Title and Description then click on .

The GeoNode map viewer is MapStore based, see the MapStore Documentation for further information.

**Timeline**

GeoNode can manage datasets with a time dimension. Those vector datasets may vary their data through time so it is useful to represent that variation on the map.

The MapStore based map viewer used in Geonode makes available the Timeline tool which allows you to observe the datasets’ evolution over time, to inspect the dataset configuration at a specific time instant and to view different dataset configurations time by time dynamically through animations (see the MapStore Documentation for further details).

**Warning:** Timeline actually works only with WMTS-Multidim extension (WMS time in capabilities is not fully supported).
Configure data

X Attribute: type

Y Attribute: fid

Operation: COUNT

Color: Blue

Display Legend:

Advanced Options

Fig. 137: Chart Widgets Creation
Fig. 138: Chart Widgets Options

Fig. 139: Text Widgets Creation
Fig. 140: *My Text Widget*

![My Text Widget](image)

Configure table options

Columns

<table>
<thead>
<tr>
<th>Name</th>
<th>Title</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scalerank</td>
<td></td>
<td></td>
</tr>
<tr>
<td>featurecla</td>
<td></td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>Point type</td>
<td>type of point of interest</td>
</tr>
<tr>
<td>name</td>
<td></td>
<td></td>
</tr>
<tr>
<td>abbrev</td>
<td></td>
<td></td>
</tr>
<tr>
<td>location</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gps_code</td>
<td></td>
<td></td>
</tr>
<tr>
<td>iata_code</td>
<td></td>
<td></td>
</tr>
<tr>
<td>wikipedia</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fig. 141: *Table Widgets Columns*
Fig. 142: *Table Widget*

Configure data

42

Use:

Select attribute

Operation:

Select attribute

Unit of measure:

Fig. 143: *Counter Widget Creation*
Fig. 144: *Counter Widget*

Fig. 145: *The Timeline*
When loading a temporal dataset into the map, the *Timeline* opens automatically.

On the left side of the *Timeline* panel you can set the time value in which you want to observe the data. You can type it directly filling out the corresponding input fields or by using the up/down arrows.

![Figure 146: The Time Control Buttons](image)

On the other side there are the buttons responsible for managing the animations. In particular you can *Play* the animation by clicking , go back to the previous time instant through , go forward to next time step using and stop the animation by clicking .

![Figure 147: The Animation Control Buttons](image)

The *Timeline* panel can be expanded through the button.

![Figure 148: The Expanded Timeline](image)

The expanded section of the *Timeline* panel contains the *Time Datasets List* and an *Histogram* which shows you:

- the distribution of the data over time
- the *Time Cursor*
You can show/hide the datasets list by clicking (it is active by default).

Through the Time Range function you can observe the data in a finite temporal interval. Click on and set the initial and the final times to use it.

Animations

The Timeline allows you to see the data configurations (one for each time in which the data are defined) through ordered sequences of steps.

As said before, you can play the resulting Animation by clicking the play button. The dataset data displayed on map will change accordingly to the time reach by the cursor on the Histogram.

By clicking on you can manage some Animation Settings.
Fig. 152: *The Timeline Settings*
You can activate the *Snap to guide dataset* so that the time cursor will snap to the selected dataset’s data. You can also set up the *Frame Duration* (by default 5 seconds).

If the *Snap to guide dataset* option is disabled, you can force the animation step to be a fixed value.

The *Animation Range* option lets you to define a temporal range within which the time cursor can move.

![Fig. 153: The Timeline Animation Range](image)

See the MapStore Documentation for more information.
Other Menu Tools

At the top of the Map and on the SideBar of the map there are more menu items which we are going to explain in this section.

Add dataset

All the datasets available in GeoNode, both uploaded and remote, can be loaded on the map through the Catalog. Click on the Add dataset option of the Menu to take a look at the catalog panel.

![The Datasets Catalog](image)

Fig. 154: The Datasets Catalog

You can navigate through datasets and look at their Thumbnail images, Title, Description and Abstract. Click on a dataset to load it into the map, it will be also visible in the Table of Contents (TOC).

Saving a map

Once all the customizations have been carried out, you can Save your map by clicking on the Save option under the Resources options of the Menu.

You could create a new map from the existing view by clicking Save As... A new popup window will open. The current map title is filled by default. You can change it to the prefered naming then click on Save. The page will reload and your map should be visible in the Finding Data list.
Printing a map

The MapStore based map viewer of GeoNode allows you to print the current view with a customizable layout.

Click the button from the SideBar, the Printing Window will open.

From this window you can:

- enter Title and Description;
- choose the Resolution in dpi;
- select the format
- select the coordinate
- add the scale
- add grid with label
- customize the Layout
  - the Sheet size (A3, A4);
  - if include the legend or not;
  - if to put the legend in a separate page;
  - the page Orientation (Landscape or Portrait);
- customize the Legend
  - the Label Font;
  - the Font Size;
  - the Font Emphasis (bold, italic);
  - if Force Labels;
  - if use Anti Aliasing Font;
  - the Icon Size;
Fig. 156: The Printing Window
Performing Measurements

Click on the pencil button from the SideBar to perform a measurement. As you can see in the picture below, this tool allows you to measure Distances, Areas and the Bearing of lines.

![The Measure Tool](image)

Fig. 157: The Measure Tool

To perform a measure draw on the map the geometry you are interested in, the result will be displayed on the left of the unit of measure select menu (this tool allows you to change the unit of measure also).

Customizing The Datasets’ GetFeatureInfo Templates

When “clicking” over a feature of a dataset into a GeoNode Map, an info window popups showing a formatted representation of the raw data identified by the coordinates (see Fig. 1)

The way how such information is presented to the user is defined by what we call “GetFeatureInfo Template”. The latter is basically an HTML snippet containing some placeholders and special inline codes that instruct GeoServer on how to generate the raw data output.

The outcome is a rendered HTML snippet with the real values replacing the placeholders of the Template.

Currently, GeoNode allows a very simple mechanism to customize the “GetFeatureInfo Template” of a dataset.

It is possible, through the dataset Metadata Editor, to assign a name, a label and also set the attributes we want to display on the GetFeatureInfo output.

As an instance, by using the example above, we can customize a bit the dataset Metadata as shown in Fig. 2

The “GetFeatureInfo” output will change accordingly as shown in Fig. 3
Fig. 158: Measuring Areas
Fig. 159: Fig. 1
Fig. 160: Fig. 2

Fig. 161: Fig. 3
**Simple Template: Assigning A Media-Type To Attribute Values**

The easiest way to render a different media-type (*image*, *audio*, *video* or *iframe*) to a property value, is to change it from the *Metadata Edit* attributes panel.

By changing the *Display Type* of an attribute from this panel as shown in Fig. 4

![Metadata for ne_10m_airports_vufl8s31](image)

**Fig. 162: Fig. 4**

GeoNode will create automatically the HTML media type when rendering by using the *value* of the selected property. So, as an example, if, in the figure above, the attribute *NAME* contains values representing some links to other resources, GeoNode will create those links automatically for you when clicking over a geometry.

![Selecting image as media-type](image)

**Fig. 163: Fig. 5**

Selecting *image* as media-type (Fig. 6) and editing the contents accordingly (Fig. 7) you will get a nice effect as shown in Fig. 8
Fig. 164: Fig. 6

Fig. 165: Fig. 7

Fig. 166: Fig. 8
Advanced Template: Use A Custom HTML Template

By selecting the option *Use a custom template?* as shown in Fig. 9

You will be able to provide your own custom HTML Template for the Feature Info output. Attributes from the dataset can be represented inside the template using `${properties.ATTRIBUTE}`, where ATTRIBUTE is the name of an attribute (field) from your dataset.

The example below uses the `wikipedia` and the `name` attributes to show the name of the airport as a link to the Wikipedia page:

This is the result shown when you click on an airport.

Optional: Customizing the HTML WYSIWYG Editor Menu Bar

The *Menu Bar* and *Tool Bar* of the HTML Editor, can be easily customized by overriding the `TINYMCE_DEFAULT_CONFIG` variable on `settings.py` (see `TINYMCE_DEFAULT_CONFIG`).

There are many plugins and options allowing you to easily customize the editor and also provides some predefined templates to speed up the editing.

For more information about the Javascript tool, please refer to [https://www.tiny.cloud/](https://www.tiny.cloud/)

Search Bar

The *Search Bar* of the map viewer allows you to find point of interests (POIs), streets or locations by name.

Let's type the name of some place then select the first record.

The map will automatically re-center on that area delimiting it by a polygon in the case of an area, by a line in the case of a linear shape (e.g. streets, streams) and by a marker in the case of a point.
Fig. 168: Fig. 10

Fig. 169: Fig. 11
Fig. 170: Fig. 12

Fig. 171: The Search Bar
Fig. 172: Result of a Search
**Navigation bar**

The *Map Viewer* makes also available the *Navigation bar*. It is a navigation panel containing various tools that help you to explore the map such as tools for zooming, changing the extent and querying objects on the map.

By default the *Navigation bar* shows you the zooming buttons and . Other options can be explored by clicking on which expands/collapses the toolbar.

![Fig. 173: The Default Navigation bar](image)

The *Navigation bar* contains the following tools:

- The *Query Objects on map* allows you to get feature information through the button. It allows you to retrieve information about the features of some datasets by clicking them directly on the map.

  When clicking on map a new panel opens. That panel will show you all the information about the clicked features for each active loaded dataset.

- You can *Zoom To Max Extent* by clicking .
Fig. 174: The Expanded Navigation bar
Basemap Switcher

By default, GeoNode allows to enrich maps with many world backgrounds. You can open available backgrounds by clicking on the map tile below:

- OpenStreetMap
- OpenTopoMap
- Sentinel-2-cloudless

You can also decide to have an Empty Background.

Footer Tools

At the bottom of the map, the Footer shows you the Scale of the map and allows you to change it.

The button allows you to see the pointer Coordinates and to change the Coordinates Reference System (CRS), WGS 84 by default.
Fig. 176: The Basemap Switcher Tool
Fig. 177: Switching the Basemap
Fig. 178: The Map Scale

Fig. 179: The Pointer Coordinates and the CRS
Map Information

From the Map Search Page (see Finding Data) you can select the map you are interested in and click on its name to see an overview about it.

The information panel reports:

- The Info tab is active by default. This tab section shows some map metadata such as its title, the abstract, date of publication etc. The metadata also indicates the map owner, what are the topic categories the map belongs to and which regions are affected.

- The Location tab shows the spatial extent of the map.

By clicking on the copy icons you have a copy of the current Bounding Box or the Center in the clipboard which once pasted will be a WKT text.

- The Linked Resources tab shows resources, that can be Documents, Datasets, GeoStories and Dashboards, linked to the map.

**Note:** When a resource is linked from the map the link is shown under the Linked by list while when a resource is linked to the map the link is shown under the Used by list.

From the right toolbar on the Info panel it is possible:

- Add map in your Favorites (see Updating the Profile), clicking on star icon.
- Share the map url
- Change the map Title by clicking on it

You can access the map details page by clicking on View map in the overview panel. That page looks like the one shown in the picture below.
### Map

![Link to Map](admin/September 29th 2023)

<table>
<thead>
<tr>
<th>Info</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>Map</td>
</tr>
<tr>
<td>Owner</td>
<td>admin</td>
</tr>
<tr>
<td>Publication</td>
<td>2023-09-29 15:13</td>
</tr>
<tr>
<td>Added to catalog</td>
<td>2023-09-29 15:13</td>
</tr>
<tr>
<td>Last catalog modification</td>
<td>2023-10-17 17:48</td>
</tr>
<tr>
<td>Resource type</td>
<td>map</td>
</tr>
<tr>
<td>Source</td>
<td>local</td>
</tr>
<tr>
<td>Point of contact</td>
<td>admin</td>
</tr>
<tr>
<td>Language</td>
<td>eng</td>
</tr>
<tr>
<td>Supplemental information</td>
<td>No information provided</td>
</tr>
</tbody>
</table>

Fig. 180: Map Info tab
Map

Fig. 181: Map Location tab
Fig. 182: *Bounding Box and Center*

Fig. 183: *Map Linked Resources tab*
Fig. 184: *Map Info toolbar*

Fig. 185: *Map page*
The Map Viewer shows the map with very basic functionalities:

- the Base Map Switcher that allows you to change the base map;
- the Zoom in/out tool to enlarge and decrease the view;
- the Zoom to max extent tool for the zoom to fit the datasets extents;
- the Query Objects tool to retrieve information about the map objects by clicking on the map;
- the Print tool to print the preview.
- the Measure tool to measure Distances, Areas and the Bearing of lines.
- the Annotations tool to add an annotation to the map

Maps Metadata

Maps Metadata can be Edited by clicking the Edit Metadata link from the Map Detail page.

Fig. 186: The Map Metadata Detail link

The Map Metadata Edit form will open. Metadata provide essential information for the identification and the comprehension of the map. They also make the map more easily retrievable through the search tools. Those Metadata can be filled out through three-steps in which you have to provide all mandatory information to complete the process. Those three steps are described below.

- **Basic Metadata**

  In the first step the system asks you to insert the following metadata (required fields are highlighted with red outlines):

  - Thumbnail of the map (click Edit to change it);
  - Title of the map, which should be clear and understandable;
  - Abstract; brief narrative summary of the content of the Map

  **Note:** The Abstract panel allows you to insert HTML code through a wysiwyg text editor

  - Creation/Publication/Revision Dates which define the time period that is covered by the map;
  - Keywords, which should be chosen within the available list;
  - Category which the map belongs to;
  - Group which the map is linked to.

  Click Next >> to go to the next step.

- **Location and Licenses**

  The following list shows what kinds of metadata you are required to enter (see also the picture below):
Fig. 187: Basic Map Metadata
Fig. 188: *Location and Licenses Metadata for Maps*
– *Language* of the layer;
– *License* of the dataset;
– *Regions* covered by the layers extent. Proposed extents cover the following scales: global, continental, regional, national;
– *Data Quality statement* (general explanation of the data producer’s knowledge about the lineage of a dataset);
– Potential *Restrictions* on layer sharing.

No further mandatory metadata are required in the next step so, once the required fields have been filled out, a green *Done* button will be visible in the screen. Click *Next >>* to go to the next step or *<< Back* to go back to the previous step.

**Optional Metadata**

Complementary information are:
– *Edition* of the map;
– *Purpose* of the map and its objectives;
– *Supplemental information* that can provide a better understanding of the map;
– *Maintenance frequency* of the map;
– *Spatial representation type*, the method used to represent geographic information in the dataset;
– Users who are *Responsible* for the layer, its *Owner*, and the *Author* of its metadata;
– *Related resources* to link one or multiple resources to the document. These will be visible inside the *Map Information* panel.

If you miss some mandatory metadata the *Completeness* bar shows you a red message like the one in the picture below.

![Completeness Progress Bar](image)

*Fig. 190: Completeness Progress Bar*

**Metadata Advanced Editing**

The *Advanced Metadata* editing button in the Metadata Edit form allows to change the map metadata.

![Advanced Metadata](image)

*Fig. 191: The Advanced Edit button*

Click on it to display the *Metadata Advanced Editing Page*. That page allows you to edit all the map metadata described in the previous paragraph.

Once you have finished to edit them click on *Update* to save your changes.

**Share Options**

In GeoNode the share options management system is indeed more complex. Administrators can choose who can do what for each map. Users can manage only the maps they own or the maps which they are authorize to manage.

By default only owners can edit and manage maps, and anyone can view them.

In order to modify the *Map Share Options* settings you can click the *Share* link in the *Map Detail Page*.

Through the *Share Options Settings Panel* you can add or remove options for users and groups. The picture below shows an example.

You can set the following options:

- *View* (allows to view the map).
Fig. 192: Map Share options settings for users and groups

- Download (allows to view and download the map).
- Edit (allows to change the map’s metadata);
- Manage allows to update, delete, change share options, publish and unpublish the map.

**Warning:** When assigning options to a group, all the group members will have that option. Be careful in case of editing them.

Click on Save link in the menu to save these settings.

### 1.10.7 Dashboard

Dashboard is a MapStore tool integrated in GeoNode that provides the user with a space to add many Widgets, such as charts, maps, tables, texts and counters, and can create connections between them in order to:

- Provide an overview to better visualize a specific data context
- Interact spatially and analytically with the data by creating connections between widgets
- Perform analysis on involved data/layers

To build a new Dashboard go to Add Resource option on the All Resources page and choose option Create dashboard or select New option on the Dashboards page

Now you landed on the Dashboard edition page that is composed of the following sections:
Fig. 193: *Create dashboard from All Resources page*

Fig. 194: *Create dashboard from Dashboards page*
Further Reading

Follow the link below to get more detailed information about the usage of Dashboard.

Dashboard Documentation

1.10.8 GeoStory

GeoStory is a MapStore tool integrated in GeoNode that provides the user a way to create inspiring and immersive stories by combining text, interactive maps, and other multimedia content like images and video or other third party contents. Through this tool you can simply tell your stories on the web and then publish and share them with different groups of GeoNode users or make them public to everyone around the world.

To build a new GeoStory go to Add Resource option on the All Resources page and choose option Create geostory or select New option on the Geostories page

Now you landed on the GeoStory edition page that is composed of the following sections:

The GeoStory content is organized in Sections, that can be added with the button in the Container area. In particular, the user can add to the story the following kind of sections:

For more information on these specific section please follow the official MapStore documentation:

- Title Section
- Banner Section
- Paragraph Section
- Immersive Section
Fig. 196: *Create geostory from All Resources page*

Fig. 197: *Create geostory from Geostories page*
Fig. 198: New GeoStory Apps option

Fig. 199: GeoStory Sections Types
Add GeoNode content to GeoStory

With GeoNode you can add content to your GeoStory using internal GeoNode documents and maps as well external sources. This ability to add internal GeoNode content makes the GeoStory creation a very useful feature.

To add GeoNode content to your GeoStory use the button on top of your GeoStory section.

From here you can add *Images, Videos and Maps*. To enable GeoNode internal catalog, on Services dropdown choose *GeoNode* as shown in picture down. On the left you get a list of media documents available with a complementary text filter feature on top.

To save your GeoStory, on the top your Geostory content choose *Save* and then *Save as...*

Now your GeoStory can be shared with everyone!

Further Reading

Follow the link below to get more detailed information about the usage of GeoStory.

GeoStory Documentation
1.10.9 Publishing Data

In GeoNode, each resource can be published in order to share it with other people. In order to publish a map, document or dataset or any other GeoNode resource, go to the settings tab in the Metadata Edit form. The check for publishing and unpublishing is available. See picture below.

![Resource publishing](image)

Fig. 201: Resource publishing

Click :guilabel:`Update` to save the changes.

1.10.10 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.

**QGIS Desktop**

QGIS is a professional GIS application that is built on top of and proud to be itself Free and Open Source Software (FOSS). QGIS is a volunteer driven project if you are interested you can find more information at [https://www.qgis.org](https://www.qgis.org).
Fig. 202: *QGIS Desktop Main Window*
How can I connect to Geonode?

Open QGIS Desktop and go to Layer Menu > Data Source Manager. At the bottom of Data Source Manager, you can see a tab with the name and an icon related to Geonode. This is because Geonode is recognized as a data source inside QGIS.

![Data Source Manager Dialog](image)

Fig. 203: *Data Source Manager Dialog*

**Note:** It’s possible as well load Geonode instances from an existence file this is useful to share between users or to backup existence connections.

To add a new Geonode instance, in the Geonode tab selected click on **New** and you will see the following dialog:

In the dialog Fill the name as you like and in the URL put the link of the Geonode instance. It’s possible edit some WFS and WMS options to optimize the connection. If everything is ok you will receive the following successful connection dialog:

After the successful dialog it’s now possible to load all layers of the Geonode instance clicking on **Connect** button. You can see both WMS and WFS connections of the Geonode and you can load to QGIS Desktop.

After select a layer (WMS or WFS) click on the **Add** button and the layer will be displayed in the main window of QGIS.
Fig. 204: Details of Geonode instance Dialog
Fig. 205: Successful connection Dialog

Fig. 206: Geonode instance layers Dialog
Fig. 207: Example of Geonode layer
Warning: This procedure only work with public layers. If the layers are for private use is necessary to do the standard qgis add remote WMS/WFS layers (through Data Source Manager) along with basic auth method and specific endpoints.

Connect to Private layers by using OAuth2

GeoNode OAuth2 Client App Setup

Login to GeoNode as a superuser

Browse to http://<geonode>/o/applications/

Create a new specific app or, better, edit the existing one (“GeoServer”) based on OAuth2 Authorization Code Grant Type

Click on “Edit” and add the Redirect URI http://127.0.0.1:7070/qgis-client as shown below

Note: This is an example. The port and path of the URI can be customized. They must be the same on both GeoNode
Also you will need the Client ID and Client Secret keys later when configuring QGis.

**Configure QGis Desktop OAuth2 Authentication**

Open the QGis Desktop Client and add a new OWS remote Layer configuration

Create a new service connection

Provide the connection details

**Note:** *It is Important that the URL ends with /gs/ows*

When finished click on “+” in order to add a new auth configuration

Provide the needed information as shown below:

- Name: *any descriptive string*
Type: OAuth2 authentication

Grant Flow: Authorization Code

Request URL: must end with /o/authorize/

Token URL and Refresh URL: must end with /o/token/

Redirect URL: must match with the one defined on GeoNode above

Client ID and Client Secret: must match with the one defined on GeoNode above

Scopes: openid write

Enable the persistent Token Session via Headers

Save and click on “Connect”. QGis will redirect you on a browser page asking to GeoNode to authenticate. Approve the Claims and go back to QGis.

Remove Saved Token Sessions From QGis and Login with another User

Edit the QGis configuration

Click on the “pencil”

Clean up the saved Tokens and save

Try to connect again.
<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Title</th>
<th>Abstract</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>GeoNode Local</td>
<td>This is a description of your Web Mx</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>geonode:ACQU...</td>
<td>acqu_cipvirene_...</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>geonode:Abruz...</td>
<td>ctr_aquila</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>geonode:CP.Cs...</td>
<td>Particelle</td>
<td>Il layer contiene la particelle catastal</td>
</tr>
<tr>
<td>7</td>
<td>geonode:CP.Cs...</td>
<td>Mappe</td>
<td>Il layer contiene la rappresentazione</td>
</tr>
<tr>
<td>9</td>
<td>geonode:RC015...</td>
<td>Praterie</td>
<td>Nam quam nunc, blandit vel, luctus</td>
</tr>
<tr>
<td>11</td>
<td>geonode:acque</td>
<td>Acque</td>
<td>Il layer contiene la rappresentazione</td>
</tr>
<tr>
<td>13</td>
<td>geonode:cadice...</td>
<td>Codice_particella</td>
<td>Il layer contiene la rappresentazione</td>
</tr>
<tr>
<td>15</td>
<td>geonode:provin...</td>
<td>Province - Uffici</td>
<td>Il layer contiene la rappresentazione</td>
</tr>
<tr>
<td>17</td>
<td>geonode:shp...</td>
<td>shp_monument...</td>
<td>No abstract provided</td>
</tr>
<tr>
<td>20</td>
<td>geonode:states</td>
<td>states</td>
<td>No abstract provided</td>
</tr>
<tr>
<td>24</td>
<td>geonode:stade</td>
<td>Strade</td>
<td>Il layer contiene la rappresentazione</td>
</tr>
</tbody>
</table>
| 26 | geonode:staphyle...| Dataset Test                        | Lorem ipsum dolor sit amet, conseq
1.10.11 Dynamic Extra Metadata

In GeoNode is possible to add metadata to each resource dynamically without extending the base model provided by the application using the extra metadata field.

**Settings**

Three main settings control the extra metadata field:

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFAULT_EXTRA_METADATA_SCHEMA</td>
<td>define the schema used to store the metadata</td>
</tr>
<tr>
<td>• id: (optional int): the identifier of the metadata. Optional for creation, required in the Upgrade phase</td>
<td></td>
</tr>
<tr>
<td>• filter_header: (required object): Can be any type, is used to generate the facet filter header. Is also an identifier.</td>
<td></td>
</tr>
<tr>
<td>• field_name: (required object): name of the metadata field</td>
<td></td>
</tr>
<tr>
<td>• field_label: (required object): a verbose string of the name. Is used as a label in the facet filters.</td>
<td></td>
</tr>
<tr>
<td>• field_value: (required object): metadata values</td>
<td></td>
</tr>
</tbody>
</table>

An example of metadata that can be ingested is the following:

```
[
  {
    "filter_header": "Bike Brand",
    "field_name": "name",
    "field_label": "Bike Name",
    "field_value": "KTM",
  },
  {
    "filter_header": "Bike Brand",
    "field_name": "name",
    "field_label": "Bike Name",
    "field_value": "Bianchi",
  }
]
```

The above schema is valid by using the `schema `<https://github.com/keleshev/schema>`

CUSTOM_METADATA_SCHEMA: environment variable used to inject additional schema to the default one. Helpful for third-party libraries

EXTRA_METADATA_SCHEMA: used to get the expected metadata schema for each resource_type.

**Metadata manipulation**

There are two possible ways to manipulate extra metadata in geonode:

- via Metadata Editor (Wizard and advanced)
- via Rest API
Metadata Editor (wizard/advanced):

The metadata section is placed under the OPTIONAL METADATA section available for all the GeoNode resources. The metadata must follow two specific rules to save to the resource:

- Must always be a list of JSON. This permits to add of more than one metadata for each resource
- The JSON must follow the schema defined in the settings.py for the selected resource.

For example, for my document resource, I can have something like the following:

```json
[  
  {  
    "field_name": "bike",
    "field_label": "KTM",
    "field_value": "ktm",
    "filter_header": "Sports Parameters"
  }
]
```

Fig. 208: Advanced edit wizard menu

After pressing the save button, the system will perform the following checks:

- Check if the text provided is a valid JSON. In case of wrong format input, the following error is shown:
- Check if the metadata schema is provided for the resource if not will raise the following error
- Check if the metadata schema is coherent with the schema defined in the settings. In case of wrong format input, the error will print the missing JSON keys
Error

- 0: extra_metadata: The value provided for the Extra metadata field is not a valid JSON

---

Fig. 209: *invalid JSON error*

Error

- 0: extra_metadata: EXTRA_METADATA_SCHEMA validation schema is not available for resource dashboard

---

Fig. 210: *missing schema error*
Facet Filtering

Automatically the web interface will create dynamically the facets if there is at least 1 metadata defined for the resource.

Suppose that a resource have the following metadata:

```json
[
  {
    "field_name": "bike",
    "field_label": "KTM",
    "field_value": "ktm",
    "filter_header": "Sports Parameters"
  },
  {
    "field_name": "mountain",
    "field_label": "Monte bianco",
    "field_value": "monte_bianco",
    "filter_header": "Italian Mountains"
  }
]
```

By default GeoNode will convert this metadata info in facets available for the resource.

The facet will convert: - `filter_header`: used as the header filter - `field_value`: used to perform the search - `field_name`: used for calculate the unique values (along with `field_value`)

After says that, the facet will be the follow:
Rest API

The api/v2/resources endpoint provide different ways to handle the metadata.

GET:
Get the metadata list of the selected resource

URL: http://host/api/v2/resources/{resource_id}/extra_metadata

Response:

```
[{
  "id": 1,
  "field_name": "bike",
  "field_label": "KTM",
  "field_value": "ktm",
  "filter_header": "Sports Parameters"
}
]
```

POST:
Adding new metadata to the resource

URL: http://host/api/v2/resources/{resource_id}/extra_metadata
data = [
  {
    "field_name": "bike",
    "field_label": "KTM",
    "field_value": "ktm",
    "filter_header": "Sports Parameters"
  }
]

Response:
status_code: 201
response json: List of the available metadata for the resource

... (continues on next page)
**PUT:**

Update specific metadata for the selected resource. In this case the metadata ID is required to perform the update.

```
http://host/api/v2/resources/{resource_id}/extra_metadata
payload:
[
    {
        "id": 1,
        "field_name": "bike",
        "field_label": "KTM - sport", <- this value need to be updated
        "field_value": "ktm",
        "filter_header": "Sports Parameters"
    }
]
```

Response:
status_code: 200
response: the available payload for the selected resource
```
[
    {
        "id": 1,
        "field_name": "bike",
        "field_label": "KTM - sport",
        "field_value": "ktm",
        "filter_header": "Sports Parameters"
    }
]
```

**DELETE:**

Delete the metadata for a given resource by ID.

```
http://host/api/v2/resources/{resource_id}/extra_metadata
payload: list of ID to be deleted
[
    1, 2, 3, 4, 5
]
```

Response:
status_code: 200
response: List of the available metadata
```
[]
```
API search

It is possible to search for resources with specific metadata. This feature is available for both API v1 and API v2.

API v1:

To perform the search, it is enough to add as query parameters the field of the metadata payload.

Assuming that the payload is the same as the example above, the URL could be something like the following:

http://host/api/base/?metadata__field_category=bike

In this way, we can retrieve all the resources that have at least 1 metadata with the field_category = 'bike'

API v2:

For the API v2, it is a bit different since the library doesn't have a support for the JSON field.

To reproduce the same search above, we need to call a URL like the following one:

http://localhost:8000/api/v2/resources?filter{metadata.metadata.icontains}=%22field_category%22:%20%22bike%22

In this way, we can retrieve all the resources that have at least 1 metadata with the field_category = 'bike'

1.10.12 Download Handlers

With GeoNode 4.2.x has been introduced the concept of Download Handler and ofc GeoNode provides a default implementation of it which processes the download via WPS.

Follow an example of how to create a custom download handler and to replace the default one or add an additional one.

1.10.13 Settings

**DEFAULT_DATASET_DOWNLOAD_HANDLER**

Default: geonode.layers.download_handler.DatasetDownloadHandler

path to the download handler location

**DATASET_DOWNLOAD_HANDLERS**

Default: []

List of paths of the additional download handlers

1.10.14 CODE

The default download handler is placed under the geonode.layers [package](https://github.com/GeoNode/geonode/blob/master/geonode/layers/download_handler.py)

Follow an example of an basic class for define the download handler:

```
```
def __repr__(self):
    return self.__str__()

def __init__(self, request, resource_name) -> None:
    self.request = request
    self.resource_name = resource_name
    self._resource = None

def get_download_response(self):
    """Main method used, this method should return the response object"""
    return response

@property
def download_url(self):
    """Used by the API, it should return the URL where the resource can be downloaded from"""
    return reverse("dataset_download", args=[resource.alternate])

If you prefer to inherit from the already existing one, the response is generated in the process_download method

1.11 GeoNode Basic Installation

1.11.1 Overview

The followings are the easiest and recommended ways to deploy a full-stack GeoNode server on your host.

1. **First Step**: Deploy *GeoNode on a local server*, running as http://localhost/ service. *GeoServer* will be also available at http://localhost/geoserver/

2. **Second Step**: Deploy *GeoNode on a production server*, running as https://my_geonode.geonode.org/ service. *GeoServer* will be also available at https://my_geonode.geonode.org/geoserver/

3. **Third Step**: Customize .env to match your needs

4. **Fourth Step**: Secure your production deployment; change the admin passwords and OAuth2 keys

5. **Further Production Enhancements**

1.11.2 First Step: Deploy GeoNode on a local server

**Docker Setup for Ubuntu (22.04)**

```
# install OS level packages...
sudo add-apt-repository universe
sudo apt-get update -y
sudo apt-get install -y git-core git-buildpackage debhelper devscripts python3.10-dev 
   python3.10-venv virtualenvwrapper
sudo apt-get install -y apt-transport-https ca-certificates curl lsb-release gnupg gnupg-agent software-properties-common vim

# add docker repo and packages...
sudo mkdir -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
sudo chmod a+r /etc/apt/keyrings/docker.gpg
sudo echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] 
   https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo tee /etc/ 
   apt/sources.list.d/docker.list > /dev/null
```
Create an instance of your geonode-project

Let’s say your project is named `my_geonode` perform the following steps:

```
git clone https://github.com/GeoNode/geonode-project.git
# Checkout the desired branch. For example, 4.2.2 to generate a project for GeoNode 4.2.2
```git checkout -b 4.2.2
# Ubuntu
```
source /usr/share/virtualenvwrapper/virtualenvwrapper.sh
mkvirtualenv --python=/usr/bin/python3 my_geonode
```
Alternatively you can also create the virtual env like below
```
python3.8 -m venv /home/geonode/dev/.venvs/my_geonode
source /home/geonode/dev/.venvs/my_geonode/bin/activate
```
pip install Django==3.2.13
# CentOS
```
virtualenv -p python3 my_geonode
source my_geonode/bin/activate
```
django-admin startproject --template=./geonode-project -e py,sh,md,rst,json,yml,ini,env, --sample,properties -n monitoring-cron -n Dockerfile my_geonode
```# If the previous command does not work for some reason, try the following one
python -m django startproject --template=./geonode-project -e py,sh,md,rst,json,yml,ini, ...
```Prepare the .env file

GeoNode uses an .env where the environment variables required by Docker and GeoNode itself must be configured. A simple way to generate a basic .env is using the create-envfile.py utility command. If run without arguments it will generate a valid .env for a local deployment.

The command accepts the following arguments:

- `-https`: Enable SSL. It’s disabled by default
- `-env_type`:
GeoNode Documentation, Release master

- When set to `prod` DEBUG is disabled and the creation of a valid SSL is requested to Letsencrypt's ACME server
- When set to `test` DEBUG is disabled and a test SSL certificate is generated for local testing
- When set to `dev` DEBUG is enabled and no SSL certificate is generated

- `--hostname`: The URL that will serve GeoNode (localhost by default)
- `--email`: The administrator's email. Notice that a real email and valid SMPT configurations are required if `--env_type` is set to `prod`. Letsencrypt uses email for issuing the SSL certificate
- `--geonodepwd`: GeoNode's administrator password. A random value is set if left empty
- `--geoserverpwd`: GeoNode's administrator password. A random value is set if left empty
- `--pgpwd`: PostgreSQL's administrator password. A random value is set if left empty
- `--dbpwd`: GeoNode DB user role's password. A random value is set if left empty
- `--geodbpwd`: GeoNode data DB user role's password. A random value is set if left empty
- `--clientid`: Client id of Geoserver’s GeoNode Oauth2 client. A random value is set if left empty
- `--clientsecret`: Client secret of Geoserver’s GeoNode Oauth2 client. A random value is set if left empty

Build and run

docker compose build
docker compose up -d

- You can follow the containers startup by running the following commands from my_geonode root folder:

  # GeoNode Container
  docker-compose logs -f django

  # GeoServer Container
  docker-compose logs -f geoserver

  # DB Container
  docker-compose logs -f db

  # NGINX Container
  docker-compose logs -f geonode

- If any error occurs, try to catch the error stacktrace by running the following commands from my_geonode root folder:

  # GeoNode “entrypoint.sh” Logs
  tail -F -n 300 invoke.log
Connect to http://localhost/

The startup typically takes some time, so be patient...

If everything goes well, you should be able to see from the geonode startup logs a line similar to the following one:

```
<some date> [UWSGI] Uwsgi running...
```

Connect to http://localhost/

The default credentials are:

- GeoNode (http://localhost/) admin:
  
  username: admin password: admin

- GeoServer (http://localhost/geoserver/) admin:
  
  username: admin password: geoserver

1.11.3 Second Step: Deploy GeoNode on a production server

In the case you would like to deploy to, let’s say, https://my_geonode.geonode.org/, you will need to change .env as follows:

```python
DOCKER_ENV=production
SITEURL=https://my_geonode.geonode.org/
NGINX_BASE_URL=https://my_geonode.geonode.org/
ALLOWED_HOSTS=['django',]
GEOSERVER_WEB_UI_LOCATION=https://my_geonode.geonode.org/geoserver/
GEOSERVER_PUBLIC_LOCATION=https://my_geonode.geonode.org/geoserver/
HTTP_HOST=
HTTPS_HOST=my_geonode.geonode.org
HTTP_PORT=80
HTTPS_PORT=443
LETSENCRYPT_MODE=production # This will use Letsencrypt and the ACME server to generate valid SSL certificates
```

These variables are automatically set by the create-envfile.py script if the --https and --hostname variables are used.

**Warning:** When LETSENCRYPT_MODE is set to production a valid email and email SMPT server are required to make the system generate a valid certificate.
Restart the containers

Whenever you change something on `.env` file, you will need to rebuild the container

```
docker-compose up -d
```

**Warning:** Note that the following command drops any change you might have done manually inside the containers, except for the static volumes.

Troubleshooting

If for some reason you are not able to reach the server on the `HTTPS` channel, please check the `NGINX` configuration files below:

1. Enter the `NGINX` container

   ```
docker-compose exec geonode sh
   ```

2. Install an editor if not present

   ```
apk add nano
   ```

3. Double check that the `nginx.https.enabled.conf` link has been correctly created

   ```
ls -lah
   ```

   ![Image of ls command output]

   If the list does not match exactly the figure above, please run the following commands, and check again

   ```
rm nginx.https.enabled.conf
ln -s nginx.https.available.conf nginx.https.enabled.conf
   ```

4. Inspect the `nginx.https.enabled.conf` contents

   ```
nano nginx.https.enabled.conf
   ```
Make sure the contents match the following

**Warning:** Change the *Hostname* accordingly. This is only an example!

```bash
# NOTE: $VARIABLES are env variables replaced by entrypoint.sh using envsubst
# not to be mistaken for nginx variables (also starting with $, but usually lowercase)

# This file is to be included in the main nginx.conf configuration if HTTPS_HOST is set
ssl_session_cache shared:SSL:10m;
ssl_session_timeout 10m;

# this is the actual HTTPS host
server {
    listen 443 ssl;
    server_name my_geonode.geonode.org;
    keepalive_timeout 70;

    ssl_certificate /certificate_symlink/fullchain.pem;
    ssl_certificate_key /certificate_symlink/privkey.pem;
    ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
    ssl_ciphers HIGH:!aNULL:!MD5;

    include sites-enabled/*.conf;
}

# if we try to connect from http, we redirect to https
server {
    listen 80;
    server_name my_geonode.geonode.org; # TODO: once geoserver supports relative urls, we should allow access through both HTTP and HTTPS at the same time and hence remove HTTP_HOST from this line

    # Except for let's encrypt challenge
    location /.well-known {
        alias /geonode-certificates/.well-known;
        include /etc/nginx/mime.types;
    }

    # Redirect to https
    location / {
        return 302 https://my_geonode.geonode.org$request_uri; # TODO: we should use 301 (permanent redirect, but not practical for debug)
    }
}
```
5. Reload the NGINX configuration

```bash
nginx -s reload
```

2020/06/24 10:00:11 [notice] 112#112: signal process started
/etc/nginx# exit

6. It may be helpful to disable https to isolate the source of errors. After reverting the HTTPS-related changes in the .env file, repeat the above steps and ensure that the nginx.http.enabled.conf link has been correctly created.

```bash
ln -s nginx.conf nginx.http.enabled.conf
nano nginx.http.enabled.conf
```

### 1.11.4 Customize .env to match your needs

In the case you would like to modify the GeoNode behavior, always use the .env file in order to update the settings.

If you need to change a setting which does not exist in .env, you can force the values inside my_geonode/settings.py

Refer to the section: Settings

You can add here any property referred as

`Env: PROPERTY_NAME`

**Restart the containers**

Whenever you change something on .env file, you will need to rebuild the containers.

**Warning: Be careful!** The following command drops any change you might have done manually inside the containers, except for the static volumes.

```bash
docker-compose up -d django
```

[Optional] **Configure your SSL Certificates**

In production deployment mode, GeoNode uses by default Let's Encrypt certificates

You may want to provide your own certificates to GeoNode

```bash
docker exec -it nginx4my_geonode_geonode sh -c 'mkdir /geonode-certificates/my_geonode'
wget --no-check-certificate 'http://<url_to_your_chain.crt>' \ 
  -O chain.crt
wget --no-check-certificate 'http://<url_to_your_key.key>' \ 
  -O my_geonode.key
```

(continues on next page)
docker cp chain.crt nginx4my_geonode_geonode:/geonode-certificates/my_geonode

docker cp my_geonode.key nginx4my_geonode_geonode:/geonode-certificates/my_geonode

docker-compose exec geonode sh
apk add vim

vim nginx.https.enabled.conf

-ssl_certificate /certificate_symlink/fullchain.pem;
-ssl_certificate_key /certificate_symlink/privkey.pem;
+ssl_certificate /geonode-certificates/my_geonode/chain.crt;
+ssl_certificate_key /geonode-certificates/my_geonode/my_geonode.key;

nginx -s reload
exit

1.12 GeoNode Advanced Installation

1.12.1 GeoNode Vanilla

Overview

The following steps will guide you to a fresh setup of GeoNode.

All guides will first install and configure the system to run it in DEBUG mode (also known as DEVELOPMENT mode) and then by configuring an HTTPD server to serve GeoNode through the standard HTTP (80) port.

**Warning:** Those guides are not meant to be used on a production system. There will be dedicated chapters that will show you some hints to optimize GeoNode for a production-ready machine. In any case, we strongly suggest to task an experienced DevOp or System Administrator before exposing your server to the WEB.

Ubuntu 22.04 LTS

This part of the documentation describes the complete setup process for GeoNode on an Ubuntu 22.04.1LTS 64-bit clean environment (Desktop or Server).

All examples use shell commands that you must enter on a local terminal or a remote shell.

- If you have a graphical desktop environment you can open the terminal application after login;
- if you are working on a remote server the provider or sysadmin should has given you access through an ssh client.
1. Install the dependencies

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

**Warning:** To follow this guide, a basic knowledge about Ubuntu Server configuration and working with a shell is required.

**Note:** This guide uses `vim` as the editor; feel free to use `nano`, `gedit` or others.

### Upgrade system packages

Check that your system is already up-to-date with the repository running the following commands:

```
sudo apt update -y
```

### Packages Installation

**Note:** You don’t need to install the **system packages** if you want to run the project using Docker

We will use `example.org` as fictitious Domain Name.

First, we are going to install all the **system packages** needed for the GeoNode setup. Login to the target machine and execute the following commands:

```
# Install packages from GeoNode core
sudo apt install -y --allow-downgrades build-essential
  python3-gdal=3.4.1+dfsg-1build4
gdal-bin=3.4.1+dfsg-1build4
libgdal-dev=3.4.1+dfsg-1build4

  python3-all-dev python3.10-dev python3.10-venv virtualenvwrapper
  libxml2 libxml2-dev gettext
  libmemcached-dev zlib1g-dev
  libxslt1-dev libjpeg-dev libpng-dev libpq-dev
  software-properties-common build-essential
  git unzip gcc zlib1g-dev libgeos-dev libproj-dev
  sqlite3 spatialite-bin libsqlite3-mod-spatialite libsqlite3-dev

# Install Openjdk
sudo apt install openjdk-11-jdk-headless default-jdk-headless -y

# Verify GDAL version
gdalinfo --version
  $> GDAL 3.4.1, released 2021/12/27

# Verify Python version
python3.10 --version
  $> Python 3.10.4
```

(continues on next page)
which python3.10
  $> /usr/bin/python3.10

# Verify Java version
java -version
  $> openjdk version "11.0.16"

# Install VIM
sudo apt install -y vim

# Cleanup the packages
sudo apt update -y; sudo apt autoremove --purge

**Warning:** GeoNode 4.2.x is not compatible with Python < 3.7

## 2. GeoNode Installation

This is the most basic installation of GeoNode. It won’t use any external server like Apache Tomcat, PostgreSQL or HTTPD.

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

Since geonode needs a large number of different python libraries and packages, its recommended to use a python virtual environment to avoid conflicts on dependencies with system wide python packages and other installed software. See also documentation of Virtualenvwrapper package for more information

**Note:** The GeoNode Virtual Environment must be created only the first time. You won’t need to create it again everytime.

which python3.10  # copy the path of python executable

# Create the GeoNode Virtual Environment (first time only)
export WORKON_HOME=~/.virtualenvs
source /usr/share/virtualenvwrapper/virtualenvwrapper.sh
mkvirtualenv --python=/usr/bin/python3.10 geonode  # Use the python path from above

# Alterantively you can also create the virtual env like below
mkdir -p ~/.virtualenvs
python3.10 -m venv ~/.virtualenvs/geonode
source ~/.virtualenvs/geonode/bin/activate

At this point your command prompt shows a (geonode) prefix, this indicates that your virtualenv is active.

**Note:** The next time you need to access the Virtual Environment just run

source /usr/share/virtualenvwrapper/virtualenvwrapper.sh
workon geonode

(continues on next page)
# Alternatively you can also create the virtual env like below
source ~/.virtualenvs/geonode/bin/activate

Note: In order to save permanently the virtualenvwrapper environment

vim ~/.bashrc

# Write to the bottom of the file the following lines
export WORKON_HOME=~/.virtualenvs
source /usr/share/virtualenvwrapper/virtualenvwrapper.sh

# Let's create the GeoNode core base folder and clone it

# Clone the GeoNode source code on /opt/geonode
cd /opt; git clone https://github.com/GeoNode/geonode.git -b 4.2.x geonode

# Install the Python packages
cd /opt/geonode
pip install -r requirements.txt --upgrade
pip install -e . --upgrade
pip install pygdal=="gdal-config --version`.*"

Edit /opt/geonode/geonode/settings.py.

```
FILE_UPLOAD_DIRECTORY_PERMISSIONS = 0o777
FILE_UPLOAD_PERMISSIONS = 0o777
```

Edit /opt/geonode/ucgi.ini.

```
chdir = /opt/geonode/
touch-reload = /opt/geonode/geonode/wsgi.py
```

3. Postgis database Setup

Be sure you have successfully completed all the steps of the section 1. Install the dependencies.

In this section, we are going to setup users and databases for GeoNode in PostgreSQL.
Install and Configure the PostgreSQL Database System

In this section we are going to install the PostgreSQL packages along with the PostGIS extension. Those steps must be done only if you don’t have the DB already installed on your system.

```bash
# Ubuntu 22.04.1 (focal)
sudo sh -c "echo "deb http://apt.postgresql.org/pub/repos/apt/ `lsb_release -cs`-pgdg-\nmain" >> /etc/apt/sources.list.d/pgdg.list"
sudo wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc | sudo apt-key \nadd -
sudo apt update -y; sudo apt install -y postgresql-13 postgresql-13-postgis-3 postgresql-\n13-postgis-3-scripts postgresql-13 postgresql-client-13
```

We now must create two databases, geonode and geonode_data, belonging to the role geonode.

**Warning:** This is our default configuration. You can use any database or role you need. The connection parameters must be correctly configured on settings, as we will see later in this section.

### Databases and Permissions

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

```bash
sudo service postgresql start
sudo -u postgres createuser -P geonode
# Use the password: geonode
```

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

**Warning:** This is a sample password used for the sake of simplicity. This password is very **weak** and should be changed in a production environment.

Create database geonode and geonode_data with owner geonode

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

Next let’s create PostGIS extensions

```bash
sudo -u postgres psql -d geonode -c 'CREATE EXTENSION postgis;'
sudo -u postgres psql -d geonode -c 'GRANT ALL ON geometry_columns TO PUBLIC;'
sudo -u postgres psql -d geonode -c 'GRANT ALL ON spatial_ref_sys TO PUBLIC;'
sudo -u postgres psql -d geonode -c 'GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO geonode;'
sudo -u postgres psql -d geonode -c 'GRANT ALL PRIVILEGES ON ALL SEQUENCES IN SCHEMA public TO geonode;'
sudo -u postgres psql -d geonode_data -c 'CREATE EXTENSION postgis;'
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;'
```

(continues on next page)
Final step is to change user access policies for local connections in the file `pg_hba.conf`

```
sudo vim /etc/postgresql/13/main/pg_hba.conf
```

Scroll down to the bottom of the document. We want to make local connection trusted for the default user.

Make sure your configuration looks like the one below.

```
...  
# DO NOT DISABLE!
# If you change this first entry you will need to make sure that the
# database superuser can access the database using some other method.
# Noninteractive access to all databases is required during automatic
# maintenance (custom daily cronjobs, replication, and similar tasks).
#
# Database administrative login by Unix domain socket
local all postgres trust

# TYPE DATABASE USER ADDRESS METHOD
#
# "local" is for Unix domain socket connections only
local all all trust

# IPv4 local connections:
host all all 127.0.0.1/32 md5

# IPv6 local connections:
host all all ::1/128 md5

# Allow replication connections from localhost, by a user with the
# replication privilege.
local replication all peer
host replication all 127.0.0.1/32 md5
host replication all ::1/128 md5
```

**Warning:** If your PostgreSQL database resides on a separate/remote machine, you’ll have to allow remote access to the databases in the `/etc/postgresql/13/main/pg_hba.conf` to the geonode user and tell PostgreSQL to accept non-local connections in your `/etc/postgresql/13/main/postgresql.conf` file.

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 role.

```
psql -U postgres geonode
# This should not ask for any password
psql -U geonode geonode
```

(continues on next page)
# This should ask for the password geonode

# Repeat the test with geonode_data DB
psql -U postgres geonode_data
psql -U geonode geonode_data

4. Install GeoServer

In this section, we are going to install the Apache Tomcat 8 Servlet Java container, which will be started by default on the internal port 8080.

We will also perform several optimizations to:

1. Correctly setup the Java VM Options, like the available heap memory and the garbage collector options.
2. Externalize the GeoServer and GeoWebcache catalogs in order to allow further updates without the risk of deleting our datasets.

**Note:** This is still a basic setup of those components. More details will be provided on sections of the documentation concerning the hardening of the system in a production environment. Nevertheless, you will need to tweak a bit those settings accordingly with your current system. As an instance, if your machine does not have enough memory, you will need to lower down the initial amount of available heap memory. **Warnings** and **notes** will be placed below the statements that will require your attention.

## Install Apache Tomcat

The reference version of Tomcat for the Geoserver for GeoNode is **Tomcat 9**.

The following steps have been adapted from [https://yallalabs.com/linux/ubuntu/how-to-install-apache-tomcat-9-ubuntu-20-04/](https://yallalabs.com/linux/ubuntu/how-to-install-apache-tomcat-9-ubuntu-20-04/)

**Warning:** Apache Tomcat 9 and GeoServer require Java 11 or newer to be installed on the server. Check the steps before in order to be sure you have OpenJDK 11 correctly installed on your system.

First, it is not recommended to run Apache Tomcat as user root, so we will create a new system user which will run the Apache Tomcat server

```
sudo useradd -m -U -d /opt/tomcat -s /bin/bash tomcat
sudo usermod -a -G www-data tomcat
```

**Warning:** Now, go to the official Apache Tomcat website and download the most recent version of the software to your server. But don't use Tomcat10 because there are still some errors between GeoServer and Tomcat.

```
VERSION=9.0.65; wget https://archive.apache.org/dist/tomcat/tomcat-9/v$\{VERSION\}/bin/\n   --apache-tomcat-$\{VERSION\}.tar.gz
```

Once the download is complete, extract the tar file to the /opt/tomcat directory:
Apache Tomcat is updated regularly. So, to have more control over versions and updates, we’ll create a symbolic link as below:

```
sudo ln -s /opt/tomcat/apache-tomcat-${VERSION} /opt/tomcat/latest
```

Now, let’s change the ownership of all Apache Tomcat files as below:

```
sudo chown -R tomcat:www-data /opt/tomcat/
```

Make the shell scripts inside the bin directory executable:

```
sudo sh -c 'chmod +x /opt/tomcat/latest/bin/*.
```

Create the a systemd file with the following content:

```
[Unit]
Description=Tomcat 9 servlet container
After=network.target

[Service]
Type=forking
User=tomcat
Group=tomcat

Environment="JAVA_HOME=/usr/lib/jvm/jre"
Environment="JAVA_OPTS=-Djava.security.egd=file:///dev/urandom -Djava.awt.headless=true"
Environment="CATALINA_BASE=/opt/tomcat/latest"
Environment="CATALINA_HOME=/opt/tomcat/latest"
Environment="CATALINA_PID=/opt/tomcat/latest/temp/tomcat.pid"
Environment="CATALINA_OPTS=-Xms512M -Xmx1024M -server -XX:+UseParallelGC"

ExecStart=/opt/tomcat/latest/bin/startup.sh
ExecStop=/opt/tomcat/latest/bin/shutdown.sh

[Install]
WantedBy=multi-user.target
```
Now you can start the Apache Tomcat 9 server and enable it to start on boot time using the following command:

```bash
sudo systemctl daemon-reload
sudo systemctl start tomcat9.service
sudo systemctl status tomcat9.service
sudo systemctl enable tomcat9.service
```

For verification, type the following ss command, which will show you the 8080 open port number, the default open port reserved for Apache Tomcat Server.

```
ss -ltn
```

In a clean Ubuntu 22.04.1, the ss command may not be found and the iproute2 library should be installed first.

```bash
sudo apt install iproute2
# Then run the ss command
ss -ltn
```

In a clean Ubuntu 22.04.1, the ss command may not be found and the iproute2 library should be installed first.

If your server is protected by a firewall and you want to access Tomcat from the outside of your local network, you need to open port 8080.

Use the following command to open the necessary port:

```bash
sudo ufw allow 8080/tcp
```

**Warning:** Generally, when running Tomcat in a production environment, you should use a load balancer or reverse proxy.

It’s a best practice to allow access to port 8080 only from your internal network.

We will use NGINX in order to provide Apache Tomcat through the standard HTTP port.

**Note:** Alternatively you can define the Tomcat Service as follow, in case you would like to use systemctl

```bash
sudo vim /usr/lib/systemd/system/tomcat9.service
```

**[Unit]**
Description=Apache Tomcat Server
After=syslog.target network.target

**[Service]**
Type=forking
User=tomcat
Group=tomcat
Environment=JAVA_HOME=/usr/lib/jvm/jre
Environment=JAVA_OPTS=-Djava.security.egd=file:///dev/urandom

(continues on next page)
Environment=CATALINA_PID=/opt/tomcat/latest/temp/tomcat.pid
Environment=CATALINA_HOME=/opt/tomcat/latest
Environment=CATALINA_BASE=/opt/tomcat/latest

ExecStart=/opt/tomcat/latest/bin/startup.sh
ExecStop=/opt/tomcat/latest/bin/shutdown.sh

RestartSec=30
Restart=always

[Install]
WantedBy=multi-user.target

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sudo systemctl</td>
<td>Reload the system control daemon.</td>
</tr>
<tr>
<td>sudo systemctl</td>
<td>Enable the tomcat9.service.</td>
</tr>
<tr>
<td>sudo systemctl</td>
<td>Start the tomcat9.service.</td>
</tr>
</tbody>
</table>

### Install GeoServer on Tomcat

Let’s externalize the GEOSERVER_DATA_DIR and logs

```bash
# Create the target folders
sudo mkdir -p /opt/data
sudo chown -Rf $USER:www-data /opt/data
sudo chmod -Rf 775 /opt/data
sudo mkdir -p /opt/data/logs
sudo chown -Rf $USER:www-data /opt/data/logs
sudo chmod -Rf 775 /opt/data/logs

# Download and extract the default GEOSERVER_DATA_DIR
GS_VERSION=2.24.2
sudo unzip data-$GS_VERSION.zip -d /opt/data/

sudo mv /opt/data/data/ /opt/data/geoserver_data
sudo chown -Rf tomatc:www-data /opt/data/geoserver_data
sudo chmod -R 775 /opt/data/geoserver_data

sudo mkdir -p /opt/data/geoserver_logs
sudo chown -Rf tomatc:www-data /opt/data/geoserver_logs
sudo chmod -Rf 775 /opt/data/geoserver_logs

sudo mkdir -p /opt/data/gwc_cache_dir
sudo chown -Rf tomatc:www-data /opt/data/gwc_cache_dir
sudo chmod -Rf 775 /opt/data/gwc_cache_dir

# Download and install GeoServer

(continues on next page)
Let’s now configure the JAVA_OPTS, i.e. the parameters to run the Servlet Container, like heap memory, garbage collector and so on.

```bash
sudo sed -i -e 's/xom-\*\.jar/xom-\*\.jar,bcprov\*\.jar/g' /opt/tomcat/latest/conf/
catalina.properties

export JAVA_HOME=$(readlink -f /usr/bin/java | sed "s:bin/java::")
echo 'JAVA_HOME="$JAVA_HOME" | sudo tee --append /opt/tomcat/latest/bin/setenv.sh
sudo sed -i -e "s/\$JAVA_OPTS=/#JAVA_OPTS=/g" /opt/tomcat/latest/bin/setenv.sh

echo 'GEOSERVER_DATA_DIR="/opt/data/geoserver_data"' | sudo tee --append /opt/tomcat/latest/bin/setenv.sh
echo 'GEOSERVER_LOG_LOCATION="/opt/data/geoserver_logs/geoserver.log"' | sudo tee --append /opt/tomcat/latest/bin/setenv.sh
echo 'GEOWEBCACHE_CACHE_DIR="/opt/data/gwc_cache_dir"' | sudo tee --append /opt/tomcat/latest/bin/setenv.sh
echo 'GEOFENCE_DIR="$GEOSERVER_DATA_DIR/geofence"' | sudo tee --append /opt/tomcat/latest/bin/setenv.sh
echo 'TIMEZONE="UTC"' | sudo tee --append /opt/tomcat/latest/bin/setenv.sh

Note: After the execution of the above statements, you should be able to see the new options written at the bottom of the file /opt/tomcat/latest/bin/setenv.sh.

...
Those options could be updated or changed manually at any time, accordingly to your needs.

**Warning:** The default options we are going to add to the Servlet Container, assume you can reserve at least 4GB of RAM to GeoServer (see the option `-Xmx4096m`). You must be sure your machine has enough memory to run both GeoServer and GeoNode, which in this case means at least 4GB for GeoServer plus at least 2GB for GeoNode. A total of at least 6GB of RAM available on your machine. If you don’t have enough RAM available, you can lower down the values `-Xms512m -Xmx4096m`. Consider that with less RAM available, the performances of your services will be highly impacted.

```bash
# Create the Logrotate config
sudo tee /etc/logrotate.d/geoserver <<EOF
/opt/data/geoserver_logs/geoserver.log
/opt/tomcat/apache-tomcat-*-logs/*.log
/opt/tomcat/apache-tomcat-*-logs/*.out
/opt/tomcat/apache-tomcat-*-logs/*.txt
{
copytruncate
daily
rotate 5
delaycompress
missingok
su tomcat tomcat
}
EOF
```

**Configuring the Geofence DB**

Before starting the service, Geofence must be configured to connect to the PostgreSQL DB, where its rules will be stored.

**Warning:** In previous versions this step was optional and a file-based H2 DB could be used. This option has been dropped since using H2 is highly discouraged.

Open the `geofence-datasource-ovr.properties` file for edit:

```
sudo vim /opt/data/geoserver_data/geofence/geofence-datasource-ovr.properties
```

And paste the following code by replace the placeholders with the required files

```
geofenceVendorAdapter.databasePlatform=org.hibernate.spatial.postgis.PostgisDialect
geofenceDataSource.driverClassName=org.postgresql.Driver
geofenceDataSource.url=jdbc:postgresql://localhost:5432/geonode_data
geofenceDataSource.username=geonode
```

(continues on next page)
geoNode Documentation, Release master

(continued from previous page)

In order to make the changes effective, you’ll need to restart the Servlet Container.

```bash
# Restart the server
sudo systemctl restart tomcat9.service

# Follow the startup logs
sudo tail -F -n 300 /opt/data/geoserver_logs/geoserver.log
```

If you can see on the logs something similar to this, without errors

```plaintext
... 2019-05-31 10:06:34,190 INFO [geoserver.wps] - Found 5 bindable processes in GeoServer in specific processes
  2019-05-31 10:06:34,311 INFO [geoserver.wps] - Blacklisting process ras:ConvolveCoverage as the input kernel of type class javax.media.jai.KernelJAI cannot be handled
  2019-05-31 10:06:34,320 INFO [geoserver.wps] - Blacklisting process ras:RasterZonalStatistics2 as the input nodata of type class it.geosolutions.jaiext.Range cannot be handled
  2019-05-31 10:06:34,918 INFO [ows.OWSHandlerMapping] - Mapped URL path [/wfs/*] onto handler 'dispatcher'
  2019-05-31 10:06:42,237 INFO [geoserver.security] - Start reloading user/groups for service named default
```

(continues on next page)
Your GeoServer should be up and running at

http://localhost:8080/geoserver/

**Warning:** In case of errors or the file geoserver.log is not created, check the Catalina logs in order to try to understand what’s happened.

```bash
sudo less /opt/tomcat/latest/logs/catalina.out
```

### 5. Web Server

Until now we have seen how to start GeoNode in DEBUG mode from the command line, through the paver utilities. This is of course not the best way to start it. Moreover you will need a dedicated HTTPD server running on port 80 if you would like to expose your server to the world.

In this section we will see:

1. How to configure NGINX HTTPD Server to host GeoNode and GeoServer. In the initial setup we will still run the services on http://localhost
2. Update the settings in order to link GeoNode and GeoServer to the PostgreSQL Database.
3. Update the settings in order to update GeoNode and GeoServer services running on a public IP or hostname.
4. Install and enable HTTPS secured connection through the Let’s Encrypt provider.

**Install and configure NGINX**

```
Warning: Seems to be possible that NGINX works with Python 3.6 and not with 3.8.
```

```bash
# Install the services
sudo apt install -y nginx uwsgi uwsgi-plugin-python3
```
Serving {“geonode”, “geoserver”} via NGINX

# Create the UWSGI config
sudo vim /opt/geonode/uwsgi.ini

[uwsgi]
# uwsgi-socket = 0.0.0.0:8000
http-socket = 0.0.0.0:8000
logto = /var/log/geonode.log
# pidfile = /tmp/geonode.pid
chdir = /opt/geonode/
module = geonode.wsgi:application
strict = false
master = true
enable-threads = true
vacuum = true  ; Delete sockets during shutdown
single-interpreter = true
die-on-term = true  ; Shutdown when receiving SIGTERM (default is respawn)
need-app = true
thunder-lock = true
touch-reload = /opt/geonode/geonode/wsgi.py
buffer-size = 32768
harakiri = 600  ; forcefully kill workers after 600 seconds
py-callos-afterfork = true
; allow workers to trap signals
max-requests = 1000  ; Restart workers after this many requests
max-worker-lifetime = 3600  ; Restart workers after this many seconds
reload-on-rss = 2048  ; Restart workers after this much resident memory
worker-reload-mercy = 60  ; How long to wait before forcefully killing workers
cheaper-algo = busyness
processes = 128  ; Maximum number of workers allowed
cheaper = 8  ; Minimum number of workers allowed
cheaper-initial = 16  ; Workers created at startup
cheaper-overload = 1
cheaper-step = 16  ; How many workers to spawn at a time
cheaper-busyness-multiplier = 30  ; How many cycles to wait before killing workers
cheaper-busyness-min = 20  ; Below this threshold, kill workers (if stable for multiplier cycles)
cheaper-busyness-max = 70  ; Above this threshold, spawn new workers
cheaper-busyness-backlog-alert = 16
cheaper-busyness-backlog-step = 2
# daemonize = /var/log/uwsgi/geonode.log

(continues on next page)
# cron = -1 -1 -1 -1 -1 /usr/local/bin/python /usr/src/geonode/manage.py collect_metrics

# Create the UWSGI system service
sudo vim /etc/systemd/system/geonode-uwsgi.service

Warning: !IMPORTANT!
Change the line ExecStart=... below with your current user home directory!
e.g.: If the user is geosolutions then ExecStart=/home/geosolutions/.virtualenvs/geonode/bin/uwsgi --ini /opt/geonode/uwsgi.ini

[Unit]
Description=GeoNode UWSGI Service
After=rc-local.service

[Service]
EnvironmentFile=/opt/geonode/.env
User=geosolutions
Group=geosolutions
PIDFile=/run/geonode-uwsgi.pid
ExecStart=/home/geosolutions/.virtualenvs/geonode/bin/uwsgi --ini /opt/geonode/uwsgi.ini
PrivateTmp=true
Type=simple
Restart=always
KillMode=process
TimeoutSec=900

[Install]
WantedBy=multi-user.target

# Enable the UWSGI service
sudo systemctl daemon-reload
sudo systemctl start geonode-uwsgi.service
sudo systemctl status geonode-uwsgi.service
sudo systemctl enable geonode-uwsgi.service

# Create the Logrotate config
sudo tee /etc/logrotate.d/uwsgi-geonode <<EOF
"/var/log/geonode.log" {
    copytruncate
daily
rotate 5
delaycompress
missingok
notifempty
    su root root
}
EOF

1.12. GeoNode Advanced Installation
# Backup the original NGINX config
```bash
sudo mv /etc/nginx/nginx.conf /etc/nginx/nginx.conf.orig
```

# Create the GeoNode Default NGINX config
```bash
sudo vim /etc/nginx/nginx.conf
```

# Make sure your nginx.config matches the following one
```bash
user www-data;
worker_processes auto;
pid /run/nginx.pid;
include /etc/nginx/modules-enabled/*.conf;

events {
  worker_connections 768;
  # multi_accept on;
}

http {
  ##
  # Basic Settings
  ##

  sendfile on;
tcp_nopush on;
tcp_nodelay on;
keepalive_timeout 65;
types_hash_max_size 2048;
  # server_tokens off;

  # server_names_hash_bucket_size 64;
  # server_name_in_redirect off;

  include /etc/nginx/mime.types;
default_type application/octet-stream;

  ##
  # SSL Settings
  ##

  ssl_protocols TLSv1 TLSv1.1 TLSv1.2; # Dropping SSLv3, ref: POODLE
  ssl_prefer_server_ciphers on;

  ##
  # Logging Settings
  ##

  access_log /var/log/nginx/access.log;
  error_log /var/log/nginx/error.log;

  ##
  # Gzip Settings
  ##
```

(continues on next page)
gzip on;
gzip_vary on;
gzip_proxied any;
gzip_http_version 1.1;
gzip_disable "MSIE [1-6]\.";
gzip_buffers 16 8k;
gzip_min_length 1100;
gzip_comp_level 6;
gzip_types video/mp4 text/plain application/javascript application/x-javascript text/plain text/xml text/css image/jpeg;

##
# Virtual Host Configs
##
include /etc/nginx/conf.d/*.conf;
include /etc/nginx/sites-enabled/*;

# Remove the Default NGINX config
sudo rm /etc/nginx/sites-enabled/default

# Create the GeoNode App NGINX config
sudo vim /etc/nginx/sites-available/geonode

uwsgi_intercept_errors on;

upstream geoserver_proxy {
    server localhost:8080;
}

# Expires map
map $sent_http_content_type $expires {
    default off;
text/html epoch;
text/css max;
application/javascript max;
~image/ max;
}

server {

    listen 80 default_server;
    listen [::]:80 default_server;

    root /var/www/html;
    index index.html index.htm index.nginx-debian.html;

    server_name _;

    charset utf-8;

}
etag on;
expires $expires;
proxy_read_timeout 600s;
# set client body size to 2M #
client_max_body_size 50000M;

location / {
  etag off;
  proxy_pass http://127.0.0.1:8000;
  include proxy_params;
}

location /static/ {
  alias /opt/geonode/geonode/static_root/;
}

location /uploaded/ {
  alias /opt/geonode/geonode/uploaded/;
}

location /geoserver {
  proxy_pass http://geoserver_proxy;
  include proxy_params;
}

# Prepare the uploaded folder
sudo mkdir -p /opt/geonode/geonode/uploaded
sudo chown -Rf tomcat:www-data /opt/geonode/geonode/uploaded
sudo chmod -Rf 777 /opt/geonode/geonode/uploaded/

sudo touch /opt/geonode/geonode/.celery_results
sudo chmod 777 /opt/geonode/geonode/.celery_results

# Enable GeoNode NGINX config
sudo ln -s /etc/nginx/sites-available/geonode /etc/nginx/sites-enabled/geonode

# Restart the services
sudo service tomcat9 restart
sudo service nginx restart
Update the settings in order to use the PostgreSQL Database

**Warning:** Make sure you already installed and configured the Database as explained in the previous sections.

**Note:** Instead of using the `local_settings.py`, you can drive the GeoNode behavior through the `.env*` variables; see as an instance the file `./paver_dev.sh` or `./manage_dev.sh` in order to understand how to use them. In that case you don’t need to create the `local_settings.py` file; you can just stick with the default one, which will take the values from the ENV. We tend to prefer this method in a production/dockerized system.

```
workon geonode
 cd /opt/geonode

 # Initialize GeoNode
 chmod +x *.sh
 ./paver_local.sh reset
 ./paver_local.sh setup
 ./paver_local.sh sync
 ./manage_local.sh collectstatic --noinput
 sudo chmod -Rf 777 geonode/static_root/ geonode/uploaded/
```

Before finalizing the configuration we will need to update the UWSGI settings

Restart UWSGI and update OAuth2 by using the new `geonode.settings`

```
# As superuser
sudo su

# Restart Tomcat
service tomcat9 restart

# Restart UWSGI
pkill -9 -f uwsgi

# Update the GeoNode ip or hostname
 cd /opt/geonode

# This must be done the first time only
 cp package/support/geonode.binary /usr/bin/geonode
 cp package/support/geonode.updateip /usr/bin/geonode_updateip
 chmod +x /usr/bin/geonode
 chmod +x /usr/bin/geonode_updateip

# Refresh GeoNode and GeoServer OAuth2 settings
 source .env_local
 PYTHONWARNINGS=ignore VIRTUAL_ENV=$VIRTUAL_ENV DJANGO_SETTINGS_MODULE=geonode.settings_
 ...GEONODE_ETC=/opt/geonode/geonode GEOSERVER_DATA_DIR=/opt/data/geoserver_data TOMCAT_
 ...SERVICE="service tomcat9" APACHE_SERVICE="service nginx" geonode_updateip -p localhost

# Go back to standard user
 exit
```

1.12. GeoNode Advanced Installation 210
Check for any error with

```
sudo tail -F -n 300 /var/log/geonode.log
```

Reload the UWSGI configuration with

```
touch /opt/geonode/geonode/wsgi.py
```

6. Update the settings in order to update GeoNode and GeoServer services running on a public IP or hostname

**Warning:** Before exposing your services to the Internet, **make sure** your system is **hardened** and **secure enough**. See the specific documentation section for more details.

Let’s say you want to run your services on a public IP or domain, e.g. `www.example.org`. You will need to slightly update your services in order to reflect the new server name.

In particular the steps to do are:

1. Update **NGINX** configuration in order to serve the new domain name.

```
sudo vim /etc/nginx/sites-enabled/geonode

# Update the 'server_name' directive
server_name example.org www.example.org;

# Restart the service
sudo service nginx restart
```

2. Update `.env` with the new domain name.

```
vim /opt/geonode/.env

# Change everywhere 'localhost' to the new hostname
:%s/localhost/www.example.org/g
:wq

# Restart the service
sudo systemctl restart geonode-uwsgi
```

3. Update **OAuth2** configuration in order to hit the new hostname.

```
workon geonode
sudo su
cd /opt/geonode

# Update the GeoNode ip or hostname
PYTHONWARNINGS=ignore VIRTUAL_ENV=$VIRTUAL_ENV DJANGO_SETTINGS_MODULE=geonode.
→ settings GEONODE_ETC=/opt/geonode/geonode GEOSERVER_DATA_DIR=/opt/data/
→ geoserver_data TOMCAT_SERVICE="service tomcat9" APACHE_SERVICE="service nginx
→ geonode_updateip -l localhost -p www.example.org
```

(continues on next page)
exit

4. Update the existing GeoNode links in order to hit the new hostname.

```bash
workon geonode

# To avoid spatialite conflict if using postgresql
vim $VIRTUAL_ENV/bin/postactivate

# Add these to make available. Change user, password and server information to yours
export DATABASE_URL='postgresql://<postgresqluser>:<postgresqlpass>@localhost:5432/geonode'

# Close virtual environment and aopen it again to update variables
deactivate

workon geonode
cd /opt/geonode

# Update the GeoNode ip or hostname
DJANGO_SETTINGS_MODULE=geonode.settings python manage.py migrate_baseurl --source-address=http://localhost --target-address=http://www.example.org
```

**Note:** If at the end you get a “bad gateway” error when accessing your geonode site, check uwsgi log with `sudo tail -f /var/log/geonode.log` and if there is an error related with port 5432 check the listening configuration from the postgresql server and allow the incoming traffic from geonode.

---

7. Install and enable HTTPS secured connection through the Let’s Encrypt provider

```bash
# Install Let's Encrypt Certbot
# sudo add-apt-repository ppa:certbot/certbot # for ubuntu 18.04 and lower
sudo apt update -y; sudo apt install python3-certbot-nginx -y

# Reload NGINX config and make sure the firewall denies access to HTTP
sudo systemctl reload nginx
sudo ufw allow 'Nginx Full'
sudo ufw delete allow 'Nginx HTTP'

# Create and dump the Let's Encrypt Certificates
sudo certbot --nginx -d example.org -d www.example.org
# ...choose the redirect option when asked for
```

Next, the steps to do are:

1. Update the GeoNode OAuth2 Redirect URIs accordingly.

   From the GeoNode Admin Dashboard go to Home › Django/GeoNode OAuth Toolkit › Applications › GeoServer
2. Update the GeoServer Proxy Base URL accordingly.
From the GeoServer Admin GUI go to About & Status > Global
3. Update the GeoServer Role Base URL accordingly.
From the GeoServer Admin GUI go to Security > Users, Groups, Roles > geonode REST role service
4. Update the GeoServer OAuth2 Service Parameters accordingly.
From the GeoServer Admin GUI go to Security > Authentication > Authentication Filters > geonode-oauth2
5. Update the .env file

```
vim /opt/geonode/.env

# Change everywhere 'http' to 'https'
%s/http/https/g

# Restart the service
sudo systemctl restart geonode-uwsgi
```
Fig. 213: Proxy Base URL

Fig. 214: Role Base URL
Fig. 215: OAuth2 Service Parameters
8. Enabling Fully Asynchronous Tasks

Install and configure “rabbitmq-server”

See also:
A March 2021 blog post from RabbitMQ provides alternative installations for other systems.

Install rabbitmq-server

```
sudo apt install curl
## Import GPG Key
sudo apt update
sudo apt install curl software-properties-common apt-transport-https lsb-release
echo 'deb [signed-by=/etc/apt/keyrings/erlang.gpg] http://binaries2.erlang-solutions.com/ →ubuntu/ jammy-esl-erlang-26 contrib' | sudo tee /etc/apt/sources.list.d/erlang.list
curl -fsSL https://packages.erlang-solutions.com/ubuntu/erlang_solutions.asc | sudo gpg - →-dearmor -o /etc/apt/keyrings/erlang.gpg
## Add Erlang Repository to Ubuntu
sudo apt update
sudo apt install esl-erlang
## Add RabbitMQ Repository to Ubuntu
curl -s https://packagecloud.io/install/repositories/rabbitmq/rabbitmq-server/script.deb. →sh | sudo bash
## Install RabbitMQ Server
sudo apt install rabbitmq-server
# check the status (it should already be running)
sudo systemctl status rabbitmq-server
# check the service is enabled (it should already be enabled)
sudo systemctl is-enabled rabbitmq-server.service
# enable the web frontend and allow access through firewall
# view this interface at http://<your ip>:15672
sudo rabbitmq-plugins enable rabbitmq_management
sudo ufw allow proto tcp from any to any port 5672,15672

Create admin user
This is the user that GeoNode will use to communicate with rabbitmq-server.

```
sudo rabbitmqctl delete_user guest
sudo rabbitmqctl add_user admin <your_rabbitmq_admin_password_here>
sudo rabbitmqctl set_user_tags admin administrator
sudo rabbitmqctl add_vhost /localhost
sudo rabbitmqctl set_permissions -p /admin ".*" ".*" ".*"
sudo rabbitmqctl set_permissions -p /localhost admin ".*" ".*" ".*"
```

Managing RabbitMQ

1.12. GeoNode Advanced Installation
You can manage the rabbitmq-server service like any other service:

```
sudo systemctl stop rabbitmq-server
sudo systemctl start rabbitmq-server
sudo systemctl restart rabbitmq-server
```

You can manage the rabbitmq-server node with `rabbitmqctl`. For example, to fully reset the server, use these commands:

```
sudo rabbitmqctl stop_app
sudo rabbitmqctl reset
sudo rabbitmqctl start_app
```

After reset, you'll need to recreate the `admin` user (see above).

## Daemonize and configure Celery

### Create the Systemd unit

```
sudo vim /etc/systemd/system/celery.service
```

```
[Unit]
Description=Celery
After=network.target

[Service]
Type=simple
# the specific user that our service will run as
EnvironmentFile=/opt/geonode/.env
User=geosolutions
Group=geosolutions
# another option for an even more restricted service is
# DynamicUser=yes
# see http://0pointer.net/blog/dynamic-users-with-systemd.html
RuntimeDirectory=celery
WorkingDirectory=/opt/geonode
ExecStart=bash -c 'source /home/geosolutions/.virtualenvs/geonode/bin/activate && /opt/geonode/celery-cmd'
ExecReload=/bin/kill -s HUP $MAINPID
Restart=always
TimeoutSec=900
TimeoutStopSec=60
PrivateTmp=true

[Install]
WantedBy=multi-user.target
```

# Create the Logrotate config
```
sudo tee /etc/logrotate.d/celery <<EOF
"/var/log/celery.log" {
copytruncate
daily
rotate 5
}
EOF
```

(continues on next page)
Manage Celery

Restart Celery

```bash
# Restart Celery
sudo systemctl restart celery

# Kill old celery workers (if any)
sudo pkill -f celery
```

Inspect the logs

```bash
# Check the celery service status
sudo systemctl status celery

# Check the celery logs
sudo tail -F -n 300 /var/log/celery.log
```

Troubleshooting

Celery might crash during startup with this error:

```
looking for plugins in '/usr/lib64/sasl2', failed to open directory, error: No such file or directory
```

The workaround is a symlink at that path.

```
ln -sfn /usr/lib/x86_64-linux-gnu/sasl2/ /usr/lib64/sasl2
```

Install Memcached

```
sudo apt install memcached

sudo systemctl start memcached
sudo systemctl enable memcached

sudo systemctl restart celery
sudo systemctl status celery
```
Docker

In this section we are going to list the passages needed to deploy a vanilla GeoNode with Docker. You can follow the instructions at Docker Setup for Ubuntu (22.04) to prepare a Ubuntu 22.04 server with Docker and Docker Compose.

1. Clone GeoNode

```bash
# Let's create the GeoNode core base folder and clone it
sudo mkdir -p /opt/geonode/
sudo usermod -a -G www-data geonode
sudo chown -Rf geonode:www-data /opt/geonode/
sudo chmod -Rf 775 /opt/geonode/

# Clone the GeoNode source code on /opt/geonode
cd /opt
git clone https://github.com/GeoNode/geonode.git
```

2. Prepare the `.env` file

Follow the instructions at Docker create env file.

3. Build and run

Follow the instructions at Docker build and run.

Test the instance and follow the logs

If you run the containers daemonized (with the `-d` option), you can either run specific Docker commands to follow the startup and initialization logs or entering the image shell and check for the GeoNode logs.

In order to follow the startup and initialization logs, you will need to run the following command from the repository folder:

```bash
cd /opt/geonode
docker logs -f django4geonode
```

Alternatively:

```bash
cd /opt/geonode
docker-compose logs -f django
```

You should be able to see several initialization messages. Once the container is up and running, you will see the following statements:

```bash
... 789 static files copied to '/mnt/volumes/statics/static'.
static data refreshed
Executing UWSGI server uwsgi --ini /usr/src/app/uwsgi.ini for Production
[uWSGI] getting INI configuration from /usr/src/app/uwsgi.ini
```
To exit just hit CTRL+C.

This message means that the GeoNode containers have been started. Browsing to http://localhost/ will show the GeoNode home page. You should be able to successfully log with the default admin user (admin / admin) and start using it right away.

With Docker it is also possible to run a shell in the container and follow the logs exactly the same as you deployed it on a physical host. To achieve this run

```
docker exec -it django4geonode /bin/bash
```

# Once logged in the GeoNode image, follow the logs by executing
tail -F -n 300 /var/log/geonode.log

Alternatively:

```
docker-compose exec django /bin/bash
```

To exit just hit CTRL+C and exit to return to the host.

**Override the ENV variables to deploy on a public IP or domain**

If you would like to start the containers on a public IP or domain, let's say www.example.org, you can follow the instructions at [Deploy to production](https://docs.geonode.org/en/dev/ getting-started/installation/advanced/installation.html) variables to customize the GeoNode instance. See the GeoNode Settings section in order to get a list of the available options.

**Remove all data and bring your running GeoNode deployment to the initial stage**

This procedure allows you to stop all the containers and reset all the data with the deletion of all the volumes.

```
cd /opt/geonode

# stop containers and remove volumes
docker-compose down -v
```

**Get rid of old Docker images and volumes (reset the environment completely)**

**Note:** For more details on Docker commands, please refer to the official Docker documentation.

It is possible to let docker show which containers are currently running (add `-a` for all containers, also stopped ones)

```
# Show the currently running containers
docker ps
```

<table>
<thead>
<tr>
<th>CONTAINER ID</th>
<th>IMAGE</th>
<th>COMMAND</th>
<th>CREATED</th>
<th>STATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>4729b3dd1de7</td>
<td>geonode/geonode:4.0</td>
<td>&quot;/usr/src/geonode/en...&quot;</td>
<td>29 minutes ago</td>
<td>Up</td>
</tr>
</tbody>
</table>
Stop all the containers by running

```
docker-compose stop
```

Force kill all containers by running

```
docker kill $(docker ps -q)
```

I you want to clean up all containers and images, without deleting the static volumes (i.e. the DB and the GeoServer catalog), issue the following commands

```
# Remove all containers
docker rm $(docker ps -a -q)

# Remove all docker images
docker rmi $(docker images -q)

# Prune the old images
docker system prune -a
```

If you want to remove a volume also

```
# List of the running volumes
docker volume ls

# Remove the GeoServer catalog by its name
docker volume rm -f geonode-gsdatadir

# Remove all dangling docker volumes
docker volume rm $(docker volume ls -qf dangling=true)

# update all images, should be run regularly to fetch published updates
```

1.12. GeoNode Advanced Installation 221
1.12.2 GeoNode Project

Overview

The following steps will guide you to a new setup of GeoNode Project. All guides will first install and configure the system to run it in DEBUG mode (also known as DEVELOPMENT mode) and then by configuring an HTTPD server to serve GeoNode through the standard HTTP (80) port.

Those guides are not meant to be used on a production system. There will be dedicated chapters that will show you some hints to optimize GeoNode for a production-ready machine. In any case, we strongly suggest to task an experienced DevOp or System Administrator before exposing your server to the WEB.

Ubuntu 22.04

This part of the documentation describes the complete setup process for GeoNode on an Ubuntu 22.04 64-bit clean environment (Desktop or Server). All examples use shell commands that you must enter on a local terminal or a remote shell. - If you have a graphical desktop environment you can open the terminal application after login; - if you are working on a remote server the provider or sysadmin should has given you access through an ssh client.

Install the dependencies

In this section, we are going to install all the basic packages and tools needed for a complete GeoNode installation. To follow this guide, a piece of basic knowledge about Ubuntu Server configuration and working with a shell is required. This guide uses vim as the editor; fill free to use nano, gedit or others.

Upgrade system packages

Check that your system is already up-to-date with the repository running the following commands:

```
sudo apt update
```

Create a Dedicated User

In the following steps a User named geonode is used: to run installation commands the user must be in the sudo group.

Create User geonode if not present:

```
# Follow the prompts to set the new user's information.
# It is fine to accept the defaults to leave all of this information blank.
sudo adduser geonode

# The following command adds the user geonode to group sudo
sudo usermod -aG sudo geonode

# make sure the newly created user is allowed to login by ssh
```

(continues on next page)
# (out of the scope of this documentation) and switch to User geonode
su geonode

## Packages Installation

Add the Ubuntu GIS packages prior to installing the other system packages.

```bash
sudo apt update
```

**Note:** You don’t need to install the **system packages** if you want to run the project using Docker

First, we are going to install all the **system packages** needed for the GeoNode setup.

```bash
# Install packages from GeoNode core
sudo apt install -y python3-gdal=3.3.2+dfsg-2~focal2 gdal-bin=3.3.2+dfsg-2~focal2
libgdal-dev=3.3.2+dfsg-2~focal2
sudo apt install -y python3-pip python3-dev python3-virtualenv python3-venv
virtualenvwrapper
sudo apt install -y libxml2 libxml2-dev gettext
sudo apt install -y libxslt1-dev libjpeg-dev libpng-dev libpq-dev libmemcached-dev
sudo apt install -y software-properties-common build-essential
sudo apt install -y git unzip gcc zlib1g-dev libgeos-dev libproj-dev
sudo apt install -y sqlite3 spatialite-bin libsqlite3-mod-spatialite

# Install Openjdk
sudo -i apt update
sudo apt install openjdk-8-jdk-headless default-jdk-headless -y

# Remember to select the correct java version /usr/lib/jvm/java-8-openjdk-amd64/jre/bin/
java
sudo update-alternatives --config java

sudo apt update -y
sudo apt autoremove -y
sudo apt autoclean -y
sudo apt purge -y
sudo apt clean -y

# Install Packages for Virtual environment management
sudo apt install -y virtualenv virtualenvwrapper

# Install text editor
sudo apt install -y vim
```
**Geonode Project Installation**

Geonode project is the proper way to run a customized installation of Geonode. The repository of geonode-project contains a minimal set of files following the structure of a django-project. Geonode itself will be installed as a requirement of your project. Inside the project structure is possible to extend, replace or modify all geonode components (e.g. css and other static files, templates, models..) and even register new django apps **without touching the original Geonode code.**

**Note:** You can call your geonode project whatever you like following the naming conventions for python packages (generally lower case with underscores (_). In the examples below, replace `my_geonode` with whatever you would like to name your project.

See also the README file on geonode-project repository

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

Prepare the environment

```
sudo mkdir -p /opt/geonode_custom/
sudo usermod -a -G www-data geonode
sudo chown -Rf geonode:www-data /opt/geonode_custom/
sudo chmod -Rf 775 /opt/geonode_custom/
```

Clone the source code

```
cd /opt/geonode_custom/
git clone https://github.com/GeoNode/geonode-project.git
```

This will clone the master branch. You will have to checkout the desidered branch or tag. As an example, if you want to generate a project for GeoNode 4.2.2 you will docker

```
git checkout -b 4.2.2
```

Make an instance out of the Django Template

**Note:** We will call our instance `my_geonode`. You can change the name at your convenience.

```
vim ~/.bashrc
# add the following line to the bottom
source /usr/share/virtualenvwrapper/virtualenvwrapper.sh
```

```
source /usr/share/virtualenvwrapper/virtualenvwrapper.sh
mkvirtualenv --python=/usr/bin/python3 my_geonode
```

Alternatively you can also create the virtual env like below

```
python3.8 -m venv /home/geonode/dev/.venvs/my_geonode
source /home/geonode/dev/.venvs/my_geonode/bin/activate
```

```
pip install Django==3.2.13
```

```
django-admin startproject --template=./geonode-project -e py,sh,md,rst,json,yml,ini,env,˓→sample,properties -n monitoring-cron -n Dockerfile my_geonode
```
# Install the Python packages
```bash
cd /opt/geonode_custom/my_geonode
pip install -r src/requirements.txt --upgrade
pip install -e src/ --upgrade
```

# Install GDAL Utilities for Python
```bash
pip install pygdal==`gdal-config --version`."
```

# Dev scripts
```bash
mv .override_dev_env.sample src/.override_dev_env
mv src/manage_dev.sh.sample src/manage_dev.sh
mv src/paver_dev.sh.sample src/paver_dev.sh
```

## Install and Configure the PostgreSQL Database System

In this section we are going to install the PostgreSQL packages along with the PostGIS extension. Those steps must be done only if you don’t have the DB already installed on your system.

```
# Ubuntu 22.04
sudo sh -c "echo "deb http://apt.postgresql.org/pub/repos/apt/ `lsb_release -cs`-pgdg main" >> /etc/apt/sources.list.d/pgdg.list"
sudo wget --no-check-certificate --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc | sudo apt-key add -
sudo apt update -y; sudo apt install -y postgresql-13 postgresql-13-postgis-3 postgresql-13-postgis-3-scripts postgresql-13 postgresql-client-13
```

We now must create two databases, `my_geonode` and `my_geonode_data`, belonging to the role `my_geonode`.

**Warning:** This is our default configuration. You can use any database or role you need. The connection parameters must be correctly configured on settings, as we will see later in this section.

### Databases and Permissions

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

```bash
sudo service postgresql start
sudo -u postgres createuser -P my_geonode
```

**# Use the password:** geonode

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

**Warning:** This is a sample password used for the sake of simplicity. This password is very **weak** and should be changed in a production environment.

Create database `my_geonode` and `my_geonode_data` with owner `my_geonode`
Next let's create PostGIS extensions

```bash
sudo -u postgres psql -d my_geonode -c 'CREATE EXTENSION postgis;'
sudo -u postgres psql -d my_geonode_data -c 'CREATE EXTENSION postgis;'
sudo -u postgres psql -d my_geonode -c 'GRANT ALL ON geometry_columns TO PUBLIC;'
sudo -u postgres psql -d my_geonode -c 'GRANT ALL ON spatial_ref_sys TO PUBLIC;'
sudo -u postgres psql -d my_geonode_data -c 'GRANT ALL ON geometry_columns TO PUBLIC;'
sudo -u postgres psql -d my_geonode_data -c 'GRANT ALL ON spatial_ref_sys TO PUBLIC;'
sudo -u postgres psql -d my_geonode -c 'GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO my_geonode;'
```

Final step is to change user access policies for local connections in the file `pg_hba.conf`

```bash
sudo vim /etc/postgresql/13/main/pg_hba.conf
```

Scroll down to the bottom of the document. We want to make local connection trusted for the default user.

Make sure your configuration looks like the one below.

```
...# DO NOT DISABLE!
# If you change this first entry you will need to make sure that the
# database superuser can access the database using some other method.
# Noninteractive access to all databases is required during automatic
# maintenance (custom daily cronjobs, replication, and similar tasks).
# Database administrative login by Unix domain socket
local all postgres trust

# TYPE DATABASE USER ADDRESS METHOD

# "local" is for Unix domain socket connections only
local all all md5

# IPv4 local connections:
host all all 127.0.0.1/32 md5

# IPv6 local connections:
host all all ::1/128 md5

# Allow replication connections from localhost, by a user with the
# replication privilege.
local replication all peer
host replication all 127.0.0.1/32 md5
host replication all ::1/128 md5
```

**Warning:** If your PostgreSQL database resides on a separate/remote machine, you’ll have to allow remote access to the databases in the `pg_hba.conf` file to the geonode user and tell PostgreSQL to accept non-local connections in your `/etc/postgresql/13/main/postgresql.conf` file.
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 role.

```
psql -U postgres my_geonode
# This should not ask for any password
psql -U my_geonode my_geonode
# This should ask for the password geonode

# Repeat the test with geonode_data DB
psql -U postgres my_geonode_data
psql -U my_geonode my_geonode_data
```

### Run GeoNode Project for the first time in DEBUG Mode

**Warning:** Be sure you have successfully completed all the steps of the section *Install the dependencies*.

This command will run both GeoNode and GeoServer locally after having prepared the Spatialite database. The server will start in DEBUG (or DEVELOPMENT) mode, and it will start the following services:


This modality is beneficial to debug issues and/or develop new features, but it cannot be used on a production system.

```
# Prepare the GeoNode Spatialite database (the first time only)
cd src/
chmod +x paver_dev.sh
./paver_dev.sh setup
./paver_dev.sh sync
```

**Note:** In case you want to start again from a clean situation, just run

```
./paver_dev.sh reset_hard
```

**Warning:** This will blow up completely your local_settings, delete the SQLite database and remove the GeoServer data dir.

```
# Run the server in DEBUG mode
./paver_dev.sh start
```

Once the server has finished the initialization and prints on the console the sentence *GeoNode is now available.*, you can open a browser and go to:
http://localhost:8000/

Sign-in with:

user: admin
password: admin

From now on, everything already said for GeoNode Core (please refer to the section configure_dbs_core and following), applies to a GeoNode Project.

Be careful to use the new paths and names everywhere:

- Everytime you’ll find the keyword geonode, you’ll need to use your geonode custom name instead (in this example my_geonode).
- Everytime you’ll find paths pointing to /opt/geonode/, you’ll need to update them to point to your custom project instead (in this example /opt/geonode_custom/my_geonode).

**Docker**

**Warning:** Before moving with this section, you should have read and clearly understood the INSTALLATION > GeoNode Core sections, and in particular the Docker one. Everything said for the GeoNode Core Vanilla applies here too, except that the Docker container names will be slightly different. As an instance if you named your project my_geonode, your containers will be called:

'django4my_geonode' instead of 'django4geonode' and so on...

**Deploy an instance of a geonode-project Django template with Docker on localhost**

Prepare the environment

```
sudo mkdir -p /opt/geonode_custom/
sudo usermod -a -G www-data geonode
sudo chown -Rf geonode:www-data /opt/geonode_custom/
sudo chmod -Rf 775 /opt/geonode_custom/
```

Clone the source code

```
cd /opt/geonode_custom/
git clone https://github.com/GeoNode/geonode-project.git
```

This will clone the master branch. You will have to checkout the desired branch or tag. As an example, if you want to generate a project for GeoNode 4.2.2 you will docker

```
git checkout -b 4.2.2
```

Make an instance out of the Django Template

**Note:** We will call our instance my_geonode. You can change the name at your convenience.
source /usr/share/virtualenvwrapper/virtualenvwrapper.sh
mkvirtualenv --python=/usr/bin/python3 my_geonode

Alternatively you can also create the virtual env like below
python3.8 -m venv /home/geonode/dev/.venvs/my_geonode
source /home/geonode/dev/.venvs/my_geonode/bin/activate

pip install Django==3.2.13
django-admin startproject --template=./geonode-project -e py,sh,md,rst,json,yml,ini,env,˓
→ sample,properties -n monitoring-cron -n Dockerfile my_geonode
cd /opt/geonode_custom/my_geonode

Create the .env file

An .env file is required to run the application. It can be created from the .env.sample either manually or with the create-envfile.py script.

The script accepts several parameters to create the file, in detail:

• **hostname**: e.g. master.demo.geonode.org, default localhost
• **https**: (boolean), default value is False
• **email**: Admin email (this is required if https is set to True since a valid email is required by Letsencrypt certbot)
• **env_type**: prod, test or dev. It will set the DEBUG variable to False (prod, test) or True (dev)
• **geonodepwd**: GeoNode admin password (required inside the .env)
• **geoserverpwd**: Geoserver admin password (required inside the .env)
• **pgpwd**: PostgreSQL password (required inside the .env)
• **dbpwd**: GeoNode DB user password (required inside the .env)
• **geodbpwd**: Geodatabase user password (required inside the .env)
• **clientid**: Oauth2 client id (required inside the .env)
• **clientsecret**: Oauth2 client secret (required inside the .env)
• **secret key**: Django secret key (required inside the .env)
• **sample_file**: absolute path to a env_sample file used to create the env_file. If not provided, the one inside the GeoNode project is used.
• **file**: absolute path to a json file that contains all the above configuration

**Note**: if the same configuration is passed in the json file and as an argument, the CLI one will overwrite the one in the JSON file. If some value is not provided, a random string is used

Example USAGE

```bash
python create-envfile.py -f /opt/core/geonode-project/file.json \
    --hostname localhost \
    --https \
    --email random@email.com \
    --geonodepwd gn_password
```

(continues on next page)
--geoserverpwd gs_password \ 
--pgpwd pg_password \ 
--dbpwd db_password \ 
--geodbpwd _db_password \ 
--clientid 12345 \ 
--clientsecret abc123

Example JSON expected:

```json
{
    "hostname": "value",
    "https": "value",
    "email": "value",
    "geonodepwd": "value",
    "geoserverpwd": "value",
    "pgpwd": "value",
    "dbpwd": "value",
    "geodbpwd": "value",
    "clientid": "value",
    "clientsecret": "value"
}
```

Modify the code and the templates and rebuild the Docker Containers

docker-compose -f docker-compose.yml build --no-cache

Finally, run the containers

docker-compose -f docker-compose.yml up -d

**Deploy an instance of a geonode-project Django template with Docker on a domain**

**Note:** We will use www.example.org as an example. You can change the name at your convenience.

Stop the containers

```bash
cd /opt/geonode_custom/my_geonode
docker-compose -f docker-compose.yml stop
```

Edit the `ENV` override file in order to deploy on www.example.org

Replace everywhere localhost with www.example.org

```bash
vim .env
```

# e.g.: %s/localhost/www.example.org/g
Note: It is possible to override here even more variables to customize the GeoNode instance. See the GeoNode Settings section in order to get a list of the available options.

Run the containers in daemon mode

```
docker-compose -f docker-compose.yml -f docker-compose.override.example-org.yml up --build -d
```

Test geonode-project with vagrant

Note: Inside geonode-project files you will find one file named Vagrantfile.compose and one named Vagrantfile.stack, copy one of them onto file Vagrantfile to use them with vagrant.

```
apt-get install -y vagrant
cp Vagrantfile.compose Vagrantfile
# this will start a vagrant virtual machine, generate and build geonode-project
vagrant up
# check services are up upon reboot
vagrant ssh geonode-compose -c 'docker ps'
vagrant destroy -f
# test docker swarm
cp Vagrantfile.stack Vagrantfile
vagrant up
# check services are up upon reboot
vagrant ssh geonode-vagrant -c 'docker service ls'
vagrant destroy -f
```

Note: Vagrant will generate a dummy project named “antani” inside vagrant, starting with the geonode-project code-base, this way it is possible to test inside vagrant almost instantly what one modifies into geonode-project.

1.12.3 Further Production Enhancements

GeoServer Production Settings

JVM Settings: Memory And GeoServer Options

The .env file provides a way to customize GeoServer JVM Options.

The variable GEOSERVER_JAVA_OPTS allows you to tune-up the GeoServer container and to enable specific GeoServer options.

```
GEOSERVER_JAVA_OPTS=
   -Djava.awt.headless=true -Xms4G -Xmx4G -XX:PerfDataSamplingInterval=500
   -XX:SoftRefLRUPolicyMSPerMB=36000 -XX:-UseGCOverheadLimit -XX:+UseConcMarkSweepGC
   -XX:+UseParNewGC -XX:ParallelGCThreads=4 -Dfile.encoding=UTF8 -Djava.awt.
```

(continues on next page)
GeoNode Documentation, Release master

(request.encoding=UTF-8
-Djavax.servlet.response.encoding=UTF-8 -Duser.timezone=GMT

-Djava.awt.headless (true)

Work with graphics-based applications in Java without an actual display, keyboard, or mouse. A typical use case of UI components running in a headless environment could be an image converter app. Though it needs graphics data for image processing, a display is not really necessary. The app could be run on a server and converted files saved or sent over the network to another machine for display.

-Xms4G -Xmx4G

This means that your JVM will be started with Xms amount of memory and will be able to use a maximum of Xmx amount of memory. Above will start a JVM like with 2 GB of memory and will allow the process to use up to 4 GB of memory. You need to adjust this value depending on your available RAM.

-DGEOSERVER_CSRF_DISABLED (True)

The GeoServer web admin employs a CSRF (Cross-Site Request Forgery) protection filter that will block any form submissions that didn't appear to originate from GeoServer. This can sometimes cause problems for certain proxy configurations. You can disable the CSRF filter by setting the GEOSERVER_CSRF_DISABLED property to true. https://docs.geoserver.org/stable/en/user/security/webadmin/csrf.html

Whenever you need to change one or more of the JVM options, you will need to restart the GeoServer Docker container.

# Hard restart of the container: the only way to update the .env variables

docker-compose up -d geoserver

This command will preserve all the GeoServer configuration and data, since the GEOSERVER_DATA_DIR is stored on a Docker static volume.

Nevertheless, any change you have made manually to the container, e.g. added a new plugin to GeoServer or updated some JARs into the WEB-INF/lib library folder, will be lost.

You will need to add the JARs again and restart GeoServer softly

# Soft restart of the container: the .env variables won't be updated

docker-compose restart geoserver

Global And Services Settings

- Check the GeoServer Memory usage and status; ensure the GEOSERVER_DATA_DIR path points to the static volume
- GeoServer Global Settings; make sure the Proxy Base Url points to the public URL and the LOGGING levels are set to Production Mode
- GeoServer Image Processing Settings; unless you are using some specific renderer or GeoServer plugin, use the following recommended options

Note: Further details at https://docs.geoserver.org/stable/en/user/configuration/image_processing/index.html#image-processing

- Tune up GeoServer Services Configuration; WCS, WFS, WMS and WPS;
Fig. 216: GeoServer Status

Fig. 217: Global Settings

Fig. 218: Image Processing Settings
– WCS: Update the limits accordingly to your needs. Do not use very high values, this will set GeoServer prone to DoS Attacks.

![Resource Consumption Limits](image)

Fig. 219: WCS Resource Consumption Limits

– WMS: Specify here the SRS List you are going to use. Empty means all the ones supported by GeoServer, but be careful since the GetCapabilities output will become huge.

![Limited SRS list](image)

Fig. 220: WMS Supported SRS List

– WMS: Raster Rendering Options allows you to tune up the WMS output for better performance or quality. Best Performance: Nearest Neighbour - Best Quality: Bicubic

**Warning:** Raster Images should be always optimized before ingested into GeoNode. The general recommendation is to **never** upload a non-processed GeoTIFF image to GeoNode.

Further details at:


– WMS: Update the limits accordingly to your needs. Do not use very high values, this will set GeoServer prone to DoS Attacks.
GeoWebCache DiskQuota On Postgis

By default GeoWebCache DiskQuota is disabled. That means that the layers cache might potentially grow up indefinitely.

GeoWebCache DiskQuota should be always enabled on a production system. In the case it is enabled, this **must** be configured to make use of a DB engine like Postgis to store its indexes.

- First of all ensure **Tile Caching** is enabled on all available layers

**Note:** GeoNode typically does this automatically for you. It is worth to double check anyway.

- Configure **Disk Quota** by providing the connection string to the DB Docker Container as specified in the `.env` file
Fig. 223: *Tile Caching: Tiled Datasets*

Fig. 224: *Tile Caching: Disk Quota Configuration*
GeoFence Security Rules On Postgis

By default GeoFence stores the security rules on an H2 db.

On a production system, this is not really recommended. You will need to update the GeoServer Docker container in order to enable GeoFence storing the rules into the DB Docker Container instead.

In order to do that, follow the procedure below:

```
# Enter the GeoServer Docker Container
docker-compose exec geoserver bash

# Install a suitable editor
apt update
apt install nano

# Edit the GeoFence DataStore .properties file
nano /geoserver_data/data/geofence/geofence-datasource-ovr.properties

Note: Make sure to provide the same connection parameters specified in the .env file
```

```
geofenceVendorAdapter.databasePlatform=org.hibernate.spatial.postgis.PostgisDialect
geofenceDataSource.driverClassName=org.postgresql.Driver
geofenceDataSource.url=jdbc:postgresql://db:5432/my_geonode_data
geofenceDataSource.username=my_geonode_data
geofenceDataSource.password=********
geofenceEntityManagerFactory.jpaPropertyMap[hibernate.default_schema]=public
```

```
# Update the GeoServer WEB-INF/lib JARs accordingly
wget --no-check-certificate "https://repo1.maven.org/maven2/org/postgis/postgis-jdbc/1.3.3/postgis-jdbc-1.3.3.jar" -O postgis-jdbc-1.3.3.jar && \
wget --no-check-certificate "https://maven.geo-solutions.it/org/hibernatespatial/hibernate-spatial-postgis/1.1.3.2/hibernate-spatial-postgis-1.1.3.2.jar" -O hibernate-spatial-postgis-1.1.3.2.jar && \
rm /usr/local/tomcat/webapps/geoserver/WEB-INF/lib/hibernate-spatial-h2-geodb-1.1.3.1.jar && \
mv hibernate-spatial-postgis-1.1.3.2.jar /usr/local/tomcat/webapps/geoserver/WEB-INF/lib/ && \
mv postgis-jdbc-1.3.3.jar /usr/local/tomcat/webapps/geoserver/WEB-INF/lib/
```

The container is ready to be restarted now.

```
Warning: Remember to do a soft restart otherwise the WEB-INF/lib JARs will be reset to the original state
```

```
# Exit the GeoServer container
exit

# Soft Restart GeoServer Docker Container
docker-compose restart geoserver

1.12. GeoNode Advanced Installation  237
```
**IMPORTANT**: The first time you perform this procedure, GeoFence won’t be able to retrieve the old security rules anymore.

You will need to **Fixup GeoNode Datasets Permissions** in order to regenerate the security rules.

**Fixup GeoNode Datasets Permissions**

The list of the GeoFence Security Rules is available from the *GeoFence Data Rules* section.

Always double check the list is accessible and the data rules are there. If empty, no layer will be accessible by standard users other than admin.

Fig. 225: *GeoFence Data Rules*

In order to re-sync the GeoFence security rules, follow the procedure below:

```bash
# Enter the GeoNode Docker Container
docker-compose exec django bash

# Run the `sync_geonode_datasets` management command
./manage.sh sync_geonode_datasets --updatepermissions
```

**Regenerate GeoNode Datasets Thumbnails**

The following procedure allows you to *batch* regenerate all Datasets Thumbnails:

```bash
# Enter the GeoNode Docker Container
docker-compose exec django bash

# Run the `sync_geonode_datasets` management command
./manage.sh sync_geonode_datasets --updatethumbnails
```
**Regenerate GeoNode Datasets BBOXES**

The following procedure allows you to *batch* regenerate all Datasets BBOXES:

```
# Enter the GeoNode Docker Container
docker-compose exec django bash

# Run the `sync_geonode_datasets` management command
./manage.sh sync_geonode_datasets --updatebbox
```

**Fixup GeoNode Datasets Metadata And Download Links**

The following procedure allows you to fix-up broken or incorrect Metadata Links:

```
# Enter the GeoNode Docker Container
docker-compose exec django bash

# Run the `set_all_datasets_metadata` management command
./manage.sh set_all_datasets_metadata -d
```

It is also possible to *force* purging the links before regenerating:

```
# Enter the GeoNode Docker Container
docker-compose exec django bash

# Run the `set_all_datasets_metadata` management command
./manage.sh set_all_datasets_metadata -d --prune
```

**Migrate GeoNode To A New Hostname**

In the case you will need to move your instance to another domain, as an example from https://my_geonode.geonode.org/ to https://prod_geonode.geonode.org/, follow the procedure below:

- Update the .env file by specifying the new name accordingly.
- Restart the GeoNode Docker Container.

```
docker-compose up -d geonode
```

- Run the following management commands from inside the GeoNode Docker Container.

```
# Enter the GeoNode Docker Container
docker-compose exec django bash

# Run the `migrate_baseurl` management command
./manage.sh migrate_baseurl --source-address=my_geonode.geonode.org --
˓→target-address=prod_geonode.geonode.org

# Run the `set_all_datasets_metadata` management command
./manage.sh set_all_datasets_metadata -d
```
**Add Huge Or DB Datasets To Your Instance**

Uploading huge datasets, or DB tables, to GeoNode from the Web Upload Interface is not really possible sometimes. The suggested procedure in such cases is the following one:

- Add the dataset to GeoServer first directly.
  
  You must upload the data into the GeoServer Docker Container Static Volume first and then adding manually the layer through the GeoServer Admin GUI.

- Once the dataset is correctly configured on GeoServer, run the following management command from inside the GeoNode Docker Container

```bash
# Enter the GeoNode Docker Container
docker-compose exec django bash

# Run the `updatelayers` management command
./manage.sh updatelayers -w <workspace_name> -f <layer_name>
```

### 1.13 GeoNode Settings

*Settings*

#### 1.13.1 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.

For most of them, default values are good. Those should be changed only for advanced configurations in production or heavily hardened systems.

The most common ones can be set through environment variables to avoid touching the `settings.py` file at all. This is a good practice and also the preferred one to configure GeoNode (and Django apps in general). Whenever you need to change them, set the environment variable accordingly (where it is available) instead of overriding it through the `local_settings`.

**ACCESS_TOKEN_EXPIRE_SECONDS**

Default: `86400`  
Env: `ACCESS_TOKEN_EXPIRE_SECONDS`

When a user logs into GeoNode, if no ACCESS_TOKEN exists, a new one will be created with a default expiration time of `ACCESS_TOKEN_EXPIRE_SECONDS` seconds (1 day by default).
ACCOUNT_ADAPTER

Default: geonode.people.adapters.LocalAccountAdapter
Custom GeoNode People (Users) Account Adapter.

ACCOUNT_APPROVAL_REQUIRED

Default: False
Env: ACCOUNT_APPROVAL_REQUIRED
If ACCOUNT_APPROVAL_REQUIRED equals True, newly registered users must be activated by a superuser through the Admin gui, before they can access GeoNode.

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 which 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

ACCOUNT_LOGIN_REDIRECT_URL

Default: SITEURL
Env: LOGIN_REDIRECT_URL
This is a django-user-accounts setting It allows specifying the default redirect URL after a successful login.
**ACCOUNT_LOGOUT_REDIRECT_URL**

Default: SITEURL  
Env: LOGOUT_REDIRECT_URL  
This is a django-user-accounts setting It allows specifying the default redirect URL after a successful logout.

**ACCOUNT_NOTIFY_ON_PASSWORD_CHANGE**

Default: True  
Env: ACCOUNT_NOTIFY_ON_PASSWORD_CHANGE  
This is a django-user-accounts setting

**ACCOUNT_OPEN_SIGNUP**

Default: True  
Env: ACCOUNT_OPEN_SIGNUP  
This is a django-user-accounts setting Whether or not people are allowed to self-register to GeoNode or not.

**ACCOUNT_SIGNUP_FORM_CLASS**

Default: geonode.people.forms.AllauthReCaptchaSignupForm  
Env: ACCOUNT_SIGNUP_FORM_CLASS  
Enabled only when the RECAPTCHA_ENABLED option is True.  
Ref. to RECAPTCHA_ENABLED

**ACTSTREAM_SETTINGS**

Default:

```python
{
    'FETCH_RELATIONS': True,
    'USE_PREFETCH': False,
    'USE_JSONFIELD': True,
    'GFK_FETCH_DEPTH': 1,
}
```

Actstream Settings.
ADDITIONAL_DATASET_FILE_TYPES

External application can define additional supported file type other than the default one declared in the SUPPORTED_DATASET_FILE_TYPES.

The variable should be declared in this way in settings.py (or via application hook):

Please rely on geonode.tests.test_utils.TestSupportedTypes for an example

ADMIN_IP_WHITELIST

Default: []

When this list is populated with a list of IPs or IP ranges (e.g. 192.168.1.0/24) requests from and admin user will be allowed only from IPs matching with the list.

ADMIN_Moderate_UPLOAdS

Default: False

When this variable is set to True, every uploaded resource must be approved before becoming visible to the public users.

Until a resource is in PENDING APPROVAL state, only the superusers, owner and group members can access it, unless specific edit permissions have been set for other users or groups.

A Group Manager can approve the resource, but he cannot publish it whenever the setting RESOURCE_PUBLISHING is set to True. Otherwise, if RESOURCE_PUBLISHING is set to False, the resource becomes accessible as soon as it is approved.

ADMINS_ONLY_NOTICE_TYPES

Default: ['monitoring_alert',]

A list of notification labels that standard users should not either see or set.

Such notifications will be hidden from the notify settings page and automatically set to false for non-superusers.

ADVANCED_EDIT_EXCLUDE_FIELD

Default: []

A list of element (item name) to exclude from the Advanced Edit page.

Example:

ADVANCED_EDIT_EXCLUDE_FIELD=['title', 'keywords', 'tkeywords']
AGON_RATINGS_CATEGORY_CHOICES

Default:

```python
{
    "maps.Map": {
        "map": "How good is this map?"
    },
    "layers.Layer": {
        "layer": "How good is this layer?"
    },
    "documents.Document": {
        "document": "How good is this document?"
    }
}
```

ALLOWED_DOCUMENT_TYPES

Default:

```python
['doc', 'docx', 'gif', 'jpg', 'jpeg', 'ods', 'odt', 'odp', 'pdf', 'png',
 'ppt', 'pptx', 'rar', 'sld', 'tif', 'tiff', 'txt', 'xls', 'xlsx', 'xml',
 'zip', 'gz', 'qml']
```

A list of acceptable file extensions that can be uploaded to the Documents app.

ANONYMOUS_USER_ID

Default: -1

Env: ANONYMOUS_USER_ID

The id of an anonymous user. This is a django-guardian setting.

API_INCLUDE_REGIONS_COUNT

Default: False

Env: API_INCLUDE_REGIONS_COUNT

If set to True, a counter with the total number of available regions will be added to the API JSON Serializer.

API_LIMIT_PER_PAGE

Default: 200

Env: API_LIMIT_PER_PAGE

The Number of items returned by the APIs 0 equals no limit. Different from CLIENT_RESULTS_LIMIT, affecting the number of items per page in the resource list.
**API_LOCKDOWN**

Default: True
Env: API_LOCKDOWN

If this is set to True users must be authenticated to get search results when search for users, groups, categories, regions, tags etc. Filtering search results of Resourcebase-objects like Layers, Maps or Documents by one of the above types does not work. Attention: If API_LOCKDOWN is set to False all details can be accessed by anonymous users.

**ASYNC_SIGNALS**

Default: False
Env: ACCOUNT_NOTIFY_ON_PASSWORD_CHANGE

**AUTH_EXEMPT_URLS**

Default:

```python
\(r'^/?$',
'/gs/*',
'/static/*',
'/o/*',
'/api/o/*',
'/api/roles',
'/api/adminRole',
'/api/users',
'/api/layers',)
```

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.

**AUTO_ASSIGN_REGISTERED_MEMBERS_TO_CONTRIBUTORS**

Default: True
Env: AUTO_ASSIGN_REGISTERED_MEMBERS_TO_CONTRIBUTORS

Assign new registered users to the contributors group. If set to False new registered members will not obtain automatic permissions to create and edit resources.
AUTO_ASSIGN_REGISTERED_MEMBERS_TO_REGISTERED_MEMBERS_GROUP_NAME

Default: True
Env: AUTO_ASSIGN_REGISTERED_MEMBERS_TO_REGISTERED_MEMBERS_GROUP_NAME
Auto assign users to a default REGISTERED_MEMBERS_GROUP_NAME private group after AUTO_ASSIGN_REGISTERED_MEMBERS_TO_REGISTERED_MEMBERS_GROUP_AT.

AUTO_ASSIGN_REGISTERED_MEMBERS_TO_REGISTERED_MEMBERS_GROUP_AT

Default: activation
Env: AUTO_ASSIGN_REGISTERED_MEMBERS_TO_REGISTERED_MEMBERS_GROUP_AT
Options: "registration" | "activation" | "login"
Auto assign users to a default REGISTERED_MEMBERS_GROUP_NAME private group after {"registration" | "activation" | "login"}.
Notice that whenever ACCOUNT_EMAIL_VERIFICATION == True and ACCOUNT_APPROVAL_REQUIRED == False, users will be able to register and they became active already, even if they won’t be able to login until the email has been verified.

AUTO_GENERATE_AVATAR_SIZES

Default: 20, 30, 32, 40, 50, 65, 70, 80, 100, 140, 200, 240
An iterable of integers representing the sizes of avatars to generate on upload. This can save rendering time later on if you pre-generate the resized versions.

AVATAR_GRAVATAR_SSL

Default: False
Env: AVATAR_GRAVATAR_SSL
Options: True | False
Force SSL when loading fallback image from gravatar.com.

AVATAR_DEFAULT_URL

Default: /geonode/img/avatar.png
Env: AVATAR_GRAVATAR_SSL
Options: "filepath to image"
Allows to set a custom fallback image in case a User has not uploaded a profile image. Needs AVATAR_PROVIDERS to be set correctly.
AVATAR_PROVIDERS

Default:

```
'avatar.providers.PrimaryAvatarProvider', 'avatar.providers.DefaultAvatarProvider', 'avatar.providers.GravatarAvatarProvider'
```

Env: AVATAR_PROVIDERS
Options: Avatar provider object

This setting configures in which order gravatar images are loaded. A common use case is the use of a local image over a fallback image loaded from gravatar.com. To do so you would change the order like:

```
'avatar.providers.PrimaryAvatarProvider', 'avatar.providers.DefaultAvatarProvider', 'avatar.providers.GravatarAvatarProvider'
```

(DefaultAvatarProvider before GravatarAvatarProvider)

B

BING_API_KEY

Default: None

Env: BING_API_KEY

This property allows to enable a Bing Aerial background.

If using mapstore client library, make sure the MAPSTORE_BASELAYERS include the following:

```
if BING_API_KEY:
    BASEMAP = {
        "type": "bing",
        "title": "Bing Aerial",
        "name": "AerialWithLabels",
        "source": "bing",
        "group": "background",
        "apiKey": "{{apiKey}}",
        "visibility": False
    }

    DEFAULT_MS2_BACKGROUNDS = [BASEMAP,] + DEFAULT_MS2_BACKGROUNDS
```
**BROKER_HEARTBEAT**

Default: 0

Heartbeats are used both by the client and the broker to detect if a connection was closed. This is a Celery setting.

**BROKER_TRANSPORT_OPTIONS**

Default:

```python
{
'fanout_prefix': True,
'fanout_patterns': True,
 'socket_timeout': 60,
 'visibility_timeout': 86400
}
```

This is a Celery setting.

**CACHES**

Default:

```python
CACHES = {
    'default': {
        'BACKEND': 'django.core.cache.backends.dummy.DummyCache',
    },
    'resources': {
        'BACKEND': 'django.core.cache.backends.locmem.LocMemCache',
        'TIMEOUT': 600,
        'OPTIONS': {
            'MAX_ENTRIES': 10000
        }
    }
}
```

A dictionary containing the settings for all caches to be used with Django. This is a Django setting.

The 'default' cache is disabled because we don’t have a mechanism to discriminate between client sessions right now, and we don’t want all users fetch the same api results.

The 'resources' is not currently used. It might be helpful for caching Django template fragments and/or Tastypie API Caching.
CACHE_BUSTING_STATIC_ENABLED

Default: False
Env: CACHE_BUSTING_STATIC_ENABLED

This is a Django Compressed Manifet storage provided by WhiteNoise. A boolean allowing you to enable the WhiteNoise CompressedManifestStaticFilesStorage storage. This works only on a production system.

**Warning:** This works only if DEBUG = False

CASCADE_WORKSPACE

Default: geonode
Env: CASCADE_WORKSPACE

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
- **USER-NAME**: 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.

CATALOGUE_METADATA_TEMPLATE

Default: catalogue/full_metadata.xml

A string with the catalogue xml file needed for the metadata.

CATALOGUE_METADATA_XSL

Default: '/static/metadataxsl/metadata.xsl

A string pointing to the XSL used to transform the metadata XML into human readable HTML.

CELERYD_POOL_RESTARTS

Default: True

This is a Celery setting.
CELERY_ACCEPT_CONTENT

Default: ['json']
This is a Celery setting.

CELERY_ACKS_LATE

Default: True
This is a Celery setting.

CELERY_BEAT_SCHEDULE

Here you can define your scheduled task.

CELERY_DISABLE_RATE_LIMITS

Default: False
This is a Celery setting.

CELERY_ENABLE_UTC

Default: True
This is a Celery setting.

CELERY_MAX_CACHED_RESULTS

Default: 32768
This is a Celery setting.

CELERY_MESSAGE_COMPRESSION

Default: gzip
This is a Celery setting.

CELERY_RESULT_PERSISTENT

Default: False
This is a Celery setting.
CELERY_RESULT_SERIALIZER

Default: json
This is a Celery setting.

CELERY_SEND_TASK_SENT_EVENT

Default: True
If enabled, a task-sent event will be sent for every task so tasks can be tracked before they are consumed by a worker. This is a Celery setting.

CELERY_TASK_ALWAYS_EAGER

Default: False if ASYNC_SIGNALS else True
This is a Celery setting.

CELERY_TASK_CREATE_MISSING_QUEUES

Default: True
This is a Celery setting.

CELERY_TASK_IGNORE_RESULT

Default: True
This is a Celery setting.

CELERY_TASK_QUEUES

Default:

```
Queue('default', GEONODE_EXCHANGE, routing_key='default'),
Queue('geonode', GEONODE_EXCHANGE, routing_key='geonode'),
Queue('update', GEONODE_EXCHANGE, routing_key='update'),
Queue('cleanup', GEONODE_EXCHANGE, routing_key='cleanup'),
Queue('email', GEONODE_EXCHANGE, routing_key='email'),
```

A tuple with registered Queues.
**CELERY_TASK_RESULT_EXPIRES**

Default: 43200
Env: CELERY_TASK_RESULT_EXPIRES
This is a Celery setting.

**CELERY_TASK_SERIALIZER**

Default: json
Env: CELERY_TASK_SERIALIZER
This is a Celery setting.

**CELERY_TIMEZONE**

Default: UTC
Env: TIME_ZONE
This is a Celery setting.

**CELERY_TRACK_STARTED**

Default: True
This is a Celery setting.

**CELERY_WORKER_DISABLE_RATE_LIMITS**

Default: False
Disable the worker rate limits (number of tasks that can be run in a given time frame).

**CELERY_WORKER_SEND_TASK_EVENTS**

Default: False
Send events so the worker can be monitored by other tools.

**CLIENT_RESULTS_LIMIT**

Default: 5
Env: CLIENT_RESULTS_LIMIT
The number of results per page listed in the GeoNode search pages. Different from API_LIMIT_PER_PAGE, affecting the number of items returned by the APIs.
**CORS_ALLOW_ALL_ORIGINS**

Default: False  
Env: CORS_ALLOW_ALL_ORIGINS

If set to true `Access-Control-Allow-Origin: *` header is set for any response. A safer option (not managed through env vars at the moment) is `CORS_ALLOWED_ORIGINS`, where a list of hosts can be configured, or `CORS_ALLOWED_ORIGIN_REGEXES`, where the list can contain regexes. Notice that the Nginx in front of GeoNode always includes `Access-Control-Allow-Credentials true`. This must also taken into account when CORS is enabled.

**CREATE_LAYER**

Default: False  
Env: CREATE_LAYER

Enable the create layer plugin.

**CKAN_ORIGINS**

Default:

```python
CKAN_ORIGINS = [{
    "label": "Humanitarian Data Exchange (HDX)",
    "url": "https://data.hdx.rwlabs.org/dataset/new?title={name}&notes={abstract}",
    "css_class": "hdx"
}]
```

A list of dictionaries that are 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)`.

**CSRF_COOKIE_HTTPONLY**

Default: False  
Env: CSRF_COOKIE_HTTPONLY

Whether to use HttpOnly flag on the CSRF cookie. If this is set to True, client-side JavaScript will not be able to access the CSRF cookie. This is a Django Setting.
CSRF_COOKIE_SECURE

Default: False
Env: CSRF_COOKIE_SECURE

Whether to use a secure cookie for the CSRF cookie. If this is set to True, the cookie will be marked as “secure,” which means browsers may ensure that the cookie is only sent with an HTTPS connection. This is a Django Setting.

CUSTOM_METADATA_SCHEMA

Default: {}

If present, will extend the available metadata schema used for store new value for each resource. By default override the existing one. The expected schema is the same as the default.

DATA_UPLOAD_MAX_NUMBER_FIELDS

Default: 100000

Maximum value of parsed attributes.

DATASET_DOWNLOAD_HANDLERS

Default: []

Additional download handlers that provides a link to download the resource.

DEBUG

Default: False
Env: DEBUG

One of the main features of debug mode is the display of detailed error pages. If your app raises an exception when DEBUG is True, Django will display a detailed traceback, including a lot of metadata about your environment, such as all the currently defined Django settings (from settings.py). This is a Django Setting.

DEBUG_STATIC

Default: False
Env: DEBUG_STATIC

Load non minified version of static files.
DEFAULT_ANONYMOUS_DOWNLOAD_PERMISSION

Default: True
Whether the uploaded resources should downloadable by default.

DEFAULT_ANONYMOUS_VIEW_PERMISSION

Default: True
Whether the uploaded resources should be public by default.

DEFAULT_AUTO_FIELD

Default: django.db.models.AutoField
Default primary key field type to use for models that don’t have a field with primary_key=True. Django documentation https://docs.djangoproject.com/it/3.2/ref/settings/#std:setting-DEFAULT_AUTO_FIELD

DEFAULT_DATASET_DOWNLOAD_HANDLER

Default: geonode.layers.download_handler.DatasetDownloadHandler
from GeoNode 4.2.x has bee introduced with this issue #11296 and later improved with this issue #11421 the concept of Download Handler and ofc GeoNode provides a default implementation of it which process the download via WPS

DEFAULT_EXTRA_METADATA_SCHEMA

Default

```python
{
    Optional("id"): int,
    "filter_header": object,
    "field_name": object,
    "field_label": object,
    "field_value": object,
}
```

Define the default metadata schema used for add to the resource extra metadata without modify the actual model. This schema is used as validation for the input metadata provided by the user

- **id**: (optional int): the identifier of the metadata. Optional for creation, required in Upgrade phase
- **filter_header**: (required object): Can be any type, is used to generate the facet filter header. Is also an identifier.
- **field_name**: (required object): name of the metadata field
- **field_label**: (required object): verbose string of the name. Is used as a label in the facet filters.
- **field_value**: (required object): metadata values

An example of metadata that can be ingested is the follow:
[{
    "filter_header": "Bike Brand",
    "field_name": "name",
    "field_label": "Bike Name",
    "field_value": "KTM",
  },
  {
    "filter_header": "Bike Brand",
    "field_name": "name",
    "field_label": "Bike Name",
    "field_value": "Bianchi",
  }]

DEFAULT_LAYER_FORMAT

Default: image/png
Env: DEFAULT_LAYER_FORMAT
The default format for requested tile images.

DEFAULT_MAP_CENTER

Default: (0, 0)
Env: DEFAULT_MAP_CENTER_X DEFAULT_MAP_CENTER_Y
A 2-tuple with the latitude/longitude coordinates of the center-point to use in newly created maps.

DEFAULT_MAP_CRS

Default: EPSG:3857
Env: DEFAULT_MAP_CRS
The default map projection. Default: EPSG:3857

DEFAULT_MAP_ZOOM

Default: 0
Env: DEFAULT_MAP_ZOOM
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.
**DEFAULT_MAX_PARALLEL_UPLOADS_PER_USER**

Default: 5

When uploading datasets, this value limits the number of parallel uploads.

The parallelism limit is set during installation using the value of this variable. After installation, only an user with administrative rights can change it. These limits can be changed in the admin panel or accessing by api.

**DEFAULT_MAX_UPLOAD_SIZE**

Default: 104857600 (100 MB in bytes)

When uploading datasets or uploading documents, the total size of the uploaded files is verified.

The size limits are set during installation using the value of this variable. After installation, only an user with administrative rights can change it. These limits can be changed in the admin panel or accessing by api.

**DEFAULT_SEARCH_SIZE**

Default: 10

Env: DEFAULT_SEARCH_SIZE

An integer that specifies the default search size when using geonode.search for querying data.

**DEFAULT_WORKSPACE**

Default: geonode

Env: DEFAULT_WORKSPACE

The standard GeoServer workspace.

**DELAYED_SECURITY_SIGNALS**

Default: False

Env: DELAYED_SECURITY_SIGNALS

This setting only works when GEOFENCE_SECURITY_ENABLED has been set to True and GeoNode is making use of the GeoServer BACKEND.

By setting this to True, every time the permissions will be updated/changed for a Layer, they won’t be applied immediately but only and only if either:

a. A Celery Worker is running and it is able to execute the geonode.security.tasks.synch_guardian periodic task; notice that the task will be executed at regular intervals, based on the interval value defined in the corresponding PeriodicTask model.

b. A periodic cron job runs the sync_security_rules management command, or either it is manually executed from the Django shell.

c. The user, owner of the Layer or with rights to change its permissions, clicks on the GeoNode UI button Sync permissions immediately
Warning: Layers won’t be accessible to public users anymore until the Security Rules are not synchronized!

**DISPLAY_COMMENTS**

Default: True  
Env: DISPLAY_COMMENTS  
If set to False comments are hidden.

**DISPLAY_RATINGS**

Default: True  
Env: DISPLAY_RATINGS  
If set to False ratings are hidden.

**DISPLAY_SOCIAL**

Default: True  
Env: DISPLAY_SOCIAL  
If set to False social sharing is hidden.

**DISPLAY_WMS_LINKS**

Default: True  
Env: DISPLAY_WMS_LINKS  
If set to False direct WMS link to GeoServer is hidden.

**DISPLAY_ORIGINAL_DATASET_LINK**

Default: True  
Env: DISPLAY_ORIGINAL_DATASET_LINK  
If set to False original dataset download is hidden.

**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:

```python
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:

```python
DOWNLOAD_FORMATS_RASTER = ['JPEG', 'PDF', 'PNG', 'Tiles',]
```

E

EMAIL_ENABLE

Default: False

Options:

- EMAIL_BACKEND
  Default: django.core.mail.backends.smtp.EmailBackend
  Env: DJANGO_EMAIL_BACKEND

- EMAIL_HOST
  Default: localhost

- EMAIL_PORT
  Default: 25

- EMAIL_HOST_USER
  Default: ''

- EMAIL_HOST_PASSWORD
  Default: ''

- EMAIL_USE_TLS
  Default: False

- EMAIL_USE_SSL
  Default: False

- DEFAULT_FROM_EMAIL
EPSG_CODE_MATCHES

Default:

```
{
  'EPSG:4326': '(4326) WGS 84',
  'EPSG:900913': '(900913) Google Maps Global Mercator',
  'EPSG:3857': '(3857) WGS 84 / Pseudo-Mercator',
  'EPSG:3785': '(3785 DEPRECATED) Popular Visualization CRS / Mercator',
  'EPSG:32647': '(32647) WGS 84 / UTM zone 47N',
  'EPSG:32736': '(32736) WGS 84 / UTM zone 36S'
}
```

Supported projections human readable descriptions associated to their EPSG Codes. This list will be presented to the user during the upload process whenever GeoNode won’t be able to recognize a suitable projection. Those codes should be aligned to the `UPLOADER` ones and available in GeoServer also.

EXTRA_METADATA_SCHEMA

Default:

```
EXTRA_METADATA_SCHEMA = {**{
  "map": os.getenv('MAP_EXTRA_METADATA_SCHEMA', DEFAULT_EXTRA_METADATA_SCHEMA),
  "layer": os.getenv('DATASET_EXTRA_METADATA_SCHEMA', DEFAULT_EXTRA_METADATA_SCHEMA),
  "document": os.getenv('DOCUMENT_EXTRA_METADATA_SCHEMA', DEFAULT_EXTRA_METADATA_SCHEMA),
  "geoapp": os.getenv('GEOAPP_EXTRA_METADATA_SCHEMA', DEFAULT_EXTRA_METADATA_SCHEMA)
}, **CUSTOM_METADATA_SCHEMA}
```

Variable used to actually get the expected metadata schema for each resource_type. In this way, each resource type can have a different metadata schema.

FREETEXT_KEYWORDS_READONLY

Default: False
Env: FREETEXT_KEYWORDS_READONLY

Make Free-Text Keywords writable from users. Or read-only when set to False.
FACET_PROVIDERS

Default: pre filled list of providers
Env: FACET_PROVIDERS
Contains the list of the providers available to perform an serve the facets. In case the user wants remove a facets, is enough to remove the path of the proider from the list

G

GEOFENCE_SECURITY_ENABLED

Default: True (False is Test is true)
Env: GEOFENCE_SECURITY_ENABLED
Whether the geofence security system is used.

GEOIP_PATH

Default: Path to project
Env: PROJECT_ROOT
The local path where GeoIPCities.dat is written to. Make sure your user has to have write permissions.

GEONODE_APPS_ENABLED

Default: True
If enabled contrib apps are used. If disabled: - the geoapps URLs are not included in the routing paths - the geoapps resources are excluded from the search - the resource detail are forwarded to the homepage
ENABLE -> DISABLE transition:
This should be done if the geoapps were enabled in an environment where they are not needed.
DISABLE -> ENABLE transition:
It should be done only once to enable geoapps in an environment where are needed

GEONODE_CLIENT_LAYER_PREVIEW_LIBRARY

Default: "mapstore"
The library to use for display preview images of layers. The library choices are:
"mapstore" "leaflet" "react"
**GEONODE_EXCHANGE**

Default: `Exchange("default", type="direct", durable=True)`

The definition of Exchanges published by geonode. Find more about Exchanges at celery docs.

**GEOSERVER_ADMIN_USER**

Default: `admin`

Env: `GEOSERVER_ADMIN_PASSWORD`

The geoserver admin username.

**GEOSERVER_ADMIN_PASSWORD**

Default: `geoserver`

Env: `GEOSERVER_ADMIN_USER`

The GeoServer admin password.

**GEOSERVER_FACTORY_PASSWORD**

Default: `geoserver`

Env: `GEOSERVER_FACTORY_PASSWORD`

The GeoServer admin factory password, required for the first time configuration fo Geoserver (Docker setup).

**GEOSERVER_EXCHANGE**

Default: `Exchange("geonode", type="topic", durable=False)`

The definition of Exchanges published by GeoServer. Find more about Exchanges at celery docs.

**GEOSERVER_LOCATION**

Default: `http://localhost:8080/geoserver/`

Env: `GEOSERVER_LOCATION`

Url under which GeoServer is available.
**GEOSERVER_PUBLIC_HOST**

Default: SITE_HOST_NAME (Variable)  
Env: GEOSERVER_PUBLIC_HOST  
Public hostname under which GeoServer is available.

**GEOSERVER_PUBLIC_LOCATION**

Default: SITE_HOST_NAME (Variable)  
Env: GEOSERVER_PUBLIC_LOCATION  
Public location under which GeoServer is available.

**GEOSERVER_PUBLIC_PORT**

Default: 8080 (Variable)  
Env: GEOSERVER_PUBLIC_PORT  
Public Port under which GeoServer is available.

**GEOSERVER_WEB_UI_LOCATION**

Default: GEOSERVER_PUBLIC_LOCATION (Variable)  
Env: GEOSERVER_WEB_UI_LOCATION  
Public location under which GeoServer is available.

**GROUP_PRIVATE_RESOURCES**

Default: False  
Env: GROUP_PRIVATE_RESOURCES  
If this option is enabled, Resources belonging to a Group won’t be visible by others.

**IMPORTER_HANDLERS**

Default: pre filled list of handlers  
Env: IMPORTER_HANDLERS  
Contains the list of the handlers available to perform an import of a resource. In case the user wants to drop the support during the import phase, is enough to remove the path of the Handler from the list.
LEAFLET_CONFIG

A dictionary used for Leaflet configuration.

LICENSES

Default:

```python
{
    'ENABLED': True,
    'DETAIL': 'above',
    'METADATA': 'verbose',
}
```

Enable Licenses User Interface

LOCAL_SIGNALS_BROKER_URL

Default: memory://

LOCKDOWN_GEONODE

Default: False
Env: LOCKDOWN_GEONODE

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.

LOGIN_URL

Default: `{}account/login/`.format(SITEURL)
Env: LOGIN_URL

The URL where requests are redirected for login.

LOGOUT_URL

Default: `{}account/login/`.format(SITEURL)
Env: LOGOUT_URL

The URL where requests are redirected for logout.
M

MAP_CLIENT_USE_CROSS_ORIGIN_CREDENTIALS

Default: False
Env: MAP_CLIENT_USE_CROSS_ORIGIN_CREDENTIALS

Enables cross origin requests for geonode-client.

MAPSTORE_BASELAYERS

Default:

```json
[
    {
        "type": "osm",
        "title": "Open Street Map",
        "name": "mapnik",
        "source": "osm",
        "group": "background",
        "visibility": true
    },
    {
        "type": "tileprovider",
        "title": "OpenTopoMap",
        "provider": "OpenTopoMap",
        "name": "OpenTopoMap",
        "source": "OpenTopoMap",
        "group": "background",
        "visibility": false
    },
    {
        "type": "wms",
        "title": "Sentinel-2 cloudless - https://s2maps.eu",
        "format": "image/jpeg",
        "id": "s2cloudless",
        "name": "s2cloudless:s2cloudless",
        "url": "https://maps.geo-solutions.it/geoserver/wms",
        "group": "background",
        "thumbURL": "%static/mapstorestyle/img/s2cloudless-s2cloudless.png" %SITEURL,
        "visibility": false
    },
    {
        "source": "ol",
        "group": "background",
        "id": "none",
        "name": "empty",
        "title": "Empty Background",
        "type": "empty",
        "visibility": false,
        "args": ["Empty Background", {"visibility": false}]
    }
]
```
Env: MAPSTORE_BASELAYERS

Allows to specify which backgrounds MapStore should use. The parameter visibility for a layer, specifies which one is the default one.

A sample configuration using the Bing background without OpenStreetMap, could be the following one:

```
[
    {
        "type": "bing",
        "title": "Bing Aerial",
        "name": "AerialWithLabels",
        "source": "bing",
        "group": "background",
        "apiKey": "{{apiKey}}",
        "visibility": true
    },
    {
        "type": "tileprovider",
        "title": "OpenTopoMap",
        "provider": "OpenTopoMap",
        "name": "OpenTopoMap",
        "source": "OpenTopoMap",
        "group": "background",
        "visibility": false
    },
    {
        "type": "wms",
        "title": "Sentinel-2 cloudless - https://s2maps.eu",
        "format": "image/jpeg",
        "id": "s2cloudless",
        "name": "s2cloudless:s2cloudless",
        "url": "https://maps.geo-solutions.it/geoserver/wms",
        "group": "background",
        "thumbURL": "%SITEURL%static/mapstorestyle/img/s2cloudless-s2cloudless.png" %
    },
    {
        "source": "ol",
        "group": "background",
        "id": "none",
        "name": "empty",
        "title": "Empty Background",
        "type": "empty",
        "visibility": false,
        "args": ["Empty Background", {"visibility": false}]
    }
]
```
**Warning:** To use a Bing background, you need to correctly set and provide a valid `BING_API_KEY`

**MAX_DOCUMENT_SIZE**

Default: 2

Env: `MAX_DOCUMENT_SIZE`

Allowed size for documents in MB.

**METADATA_PARSERS**

Is possible to define multiple XML parsers for ingest XML during the layer upload.

The variable should be declared in this way in `settings.py`:

```python
METADATA_PARSERS = ['list', 'of', 'parsing', 'functions']
```

If you want to always use the default metadata parser and after use your own, the variable must be set with first value as `__DEFAULT__`. Example:

```python
METADATA_PARSERS = ['__DEFAULT__', 'custom_parsing_function']
```

If not set, the system will use the `__DEFAULT__` parser.

The custom parsing function must be accept in input 6 parameter that are:

- `exml` (xmlfile)
- `uuid` (str)
- `vals` (dict)
- `regions` (list)
- `keywords` (list)
- `custom` (dict)

If you want to use your parser after the default one, here is how the variable are populated:

- `exml`: the XML file to parse
- `uuid`: the UUID of the layer
- `vals`: Dictionary of information that belong to ResourceBase
- `regions`: List of regions extracted from the XML
- `keywords`: List of dict of keywords already divided between free-text and thesaurus
- `custom`: Custom variable

NOTE: the keywords must be in a specific format, since later this dict, will be ingested by the `KeywordHandler` which will assign the keywords/thesaurus to the layer.

Here is an example of expected parser function

For more information, please rely to `TestCustomMetadataParser` which contain a smoke test to explain the functionality
METADATA_STORERS

Is possible to define multiple Layer storer during the layer upload.
The variable should be declared in this way:

METADATA_STORERS = ['custom_storer_function']

NOTE: By default the Layer is always saved with the default behaviour.
The custom storer function must be accept in input 2 parameter that are:
- Layer (layer model instance)
- custom (dict)

Here is how the variable are populated by default:
- layer (layer model instance) that we wanto to change
- custom: custom dict populated by the parser

Here is an example of expected storer function

For more information, please rely to TestMetadataStorers which contain a smoke test to explain the functionality

MISSING_THUMBNAIL

Default: geonode/img/missing_thumb.png

The path to an image used as thumbnail placeholder.

MEMCACHED_BACKEND

Default: django.core.cache.backends.memcached.PyMemcacheCache

Define which backend of memcached will be used

MEMCACHED_ENABLED

Default: False

If True, will use MEMCACHED_BACKEND as default backend in CACHES

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 customization is still possible changing the is_choice and the GeoNode description fields.

In case it is necessary to add/delete/update categories, it is possible to set the MODIFY_TOPICCATEGORY setting to True.
**MONITORING_ENABLED**

Default: False

Enable internal monitoring application (`geonode.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.monitoring',)
    MIDDLEWARE_CLASSES + ('geonode.monitoring.middleware.MonitoringMiddleware',)
```

See *Read-Only and Maintenance Mode* for details.

**MONITORING_DATA_AGGREGATION**

Default:

```
(  
    (timedelta(seconds=0), timedelta(minutes=1),),
    (timedelta(days=1), timedelta(minutes=60),),
    (timedelta(days=14), timedelta(days=1),),
  )
```

Configure aggregation of past data to control data resolution. It lists data age and aggregation in reverse order, by default:

- for current data, 1 minute resolution
- for data older than 1 day, 1-hour resolution
- for data older than 2 weeks, 1 day resolution

See *Read-Only and Maintenance Mode* for further details.

This setting takes effects only if `UUID HANDLER` is true.

**MONITORING_DATA_TTL**

Default: 365

Env: `MONITORING_DATA_TTL`

How long monitoring data should be stored in days.
**MONITORING_DISABLE_CSRF**

Default: False  
Env: MONITORING_DISABLE_CSRF  
Set this to true to disable csrf check for notification config views, use with caution - for dev purpose only.

**MONITORING_SKIP_PATHS**

Default:

```python
(
    '/api/o/',
    '/monitoring/',
    '/admin',
    '/jsi18n',
    STATIC_URL,
    MEDIA_URL,
    re.compile('^/[a-z]{2}/admin/'),
)
```

Skip certain useless paths to not to mud analytics stats too much. See *Read-Only and Maintenance Mode* to learn more about it.  
This setting takes effects only if *UUID HANDLER* is true.

**NOTIFICATIONS_MODULE**

Default: pinax.notifications  
App used for notifications. (pinax.notifications or notification)

**NOTIFICATION_ENABLED**

Default: True  
Env: NOTIFICATION_ENABLED  
Enable or disable the notification system.

**OAUTH2_API_KEY**

Default: None  
Env: OAUTH2_API_KEY  
In order to protect oauth2 REST endpoints, used by GeoServer to fetch user roles and infos, you should set this key and configure the *geonode REST role service* accordingly. Keep it secret!
Warning: If not set, the endpoint can be accessed by users without authorization.

**OAUTH2_PROVIDER**

Ref.: OAuth Toolkit settings

**OAUTH2_PROVIDER_APPLICATION_MODEL**

Default: oauth2_provider.Application
Ref.: OAuth Toolkit settings

**OAUTH2_PROVIDER_ACCESS_TOKEN_MODEL**

Default: oauth2_provider.AccessToken
Ref.: OAuth Toolkit settings

**OAUTH2_PROVIDER_ID_TOKEN_MODEL**

Default: oauth2_provider.IDToken
Ref.: OAuth Toolkit settings

**OAUTH2_PROVIDER_GRANT_MODEL**

Default: oauth2_provider.Grant
Ref.: OAuth Toolkit settings

**OAUTH2_PROVIDER_REFRESH_TOKEN_MODEL**

Default: oauth2_provider.RefreshToken
Ref.: OAuth Toolkit settings

**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:

```
oauth2_provider = {
    'default': {
        'LOCATION': 'http://localhost:8080/geoserver/',
        'USER': 'admin',
    },
```

(continues on next page)
• 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, where Geonode uploads
  are imported into. To support vector datastore imports there also needs to be an entry for
  the datastore in the DATABASES dictionary with the same name. Example:

  ```python
  OGC_SERVER = {
    'default': {
      'LOCATION': 'http://localhost:8080/geoserver/',
      'USER': 'admin',
      'PASSWORD': 'geoserver',
      'DATASTORE': 'geonode_imports'
    }
  }
  ```

  ```python
  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',
    }
  }
  ```

• 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 setting 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.
**OGP_URL**

Default: `http://geodata.tufts.edu/solr/select`
Env: OGP_URL

Endpoint of geodata.tufts.edu getCapabilities.

**OPENGRAPH_ENABLED**

Default: `True`

A boolean that specifies whether Open Graph is enabled. Open Graph is used by Facebook and Slack.

**PINAX_NOTIFICATIONS_BACKENDS**

Default: `("email", _EMAIL_BACKEND, 0)`,

Used notification backend. This is a pinax notification setting:

**PINAX_NOTIFICATIONS_LOCK_WAIT_TIMEOUT**

Default: `-1`
Env: NOTIFICATIONS_LOCK_WAIT_TIMEOUT

It defines how long to wait for the lock to become available. Default of -1 means to never wait for the lock to become available. This is a pinax notification setting:

**PINAX_NOTIFICATIONS_QUEUE_ALL**

Default: `-1`
Env: NOTIFICATIONS_LOCK_WAIT_TIMEOUT

By default, calling notification.send will send the notification immediately, however, if you set this setting to True, then the default behavior of the send method will be to queue messages in the database for sending via the emit_notices command. This is a pinax notification setting:

**PINAX_RATINGS_CATEGORY_CHOICES**

Default:

```python
{
    "maps.Map": {
        "map": "How good is this map?"
    },
    "layers.Layer": {
        "layer": "How good is this layer?"
    },
    "documents.Document": {
```

(continues on next page)
PROFILE_EDIT_EXCLUDE_FIELD

Default: []

A list of element (item name) to exclude from the Profile Edit page.

Example:

PROFILE_EDIT_EXCLUDE_FIELD=['organization', 'language']

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.

PYCSW

A dict with pycsw’s configuration with two possible keys CONFIGURATION and FILTER.

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.

FILTER Optional settings in order to add a filter to the CSW filtering. The filter follow the django orm structure and must be a ResourceBase field/related field. By default CSW will filter only for layer resource_type

Example of PYCSW configuration. PYCSW: {

    ‘CONFIGURATION’: {...}, ‘FILTER’: ‘resource_type__in’:[‘layer’] }

}
RABBITMQ_SIGNALS_BROKER_URL

Default: amqp://localhost:5672

The Rabbitmq endpoint

RECAPTCHA_ENABLED

Default: False
Env: RECAPTCHA_ENABLED

Allows enabling reCaptcha field on signup form. Valid Captcha Public and Private keys will be needed as specified here https://pypi.org/project/django-recaptcha/#installation

You will need to generate a keys pair for reCaptcha v2 for your domain from https://www.google.com/recaptcha/admin/create

More options will be available by enabling this setting:

• ACCOUNT_SIGNUP_FORM_CLASS
  Default: geonode.people.forms.AllauthReCaptchaSignupForm
  Env: ACCOUNT_SIGNUP_FORM_CLASS
  Enabled only when the RECAPTCHA_ENABLED option is True.

• INSTALLED_APPS
  The captcha must be present on INSTALLED_APPS, otherwise you’ll get an error.

  When enabling the RECAPTCHA_ENABLED option through the environment, this setting will be automatically added by GeoNode as follows:

  ```python
  if 'captcha' not in INSTALLED_APPS:
      INSTALLED_APPS += ('captcha',)
  ```

• RECAPTCHA_PUBLIC_KEY
  Default: geonode_RECAPTCHA_PUBLIC_KEY
  Env: RECAPTCHA_PUBLIC_KEY
  You will need to generate a keys pair for reCaptcha v2 for your domain from https://www.google.com/recaptcha/admin/create

  For mode details on the reCaptcha package, please see:

  1. https://pypi.org/project/django-recaptcha/#installation

• RECAPTCHA_PRIVATE_KEY
  Default: geonode_RECAPTCHA_PRIVATE_KEY
  Env: RECAPTCHA_PRIVATE_KEY
  You will need to generate a keys pair for reCaptcha v2 for your domain from https://www.google.com/recaptcha/admin/create

  For mode details on the reCaptcha package, please see:

1.13. GeoNode Settings
RECAPTCHA_PUBLIC_KEY

Default: geonode_RECAPTCHA_PUBLIC_KEY
Env: RECAPTCHA_PUBLIC_KEY

You will need to generate a keys pair for reCaptcha v2 for your domain from https://www.google.com/recaptcha/admin/create

Ref. to RECAPTCHA_ENABLED

RECAPTCHA_PRIVATE_KEY

Default: geonode_RECAPTCHA_PRIVATE_KEY
Env: RECAPTCHA_PRIVATE_KEY

You will need to generate a keys pair for reCaptcha v2 for your domain from https://www.google.com/recaptcha/admin/create

Ref. to RECAPTCHA_ENABLED

REDIS_SIGNALS_BROKER_URL

Default: redis://localhost:6379/0

The Redis endpoint.

REGISTERED_MEMBERS_GROUP_NAME

Default: registered-members
Env: REGISTERED_MEMBERS_GROUP_NAME

Used by AUTO_ASSIGN_REGISTERED_MEMBERS_TO_REGISTERED_MEMBERS_GROUP_NAME settings.

REGISTERED_MEMBERS_GROUP_TITLE

Default: Registered Members
Env: REGISTERED_MEMBERS_GROUP_TITLE

Used by AUTO_ASSIGN_REGISTERED_MEMBERS_TO_REGISTERED_MEMBERS_GROUP_NAME settings.
REGISTRATION_OPEN

Default: False

A boolean that specifies whether users can self-register for an account on your site.

RESOURCE_PUBLISHING

Default: False

By default, the GeoNode application allows GeoNode staff members to publish/unpublish resources. By
default, resources are published when created. When this setting is set to True the staff members will be
able to unpublish a resource (and eventually publish it back).

SEARCH_FILTERS

Default:

```
'TEXT_ENABLED': True,
'TYPE_ENABLED': True,
'CATEGORIES_ENABLED': True,
'OWNERS_ENABLED': True,
'KEYWORDS_ENABLED': True,
'H_KEYWORDS_ENABLED': True,
'T_KEYWORDS_ENABLED': True,
'DATE_ENABLED': True,
'REGION_ENABLED': True,
'EXTENT_ENABLED': True,
```

Enabled Search Filters for filtering resources.

SECURE_BROWSER_XSS_FILTER

Default: True

Env: SECURE_BROWSER_XSS_FILTER

If True, the SecurityMiddleware sets the X-XSS-Protection: 1; mode=block header on all responses
that do not already have it. This is Django settings. https://docs.djangoproject.com/en/3.2/ref/settings/
#secure-browser-xss-filter
SECURE_CONTENT_TYPE_NOSNIFF

Default: True
Env: SECURE_CONTENT_TYPE_NOSNIFF

If True, the SecurityMiddleware sets the X-Content-Type-Options: nosniff header on all responses that do not already have it. This is Django settings:

SECURE_HSTS_INCLUDE_SUBDOMAINS

Default: True
Env: SECURE_HSTS_INCLUDE_SUBDOMAINS

This is Django settings: https://docs.djangoproject.com/en/3.2/ref/settings/#secure-hsts-include-subdomains

SECURE_HSTS_SECONDS

Default: 3600
Env: SECURE_HSTS_SECONDS

This is Django settings: If set to a non-zero integer value, the SecurityMiddleware sets the HTTP Strict Transport Security header on all responses that do not already have it.

SECURE_SSL_REDIRECT

If True, the SecurityMiddleware redirects all non-HTTPS requests to HTTPS (except for those URLs matching a regular expression listed in SECURE_REDIRECT_EXEMPT). This is Django settings:

SERVICES_TYPE_MODULES

It’s possible to define multiple Service Types Modules for custom service type with it’s own Handler.

The variable should be declared in this way in settings.py:

`SERVICES_TYPE_MODULES = [ 'path.to.module1','path.to.module2', ... ]`

Default service types are already included

Inside each module in the list we need to define a variable:

`services_type = {`

```
"<key_of_service_type>": {
  "OWS": True/False, "handler": "<path.to.Handler>", "label": "<label to show in remote service page>",
  "management_view": "<path.to.view>"
}
```

the key_of_service_type is just an identifier to assign at the service type. OWS is True if the service type is an OGC Service Compliant. The handler key must contain the path to the class who will provide all methods to manage the service type The label is what is shown in the service form when adding a new service. The management_view, if exists, must contain the path to the method where the management page is opened.
**SERVICE_UPDATE_INTERVAL**

Default: 0

The Interval services are updated.

**SESSION_COOKIE_SECURE**

Default: False

Env: SESSION_COOKIE_SECURE

This is a Django setting:

**SESSION_EXPIRED_CONTROL_ENABLED**

Default: True

Env: SESSION_EXPIRED_CONTROL_ENABLED

By enabling this variable, a new middleware `geonode.security.middleware.SessionControlMiddleware` will be added to the `MIDDLEWARE_CLASSES`. The class will check every request to GeoNode and it will force a log out whenever one of the following conditions occurs:

1. The OAuth2 Access Token is not valid anymore or it is expired.

   Warning: The Access Token might be invalid for various reasons. Usually a misconfiguration of the OAuth2 GeoServer application. The latter is typically installed and configured automatically at GeoNode bootstrap through the default fixtures.

2. The user has been deactivated for some reason; an Admin has disabled it or its password has expired.

Whenever the middleware terminates the session and the user forced to log out, a message will appear to the GeoNode interface.

**SHOW_PROFILE_EMAIL**

Default: False

A boolean which specifies whether to display the email in the user’s profile.

**SITE_HOST_NAME**

Default: localhost

Env: SITE_HOST_NAME

The hostname used for GeoNode.
**SITE_HOST_PORT**

Default: `8000`

Env: SITE_HOST_PORT

The Site hostport.

**SITEURL**

Default: `/http://localhost:8000/`

A base URL for use in creating absolute links to Django views and generating links in metadata.

**SIZE_RESTRICTED_FILE_UPLOAD_ELEGIBLE_URL_NAMES**

Default: `"("data_upload", "uploads-upload", "document_upload",),")`

Rappresent the list of the urls basename that are under file_size restriction

**SKIP_PERMS_FILTER**

Default: `False`

Env: SKIP_PERMS_FILTER

If set to true permissions prefiltering is avoided.

**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_AUTO_SIGNUP**

Default: `True`

Attempt to bypass the signup form by using fields (e.g. username, email) retrieved from the social account provider. This is a Django-allauth setting:

**SOCIALACCOUNT_PROVIDERS**

Default:

```python
{
    'linkedin_oauth2': {
        'SCOPE': [
            'r_emailaddress',
            'r_basicprofile',
        ],
    }
}
```
This is a django-allauth setting It should be a dictionary with provider specific settings

**SOCIALACCOUNT_PROFILE_EXTRACTORS**

Default:

```python
{,
    "facebook": "geonode.people.profileextractors.FacebookExtractor",
    "linkedin_oauth2": "geonode.people.profileextractors.LinkedInExtractor",
}
```

A dictionary with provider ids as keys and path to custom profile extractor classes as values.
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}"
}, {
    "label": "Facebook",
}, {
    "label": "Twitter",
    "url": "https://twitter.com/share?url={url}"
}, {
    "label": "Google +",
    "url": "https://plus.google.com/share?url={url}"
}]
```

A list of dictionaries that are 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).

SOCIALACCOUNT_WITH_GEONODE_LOCAL_SINGUP

Default: True
Variable which controls displaying local account registration form. By default form is visible.

SRID

Default:

```
{
    'DETAIL': 'never',
}
```
SEARCH_RESOURCES_EXTENDED

Default: True

This will extend search with additional properties. By default, its on and search engine will check resource title or purpose or abstract. When set to False just title lookup is performed.

SUPPORTED_DATASET_FILETYPES

Default:

```python
SUPPORTED_DATASET_FILE_TYPES = [
    {
        "id": "shp", "label": "ESRI Shapefile", "format": "vector", "ext": ["shp"], "requires": ["shp", "prj", "dbf", "shx"], "optional": ["xml", "sld"]
    },
    {
        "id": "tiff", "label": "GeoTIFF", "format": "raster", "ext": ["tiff", "tif"], "mimeType": ["image/tiff"], "optional": ["xml", "sld"]
    },
    {
        "id": "csv", "label": "Comma Separated Value (CSV)", "format": "vector", "ext": ["csv"], "mimeType": ["text/csv"], "optional": ["xml", "sld"]
    },
    {
        "id": "zip", "label": "Zip Archive", "format": "archive", "ext": ["zip"], "mimeType": ["application/zip"], "optional": ["xml", "sld"]
    },
    {
        "id": "xml", "label": "XML Metadata File", "format": "metadata", "ext": ["xml"], "mimeType": ["application/json"], "needsFiles": ["shp", "prj", "dbf", "shx", "csv", "tiff", "zip", "sld"]
    },
    {
        "id": "sld", "label": "Styled Layer Descriptor (SLD)", "format": "metadata", "ext": ["sld"], "mimeType": ["application/json"], "needsFiles": ["shp", "prj", "dbf", "shx", "csv", "tiff", "zip", "xml"]
    }
]
```

Represent the list of the supported file type in Geonode that can be ingested by the platform.

For example, the following configuration is needed to add the GeoJSON as supported file:

Default:

```python
{
    "id": "geojson", "label": "GeoJSON", "format": "metadata", "ext": ["geojson"], "mimeType": ["application/json"]
}
```
T

TASTYPIE_DEFAULT_FORMATS

Default: json
This setting allows you to globally configure the list of allowed serialization formats for your entire site. This is a tastypie setting:

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.

THESAURI

Default = []
A list of Keywords thesauri settings: For example THESAURI = [{'name':'inspire_themes', 'required':True, 'filter':True}, {'name':'inspire_concepts', 'filter':True}, ]

TOPICCATEGORY_MANDATORY

Default: False
Env: TOPICCATEGORY_MANDATORY
If this option is enabled, Topic Categories will become strictly Mandatory on Metadata Wizard

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.

TINYMCE_DEFAULT_CONFIG

Default:

```json
{
    "selector": "textarea#id_resource-featureinfo_custom_template",
    "theme": "silver",
    "height": 500,
    "plugins": 'print preview paste importcss searchreplace autolink autosave_
          save directionality code visualblocks visualchars fullscreen image link._
          media template codesample table charmap hr pagebreak nonbreaking anchor toc_
          insertdatetime advlist lists wordcount imagetools textpattern noneditable_
          help charmap quickbars emoticons',
    "imagetools_cors_hosts": ['picsum.photos'],
    "menubar": 'file edit view insert format tools table help',
    "toolbar": 'undo redo | bold italic underline strikethrough | fontselect_
          fontsize select formatselect | alignleft aligncenter alignright alignjustify_
          | outdent indent | numlist bullist | forecolor backcolor removeformat |_
          pagebreak | charmap emoticons | full screen preview save | insert file image_
          media template link anchor codesample | ltr rtl',
    "toolbar_sticky": "true",
    "autosave_ask_before_unload": "true",
    "autosave_interval": "30s",
    "autosave_prefix": "{path}{query}-{id}-",
    "autosave_restore_when_empty": "false",
    "autosave_retention": "2m",
    "image_advtab": "true",
    "content_css": "//www.tiny.cloud/css/codepen.min.css",
    "importcss_append": "true",
    "image_caption": "true",
    "quickbars_selection_toolbar": 'bold italic | quicklink h2 h3 blockquote_
          quick image quicktable',
    "noneditable_noneditable_class": "mceNonEditable",
    "toolbar_mode": 'sliding',
    "contextmenu": "link image imagetools table",
    "templates": [
        {
            "title": 'New Table',
            "description": 'creates a new table',
            "content": '<div class="mceTmpl"><table width="98%" border="0" _
          cellspacing="0" cellpadding="0"><tr><th scope="col"> </th>
          <th scope="col"> </th></tr><tr><td> </td><td> </td></tr></table></div>
        }
    ],
    {
        "title": 'Starting my story',
        "description": 'A cure for writers block',
        "content": 'Once upon a time...
        (continues on next page)'
    }
}
```

(continues on next page)
HTML WYSIWYG Editor (TINYMCE) Menu Bar Settings. For more info see:

- https://django-tinymce.readthedocs.io/en/latest/installation.html#configuration
- Customizing The Datasets’ GetFeatureInfo Templates

UI

UI_REQUIRED_FIELDS

If this option is enabled, the input selected (we are referring to the one present in the optional Metadata-Tab on the Metadata-Wizard) will become mandatory.

The fields that can be mandatory are:

- id_resource-edition => Label: Edition
- id_resource-purpose => Label: Purpose
- id_resource-supplemental_information => Label: Supplemental information
- id_resource-temporal_extent_start_pickers => Label: temporal extent start
- id_resource-temporal_extent_end => Label: temporal extent end
- id_resource-maintenance_frequency => Label: Maintenance frequency
- id_resource-spatial_representation_type => Label: Spatial representation type

If at least one on the above ids is set in this configuration, the panel header will change from Optional to Mandatory

Configuration Example:

UI_REQUIRED_FIELDS = [‘id_resource-edition’]

UNOCONV_ENABLE

Default: False
Env: UNOCONV_ENABLE
**Uploader**

Default:

```python
{
    'BACKEND': 'geonode.importer',
    'OPTIONS': {
        'TIME_ENABLED': False,
    }
}
```

A dictionary of Uploader settings and their values.

- **BACKEND**
  
  Default: 'geonode.importer'

  The importer backend requires the GeoServer importer extension to be enabled.

- **OPTIONS**
  
  Default:

  ```python
  {'OPTIONS': {
      'TIME_ENABLED': False,
  }}
  ```

  - **TIME_ENABLED**
    
    Default: False

    A boolean that specifies whether the upload should allow the user to enable time support when uploading data.

**USER_MESSAGES_ALLOW_MULTIPLE_RECIPIENTS**

Default: True

Env: USER_MESSAGES_ALLOW_MULTIPLE_RECIPIENTS

Set to true to have multiple recipients in /message/create/

**USER_DELETION_RULES**

Default: ["geonode.people.utils.user_has_resources"]

Env: USER_DELETION_RULES

List of callables that will be called the deletion of a user account is requested. The deletion will fail if any of the callables return False. New rules can be added, as a string path to the callable, as long as they take as parameter the user object and return a boolean.
UUID HANDLER

Is possible to define an own uuidhandler for the Layer.
To start using your own handler, is needed to add the following configuration:

```
LAYER_UUID_HANDLER = "mymodule.myfile.MyObject"
```

The Object must accept as init the instance of the layer and have a method named create_uuid()

here is an example:

```python
class MyObject:
    def __init__(self, instance):
        self.instance = instance

    def create_uuid(self):
        # here your code
        pass
```

X

X_FRAME_OPTIONS

Default: 'ALLOW-FROM %s' % SITEURL
This is a Django setting

1.14 Docker Environmental variables

Docker Environmental variables

1.14.1 Docker Environmental variables

This list reports the specific environmental variables requiried for the configuration and execution of GeoNode Docker services

Geoserver

DATABASE_HOST

Default: db
Internal IP or hostname for the Postgresql (db) Docker container.

DATABASE_PORT

Default: django
Internal port for the Postgresql (db) Docker container.
**GEONODE\_LB\_HOST\_IP**

Default: `django`

Internal IP or hostname for the GeoNode (django) Docker container.

**GEONODE\_LB\_PORT**

Default: `8000`

Internal port for the GeoNode (django) Docker container.

**GEONODE\_GEODATABASE**

Default: `geonode\_data`

GeoNode geodatabase name.

**GEONODE\_GEODATABASE\_USER**

Default: `geonode\_data`

GeoNode geodatabase user.

**GEONODE\_GEODATABASE\_PASSWORD**

Default: `geonode\_data`

GeoNode geodatabase password.

**GEONODE\_GEODATABASE\_SCHEMA**

Default: `publix`

GeoNode geodatabase schema.

**GEOSERVER\_JAVA\_OPTS**

Default: '

- `Djava.awt.headless=true`  
- `Xms4G`  
- `Xmx4G`  
- `Dgwc.context.suffix=gwc`  
- `XX:+UnlockDiagnosticVMOptions`  
- `XX:+LogVMOutput`  
- `XX:LogFile=/var/log/jvm.log`  
- `XX:PerfDataSamplingInterval=500`  
- `XX:SoftRefLRUPolicyMSPerMB=36000`  
- `XX:+UseGCOverheadLimit`  
- `XX:ParallelGCThreads=4`  
- `Dfile.encoding=UTF8`  
- `Djavax.servlet.request.encoding=UTF-8`  
- `Djavax.servlet.response.encoding=UTF-8`  
- `Duser.timezone=GMT`  
- `Dorg.geotools.shapefile.datetime=false`  
- `DGS-SHAPEFILE-CHARSET=UTF-8`  
- `DGEOSERVER_CSRF_DISABLED=true`  
- `DPRINT\_BASE\_URL={geoserver\_ui}/geoserver/pdf`  
- `DALLOW\_ENV\_PARAMETRIZATION=true`  
- `Xbootclasspath/a:/usr/local/tomcat/webapps/geoserver/WEB-INF/lib/marlin-0.9.3-Unsafe.jar`  
- `Dsun.java2d.renderer=org.marlin.pisces.MarlinRenderingEngine`

Geoserver JAVA OPTS.

**GEOSERVER\_LB\_HOST\_IP**

Default: `geoserver`

Internal IP or hostname for the GeoNode (geoserver) Docker container.
**GEOSERVER_LB_PORT**

Default: 8080

Internal port for the GeoNode (geoserver) Docker container.

**NGINX_BASE_URL**

Default: http(s):/localhost

Public URL of the nginx service. This value must match GeoNodes's SITEURL setting.

**OAUTH2_API_KEY**

Default: (empty)

Optional API Key for the communication between Geoserver and GeoNode Oauth2 services.

**OAUTH2_CLIENT_ID**

Default: (empty)

Geoserer Client ID for GeoNode Oauth2 server. Cannot be empty.

**OAUTH2_CLIENT_SECRET**

Default: (empty)

Geoserer Client secret for GeoNode Oauth2 server. Cannot be empty.

**Nginx**

**GEONODE_LB_HOST_IP**

Default: django

Internal IP or hostname for the GeoNode (django) Docker container.

**GEONODE_LB_PORT**

Default: 8000

Internal port for the GeoNode (django) Docker container.

**GEOSERVER_LB_HOST_IP**

Default: geoserver

Internal IP or hostname for the GeoNode (geoserver) Docker container.

**GEOSERVER_LB_PORT**

Default: 8080

Internal port for the GeoNode (geoserver) Docker container.
HTTPS_HOST

Default: localhost
The public hostname for HTTPS (without http:/ scheme prefix).

HTTPS_HOST

Default: localhost
The public hostname for HTTPS (without https:/ scheme prefix).

HTTP_PORT

Default: 80
Host port on which to expose the Nginx service for HTTP.

HTTPS_PORT

Default: 443
Host port on which to expose the Nginx service for HTTPS.

LETCENCRYPT_MODE

Default: disabled
Possible values: disabled (SSL disabled), staging (SSL local cert), production (SSL released by an ACME server)

POSTGRESQL / PostGIS

GEONODE_DATABASE

Default: geonode
GeoNode database name.

GEONODE_DATABASE_USER

Default: geonode_data
GeoNode database user.

GEONODE_DATABASE_PASSWORD

Default: geonode_data
GeoNode database password.

GEONODE_GEODATABASE

Default: geonode_data
GeoNode geodatabase name.
1.15 Customize the Look and Feel

1.15.1 GeoNode Themes

We have already explained in Simple Theming how to change the GeoNode theme directly from the Admin Interface. This is an easy way for customizing GeoNode appearance but, in some cases, you might want to have more control on it.

In those cases, you have to venture into the code and it is highly recommended to use a GeoNode Project and customize it instead of the GeoNode default HTML/CSS code. See the following sections to learn more about that.

1.15.2 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.

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:
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:

```
$ wget https://upload.wikimedia.org/wikipedia/commons/thumb/a/ac/Service_mark.svg/500px-Service_mark.svg.png
$ wget https://upload.wikimedia.org/wikipedia/commons/thumb/c/c8/Wikimapia_logo_without_label.svg/426px-Wikimapia_logo_without_label.svg.png -O logo.png
```

3. Create snippets directory:

```
$ cd ..;/..;
$ mkdir <my_geonode>/templates/geonode-mapstore-client/snippets
$ cd <my_geonode>/templates/geonode-mapstore-client/snippets
```

4. Create a new HTML file named `brand_navbar.html`

```
$ sudo vi brand_navbar.html
```

```
{% extends "geonode-mapstore-client/snippets/brand_navbar.html" %}
{% load static %}
{% block extra_style %}
<style>
#gn-brand-navbar {
  background: transparent url("/static/img/500px-Service_mark.svg.png") no-repeat;
  background-size: 300px 70px;
  background-position: left center;
  background-position-x: 100px;
}
</style>
{% endblock %}
{% block logo_src %}
{% static 'img/logo.png' %}
{% endblock %}
```

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/](http://localhost/) or the remote URL for your site.

In the following sections you will learn how to customize this header to make it as you want.
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/brand_navbar.html` in your editor:

   ```bash
   $ cd <my_geonode>/templates/geonode-mapstore-client/snippets
   $ sudo vi brand_navbar.html
   ```

1. Append a rule to change the background color of the header to match the logo graphic:

   ```css
   #gn-brand-navbar {
   ....
   background-color: #ff0000 !important;
   }
   ```

1. Create new file to manipulate hero section:

   ```bash
   $ cd <my_geonode>/templates/geonode-mapstore-client/snippets
   $ sudo vi hero.html
   ```

1. Add the following code to change the background image and font for the hero section:

   ```html
   {% extends "geonode-mapstore-client/snippets/hero.html" %}
   {% block extra_style %}
   <style>
   #gn-hero {
   background-image: url('https://cdn.pixabay.com/photo/2017/09/16/16/09/151.png');
   }
   </style>
   ```

(continues on next page)
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.

**Note:** 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.
Modify GeoNode Homepage

So far we learned how to modify some template sections of your GeoNode main page. You can do it individually per section template, adding a new page under `<my_geonode>/templates/geonode-mapstore-client/snippets` folder with the section name (ex: `brand_navbar.html`) or by extending the base template file `custom_theme.html` where you can add different theme settings in one place.

1. Remove the previous `hero` section `hero.html` file:

```
$ rm <my_geonode>/templates/geonode-mapstore-client/snippets/hero.html
```

1. Create a new `custom_theme.html` file:

```
$ cd <my_geonode>/templates/geonode-mapstore-client/snippets
$ sudo vi custom_theme.html
```

1. Add the following content to this page:

```html
{% load static %}
{% block content %}
<style>
  .msgapi .gn-theme {
    --gn-primary: #df7656;
    --gn-primary-contrast: #e3dcdc;
    --gn-link-color: #fcd823;
    --gn-focus-color: rgba(57, 122, 171, 0.4);
    --gn-footer-bg: #dbb051;
  }
</style>
```

(continues on next page)
1. Restart httpd server

```
$ python manage.py collectstatic
$ sudo service apache2 restart
```

1. Your customized layout should be similar to the next picture:

1. Edit title and intro message

Login as administrator on GeoNode and go to Admin page:

Create a new theme under GeoNode Themes Library and Themes:

Add a Name, Description and turn on Is enabled option. At the bottom, add a Jumbotron title and Jumbotron content. This will override the default GeoNode welcome title and message. Click Save at the bottom in the end.

After this, reload your GeoNode homepage. The output should be similar to this:
Welcome to GeoNode

Profile
Recent activity
Favorites
Inbox

Admin
GeoServer

Monitoring & Analytics
Help
Log out

GeoNode Themes Library

Jumbotron theme slides
Themes
1.15. Customize the Look and Feel

GeoNode provides a Jumbotron for customizing the look and feel of the website. To add a Jumbotron, navigate to the Customize the Look and Feel section.

### Jumbotron Configuration

- **Name**: my test theme
  - This will not appear anywhere.

- **Description**: My first test theme
  - This will not appear anywhere.
  - **is enabled**: Yes
  - Enabling this theme will disable the current enabled theme (if any).

- **Jumbotron title**: My GeoNode

- **Jumbotron content**: my first GeoNode customization

---

In the example provided, the GeoNode website is shown with a custom Jumbotron displaying "My GeoNode" and its customization."Welcome to GeoNode" is displayed prominently below the Jumbotron.
1.16 GeoNode permissions

1.16.1 Permissions

Permissions in GeoNode are set per resource, where a resource can be a dataset, a map, a document, a service or a geoapp. The way the permissions are set is the same for all of them.

**Warning:** GeoNode has a set of default permissions that are applied on resource creation when you don’t explicitly declare them. This is particularly relevant when creating and saving a map, where you won’t have the possibility to set its permissions during the creation phase. GeoNode can be tuned to make sure that by default the new created resource are not public, this can be done by changing two settings, see Default view permissions and Default download permissions.

Single Resource permissions

Resource permissions can be generally set from the resource detail page. The detail page has a menu item Share which is visible to people who are permitted to set permissions on a resource.

The share link opens a page on the right with a provision to edit user and group permissions on the resource. see picture below

![Change Dataset Permissions](image)

**Fig. 229: Change Dataset Permissions**

The page for setting the permissions, allows addition of users/groups and selection of a permission to assign each of them.
You can set the following types of permissions:

- **View**: allows to view the resource;
- **Download**: allows to download the resource;
- **Edit**: allows to change attributes, properties of the datasets features, styles and metadata for the specified resource;
- **Manage**: allows to update, delete, change permissions, publish and unpublish the resource.

**Warning**: When assigning permissions to a group, all the group members will have those permissions. Be careful in case of editing permissions.

**Geo Limits permissions**

**Note**: This feature is available **only** when enabling `GeoServer` as geospatial backend. Also make sure that the properties `GEONODE_SECURITY_ENABLED`, `GEOFENCE_SECURITY_ENABLED` and `GEOFENCE_URL` are correctly set for the `OGC_SERVER`.

**Geo Limits** are an extension of the GeoNode standard permissions. **Geo Limits** allows the owner of the resource, or the administrator, to restrict users or groups to a specific geographical area, in order to limit the access to the dataset to only the portions contained within that geographic restriction, excluding data outside of it.

In order to be able to set **Geo Limits** you must be an `administrator` of the system or the `owner` of the resource or you must have `Manage Permissions` rights to the resource.

If you have the permissions to set the **Geo Limits**, you should be able to see the permissions section and the globe icon on each user or group.

You should be able to see an interactive preview of the resource along with few small drawing tools, that allows you to start creating limits on the map manually if you want.

This opens a map dialog, with 3 options at the top:

- The pencil icon allows you to draw limits on a map for which a user will be able to see. Click on it to start drawing on the map. Once you are done drawing, click on it again to deactivate drawing mode.
- The eraser icon enables you to remove the limits you have drawn. Click on the limit drawn, and then click the delete icon.
- The refresh icon removes all changes that are not saved.
Fig. 231: Geo Limits Icon

Fig. 232: Geo Limits: Preview Window with Drawing Tools
Once you finished editing your geometries, save them into the DB using the *Save* link in the resource menu. The user with the specified geometries won’t be able from now on to access the whole dataset data.

**Warning:** The *Geo Limits* will be persisted on GeoNode DB for that resource. That means that everytime you will update the general permissions, also the geospatial restrictions will be applied.

In order to remove the *Geo Limits* for a certain user or group, you can just *Save* an **empty geometry**. This will **delete** the entry from the DB also.

### 1.17 Read-Only and Maintenance Mode

#### 1.17.1 Read-Only and Maintenance Modes

**Overview**

GeoNode gives an option to operate in different modes, according to the needs and demands of the certain application system.

Changing the currently used mode can be done in the admin panel by the user with super-user privileges, by modifying **Configuration** singleton model in the **BASE** application:
Fig. 234: Geo Limits: Geospatial restrictions applies for the user

Fig. 235: Configuration change in the admin panel
Read-Only Mode

Activating the Read-Only Mode (by setting Read only True in the Configuration) activates a middleware rejecting all modifying requests (POST/PUT/DELETE), with an exception for:

- POST to login view
- POST to logout view
- POST to admin login view
- POST to admin logout view
- all requests to OWS endpoint
- all requests ordered by a super-user

Additionally, all UI elements allowing modifying GeoNode's content are hidden, so e.g. the button “Upload Layer” is not rendered in the templates.

In case a user tries to perform a forbidden request, they will be presented with a static HTML page informing them, the GeoNode is in the Read-Only mode and this action is currently forbidden.

Maintenance Mode

Activating the Maintenance Mode (by setting Maintenance True in the Configuration) activates the highest level middleware (the one executed as the first) rejecting all requests to the GeoNode instance, with an exception for:

- POST to admin login view
- POST to admin logout view
- all requests ordered by a super-user

In case a user tries to perform any request against the GeoNode (including GET requests), they will be presented with a static HTML page informing them, the maintenance actions are taken on the GeoNode instance, and asking them to try again soon.

The maintenance mode was implemented with a thought of the backup and restore procedures without a necessity to put down the instance, but at the same time with a restriction of any outer interference.

1.18 Harvesting resources from remote services

1.18.1 Harvesting resources from remote services

GeoNode is able to harvest resource metadata from multiple remote services.

Harvesting is the process by which a metadata catalogue, i.e. GeoNode, is able to connect to other remote catalogues and retrieve information about their resources. This process is usually performed periodically, in order to keep the local catalogue in sync with the remote.

When appropriately configured, GeoNode will contact the remote service, extract a list of relevant resources that can be harvested and create local resources for each remote resource. It will also keep the resources synchronized with the remote service by periodically updating them.

Out of the box, GeoNode ships with support for harvesting from:

1. Other remote GeoNode instances;
2. OGC WMS servers;
3. *ArcGIS REST services*.

Adding support for *additional harvesting sources* is also possible.

**GeoNode harvesting concepts**

When a **harvester** is configured, GeoNode is able to use its corresponding **harvester worker** to contact the remote service and generate a list of **harvestable resources**. The user is then able to select which of those resources are of interest. Depending on its configured update frequency, sometime later, the **harvesting scheduler** will create new **harvesting sessions** in order to create local GeoNode resources from the remote harvestable resources that had been marked as relevant by the user.

The above description uses the following key concepts:

- **harvester**
  This is the configuration object that is used to parametrize harvesting of a remote service. It is configurable at runtime and is preserved in the GeoNode database.

  Harвестers and their properties can be managed by visiting the **Harvesting -> Harvesters** section of the GeoNode admin area, or by visiting the `api/v2/harvesters/` API endpoint with an admin user.

  Among other parameters, a harvester holds:

  - **remote_url**
    Base URL of the remote service being harvested, e.g. `https://stable.demo.geonode.org`

  - **harvester_type**
    Type of harvester worker that will be used to perform harvesting. See the **Harvester worker concept** and the **standard harvester workers** sections below for more detail. Example: `geonode.harvesting.harvesters.geonodeharvester.GeonodeUnifiedHarvesterWorker`.

  - **scheduling_enabled**
    Whether harvesting shall be performed periodically by the **harvesting scheduler** or not.

  - **harvesting_session_update_frequency**
    How often (in minutes) should new **harvesting sessions** be automatically scheduled?

  - **refresh_harvestable_resources_update_frequency**
    How often (in minutes) should new **refresh sessions** be automatically scheduled?

  - **default_owner**
    Which GeoNode user shall be made the owner of harvested resources

  - **harvest_new_resources_by_default**
    Should new remote resources be harvested automatically? When this option is selected, the user does not need to specify which **harvestable resources** should be harvested, as all of them will be automatically marked for harvesting by GeoNode.

  - **delete_orphan_resources_automatically**
    Orphan resources are those that have previously been created by means of a harvesting operation but that GeoNode can no longer find on the remote service being harvested. Should these resources be deleted from GeoNode automatically? This also applies to when a harvester configuration is deleted, in which case all of the resources that originated from that harvester are now considered to be orphan.

- **harvester worker**
  Harvester workers implement retrieval for concrete remote service types. Each harvester uses a specific worker, depending on the type of remote service that it gets data from. Harvester workers may accept their own additional configuration parameters.

  Harvester workers are set as the **harvester_type** attribute on a **harvester**. Their configuration is set as a JSON object on the **harvester_type_specific_configuration** attribute of the harvester.
GeoNode ships with the following harvester workers:

1. **GeoNode** - Enables harvesting from other GeoNode deployments
2. **WMS** - Enables harvesting from OGC WMS servers
3. **ArcGIS REST services** - Enables harvesting from ArcGIS REST services

Adding new harvester workers is also possible. This allows custom GeoNode deployments to add support for harvesting from other remote sources.

**Harvestable resource**

A resource that is available on the remote server. Harvestable resources are persisted in the GeoNode DB. They are created during **refresh operations**, when the harvester worker interacts with the remote service in order to discover which remote resources can be harvested.

Harvestable resources can be managed by visiting the **Harvesting -> Harvestable resources** section of the GeoNode admin area, or by visiting the api/v2/harvesters/{harvester-id}/harvestable-resources API endpoint with an admin user.

In order to be harvested by the **harvesting scheduler**, a harvestable resource must have its **should_be_harvested** attribute set to True. This attribute can be set manually by the user or it can be set automatically by the harvester worker, in case the corresponding harvester is configured with **harvest_new_resources_by_default = True**.

**Harvesting session**

In GeoNode, discovering remote resources and harvesting them is always done under the scope of a harvesting session. These sessions are stored in the GeoNode DB and can be inspected by visiting the **Harvesting -> Asynchronous harvesting sessions** section of the GeoNode admin area.

Harvesting sessions are used to keep track of the progress of execution of the relevant harvesting operations. They are updated while each operation is running. There are two types of sessions:

- **refresh session**
  This session is created during the **update of harvestable resources operation**. It has **type=discover-harvestable-resources**. During a refresh session, the harvester worker discovers remote resources and creates their respective harvestable resources on the GeoNode DB. After such session is finished, the user can inspect the found harvestable resources and mark those that are relevant with **should_be_harvested=True**.

- **harvesting session**
  This session is created during the **perform harvesting operation**. It has **type=harvesting**. During a harvesting session, the harvester worker creates or updates new GeoNode resources based on the harvestable resources that have been configured with **should_be_harvested=True**.

In addition to the aforementioned **type**, harvesting sessions also carry the **status** attribute, which provides context on the current status of the session (and consequently of the underlying harvesting operation).

**Harvesting scheduler**

The scheduler is responsible for initiating new **harvesting operations** in an automated fashion. Periodically, the scheduler goes through the list of existing harvesters, checking if it is time to dispatch one of the harvesting operations mentioned in the next section.

The scheduler’s operation frequency is configurable by defining a **HARVESTER_SCHEDULER_FREQUENCY_MINUTES** setting - the default is to trigger the scheduler every 30 seconds.

**Note:** Since the harvesting scheduler only checks if there is work to do once every x seconds (defaulting to 30 seconds, as mentioned above), there will usually be a delay between the time a harvesting operation is supposed to be scheduled and the actual time when it is indeed scheduled. Moreover, the harvesting scheduler is implemented...
as a celery task. This means that, if the celery worker is busy, that may also cause a delay in scheduling harvesting operations, as the scheduler’s celery task may not be triggered immediately.

Harvesting workflows

There are two main possible harvesting workflows:

1. Continuous harvesting
2. One-time harvesting

Continuous harvesting

This workflow relies on the harvesting scheduler in order to ensure harvested resources are continuously kept up to date with their remote counterparts.

1. User creates harvester and sets its scheduling_enabled attribute to True;
2. When the time comes, the harvesting scheduler calls the update list of harvestable resources operation. Alternatively, the user may call this operation manually the first time.
3. When the previous operation is done, user goes through the list of generated harvestable resources and, for each relevant harvestable resource, sets it should_be_harvested attribute to True. Alternatively, if the harvester has its harvest_new_resources_automatically attribute set to True, the harvestable resources will already be marked as to be harvested, without requiring manual user intervention;
4. When the time comes, the harvesting scheduler calls the perform harvesting operation. This causes the remote resources to be harvested. These now show up as resources on the local GeoNode.

One-time harvesting

This workflow is mostly executed manually by the user.

1. User creates harvester and sets its scheduling_enabled attribute to False;
2. User calls the update list of harvestable resources operation;
3. When the previous operation is done, user goes through the list of generated harvestable resources and, for each relevant harvestable resource, sets it should_be_harvested attribute to True;
4. User then proceeds to call the perform harvesting operation. This causes the remote resources to be harvested. These now show up as resources on the local GeoNode.

Harvester operations

Each GeoNode harvester is able to perform a finite set of operations. These can be performed either:

1. In an automated fashion, being dispatched by the harvesting scheduler. Automated harvesting is only performed when the corresponding harvester has scheduling_enabled=True;
2. On-demand, by explicit request of the user. On-demand execution can be requested by one of two ways:
   1. By selecting the relevant harvester(s) in the Harvesting -> Harvesters section of the GeoNode admin area and then selecting and running an action from the drop-down menu;
2. By interacting with the GeoNode REST API. Harvester actions are requested by issuing HTTP PATCH requests to the /api/v2/harvesters/{harvester-id}/ endpoint. The payload of such requests must specify the corresponding status. For example, by issuing a request like:

```bash
$ curl -X PATCH http://localhost/api/v2/harvesters/1/ "Content-Type: application/json" -u "myuser:mypass" --data '{"status": "updating-harvestable-resources"}"
```

We are asking that the harvester's status be changed to updating-harvestable-resources. If the server accepts this request, then the update list of harvestable resources operation is triggered.

---

**Note:** The server will not accept the API request if the harvester's current status is not ready.

While performing an action, the harvester’s status property transitions from ready to whatever action-related status is appropriate (as indicated below). As the operation finishes execution, the harvester’s status transitions back to ready. If the harvester has any status other than ready, then it is currently busy. When a harvester is busy it cannot execute other operations, you’ll need to wait until the current operation finishes.

### Check if the remote service is available operation

This operation causes the harvester to perform a simple health check on the remote service, in order to check whether it responds successfully. The response is stored in the harvester’s remote_available property. This operation is performed in the same process of the main GeoNode (i.e. it runs synchronously).

When triggered, this operation causes the harvester’s status to transition to checking-availability. As the operation finishes, the harvester’s status transitions back to ready.

Invocation via the GeoNode admin is performed by selecting the Check availability of selected harvesters command.

Invocation via the GeoNode REST API is performed by issuing an HTTP PATCH request with a payload that sets the harvester status.

### Update the list of harvestable resources operation

This operation causes the harvester to interact with the remote service in order to discover which resources are available for being harvested. Existing remote resources are then saved as harvestable resources.

Since this operation can potentially take a long time to complete (as we don’t know how may resources may exist on the remote service), it is run using a background process. GeoNode creates a new refresh session and uses it to track the progress of this operation.

When triggered, this operation causes the harvester’s status to transition to updating-harvestable-resources. As the operation finishes, the harvester’s status transitions back to ready.

Invocation via the GeoNode admin is performed by selecting the Update harvestable resources for selected harvesters command.

Invocation via the GeoNode REST API is performed by issuing an HTTP PATCH request with a payload that sets the harvester status.
Perform harvesting operation

This operation causes the harvester to check which harvestable resources are currently marked as being harvestable and then, for each one, harvest the resource from the remote server.

Since this operation can potentially take a long time to complete (as we don’t know how many resources may exist on the remote service), it is run using a background process. GeoNode creates a new *harvesting session* and uses it to track the progress of this operation.

When triggered, this operation causes the harvester’s status to transition to *harvesting-resources*. As the operation finishes, the harvester’s status transitions back to *ready*.

Invocation via the GeoNode admin is performed by selecting the *Perform harvesting on selected harvesters* command.

Invocation via the GeoNode REST API is performed by issuing an HTTP PATCH request with a payload that sets the harvester status.

Abort update of harvestable resources operation

This operation causes the harvester to abort an on-going *update list of harvestable resources operation*.

When triggered, this operation causes the harvester’s status to transition to *aborting-update-harvestable-resources*. As the operation finishes, the harvester’s status transitions back to *ready*.

Invocation via the GeoNode admin is performed by selecting the *Abort on-going update of harvestable resources for selected harvesters* command.

Invocation via the GeoNode REST API is performed by issuing an HTTP PATCH request with a payload that sets the harvester status.

Abort harvesting operation

This operation causes the harvester to abort an on-going *perform harvesting operation*.

When triggered, this operation causes the harvester’s status to transition to *aborting-performing-harvesting*. As the operation finishes, the harvester’s status transitions back to *ready*.

Invocation via the GeoNode admin is performed by selecting the *Abort on-going harvesting sessions for selected harvesters* command.

Invocation via the GeoNode REST API is performed by issuing an HTTP PATCH request with a payload that sets the harvester status.

Reset harvester operation

This operation causes the harvester’s status to be reset back to *ready*. It is mainly useful for troubleshooting potential errors, in order to unlock harvesters that may get stuck in a non-actionable status when some unforeseen error occurs.

When triggered, this operation causes the harvester’s status to transition to *ready* immediately.

Invocation via the GeoNode admin is performed by selecting the *Reset harvester status* command.

This operation cannot be called via the GeoNode API.
Standard harvester workers

Note: Remember that, as stated above, a harvester worker is configured by means of setting the `harvester_type` and `harvester_type_specific_configuration` attributes on the `harvester`.

Moreover, the format of the `harvester_type_specific_configuration` attribute must be a JSON object.

GeoNode harvester worker

This worker is able to harvest remote GeoNode deployments. In addition to creating local resources by retrieving the remote metadata, this harvester is also able to copy remote datasets over to the local GeoNode. This means that this harvester can even be used in order to generate replicated GeoNode instances.

This harvester can be used by setting `harvester_type=geonode.harvesting.harvesters.geonodeharvester. GeonodeUnifiedHarvesterWorker` in the harvester configuration.

It recognizes the following `harvester_type_specific_configuration` parameters:

**harvest_datasets**
Whether to harvest remote resources of type `dataset` or not. Acceptable values: `true` (the default) or `false`.

**copy_datasets**
Whether to copy remote resources of type `dataset` over to the local GeoNode. Acceptable values: `true` or `false` (the default).

**harvest_documents**
Whether to harvest remote resources of type `document` or not. Acceptable values: `true` (the default) or `false`.

**copy_documents**
Whether to copy remote resources of type `document` over to the local GeoNode. Acceptable values: `true` or `false` (the default).

**resource_title_filter**
A string that must be present in the remote resources' `title` in order for them to be acknowledged as harvestable resources. This allows filtering out resources that are not relevant. Acceptable values: any alphanumeric value.

Example: setting this to a value of "water" would mean that the harvester would generate harvestable resources for remote resources that are titled `water basins`, `Water territories`, etc. The harvester would not generate harvestable resources for remote resources whose title does not contain the word `water`.

**start_date_filter**
A string specifying a datetime that is used to filter out resources by their `start_date`. This is parsed with `dateutil.parser.parse()`, which means that it accepts many different formats (e.g. `2021-06-31T13:04:05Z`)

**end_date_filter**
Similar to `start_date_filter` but uses resources' `end_date` as a filter parameter.

**keywords_filter**
A list of keywords that are used to filter remote resources.

**categories_filter**
A list of categories that are used to filter remote resources.
WMS harvester worker

This worker is able to harvest from remote OGC WMS servers.

This harvester can be used by setting `harvester_type=geonode.harvesting.harvesters.wms.OgcWmsHarvester` in the harvester configuration.

It recognizes the following `harvester_type_specific_configuration` parameters:

**dataset_title_filter**
A string that is used to filter remote WMS layers by their `title` property. If a remote layer's title contains the string defined by this parameter, then the layer is recognized by the harvester worker.

---

ArcGIS REST Services harvester worker

This worker is able to harvest from remote ArcGIS REST Services catalogs.

This worker is able to recognize two types of `remote_url`:

1. URL of the ArcGIS REST services catalog. This URL usually ends in `rest/services`. A catalog may expose several different services. This harvester worker is able to descend into the available ArcGIS Rest services and retrieve their respective resources. Example:

   [https://sampleserver6.arcgisonline.com/arcgis/rest/services](https://sampleserver6.arcgisonline.com/arcgis/rest/services)

2. URL of the ArcGIS REST services Service. This URL usually takes the form `{base-url}/rest/services/{service-name}/{service-type}`. Example:


This harvester worker can be used by setting `harvester_type=geonode.harvesting.harvesters.arcgis.ArcgisHarvesterWorker` in the harvester configuration.

It recognizes the following `harvester_type_specific_configuration` parameters:

**harvest_map_services**
Whether services of type `MapServer` ought to be harvested. Defaults to `True`.

**harvest_image_services**
Whether services of type `ImageServer` ought to be harvested. Defaults to `True`.

**resource_name_filter**
A string that is used to filter remote WMS layers by their `title` property. If a remote layer's name contains the string defined by this parameter, then the layer is recognized by the harvester worker.

**service_names_filter**
A list of names that are used to filter the remote ArcGIS catalog.
Creating new harvesting workers

New harvesting workers can be created by writing classes derived from geonode.harvesting.harvesters.base.BaseGeonodeHarvesterWorker. This class defines an abstract interface that must be implemented. All methods decorated with abc.abstractmethod must be implemented in the custom harvester worker class. Study the implementation of the standard GeoNode harvester worker classes in order to gain insight on how to implement custom ones.

After writing a custom harvester worker class, it can be added to the list of known harvester workers by defining the HARVESTER_CLASSES GeoNode setting. This setting is a list of strings, containing the Python class path to each harvester worker class. It has a default value of:

```python
HARVESTER_CLASSES = [
    "geonode.harvesting.harvesters.geonodeharvester.GeonodeUnifiedHarvesterWorker",
    "geonode.harvesting.harvesters.wms.OgcWmsHarvester",
    "geonode.harvesting.harvesters.arcgis.ArcgisHarvesterWorker",
]
```

These are the standard harvester worker classes shipped by GeoNode. If this setting is defined, its value will simply extend the default list. This means that it is not possible to disable the standard worker classes, only to add new ones.

1.19 GeoNode Backup and Restore

1.19.1 Full GeoNode Backup & Restore

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.

In particular the tool helps developers and administrators to correctly extract and serialize the following resources:

- **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

The usage of those commands is quite easy and straightforward.

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 to 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 created before running a backup or restore.

The default files can be found at `geonode/br/management/commands/settings_sample.ini` and `geonode/br/management/commands/settings_docker_sample.ini` for the classic and Docker environments accordingly. The content is similar in both of them (an example from `settings_sample.ini`):

```
[database]
pdgdump = pg_dump
pgrestore = pg_restore

[geoserver]
datadir = geoserver/data
dumpvectordata = yes
dumprasterdata = yes

[fixtures]
# NOTE: Order is important
apps = contenttypes,auth,people,groups,account,guardian,admin,actstream,announcements,
      avatar,base,dialogos,documents,geoserver,invitations,pinax_notifications,layers,maps,
      oauth2_provider,services,sites,socialaccount,taggit,tastypie,upload,user_messages
dumps = contenttypes,auth,people,groups,account,guardian,admin,actstream,announcements,
      avatar,base,dialogos,documents,geoserver,invitations,pinax_notifications,layers,maps,
      oauth2_provider,services,sites,socialaccount,taggit,tastypie,upload,user_messages
```

The `settings.ini` file can be created in any directory accessible by GeoNode, and it’s path can be passed to the backup / restore procedures using `-c` (`--config`) argument.
There are few different sections of the configuration file, that must be carefully checked before running a backup / restore command.

**Settings: [database] Section**

```yaml
[database]
pgdump = pg_dump
pgrestore = pg_restore
```

This section is quite simple. It contains only two 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. Make 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**

```yaml
[geoserver]
datadir = /opt/gs_data_dir
datadir_exclude_file_path =
dumpvectordata = yes
dumprasterdata = yes
data_dt_filter =
data_layername_filter =
data_layername_exclude_filter =
```

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.
- `datadir_exclude_file_path`: comma separated list of paths to exclude from `geoserver_catalog.zip`; This list will be sent and managed directly by the GeoServer Backup REST API.
- `dumpvectordata`: a boolean flag enabling or disabling creation of a vector data dump from GeoServer (shapefiles or DB tables). If `false` (or `no`) vector data won’t be stored / re-stored.
- `dumprasterdata`: a boolean flag enabling or disabling creation of a raster data dump from GeoServer (geotiffs). If `false` (or `no`) raster data won’t be stored / re-stored.
- `data_dt_filter`: `{cmp_operator} {ISO8601}` e.g. > 2019-04-05T24:00 which means “include on backup archive only the files that have been modified later than 2019-04-05T24:00
• `data_layername_filter`: comma separated list of `layer names`, optionally with glob syntax e.g.: tuscany_*,italy; Only RASTER original data and VECTORIAL table dumps matching those filters will be **included** into the backup ZIP archive

• `data_layername_exclude_filter`: comma separated list of `layer names`, optionally with glob syntax e.g.: tuscany_*,italy; The RASTER original data and VECTORIAL table dumps matching those filters will be **excluded** from the backup ZIP archive

**Warning**: Enabling these options **requires** the GeoServer Data Dir to be accessible and **fully writable** for 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.
→spatialrepresentationtype,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,spatialrepresentationtypes,userpermissions,
→grouppermissions,uploadsessions,styles,layers,attributes,layerfiles,maps,maplayers,
→mapsnapshots,documents,tags
```

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`; an ordered list of GeoNode Django applications. The backup / restore procedure will dump / restore the fixtures in a portable format.

• `dumps`; this is the list of `files` associated to the Django applications. The order **must** be the same as in the `apps` property above. Each name represents the `file name` where to dump to / read from the single app’s fixtures.

**Executing from the CLI**

The following sections shows instructions on how to perform backup / restore from the command line by using the Django 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.

By default both procedures activate *Read Only* mode, disabling any content modifying requests, which is reverted to the previous state (from before the execution) after finish, regardless of the command’s result (success or failure). To disable activation of this mode, `--skip-read-only` argument can be passed to the command.
It is worth notice that both commands allows the following option

```
python manage.py backup --force / -f
python manage.py restore --force / -f
```

Which enables a non-interactive mode, meaning the user will not be asked for an explicit confirmation.

**Backup**

In order to perform a backup just run the command:

```
python manage.py backup --backup-dir=<target_bk_folder_path> --config=/path/to/settings.ini
```

The management command will automatically generate a .zip archive file on the target folder in case of success. In the target directory .md5 file with the same name as backup will be created. It contains the MD5 hash of the backup file, which can be used to check archive’s integrity before restoration.

It is worth to mention that br (Backup & Restore GeoNode application) will not be dumped, even if specified in the settings.ini as its content is strictly related to the certain GeoNode instance.

Currently, GeoNode does not support any automatic extraction of the backup file. It should be manually transferred, if needed to the target instance environment.

**Restore**

The restore command has a number of arguments, modifying its execution:

1. `--skip-geoserver`: the GeoServer backup restoration won’t be performed
2. `--skip-geoserver-info`: {Default: True} Skips GeoServer Global Infos, like the proxy base url and other global GeoServer metadata info
3. `--skip-geoserver-security`: {Default: True} Skips GeoServer all the Security Settings
4. `--backup-file`: (exclusive together with `--backup-files-dir`) path to the backup .zip archive
5. `--backup-files-dir`: (exclusive together with `--backup-file`) directory containing backup archives. The directory may contain a number of files, but only backup archives are allowed with a .zip extension. In case multiple archives are present in the directory, the newest one, created after the last already restored backup creation time, will be restored. This option was implemented with a thought of automated restores.
6. `--recovery-file`: Backup archive containing GeoNode data to restore in case of failure.
7. `-l / --with-logs`: the backup file will be checked against the restoration logs (history). In case this backup has already been restored (MD5 based comparision), RuntimeError is raised, preventing restore execution.
8. `-n / --notify`: the restore procedure outcome will be send by an e-mail notification to the superusers of the instance (note: notification will be sent to the superusers of the instance before restoration).
9. `--skip-read-only`: the restore procedure will be conducted without setting Read Only mode during execution.
10. `--soft-reset`: the restore procedure will preserve geoserver table / resources during the restore. By default the procedure will drop tables and resources

1.19. GeoNode Backup and Restore
In order to perform a default backup restoration just run the command:

```
python manage.py restore --backup-file=<target_restore_file_path> --config=/path/to/settings.ini
```

For restore to run it requires either --backup-file or --backup-files-dir argument defined.

**Warning:** The Restore will overwrite the whole target instances of GeoNode (and by default GeoServer) including users, catalog and database, so be very careful.

**GeoNode Admin GUI Inspection**

The history of restored backups can be verified in the admin panel.

Login to the admin panel and select Restored backups table from BACKUP/RESTORE application.

A list will be displayed with a history of all restored backups. You can select a certain backup to view it's data.

The detailed view of the restored backup shows backup archive's name, it's MD5 hash, it's creation/modification date (in the target folder), and the date of the restoration. Please note Restored Backup history cannot be modified.

**B/R in Docker environment**

When executing B/R in the Docker environment, creation backup to / restoration from should be executed in /backup_restore directory. It is a shared volume between Geoserver and Geonode images, created for this purpose only. Pointing at another location will fail, as one of the images won’t have an access to the files.

**Warning:** When executing B/R in Docker environment remember to create settings.ini file basing on settings_docker_sample.ini to point at a proper Geoserver data directory! In other case configuration mismatch may cause unexpected errors.

**Warning:** The only other volume shared between images is /geoserver_data/data, but backup creation should not be performed there, as the recursive Geoserver backups may be created in such case.
B/R Jenkins Job in Docker environment

When installing GeoNode through the geonode-project Docker (see GeoNode Basic Installation), an instance of Jenkins CI/CD is also automatically deployed and available through http://<geonode_host>/jenkins.

Configure Jenkins at first startup

The very first time you try to access Jenkins, you will need to unlock it and generate a new administrator username and password.

In order to do that, you need to print the contents of the auto-generated file /var/jenkins_home/secrets/initialAdminPassword

1. First of all search for the Jenkins container ID, usually jenkins4{{project_name}} where {{project_name}} is the name of your geonode-project instance (e.g. my_geonode)

```bash
$> docker ps
```

<table>
<thead>
<tr>
<th>CONTAINER ID</th>
<th>IMAGE</th>
<th>COMMAND</th>
<th>CREATED</th>
</tr>
</thead>
<tbody>
<tr>
<td>e9fc97a75d1a</td>
<td>geonode/nginx:geoserver</td>
<td>&quot;/docker-entrypoint....&quot;</td>
<td>2 hours</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.0.0.0:80-&gt;80/tcp, 0.0.0.0:443-&gt;443/tcp</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>nginx4my_geonode</td>
<td></td>
</tr>
<tr>
<td>c5496400b1b9</td>
<td>my_geonode_django</td>
<td>&quot;/bin/sh -c 'service...&quot;</td>
<td>2 hours</td>
</tr>
<tr>
<td></td>
<td></td>
<td>my_geonode_django</td>
<td></td>
</tr>
<tr>
<td>bc899f81fa28</td>
<td>my_geonode_celery</td>
<td>&quot;/bin/sh -c 'service...&quot;</td>
<td>2 hours</td>
</tr>
<tr>
<td></td>
<td></td>
<td>my_geonode_celery</td>
<td></td>
</tr>
<tr>
<td>3b213400d6d0</td>
<td>geoserver:2.17.1</td>
<td>&quot;/usr/local/tomcat/t...&quot;</td>
<td>2 hours</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8080/tcp</td>
<td></td>
</tr>
</tbody>
</table>

(continues on next page)
Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log (not sure where to find it?) and this file on the server:

/var/jenkins_home/secrets/initialAdminPassword

Please copy the password from either location and paste it below.

Administrator password

Continue
1. Now just cat the file above inside the Jenkins container

```bash
$ docker container exec -u 0 -it jenkins4my_geonode sh -c 'cat /var/jenkins_home/secrets/initialAdminPassword'
```

1. Copy the hash code you just got from the print above, and copy-and-paste to the browser window

In the next step just install the Default Plugins. You can install more of them later on from the management page.

Provide the administrator credentials as requested

Confirm the Jenkins instance URL, this can be changed from the configuration later in case you will need to update the server address

Well done, Jenkins is ready now

The next step is to configure a Jenkins Job able to interact with the Django Docker container and run a full backup

**Configure a Jenkins Job to run a full backup on the Django Container**

Before creating the new Jenkins job, we need to install and configure a new plugin, Publish over SSH

In order to do that, once logged in as admin, go to the Jenkins Management Page > Manage Plugins tab

Click on Available tab and search for SSH available plugins

Select and check the Publish over SSH one

Install the plugins and restart Jenkins

The next step is to configure the SSH Server Connection for the Publish over SSH plugin.

Move to Jenkins Configuration

Scroll down until you find the Publish over SSH plugin section

Depending on how your HOST SSH service has been configured, you might need several information in order to set up the connection.

Here below an example using a global host (master.demo.geonode.org) accepting SSH connections via RSA keys
Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log (not sure where to find it?) and this file on the server:

/var/jenkins_home/secrets/initialAdminPassword

Please copy the password from either location and paste it below.

Administrator password

[Password field]

Continue
Customize Jenkins
Plugins extend Jenkins with additional features to support many different needs.

- **Install suggested plugins**
  - Install plugins the Jenkins community finds most useful.

- **Select plugins to install**
  - Select and install plugins most suitable for your needs.
Getting Started
Getting Started

Instance Configuration

Jenkins URL: [http://localhost/jenkins/]

The Jenkins URL is used to provide the root URL for absolute links to various Jenkins resources. That means this value is required for proper operation of many Jenkins features including email notifications, PR status updates, and the BUILD_URL environment variable provided to build steps.

The proposed default value shown is not saved yet and is generated from the current request, if possible. The best practice is to set this value to the URL that users are expected to use. This will avoid confusion when sharing or viewing links.
Jenkins is ready!

Your Jenkins setup is complete.

Start using Jenkins
1.19. GeoNode Backup and Restore
Go back to the top page

(you can start using the installed plugins right away)

☑ Restart Jenkins when installation is complete and no jobs are running
1.19. GeoNode Backup and Restore
Note: Before saving the configuration always ensure the connection is ok by using the Test Configuration button.

It is possible also to run and configure Jenkins to run locally, as an instance on localhost. In that case you will need to change few things in order to allow Jenkins to access your local network.

1. First of all, be sure OpenSSH Server is correctly installed and running on your PC. Eventually check any firewall rules.

   $> sudo apt install openssh-server
   
   # Test your connection locally
   $> ssh -p 22 user@localhost
   user@localhost's password:

2. You will need to do some changed to your docker-compose.yml file in order to enable the host network configuration.
Note: Enable network_mode: "host" on Jenkins container

```bash
$> vim docker-compose.yml
...
jenkins:
  image: jenkins/jenkins:lts
  container_name: jenkins4${COMPOSE_PROJECT_NAME}
  user: jenkins
  ports:
    - '${JENKINS_HTTP_PORT}:${JENKINS_HTTP_PORT}'
    - '${JENKINS_HTTPS_PORT}:${JENKINS_HTTPS_PORT}'
    - '50000:50000'
  network_mode: "host"
  volumes:
    - jenkins_data:/var/jenkins_home
    - backup-restore:/backup_restore
  # - /var/run/docker.sock:/var/run/docker.sock
  environment:
    - 'JENKINS_OPTS=--httpPort=${JENKINS_HTTP_PORT} --httpsPort=${JENKINS_HTTPS_PORT} --prefix=/jenkins'
...

# Recreate the Jenkins container
$> docker-compose stop jenkins
$> docker-compose rm jenkins
$> docker-compose up -d jenkins
```

Warning: From now on, your local Jenkins instance will be accessible from http://localhost:9080/jenkins

3. Add localhost Server to the Publish over SSH plugin configuration

Mode to http://localhost:9080/jenkins/configure and fill the required information

Note: Before saving the configuration always ensure the connection is ok by using the Test Configuration button

We are now ready to create the Jenkins Job which will run a full backup & restore of our GeoNode dockerized instance.
1. Move to the Jenkins Home and click on Create a Job button
2. Provide a name to the Job and select Freestyle project
3. Enable the Log rotation strategy if needed
4. Configure the Job Parameters which will be used by the script later on.
   Add three String Parameters
   as shown below
   1. **BKP_FOLDER_NAME**
   2. **SOURCE_URL**

   **Warning:** Provide the correct URL of your GeoNode instance

3. **TARGET_URL**

   **Warning:** Provide the correct URL of your GeoNode instance

5. Enable the Delete workspace before build starts and Add timestamps to the Console Output Build Environment options
6. Finally let’s create the SSH Build Step
   Select the correct SSH Server and provide the Exec Command below
1.19. GeoNode Backup and Restore
1.19. GeoNode Backup and Restore
Warning: Replace \{project_name\} with your geonode-project instance name (e.g. my_geonode)

```bash
# Replace \{project_name\} with your geonode-project instance name (e.g. my_geonode)
# docker exec -u 0 -it django4\{project_name\} sh -c 'SOURCE_URL=$SOURCE_URL TARGET_URL=$TARGET_URL ./\{project_name\}/br/backup.sh $BKP_FOLDER_NAME'
# e.g.:
docker exec -u 0 -it django4my_geonode sh -c 'SOURCE_URL=$SOURCE_URL TARGET_URL=$TARGET_URL ./my_geonode/br/backup.sh $BKP_FOLDER_NAME'
```

Click on Advanced and change the parameters as shown below

Save! You are ready to run the Job…
1.19. GeoNode Backup and Restore
1.19. GeoNode Backup and Restore
Link the `backup_restore` folder to a local folder on the HOST

In the case you need to save the backup archives outside the docker container, there’s the possibility to directly link the `backup_restore` folder to a local folder on the HOST.

In that case you won’t need to `docker cp` the files everytime from the containers, they will be directly available on the host filesystem.

**Warning:** Always keep an eye to the disk space. Backups archives may be huge.

**Note:** You might want also to consider filtering the files through the backup dt filters on the `settings.ini` in order to reduce the size of the archive files, including only the new ones.

Modify the `docker-compose.override.yml` as follows in order to link the backup folders outside.

**Note:** `/data/backup_restore` is a folder physically located into the host filesystem.

```
$> vim docker-compose.override.yml

version: '2.2'
services:
django:
  build: .
  # Loading the app is defined here to allow for
  # autoreload on changes it is mounted on top of the
  # old copy that docker added when creating the image
  volumes:
    - './:/usr/src/my_geonode'
    - '/data/backup_restore:/backup_restore'  # Link to local volume in the HOST
celery:
  volumes:
    - '/data/backup_restore:/backup_restore'  # Link to local volume in the HOST
geoserver:
  volumes:
    - '/data/backup_restore:/backup_restore'  # Link to local volume in the HOST
jenkins:
  volumes:
    - '/data/backup_restore:/backup_restore'  # Link to local volume in the HOST

# Restart the containers
$> docker-compose up -d
```
1.20 GeoNode Components and Architecture

1.20.1 OAuth2 Security: Authentication and Authorization

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 a 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/3.2/topics/auth/](https://docs.djangoproject.com/en/3.2/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 later. 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:

```python
# 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,
        'GEONODE_SECURITY_ENABLED': True,
        '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',
        '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 Access Authentication](https://tools.ietf.org/id/rfc2617) in order to check for user's credentials.

   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 an 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 GeoNodeCookieProcessingFiler.

Summarizing we will have different ways to access GeoNode Layers:

1. Through GeoNode via Django Authentication and GeoNodeCookieProcessingFiler; 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).

Let’s 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** key 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: 

```python
AUTH_IP_WHITELIST = ['192.168.1.158', '192.168.1.159']
```

Default values are:

```python
... 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:

```python
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:

```python
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',  
    # If you use SessionAuthenticationMiddleware, be sure it appears before   
    # OAuth2TokenMiddleware.  
    # SessionAuthenticationMiddleware is NOT required for using django-oauth-toolkit.  
    'oauth2_provider.middleware.OAuth2TokenMiddleware',  
)  
```

1.20. GeoNode Components and Architecture 349
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:

```python
# 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
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
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

   ![GeoServer Login](image)

   **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
Security
- Settings
- Authentication
- Passwords
- Users, Groups, Roles
- WPS security
- GeoFence
- GeoFence Data Rules
- GeoFence Admin Rules

Role Services
- Add new
- Remove selected

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Administrator Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>default</td>
<td>Default XML role service</td>
<td>ADMIN</td>
</tr>
<tr>
<td>geonode REST role service</td>
<td>Author/REY REST Role Service</td>
<td>ROLE_ADMIN</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>default</td>
<td>Default XML user/group service</td>
</tr>
</tbody>
</table>

Results 1 to 1 (out of 1 item)

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

<table>
<thead>
<tr>
<th>Settings</th>
<th>Roles</th>
</tr>
</thead>
</table>

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
...

Security

- Settings
- Authentication
- Users, Groups, Roles
- Data
Security Settings

Configure security settings

Active role service

- geonode REST role service
- default
- geonode REST role service

- Encrypt web admin URL parameters

Password encryption

- Weak PBE

⚠️ 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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>anonymous</td>
<td>Anonymous authentication</td>
</tr>
<tr>
<td>basic</td>
<td>Basic HTTP authentication</td>
</tr>
<tr>
<td>form</td>
<td>Form authentication</td>
</tr>
<tr>
<td>geonode-oauth2</td>
<td>Authentication using a GeoNode OAuth2</td>
</tr>
<tr>
<td>geonodeAnonymousFilter</td>
<td>org.geonode.security.GeoNodeAnonymousProcessingFilter</td>
</tr>
<tr>
<td>geonodeCookieFilter</td>
<td>org.geonode.security.GeoNodeCookieProcessingFilter</td>
</tr>
<tr>
<td>rememberme</td>
<td>Remember me authentication</td>
</tr>
</tbody>
</table>

   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

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

4. Create / update the geonode-oauth2 - Authentication using a GeoNode OAuth2 accordingly
   - Name: Must be geonode-oauth2
New Authentication Filter

Create and configure a new Authentication Filter

**J2EE** - Delegates to servlet container for authentication
**GeoNode CAuth2** - Authenticates by looking up for a valid GeoServer

Authentication using a GeoNode OAuth2 geonode-oauth2

Authenticates by looking up for a valid GeoNode OAuth2 access_token key sent as URL parameter.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>geonode-oauth2</td>
<td></td>
</tr>
<tr>
<td>OAuth2 provider connection</td>
<td></td>
</tr>
<tr>
<td>Enable Redirect Authentication Endpoint</td>
<td></td>
</tr>
<tr>
<td>Login Authentication Endpoint</td>
<td>j_spring_oauth2_geonode_login</td>
</tr>
<tr>
<td>Logout Authentication Endpoint</td>
<td>j_spring_oauth2_geonode_logout</td>
</tr>
<tr>
<td>Force Access Token URI HTTPS Secured Protocol</td>
<td>False</td>
</tr>
<tr>
<td>Access Token URI</td>
<td>http://&lt;geonode_host_base_url&gt;/o/token/</td>
</tr>
<tr>
<td>Force User Authorization URI HTTPS Secured Protocol</td>
<td>False</td>
</tr>
<tr>
<td>User Authorization URI</td>
<td>http://&lt;geonode_host_base_url&gt;/o/authorize/</td>
</tr>
<tr>
<td>Redirect URI</td>
<td>http://&lt;geoserver_host&gt;/geoserver. This address must be present on the Redirect uris of GeoNode OAuth2 &gt; Applications &gt; GeoServer (see above)</td>
</tr>
<tr>
<td>Check Token Endpoint URL</td>
<td>http://&lt;geonode_host_base_url&gt;/api/v4/tokeninfo/</td>
</tr>
<tr>
<td>Logout URI</td>
<td>http://&lt;geonode_host_base_url&gt;/account/logout/</td>
</tr>
</tbody>
</table>

- Enable Redirect Authentication Endpoint: 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
- 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/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.
Filter Chains

- Add service chain
- Add HTML chain

<table>
<thead>
<tr>
<th>Position</th>
<th>Name</th>
<th>Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>↓</td>
<td>web</td>
<td>/web/<strong>,/gwc/rest/web/</strong>,/</td>
</tr>
<tr>
<td>↑ ↓</td>
<td>webLogin</td>
<td>/j_spring_security_check,/j_spring_security_check,/</td>
</tr>
<tr>
<td>↑ ↓</td>
<td>webLogout</td>
<td>/j_spring_security_logout,/j_spring_security_logout,/</td>
</tr>
<tr>
<td>↑ ↓</td>
<td>rest</td>
<td>/rest/**</td>
</tr>
<tr>
<td>↑ ↓</td>
<td>gwc</td>
<td>/gwc/rest/**</td>
</tr>
<tr>
<td>↑</td>
<td>default</td>
<td>/**</td>
</tr>
</tbody>
</table>

Results 1 to 6 (out of 6 items)

<table>
<thead>
<tr>
<th>Available</th>
<th>Selected</th>
</tr>
</thead>
<tbody>
<tr>
<td>basic</td>
<td>geonode-cauth2</td>
</tr>
<tr>
<td>geonodeAnonymousFilter</td>
<td>rememberme</td>
</tr>
<tr>
<td>geonodesCookieFilter</td>
<td>form</td>
</tr>
<tr>
<td></td>
<td>anonymous</td>
</tr>
</tbody>
</table>
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

![Filter Chain Configuration]

**Warning:** Every time you modify a Filter Chain, *don't forget to save* the Authentication settings. This *must* be done for each change.
6. Make sure the default 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.
7. Add the GeoNode Login Endpoints to the comma-delimited list of the webLogin Filter Chain

Filter chain

Configure an individual filter chain

Chain settings

Name
webLogin

Comma-delimited list of ANT patterns (with optional query string)

[jh2.geonode_login,j_spring_security.geonode_login]

- Disable security for this chain
- Allow creation of an HTTP session for storing the authentication token
- Accept only SSL requests

Role filter

**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

Filter chain

Configure an individual filter chain

Chain settings

Name
webLogout

Comma delimited list of ANT patterns (with optional query string)
2_geonode_logout/j_spring_oauth2.geonode_logout

- Disable security for this chain
- Allow creation of an HTTP session for storing the auth
- Accept only SSL requests

Role filter

**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

```
<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

   ![Security tab](image)

   1. Access the Security > Authentication section

   ![Authentication Providers](image)

   2. Identify the section Authentication Providers and make sure the geofence Authentication Provider is present

   ![Authentication Providers table](image)

   3. Make sure the Provider Chain is configured as shown below

   ![Warning](image)

   Warning: Every time you modify an 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
Security

- Settings
- Authentication
- Passwords
- Users, Groups, Roles
- Data
- Services
- WPS security
- GeoFence
- GeoFence Data Rules
- GeoFence Admin Page

Connection successful

GeoFence Admin Page

GeoFence options Administration Page

General settings

GeoServer Instance name for GeoFence

default-gs

GeoFence services URL (GeoServer restart is required if changed)

internal:/

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
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

   **Security**
   - Settings
   - Authentication
   - Passwords
   - Users, Groups, Roles
   - Data
   - Services
   - WPS security
   - GeoFence
   - GeoFence Data Rules
   - GeoFence Admin Rules

   Configure data rules for the internal GeoFence server.

   • Make sure the DENY ALL Rule is present by default, otherwise your data will be accessible to everyone

   **Note:** This rule is always the last one

   **GeoFence Data Rules**
   Configure data rules for the internal GeoFence server.

   • Add new rule
   • Remove selected rules

<table>
<thead>
<tr>
<th>P</th>
<th>Role</th>
<th>User</th>
<th>Service</th>
<th>Request</th>
<th>Workspace</th>
<th>Layer</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DENY</td>
</tr>
</tbody>
</table>

   **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.

        – 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 debug_geonode. 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 debug_geonode. 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 debug_geonode. 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

```bash
curl http://localhost/api/adminRole
$> {"adminRole": "admin"}

curl http://localhost/api/users
$> {"users": [{"username": "AnonymousUser", "groups": ["anonymous"]}, {"username": "afabiani", "groups": ["anonymous", "test"]}, {"username": "admin", "groups": ["anonymous", "test", "admin"]}]

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 debug_geonode. 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>
   
   ```bash
   openssl s_client -connect <geonode_host_base_url>:443
   ```

2. Store SSL Certificate on local hard-disk
   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)

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)
1.20. GeoNode Components and Architecture

GeoNode Documentation, Release master
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.
No keystore loaded
Could not establish a trust path for the certificate.
The certificate information will now be displayed after which you may confirm whether or not you trust the certificate.

Keystore type: JKS, provider: SUN, size: 3 entries
Certificate Details for 'google.cert'

- Version: 3
- Subject: CN=accounts.google.com, C=Google Inc, L=Mountain View, ST=California
- Issuer: CN=Google Internet Authority G2, C=Google Inc, C=US
- Serial Number: 7843 FCEF 470D D514
- Valid From: Oct 19, 2016 7:16:57 PM CEST
- Valid Until: Jan 11, 2017 6:13:00 PM CET
- Public Key: RSA (2,048 bits)
- Signature Algorithm: SHA256withRSA

Keystore type: JKS, provider: SUN, size: 3 entries
1.20. GeoNode Components and Architecture

Keystore type: JKS, provider: SUN, size: 3 entries
<table>
<thead>
<tr>
<th>Alias Name</th>
<th>Last Modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>* googleapis.com (google internet authority g2)</td>
<td>Oct 28, 2016 11:20:59 AM CEST</td>
</tr>
<tr>
<td>accounts.google.com (google internet authority g2)</td>
<td>Oct 28, 2016 11:09:09 AM CEST</td>
</tr>
<tr>
<td>geonode-dev-server</td>
<td>Sep 23, 2016 3:26:18 PM CEST</td>
</tr>
</tbody>
</table>

Keystore type: JKS, provider: SUN, size: 3 entries
F:\tmp\keystore.key* - PKCS#12 Keystore

<table>
<thead>
<tr>
<th>Alias Name</th>
<th>Last Modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>* googleapis.com (google internet authority g2)</td>
<td>Oct 28, 2016 11:20:59 AM CEST</td>
</tr>
<tr>
<td>accounts.google.com (google internet authority g2)</td>
<td>Oct 28, 2016 12:36:45 PM CEST</td>
</tr>
<tr>
<td>geonode-dev-server</td>
<td>Sep 23, 2016 3:26:18 PM CEST</td>
</tr>
</tbody>
</table>

Keystore type: JKS, provider: SUN, size: 3 entries

Import Trusted Certificate

Trusted Certificate Import Successful

OK
Save the loaded keystore to disk
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. For more details please refer to the official GeoFence documentation here.

1. Add Java Libraries to GeoServer

   wget --no-check-certificate "https://maven.geo-solutions.it/org/hibernatespatial/hibernate-spatial-postgis/1.1.3.2/hibernate-spatial-postgis-1.1.3.2.jar"
   wget --no-check-certificate "https://repo1.maven.org/maven2/org/postgis/postgis-jdbc/1.3.3/postgis-jdbc-1.3.3.jar"
   cp hibernate-spatial-postgis-1.1.3.2.jar <GEOSERVER_WEBAPP_DIR>/WEB-INF/lib
   cp postgis-jdbc-1.3.3.jar <GEOSERVER_WEBAPP_DIR>/WEB-INF/lib

   restart geoserver

2. Either create a DB with the updated schema here https://github.com/geoserver/geofence/blob/master/doc/setup/sql/002_create_schema_postgres.sql or enable the hbm2ddl auto creation through the configuration file (see step 3)
Note: Notice that “update” also creates the tables if they do not exist. In production, however, I would suggest to change it to “validate”

```sh
# If you want to create a new DB for GeoFence
sudo -u postgres createdb -O geonode geofence;
```

```sh
sudo -u postgres psql -d geofence -c "CREATE EXTENSION postgis;"
```

```sh
sudo -u postgres psql -d geofence -c "GRANT ALL ON geometry_columns TO PUBLIC;"
```

```sh
sudo -u postgres psql -d geofence -c "GRANT ALL ON spatial_ref_sys TO PUBLIC;"
```

```sh
sudo -u postgres psql -d geofence -c "GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO geonode;"
```

3. Add configuration similar to `geofence-datasource-ovr.properties` sample below (if loaded as GeoServer extension)

```properties
<GEOSERVER_DATA_DIR>/geofence/geofence-datasource-ovr.properties
```

```properties
# /* (c) 2019 Open Source Geospatial Foundation - all rights reserved
# * This code is licensed under the GPL 2.0 license, available at the root
# * application directory.
# */
#
geofenceVendorAdapter.databasePlatform=org.hibernate.spatial.postgis.PostgisDialect
geofenceDataSource.driverClassName=org.postgresql.Driver
geofenceDataSource.url=jdbc:postgresql://localhost:5432/geofence
geofenceDataSource.username=postgres
geofenceDataSource.password=postgres
geofenceEntityManagerFactory.jpaPropertyMap[hibernate.default_schema]=public

###########################################################################
## Other setup entries
###########################################################################
## hbm2ddl.auto may assume one of these values:
## - validate: validates the DB schema at startup against the internal model. May fail on oracle spatial.
## - update: updates the schema, according to the internal model. Updating automatically the production DB is dangerous.
## - create-drop: drop the existing schema and recreates it according to the internal model. REALLY DANGEROUS, YOU WILL LOSE YOUR DATA.
## You may want not to redefine the property entirely, in order to leave the default value (no action).

default:
geofenceEntityManagerFactory.jpaPropertyMap[hibernate.hbm2ddl.auto]=update
default
geofenceEntityManagerFactory.jpaPropertyMap[javax.persistence.validation.mode]=none
default
geofenceEntityManagerFactory.jpaPropertyMap[hibernate.validator.apply_to_ddl]=false
```

(continues on next page)
Hardening GeoNode

1.21.1 Publish on other than HTTP port (for e.g. 8082)

By default geonode will be installed in the port 80 (i.e. HTTP) port. But what if you want to change the port of the geonode to other than HTTP port (For this example, I am taking 8082 port)? We need to edit couple of things in the web configuration. First things is, we need to update the /etc/uwsgi/apps-enabled/geonode.ini file,

```
sudo vi /etc/uwsgi/apps-enabled/geonode.ini
```

Edit the following lines,

```
env = SITE_HOST_NAME=localhost:8082
env = SITEURL=http://localhost:8082
SITE_HOST_NAME=localhost
SITE_HOST_PORT=8082
```
After that we need to update the `/etc/nginx/sites-enabled/geonode` file.

```bash
sudo vi /etc/nginx/sites-enabled/geonode
```

Edit the following lines,

```plaintext
server {
    listen 8082 default_server;
    listen [::]:8082 default_server;
}
```

### 1.22 Social Login

#### 1.22.1 GeoNode Social Accounts

**Contents**

- *GeoNode Social Accounts*
  - Allow GeoNode to Login through Social Accounts (OIDC)
    - *Introduction*
    - *Usage*
    - *Quick Configuration*
    - *Advanced Configuration*

**Allow GeoNode to Login through Social Accounts (OIDC)**

**Introduction**

Through the so-called “social accounts,” GeoNode allows authentication through external providers that support the OIDC (OpenID Connect) protocol, such as Google and Microsoft Azure.

How does it work? Once the authentication provider is configured, GeoNode’s sign-on will display a new login button that redirects the user to the external authentication page. After successfully authenticating with the external provider and accepting privacy consents, the browser will redirect the user back to the GeoNode page, prompting them to enter any missing information the first time and automatically authenticating them on subsequent occasions.
Usage

Once the provider has been correctly configured (see below), GeoNode will allow the user to login through it.

The first time you login you will probably need to confirm your e-mail and other fields of the profile.

Note:

If you want a user to be automatically signed as a member of a group, you will need to:

1. Create the GroupProfile in GeoNode
2. Provide the groups or roles the user belongs to through the id_token or user_info metadata from the OIDC provider itself.

Notice that, in the case you would like to benefit from this functionality:

1. Every time the user sing-in again, the groups will be automatically re-assigned by GeoNode, and therefore it won’t be possible to assign them manually anymore.
2. If you need a user to be recognized as a manager of the groups declared from the provider, you will need to send a claim is_manager: True on the user info metadata.

Quick Configuration

Currently GeoNode comes with two predefined configurations that you can use to enable either Google or Microsoft Azure.

Google

1. Add to your .env the following settings

```
SOCIALACCOUNT_OIDC_PROVIDER_ENABLED=True
SOCIALACCOUNT_PROVIDER=google
```

2. Login into GeoNode as an admin, go to the Social Account settings, create a new geonode_openid_connect provider and insert the

| Client ID | Client Secret |

Microsoft Azure

1. Add to your .env the following settings

```
MICROSOFT_TENANT_ID=<the_tenant_id>
SOCIALACCOUNT_OIDC_PROVIDER_ENABLED=True
SOCIALACCOUNT_PROVIDER=azure
```

2. Login into GeoNode as an admin, go to the Social Account settings, create a new geonode_openid_connect provider and insert the

| Client ID | Client Secret |
Advanced Configuration

In the case you need to change the default behavior of GeoNode or add a new/custom OIDC provider, you will need to update the settings manually as follows.

```python
SOCIALACCOUNT_PROVIDERS = {
    SOCIALACCOUNT_OIDC_PROVIDER: {
        "NAME": "Your Custom Provider",
        "SCOPE": [
            # Custom scopes comma-separated
        ],
        "AUTH_PARAMS": {
            # Custom AUTH PARAMS
        },
        "COMMON_FIELDS": {
            "email": "email",
            "last_name": "family_name",
            "first_name": "given_name",
            "IS_MANAGER_FIELD": "the_custom_manager_claim",  # This is optional
            "ACCOUNT_CLASS": "the_custom_account_class",
            "ACCESS_TOKEN_URL": "the_custom_token_uri",
            "AUTHORIZE_URL": "the_custom_auth_uri",
            "ID_TOKEN_ISSUER": "the_custom_uri",  # or "PROFILE_URL": "the_custom_user_info_uri"; if you specify the "ID_TOKEN_ISSUER" this will take precedence
            "OAUTH_PKCE_ENABLED": True,
        }
    }
}
```

**Note:** If you specify the “ID_TOKEN_ISSUER” this will take precedence trying to fetch the user info metadata from the id_token.

If the id_token won’t be available, it will try to fallback to the “PROFILE_URL” uri.

In the case you will need to customize how the Adapter works and manages the Groups registration, you can inject a new class through the settings:

```
SOCIALACCOUNT_ADAPTER="geonode.people.adapters.
庆幸GenericOpenIDConnectAdapter"  # This is the default value
```

1.23 GeoNode Django Contrib Apps

1.23.1 Geonode auth via LDAP

This package provides utilities for using LDAP as an authentication and authorization backend for geonode.

The django_auth_ldap package is a very capable way to add LDAP integration with django projects. It provides a lot of flexibility in mapping LDAP users to geonode users and is able to manage user authentication.

However, in order to provide full support for mapping LDAP groups with geonode’s and enforce group permissions on resources, a custom geonode authentication backend is required. This contrib package provides such a backend, based on django_auth_ldap.
Installation

Installing this contrib package is a matter of:

1. Installing geonode
2. Installing system LDAP libraries (development packages needed)
3. Cloning this repository locally
4. Change to the `ldap` directory and install this contrib package

```
# 1. install geonode (not shown here for brevity)
# 2. install systemwide LDAP libraries
sudo apt install \
  libldap2-dev \
  libtasl2-dev

# 3. get geonode/contribs code
git clone https://github.com/GeoNode/geonode-contribs.git

# 4. install geonode ldap contrib package
cd geonode-contribs/ldap
pip install .
```

Configuration

1. Add `geonode_ldap.backend.GeonodeLdapBackend` as an additional auth backend.

```
# e.g. by updating your settings.py or local_settings.py
AUTHENTICATION_BACKENDS += (  
    "geonode_ldap.backend.GeonodeLdapBackend",  
)
```

You may use additional auth backends, the django authentication framework tries them all according to the order listed in the settings. This means that geonode can be setup in such a way as to permit internal organization users to login with their LDAP credentials, while at the same time allowing for casual users to use their facebook login (as long as you enable facebook social auth provider).

**Note:** The django's `django.contrib.auth.backends.ModelBackend` must also be used in order to provide full geonode integration with LDAP. However this is included by default on GeoNode settings

```
# The GeoNode default settings are the following
AUTHENTICATION_BACKENDS = (  
    'oauth2_provider.backends.OAuth2Backend',  
    'django.contrib.auth.backends.ModelBackend',  
    'guardian.backends.ObjectPermissionBackend',  
    'allauth.account.auth_backends.AuthenticationBackend',  
)
```

2. Set some additional configuration values. Some of these variables are prefixed with `AUTH_LDAP` (these are used directly by `django_auth_ldap`) while others are prefixed with `GEONODE_LDAP` (these are used by geonode_ldap). The geonode custom variables are:
• **GEONODE_LDAP_GROUP_PROFILE_FILTERSTR** - This is an LDAP search fragment with the filter that allows querying for existing groups. See example below

• **GEONODE_LDAP_GROUP_NAME_ATTRIBUTE** - This is the name of the LDAP attribute that will be used for deriving the geonode group name. If not specified it will default to `cn`, which means that the LDAP object’s common name will be used for generating the name of the geonode group

• **GEONODE_LDAP_GROUP_PROFILE_MEMBER_ATTR** - This is the name of the LDAP attribute that will be used for deriving the geonode membership. If not specified it will default to `member`

Example configuration:

```python
# add these import lines to the top of your geonode settings file
from django_auth_ldap import config as ldap_config
from geonode_ldap.config import GeonodeNestedGroupOfNamesType
import ldap

# enable logging
import logging
logger = logging.getLogger('django_auth_ldap')
logger.addHandler(logging.StreamHandler())
logger.setLevel(logging.DEBUG)

# add both standard ModelBackend auth and geonode.contrib.ldap auth
AUTHENTICATION_BACKENDS += ('geonode_ldap.backend.GeonodeLdapBackend',)

# django_auth_ldap configuration
AUTH_LDAP_SERVER_URI = os.getenv("LDAP_SERVER_URL")
AUTH_LDAP_BIND_DN = os.getenv("LDAP_BIND_DN")
AUTH_LDAP_BIND_PASSWORD = os.getenv("LDAP_BIND_PASSWORD")
AUTH_LDAP_USER_SEARCH = ldap_config.LDAPSearch(
    os.getenv("LDAP_USER_SEARCH_DN"),
    ldap.SCOPE_SUBTREE,
    os.getenv("LDAP_USER_SEARCH_FILTERSTR"))

# should LDAP groups be used to spawn groups in GeoNode?
AUTH_LDAP_MIRROR_GROUPS = strtobool(os.getenv("LDAP_MIRROR_GROUPS", 'True'))
AUTH_LDAP_GROUP_SEARCH = ldap_config.LDAPSearch(
    os.getenv("LDAP_GROUP_SEARCH_DN"),
    ldap.SCOPE_SUBTREE,
    os.getenv("LDAP_GROUP_SEARCH_FILTERSTR"))

AUTH_LDAP_GROUP_TYPE = GeonodeNestedGroupOfNamesType()
AUTH_LDAP_USER_ATTR_MAP = {
    "first_name": "givenName",
    "last_name": "sn",
    "email": "mailPrimaryAddress"
}

AUTH_LDAP_FIND_GROUP_PERMS = True
AUTH_LDAP_MIRROR_GROUPS_EXCEPT = ['test_group']
```

(continues on next page)
# these are not needed by django_auth_ldap - we use them to find and match

GroupProfiles and GroupCategories

GEONODE_LDAP_GROUP_NAME_ATTRIBUTE = os.getenv("LDAP_GROUP_NAME_ATTRIBUTE", default="cn")
GEONODE_LDAP_GROUP_PROFILE_FILTERSTR = os.getenv("LDAP_GROUP_SEARCH_FILTERSTR", default=→'(ou=research group)')
GEONODE_LDAP_GROUP_PROFILE_MEMBER_ATTR = os.getenv("LDAP_GROUP_PROFILE_MEMBER_ATTR", →default='member')

Example environment variables:

```
LDAP_SERVER_URL=ldap://<the_ldap_server>
LDAP_BIND_DN=uid=ldapinfo,cn=users,dc=ad,dc=example,dc=org
LDAP_BIND_PASSWORD=<something_secret>
LDAP_USER_SEARCH_DN=dc=ad,dc=example,dc=org
LDAP_USER_SEARCH_FILTERSTR=(&(uid=%(user)s)(objectClass=person))
LDAP_MIRROR_GROUPS=True
LDAP_GROUP_SEARCH_DN=cn=groups,dc=ad,dc=example,dc=org
LDAP_GROUP_SEARCH_FILTERSTR=(|(cn=abt1)(cn=abt2)(cn=abt3)(cn=abt4)(cn=abt5)(cn=abt6))
LDAP_GROUP_PROFILE_MEMBER_ATTR=uniqueMember
```

The configuration seen in the example above will allow LDAP users to login to geonode with their LDAP credentials. On first login, a geonode user is created from the LDAP user and its LDAP attributes cn and sn are used to populate the geonode user's first_name and last_name profile fields.

Any groups that the user is a member of in LDAP (under the cn=groups,dc=ad,dc=example,dc=org search base and belonging to one of (|(cn=abt1)(cn=abt2)(cn=abt3)(cn=abt4)(cn=abt5)(cn=abt6)) groups) will be mapped to the corresponding geonode groups, even creating these groups in geonode in case they do not exist yet. The geonode user is also made a member of these geonode groups.

Upon each login, the user's geonode group memberships are re-evaluated according to the information extracted from LDAP. The AUTH_LDAP_MIRROR_GROUPS_EXCEPT setting can be used to specify groups whose memberships will not be re-evaluated.

If no LDAP groups shall be mirrored LDAP_MIRROR_GROUPS and LDAP_MIRROR_GROUPS_EXCEPT must be set to False.

**Note:** Users mapped from LDAP will be marked with an `ldap` tag. This will be used to keep them in sync.

**Warning:** If you remove the `ldap` tag, the users will be threaten as pure internal GeoNode ones.

You may also manually generate the geonode groups in advance, before users login. In this case, when a user logs in and the mapped LDAP group already exists, the user is merely added to the geonode group

Be sure to check out `django_auth_ldap` for more information on the various configuration options.
Keep Users and Groups Synchronized

In order to constantly keep the remote LDAP Users and Groups **synchronized** with GeoNode, you will need to run periodically some specific management commands.

```
*/10 * * * * /opt/geonode/my-geonode/manage.sh updateldapgroups >> /var/log/cron.log 2>&1
*/10 * * * * /opt/geonode/my-geonode/manage.sh updateldapusers >> /var/log/cron.log 2>&1
```

Where the `manage.sh` is a bash script similar to the following one:

```
manage.sh
export $(grep -v '^#' /opt/geonode/my-geonode/.env | xargs -d ' ')/home/<my_user>/virtualenvs/geonode/bin/python /opt/geonode/my-geonode/manage.py $@
```

and the `/opt/geonode/my-geonode/.env` is something similar to the following one:

```
[ENV]

DEBUG=False
DJANGO_ALLOWED_HOSTS=<geonode_public_host>,localhost,127.0.0.1
DJANGO_DATABASE_URL=postgresql://my_geonode:**********@localhost:5432/my_geonode_db
DEFAULT_BACKEND_UPLOADER=geonode.importer
DEFAULT_FROM_EMAIL=geonode@example.org
DJANGO_EMAIL_HOST=smtp.example.org
DJANGO_EMAIL_HOST_PASSWORD=**********
DJANGO_EMAIL_HOST_USER=geonode
DJANGO_EMAIL_PORT=465
DJANGO_EMAIL_USE_SSL=True
DJANGO_SETTINGS_MODULE=my_geonode.settings
DJANGO_SECRET_KEY=**********
OAUTH2_API_KEY=**********
PROXY_URL=/proxy/?url=
EXIF_ENABLED=True
EMAIL_ENABLE=True
TIME_ENABLE=True
ACCOUNT_OPEN_SIGNUP=True
ACCOUNT_APPROVAL_REQUIRED=True
ACCOUNT_EMAIL_REQUIRED=True
ACCOUNT_EMAIL_VERIFICATION=optional
AVATAR_GRAVATAR_SSL=True
GEONODE_DB_URL=postgresql://my_geonode:**********@localhost:5432/my_geonode_data
GEOSERVER_ADMIN_PASSWORD=**********
GEOSERVER_LOCATION=https://<geonode_public_host>/geoserver/
GEOSERVER_PUBLIC_HOST=<geonode_public_host>
GEOSERVER_PUBLIC_LOCATION=https://<geonode_public_host>/geoserver/
GEOSERVER_WEB_UI_LOCATION=https://<geonode_public_host>/geoserver/
LDAP_SERVER_URL=ldap://<the_ldap_server>
LDAP_BIND_DN=uid=ldapinfo,cn=users,dc=ad,dc=example,dc=org
LDAP_BIND_PASSWORD=**********
LDAP_USER_SEARCH_DN=dc=ad,dc=example,dc=org
LDAP_USER_SEARCH_FILTERSTR=(&(uid=%%(user)s)(objectClass=person))
LDAP_MIRROR_GROUPS=True
```

(continues on next page)
LDAP_GROUP_SEARCH_DN=cn=groups,dc=ad,dc=example,dc=org
LDAP_GROUP_SEARCH_FILTERSTR=((cn=abt1)(cn=abt2)(cn=abt3)(cn=abt4)(cn=abt5)(cn=abt6))
LDAP_GROUP_PROFILE_MEMBER_ATTR=uniqueMember
OGC_REQUEST_MAX_RETRIES=3
OGC_REQUEST_POOL_CONNECTIONS=100
OGC_REQUEST_POOL_MAXSIZE=100
OGC_REQUEST_TIMEOUT=60
SITEURL=https://<geonode_public_host>/
SITE_HOST_NAME=<geonode_public_host>
FREETEXT_KEYWORDS_READONLY=False

# Advanced Workflow Settings
ADMIN_MODERATE_UPLOADS=False
GROUP_MANDATORY_RESOURCES=False
GROUP_PRIVATE_RESOURCES=False
RESOURCE_PUBLISHING=False

Note: You might want to use the same /opt/geonode/my-geonode/.env for your UWSGI configuration too:

```
[uwsgi]
socket = 0.0.0.0:8000
uid = <my_user>
gid = www-data

plugins = python3
virtualenv = /home/<my_user>/virtualenvs/geonode

# set environment variables from .env file
env LANG=en_US.utf8
env LC_ALL=en_US.UTF-8
env LC_LANG=en_US.UTF-8

for readline = /opt/geonode/my-geonode/.env
    env = %(_)
endfor =

chdir = /opt/geonode/my-geonode
module = my_geonode.wsgi:application

processes = 12
threads = 2
enable-threads = true
master = true

# logging
# path to where uwsgi logs will be saved
logto = /storage/my_geonode/logs/geonode.log
daemonize = /storage/my_geonode/logs/geonode.log
touch-reload = /opt/geonode/my-geonode/my_geonode/wsgi.py
buffer-size = 32768
max-requests = 500
harakiri = 300 # respawn processes taking more than 5 minutes (300 seconds)
```

(continues on next page)
1.24 GeoNode Admins Guide

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.

The following sections will explain more in depth what functionalities the admin panel makes available to you. It should be highlighted that the sections not covered in this guide are meant to be managed through GeoNode UI.

1.24.1 Accessing the panel

The Admin Panel is a model-centric interface where trusted users can manage content on GeoNode. Only the staff users can access the admin interface.

Note: The “staff” flag, which controls whether the user is allowed to log in to the admin interface, can be set by the admin panel itself.

The panel can be reached from Admin link of the User Menu in the navigation bar (see the picture below) or through this URL: http://<your_geonode_host>/admin.

Fig. 236: The Admin Link of the User Menu
When clicking on that link the Django-based Admin Interface page opens and shows you all the Django models registered in GeoNode.

![Django administration](image)

**Fig. 237:** The GeoNode Admin Interface

### 1.24.2 Reset or Change the admin password

From the Admin Interface you can access the Change password link by clicking on the username on the right side of the navigation bar, which will open a dropdown.

It allows you to access the Change Password Form through which you can change your password.

Once the fields have been filled out, click on Change my password to perform the change.
Fig. 238: *The Change Password Link*

Fig. 239: *The Change Password Form*
1.24.3 Simple Theming

GeoNode provides by default some theming options manageable directly from the Administration panel. Most of the times those options allows you to easily change the GeoNode look and feel without touching a single line of HTML or CSS.

As an administrator go to http://<your_geonode_host>/admin/geonode_themes/geonodethemecustomization/.

![Fig. 240: List of available Themes](image.jpg)

The panel shows all the available GeoNode themes, if any, and allows you to create new ones.

**Warning:** Only one theme at a time can be activated (aka enabled). By disabling or deleting all the available themes, GeoNode will turn the gui back to the default one.

Editing or creating a new Theme, will actually allow you to customize several properties.

At least you’ll need to provide a Name for the Theme. Optionally you can specify also a Description, which will allow you to better identify the type of Theme you created.

![Fig. 241: Theme Name and Description](image.jpg)

Just below the Description field, you will find the Enabled checkbox, allowing you to toggle the Theme.
Jumbotron and Get Started link

**Note:** Remember, everytime you want to apply some changes to the Theme, you **must** save the Theme and reload the GeoNode browser tab. In order to quickly switch back to the Home page, you can just click the VIEW SITE link on the top-right corner of the Admin dashboard.

![Admin Dashboard](image)

The next section, allows you to define the first important Theme properties. This part involves the GeoNode main page sections.

By changing those properties as shown above, you will easily change your default home page from this to this

It is possible to optionally **hide** the Jumbotron text.

**Slide show**

To switch between a slide show and a jumbotron, flip the value of the welcome theme from “slide show” to “jumbotron” and vice versa to either display a jumbotron with content or a slide show in the home page

For example, to display a slide show, change the welcome theme from jumbotron background to slide show

Before creating a slide show, make sure you have slides to select from (in the multi-select widget) to make up the slide show.

If no slides exist, click the plus (+) button beside the slide show multi-select widget to add a new slide.
Fig. 243: *Jumbotron and Logo options*

Fig. 244: *GeoNode Default Home*
Fig. 245: *Updating Jumbotron and Logo*

The Jumbotron background can be changed and Hide text in the jumbotron can be toggled. The Welcome theme can be chosen between jumbotron background and slide show.

Fig. 246: *Hide Jumbotron text*

The Welcome theme can be chosen between jumbotron background and slide show. The Jumbotron slide show can be selected by holding down "Control", or "Command" on a Mac, to select more than one.
Fill in the slide name, slide content using markdown formatting, and upload a slide image (the image that will be displayed when the slide is in view).

**Add jumbotron theme slide**

<table>
<thead>
<tr>
<th>Slide name</th>
<th>New Slide</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jumbotron slide background</td>
<td>Choose file</td>
</tr>
<tr>
<td>Jumbotron slide content</td>
<td># Write Markdown Here To Format Slide Content</td>
</tr>
<tr>
<td></td>
<td>This is a paragraph in a slide content</td>
</tr>
<tr>
<td></td>
<td>Fill in this section with markdown</td>
</tr>
<tr>
<td></td>
<td>Hide text in the jumbotron slide</td>
</tr>
<tr>
<td></td>
<td>Check this if the jumbotron background image already contains text</td>
</tr>
<tr>
<td></td>
<td>Is enabled</td>
</tr>
<tr>
<td></td>
<td>Disabling this slide will hide it from the slide show</td>
</tr>
</tbody>
</table>

For slide images that already contain text, hide slide content by checking the checkbox labeled “Hide text in the jumbotron slide” as shown below, then save the slide.

It is also possible to hide a slide from all slide show themes that use it by unchecking the checkbox labeled “Is enabled” as shown below.

Selecting the above slide in a slide show and enabling slide show (using the “welcome theme” configuration) will create a slide show with a slide as shown below:
Switching between different themes

In the case you have defined more Themes, switching between them is as easy as enabling one and disabling the others.

Remember to save the Themes everytime and refresh the GeoNode home page on the browser to see the changes.

It is also important that there is only one Theme enabled at a time.

In order to go back to the standard GeoNode behavior, just disable or delete all the available Themes.

1.24.4 Add a new user

In GeoNode, administrators can manage other users. For example, they can Add New Users through the following form.

The form above can be reached from the Admin Panel at the following path: Home > People > Users. Click on + Add user to open the form page.

It is also available, in the GeoNode UI, the Add User link of the About menu in the navigation bar.

To perform the user creation fill out the required fields (username and password) and click on Save. You will be redirected to the User Details Page which allows to insert further information about the user.

The user will be visible into the Users List Page of the Admin Panel and in the People Page (see Viewing other users information).
**Django administration**

**Add user**

![Add user form](image)

**Fig. 247: Adding New Users**

**Django administration**

**Users**

![Users list](image)

**Fig. 248: The Add User button in the Users List page**

**Fig. 249: Add User Link**
Fig. 250: The User Details Page
1.24.5 Activate/Disable a User

When created, new users are active by default. You can check that in the User Details Page from the Admin Panel (see the picture below).

Active users can interact with other users and groups, can manage resources and, more in general, can take actions on the GeoNode platform. Untick the Active checkbox to disable the user. It will be not considered as user by the GeoNode system.

1.24.6 Change a User password

GeoNode administrators can also change/reset the password for those users who forget it. As shown in the picture below, click on this form link from the User Details Page to access the Change Password Form.

The Change User Password Form should looks like the following one. Insert the new password two times and click on CHANGE PASSWORD.
Fig. 252: *New Users Active by default*

Fig. 253: *Disabled Users*
Fig. 254: Changing Users Passwords

Fig. 255: Changing Users Passwords
1.24.7 Promoting a User to Staff member or superuser

Active users have not access to admin tools. GeoNode makes available those tools only to Staff Members who have the needed permissions. Superusers are staff members with full access to admin tools (all permissions are assigned to them).

Administrators can promote a user to Staff Member by ticking the Staff status checkbox in the User Details Page. To make some user a Superuser, the Superuser status checkbox should be ticked. See the picture below.

![Fig. 256: Staff and Superuser permissions](image)

1.24.8 Creating a Group

In GeoNode is possible to create new groups with set of permissions which will be inherited by all the group members.

The creation of a Group can be done both on the GeoNode UI and on the Admin Panel, we will explain how in this paragraph.

The Create Groups link of About menu in the navigation bar allows administrators to reach the Group Creation Page. The following form will open.

Fill out all the required fields and click Create to create the group. The Group Details Page will open.

The new created group will be searchable in the Groups List Page.

**Note:** The Create a New Group button on the Groups List Page allows to reach the Group Creation Form.

As already mentioned above, groups can also be created from the Django-based Admin Interface of GeoNode.
The Groups link of the AUTHENTICATION AND AUTHORIZATION section allows to manage basic Django groups which only care about permissions.
To create a GeoNode group you should take a look at the GROUPS section.

As you can see, GeoNode provides two types of groups. You will learn more about that in the next paragraph.

Types of Groups

In GeoNode users can be grouped through a Group Profile, an enhanced Django group which can be enriched with some further information such as a description, a logo, an email address, some keywords, etc. It also possible to define some Group Categories based on which those group profiles can be divided and filtered.

A new Group Profile can be created as follow:
- click on the Group Profile + Add button
- fill out all the required fields (see the picture below), Group Profiles can be explicitly related to group categories
- click on Save to perform the creation, the new created group profile will be visible in the Group Profiles List

Group Categories

Group Profiles can also be related to Group Categories which represents common topics between groups. In order to add a new Group Category follow these steps:
- click on the Group Categories + Add group category button
- fill out the creation form (type name and description)
- click on Save to perform the creation, the new created category will be visible in the Group Categories List
Create a Group

Title

Logo

Choose File

No file chosen

Description

Email

Email used to contact one or all group members, such as a mailing list, shared email, or exchange group.

Keywords

A space or comma-separated list of keywords

Access

Public

Public: Any registered user can view and join a public group.

Public (invite-only): Any registered user can view the group. Only invited users can join.

Private: Registered users cannot see any details about the group, including membership. Only invited users can join.

Categories

test group category

Create

Fig. 258: The Group Creation Form
Cartographers

Last Modified: Nov. 29, 2021, 9:56 a.m.

This group consists of GeoNode Cartographers

cartography

cartographers@gmail.com

test group category

Edit Group Details
Manage Group Members
Delete this Group
Group Activities
Permissions
This group is Public (invite-only)
Anyone may view this group but membership is by invitation only.

Managers

admin

No Group

Members

admin

No Organization info

Fig. 259: The Group Details Page
Fig. 260: The Groups List Page

Fig. 261: The Groups Section on the Admin Panel
### Add group profile

<table>
<thead>
<tr>
<th>Title</th>
<th>Transportation planners</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slug</td>
<td>transportation-planners</td>
</tr>
<tr>
<td>Logo</td>
<td>Choose File: No file chosen</td>
</tr>
<tr>
<td>Description</td>
<td>Users interested in transport</td>
</tr>
<tr>
<td>Email</td>
<td><a href="mailto:transportgroup@gmail.com">transportgroup@gmail.com</a></td>
</tr>
<tr>
<td>Keywords</td>
<td></td>
</tr>
<tr>
<td>Access</td>
<td>Public</td>
</tr>
<tr>
<td>Categories</td>
<td>Test group category</td>
</tr>
</tbody>
</table>

**Group members**

<table>
<thead>
<tr>
<th>User</th>
<th>Role</th>
<th>Joined</th>
</tr>
</thead>
<tbody>
<tr>
<td>Johnsmith</td>
<td>Manager</td>
<td>2021-11-20 10:01:56</td>
</tr>
<tr>
<td>Joe</td>
<td>Member</td>
<td>2021-11-29 10:01:56</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2021-11-29 10:01:56</td>
</tr>
</tbody>
</table>

**Fig. 262: A new Group Profile**
Fig. 263: The Group Profiles List

Fig. 264: A new Group Category
1.24.9 Managing a Group

Through the Groups link of About menu in the navigation bar, administrators can reach the Groups List Page.

In that page all the GeoNode Group Profiles are listed.

For each group some summary information (such as the title, the description, the number of members and managers) are displayed near the Group Logo.

Administrators can manage a group from the Group Profile Details Page which is reachable by clicking on the title of the group.

As shown in the picture above, all information about the group are available on that page:

- the group Title;
- the Last Editing Date which shows a timestamp corresponding to the last editing of the group properties;
- the Keywords associated with the group;
Explore Groups

Total: 3

- **test group**
  Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum ...

  1 Member  1 Manager

- **Cartographers**
  This group consists of Geonode Cartographers

  1 Member  1 Manager

- **Transportation planners**
  Users interested in transport

  2 Members  1 Manager

Fig. 267: *Group Profiles List Page*
Cartographers
Last Modified: Nov. 29, 2021, 9:56 a.m.

This group consists of GeoNode Cartographers
- cartography
- cartographers@gmail.com
- test group category

Members
- admin
  - No Organization info
  - 0 2 4

Edit Group Details
Manage Group Members
Delete this Group
Group Activities
Permissions
This group is Public (invite-only)
Anyone may view this group but membership is by invitation only.

Managers
- admin
- No Group

Fig. 268: Group Profile Details Page
• **Permissions** on the group (Public, Public(invite-only), Private);

• **Members** who join the group;

• **Managers** who manage the group.

There are also four links:

• The *Edit Group Details* link opens the *Group Profile Form* through which the following properties can be changed:
  – **Title**.
  – **Logo** (see next paragraphs).
  – **Description**.
  – **Email**, to contact one or all group members.
  – **Keywords**, a comma-separated list of keywords.
  – **Access**, which regulates permissions:
    * **Public**: any registered user can view and join a public group.
    * **Public (invite-only)**: only invited users can join, any registered user can view the group.
    * **Private**: only invited users can join the group, registered users cannot see any details about the group, including membership.
  – **Categories**, the group categories the group belongs to.

• **Manage Group Members** (see next paragraphs).

• the *Delete this Group*, click on it to delete the Group Profile. GeoNode requires you to confirm this action.

![Confirm Group Deletion](image)

• the *Group Activities* drives you to the *Group Activities Page* where you can see all datasets, maps and documents associated with the group. There is also a *Comments* tab which shows comments on those resources.

**Group Logo**

Each group represents something in common between its members. So each group should have a *Logo* which graphically represents the idea that identify the group.

On the *Group Profile Form* page you can insert a logo from your disk by click on *Browse*…

Click on *Update* to apply the changes.
Take a look at your group now, you should be able to see that logo.
Activity Feed for Cartographers

Fig. 270: Group Activities

Update Group

Fig. 271: Editing the Group Logo
Managing Group members

The Manage Group Members link opens the Group Members Page which shows Group Members and Group Managers. Managers can edit group details, can delete the group, can see the group activities and can manage memberships. Other Members can only see the group activities.

In Public Groups, users can join the group without any approval. Other types of groups require the user to be invited by the group managers.

Only group managers can Add new members. In the picture below, you can see the manager can search for users by typing their names into the User Identifiers search bar. Once found, he can add them to the group by clicking the Add Group Members button. The Assign manager role flag implies that all the users found will become managers of the group.

The following picture shows you the results.

If you want to change the role of group members after adding them, you can use the “promote” button to make a member into a manager, and the “demote” button to make a manager into a regular member.
Fig. 273: *Adding a new Member to the Group*

Fig. 274: *New Members of the Group*
1.24.10 Group based advanced data workflow

By default GeoNode is configured to make every resource suddenly available to everyone, i.e. publicly accessible even from anonymous/non-logged in users.

It is actually possible to change few configuration settings in order to allow GeoNode to enable an advanced publication workflow.

With the advanced workflow enabled, your resources won’t be automatically published (i.e. made visible and accessible for all, contributors or simple users).

For now, your item is only visible by yourself, the manager of the group to which the resource is linked (this information is filled in the metadata), the members of this group, and the GeoNode Administrators.

Before being published, the resource will follow a two-stage review process, which is described below:

![Diagram](Fig. 275: From upload to publication: the review process on GeoNode)

**How to enable the advanced workflow**

You have to tweak the GeoNode settings accordingly.

Please see the details of the following GeoNode Settings:

- RESOURCE_PUBLISHING
- ADMIN_MODERATE_UPLOADS
- GROUP_PRIVATE_RESOURCES

Summarizing, when all the options above of the Advanced Workflow are enabled, upon a new upload we will have:

- The “unpublished” resources will be hidden to anonymous users only. The registered users will be still able to access the resources (if they have the rights to do that, of course).
- The “unpublished” resources will remain hidden to users if the permission (see Admin Guide section: ‘Manage Permissions’) will be explicitly removed.
During the upload, whenever the advanced workflow is enabled, the owner’s Groups are automatically allowed to access the resource, even if the “anonymous” flag has been disabled. Those permissions can be removed later on.

During the upload, “managers” of the owner’s Groups associated to the resource, are always allowed to edit the resource, the same as they are admin for that resource.

“managers” of the owner’s Groups associated to the resource are allowed to “publish” also the resources, not only to “approve” them.

Change the owner rights in case of advanced workflow is on

After switching ADMIN_MODERATE_UPLOADS to True and resource is approved owner is no longer able to modify it. He will see new button on the resource detail page: Request change. After clicking this, view with short form is shown. On this view user can write short message why he want to modify the resource. This message will be sent through messaging and email system to administrators:

After administrator unapprove the resource owner is again able to modify it.

The group Manager approval

Here, the role of the Manager of the group to which your dataset, document or map is linked is to check that the uploaded item is correct. Particularly, in the case of a dataset or a map, it consists of checking that the chosen cartographic representation and the style are fitting but also that the discretization is appropriate.

The Manager must also check that the metadata are properly completed and that the mandatory information (Title, Abstract, Edition, Keywords, Category, Group, Region) are filled.

If needed, the Manager can contact the contributor responsible of the dataset, document or map in order to report potential comments or request clarifications.

Members of the group can also take part in the reviewing process and give some potential inputs to the responsible of the dataset, document or map.

When the Manager considers that the resource is ready to be published, he should approve it. To do so, the Manager goes to the resource detail page, then opens the Edit Metadata. In the Settings tab, the manager checks the Approved box, and then updates the metadata and saves the changes:

Following this approval, the GeoNode Administrators receive a notification informing them that an item is now waiting for publication.

The publication by the GeoNode Administrator

Prior to the public release of an approved resource, the Administrator of the platform performs a final validation of the item and its metadata, notably to check that it is in line with license policies.

If needed, the GeoNode Administrator can contact the Manager who has approved the resource, as well as its responsible.

Once the resource is validated, the item is made public by the Administrator. It can now be viewed, accessed, and downloaded in accordance with the Permissions set by the responsible contributor.
Promotion, Demotion and Removal of Group Members

If the owner is a group Manager, They have permissions to edit, approve, and publish the resource.

When a group member is promoted to a manager role, they gain permissions to edit, approve and publish the resource.

When a group manager is demoted to a member role, they lose edit permissions of the resource and only remain with view and download permissions.

When a member is removed from the group, they can no longer access the unpublished resource anymore.

1.24.11 Manage profiles using the admin panel

So far GeoNode implements two distinct roles, that can be assigned to resources such as datasets, 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.

Is possible for an administrator to add new roles if needed, by clicking on the Add contact role button in the Base -> Contact Roles section:

Clicking on the People section (see figure) will open a web form 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.
1.24.12 Manage datasets using the admin panel

Some of the datasets information can be edited directly through the admin interface although the best place is in the Dataset -> Metadata Edit in GeoNode.

Clicking on the Admin > Dataset > Datasets link will show the list of available datasets.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Attributes</th>
<th>+</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Datasets</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td>Styles</td>
<td>+</td>
</tr>
</tbody>
</table>

**Warning:** It is not recommended to modify the Datasets’ Attributes or Styles directly from the Admin dashboard unless you are aware of your actions.

The Metadata information can be changed for multiple datasets at once through the Metadata batch edit action. Select the datasets you want to edit in the batch and at the bottom, enter the Metadata batch edit action then click Go.

This will open a form with the information you can edit in a batch. see picture below.

By clicking over one Dataset link, it will show a detail page allowing you to modify some of the resource info like the metadata, the keywords, the title, etc.

**Note:** It is strongly recommended to always use the GeoNode resource Edit Metadata or Advanced Metadata tools in order to edit the metadata info.

The Permissions can be changed also for multiple Datasets at once through the Set permissions action.

By clicking over one Dataset link, it will show a detail page allowing you to modify the permissions for the selected resources.
1.24.13 Manage the maps using the admin panel

Similarly to the Datasets, it is possible to manage the available GeoNode Maps through the Admin panel also.

Move to Admin > Maps to access the Maps list.

Notice that by enabling the Featured option here, will allow GeoNode to show the Map thumbnail and the Map detail link at the top under featured resources on the Home Page.

1.24.14 Manage the documents using the admin panel

Similarly to the Datasets and Maps, it is possible to manage the available GeoNode Documents through the Admin panel also.

Move to Admin > Documents to access the Documents list.

By clicking over one Document link, it will show a detail page allowing you to modify some of the resource info like the metadata, the keywords, the title, etc.
1.24.15 Manage the base metadata choices using the admin panel

Admin > Base contains almost all the objects you need to populate the resources metadata choices.

In other words the options available from the select-boxes of the resource Edit Metadata and Advanced Metadata forms.

**Note:** When editing the resource metadata through the Edit Metadata, some fields are marked as mandatory and by filling those information the Completeness progress will advance accordingly.

Even if not all the fields have been filled, the system won’t prevent you to update the metadata; this is why the Mandatory fields are mandatory to be fully compliant with an ISO 19115 metadata schema, but are only recommended to be compliant with GeoNode.

Also the Completeness indicates how far the metadata is to be compliant with an ISO 19115 metadata schema.
Of course, it is **highly** recommended to always fill as much as possible at least all the metadata fields marked as Mandatory.

This will improve not only the quality of the data stored into the system, but will help the users to easily search for them on GeoNode.

All the Search & Filter panels and options of GeoNode are, in fact, based on the resources metadata fields. Too much generic descriptions and too empty metadata fields, will give highly un-precise and very wide search results to the users.

**Hierarchical keywords**

Through the Admin > Base > **Hierarchical keywords** panel it will be possible to manage all the keywords associated to the resources.

- The **Name** is the human readable text of the keyword, what users will see.
- The **Slug** is a unique label used by the system to identify the keyword; most of the times it is equal to the name.

Notice that through the Position and Relative to selectors, it is possible to establish a hierarchy between the available keywords. The hierarchy will be reflected in the form of a tree from the metadata panels.

By default each user with editing metadata rights on any resource, will be able to insert new keywords into the system by simply typing a free text on the keywords metadata field.

It is possible to force the user to select from a fixed list of keywords through the **FREE-TEXT_KEYWORDS_READONLY** setting.
Fig. 278: Advanced Metadata Form
Fig. 280: Hierarchical keywords list

Fig. 281: Hierarchical keywords edit
When set to *True* keywords won’t be writable from users anymore. Only admins can will be able to manage them through the *Admin > Base > Hierarchical keywords* panel.

### Licenses

Through the *Admin > Base > Licenses* panel it will be possible to manage all the licenses associated to the resources.

![Metadata editor Licenses](image)

Fig. 282: *Metadata editor Licenses*

**Warning:** It is strongly recommended to not publish resources without an appropriate license. Always make sure the data provider specifies the correct license and that all the restrictions have been honored.
**Metadata Regions**

Through the *Admin > Base > Metadata Regions* panel it will be possible to manage all the admin areas associated to the resources.

Notice that those regions are used by GeoNode to filter search results also through the resource list view.

**Note:** GeoNode tries to guess the Regions intersecting the data bounding boxes when uploading a new dataset. Those should be refined by the user dataset on anyway.

---

**Metadata Restriction Code Types and Spatial Representation Types**

Through the *Admin > Base > Metadata Restriction Code Types* and *Admin > Base > Metadata Spatial Representation Types* panels, it will be possible to update only the metadata descriptions for restrictions and spatial representation types.

Such lists are read-only by default since they have been associated to the specific codes of the ISO 19115 metadata schema. Changing them would require the system to provide a custom dictionary through the metadata catalog too. Such functionality is not supported actually by GeoNode.
**Metadata Topic Categories**

Through the Admin > Base > Metadata Topic Categories panel it will be possible to manage all the resource metadata categories available into the system.

Notice that by default, GeoNode provides the standard topic categories available with the ISO 19115 metadata schema. Changing them means that the system won’t be compliant with the standard ISO 19115 metadata schema anymore. ISO 19115 metadata schema extensions are not currently supported natively by GeoNode.

It is worth notice that GeoNode allows you to associate Font Awesome Icons to each topic category through their fa-icon code. Those icons will be used by GeoNode to represent the topic category on both the Search & Filter menus and Metadata panels.

**Warning:** The list of the Metadata Topic Categories on the home page is currently fixed. To change it you will need to update or override the GeoNode index.html HTML template.

By default the Metadata Topic Categories are writable. Meaning that they can be removed or created by the Admin panel.

It is possible to make them fixed (it will be possible to update their descriptions and icons only) through the MODIFY_TOPICCATEGORY setting.

### 1.24.16 Announcements

As an Administrator you might need to broadcast announcements to the world about your portal or simply to the internal contributors.

GeoNode Announcements allow actually to do that; an admin has the possibility to create three types of messages, accordingly to their severity, decide their validity in terms of time period (start date and expiring date of the announcement), who can view them or not (everyone or just the registerd members) and whenever a user can hide the message or not and how long.

There are three types of announcements accordingly to their severity level: General, Warning and Critical The difference is mainly the color of the announcement box.

Only administrators and staff members can create and manage announcements.

Currently there two ways to access and manage the announcements list:

1. Via the GeoNode interface, from the Profile panel

   **Note:** Those are accessible by both admins and staff members.

2. Via the GeoNode Admin panel

   **Note:** Those are accessible by admins only.

The functionalities are almost the same for both the interfaces, except that from the Admin panel it is possible to manage the dismissals too.

Dismissals are basically records of members that have read the announcement and closed the message box. An announcement can have one dismissal type among the three below:

1. No Dismissal Allowed it won’t be possible to close the announcement’s message box at all.
2. **Session Only Dismissal** (*) the default one, it will be possible to close the announcement’s message box for the current browser session. It will show up again at next access.

3. **Permanent Dismissal Allowed** once the announcement’s message box is closed, it won’t appear again for the current member.

### How to create and manage Announcements

From the **Profile** panel, click on **Announcements** link

Click either on **New Announcement** to create a new one or over a title of an existing one to manage its contents.

Create a new announcement is quite straight; you have to fill the fields provided by the form.

*Warning:* In order to be visible, you will need to check the **Site wide** option in any case. You might want to hide the message to **anonymous** users by enabling the **Members only** option too.

Managing announcements form the **Admin** panel, is basically the same; the fields for the form will be exactly the same. Accessing announcements options from the **Admin** panel, allows you to manage dismissals also. Through this interface you will be able to selectively decide members which can or cannot view a specific announcement, or force them to visualize the messages again by deleting the dismissals accordingly.
Fig. 285: Announcements from the Admin panel

Fig. 286: Announcements List from the Profile panel
Create Announcement

Title

Level

- General

Content

- Site wide
- Members only

Dismissal type

- Session Only Dismissal

Publish_start


Publish_end

Fig. 287: Create Announcement from the Profile panel
Fig. 288: Create Announcement from the Admin panel

Fig. 289: Create Dismissal from the Admin panel
1.24.17 Menus, Items and Placeholders

GeoNode provides some integrated functionalities allowing you to quickly and easily customize the top-bar menu (see the example below).

![GeoNode Top-Bar Menu customization](image)

With minor changes of the `basic.html` template, potentially, it could be possible to use the same approach for a more complex customization. Let’s start with the simple one.

By default GeoNode provides custom placeholders already defined into the `basic.html` template, called `CARDS MENU`, `TOPBAR MENU RIGHT`, `TOPBAR MENU LEFT`, `TOPBAR MENU`.

From the `Admin > Base` panel, it is possible to access to the `Menu`, `Menu Items` and `Menu Placeholder` options.

The hierarchical structure of a custom `Menu` is the following one:

1. **Menu Placeholder:** first of all you need to define a placeholder both into the `Admin > Base` panel and the `basic.html` template, using the same keyword.
   By default GeoNode provides already defined menus.

2. **Menu:** second thing to do is to create a new menu associated to the corresponding placeholder.
   This is still possible from the `Admin > Base` panel
   You will need to provide:
   - A Title, representing the name of the Menu visible by the users

   **Warning:** By using this approach, internationalization won’t be supported. For the time being GeoNode does not support this for menus created from the `Admin > Base` panel.

   - A Menu Placeholder from the existing ones.
   - A Order in the case you’ll create more menus associated to the same placeholder.
Fig. 291: Menu, Menu Items and Menu Placeholder options on the Admin panel
Fig. 292: The default `TOPBAR_MENU` Menu Placeholder on the Admin panel

Fig. 293: Create a new Menu from the Admin panel
GeoNode Documentation, Release master

3. **Menu Item**: finally you will need to create voices belonging to the *menu*. For the time being, GeoNode allows you to create only `href` links.

![Django administration panel](image)

**Fig. 294: Create a new Menu Item from the Admin panel**

**Warning:** The *Menu* won’t be visible until you add more than one *Menu Item*. If you have 1, item, the item will be showed (but not under the menu).

1.24.18 OAuth2 Access Tokens

This small section won’t cover entirely the GeoNode OAuth2 security integration, this is explained in detail in other sections of the documentation (refer to oauth2_fixtures_and_migration and oauth2_tokens_and_sessions).

Here we will focus mainly on the `Admin > DJANGO/GEONODE/OAUTH TOOLKIT` panel items with a specific attention to the *Access tokens* management.

The `Admin > DJANGO/GEONODE/OAUTH TOOLKIT` panel (as shown in the figure below) allows an admin to manage everything related to GeoNode OAuth2 grants and permissions.

As better explained in other sections of the documentation, this is needed to correctly handle the communication between GeoNode and GeoServer.

Specifically from this panel an admin can create, delete or extend OAuth2 *Access tokens*.

The section `oauth2_tokens_and_sessions` better explains the concepts behind OAuth2 sessions; we want just to refresh the mind here about the basic concepts:

- If the `SESSION_EXPIRED_CONTROL_ENABLED` setting is set to *True* (by default it is set to *True*) a registered user cannot login to neither GeoNode nor GeoServer without a valid *Access token*.
- When logging-in into GeoNode through the sign-up form, GeoNode checks if a valid *Access token* exists and it creates a new one if not, or extends the existing one if expired.
- New *Access tokens* expire automatically after `ACCESS_TOKEN_EXPIRE_SECONDS` setting (by default 86400)
When an Access token expires, the user will be kicked out from the session and forced to login again.

Create a new token or extend an existing one

It is possible from the Admin > DJANGO/GEONODE OAUTH TOOLKIT panel to create a new Access token for a user.

In order to do that, just click on the Add button beside Access tokens topic.

On the new form, select the followings:

1. User: use the search tool in order to select the correct user. The form want the user PK, which is a number, and not the username. The search tool will do everything for you.

2. Source refresh token: this is not mandatory, leave it blank.

3. Token: write here any alphanumeric string. This will be the access_token that the member can use to access the OWS services. We suggest to use a service like https://passwordsgenerator.net/ in order to generate a strong token string.

4. Application: select GeoServer, this is mandatory.

5. Expires: select an expiration date by using the date-time widgets.

6. Scope: select write, this is mandatory.

Do not forget to Save.

From now on, GeoNode will use this Access Token to control the user session (notice that the user need to login again if closing the browser session), and the user will be able to access the OWS Services by using the new Access Token, e.g.:
Django OAuth Toolkit administration

Fig. 296: Add a new "Access token"

Add access token

Fig. 297: Create an "Access token"
1.24. GeoNode Admins Guide

Fig. 298: *Select a User*

Fig. 299: *Select a Token*
Add access token

User: 1000

Token: 123456

Id token: [redacted]

Application: GeoServer

Expires: 2022-01-04 11:00

Scope: write

Fig. 300: Select the GeoServer Application

Fig. 301: Select the Token Expiration

Fig. 302: Select the Application Scope
https://dev.geonode.geo-solutions.it/geoserver/ows?service=wms&version=1.3.0&request=GetCapabilities&access_token=123456

Notice the `request=GetCapabilities&access_token=123456` (access_token) parameter at the end of the URL.

### Force a User Session to expire

Everything said about the creation of a new Access Token, applies to the deletion of the latter.

From the same interface an admin can either select an expiration date or delete all the Access Tokens associated to a user, in order to force its session to expire.

Remember that the user could activate another session by logging-in again on GeoNode with its credentials.

In order to be sure the user won’t force GeoNode to refresh the token, reset first its password or de-activate it.

### 1.25 GeoNode Management Commands

Management commands are utility functions for GeoNode maintenance tasks. They are usually run from an SSH/bash shell on the server running GeoNode. Any call to python is prepended with a configuration parameter to indicate the GeoNode settings module to be used.

```
DJANGO_SETTINGS_MODULE=geonode.settings python manage.py migrate_baseurl --help
```

**Note:** If you have enabled `local_settings.py` the command will change as follows:

```
DJANGO_SETTINGS_MODULE=geonode.local_settings python manage.py migrate_baseurl --help
```

### 1.25.1 Migrate GeoNode Base URL

The `migrate_baseurl` Management Command allows you to fix all the GeoNode Links whenever, for some reason, you need to change the Domain Name of IP Address of GeoNode.

This **must** be used also in the cases you’ll need to change the network schema from HTTP to HTTPS, as an instance.

First of all let’s take a look at the `--help` option of the `migrate_baseurl` management command in order to inspect all the command options and features.

Run

```
DJANGO_SETTINGS_MODULE=geonode.settings python manage.py migrate_baseurl --help
```

This will produce output that looks like the following

```
```

(continues on next page)
Migrate GeoNode VM Base URL

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. "/home/djangoprojects/myproject".
--traceback Raise on CommandError exceptions
--no-color Don't colorize the command output.
-f, --force Forces the execution without asking for confirmation.
--source-address SOURCE_ADDRESS Source Address (the one currently on DB e.g. http://192.168.1.23)
--target-address TARGET_ADDRESS Target Address (the one to be changed e.g. http://my-public.geonode.org)

• Example 1: I want to move my GeoNode instance from http://127.0.0.1 to http://example.org

Warning: Make always sure you are using the correct settings

DJANGO_SETTINGS_MODULE=geonode.settings python manage.py migrate_baseurl -- --source-address=127.0.0.1 --target-address=example.org

• Example 2: I want to move my GeoNode instance from http://example.org to https://example.org

Warning: Make always sure you are using the correct settings

DJANGO_SETTINGS_MODULE=geonode.settings python manage.py migrate_baseurl -- --source-address=http://example.org --target-address=https://example.org

• Example 3: I want to move my GeoNode instance from https://example.org to https://geonode.example.org

Warning: Make always sure you are using the correct settings

DJANGO_SETTINGS_MODULE=geonode.settings python manage.py migrate_baseurl -- --source-address=example.org --target-address=geonode.example.org
Note: After migrating the base URL, make sure to sanitize the links and catalog metadata also (Update Permissions, Metadata, Legends and Download Links).

1.25.2 Update Permissions, Metadata, Legends and Download Links

The following three utility Management Commands, allow to fix up:

1. Users/Groups Permissions on Datasets; those will be refreshed and synchronized with the GIS Server ones also
2. Metadata, Legend and Download links on Datasets and Maps
3. Cleanup Duplicated Links and Outdated Thumbnails

Management Command `sync_geonode_datasets`

This command allows to sync already existing permissions on Datasets. In order to change/set Datasets’ permissions refer to the section Batch Sync Permissions

The options are:

- **filter**: Only update data the Dataset names that match the given filter.
- **username**: Only update data owned by the specified username.
- **updatepermissions**: Update the Dataset permissions; synchronize it back to the GeoSpatial Server. This option is also available from the Layer Details page.
- **updateattributes**: Update the Dataset attributes; synchronize it back to the GeoSpatial Server. This option is also available from the Layer Details page.
- **updatethumbnails**: Update the Dataset thumbnail. This option is also available from the Layer Details page.
- **updatebbox**: Update the Dataset BBOX and LoLan BBOX. This option is also available from the Layer Details page.
- **remove-duplicates**: Removes duplicated Links.

First of all let’s take a look at the –help option of the `sync_geonode_datasets` management command in order to inspect all the command options and features.

Run

```
DJANGO_SETTINGS_MODULE=geonode.settings python manage.py sync_geonode_datasets --help
```

Note: If you enabled local_settings.py the command will change as following:

```
DJANGO_SETTINGS_MODULE=geonode.local_settings python manage.py sync_geonode_datasets --help
```

This will produce output that looks like the following

```
usage: manage.py sync_geonode_datasets [-h] [--version] [-v {0,1,2,3}]
   [--settings SETTINGS] [--pythonpath PYTHONPATH] [--traceback]
   [--no-color] [-i] [-d] [-f FILTER]
```
Update the GeoNode Datasets: permissions (including GeoFence database), statistics, thumbnails

optional arguments:
- `--version` show program's version number and exit
- `--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. "/home/djangoprojects/myproject".
- `--traceback` Raise on CommandError exceptions
- `--no-color` Don't colorize the command output.
- `--ignore-errors` Stop after any errors are encountered.
- `--remove-duplicates` Remove duplicates first.
- `--filter FILTER` Only update data the Datasets that match the given filter.
- `--username USERNAME` Only update data owned by the specified username.
- `--updatepermissions` Update the Dataset permissions.
- `--updatethumbnails` Update the Dataset styles and thumbnails.
- `--updateattributes` Update the Dataset attributes.
- `--updatebbox` Update the Dataset BBOX.

1. **Example 1**: I want to update/sync all Datasets permissions and attributes with the GeoSpatial Server

   ```
   Warning: Make always sure you are using the correct settings
   DJANGO_SETTINGS_MODULE=geonode.settings python manage.py sync_geonode__datasets --updatepermissions --updateattributes
   ```

2. **Example 2**: I want to regenerate the Thumbnails of all the Datasets belonging to afabiani

   ```
   Warning: Make always sure you are using the correct settings
   DJANGO_SETTINGS_MODULE=geonode.settings python manage.py sync_geonode__datasets -u afabiani --updatethumbnails
   ```
Management Command `sync_geonode_maps`

This command is basically similar to the previous one, but affects the Maps; with some limitations.

The options are:

- **filter**: Only update data the maps titles that match the given filter.
- **username**: Only update data owned by the specified username.
- **updatethumbnails**: Update the map styles and thumbnails. This option is also available from the Map Details page.
- **remove-duplicates**: Removes duplicated Links.

First of all let's take a look at the `--help` option of the `sync_geonode_maps` management command in order to inspect all the command options and features.

Run

```
DJANGO_SETTINGS_MODULE=geonode.settings python manage.py sync_geonode_maps --help
```

**Note**: If you enabled `local_settings.py` the command will change as following:

```
DJANGO_SETTINGS_MODULE=geonode.local_settings python manage.py sync_geonode_maps --help
```

This will produce output that looks like the following

```
usage: manage.py sync_geonode_maps [-h] [--version] [-v {0,1,2,3}] 
[--settings SETTINGS] 
[--pythonpath PYTHONPATH] [--traceback] 
[--no-color] [-i] [-d] [-f FILTER] 
[-u USERNAME] [--updatethumbnails]
```

Update the GeoNode maps: permissions, thumbnails

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. 
  
  
  "'/home/djangoprojects/myproject".
- `--traceback` Raise on CommandError exceptions
- `--no-color` Don't colorize the command output.
- `-i, --ignore-errors` Stop after any errors are encountered.
- `-d, --remove-duplicates` Remove duplicates first.
- `-f FILTER, --filter FILTER`

(continues on next page)
Only update data the maps that match the given filter.
-\texttt{u} \texttt{USERNAME}, --\texttt{username} \texttt{USERNAME} 
Only update data owned by the specified username.
--\texttt{updatethumbnails} Update the map styles and thumbnails.

\begin{itemize}
\item \textbf{Example 1}: I want to regenerate the Thumbnail of the Map \texttt{This is a test Map}
\end{itemize}

\begin{verbatim}
Warning: Make always sure you are using the correct settings

DJANGO_SETTINGS_MODULE=geonode.settings python manage.py sync_geonode_maps
\rightarrow --updatethumbnails -f 'This is a test Map'
\end{verbatim}

\textbf{Management Command set\_all\_datasets\_metadata}

This command allows to reset \textbf{Metadata Attributes} and \textbf{Catalogue Schema} on Datasets. The command will also update the \textit{CSW Catalogue XML} and Links of GeoNode.

The options are:

\begin{itemize}
\item \texttt{filter}; Only update data the Datasets that match the given filter.
\item \texttt{username}; Only update data owned by the specified username.
\item \texttt{remove-duplicates}; Update the map styles and thumbnails.
\item \texttt{delete-orphaned-thumbs}; Removes duplicated Links.
\item \texttt{set-uuid}; will refresh the UUID based on the UUID\_HANDLER if configured (Default False).
\item \texttt{set_attrib}; If set will refresh the attributes of the resource taken from Geoserver. (Default True).
\item \texttt{set_links}; If set will refresh the links of the resource. (Default True).
\end{itemize}

First of all let's take a look at the \texttt{--help} option of the \texttt{set\_all\_datasets\_metadata} management command in order to inspect all the command options and features.

Run

\begin{verbatim}
DJANGO_SETTINGS_MODULE=geonode.settings python manage.py set_all_datasets_metadata --help
\end{verbatim}

\textbf{Note}: If you enabled local\_settings\_py the command will change as following:

\begin{verbatim}
DJANGO_SETTINGS_MODULE=geonode.local_settings python manage.py set_all_datasets_metadata --help
\end{verbatim}

This will produce output that looks like the following

1.25. GeoNode Management Commands
usage: manage.py set_all_datasets_metadata [-h] [--version] [-v {0,1,2,3}]  
                                      [--settings SETTINGS]  
                                      [--pythonpath PYTHONPATH]  
                                      [--traceback] [--no-color] [-i] [-d]  
                                      [-t] [-f FILTER] [-u USERNAME]

Resets Metadata Attributes and Schema to All Datasets

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.  
                                      "/home/djangoprojects/myproject".
--traceback                  Raise on CommandError exceptions
--no-color                   Don't colorize the command output.
-i, --ignore-errors          Stop after any errors are encountered.
-d, --remove-duplicates      Remove duplicates first.
-t, --delete-orphaned-thumbs Delete Orphaned Thumbnails.
-f FILTER, --filter FILTER   Only update data the Datasets that match the given  
                                      filter
-u USERNAME, --username USERNAME  
                                      Only update data owned by the specified username

• Example 1: After having changed the Base URL, I want to regenerate all the Catalogue Schema and eventually remove all duplicates.

[Warning: Make always sure you are using the correct settings]

DJANGO_SETTINGS_MODULE=geonode.settings python manage.py set_all_datasets_metadata -d
**Management Command regenerate_xml**

The command will regenerate the CSW Catalogue XML metadata files.

The main options are:

- **layer**: Only process specified layers.
- **dry-run**: Do not actually perform any change.

When run with the `--help` option the full list of available options will be presented.

```bash
DJANGO_SETTINGS_MODULE=geonode.settings python manage.py regenerate_xml --help
```

This will produce the following output:

```bash
usage: manage.py regenerate_xml [-h] [-l LAYERS] [--skip-logger-setup] [-d] [--version] [˓→]
     [-v {0,1,2,3}] [--settings SETTINGS] [--pythonpath PYTHONPATH] [--traceback] [--no-
     --color] [--force-color] [-->]
     [--skip-checks]

Re-create XML metadata documents

options:
-h, --help           show this help message and exit
-l LAYERS, --layer LAYERS
   Only process specified layers
--skip-logger-setup  Skips setup of the "geonode.br" logger, "br" handler and "br"␣
   format if not present in settings
-d, --dry-run        Do not actually perform any change
--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. "/home/
   djangoprojects/myproject".
--traceback          Raise on CommandError exceptions.
--no-color           Don't colorize the command output.
--force-color        Force colorization of the command output.
--skip-checks        Skip system checks.

• Example:

```bash
Warning: Make sure you are using the correct settings
```

```bash
DJANGO_SETTINGS_MODULE=geonode.settings python manage.py regenerate_xml -d
```
1.25.3 Loading Data into GeoNode

There are situations where it is not possible or not convenient to use the *Upload Form* to add new Datasets to GeoNode via the web interface. For instance:

- The dataset is too big to be uploaded through a web interface.
- Import data from a mass storage programmatically.
- Import tables from a database.

This section will walk you through the various options available to load data into your GeoNode from GeoServer, from the command-line or programmatically.

**Warning:** Some parts of this section have been taken from the GeoServer project and training documentation.

**Management Command `importlayers`**

The `geonode.geoserver` Django app includes 2 management commands that you can use to load data in your GeoNode.

Both of them can be invoked by using the `manage.py` script.

First of all let’s take a look at the `--help` option of the `importlayers` management command in order to inspect all the command options and features.

Run

```
DJANGO_SETTINGS_MODULE=geonode.settings python manage.py importlayers --help
```

**Note:** If you enabled `local_settings.py` the command will change as following:

```
DJANGO_SETTINGS_MODULE=geonode.local_settings python manage.py importlayers --help
```

This will produce output that looks like the following:

```
usage: manage.py importlayers [-h] [-hh HOST] [-u USERNAME] [-p PASSWORD]  
[---version] [-v {0,1,2,3}] [---settings SETTINGS]  
[--pythonpath PYTHONPATH] [---traceback] [---no-color]  
[--force-color] [---skip-checks]  
[path [path ...]]
```

Brings files from a local directory, including subfolders, into a GeoNode site. The datasets are added to the Django database, the GeoServer configuration, and the pycsw metadata index. At this moment only files of type Esri Shapefile (.shp) and GeoTiff (.tif) are supported.

In order to perform the import, GeoNode must be up and running.

**Positional arguments:**

- `path`: path [path...]

**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.
   "/home/djangoprojects/myproject".
-hh HOST, --host HOST  Geonode host url
-u USERNAME, --username USERNAME  Geonode username
-p PASSWORD, --password PASSWORD  Geonode password

While the description of most of the options should be self explanatory, its worth reviewing some of the key options a
bit more in details.

- The -hh Identifies the GeoNode server where we want to upload our Datasets. The default value is
- The -u Identifies the username for the login. The default value is admin.
- The -p Identifies the password for the login. The default value is admin.

The import Datasets management command is invoked by specifying options as described above and specifying the
path to a directory that contains multiple files. For purposes of this exercise, let's use the default set of testing Datasets
that ship with geonode. You can replace this path with a directory to your own shapefiles.

This command will produce the following output to your terminal

```
san_andres_y_providencia_poi.shp: 201
san_andres_y_providencia_location.shp: 201
san_andres_y_providencia_administrative.shp: 201
san_andres_y_providencia_coastline.shp: 201
san_andres_y_providencia_highway.shp: 201
single_point.shp: 201
san_andres_y_providencia_water.shp: 201
san_andres_y_providencia_natural.shp: 201

1.7456605294117646 seconds per Dataset
```

Output data: {
   "success": [
      "san_andres_y_providencia_poi.shp",
      "san_andres_y_providencia_location.shp",
      "san_andres_y_providencia_administrative.shp",
      "san_andres_y_providencia_coastline.shp",
      "san_andres_y_providencia_highway.shp",
      "single_point.shp",
      "san_andres_y_providencia_water.shp",
      "san_andres_y_providencia_natural.shp"
   ]
}
As output the command will print:
The status code, is the response coming from GeoNode. For example 201 means that the Dataset has been correctly uploaded.

If you encounter errors while running this command, please check the GeoNode logs for more information.

**Management Command `updatelayers`**

While it is possible to import Datasets directly from your server’s 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, some of them may not be supported as GeoNode upload formats. You can add them to your GeoNode by following the procedure described below.

GeoServer supports 4 types of data: **Raster**, **Vector**, **Databases** and **Cascaded**.

For a list of the supported formats for each type of data, consult the following pages:

- [https://docs.geoserver.org/latest/en/user/data/raster/index.html](https://docs.geoserver.org/latest/en/user/data/raster/index.html)
- [https://docs.geoserver.org/latest/en/user/data/cascaded/index.html](https://docs.geoserver.org/latest/en/user/data/cascaded/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 them. Please consult the GeoServer documentation for more information.

**Data from a PostGIS database**

Let’s walk through an example of configuring a new PostGIS database in GeoServer and then configuring those Datasets 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/](http://localhost:8080/geoserver/web/)

1. 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.

   **Note:** The number of stores, Datasets and workspaces may be different depending on what you already have configured in your GeoServer.

2. 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.
GeoServer Documentation, Release master

Welcome
This GeoServer belongs to...

Layer Preview
Import Data
Workspaces
Stores
Layers
Layer Groups
Styles
Backup & Restore

Welcome

New data source

Choose the type of data source you wish to configure

Vector Data Sources

- CSV - Comma delimited text file
- Directory of spatial files (shapefiles) - Takes a directory of shapefiles and exposes it as a data store
- GeoPackage - GeoPackage
- H2 - H2 Embedded Database
- H2 (JNDI) - H2 Embedded Database (JNDI)
- Microsoft SQL Server (JNDI) - Microsoft SQL Server (JNDI)
- Microsoft SQL Server (JTDIS Driver) - Microsoft SQL Server (JTDIS Driver)
- Microsoft SQL Server (JTDIS Driver) - Microsoft SQL Server (JTDIS Driver)
- Oracle NG (JNDI) - Oracle Database (JNDI)
- PostGIS - PostGIS Database
- PostgreSQL (JNDI) - PostgreSQL Database (JNDI)
- Properties - Allows access to Java Property files containing Feature information
- Shapefile - ESRI(tm) Shapefiles (*.shp)
- Web Feature Server (NG) - Provides access to the Features published a Web Feature Service, and the

1.25. GeoNode Management Commands
3. 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.

4. The next screen lets you configure the Datasets in your database. This will of course be different depending on the Datasets in your database.

5. Select the “Publish” button for one of the Datasets and the next screen will be displayed where you can enter metadata for this Dataset. Since we will be managing this metadata in GeoNode, we can leave these alone for now.
6. 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.

7. Click save and this Dataset will now be configured for use in your GeoServer.

8. The next step is to configure these Datasets in GeoNode. The updatelayers management command can be used for this purpose. As with importlayers, it’s useful to look at the command line options for this command by passing the –help option.

   Run

   ```bash
   DJANGO_SETTINGS_MODULE=geonode.settings python manage.py updatelayers --help
   
   Note: If you enabled local_settings.py the command will change as following:
   ```

   ```bash
   DJANGO_SETTINGS_MODULE=geonode.local_settings python manage.py updatelayers --help
   ```
Keywors

Current Keywords

<table>
<thead>
<tr>
<th>Feature</th>
<th>Remove selected</th>
</tr>
</thead>
<tbody>
<tr>
<td>_1_SARMIENTO_ENERO_2018</td>
<td></td>
</tr>
</tbody>
</table>

New Keyword

<table>
<thead>
<tr>
<th>Vocabulary</th>
</tr>
</thead>
</table>

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

Data links

No data links so far

Add link

Coordinate Reference Systems

Native SRS

EPSG:4326

Find... EPSG/WGS 84...

Declared SRS

EPSG:4326

SRS handling

Force declared

Bounding Boxes

Native Bounding Box

<table>
<thead>
<tr>
<th>Min X</th>
<th>Min Y</th>
<th>Max X</th>
<th>Max Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>-59.078485273</td>
<td>-45.060003889</td>
<td>-45.057928671999</td>
<td>45.579002872</td>
</tr>
</tbody>
</table>

Compute from data

Compute from SRS bounds

Lat/Lon Bounding Box

<table>
<thead>
<tr>
<th>Min X</th>
<th>Min Y</th>
<th>Max X</th>
<th>Max Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>-59.078485273</td>
<td>-45.060003889</td>
<td>-45.057928671999</td>
<td>45.579002872</td>
</tr>
</tbody>
</table>

Compute from native bounds

Curved geometries control

Layer Preview

List of all layers configured in GeoServer and provides previews in various formats for each.

<table>
<thead>
<tr>
<th>Type</th>
<th>Title</th>
<th>Common Formats</th>
<th>All Formats</th>
</tr>
</thead>
<tbody>
<tr>
<td>_1_SARMIENTO_ENERO_2018</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1.25. GeoNode Management Commands
This will produce output that looks like the following

```
usage: manage.py updatelayers [-h] [--version] [-v {0,1,2,3}]
                        [--settings SETTINGS] [--pythonpath PYTHONPATH]
                        [--traceback] [--no-color] [-i]
                        [--skip-unadvertised]
                        [--skip-geonode-registered] [--remove-deleted]
                        [-u USER] [-f FILTER] [-s STORE] [-w WORKSPACE]
                        [-p PERMISSIONS]

Update the GeoNode application with data from GeoServer

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,
                        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 CommandError exceptions
  --no-color            Don't colorize the command output.
  -i, --ignore-errors   Stop after any errors are encountered.
  --skip-unadvertised   Skip processing unadvertised Datasets from GeoSever.
  --skip-geonode-registered
                        Just processing GeoServer Datasets still not in GeoNode.
  -u USER, --user USER  Name of the user account which should own the
                        imported Datasets
  -f FILTER, --filter FILTER
                        Only update data the Datasets that match the given
                        filter
  -s STORE, --store STORE
                        Only update data the Datasets for the given
                        geoserver store name
  -w WORKSPACE, --workspace WORKSPACE
                        Only update data on specified workspace
  -p PERMISSIONS, --permissions PERMISSIONS
                        Permissions to apply to each Dataset
```

The update procedure includes the following steps:
• The process fetches from GeoServer the relevant WMS layers (all, by store or by workspace)
• If a filter is defined, the GeoServer layers are filtered
• For each of the layers, a GeoNode dataset is created based on the metadata registered on GeoServer (title, abstract, bounds)
• New layers are added, existing layers are replaced, unless the \texttt{–skip-geonode-registered} option is used
• The GeoNode layers, added in previous runs of the update process, which are no longer available in GeoServer are removed, if the \texttt{–remove-delete} option is set

\textbf{Warning:} One of the \texttt{–workspace} or \texttt{–store} must be always specified if you want to ingest Datasets belonging to a specific \texttt{Workspace}. As an instance, in order to ingest the Datasets present into the \texttt{geonode} workspace, you will need to specify the option \texttt{–w geonode}.

9. Let's ingest the Dataset \texttt{geonode:_1_SARMIENTO_ENERO_2018} from the \texttt{geonode} workspace.

```
DJANGO_SETTINGS_MODULE=geonode.settings python manage.py updatelayers -w \texttt{geonode} -f \texttt{_1_SARMIENTO_ENERO_2018}
```

Inspecting the available Datasets in GeoServer ...

```
Found 1 Datasets, starting processing
/usr/local/lib/python2.7/site-packages/owslib/iso.py:117: FutureWarning:
the .identification and .serviceidentification properties will merge into .identification being a list of properties. This is currently implemented in .identificationinfo. Please see https://github.com/geopython/OWSLib/issues/38 for more information
```

```
/usr/local/lib/python2.7/site-packages/owslib/iso.py:495: FutureWarning:
The .keywords and .keywords2 properties will merge into the .keywords property in the future, with .keywords becoming a list of MD_Keywordsinstances. This is currently implemented in .keywords2. Please see https://github.com/geopython/OWSLib/issues/301 for more information
```

Subject: [master.demo.geonode.org] A new Dataset has been uploaded
From: webmaster@localhost
To: mapadeldelito@chubut.gov.ar
Reply-To: webmaster@localhost
Date: Tue, 08 Oct 2019 12:26:17 -0000
Message-ID: <20191008122617.28801.94967@d3cf85425231>

<body>
You have received the following notice from master.demo.geonode.org:
<p>
The user <i><a href="http://master.demo.geonode.org/people/profile/admin">admin</a></i> uploaded the following Dataset:<br/>
<strong>_1_SARMIENTO_ENERO_2018</strong><br/>
You can visit the Dataset's detail page here: http://master.demo.geonode.org
</p>
```

(continues on next page)
To change how you receive notifications, please go to http://master.demo.geonode.org.

You can visit the Dataset's detail page here: http://master.demo.geonode.org/Datasets/geonode:_1_SARMIENTO_ENERO_2018

To change how you receive notifications, please go to http://master.demo.geonode.org.

You have received the following notice from master.demo.geonode.org:

The user <i><a href="http://master.demo.geonode.org/people/profile/admin">admin</a></i> uploaded the following Dataset:<br/>

You can visit the Dataset's detail page here: http://master.demo.geonode.org/Datasets/geonode:_1_SARMIENTO_ENERO_2018

To change how you receive notifications, please go to http://master.demo.geonode.org.
You have received the following notice from master.demo.geonode.org:

The user <i><a href="http://master.demo.geonode.org/people/profile/admin">admin</a></i> uploaded the following Dataset:<br/>
<strong>_1_SARMIENTO_ENERO_2018</strong><br/>
You can visit the Dataset's detail page here: http://master.demo.geonode.org/Datasets/geonode:_1_SARMIENTO_ENERO_2018

To change how you receive notifications, please go to http://master.demo.geonode.org

---

Found geoserver resource for this Dataset: _1_SARMIENTO_ENERO_2018
... Creating Default Resource Links for Layer [geonode:_1_SARMIENTO_ENERO_2018]
-- Resource Links[Prune old links]...
-- Resource Links[Prune old links]...done!
-- Resource Links[Compute parameters for the new links]...
-- Resource Links[Create Raw Data download link]...
-- Resource Links[Create Raw Data download link]...done!
-- Resource Links[Set download links for WMS, WCS or WFS and KML]...
-- Resource Links[Set download links for WMS, WCS or WFS and KML]...done!
-- Resource Links[Legend link]...
-- Resource Links[Legend link]...done!
-- Resource Links[Thumbnail link]...
-- Resource Links[Thumbnail link]...done!
-- Resource Links[OWS Links]...
-- Resource Links[OWS Links]...done!

1.25. GeoNode Management Commands
You can visit the Dataset's detail page here: http://master.demo.geonode.org/Datasets/geonode:_1_SARMIENTO_ENERO_2018

To change how you receive notifications, please go to http://master.demo.geonode.org

1.25. GeoNode Management Commands
You have received the following notice from master.demo.geonode.org:

The following Dataset was updated:

The following Dataset was updated: _1_SARMIENTO_ENERO_2018, owned by admin

You can visit the Dataset's detail page here: http://master.demo.geonode.org/Datasets/geonode:_1_SARMIENTO_ENERO_2018

To change how you receive notifications, please go to http://master.demo.geonode.org

---

Found geoserver resource for this Dataset: _1_SARMIENTO_ENERO_2018

FutureWarning: The behavior of this method will change in future versions. Use specific 'len(elem)' or 'elem is not None' test instead.

if not user_style:
    FutureWarning: The behavior of this method will change in future versions. Use specific 'len(elem)' or 'elem is not None' test instead.

if user_style:
    ... Creating Default Resource Links for Layer [geonode:_1_SARMIENTO_ENERO_2018]
    -- Resource Links[Prune old links]...
    -- Resource Links[Prune old links]...done!
    -- Resource Links[Compute parameters for the new links]...
    -- Resource Links[Create Raw Data download link]...
    -- Resource Links[Create Raw Data download link]...done!
    -- Resource Links[Set download links for WMS, WCS or WFS and KML]...
    -- Resource Links[Set download links for WMS, WCS or WFS and KML]...done!
    -- Resource Links[Legend link]...
    -- Resource Links[Legend link]...done!
    -- Resource Links[Thumbnail link]...
    -- Resource Links[Thumbnail link]...done!
    -- Resource Links[OWS Links]...
    -- Resource Links[OWS Links]...done!
[created] Layer _1_SARMIENTO_ENERO_2018 (1/1)

Finished processing 1 Datasets in 5.0 seconds.

1 Created Datasets
0 Updated Datasets
Note: In case you don’t specify the -f option, the Datasets that already exists in your GeoNode will be just updated and the configuration synchronized between GeoServer and GeoNode.

Warning: When updating from GeoServer, the configuration on GeoNode will be changed!

Using GDAL and OGR to convert your Data for use in GeoNode

GeoNode supports uploading data in ESRI shapefiles, GeoTIFF, CSV, GeoJSON, ASCII-GRID and KML/KMZ formats (for the last three formats only if you are using the geonode.importer backend).

- If your data is in other formats, you will need to convert it into one of these formats for use in GeoNode.
- If your Raster data is not correctly processed, it might be almost unusable with GeoServer and GeoNode. You will need to process it using GDAL.

You 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=sampledata

You can download the Admin;(Postcode) Dataset by issuing the following command:

```
```

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:

```
|-- ANZVI08030083025.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
```

(continues on next page)
First, let's inspect this file set using the following command:

```bash
$ ogrinfo -so vicgrid94/mif/lga_polygon/macedon\ ranges/VMADMIN/POSTCODE_POLYGON.mid

```

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.
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 Dataset.

Next, let's go ahead and convert this Dataset into a shapefile by issuing the following command:

```bash
$ ogr2ogr -t_srs EPSG:4326 postcode_polygon.shp vicgrid94/mif/lga_polygon/macedon\ ranges/VMADMIN/POSTCODE_POLYGON.mid
```

Note that we have also reprojected the Dataset to the WGS84 spatial reference system with the `-t_srs` ogr2ogr option.

The output of this command will look like the following:
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:

```bash
$ 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.

As soon as the import process completes, you will have the possibility to go straight to the Dataset info page (“Layer Info” button), or to edit the metadata for that Dataset (“Edit Metadata” button), or to manage the styles for that Dataset (“Manage Styles”).
1.25. GeoNode Management Commands
GDAL (Raster Data)

Let's see several examples on how to either convert raster data into different formats and/or process it to get the best performances.

References:
   a) https://geoserver.geo-solutions.it/edu/en/raster_data/processing.html
   b) https://geoserver.geo-solutions.it/edu/en/raster_data/advanced_gdal/

Raster Data Conversion: Arc/Info Binary and ASCII Grid data into GeoTIFF format.

Let's assume we have a sample ASCII Grid file compressed as an archive.

```
# Un-tar the files
$ 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 = (239681.000000000000000,6050551.000000000000000)
Pixel Size = (100.000000000000000,-100.000000000000000)
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:

<table>
<thead>
<tr>
<th>Driver: AIG/Arc/Info Binary Grid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Files: batemans_ele</td>
</tr>
<tr>
<td>batemans_ele/dblbnd.adf</td>
</tr>
<tr>
<td>batemans_ele/hdr.adf</td>
</tr>
<tr>
<td>batemans_ele/metadata.xml</td>
</tr>
<tr>
<td>batemans_ele/prj.adf</td>
</tr>
<tr>
<td>batemans_ele/sta.adf</td>
</tr>
<tr>
<td>batemans_ele/w001001.adf</td>
</tr>
<tr>
<td>batemans_ele/w001001x.adf</td>
</tr>
</tbody>
</table>

Size is 155, 142  
Coordinate System is:  
PROJCS["unnamed",  
    GEOCS["GD94",  
    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",10000000],  
    UNIT["METERS",1]]  
Origin = (239681.000000000000000,6050551.000000000000000)  
Pixel Size = (100.000000000000000,-100.000000000000000)  
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)  
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:

```bash  
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 = (0.001011114155919,-0.001011114155919)
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
```

**Raster Data Optimization: Optimizing and serving big raster data**

(ref: https://geoserver.geo-solutions.it/edu/en/raster_data/advanced_gdal/example5.html)

When dealing with big raster datasets it could be very useful to use tiles.

Tiling allows large raster datasets to be broken-up into manageable pieces and are fundamental in defining and implementing a higher level raster I/O interface.

In this example we will use the original dataset of the *chiangMai_ortho_optimized* public raster Dataset which is currently available on the Thai CHIANG MAI Urban Flooding GeoNode platform.

This dataset contains an orthorectified image stored as RGBA GeoTiff with 4 bands, three bands for the RGB and one for transparency (the alpha channel).

Calling the gdalinfo command to see detailed information:

```
gdalinfo chiangMai_ortho.tif
```

It will produce the following results:
Driver: GTiff/GeoTIFF
Files: chiangMai_ortho.tif
Size is 63203, 66211
Coordinate System is:
PROJCS["WGS 84 / UTM zone 47N",
    GEOGCS["WGS 84",
        DATUM["WGS_1984",
            SPHEROID["WGS 84",6378137.298.257223563,
                AUTHORITY["EPSG","7030"]],
            AUTHORITY["EPSG","6326"]],
        PRIMEM["Greenwich",0,
            AUTHORITY["EPSG","8901"]],
        UNIT["degree",0.0174532925199433,
            AUTHORITY["EPSG","9122"]],
        AUTHORITY["EPSG","4326"]],
    PROJECTION["Transverse_Mercator"],
    PARAMETER["latitude_of_origin",0],
    PARAMETER["central_meridian",99],
    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","32647"]]
Origin = (487068.775, 2057413.890)
Pixel Size = (0.028850000000000, -0.028850000000000)
Metadata:
AREA_OR_POINT=Area
TIFFTAG_SOFTWARE=pix4dmapper
Image Structure Metadata:
COMPRESSION=LZW
INTERLEAVE=PIXEL
Corner Coordinates:
Upper Left ( 487068.775, 2057413.890) (98d52'38.72"E, 18d36'27.34"N)
Lower Left ( 487068.775, 2055593.702) (98d52'38.77"E, 18d35'25.19"N)
Upper Right ( 488892.181, 2057413.890) (98d53'40.94"E, 18d36'27.38"N)
Lower Right ( 488892.181, 2055593.702) (98d53'40.98"E, 18d35'25.22"N)
Center ( 487980.478, 2056458.796) (98d53' 9.85"E, 18d35'56.28"N)
Band 1 Block=63203x1 Type=Byte, ColorInterp=Red
NoData Value=-10000
Mask Flags: PER_DATASET ALPHA
Band 2 Block=63203x1 Type=Byte, ColorInterp=Green
NoData Value=-10000
Mask Flags: PER_DATASET ALPHA
Band 3 Block=63203x1 Type=Byte, ColorInterp=Blue
NoData Value=-10000
Mask Flags: PER_DATASET ALPHA
Band 4 Block=63203x1 Type=Byte, ColorInterp=Alpha
NoData Value=-10000

As you can see, this GeoTiff has not been tiled. For accessing subsets though, tiling can make a difference. With tiling,
data are stored and compressed in blocks (tiled) rather than line by line (stripped).

In the command output above it is visible that each band has blocks with the same width of the image (63203) and a unit length. The grids in the picture below show an image with equally sized tiles (left) and the same number of strips (right). To read data from the red subset, the intersected area will have to be decompressed.

In the tiled image we will have to decompress only 16 tiles, whereas in the stripped image on the right we’ll have to decompress many more strips.

Drone images data usually have a stripped structure so, in most cases, they need to be optimized to increase performances.

Let’s take a look at the gdal_translate command used to optimize our GeoTiff:

```
gdal_translate -co TILED=YES -co COMPRESS=JPEG -co PHOTOMETRIC=YCBCR
   --config GDAL_TIFF_INTERNAL_MASK YES -b 1 -b 2 -b 3 -mask 4
   chiangMai_ortho.tif
   chiangMai_ortho_optimized.tif
```

**Note:** For the details about the command parameters see [https://geoserver.geo-solutions.it/edu/en/raster_data/advanced_gdal/example5.html](https://geoserver.geo-solutions.it/edu/en/raster_data/advanced_gdal/example5.html)

Once the process ended, call the gdalinfo command on the resulting tif file:

```
gdalinfo chiangMai_ortho_optimized.tif
```

The following should be the results:

```
Driver: GTiff/GeoTIFF
Files: chiangMai_ortho_optimized.tif
Size is 63203, 66211
Coordinate System is:
PROJCS["WGS 84 / UTM zone 47N",
   GEOGCS["WGS 84",
   DATUM["WGS_1984",
```
Our GeoTiff is now tiled with 256x256 tiles, has 3 bands and a 1-bit mask for nodata.

We can also add internal overviews to the file using the gdaladdo command:

```
gdaladdo -r average chiangMai_ortho_optimized.tif 2 4 8 16 32 64 128 256 512
```

Overviews are duplicate versions of your original data, but resampled to a lower resolution, they can also be compressed with various algorithms, much in the same way as the original dataset.

By default, overviews take the same compression type and transparency masks of the input dataset (applied through

---

1.25. GeoNode Management Commands 482
the gdal_translate command), so the parameters to be specified are:

- `-r average`: computes the average of all non-NODATA contributing pixels
- `2 4 8 16 32 64 128 256 512`: the list of integral overview levels to build (from gdal version 2.3 levels are no longer required to build overviews)

Calling the gdalinfo command again:

```
gdalinfo chiangMai_ortho_optimized.tif
```

It results in:

```
| Driver: GTiff/GeoTIFF                  |
| Files: chiangMai_ortho_optimized.tif   |
| Size is 63203, 66211                   |
| Coordinate System is:                 |
| PROJCS["WGS 84 / UTM zone 47N",      |
|   GEOGCS["WGS 84",                  |
|     DATUM["WGS_1984",               |
|       SPHEROID["WGS 84",6378137.298.257223563, |
|         AUTHORITY["EPSG","7030"], |
|         AUTHORITY["EPSG","6326"], |
|        PRIMEM["Greenwich",0,        |
|          AUTHORITY["EPSG","8901"], |
|          UNIT["degree",0.0174532925199433, |
|            AUTHORITY["EPSG","9122"], |
|            AUTHORITY["EPSG","4326"], |
|        PROJECTION["Transverse_Mercator"], |
|        PARAMETER["latitude_of_origin",0], |
|        PARAMETER["central_meridian",99], |
|        PARAMETER["scale_factor",0.9996], |
|        PARAMETER["false_easting",5000000], |
|        PARAMETER["false_northing",0], |
|        UNIT["metre",1, |
|          AUTHORITY["EPSG","9001"], |
|          AXIS["East",EAST], |
|          AXIS["North",NORTH], |
|           AUTHORITY["EPSG","32647"]]] |
| Origin = (487068.775, 2057413.890) ( 98d52'38.72"E, 18d36'27.34"N) |
| Pixel Size = (0.0288500000000000,-0.0288500000000000) |
| Metadata: |
| AREA OR POINT/Area |
| TIFTAG_SOFTWARE=pix4dmapper |
| Image Structure Metadata: |
| COMPRESSION=YCbCr JPEG |
| INTERLEAVE=PIXEL |
| SOURCE_COLOR_SPACE=YCbCr |
| Corner Coordinates: |
| Upper Left ( 487068.775, 2057413.890 ) ( 98d52'38.72"E, 18d36'27.34"N) |
| Lower Left ( 487068.775, 2055503.702 ) ( 98d52'38.77"E, 18d35'25.19"N) |
| Upper Right ( 488892.181, 2057413.890 ) ( 98d53'40.94"E, 18d36'27.38"N) |
| Lower Right ( 488892.181, 2055503.702 ) ( 98d53'40.98"E, 18d35'25.22"N) |
| Center ( 487980.478, 2056458.796 ) ( 98d53' 9.85"E, 18d35'56.28"N) |
| Band 1 Block=256x256 Type=Byte, ColorInterp=Red |
```

(continues on next page)
NoData Value=-10000
Overviews: 31602x33106, 15801x16553, 7901x8277, 3951x4139, 1976x2070, 988x1035, 494x518, 247x259, 124x130
Mask Flags: PER_DATASET
Overviews of mask band: 31602x33106, 15801x16553, 7901x8277, 3951x4139, 1976x2070, 988x1035, 494x518, 247x259, 124x130
Band 2 Block=256x256 Type=Byte, ColorInterp=Green
NoData Value=-10000
Overviews: 31602x33106, 15801x16553, 7901x8277, 3951x4139, 1976x2070, 988x1035, 494x518, 247x259, 124x130
Mask Flags: PER_DATASET
Overviews of mask band: 31602x33106, 15801x16553, 7901x8277, 3951x4139, 1976x2070, 988x1035, 494x518, 247x259, 124x130
Band 3 Block=256x256 Type=Byte, ColorInterp=Blue
NoData Value=-10000
Overviews: 31602x33106, 15801x16553, 7901x8277, 3951x4139, 1976x2070, 988x1035, 494x518, 247x259, 124x130
Mask Flags: PER_DATASET
Overviews of mask band: 31602x33106, 15801x16553, 7901x8277, 3951x4139, 1976x2070, 988x1035, 494x518, 247x259, 124x130

Notice that the transparency masks of internal overviews have been applied (their compression does not show up in the file metadata).

UAVs usually provide also two other types of data: DTM (Digital Terrain Model) and DSM (Digital Surface Model).

Those data require different processes to be optimized. Let’s look at some examples to better understand how to use gdal to accomplish that task.

From the CHIANG MAI Urban Flooding GeoNode platform it is currently available the chiangMai_dtm_optimized Dataset, let’s download its original dataset.

This dataset should contain the DTM file chiangMai_dtm.tif.

Calling the gdalinfo command on it:

gdalinfo chiangMai_dtm.tif

The following information will be displayed:

Driver: GTiff/GeoTIFF
Files: chiangMai_dtm.tif
Size is 12638, 13240
Coordinate System is:
PROJCS["WGS 84 / UTM zone 47N",
    GEOGCS["WGS 84",
        DATUM["WGS_1984",
            SPHEROID["WGS 84", 6378137, 298.257223563,
                AUTHORITY["EPSG","7030"]],
            AUTHORITY["EPSG","6326"]],
        PRIMEM["Greenwich", 0,
            AUTHORITY["EPSG","8901"]],
        UNIT["degree", 0.0174532925199433,
            AUTHORITY["EPSG","9122"]],]
AUTHORITY["EPSG","4326"],
PROJECTION["Transverse_Mercator"],
PARAMETER["latitude_of_origin",0],
PARAMETER["central_meridian",99],
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","32647"]
Origin = (487068.774750000040513,2057413.889810000080615)
Pixel Size = (0.144270000000000,-0.144270000000000)
Metadata:
AREA_OR_POINT=Area
TIFFTAG_SOFTWARE=pix4dmapper
Image Structure Metadata:
COMPRESSION=LZW
INTERLEAVE=BAND
Corner Coordinates:
Upper Left ( 487068.775, 2057413.890) ( 98d52'38.72"E, 18d36'27.34"N)
Lower Left ( 487068.775, 2055503.755) ( 98d52'38.77"E, 18d35'25.19"N)
Upper Right ( 488892.059, 2057413.890) ( 98d53'40.94"E, 18d36'27.37"N)
Lower Right ( 488892.059, 2055503.755) ( 98d53'40.98"E, 18d35'25.22"N)
Center ( 487980.417, 2056458.822) ( 98d53' 9.85"E, 18d35'56.28"N)
Band 1 Block=12638x1 Type=Float32, ColorInterp=Gray
NoData Value=-10000

Reading this image could be very slow because it has not been tiled yet. So, as discussed above, its data need to be stored and compressed in tiles to increase performances.

The following gdal_translate command should be appropriate for that purpose:

```bash
gdal_translate -co TILED=YES -co COMPRESS=DEFLATE chiangMai_dtm.tif chiangMai_dtm_optimized.tif
```

When the data to compress consists of imagery (e.g., aerial photographs, true-color satellite images, or colored maps) you can use lossy algorithms such as JPEG. We are now compressing data where the precision is important, the band data type is Float32 and elevation values should not be altered, so a lossy algorithm such as JPEG is not suitable. JPEG should generally only be used with Byte data (8 bit per channel) so we have chosen the lossless DEFLATE compression through the COMPRESS=DEFLATE creation option.

Calling the gdalinfo command again:

```bash
gdalinfo chiangMai_dtm_optimized.tif
```

We can observe the following results:

```text
Driver: GTiff/GeoTIFF
Files: chiangMai_dtm_optimized.tif
Size is 12638, 13240
Coordinate System is:
PROJCS["WGS 84 / UTM zone 47N",]
```
GEOGCS["WGS 84", DATUM["WGS_1984", Spheroid["WGS 84", 6378137.298, 257223563, AUTHORITY["EPSG", "7030"], AUTHORITY["EPSG", "6326"], PRIMEM["Greenwich", 0, AUTHORITY["EPSG", "8901"], UNIT["degree", 0.0174532925199433, AUTHORITY["EPSG", "9122"], AUTHORITY["EPSG", "4326"], PROJECTION["Transverse_Mercator"], PARAMETER["latitude_of_origin", 0], PARAMETER["central_meridian", 99], 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", "32647"]],
 Origin = (487068.775, 2057413.890) (98°52'38.72"E, 18°36'27.34"N)
 Pixel Size = (0.144270000000000, -0.144270000000000)
 Metadata:
 AREA_OR_POINT=Area
 TIFFTAG_SOFTWARE=pix4dmapper
 Image Structure Metadata:
 COMPRESSION=DEFLATE
 INTERLEAVE=BAND
 Corner Coordinates:
 Upper Left (487068.775, 2057413.890) (98°52'38.72"E, 18°36'27.34"N)
 Lower Left (487068.775, 2055503.755) (98°52'38.77"E, 18°35'25.19"N)
 Upper Right (488892.059, 2057413.890) (98°53'40.94"E, 18°36'27.37"N)
 Lower Right (488892.059, 2055503.755) (98°53'40.98"E, 18°35'25.22"N)
 Center (487980.417, 2056458.822) (98°53'9.85"E, 18°35'56.28"N)
 Band 1 Block=256x256 Type=Float32, ColorInterp=Gray
 NoData Value=-10000

We need also to create overviews through the gdaladdo command:

```
gdaladdo -r nearest chiangMai_dtm_optimized.tif 2 4 8 16 32 64
```

Unlike the previous example, overviews will be created with the **nearest resampling algorithm**. That is due to the nature of the data we are representing: we should not consider the average between two elevation values but simply the closer one, it is more reliable regarding the conservation of the original data.

Calling the gdalinfo command again:

```
gdalinfo chiangMai_dtm_optimized.tif
```

We can see the following information:
Driver: GTiff/GeoTIFF
Files: chiangMai_dtm_optimized.tif
Size is 12638, 13240
Coordinate System is:
PROJCS["WGS 84 / UTM zone 47N",
  GEOGCS["WGS 84",
    DATUM["WGS_1984",
      SPHEROID["WGS 84",6378137,298.257223563,1],
      AUTHORITY["EPSG","7030"],
      AUTHORITY["EPSG","6326"],
      PRIMEM["Greenwich",0,
        AUTHORITY["EPSG","8901"],
      UNIT["degree",0.0174532925199433,
        AUTHORITY["EPSG","9122"],
        AUTHORITY["EPSG","4326"],
      PROJECTION["Transverse_Mercator"],
      PARAMETER["latitude_of_origin",0],
      PARAMETER["central_meridian",99],
      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","32647"],
      ORIGIN = (487068.774750000040513,2057413.889810000080615)
    ],
  UNIT["metre",1,
    AUTHORITY["EPSG","9001"],
  AXISS["Easting",EAST],
  AXIS["Northing",NORTH],
  AUTHORITY["EPSG","32647"],
  ORIGIN = (487068.774750000040513,2057413.889810000080615)
  ],
  Pixel Size = (0.144270000000000,-0.144270000000000)
Metadata:
AREA_OR_POINT=Area
TIFFTAG_SOFTWARE=pix4dmapper
Image Structure Metadata:
COMPRESSION=DEFLATE
INTERLEAVE=BAND
Corner Coordinates:
Upper Left ( 487068.775, 2057413.890) ( 9d52'38.72"E, 18d36'27.34"N)
Lower Left ( 487068.775, 2055503.755) ( 9d52'38.77"E, 18d35'25.19"N)
Upper Right ( 488892.059, 2057413.890) ( 9d53'40.94"E, 18d36'27.37"N)
Lower Right ( 488892.059, 2055503.755) ( 9d53'40.98"E, 18d35'25.22"N)
Center ( 487980.417, 2056458.822) ( 9d53' 9.85"E, 18d35'56.28"N)
Band 1 Block=256x256 Type=Float32, ColorInterp=Gray
NoData Value=-10000
Overviews: 6319x6620, 3160x3310, 1580x1655, 790x828, 395x414, 198x207
Overviews have been created. By default, they inherit the same compression type of the original dataset (there is no evidence of it in the gdalinfo output).
Other Raster Data Use Cases

- Serving a large number of GrayScale GeoTiff with Palette
- Serving a large number of DTM ASCII Grid Files
- Serving a large number of Cartographic Black/White GeoTiff with Palette
- Serving a large number of satellite/aerial RGB GeoTiff with compression
- Optimizing and serving UAV data
- Optimizing and serving 16-bits satellite/aerial RGB GeoTiff

Process Raster Datasets Programmatically

In this section we will provide a set of shell scripts which might be very useful to batch process a lot of raster datasets programmatically.

1. process_gray.sh

```bash
for filename in *.tif; do
echo gdal_translate -co TILED=YES -co COMPRESS=DEFLATE $filename ${filename//.tif/.optimized.tif};
done > gdal_translate.sh
chmod +x gdal_translate.sh
./gdal_translate.sh
```

2. process_rgb.sh

```bash
for filename in *.tif; do
echo gdal_translate -co TILED=YES -co COMPRESS=JPEG -co PHOTOMETRIC=YCBCR -b 1 -b 2 -b 3 $filename ${filename//.tif/.optimized.tif};
done > gdal_translate.sh
chmod +x gdal_translate.sh
./gdal_translate.sh
```

3. process_rgb_alpha.sh

```bash
for filename in *.tif; do
echo gdal_translate -co TILED=YES -co COMPRESS=JPEG -co PHOTOMETRIC=YCBCR --config GDAL_TIFF_INTERNAL_MASK YES $filename ${filename//.tif/.optimized.tif};
done > gdal_translate.sh
chmod +x gdal_translate.sh
./gdal_translate.sh
```

(continues on next page)
4. process_rgb_palette.sh

```bash
for filename in *.optimized.tif*; do echo gdal_translate -co TILED=YES -co COMPRESS=DEFLATE $filename ${filename//.tif/.optimized.tif}; done > gdal_translate.sh
chmod +x gdal_translate.sh
./gdal_translate.sh
```

1.25.4 Thesaurus Import and Export

See Import via the load_thesaurus command and Exporting a thesaurus as RDF via the dump_thesaurus command.

1.25.5 Create Users and Super 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:

```bash
$ DJANGO_SETTINGS_MODULE=geonode.settings python manage.py createsuperuser
```

Note: If you enabled local_settings.py the command will change as following:

```bash
$ DJANGO_SETTINGS_MODULE=geonode.local_settings 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, Datasets, 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:

Menu

- Upload Layers
- Profile
- Recent Activity
- Inbox
- Announcements
- Remote Services
- Invite User
- GeoServer
- Admin
- Help

Clicking on Admin causes the interface to show up.

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.

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.
## Django administration

### Site administration

#### Account
- Account deletions
- Accounts
- Signup codes

#### Actstream
- Actions
- Follows

#### Announcements
- Announcements
- Dismissals

#### Auth
- Groups
- Users

#### Avatar
- Avatars

#### Base
- Contact roles
- Links
- Metadata Regions

### 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.
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*.

   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.

   [Image of Django form]

   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).

   [Image of GeoNode website]

   Hit the button and again a form will appear for you to fill out. This user will be named *geonode_user*

   By hitting *Sign up* the user will be signed up, as default only with the status *active*.
1.25.6 Batch Sync Permissions

GeoNode provides a very useful management command `set_layers_permissions` allowing an administrator to easily add / remove permissions to groups and users on one or more Datasets.

The `set_layers_permissions` command arguments are:

- **permissions** to set/unset -> read, download, edit, manage

```python
READ_PERMISSIONS = ['view_resourcebase']
DOWNLOAD_PERMISSIONS = ['view_resourcebase', 'download_resourcebase']
EDIT_PERMISSIONS = ['view_resourcebase', 'change_dataset_style', 'download_resourcebase', 'change_resourcebase_metadata', 'change_dataset_data', 'change_resourcebase']
MANAGE_PERMISSIONS = ['delete_resourcebase', 'change_resourcebase', 'view_resourcebase', 'change_resourcebase_permissions', 'change_dataset_style', 'change_resourcebase_metadata', 'publish_resourcebase']
```

(continues on next page)
NB: the above permissions list may change with the ADVANCED_WORKFLOW enabled. For additional info: https://docs.geonode.org/en/master/admin/admin_panel/index.html#how-to-enable-the-advanced-workflow

- **resources** (Datasets) which permissions will be assigned on –> type the Dataset id, multiple choices can be typed with comma separator, if no ids are provided all the Datasets will be considered
- **users** who permissions will be assigned to, multiple choices can be typed with a comma separator
- **groups** who permissions will be assigned to, multiple choices can be typed with a comma separator
- **delete** flag (optional) which means the permissions will be unset

**Usage examples:**

1. Assign `edit` permissions on the Datasets with id 1 and 2 to the users `username1` and `username2` and to the group `group_name1`.

   ```
   python manage.py set_layers-permissions -p edit -u username1,username2 -g group_name1 -r 1,2
   ```

2. Assign `manage` permissions on all the Datasets to the group `group_name1`.

   ```
   python manage.py set_layers-permissions -p manage -g group_C
   ```

3. Unset `download` permissions on the Dataset with id 1 for the user `username1`.

   ```
   python manage.py set_layers-permissions -p download -u username1 -r 1 -d
   ```

The same functionalities, with some limitations, are available also from the Admin Dashboard >> Users or Admin Dashboard >> Groups >> Group profiles.

An action named Set layer permissions is available from the list, redirecting the administrator to a form to set / unset read, edit, download permissions on the selected Users/group profile.

Is enough to select the dataset and press “Submit”. If the async mode is activated, the permission assign is asynchronous.

### 1.25.7 Delete Certain GeoNode Resources

The `delete_resources Management Command` allows to remove resources meeting a certain condition, specified in a form of a serialized django Q() expression.

First of all let’s take a look at the `--help` option of the `delete_resources` management command in order to inspect all the command options and features.

Run

```
DJANGO_SETTINGS_MODULE=geonode.settings python manage.py delete_resources --help
```

**Note:** If you enabled `local_settings.py` the command will change as following:
GeoNode Documentation, Release master

1.25. GeoNode Management Commands

Users

<table>
<thead>
<tr>
<th>ID</th>
<th>Username</th>
<th>Organization Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>AnonymousUser</td>
<td>-</td>
</tr>
<tr>
<td>1000</td>
<td>admin</td>
<td>-</td>
</tr>
<tr>
<td>1010</td>
<td>new.user</td>
<td>-</td>
</tr>
<tr>
<td>1004</td>
<td>username1</td>
<td>-</td>
</tr>
<tr>
<td>1009</td>
<td>username2</td>
<td>-</td>
</tr>
</tbody>
</table>

Set layer permissions
Batch Edit

Layers

stations
stations0
stations1
stations2

Permission type

Read
Edit
Download

Mode

Set
Unset

Cancel Submit

DJANGO_SETTINGS_MODULE=geonode.local_settings python manage.py delete_resources --help

This will produce output the following output:

usage: manage.py delete_resources [-h] [-c CONFIG_PATH]
    [-l LAYER_FILTERS [LAYER_FILTERS ...]]
    [-m MAP_FILTERS [MAP_FILTERS ...]]
    [-d DOCUMENT_FILTERS [DOCUMENT_FILTERS ...]]
    [--version] [-v {0,1,2,3}]
    [--settings SETTINGS]
    [--pythonpath PYTHONPATH] [--traceback]
    [--no-color] [--force-color]

Delete resources meeting a certain condition

optional arguments:
  -h, --help            show this help message and exit
  -c CONFIG_PATH, --config CONFIG_PATH
                         Configuration file path. Default is: delete_resources.json
  -l LAYER_FILTERS [LAYER_FILTERS ...], --layer_filters LAYER_FILTERS [LAYER_FILTERS ...]
  -m MAP_FILTERS [MAP_FILTERS ...], --map_filters MAP_FILTERS [MAP_FILTERS ...]
  -d DOCUMENT_FILTERS [DOCUMENT_FILTERS ...], --document_filters DOCUMENT_FILTERS...
  --version             show program's version number and exit
  -v {0,1,2,3}, --verbosity {0,1,2,3}

(continues on next page)
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 CommandError exceptions

--no-color  Don't colorize the command output.

--force-color  Force colorization of the command output.

There are two ways to declare Q() expressions filtering which resources should be deleted:

1. With a JSON configuration file: passing -c argument specifying the path to the JSON configuration file.
   
   • **Example 1**: Relative path to the config file (to manage.py)

   ```
   DJANGO_SETTINGS_MODULE=geonode.settings python manage.py delete_resources -c geonode/base/management/commands/delete_resources.json
   ```

   • **Example 2**: Absolute path to the config file

   ```
   DJANGO_SETTINGS_MODULE=geonode.settings python manage.py delete_resources -c /home/User/Geonode/configs/delete_resources.json
   ```

2. With CLI: passing -l -d -m list arguments for each of resources (Datasets, documents, maps)
   
   • **Example 3**: Delete resources without configuration file

   ```
   DJANGO_SETTINGS_MODULE=geonode.settings python manage.py delete_resources -l 'Q(pk__in=[1, 2]) | Q(title__icontains:"italy")' -d '*' -m "Q(pk__in=[1, 2])"
   ```

**Configuration File**

The JSON configuration file should contain a single filters object, which consists of Dataset, map and document lists. Each list specifies the filter conditions applied to a corresponding queryset, defining which items will be deleted. The filters are evaluated and directly inserted into Django .filter() method, which means the filters occurring as separated list items are treated as AND condition. To create OR query | operator should be used. For more info please check Django documentation(https://docs.djangoproject.com/en/3.2/topics/db/queries/#complex-lookups-with-q-objects)). The only exception is passing a list with '*' which will cause deleting all the queryset of the resource.

• **Example 4**: Example content of the configuration file, which will delete Datasets with ID’s 1, 2, and 3, those owned by admin user, along with all defined maps.

```json
{
  "filters": {
    "Dataset": [
      "Q(pk__in=[1, 2, 3]) | Q(title__icontains='italy')",
      "Q(user__name=admin)"
    ]
  }
}
```
CLI

The CLI configuration can be specified with --l -d -m list arguments, which in fact are a translation of the configuration JSON file. --l -d -m arguments are evaluated in the same manner as filters.Dataset, filters.map and filter.document accordingly from the Example 4. The following example's result will be equivalent to Example 4:

- **Example 5**: Example CLI configuration, which will delete Datasets with ID's 1, 2, and 3, along with all maps.

```bash
DJANGO_SETTINGS_MODULE=geonode.settings python manage.py delete_resources --l 'Q(pk__in: [1, 2, 3]) | Q(title__icontains:"italy")' --m 'Q(owner__name=admin)' --m '*'
```

1.25.8 Async execution over http

It is possible to expose and run management commands over http.

To run custom django management commands usually we make use of the command line:

```bash
python manage.py ping_mngmt_commands_http
$> pong
```

The `management_commands_http` app allows us to run commands when we have no access to the command line. It's possible to run a command using the API or the django admin GUI.

For security reasons, only admin users can access the feature and the desired command needs to be explicitly exposed. By default the following commands are exposed: `ping_mngmt_commands_http`, `updatelayers`, `sync_geonode_datasets`, `sync_geonode_maps`, `importlayers` and `set_all_datasets_metadata`.

To expose more command you can change the environment variable `MANAGEMENT_COMMANDS_EXPOSED_OVER_HTTP` and the added commands will be exposed in your application.

The list of exposed commands is available by the endpoint `list_management_commands` and also presented by the form in the admin page create management command job.

**Note:** To use the commands in an asynchronous approach `ASYNC_SIGNALS` needs to be set to True and celery should be running.
Manage using django admin interface

Creating a job

Access the admin panel: http://<your_geonode_host>/admin and go to “Management command jobs”.

![Management command admin section](image)

You will arrive at http://<your_geonode_host>/en/admin/management_commands_http/managementcommandjob/, then click on the button + Add management command job (http://<your_geonode_host>/en/admin/management_commands_http/managementcommandjob/add/).

![Add management command job](image)

Select the command and fill the form, with the arguments and/or key-arguments if needed. Save your job and in the list select the start action, alternatively you can mark the autostart option and the command will be automatic started when created.

Starting a job

To start a job:

1. Select the job to be started.
2. Select the start action.
3. Click in Go.
Fig. 305: Creating a management command job form

Fig. 306: Starting a job
4. The page will refresh and the job status will have changed. If it takes a long to run, refresh the page to see the updated status.

5. A stop option is also available.

**Note:** If it takes too long to load the page, ASYNC_SIGNALS may not be activated. If its status gets stuck at QUEUED, verify if celery is running and properly configured.

### Job status

Clicking at the link in the ID of a job, we can see the details of this job. For the job we just created, we can verify the output message and celery job status.

![Example job status](image)

**Fig. 307: Example job status**

When we have an error during execution the traceback message will be available in the *Celery traceback*. In the next image a `ping_mngmt_commands_http` job was created with the arguments `['--force_exception', true]`. Checking the text in this field can be useful when troubleshooting errors.

![Example job traceback message](image)

**Fig. 308: Example job traceback message**
Manage using API endpoints

The execution of the management commands can be handled by http requests to an API: http://<your_geonode_host>/api/v2/management/.

All the requests need to be authenticated with administrative permissions (superuser).

You can find here a postman collection with all the examples listed here and other available endpoints:
geonode_mngmt_commands.postman_collection.json

List exposed commands

Getting a list of the exposed commands:

```
curl --location --request GET http://<your_geonode_host>/api/v2/management/commands/ --header 'Authorization: Basic YWRtaW46YWRtaW4='
```

Response:

```json
{
    "success": true,
    "error": null,
    "data": [
        "ping_mngmt_commands_http",
        "updatelayers",
        "set_all_datasets_metadata",
        "sync_geonode_maps",
        "importlayers",
        "sync_geonode_datasets"
    ]
}
```

Note: You should change the header Authorization (Basic YWRtaW46YWRtaW4=) to your Auth token, in this example I am using a token for admin as username and admin as password.

Creating a job

Optionally, before creating the job you can get its help message with the following call:

```
```

Creating a job for running ping_mngmt_commands_http with 30 seconds of sleep time:

```
    "args": ["--sleep", 30],
}
```

(continues on next page)
"kwargs": {},
"autostart": false
}

Response:
{
  "success": true,
  "error": null,
  "data": {
    "id": 8,
    "command": "ping_mngmt_commands_http",
    "app_name": "geonode.management_commands_http",
    "user": 1000,
    "status": "CREATED",
    "created_at": "2021-10-08T18:25:04572Z",
    "start_time": null,
    "end_time": null,
    "args": [
      "--sleep",
      30
    ],
    "kwargs": {},
    "celery_result_id": null,
    "output_message": null
  }
}

Note: Alternatively you can omit the jobs part of the url to create a job. (Using http://<your_geonode_host>/api/v2/management/commands/ping_mngmt_commands_http/ as url)

Start/Stop actions

To start the created job:

curl --location --request PATCH 'http://<your_geonode_host>/api/v2/management/jobs/8/
  --start/' --header 'Authorization: Basic YWRtaW46YWRtaW4='

Response:
{
  "success": true,
  "error": null,
  "data": {
    "id": 8,
    "command": "ping_mngmt_commands_http",
    "app_name": "geonode.management_commands_http",
    "user": 1000,
    "status": "QUEUED",
    "created_at": "2021-10-08T18:25:04572Z",

  }
}
"start_time": null,
"end_time": null,
"args": [
    "--sleep",
    30
],
"kwargs": {},
"celery_result_id": null,
"output_message": null
}

Note: During execution the job can be interrupted using the following call:

curl --location --request PATCH 'http://<your_geonode_host>/api/v2/management/jobs/8/
\→stop/' --header 'Authorization: Basic YWRtaW46YW1taW4=

Note that the status changed from CREATED to QUEUED, during execution it will be STARTED and at the end FINISHED.

Jobs list and status

You can verify your job status and details with the following call:

curl --location --request GET 'http://<your_geonode_host>/api/v2/management/jobs/8/
\→status/' --header 'Authorization: Basic YWRtaW46YW1taW4=

Response:

{
    "id": 8,
    "command": "ping_mngmt_commands_http",
    "app_name": "geonode.management_commands_http",
    "user": 1000,
    "status": "FINISHED",
    "created_at": "2021-10-08T18:17:25.045752Z",
    "start_time": "2021-10-08T18:20:02.761475Z",
    "end_time": "2021-10-08T18:20:32.802007Z",
    "args": [
        "--sleep",
        30
    ],
    "kwargs": {},
    "celery_result_id": "fe7359a6-5f8c-47bf-859a-84351b5ed80c",
    "output_message": "Sleeping for 30.0 seconds...
pong",
    "celery_task_meta": {
        "date_done": "2021-10-08T18:20:32.810649Z",
        "status": "SUCCESS",
        "traceback": null,
        "worker": "worker1@4f641ff9a9c0b"
    }
}
When running multiple jobs and to audit already ran jobs. A list of jobs can be retrieved using the following call:

```
curl --location --request GET 'http://<your_geonode_host>/api/v2/management/jobs/' --header 'Authorization: Basic YWRtaW46YWRtaW4='
```

Response:

```
{
   "links": {
      "next": null,
      "previous": null
   },
   "total": 1,
   "page": 1,
   "page_size": 10,
   "data": [
      {
         "id": 1,
         "command": "ping_mngmt_commands_http",
         "app_name": "geonode.management_commands_http",
         "user": 1000,
         "status": "FINISHED",
         "created_at": "2021-10-08T18:17:25.045752Z"
      }
   ]
}
```

Note: This list can be filtered by the fields “celery_result_id”, “command”, “app_name”, “status”, “user” and “user__username”.

### 1.26 Changing the default Languages

#### 1.26.1 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.
1.26.2 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:

```python
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.

1.26.3 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

```bash
mkdir /usr/lib/python2.7/dist-packages/setmydefaultlanguage
```

or

```bash
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

```python
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):
        if request.META.has_key('HTTP_ACCEPT_LANGUAGE'):
            del request.META['HTTP_ACCEPT_LANGUAGE']
```

1.26. Changing the default Languages
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:

```python
MIDDLEWARE_CLASSES += (
    'setmydefaultlanguage.ForceDefaultLanguageMiddleware',
)
```

### 1.26.4 Restart

You will need to restart GeoNode accordingly to the installation method you have chosen.

As an instance in case you are using **NGINX** with **UWSGI**, as root you will need to run the following commands:

```bash
service uwsgi restart
service nginx restart
```

Please refer to Translating GeoNode for information on editing GeoNode pages in different languages and create new GeoNode Translations.

### 1.27 GeoNode Upgrade from older versions

#### 1.27.1 Upgrade from 3.2.x / 3.3.x

1. Upgrade the dependencies
2. Perform the `migrations` management command; in case some attribute is conflicting, remove it manually from the DB
3. Proform the `collectstatic` management command

**Upgrade the instance dependencies**

Check the 1. **Install the dependencies** and 2. **GeoNode Installation** sections in order to upgrade your Python environment.

Also, make sure the code is **Python 3.8** compatible and that you switched and aligned the **source code** and the `requirements.txt` to the **master branch**.

This must be done manually and with particular attention.

```bash
workon <project environment>
cd <project_name>
pip install -r requirements.txt

cd /<full_path_to_geonode>
pip install pip --upgrade
pip install -r requirements.txt --upgrade
pip install -e . --upgrade
pip install pygdal=="gdal-config --version\`.*"
./manage.sh collectstatic --noinput
```
Run GeoNode migrations

Activate your GeoNode virtualenv and set the env vars:

```bash
. env/bin/Activate
export vars_210
```

Here are the variables to export - update them to your environment settings:

```bash
export DATABASE_URL=postgresql://user:***@localhost:5432/dbname
export DEFAULT_BACKEND_DATASTORE=data
export GEODATABASE_URL=postgresql://user:***@localhost:5432/geonode_data
export ALLOWED_HOSTS=['localhost', '192.168.100.10']
export STATIC_ROOT=~/.geonode/static/
export GEOSERVER_LOCATION=http://localhost:8080/geoserver/
export GEOSERVER_PUBLIC_LOCATION=http://localhost:8080/geoserver/
export GEOSERVER_ADMIN_PASSWORD=geoserver
export SESSION_EXPIRED_CONTROL_ENABLED=False
```

Apply migrations and apply basic fixtures:

```bash
./manage.py migrate --fake-initial
paver sync
```

Note: In case of an error of `django.db.utils.ProgrammingError: column "colum-name" of relation "table-name" already exists` on running migrations, you can backup the field data with the following steps.

```bash
./manage.sh dbshell

ALTER TABLE <table> ADD COLUMN <colum-name>_bkp varchar;
UPDATE <table> SET <colum-name>_bkp = colum-name;
ALTER TABLE <table> DROP COLUMN <colum-name>;
\q
```

Run migration then:

```bash
./manage.sh dbshell

UPDATE <table> SET <colum-name> = <colum-name>_bkp;
ALTER TABLE <table> DROP COLUMN <colum-name>_bkp;
\q
```
Create superuser

To create a superuser you should drop the following constraints (they can be re-enabled if needed):

```
alter table people_profile alter column last_login drop not null;
```

```
./manage createsuperuser
```

Update Templates

Update available templates to use `{% load static %}` instead of `{% load staticfiles %}`

1.28 GeoNode Async Signals

1.28.1 Supervisord and Systemd

1.28.2 Celery

1.28.3 Rabbitmq and Redis

1.28.4 How to: Async Upload via API

In geonode is possible to upload resources via API in async/sync way.

Here is available a full example of upload via API https://github.com/GeoNode/geonode/blob/582d6efda74adb8042d1d897004bbf764e6e0285/geonode/upload/api/tests.py#L416

Step 1

Create a common client session, this is fundamental due the fact that geonode will check the request session. For example with requests we will do something like:

```
import requests
client = requests.session()
```

Note: in Django this part is already managed

Step 2

Call the `api/v2/uploads/upload` endpoint in PUT (is a form-data endpoint) by specifying in files a dictionary with the names and the files that we want to uploads and a data payload with the required informations. For example:

```
params = {
    "permissions": '{"users": {"AnonymousUser": ["view_resourcebase"]}, "groups": []}',
    "time": "false",
    "layer_title": "layer_title",
    "charset": "UTF-8",
    # layer permissions
```

(continues on next page)
files = {
    "filename":<_io.BufferedReader name="filename">,
}

client.put(
    auth=HTTPBasicAuth(username, password),
    data=params,
    files=files,
)

Returns:
- dict with import id of the resource

### Step 3

Call in the final upload page in order to trigger the actual import. If correctly set, Geoserver will manage the upload asynchronously.

```python
client.get("http://localhost:8000/upload/final?id={import_id}")
```

The `import_id` is returned from the previous step

### Step 4

The upload as been completed on GeoNode, we should check until Geoserver has complete his part. To do so, is enough to call the detailed information about the upload that we are performing.

```python
client.get(f"http://localhost:8000/api/v2/uploads/{upload_id}"
)
```

When the status is *PROCESSED* and the completion is 100% we are able to see the resource in geonode and geoserver

### 1.29 GeoNode Add a thesaurus

#### 1.29.1 Introduction

GeoNode can import a thesaurus (or multiple thesauri) in order to index resources against subject terms or keywords. Thesauri can be managed manually in the admin panel, or imported as SKOS RDF using either the admin panel or the command-line:
1.29.2 Upload via the Admin panel

Navigate to the thesaurus page in the admin panel http://<your_geonode_host>/admin/base/thesaurus. On the top-right of the page a button named Upload thesaurus will be available:

After clicking on it, a simple form for the upload will be shown which will allow you to select your desired RDF file:

By clicking on Upload RDF, the system will load the thesaurus and assign it a “slugified” name based on the file name. The name can be easily change later in the edit page.

If everything goes fine, a success message will be shown:

Otherwise the UI will show the error message:

1.29.3 Import via the load_thesaurus command

A thesaurus can also be loaded into GeoNode by using the load_thesaurus management command:

```
python manage.py load_thesaurus --help
```

- `--dry-run` Only parse and print the thesaurus file, without perform insertion in the DB.
- `--name=NAME` Identifier name for the thesaurus in this GeoNode instance.
- `--file=FILE` Full path to a thesaurus in RDF format.

For example, in order add the INSPIRE Spatial Data Themes thesaurus into a GeoNode instance, download it as file inspire-theme.rdf with the command:

```
python manage.py load_thesaurus --name=inspire-theme --file=inspire-theme.rdf
```

and then issue the command:

```
python manage.py load_thesaurus --file inspire-theme.rdf --name inspire_themes
```

The name is the identifier you’ll use to refer to this thesaurus in your GeoNode instance.

If you only want to make sure that a thesaurus file will be properly parsed, give the --dry-run parameter, so that nothing will be added to the DB.

*Note:* if the name starts with the string fake, the file will not be accessed at all, and some test keywords will be added to a fake new thesaurus. In this case the dry-run param will not be used.

### 1.29.4 Configure a thesaurus in GeoNode

#### Configuration from Admin

After you loaded a thesaurus into GeoNode, it should be configured in the Admin panel.

The panel can be reached from Admin link of the User Menu in the navigation bar or through this URL: http://<your_geonode_host>/admin/base/thesaurus.

Once you are on the Thesaurus lists, select one thesaurus to open the Edit page

- **identifier**: (mandatory string) the identifier you used in the load_thesaurus commands.
- **title**: (mandatory string) The title of the thesaurus, set initially by the load_thesaurus command.
- **date**: (mandatory date) The Date of the thesaurus, set initially by the load_thesaurus command.
- **description**: (mandatory string) The description of the thesaurus, set initially by the load_thesaurus command.
- **slug**: (mandatory string) The slug of the thesaurus, set initially by the load_thesaurus command.
- **about**: (optional string) The about of the thesaurus, set initially by the load_thesaurus command.
- **card min**: (optional integer) The minimum cardinality, default = 0
- **card max**: (optional integer) The maximum cardinality, default = -1
- **facet**: (boolean) Decide if the thesaurus will be shown in the facet list, default: True
- **order**: (integer) Decide the listing order of the thesaurus in the facet list and in the metadata editor, default: 0, asc order from 0 to N

Cardinality:
Change thesaurus

<table>
<thead>
<tr>
<th>Identifier</th>
<th>thesaurus_unique_identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>Thesaurus Title</td>
</tr>
<tr>
<td>Data</td>
<td>2010-05-23T10:25:56</td>
</tr>
<tr>
<td>Description</td>
<td>Thesaurus description</td>
</tr>
<tr>
<td>Slug</td>
<td>slug</td>
</tr>
<tr>
<td>About</td>
<td><a href="http://about-thesaurus.com">http://about-thesaurus.com</a></td>
</tr>
</tbody>
</table>

Card min 1

Card max 0

Card max 0 –> Disabled, The Thesaurus will not appear in the GUI

Card max = 1 & card min = 0 –> Single choice, optional.

Card max = 1 & card min = 1 –> Single choice, required

Card max = -1 & card min = 0 –> [0..N] Multiple choices, optional

Card max = -1 & card min = 1 –> [1..N] Multiple choices, required

After the setup, in Editing Tools -> Metadata -> Wizard the thesaurus block will be shown like the following image:

Fig. 309: The GeoNode Thesaurus edit Interface

Fig. 310: The metadata interface with the Thesaurus enabled
Configuration via settings.py

**Warning:** *Deprecated* The Thesaurus configuration via settings is deprecated, will be removed in the future.

After you loaded a thesaurus into GeoNode, it should be configured in the settings.py file (or in the local_settings) in this way:

```
THESAURUS = {
    'name': 'THESAURUS NAME',
    'required': True|False,
    'filter': True|False,
}
```

- **name**: (mandatory string) the identifier you used in the load_thesaurus commands.
- **required**: (optional boolean) if True, a keyword of this thesaurus is mandatory to complete the metadata. *Currently not implemented.*
- **filter**: (optional boolean) if True, a faceted list of keywords of this thesaurus will be presented on the search page.

So, in order to set up the INSPIRE themes thesaurus you may set the THESAURUS value as:

```
THESAURUS = {
    'name': 'inspire_themes',
    'required': True,
    'filter': True
}
```

### 1.29.5 Apply a thesaurus to a resource

After you’ve finished the setup you should find a new input widget in each resource metadata wizard allowing you to choose a thesaurus for your resource.

After applying a thesaurus to resources those should be listed in the filter section in GeoNode’s resource list views:

- Atmospheric conditions
- Coordinate reference syst...
- Sea regions

### 1.29.6 Exporting a thesaurus as RDF via the dump_thesaurus command

GeoNode thesauri can be exported as RDF using the dump_thesaurus command:

```
python manage.py dump_thesaurus --help
```

- `-n NAME, --name NAME` Dump the thesaurus with the given name
- `-f FORMAT, --format FORMAT` Format string supported by rdflib, e.g.: pretty-xml (default),
- `--json-ld, n3, nt, pretty-xml, trig, ttl, xml`

(continues on next page)
--default-lang LANG  Default language code for untagged string literals
-1, --list  List available thesauri

The -n|--name argument refers, like the load_thesaurus command, to the thesaurus’s identifier in GeoNode, as opposed to its title. If uploaded via the admin interface this is derived automatically from its file name. Information about thesauri can be shown on the command-line using dump_thesaurus with just the -l|--list option.

1.30  Participate in the Discussion

1.30.1  Join the community, ask for help or report bugs

In case of general questions the GeoNode Community is present at following channels

- User Mailing List: https://lists.osgeo.org/cgi-bin/mailman/listinfo/geonode-users
- Developer Mailing List: https://lists.osgeo.org/cgi-bin/mailman/listinfo/geonode-devel
- Gitter Chat: https://gitter.im/GeoNode/general

For reporting bugs please open a ticket at Github issues:

- https://github.com/GeoNode/geonode/issues

1.31  Write Documentation

1.31.1  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.

To contribute to the GeoNode documentation you should:

- Read the GeoServer Style Guidelines
- Create an account on GitHub
- Fork the GeoNode repository
- Edit the files
- Submit pull requests

All these things can generally be done within your browser, you won’t need to download anything. However, if you need to add images or planning bigger changes working locally is recommended.
Style Guidelines

While we do not have strict rules for writing docs, we encourage you to read GeoServer Style Guidelines before you start writing: https://docs.geoserver.org/latest/en/docguide/style.html

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. After you’ve signed in, visit the geonode_documentation repository https://github.com/geonode/documentation.

Fork the documentation repository

In order to make changes, you first have to fork the repository. On the top right of the website, you will find a button named “fork” to do so.

If you want to read more about forking please visit the official GitHub docs: https://help.github.com/articles/fork-a-repo.

Edit files on Github

For smaller changes you can use the GitHub website. Navigate your Browser to your forked repository. To make changes to files, navigate to the file in question and hit the edit button on the right top.

Note: The documentation is written in reStructuredText, a lightweight markup language. To learn how to use it see: https://docutils.sourceforge.net/docs/user/rst/quickref.html.

By hitting the preview button you will be able to see how your changes will look like. To save your changes, click on Commit Changes at the bottom of the site.

To ask the documentation maintainers to integrate your changes the creation of a Pull Request is needed. Therefore use the new pull request button to start the process. Find more about Pull requests at the official GitHub documentation: https://help.github.com/en/github/collaborating-with-issues-and-pull-requests/about-pull-requests.

Edit files locally

If you’re planning bigger changes on the structure of the documentation, it is advisable to make your changes locally. Further, while you can work on your master branch, it is recommended to create a dedicated branch for your changes.

Start by navigating to a folder where you like to keep your repository locally and install the needed dependencies

```
$ cd /opt
$ git clone https://github.com/your_documentation_repository
$ git remote add upstream https://github.com/geonode/documentation
# add the GeoNode documentation repository as "upstream" source

$ cd your_documentation_repository
$ git fetch upstream;
# get last commits from upstream

$ git merge upstream/master master
```

(continues on next page)
# merge the upstream with your fork
# if you like, you can also use 'git pull', which is nothing else than fetching and merging in one step

$ git push
# update your repository at GitHub (origin)

Your repository should now be up to date! For more information on those commands go to https://git-scm.com/docs.

Let's install the dependencies

$ pip install virtualenv
$ virtualenv docs_env
$ source docs_env/bin/activate
$ pip install sphinx sphinx_rtd_theme sphinx-autobuild

You can now start the sphinx development server which will serve and live-reload your docs at https://localhost:8000

$ sphinx-autobuild . _build

When finished create a build with following command

$ make html
# for a last check you can open the index.html in _build subdirectory

**Create a pull request**

As with directly editing files in your browser, you will need to create a Pull request to ask for integrating your changes into the main repository.

$ git status
# will list all changed files

$ git add ...
# add the files of interest

$ git commit -m 'Fixes #1234 Updated docs for ...'
# choose a meaningful commit message

$ git push <branch>

After running these commands, navigate your browser to your GitHub repository and create a pull request as explained above.
1.32 Provide Translations

1.32.1 Contribute to Translations

Behind the scenes, GeoNode is using a software called GNU gettext further text-based translation files (django.po and djangojs.po) for translating content. If you’d like to know more about how all of this works you’ll find a full description at the Django Docs. Following will concentrate on what is needed for edit existing or contribute a new translation.

Download the translation File

All language files live in a specific subfolder called after their iso code within the locale folder. For example, for French, the main translation file called django.po can be downloaded from here.

Next, to download the language file, we need to install an OpenSource Editor called “poedit” for editing from: https://poedit.net/download

Translation process

Make a copy of the file before starting the translation so that you can revert in case of errors.

After installing ‘poedit’, you should be able to double click on the ‘.po’ file to open it. Poedit’s interface should look similar to the one shown in the picture below:
Identifying translation issues

From the ‘poedit’ menu ‘View’, make sure that ‘Entries with Errors first’ is checked:

Next click on ‘Validate Translations’ from the ‘Catalogue’ menu:

‘Poedit’ will place translations which may require additional consideration on top of the list. A warning mark means that the interpretation might be not entirely consistent with the original phrase. This is not necessarily an error, just a warning asking the user to double check.

Following to marked phrases, ‘Poedit’ will show untranslated sentences. When clicking on one, it can be translated through the bottom panel.

During translation pay special attention to the button saying ‘needs work’. In case this button is checked, the phrase will be marked as ‘fuzzy’ and ignored in GeoNode.
1.32. Provide Translations
Saving translations

As soon as the translation is complete, it must be saved and compiled. Saving is straightforward. All you have to do is clicking the ‘Save’ button from the top menu.

As a last step we compile the file. Compiling the translation means to create a binary “.mo” file out of the edited “.po” file. To do so, click on “Compile to MO”.

Poedit will ask where to write the “.mo” file to, by default, this is the same folder as the edited ‘.po’ resides in. The ‘.mo’ file can be overwritten if necessary.
Push translations to the repository

For sharing our updates, we must upload the files to GeoNode’s GitHub repository. Go to the correct file position which, in case for French is: https://github.com/GeoNode/geonode/tree/master/geonode/locale/fr/LC_MESSAGES

Click on “Upload Files”

Drag the updated files into the Upload form, and write a title/description of the changes

Click on “Create a new branch for this commit…” and then click on the green button.

The last step will create a PULL REQUEST which can be reviewed and then approved by a developer.
Activate updated translation at your server

Once the files have been pushed to GitHub, it will be necessary to update your server to respect changed files. At this time, this can be done only by an administrator. From the server’s shell following commands need to be executed:

```
workon geonode
cd /opt/geonode
DJANGO_SETTINGS_MODULE=geonode.settings python -W ignore manage.py collectstatic --noinput
sudo service uwsgi restart
```

Texts not listed in .po files

In case you find a template output without corresponding translation you can add it as follows:

Identify the corresponding template file which is responsible for outputting the text. Add a `{% trans "TEXT" %}` tag. Save the template file and run the following:

```
django-admin makemessages --no-location -l en -d django -e "html,txt,py" -i docs
django-admin makemessages --no-location -l en -d djangojs -e "js" -i docs -i node_modules -i lib
```

This will update the english .po file. also to update the language which should be edited by setting the `-l fr` parameter. Continue with updating the .po file as described above.
1.33 Write Code

1.34 Frontend Development

1.34.1 Frontend development

Knowledge of handling node/npm is required.

The GeoNode frontend dependencies can be found in ./geonode/static. To manage dependencies, we recommend the use of yarn package manager (https://yarnpkg.com/lang/en).

First steps:

yarn install
Installs the required libraries to ./node_modules

yarn install <package>@version [--dev]
Installs a package with a defined version. Using --dev installs a dependency that is only available for the build process (see: package.json devDependencies).

yarn remove <package>
Removes a package.

yarn outdated
Shows version information.

yarn why <package>
Get information on why this package was installed.

For further information on how to install or use please visit the official yarn documentation.

File/Folder overview:

./static_dependencies.json
includes all dependencies associated with each file. For example all files which should be minified to assets.min.js are named as values. All files that should be copied to lib folder (for DEBUG_STATIC) are values of key other_dependencies and so on. Before you can use a dependency it has to be added to package.json by use of yarn.

./Gruntfile.js
reads the dependencies from static_dependencies.json and contains all workflows.
geonode/static/geonode

The ./geonode folder contains GeoNode’s stylesheets and javascript files. The CSS files are generated via less. CSS files should therefore never be changed directly but it’s corresponding less file. Further this folder should never be deleted!

geonode/static/lib

The ./lib folder contains all the third-party files. This folder can be deleted as it will be fully generated by use of grunt development|production

Example 1 – Change styling:

1. In your settings set DEBUG_STATIC=True. This will load unminified assets in your template.
2. Start the development server with paver start.
3. Use grunt watch to watch all less files for change.
4. Change styling in ./geonode/static/geonode/less
5. If our changes are as expected create a new build with grunt development (files are not minimized) or grunt production (files are minimized)

Example 2 – add/update a new library:

1. In your settings set DEBUG_STATIC=True. This will load unminified assets in your template.
2. yarn add angular@1.7
3. vim static_dependencies.json Edit the file and add your dependency to its fitting destination. For example, assets.min.js
4. Check if some Django template (for example, base.html) includes the file and add it or adjust the version
5. use grunt production to build the package

For further tasks have a look at gruntfile.js or ask for help in the development mailing list

Note: Please make maintainers work easier and add a message to your commit why a library has been added. (For example, commit -m ‘select2 added for permissions form on layer detail page’)

1.35 GeoNode API

1.35.1 API v2 - Schema

OpenAPI 3.0 Schema

GET /api/v2/

Status Codes

• 200 OK – No response body
GET /api/v2/categories/
API endpoint that lists categories.

Query Parameters
- `ordering` *(string)* – Which field to use when ordering the results.
- `page` *(integer)* – A page number within the paginated result set.
- `page_size` *(integer)* – Number of results to return per page.
- `search` *(string)* – A search term.

Status Codes
- 200 OK –

GET /api/v2/categories/{id}/
API endpoint that lists categories.

Parameters
- `id` *(integer)* – A unique integer value identifying this topic category.

Status Codes
- 200 OK –

GET /api/v2/datasets/
API endpoint that allows layers to be viewed or edited.

Query Parameters
- `ordering` *(string)* – Which field to use when ordering the results.
- `page` *(integer)* – A page number within the paginated result set.
- `page_size` *(integer)* – Number of results to return per page.
- `search` *(string)* – A search term.

Status Codes
- 200 OK –

PATCH /api/v2/datasets/
API endpoint that allows layers to be viewed or edited.

Status Codes
- 200 OK –

GET /api/v2/datasets/{id}/
API endpoint that allows layers to be viewed or edited.

Parameters
- `id` *(integer)* – A unique integer value identifying this dataset.

Status Codes
- 200 OK –
PUT /api/v2/datasets/{id}/

Update one or more model instances.

If ENABLE_BULK_UPDATE is set, multiple previously-fetched records may be updated in a single call, provided their IDs.

If ENABLE_PATCH_ALL is set, multiple records may be updated in a single PATCH call, even without knowing their IDs.

**WARNING**: ENABLE_PATCH_ALL should be considered an advanced feature and used with caution. This feature must be enabled at the viewset level and must also be requested explicitly by the client via the “patch-all” query parameter.

This parameter can have one of the following values:

**true (or 1): records will be fetched and then updated in a transaction loop**
- The Model.save method will be called and model signals will run
- This can be slow if there are too many signals or many records in the query
- This is considered the more safe and default behavior

**query: records will be updated in a single query**
- The QuerySet.update method will be called and model signals will not run
- This will be fast, but may break data constraints that are controlled by signals
- This is considered unsafe but useful in certain situations

The server's successful response to a patch-all request will NOT include any individual records. Instead, the response content will contain a “meta” object with an “updated” count of updated records.

Examples:
Update many dogs by ID:

PATCH /dogs/ [ 
  {'id': 1, 'fur': 'white'}, {'id': 2, 'fur': 'black'}, {'id': 3, 'fur': 'yellow'} 
]

Update all dogs in a query:

PATCH /dogs/?filter{fur.contains}=brown&patch-all=true { 
  'fur': 'gold' 
}

Parameters

- **id (integer)** – A unique integer value identifying this dataset.

Status Codes

- **200 OK** –
PATCH /dataset/{id}/ {
  '{metadata_1_name}': '{metadata_1_value}', '{metadata_n_name}': '{metadata_n_value}'
}

Parameters

- **id** *(integer)* – A unique integer value identifying this dataset.

Status Codes

- **200 OK** –

GET /api/v2/datasets/{id}/{field_name}/
Fetch related object(s), as if sideloaded (used to support link objects).

This method gets mapped to /<resource>/<pk>/<field_name>/ by DynamicRouter for all DynamicRelation-Field fields. Generally, this method probably shouldn’t be overridden.

An alternative implementation would be to generate reverse queries. For an exploration of that approach, see:

https://gist.github.com/ryochiji/54687d675978c7d96503

Parameters

- **field_name** *(string)* –
- **id** *(integer)* – A unique integer value identifying this dataset.

Status Codes

- **200 OK** –

GET /api/v2/datasets/{id}/maplayers/
API endpoint allowing to retrieve the MapLayers list.

Parameters

- **id** *(integer)* – A unique integer value identifying this dataset.

Query Parameters

- **ordering** *(string)* – Which field to use when ordering the results.
- **page** *(integer)* – A page number within the paginated result set.
- **page_size** *(integer)* – Number of results to return per page.
- **search** *(string)* – A search term.

Status Codes

- **200 OK** –

GET /api/v2/datasets/{id}/maps/
API endpoint allowing to retrieve maps using the dataset.

Parameters

- **id** *(integer)* – A unique integer value identifying this dataset.

Query Parameters

- **ordering** *(string)* – Which field to use when ordering the results.
• **page** *(integer)* – A page number within the paginated result set.
• **page_size** *(integer)* – Number of results to return per page.
• **search** *(string)* – A search term.

**Status Codes**
• 200 OK –

**PUT /api/v2/datasets/{id}/metadata/**
API endpoint allowing to upload ISO-19115 compliant XML metadata for the dataset.

**Parameters**
• **id** *(integer)* – A unique integer value identifying this dataset.

**Status Codes**
• 200 OK –

**GET /api/v2/documents/**
API endpoint that allows documents to be viewed or edited.

**Query Parameters**
• **ordering** *(string)* – Which field to use when ordering the results.
• **page** *(integer)* – A page number within the paginated result set.
• **page_size** *(integer)* – Number of results to return per page.
• **search** *(string)* – A search term.

**Status Codes**
• 200 OK –

**PATCH /api/v2/documents/**
API endpoint that allows documents to be viewed or edited.

**Status Codes**
• 200 OK –

**GET /api/v2/documents/{id}/**
API endpoint that allows documents to be viewed or edited.

**Parameters**
• **id** *(integer)* – A unique integer value identifying this document.

**Status Codes**
• 200 OK –

**PUT /api/v2/documents/{id}/**
Update one or more model instances.

If ENABLE_BULK_UPDATE is set, multiple previously-fetched records may be updated in a single call, provided their IDs.

If ENABLE_PATCH_ALL is set, multiple records may be updated in a single PATCH call, even without knowing their IDs.
WARNING: ENABLE_PATCH_ALL should be considered an advanced feature and used with caution. This feature must be enabled at the viewset level and must also be requested explicitly by the client via the “patch-all” query parameter.

This parameter can have one of the following values:

**true (or 1): records will be fetched and then updated in a transaction loop**

- The `Model.save` method will be called and model signals will run
- This can be slow if there are too many signals or many records in the query
- This is considered the more safe and default behavior

**query: records will be updated in a single query**

- The `QuerySet.update` method will be called and model signals will not run
- This will be fast, but may break data constraints that are controlled by signals
- This is considered unsafe but useful in certain situations

The server’s successful response to a patch-all request will NOT include any individual records. Instead, the response content will contain a “meta” object with an “updated” count of updated records.

**Examples:**

Update one dog:

```bash
PATCH /dogs/1/ {
  'fur': 'white'
}
```

Update many dogs by ID:

```bash
PATCH /dogs/ [  
  {‘id’: 1, ‘fur’: ‘white’}, {‘id’: 2, ‘fur’: ‘black’}, {‘id’: 3, ‘fur’: ‘yellow’}  
]
```

Update all dogs in a query:

```bash
PATCH /dogs/?filter{fur.contains}=brown&patch-all=true {  
  ‘fur’: ‘gold’  
}
```

**Parameters**

- **id** *(integer)* – A unique integer value identifying this document.

**Status Codes**

- **200 OK** –

**PATCH /api/v2/documents/{id}/**

API endpoint that allows documents to be viewed or edited.

**Parameters**

- **id** *(integer)* – A unique integer value identifying this document.

**Status Codes**

- **200 OK** –
GET /api/v2/documents/{id}/linked_resources/
API endpoint allowing to retrieve the DocumentResourceLink(s).

Parameters
- **id** (integer) – A unique integer value identifying this document.

Query Parameters
- **ordering** (string) – Which field to use when ordering the results.
- **page** (integer) – A page number within the paginated result set.
- **page_size** (integer) – Number of results to return per page.
- **search** (string) – A search term.

Status Codes
- **200 OK** –

GET /api/v2/geoapps/
API endpoint that allows geoapps to be viewed or edited.

Query Parameters
- **ordering** (string) – Which field to use when ordering the results.
- **page** (integer) – A page number within the paginated result set.
- **page_size** (integer) – Number of results to return per page.
- **search** (string) – A search term.

Status Codes
- **200 OK** –

POST /api/v2/geoapps/
Either create a single or many model instances in bulk using the Serializer’s many=True ability from Django REST >= 2.2.5.

The data can be represented by the serializer name (single or plural forms), dict or list.

Examples:

POST /dogs/ {
    "name": "Fido", "age": 2
}

POST /dogs/ {
    "dog": {
        "name": "Lucky", "age": 3
    }
}

POST /dogs/ {
    "dogs": [
        {"name": "Fido", "age": 2}, {"name": "Lucky", "age": 3}
    ]
}
POST /dogs/ [ 
  {"name": "Fido", "age": 2}, {"name": "Lucky", "age": 3} 
]

Status Codes
• 201 Created –

PATCH /api/v2/geoapps/
API endpoint that allows geoapps to be viewed or edited.

Status Codes
• 200 OK –

GET /api/v2/geoapps/{id}/
API endpoint that allows geoapps to be viewed or edited.

Parameters
• id (integer) – A unique integer value identifying this geo app.

Status Codes
• 200 OK –

PUT /api/v2/geoapps/{id}/
Update one or more model instances.

If ENABLE_BULK_UPDATE is set, multiple previously-fetched records may be updated in a single call, provided their IDs.

If ENABLE_PATCH_ALL is set, multiple records may be updated in a single PATCH call, even without knowing their IDs.

WARNING: ENABLE_PATCH_ALL should be considered an advanced feature and used with caution. This feature must be enabled at the viewset level and must also be requested explicitly by the client via the “patch-all” query parameter.

This parameter can have one of the following values:

true (or 1): records will be fetched and then updated in a transaction loop
• The Model.save method will be called and model signals will run
• This can be slow if there are too many signals or many records in the query
• This is considered the more safe and default behavior

query: records will be updated in a single query
• The QuerySet.update method will be called and model signals will not run
• This will be fast, but may break data constraints that are controlled by signals
• This is considered unsafe but useful in certain situations

The server’s successful response to a patch-all request will NOT include any individual records. Instead, the response content will contain a “meta” object with an “updated” count of updated records.

Examples:
Update one dog:
PATCH /dogs/1/ {
    ‘fur’: ‘white’
}

Update many dogs by ID:

PATCH /dogs/ {
    {'id': 1, ‘fur’: ‘white’}, {'id': 2, ‘fur’: ‘black’}, {'id': 3, ‘fur’: ‘yellow’}
}

Update all dogs in a query:

PATCH /dogs/?filter{fur.contains}=brown&patch-all=true {
    ‘fur’: ‘gold’
}

Parameters

• **id** (integer) – A unique integer value identifying this geo app.

Status Codes

• **200 OK** –

PATCH /api/v2/geoapps/{id}/

API endpoint that allows geoapps to be viewed or edited.

Parameters

• **id** (integer) – A unique integer value identifying this geo app.

Status Codes

• **200 OK** –

GET /api/v2/geoapps/{id}/{field_name}/

Fetch related object(s), as if sideloaded (used to support link objects).

This method gets mapped to /<resource>/<pk>/<field_name>/ by DynamicRouter for all DynamicRelation-Field fields. Generally, this method probably shouldn’t be overridden.

An alternative implementation would be to generate reverse queries. For an exploration of that approach, see:

https://gist.github.com/ryochiji/54687d675978c7d96503

Parameters

• **field_name** (string) –

• **id** (integer) – A unique integer value identifying this geo app.

Status Codes

• **200 OK** –

GET /api/v2/groups/

API endpoint that allows gropus to be viewed or edited.

Query Parameters

• **ordering** (string) – Which field to use when ordering the results.
**page** (*integer*) – A page number within the paginated result set.

**page_size** (*integer*) – Number of results to return per page.

**search** (*string*) – A search term.

**Status Codes**

- **200 OK** –

**POST /api/v2/groups/**

Either create a single or many model instances in bulk using the Serializer’s many=True ability from Django REST >= 2.2.5.

The data can be represented by the serializer name (single or plural forms), dict or list.

Examples:

```
POST /dogs/
{
    "name": "Fido", "age": 2
}

POST /dogs/
{
    "dog": {
        "name": "Lucky", "age": 3
    }
}

POST /dogs/
{
    "dogs": [
        {"name": "Fido", "age": 2}, {"name": "Lucky", "age": 3}
    ]
}

POST /dogs/
{
    {"name": "Fido", "age": 2}, {"name": "Lucky", "age": 3}
}
```

**Status Codes**

- **201 Created** –

**PATCH /api/v2/groups/**

API endpoint that allows groups to be viewed or edited.

**Status Codes**

- **200 OK** –

**DELETE /api/v2/groups/**

Either delete a single or many model instances in bulk

```
DELETE /dogs/
{
    "dogs": [
        {"id": 1}, {"id": 2}
    ]
```

1.35. GeoNode API
DELETE /dogs/ [
    {“id”: 1}, {“id”: 2}
]

Status Codes
• 204 No Content – No response body

GET /api/v2/groups/{id}/
API endpoint that allows groups to be viewed or edited.

Parameters
• id (integer) – A unique integer value identifying this group profile.

Status Codes
• 200 OK –

PUT /api/v2/groups/{id}/
Update one or more model instances.

If ENABLE_BULK_UPDATE is set, multiple previously-fetched records may be updated in a single call, pro-
vided their IDs.

If ENABLE_PATCH_ALL is set, multiple records may be updated in a single PATCH call, even without knowing
their IDs.

WARNING: ENABLE_PATCH_ALL should be considered an advanced feature and used with caution. This
feature must be enabled at the viewset level and must also be requested explicitly by the client via the “patch-all”
query parameter.

This parameter can have one of the following values:

true (or 1): records will be fetched and then updated in a transaction loop
• The Model.save method will be called and model signals will run
• This can be slow if there are too many signals or many records in the query
• This is considered the more safe and default behavior

query: records will be updated in a single query
• The QuerySet.update method will be called and model signals will not run
• This will be fast, but may break data constraints that are controlled by signals
• This is considered unsafe but useful in certain situations

The server’s successful response to a patch-all request will NOT include any individual records. Instead, the
response content will contain a “meta” object with an “updated” count of updated records.

Examples:
Update one dog:
    PATCH /dogs/1/ {
        ‘fur’: ‘white’
    }
Update many dogs by ID:
PATCH /dogs/ [
    {'id': 1, 'fur': 'white'}, {'id': 2, 'fur': 'black'}, {'id': 3, 'fur': 'yellow'}
]

Update all dogs in a query:
PATCH /dogs/?filter{fur.contains}=brown&patch-all=true {
    'fur': 'gold'
}

Parameters

• **id** (integer) – A unique integer value identifying this group profile.

Status Codes

• **200 OK** –

PATCH /api/v2/groups/{id}/

API endpoint that allows groups to be viewed or edited.

Parameters

• **id** (integer) – A unique integer value identifying this group profile.

Status Codes

• **200 OK** –

DELETE /api/v2/groups/{id}/

Either delete a single or many model instances in bulk
DELETE /dogs/ {
    “dogs”: [
        {“id”: 1}, {“id”: 2}
    ]
}
DELETE /dogs/ [
    {“id”: 1}, {“id”: 2}
]

Parameters

• **id** (integer) – A unique integer value identifying this group profile.

Status Codes

• **204 No Content** – No response body

GET /api/v2/groups/{id}/managers/

API endpoint allowing to retrieve the Group managers.

Parameters

• **id** (integer) – A unique integer value identifying this group profile.

Query Parameters
• **ordering** *(string)* – Which field to use when ordering the results.
• **page** *(integer)* – A page number within the paginated result set.
• **page_size** *(integer)* – Number of results to return per page.
• **search** *(string)* – A search term.

**Status Codes**

• **200 OK** –

**GET** /api/v2/groups/{id}/members/
API endpoint allowing to retrieve the Group members.

**Parameters**

• **id** *(integer)* – A unique integer value identifying this group profile.

**Query Parameters**

• **ordering** *(string)* – Which field to use when ordering the results.
• **page** *(integer)* – A page number within the paginated result set.
• **page_size** *(integer)* – Number of results to return per page.
• **search** *(string)* – A search term.

**Status Codes**

• **200 OK** –

**GET** /api/v2/groups/{id}/resources/
API endpoint allowing to retrieve the Group specific resources.

**Parameters**

• **id** *(integer)* – A unique integer value identifying this group profile.

**Query Parameters**

• **ordering** *(string)* – Which field to use when ordering the results.
• **page** *(integer)* – A page number within the paginated result set.
• **page_size** *(integer)* – Number of results to return per page.
• **search** *(string)* – A search term.

**Status Codes**

• **200 OK** –

**GET** /api/v2/harvesters/
A viewset that can support dynamic API features.

**Attributes:**

features: A list of features supported by the viewset. meta: Extra data that is added to the response by the DynamicRenderer.

**Query Parameters**

• **ordering** *(string)* – Which field to use when ordering the results.
• **page** *(integer)* – A page number within the paginated result set.
• **page_size** *(integer)* – Number of results to return per page.
Status Codes

- 200 OK –

POST /api/v2/harvesters/
Either create a single or many model instances in bulk using the Serializer’s many=True ability from Django REST >= 2.2.5.

The data can be represented by the serializer name (single or plural forms), dict or list.

Examples:
POST /dogs/ {
  “name”: “Fido”, “age”: 2
}

POST /dogs/ {
  “dog”: {
    “name”: “Lucky”, “age”: 3
  }
}

POST /dogs/ {
  “dogs”: [
    {“name”: “Fido”, “age”: 2}, {“name”: “Lucky”, “age”: 3}
  ]
}

POST /dogs/ {
  {“name”: “Fido”, “age”: 2}, {“name”: “Lucky”, “age”: 3}
}

Status Codes

- 201 Created –

GET /api/v2/harvesters/{harvester_id}/harvestable-resources/

Adds the update_list method to a viewset

update_list is used by api.routers.ListPatchRouter in order to allow performing PATCH requests against a viewset’s list endpoint

Parameters

- harvester_id (integer) –

Status Codes

- 200 OK – No response body

PATCH /api/v2/harvesters/{harvester_id}/harvestable-resources/

Adds the update_list method to a viewset

update_list is used by api.routers.ListPatchRouter in order to allow performing PATCH requests against a viewset’s list endpoint

Parameters
• **harvester_id** *(integer)* –

**Status Codes**

• **200 OK** – No response body

**GET /api/v2/harvesters/{id}/**

A viewset that can support dynamic API features.

**Attributes:**

- *features*: A list of features supported by the viewset.
- *meta*: Extra data that is added to the response by the DynamicRenderer.

**Parameters**

• **id** *(integer)* – A unique integer value identifying this harvester.

**Status Codes**

• **200 OK** –

**PUT /api/v2/harvesters/{id}/**

Update one or more model instances.

If **ENABLE_BULK_UPDATE** is set, multiple previously-fetched records may be updated in a single call, provided their IDs.

If **ENABLE_PATCH_ALL** is set, multiple records may be updated in a single PATCH call, even without knowing their IDs.

**WARNING**: **ENABLE_PATCH_ALL** should be considered an advanced feature and used with caution. This feature must be enabled at the viewset level and must also be requested explicitly by the client via the “patch-all” query parameter.

This parameter can have one of the following values:

**true (or 1)**: **records will be fetched and then updated in a transaction loop**

- The **Model.save** method will be called and model signals will run
- This can be slow if there are too many signals or many records in the query
- This is considered the more safe and default behavior

**query**: **records will be updated in a single query**

- The **QuerySet.update** method will be called and model signals will not run
- This will be fast, but may break data constraints that are controlled by signals
- This is considered unsafe but useful in certain situations

The server’s successful response to a patch-all request will NOT include any individual records. Instead, the response content will contain a “meta” object with an “updated” count of updated records.

**Examples:**

Update one dog:

```
PATCH /dogs/1/ {
    ‘fur’: ‘white’
}
```

Update many dogs by ID:
PATCH /dogs/ {
    [‘id’: 1, ‘fur’: ‘white’}, [‘id’: 2, ‘fur’: ‘black’}, [‘id’: 3, ‘fur’: ‘yellow’]
}

Update all dogs in a query:
PATCH /dogs/?filter{fur.contains}=brown&patch-all=true {
    ‘fur’: ‘gold’
}

Parameters
• **id (integer)** – A unique integer value identifying this harvester.

Status Codes
• **200 OK** –

PATCH /api/v2/harvesters/{id}/
A viewset that can support dynamic API features.

Attributes:
features: A list of features supported by the viewset. meta: Extra data that is added to the response by the DynamicRenderer.

Parameters
• **id (integer)** – A unique integer value identifying this harvester.

Status Codes
• **200 OK** –

DELETE /api/v2/harvesters/{id}/
Either delete a single or many model instances in bulk
DELETE /dogs/ {
    “dogs”: [“id”: 1}, {“id”: 2}
}

DELETE /dogs/ {
    {“id”: 1}, {“id”: 2}
}

Parameters
• **id (integer)** – A unique integer value identifying this harvester.

Status Codes
• **204 No Content** – No response body
GET /api/v2/harvesting-sessions/
A viewset that can support dynamic API features.

Attributes:
- features: A list of features supported by the viewset.
- meta: Extra data that is added to the response by the DynamicRenderer.

Query Parameters
- ordering (string) – Which field to use when ordering the results.
- page (integer) – A page number within the paginated result set.
- page_size (integer) – Number of results to return per page.

Status Codes
- 200 OK –

GET /api/v2/harvesting-sessions/{id}/
A viewset that can support dynamic API features.

Attributes:
- features: A list of features supported by the viewset.
- meta: Extra data that is added to the response by the DynamicRenderer.

Parameters
- id (integer) – A unique integer value identifying this asynchronous harvesting session.

Status Codes
- 200 OK –

GET /api/v2/keywords/
API endpoint that lists hierarchical keywords.

Query Parameters
- ordering (string) – Which field to use when ordering the results.
- page (integer) – A page number within the paginated result set.
- page_size (integer) – Number of results to return per page.
- search (string) – A search term.

Status Codes
- 200 OK –

GET /api/v2/keywords/{id}/
API endpoint that lists hierarchical keywords.

Parameters
- id (integer) – A unique integer value identifying this hierarchical keyword.

Status Codes
- 200 OK –
GET /api/v2/management/commands/

Handle the exposed management commands usage:

- GET: List of exposed commands
- GET detail: Help for a specific command
- POST: Create a job (and automatic runs) for a specific command.

Status Codes

- 200 OK –

POST /api/v2/management/commands/

Creates and runs a management command job. Expects application/json content type in a following shape:

```json
{
    "args": [<arg1>, <arg2>],
    "kwargs": {<key1>: <val1>, <key2>: <val2>},
    "autostart": bool
}
```

By default, autostart is set to true.

Status Codes

- 200 OK –

GET /api/v2/management/commands/{cmd_name}/

Handle the exposed management commands usage:

- GET: List of exposed commands
- GET detail: Help for a specific command
- POST: Create a job (and automatic runs) for a specific command.

Parameters

- **cmd_name** (string) –

Status Codes

- 200 OK –

POST /api/v2/management/commands/{cmd_name}/

Creates and runs a management command job. Expects application/json content type in a following shape:

```json
{
    "args": [<arg1>, <arg2>],
    "kwargs": {<key1>: <val1>, <key2>: <val2>},
    "autostart": bool
}
```

By default, autostart is set to true.

Parameters

- **cmd_name** (string) –

Status Codes

- 200 OK –

GET /api/v2/management/commands/{cmd_name}/jobs/

Create, List, Retrieve, Start, Stop and Get Status of a Management Command Job.

Parameters

- **cmd_name** (string) –

Query Parameters
• **app_name** *(string)* –
• **celery_result_id** *(string)* –
• **command** *(string)* –
• **page** *(integer)* – A page number within the paginated result set.
• **page_size** *(integer)* – Number of results to return per page.
• **status** *(string)* –
• **user** *(integer)* –
• **user__username** *(string)* –

**Status Codes**

• **200 OK** –

**POST /api/v2/management/commands/{cmd_name}/jobs/**
Create, List, Retrieve, Start, Stop and Get Status of a Management Command Job.

**Parameters**

• **cmd_name** *(string)* –

**Status Codes**

• **201 Created** –

**GET /api/v2/management/commands/{cmd_name}/jobs/{id}/**
Create, List, Retrieve, Start, Stop and Get Status of a Management Command Job.

**Parameters**

• **cmd_name** *(string)* –

• **id** *(integer)* – A unique integer value identifying this management command job.

**Status Codes**

• **200 OK** –

**PATCH /api/v2/management/commands/{cmd_name}/jobs/{id}/start/**
Create, List, Retrieve, Start, Stop and Get Status of a Management Command Job.

**Parameters**

• **cmd_name** *(string)* –

• **id** *(integer)* – A unique integer value identifying this management command job.

**Status Codes**

• **200 OK** –

**GET /api/v2/management/commands/{cmd_name}/jobs/{id}/status/**
Create, List, Retrieve, Start, Stop and Get Status of a Management Command Job.

**Parameters**

• **cmd_name** *(string)* –

• **id** *(integer)* – A unique integer value identifying this management command job.

**Status Codes**

• **200 OK** –
PATCH /api/v2/management/commands/{cmd_name}/jobs/{id}/stop/
Create, List, Retrieve, Start, Stop and Get Status of a Management Command Job.

Parameters
- **cmd_name** (string) –
- **id** (integer) – A unique integer value identifying this management command job.

Status Codes
- 200 OK –

GET /api/v2/management/jobs/
Create, List, Retrieve, Start, Stop and Get Status of a Management Command Job.

Query Parameters
- **app_name** (string) –
- **celery_result_id** (string) –
- **command** (string) –
- **page** (integer) – A page number within the paginated result set.
- **page_size** (integer) – Number of results to return per page.
- **status** (string) –
- **user** (integer) –
- **user__username** (string) –

Status Codes
- 200 OK –

POST /api/v2/management/jobs/
Create, List, Retrieve, Start, Stop and Get Status of a Management Command Job.

Status Codes
- 201 Created –

GET /api/v2/management/jobs/{id}/
Create, List, Retrieve, Start, Stop and Get Status of a Management Command Job.

Parameters
- **id** (integer) – A unique integer value identifying this management command job.

Status Codes
- 200 OK –

PATCH /api/v2/management/jobs/{id}/start/
Create, List, Retrieve, Start, Stop and Get Status of a Management Command Job.

Parameters
- **id** (integer) – A unique integer value identifying this management command job.

Status Codes
- 200 OK –
GET /api/v2/management/jobs/{id}/status/
Create, List, Retrieve, Start, Stop and Get Status of a Management Command Job.

Parameters
• id (integer) – A unique integer value identifying this management command job.

Status Codes
• 200 OK –

PATCH /api/v2/management/jobs/{id}/stop/
Create, List, Retrieve, Start, Stop and Get Status of a Management Command Job.

Parameters
• id (integer) – A unique integer value identifying this management command job.

Status Codes
• 200 OK –

GET /api/v2/maps/
API endpoint that allows maps to be viewed or edited.

Query Parameters
• ordering (string) – Which field to use when ordering the results.
• page (integer) – A page number within the paginated result set.
• page_size (integer) – Number of results to return per page.
• search (string) – A search term.

Status Codes
• 200 OK –

POST /api/v2/maps/
Changes in the m2m maplayers are committed before object changes. To protect the db, this action is done within an atomic transation.

Status Codes
• 201 Created –

PATCH /api/v2/maps/
API endpoint that allows maps to be viewed or edited.

Status Codes
• 200 OK –

GET /api/v2/maps/{id}/
API endpoint that allows maps to be viewed or edited.

Parameters
• id (integer) – A unique integer value identifying this map.

Status Codes
• 200 OK –
PUT /api/v2/maps/{id}/
Changes in the m2m `maplayers` are committed before object changes. To protect the db, this action is done within an atomic transaction.

Parameters
- `id` (integer) – A unique integer value identifying this map.

Status Codes
- 200 OK –

PATCH /api/v2/maps/{id}/
API endpoint that allows maps to be viewed or edited.

Parameters
- `id` (integer) – A unique integer value identifying this map.

Status Codes
- 200 OK –

GET /api/v2/maps/{id}/{field_name}/
Fetch related object(s), as if sideloaded (used to support link objects).

This method gets mapped to `<resource>/<pk>/<field_name>/` by DynamicRouter for all DynamicRelation-Field fields. Generally, this method probably shouldn’t be overridden.

An alternative implementation would be to generate reverse queries. For an exploration of that approach, see:

https://gist.github.com/ryochiji/54687d675978c7d96503

Parameters
- `field_name` (string) –
- `id` (integer) – A unique integer value identifying this map.

Status Codes
- 200 OK –

GET /api/v2/maps/{id}/datasets/
API endpoint allowing to retrieve the local MapLayers.

Parameters
- `id` (integer) – A unique integer value identifying this map.

Query Parameters
- `ordering` (string) – Which field to use when ordering the results.
- `page` (integer) – A page number within the paginated result set.
- `page_size` (integer) – Number of results to return per page.
- `search` (string) – A search term.

Status Codes
- 200 OK –
GET /api/v2/maps/{id}/maplayers/
API endpoint allowing to retrieve the MapLayers list.

Parameters

• id (integer) – A unique integer value identifying this map.

Query Parameters

• ordering (string) – Which field to use when ordering the results.
• page (integer) – A page number within the paginated result set.
• page_size (integer) – Number of results to return per page.
• search (string) – A search term.

Status Codes

• 200 OK –

GET /api/v2/owners/
API endpoint that lists all possible owners.

Query Parameters

• ordering (string) – Which field to use when ordering the results.
• page (integer) – A page number within the paginated result set.
• page_size (integer) – Number of results to return per page.
• search (string) – A search term.

Status Codes

• 200 OK –

GET /api/v2/owners/{id}/
API endpoint that lists all possible owners.

Parameters

• id (integer) – A unique integer value identifying this user.

Status Codes

• 200 OK –

GET /api/v2/regions/
API endpoint that lists regions.

Query Parameters

• ordering (string) – Which field to use when ordering the results.
• page (integer) – A page number within the paginated result set.
• page_size (integer) – Number of results to return per page.
• search (string) – A search term.

Status Codes

• 200 OK –
GET /api/v2/regions/{id}/
API endpoint that lists regions.

Parameters
- id (integer) – A unique integer value identifying this region.

Status Codes
- 200 OK –

GET /api/v2/resource-service/execution-status/{execution_id}
Main dispatcher endpoint to follow an API request status progress

- GET input: <str: execution_id>
- output: <ExecutionRequest>

Parameters
- execution_id (string) –

Status Codes
- 200 OK – No response body

GET /api/v2/resources/
API endpoint that allows base resources to be viewed or edited.

Query Parameters
- ordering (string) – Which field to use when ordering the results.
- page (integer) – A page number within the paginated result set.
- page_size (integer) – Number of results to return per page.
- search (string) – A search term.

Status Codes
- 200 OK –

POST /api/v2/resources/
Either create a single or many model instances in bulk using the Serializer’s many=True ability from Django REST >= 2.2.5.

The data can be represented by the serializer name (single or plural forms), dict or list.

Examples:
POST /dogs/ {
  "name": "Fido", "age": 2
}
POST /dogs/ {
  "dog": {
    "name": "Lucky", "age": 3
  }
}
POST /dogs/ {
“dogs”: [
    {“name”: “Fido”, “age”: 2}, {“name”: “Lucky”, “age”: 3}
]
]
POST /dogs/ [
    {“name”: “Fido”, “age”: 2}, {“name”: “Lucky”, “age”: 3}
]
]

Status Codes
• 201 Created –

PATCH /api/v2/resources/
  API endpoint that allows base resources to be viewed or edited.
  Status Codes
  • 200 OK –

DELETE /api/v2/resources/
  Either delete a single or many model instances in bulk
DELETE /dogs/ {
    “dogs”: [
        {“id”: 1}, {“id”: 2}
    ]
}
DELETE /dogs/ [
    {“id”: 1}, {“id”: 2}
]
]

Status Codes
• 204 No Content – No response body

GET /api/v2/resources/{id}/
  API endpoint that allows base resources to be viewed or edited.
  Parameters
  • id (integer) – A unique integer value identifying this resource base.
  Status Codes
  • 200 OK –

PUT /api/v2/resources/{id}/
  Update one or more model instances.
  If ENABLE_BULK_UPDATE is set, multiple previously-fetched records may be updated in a single call, pro-
vided their IDs.
  If ENABLE_PATCH_ALL is set, multiple records may be updated in a single PATCH call, even without knowing
their IDs.
WARNING: ENABLE_PATCH_ALL should be considered an advanced feature and used with caution. This feature must be enabled at the viewset level and must also be requested explicitly by the client via the “patch-all” query parameter.

This parameter can have one of the following values:

- **true (or 1): records will be fetched and then updated in a transaction loop**
  - The `Model.save` method will be called and model signals will run
  - This can be slow if there are too many signals or many records in the query
  - This is considered the more safe and default behavior

- **query: records will be updated in a single query**
  - The `QuerySet.update` method will be called and model signals will not run
  - This will be fast, but may break data constraints that are controlled by signals
  - This is considered unsafe but useful in certain situations

The server’s successful response to a patch-all request will NOT include any individual records. Instead, the response content will contain a “meta” object with an “updated” count of updated records.

Examples:

Update one dog:

```json
PATCH /dogs/1/ {
  'fur': 'white'
}
```

Update many dogs by ID:

```json
PATCH /dogs/ [ { 'id': 1, 'fur': 'white' }, { 'id': 2, 'fur': 'black' }, { 'id': 3, 'fur': 'yellow' } ]
```

Update all dogs in a query:

```json
PATCH /dogs/?filter{fur.contains}=brown&patch-all=true {
  'fur': 'gold'
}
```

**Parameters**

- **id** *(integer)* – A unique integer value identifying this resource base.

**Status Codes**

- **200 OK** –

**PATCH /api/v2/resources/{id}/**

API endpoint that allows base resources to be viewed or edited.

**Parameters**

- **id** *(integer)* – A unique integer value identifying this resource base.

**Status Codes**

- **200 OK** –
DELETE /api/v2/resources/{id}/

Either delete a single or many model instances in bulk

DELETE /dogs/ 

```
   "dogs": [ 
      {"id": 1}, {"id": 2} 
   ] 
```

DELETE /dogs/ 

```
{"id": 1}, {"id": 2} 
```

Parameters

- **id** (integer) – A unique integer value identifying this resource base.

Status Codes

- **204 No Content** – No response body

GET /api/v2/resources/{id}/{field_name}/

Fetch related object(s), as if sideloaded (used to support link objects).

This method gets mapped to /<resource>/<pk>/<field_name>/ by DynamicRouter for all DynamicRelationField fields. Generally, this method probably shouldn’t be overridden.

An alternative implementation would be to generate reverse queries. For an exploration of that approach, see:

https://gist.github.com/ryochiji/54687d675978c7d96503

Parameters

- **field_name** (string) –

- **id** (integer) – A unique integer value identifying this resource base.

Status Codes

- **200 OK** –

PUT /api/v2/resources/{id}/copy/

Instructs the Async dispatcher to execute a ‘COPY’ operation over a valid ‘uuid’.

Parameters

- **id** (integer) – A unique integer value identifying this resource base.

Status Codes

- **200 OK** –

DELETE /api/v2/resources/{id}/delete/

Instructs the Async dispatcher to execute a ‘DELETE’ operation over a valid ‘uuid’.

Parameters

- **id** (integer) – A unique integer value identifying this resource base.

Status Codes

- **200 OK** –
GET /api/v2/resources/{id}/extra_metadata/
Get/Update/Delete/Add extra metadata for resource

Parameters

• id (integer) – A unique integer value identifying this resource base.

Status Codes

• 200 OK –

POST /api/v2/resources/{id}/extra_metadata/
Get/Update/Delete/Add extra metadata for resource

Parameters

• id (integer) – A unique integer value identifying this resource base.

Status Codes

• 200 OK –

PUT /api/v2/resources/{id}/extra_metadata/
Get/Update/Delete/Add extra metadata for resource

Parameters

• id (integer) – A unique integer value identifying this resource base.

Status Codes

• 200 OK –

DELETE /api/v2/resources/{id}/extra_metadata/
Get/Update/Delete/Add extra metadata for resource

Parameters

• id (integer) – A unique integer value identifying this resource base.

Status Codes

• 204 No Content – No response body

POST /api/v2/resources/{id}/favorite/
API endpoint allowing to retrieve the favorite Resources.

Parameters

• id (integer) – A unique integer value identifying this resource base.

Query Parameters

• ordering (string) – Which field to use when ordering the results.
• page (integer) – A page number within the paginated result set.
• page_size (integer) – Number of results to return per page.
• search (string) – A search term.

Status Codes

• 200 OK –
DELETE /api/v2/resources/{id}/favorite/

API endpoint allowing to retrieve the favorite Resources.

Parameters

- **id** *(integer)* – A unique integer value identifying this resource base.

Query Parameters

- **ordering** *(string)* – Which field to use when ordering the results.
- **page** *(integer)* – A page number within the paginated result set.
- **page_size** *(integer)* – Number of results to return per page.
- **search** *(string)* – A search term.

Status Codes

- **200 OK** –

GET /api/v2/resources/{id}/permissions/

Sets an object’s the permission levels based on the perm_spec JSON.

the mapping looks like:

```json
{  
'users': {    
'AnonymousUser': ['view'], <username>: ['perm1','perm2','perm3'], <username2>: ['perm1','perm2','perm3']...  
}, 'groups': {  
<groupname>: ['perm1','perm2','perm3'], <groupname2>: ['perm1','perm2','perm3'], ...  
}
}
```

Parameters

- **id** *(integer)* – A unique integer value identifying this resource base.

Status Codes

- **200 OK** – No response body

PUT /api/v2/resources/{id}/permissions/

Sets an object’s the permission levels based on the perm_spec JSON.

the mapping looks like:

```json
{  
'users': {    
'AnonymousUser': ['view'], <username>: ['perm1','perm2','perm3'], <username2>: ['perm1','perm2','perm3']...  
}, 'groups': {  
<groupname>: ['perm1','perm2','perm3'], <groupname2>: ['perm1','perm2','perm3'], ...  
}
}
```

Parameters

- **id** *(integer)* – A unique integer value identifying this resource base.
Status Codes

- **200 OK** – No response body

**PATCH /api/v2/resources/{id}/permissions/**

Sets an object’s the permission levels based on the perm_spec JSON.

the mapping looks like:

```
{
    "users": {
        "AnonymousUser": ['view'],
        <username>: ['perm1','perm2','perm3'], <username2>: ['perm1','perm2','perm3'] ...
    },
    "groups": {
        <groupname>: ['perm1','perm2','perm3'], <groupname2>: ['perm1','perm2','perm3'], ...
    }
}
```

**Parameters**

- **id (integer)** – A unique integer value identifying this resource base.

**Status Codes**

- **200 OK** – No response body

**DELETE /api/v2/resources/{id}/permissions/**

Sets an object’s the permission levels based on the perm_spec JSON.

the mapping looks like:

```
{
    "users": {
        "AnonymousUser": ['view'],
        <username>: ['perm1','perm2','perm3'], <username2>: ['perm1','perm2','perm3'] ...
    },
    "groups": {
        <groupname>: ['perm1','perm2','perm3'], <groupname2>: ['perm1','perm2','perm3'], ...
    }
}
```

**Parameters**

- **id (integer)** – A unique integer value identifying this resource base.

**Status Codes**

- **200 OK** – No response body

**GET /api/v2/resources/{id}/ratings/**

API endpoint allowing to rate and get overall rating of the Resource.

**Parameters**

- **id (integer)** – A unique integer value identifying this resource base.

**Status Codes**

- **200 OK** –
POST /api/v2/resources/{id}/ratings/
API endpoint allowing to rate and get overall rating of the Resource.

Parameters
- **id (integer)** – A unique integer value identifying this resource base.

Status Codes
- **200 OK** –

PUT /api/v2/resources/{id}/set_thumbnail/
API endpoint allowing to set thumbnail of the Resource.

Parameters
- **id (integer)** – A unique integer value identifying this resource base.

Status Codes
- **200 OK** –

PUT /api/v2/resources/{id}/update/
Instructs the Async dispatcher to execute a ‘UPDATE’ operation over a valid ‘uuid’.

Parameters
- **id (integer)** – A unique integer value identifying this resource base.

Status Codes
- **200 OK** –

POST /api/v2/resources/{resource_id}/set_thumbnail_from_bbox/
API endpoint allowing to set the thumbnail url for an existing dataset.

Parameters
- **resource_id (string)** –

Status Codes
- **200 OK** –

GET /api/v2/resources/approved/
API endpoint allowing to retrieve the approved Resources.

Query Parameters
- **ordering (string)** – Which field to use when ordering the results.
- **page (integer)** – A page number within the paginated result set.
- **page_size (integer)** – Number of results to return per page.
- **search (string)** – A search term.

Status Codes
- **200 OK** –

POST /api/v2/resources/create/{resource_type}/
Instructs the Async dispatcher to execute a ‘CREATE’ operation.

Parameters
- **resource_type (string)** –
Status Codes

• 200 OK –

GET /api/v2/resources/favorites/
API endpoint allowing to retrieve the favorite Resources.

Query Parameters

• ordering (string) – Which field to use when ordering the results.
• page (integer) – A page number within the paginated result set.
• page_size (integer) – Number of results to return per page.
• search (string) – A search term.

Status Codes

• 200 OK –

GET /api/v2/resources/featured/
API endpoint allowing to retrieve the featured Resources.

Query Parameters

• ordering (string) – Which field to use when ordering the results.
• page (integer) – A page number within the paginated result set.
• page_size (integer) – Number of results to return per page.
• search (string) – A search term.

Status Codes

• 200 OK –

POST /api/v2/resources/ingest/{resource_type}/
Instructs the Async dispatcher to execute a ‘INGEST’ operation.

Parameters

• resource_type (string) –

Status Codes

• 200 OK –

GET /api/v2/resources/published/
API endpoint allowing to retrieve the published Resources.

Query Parameters

• ordering (string) – Which field to use when ordering the results.
• page (integer) – A page number within the paginated result set.
• page_size (integer) – Number of results to return per page.
• search (string) – A search term.

Status Codes

• 200 OK –
GET /api/v2/resources/resource_types/

Returns the list of available ResourceBase polymorphic_ctypes.
the mapping looks like: ```
{
  "resource_types": [
    {
      "name": "layer", "count": <number of layers>
    },
    {
      "name": "map", "count": <number of maps>
    },
    {
      "name": "document", "count": <number of documents>
    },
    {
      "name": "geostory", "count": <number of geostories>
    }
  ]
}
```

Status Codes
• 200 OK –

GET /api/v2/schema/

OpenApi3 schema for this API. Format can be selected via content negotiation.
• YAML: application/vnd.oai.openapi
• JSON: application/vnd.oai.openapi+json

Query Parameters
• format (string) –
• lang (string) –

Status Codes
• 200 OK –

GET /api/v2/tkeywords/

API endpoint that lists Thesaurus keywords.

Query Parameters
• ordering (string) – Which field to use when ordering the results.
• page (integer) – A page number within the paginated result set.
• page_size (integer) – Number of results to return per page.
• search (string) – A search term.

Status Codes
• 200 OK –
GET /api/v2/tkeywords/{id}/
- API endpoint that lists Thesaurus keywords.

Parameters
- **id** (integer) – A unique integer value identifying this thesaurus keyword.

Status Codes
- **200 OK** –

GET /api/v2/upload-parallelism-limits/
- A viewset that can support dynamic API features.

Attributes:
- **features**: A list of features supported by the viewset.
- **meta**: Extra data that is added to the response by the DynamicRenderer.

Query Parameters
- **ordering** (string) – Which field to use when ordering the results.
- **page** (integer) – A page number within the paginated result set.
- **page_size** (integer) – Number of results to return per page.

Status Codes
- **200 OK** –

POST /api/v2/upload-parallelism-limits/
- Either create a single or many model instances in bulk using the Serializer’s many=True ability from Django REST >= 2.2.5.

The data can be represented by the serializer name (single or plural forms), dict or list.

Examples:
POST /dogs/ {
   "name": "Fido", "age": 2
}

POST /dogs/ {
   "dog": {
      "name": "Lucky", "age": 3
   }
}

POST /dogs/ {
   "dogs": [
      {"name": "Fido", "age": 2}, {"name": "Lucky", "age": 3}
   ]
}

POST /dogs/ [
   {"name": "Fido", "age": 2}, {"name": "Lucky", "age": 3}
]
Status Codes

- 201 Created –

GET /api/v2/upload-parallelism-limits/{slug}/
A viewset that can support dynamic API features.

Attributes:
- features: A list of features supported by the viewset.
- meta: Extra data that is added to the response by the DynamicRenderer.

Parameters
- slug (string) – A unique value identifying this upload parallelism limit.

Status Codes

- 200 OK –

GET /api/v2/upload-size-limits/
A viewset that can support dynamic API features.

Attributes:
- features: A list of features supported by the viewset.
- meta: Extra data that is added to the response by the DynamicRenderer.

Query Parameters
- ordering (string) – Which field to use when ordering the results.
- page (integer) – A page number within the paginated result set.
- page_size (integer) – Number of results to return per page.

Status Codes

- 200 OK –

POST /api/v2/upload-size-limits/
Either create a single or many model instances in bulk using the Serializer’s many=True ability from Django REST >= 2.2.5.
The data can be represented by the serializer name (single or plural forms), dict or list.

Examples:
POST /dogs/ {
    "name": "Fido", "age": 2
}
POST /dogs/ {
    "dog": {
        "name": "Lucky", "age": 3
    }
}
POST /dogs/ {

"dogs": [
    {"name": "Fido", "age": 2}, {"name": "Lucky", "age": 3}
]
]

POST /dogs/ [
    {"name": "Fido", "age": 2}, {"name": "Lucky", "age": 3}
]

Status Codes
• 201 Created –

GET /api/v2/upload-size-limits/{slug}/
A viewset that can support dynamic API features.

Attributes:
features: A list of features supported by the viewset. meta: Extra data that is added to the response by the DynamicRenderer.

Parameters
• slug (string) – A unique value identifying this upload size limit.

Status Codes
• 200 OK –

GET /api/v2/uploads/
API endpoint that allows uploads to be viewed or edited.

Query Parameters
• ordering (string) – Which field to use when ordering the results.
• page (integer) – A page number within the paginated result set.
• page_size (integer) – Number of results to return per page.
• search (string) – A search term.

Status Codes
• 200 OK –

POST /api/v2/uploads/
Either create a single or many model instances in bulk using the Serializer’s many=True ability from Django REST >= 2.2.5.

The data can be represented by the serializer name (single or plural forms), dict or list.

Examples:
POST /dogs/ {
    "name": "Fido", "age": 2
}
POST /dogs/ {
“dog”:
   “name”: “Lucky”, “age”: 3
}

POST /dogs/ {
   “dogs”: [
      {“name”: “Fido”, “age”: 2}, {“name”: “Lucky”, “age”: 3}
   ]
}

POST /dogs/ {
   {“name”: “Fido”, “age”: 2}, {“name”: “Lucky”, “age”: 3}
}

Status Codes
• 201 Created –

GET /api/v2/uploads/{id}/
API endpoint that allows uploads to be viewed or edited.

Parameters
• id (integer) – A unique integer value identifying this upload.

Status Codes
• 200 OK –

POST /api/v2/uploads/upload/

Starts an upload session based on the Dataset Upload Form.

the form params look like: 

   ‘csrfmiddlewaretoken’: self.csrf_token, ‘permissions’: ‘{ “users”: {“AnonymousUser”:

Status Codes
• 201 Created – No response body

GET /api/v2/users/
API endpoint that allows users to be viewed or edited.

Query Parameters
• ordering (string) – Which field to use when ordering the results.
• page (integer) – A page number within the paginated result set.
• page_size (integer) – Number of results to return per page.
• search (string) – A search term.

Status Codes
• 200 OK –

POST /api/v2/users/

Either create a single or many model instances in bulk using the Serializer’s many=True ability from Django
REST >= 2.2.5.

The data can be represented by the serializer name (single or plural forms), dict or list.

Examples:

```json
POST /dogs/ {
    "name": "Fido", "age": 2
}

POST /dogs/ {
    "dog": {
        "name": "Lucky", "age": 3
    }
}

POST /dogs/ {
    "dogs": [
        {"name": "Fido", "age": 2}, {"name": "Lucky", "age": 3}
    ]
}

POST /dogs/ {
    {"name": "Fido", "age": 2}, {"name": "Lucky", "age": 3}
}
```

Status Codes

• 201 Created –

PATCH /api/v2/users/

API endpoint that allows users to be viewed or edited.

Status Codes

• 200 OK –

DELETE /api/v2/users/

Either delete a single or many model instances in bulk

DELETE /dogs/ {
    "dogs": [
        {"id": 1}, {"id": 2}
    ]
}

DELETE /dogs/ {
    {"id": 1}, {"id": 2}
}
Status Codes

• 204 No Content – No response body

GET /api/v2/users/{id}/
API endpoint that allows users to be viewed or edited.

Parameters

• id (integer) – A unique integer value identifying this user.

Status Codes

• 200 OK –

PUT /api/v2/users/{id}/
Update one or more model instances.

If ENABLE_BULK_UPDATE is set, multiple previously-fetched records may be updated in a single call, provided their IDs.

If ENABLE_PATCH_ALL is set, multiple records may be updated in a single PATCH call, even without knowing their IDs.

WARNING: ENABLE_PATCH_ALL should be considered an advanced feature and used with caution. This feature must be enabled at the viewset level and must also be requested explicitly by the client via the “patch-all” query parameter.

This parameter can have one of the following values:

true (or 1): records will be fetched and then updated in a transaction loop

• The Model.save method will be called and model signals will run
• This can be slow if there are too many signals or many records in the query
• This is considered the more safe and default behavior

query: records will be updated in a single query

• The QuerySet.update method will be called and model signals will not run
• This will be fast, but may break data constraints that are controlled by signals
• This is considered unsafe but useful in certain situations

The server's successful response to a patch-all request will NOT include any individual records. Instead, the response content will contain a “meta” object with an “updated” count of updated records.

Examples:
Update one dog:

PATCH /dogs/1/ {
    ‘fur’: ‘white’
}

Update many dogs by ID:

PATCH /dogs/ [
    {‘id’: 1, ‘fur’: ‘white’}, {‘id’: 2, ‘fur’: ‘black’}, {‘id’: 3, ‘fur’: ‘yellow’}
]

Update all dogs in a query:
PATCH /dogs/?filter{fur.contains}=brown&patch-all=true {
    ‘fur’: ‘gold’
}

Parameters

• **id** *(integer)* – A unique integer value identifying this user.

Status Codes

• **200 OK** –

PATCH /api/v2/users/{id}/

API endpoint that allows users to be viewed or edited.

Parameters

• **id** *(integer)* – A unique integer value identifying this user.

Status Codes

• **200 OK** –

DELETE /api/v2/users/{id}/

Either delete a single or many model instances in bulk

DELETE /dogs/ {
    “dogs”: [
        {“id”: 1}, {“id”: 2}
    ]
}

DELETE /dogs/ [
    {“id”: 1}, {“id”: 2}
]

Parameters

• **id** *(integer)* – A unique integer value identifying this user.

Status Codes

• **204 No Content** – No response body

GET /api/v2/users/{id}/groups/

API endpoint allowing to retrieve the Groups the user is member of.

Parameters

• **id** *(integer)* – A unique integer value identifying this user.

Query Parameters

• **ordering** *(string)* – Which field to use when ordering the results.
• **page** *(integer)* – A page number within the paginated result set.
• **page_size** *(integer)* – Number of results to return per page.
• **search** *(string)* – A search term.
Status Codes

- 200 OK –

GET /api/v2/users/{id}/resources/

API endpoint allowing to retrieve the Resources visible to the user.

Parameters

- id (integer) – A unique integer value identifying this user.

Query Parameters

- ordering (string) – Which field to use when ordering the results.
- page (integer) – A page number within the paginated result set.
- page_size (integer) – Number of results to return per page.
- search (string) – A search term.

Status Codes

- 200 OK –

1.35.2 API usage examples

In this section, we are going to demonstrate how GeoNode API can be utilized/integrated with other applications using Python.

Resource Listing and Details

As mentioned in previous chapters, GeoNode resources are categorized in different types e.g. datasets, maps, documents. Etc. All available resources can be listed with API GET /api/v2/resources.

To obtain a single resource, a primary key is provided in the url. Eg GET /api/v2/resources/{resource.pk}.

Example Requests:

1. Listing

```python
import requests
url = "https://master.demo.geonode.org/api/v2/resources"
response = requests.request("GET", url)
```

2. Detail

```python
import requests
url = "https://master.demo.geonode.org/api/v2/resources/1797"
response = requests.request("GET", url)
```

Note: The above requests work for publicly visible resources. If a resource is private either the Basic Auth or the Bearer token must be included inside the headers.
3. Listing with basic auth:

```python
import requests
url = "https://master.demo.geonode.org/api/v2/resources"
headers = {
    'Authorization': 'Basic dXNlcjpwYXNzd29yZA=='
}
response = requests.request("GET", url, headers=headers)
```

A token can be used in place of Basic Auth by setting 'Authorization': 'Bearer <token>'.

**Resource Download**

The download URL for a resource can be obtained from `resource.download_url`. This URL executes the synchronous download of a resource in its default download format (ESRI Shapefile for vector, Geotiff for rasters and the original format for documents). Additional export formats for datasets are available through the UI. At the moment the API doesn’t expose them.

**Resource Links**

From the resource detail response, URIs and links to services can be obtained from the `resource.links[]` array value. The purpose of each link is defined by its `link_type`. The “name” also can specify additional information about the linked resource.

**Metadata**

Links to each metadata format can be obtained from links with `link_type = "metadata"`

**OGC services**

OGC requests can be built by taking: the OGC base url from links from `resource.links[]` with "link_type"= ("OGC:WMS | OGC:WFS | OGC:WCS") the OGC service layername obtained from the `resource.alternate` property

**Embedding**

A resource can be embedded inside a third party website. The “embed view” of a resource is suitable to be placed inside an iframe. The URL for the embedded view can be obtained from the `resource.embed_url` property.
Resource Searching and Filtering

GeoNode resources can be filtered with the following query parameters:

<table>
<thead>
<tr>
<th>Parameters</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>title and abstract</td>
<td>/api/v2/resources?page=1&amp;search=text-to-search&amp;search_fields=title&amp;search_fields=abstract</td>
</tr>
<tr>
<td>resource_type</td>
<td>/api/v2/resources?filter{resource_type}=map</td>
</tr>
<tr>
<td>subtype</td>
<td>/api/v2/resources?filter{resource_type}=vector</td>
</tr>
<tr>
<td>favorite</td>
<td>/api/v2/resources?favorite=true</td>
</tr>
<tr>
<td>featured</td>
<td>/api/v2/resources?filter{featured}=true</td>
</tr>
<tr>
<td>published</td>
<td>/api/v2/resources?filter{is_published}=true</td>
</tr>
<tr>
<td>approved</td>
<td>/api/v2/resources?filter{is_approved}=true</td>
</tr>
<tr>
<td>category</td>
<td>/api/v2/resources?filter{category.identifier}=example</td>
</tr>
<tr>
<td>keywords</td>
<td>/api/v2/resources?filter{keywords.name}=example</td>
</tr>
<tr>
<td>regions</td>
<td>/api/v2/resources?filter{regions.name}=global</td>
</tr>
<tr>
<td>owner</td>
<td>/api/v2/resources?filter{owner.username}=test_user</td>
</tr>
<tr>
<td>extent</td>
<td>/api/v2/resources?extent=-180,-90,180,90</td>
</tr>
</tbody>
</table>

Examples:

1. Filter with a single value

```python
import requests

url = "https://master.demo.geonode.org/api/v2/resources/?filter{resource_type}=map"
response = requests.request("GET", url, headers=headers, data=payload)
```

2. Filter with multiple values

```python
import requests

url = "https://master.demo.geonode.org/api/v2/resources/?filter{resource_type.in}=map&filter{resource_type.in}=dataset"
response = requests.request("GET", url, headers=headers, data=payload)
```

Note: With filter APIs of format /api/v2/resources?filter{filter_key}=value, additional methods(in and
icontains) can be used on them to provide extensively filtered results. Eg /api/v2/resources?filter{regions.name.icontains}=global /api/v2/resources?filter{regions.name.in}=global.

It's important to note that other methods are case sensitive except the icontains.

**Dataset specific resources**

Get the metadata of uploaded datasets with:

- API: GET /api/v2/datasets/{id}
- Status Code: 200

**Note:** This is very similar to GET /api/v2/resources but provides additional metadata specifically for datasets like featureinfo_custom_template or attribute_set

Example:

```python
import requests

DATASET_ID = "the dataset id"
url = f'https://master.demo.geonode.org/api/v2/datasets/{DATASET_ID}'
headers = {
    'Authorization': 'Basic dXNlcjpwYXNzd29yZA=='
}
response = requests.request("GET", url, headers=headers)
```

**Resource Upload**

The API supports the upload of datasets and documents.

The dataset upload form accepts file formats of ESRI Shapefile, GeoTIFF, Comma Separated Value (CSV), Zip Archive, XML Metadata File, and Styled Layer Descriptor (SLD). For a successful upload, the form requires base_file, dbf_file, shx_file, and prj_file. The xml_file, and Sld_file are optional.

- API: POST /api/v2/uploads/upload
- Status Code: 200

Example:

```python
import requests

url = "https://master.demo.geonode.org/api/v2/uploads/upload"
files= [
    ('sld_file',('BoulderCityLimits.sld',open('/home/myuser/BoulderCityLimits.sld','rb'),'application/octet-stream')), ('base_file',('BoulderCityLimits.shp',open('/home/BoulderCityLimits.shp','rb'),'application/octet-stream')), ('dbf_file',('BoulderCityLimits.dbf',open('/home/BoulderCityLimits.dbf','rb'),'application/octet-stream')), ('shx_file',('BoulderCityLimits.shx',open('/home/BoulderCityLimits.shx','rb'),'application/octet-stream')), ('prj_file',('BoulderCityLimits.prj',open('/home/myuser/BoulderCityLimits.prj','rb'),'application/octet-stream'))
]
... (continues on next page)
headers = {
    'Authorization': 'Basic dXNlcjpwYXNzd29yZA=='
}
response = requests.request("POST", url, headers=headers, files=files)

Documents can be uploaded as form data.

- API: POST /api/v2/documents
- Status Code: 200

Example:

```python
import requests
url = "http://localhost:8000/api/v2/documents"
payload=
    'title': 'An example image'
}
files=[
    ('doc_file',('image.jpg',open('/home/myuser/image.jpg','rb'),'image/jpeg'))
]
headers = {
    'Authorization': 'Basic dXNlcjpwYXNzd29yZA=='
}
response = requests.request("POST", url, headers=headers, data=payload, files=files)
```

Documents can also be created to reference remote resources. In this case the doc_url parameter must be used to set the URL of the remote document.

- API: POST /api/v2/documents
- Status Code: 200

Example:

```python
import requests
url = "http://localhost:8000/api/v2/documents"
payload=
    'title': 'An example image',
    'doc_url': 'http://examples.com/image.jpg'
}
headers = {
    'Authorization': 'Basic dXNlcjpwYXNzd29yZA=='
}
response = requests.request("POST", url, headers=headers, data=payload, files=files)
```

Notice that if the URL doesn’t end with a valid doc extension, the extension parameter must be used (e.g. extension: 'jpeg').
Tracking dataset upload progress

When an upload request is executed, GeoNode creates an “Execution request” and keeps updating its state and progress (it’s a property attribute, calculated on getting the response) attributes as the resource is being created and configured in Geoserver. An execution can be in one of the following status:

- ready
- running
- failed
- finished

When the dataset is successfully uploaded, the final state of the upload is set to finished.

In order to view status of the execution, the API method GET /api/v2/executionrequest/{execution_id} where {execution_id} is the value returned by the initial call to the upload API.

The returned object contains, beyond all the information related to the execution, the inputs that were passed to the execution request, and output params specific to the type of execution. In the case of a dataset upload, the output params contain the URL of the catalog page for the new dataset.

```
"output_params": {
    "detail_url": ["/catalogue/#/dataset/9881"]
},
```

You can also filter executions by status. Eg GET /api/v2/executionrequest?filter{action}=import&filter{source}=upload&filter{status}=finished

Example:

```
import requests
url = "https://stable.demo.geonode.org/api/v2/executionrequest/5f640b6b-8c51-4514-a054-995133fee107"
headers = {
    'Authorization': 'Basic dXNlcjpwYXNzd29yZA=='
}
response = requests.request("GET", url, headers=headers)
```

Overwriting a dataset

Uploading a resource will create by default a new dataset. This behaviour can be changed by setting the overwrite_existing_layer parameter to True. In this case the upload procedure will overwrite a resource whose name matches with the new one.
Skip existing dataset

If the parameter `skip_existing_layers` is set to true True the upload procedure will ignore files whose name matched with already existing resources.

Upload of a metadata file

A complete metadata file conforming to ISO-19115 can be uploaded for a dataset.

- API: `PUT /api/v2/datasets/{dataset_id}/metadata`
- Status Code: `200`

Example:

```python
import requests
url = "http://localhost:8000/api/v2/datasets/1/metadata"
files=[
    ('metadata_file',('metadata.xml',open('/home/user/metadata.xml','rb'),'text/xml'))
]
headers = {
    'Authorization': 'Basic dXNlcjpwYXNzd29yZA=='
}
response = requests.request("PUT", url, headers=headers, data={}, files=files)
```

Resource Delete

- API: `DELETE /api/v2/resources/{pk}/delete`
- Status Code: `204`

Example:

```python
import requests
url = "https://master.demo.geonode.org/api/v2/resources/1778"
headers = {
    'Authorization': 'Basic dXNlcjpwYXNzd29yZA=='
}
response = requests.request("DELETE", url, headers=headers)
```
Resource Download

GeoNode offers a download option to resources of resource_type dataset and document. For datasets, the download option is available for only datasets with uploaded files.

Datasets

- API: GET /datasets/{resource.alternate}/dataset_download
  - Status Code: 200

Example:

```python
import requests
url = "https://master.demo.geonode.org/datasets/geonode:BoulderCityLimits3/dataset_download"
response = requests.request("GET", url)
```

Documents

- API: GET /documents/{resource.pk}/download
  - Status Code: 200

Example:

```python
import requests
url = "https://master.demo.geonode.org/documents/1781/download"
response = requests.request("GET", url)
```

Dataset Update Metadata

- API: PATCH /api/v2/datasets/{id}
  - Status Code: 200

The following example changes the title and the license of a dataset.

```python
import requests
url = ROOT + "api/v2/datasets/" + DATASET_ID
auth = (LOGIN_NAME, LOGIN_PASSWORD)
data = {
    "title": "a new title",
    "license": 4,
}
response = requests.patch(url, auth=auth, json=data)
```

Note: bbox_polygon and ll_bbox_polygon are derived values which cannot be changed.
Users, Groups and Permissions

Users

Listing

- API: POST /api/v2/users
  - Status Code: 200

Example:

```python
import requests
url = "https://master.demo.geonode.org/api/v2/users"
headers = {
    'Authorization': 'Basic dXNlcjpwYXNzd29yZA=='
}
response = requests.request("GET", url, headers=headers)
```

User detail

- API: POST /api/v2/users/{pk}
  - Status Code: 200

Example:

```python
import requests
url = "https://master.demo.geonode.org/api/v2/users/1000"
headers = {
    'Authorization': 'Basic dXNlcjpwYXNzd29yZA=='
}
response = requests.request("GET", url, headers=headers)
```

Create a new user

- API: POST /api/v2/users
  - Status Code: 200

Example:

```python
import requests
url = "https://master.demo.geonode.org/api/v2/users"
headers = {
    'Authorization': 'Basic dXNlcjpwYXNzd29yZA=='
}
payload={
    "username": "username",
    "password": "password",
    "email": "email@email.com",
}
response = requests.request("POST", url, headers=headers, json=payload)
```
Edit a User

- API: PATCH /api/v2/users/{pk}
- Status Code: 200

Example:

```python
import requests

url = "https://master.demo.geonode.org/api/v2/users/1000"
headers = {
    'Authorization': 'Basic dXNlcjpwYXNzd29yZA=='
}
payload = {'password': 'new_password'}
response = requests.request("PATCH", url, headers=headers, data=payload)
```

Delete a User

- API: DELETE /api/v2/users/{pk}
- Status Code: 200

Example:

```python
import requests

url = "https://master.demo.geonode.org/api/v2/users/1000"
headers = {
    'Authorization': 'Basic dXNlcjpwYXNzd29yZA=='
}
payload = {'password': 'new_password'}
response = requests.request("DELETE", url, headers=headers, data=payload)
```

In this case the list of validation rules configured in USER_DELETION_RULES are checked before the deletion is executed.
List user groups

- API: POST /api/v2/users/{pk}/groups
- Status Code: 200

Example:

```python
import requests
url = "https://master.demo.geonode.org/api/v2/users/1000/groups"
headers = {
    'Authorization': 'Basic dXNlcjpwYXNzd29yZA=='
}
response = requests.request("GET", url, headers=headers)
```

Transfer resources owned by a user to another

- API: POST /api/v2/users/{pk}/transfer_resources
- Status Code: 200

Example:

```python
import requests
payload={"owner": 1001}
url = "https://master.demo.geonode.org/api/v2/users/1000/transfer_resources"
headers = {
    'Authorization': 'Basic dXNlcjpwYXNzd29yZA=='
}
response = requests.request("POST", url, headers=headers, data=payload)
```

In this case the resources will be transfered to the user with id 1001, instead using the payload={"owner": "DEFAULT"}
the resources will be transfered to the principal user

Remove user as a group manager

- API: POST /api/v2/users/{pk}/remove_from_group_manager
- Status Code: 200

Example:

```python
import requests
payload={"groups": [1,2,3]}
url = "https://master.demo.geonode.org/api/v2/users/1000/remove_from_group_manager"
headers = {
    'Authorization': 'Basic dXNlcjpwYXNzd29yZA=='
}
response = requests.request("POST", url, headers=headers, data=payload)
```

In this case the user shall be removed as a group manager from the following group ids, if the payload would be
payload={"groups": "ALL"} the user will be removed as a group manager from all the groups its part of
Groups

In GeoNode, on listing groups, the API returns groups which have group profiles. Therefore for Django groups which are not related to a group profile are not included in the response. However these can be accessed in the Django Administration panel.

- API: POST /api/v2/groups
- Status Code: 200

Example:

```python
import requests
url = "https://master.demo.geonode.org/api/v2/groups"
headers = {
    'Authorization': 'Basic dXNlcjpXNzd29yZA=='
}
response = requests.request("GET", url, headers=headers)
```

Permissions

Permissions in GeoNode are set per resource and per user or group. The following are general permissions that can be assigned:

- **View**: allows to view the resource. [view_resourcebase]
- **Download**: allows to download the resource specifically datasets and documents. [view_resourcebase, download_resourcebase]
- **Edit**: allows to change attributes, properties of the datasets features, styles and metadata for the specified resource. [view_resourcebase, download_resourcebase, change_resourcebase, change_dataset_style, change_dataset_data, change_resourcebase_metadata]
- **Manage**: allows to update, delete, change permissions, publish and unpublish the resource. [view_resourcebase, download_resourcebase, change_resourcebase, change_dataset_style, change_dataset_data, publish_resourcebase, delete_resourcebase, change_resourcebase_metadata, change_resourcebase_permissions]

Obtaining permissions on a resource

On listing the resources or on resource detail API, GeoNode includes perms attribute to each resource with a list of permissions a user making the request has on the respective resource.

GeoNode also provides an API to get an overview of permissions set on a resource. The response contains users and groups with permissions set on them. However this API returns 200 if a requesting user has manage permissions on the resource otherwise it will return 403 (Forbidden).

- API: GET /api/v2/resources/1791/permissions

Example:

```python
import requests
url = "https://master.demo.geonode.org/api/v2/resources/1791/permissions"
headers = {
(continues on next page)
Authorization: Basic dXNlcjpwYXNzd29yZA==
}
response = requests.request("GET", url, headers=headers)

Changing permissions on a resource

Permissions are configured with a so called perms spec, which is a JSON structured where permissions for single users and groups can be specified.

The example below shows a perm specification for following rules:
- user1 can edit
- user2 can manage
- group1 can edit
- anonymous users (public) can view
- registered members can download

NOTE: The id of the “anonymous” and “registered members” groups can be obtained from the permissions of the resource. They are not listed inside the groups API, since these are speceial internal groups

```
{
    "users": [
        {
            "id": <id_of_user1>,
            "permissions": "edit"
        },
        {
            "id": <id_of_user2>,
            "permissions": "manage"
        }
    ],
    "organizations": [
        {
            "id": <id_of_group1>,
            "permissions": "edit"
        }
    ],
    "groups": [
        {
            "id": <id_of_anonymous_group>,
            "permissions": "view"
        },
        {
            "id": <id_of_regisdtered-members_group>,
            "permissions": "download"
        }
    ]
}
```

The perm spec is sent as JSON, with application/json Content-Type, inside a PUT request.
import requests
import json

url = "https://master.demo.geonode.org/api/v2/resources/1791/permissions"
payload = json.dumps({
    "users": [
        {
            "id": 1001,
            "permissions": "edit"
        },
        {
            "id": 1002,
            "permissions": "manage"
        }
    ],
    "organizations": [
        {
            "id": 1,
            "permissions": "edit"
        }
    ],
    "groups": [
        {
            "id": 2,
            "permissions": "view"
        },
        {
            "id": 3,
            "permissions": "download"
        }
    ]
})
headers = {
    'Authorization': 'Basic dXNlcjpwYXNzd29yZA==',
    'Content-Type': 'application/json',
}

response = requests.request("PUT", url, headers=headers, data=payload)

This is an asynchronous operation which returns a response similar to the following:

```json
{
    "status": "ready",
    "execution_id": "7ed578c2-7db8-47fe-a3f5-6ed3ca545b67",
    "status_url": "https://master.demo.geonode.org/api/v2/resource-service/execution-status/7ed578c2-7db8-47fe-a3f5-6ed3ca545b67"
}
```

The `status_url` property returns the URL to track the progress of the request. Querying the URL a result similar to the following will be returned:

```json
{
    "user": "admin",
    "execution_id": "7ed578c2-7db8-47fe-a3f5-6ed3ca545b67",
    "status": "ready",
    "status_url": "https://master.demo.geonode.org/api/v2/resource-service/execution-status/7ed578c2-7db8-47fe-a3f5-6ed3ca545b67"
}
```

(continues on next page)
"status": "running",
"func_name": "set_permissions",
"created": "2022-07-08T11:16:32.240453Z",
"finished": null,
"last_updated": "2022-07-08T11:16:32.240485Z",
"input_params": {
  ...
}
}
}

The operation will be completed once the status property is updated with the value finished.

Linked Resources Listing and Details

All available linked_resources can be listed with API GET /api/v2/resources/{pk}/linked_resources. where pk Resource base id

Example Requests:

1. List all resource links

```python
import requests
url = "https://master.demo.geonode.org/api/v2/resources/{pk}/linked_resources"
response = requests.request("GET", url)
```

1.36 How to Develop

1.36.1 Start to develop with Docker

How to run the instance for development

There are two options to develop using Docker containers:

- **Alternative A**: Running by command line and editing the code using your preferred editor (usually harder).
- **Alternative B**: Using the vscode remote containers extension (easier).

Alternative A: Building and running Docker for development

Build (first time only):

```bash
docker-compose --project-name geonode -f docker-compose-dev.yml -f .devcontainer/docker-compose.yml build
```

Running:

```bash
docker-compose --project-name geonode -f docker-compose-dev.yml -f .devcontainer/docker-compose.yml up
```
Note: If you are running `postgresql` and `tomcat9` services, you need to stop them, `docker-compose` will take care of running the database and geonode service. Otherwise, you will get the following error:

```
ERROR: for db Cannot start service db: driver failed programming external connectivity on endpoint db4geonode: Error starting userland proxy: listen tcp4 0.0.0.0:5432: bind: address already in use
ERROR: Encountered errors while bringing up the project.
```

Running the geonode application in debug mode:

```
docker exec -it django4geonode bash -c "python manage.py runserver 0.0.0.0:8000"
```

When running, you can debug the application using your preferred method. For example, you can edit a file, save it and see your modifications. You can also use `ipdb` to add breakpoints and inspect your code (Writing `import ipdb; ipdb.set_trace()` in the line you want your breakpoint).

Another option is to use `debugpy` alongside with `vscode`, for this you have to enable `debugpy` inside your `django4geonode` container:

```
docker exec -it django4geonode bash -c "pip install debugpy -t /tmp && python /tmp/debugpy --wait-for-client --listen 0.0.0.0:5678 manage.py runserver 0.0.0.0:8000 --nothreading --noreload"
```

Select “Run and Debug” in vscode and use the following launch instruction in your `.vscode/launch.json` file: `launch.json`

**Alternative B: Using vscode extension**

Alternatively, you can develop using the vscode `remote containers` extension. In this approach you need to:

- Install the extension in your vscode: `ms-vscode-remote.remote-containers`
- On your command pallet, select: “Remote-Containers: Reopen in Container”
- If it’s the first time, vscode will take care of building the images. This might take some time.
- Then a new vscode window will open, and it’ll be connected to your docker container.
- The message “Dev Container: Debug Docker Compose” will appear in the bottom-left corner of that window.
- In the vscode terminal, you’re going to see something similar to `root@77e80acc89b8:/usr/src/geonode#`.
- To run your application, you can use the integrated terminal (`./manage.py runserver 0.0.0.0:8000`) or the vscode “Run and Debug” option. For launching with “Run and Debug”, use the following instruction file: `launch.json`

For more information, take a read at vscode remote containers `help page`. 
1.36.2 How to Install GeoNode-Core for development

Summary of installation

This section demonstrates a summarization of the steps to be followed in order to install GeoNode-Core for development using Ubuntu 18.04. The following steps will be customized to fit both GeoNode-Project and GeoNode-Core for development purpose.

The steps to be followed are:

1- Install build tools and libraries
2- Install dependencies and supporting tools
3- Setup Python virtual environment
4- Clone and install GeoNode from Github
5- Install and start Geoserver
6- Start GeoNode

Note: The following commands/steps will be executed on your terminal

Warning: If you have a running GeoNode service, you will need to stop it before starting the following steps. To stop GeoNode you will need to run:

```
service apache2 stop  # or your installed server
service tomcat7 stop   # or your version of tomcat
```

Install GeoNode-Core for development

GeoNode-Core installation is considered the most basic form of GeoNode. It doesn’t require any external server to be installed and it can run locally against a file-system based Spatialite database.

Installation steps

1- Install build tools and libraries

Warning: Those instructions might be outdated. Please refer to 1. Install the dependencies

```
$ sudo apt-get install -y build-essential libxml2-dev libxslt1-dev libpq-dev zlib1g-dev
```

2- Install dependencies and supporting tools
   
   Install python native libraries and tools


Warning: Those instructions might be outdated. Please refer to 1. Install the dependencies

$ sudo apt-get install -y python3-dev python3-pil python3-lxml python3-pyproj python3-shapely python3-nose python3-httplib2 python3-pip software-properties-common

Install python virtual environment

Warning: Those instructions might be outdated. Please refer to 2. GeoNode Installation

$ sudo pip install virtualenvwrapper

Install postgresql and postgis

Warning: Those instructions might be outdated. Please refer to configure_dbs_core

$ sudo apt-get install postgresql-10 postgresql-10-postgis-2.4

Change postgres password expiry and set a password

$ sudo passwd -u postgres # change password expiry information
$ sudo passwd postgres # change unix password for postgres

Create geonode role and database

$ su postgres
$ createdb geonode_dev
$ createdb geonode_dev-imports
$ psql
$ 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=# \
$ 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/10/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

To be as follows
"local" is for Unix domain socket connections only

```
local all all trust
```

Then restart PostgreSQL to make the changes effective

```
sudo service postgresql restart
```

Java dependencies

```
$ sudo apt-get install -y openjdk-11-jdk --no-install-recommends
```

Install supporting tools

```
$ sudo apt-get install -y ant maven git gettext
```

3- Setup Python virtual environment (Here is where Geonode will be running)

Add the virtualenvwrapper to your new environment.

Since we are using Ubuntu, you can add the following settings to your .bashrc file. Please note that the Ubuntu account here is called “geonode”. So you will need to change it according to the name you picked.

```
$ 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
```

And reload the settings by running

```
$ source ~/.bashrc
```

Set up the local virtual environment for Geonode

```
$ vim ~/.bashrc
# add the following line to the bottom
$ source /usr/share/virtualenvwrapper/virtualenvwrapper.sh
```

```
$ source /usr/share/virtualenvwrapper/virtualenvwrapper.sh
$ mkvirtualenv --python=/usr/bin/python3 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 virtual environment

Alternatively you can also create the virtual env like below

```
$ python3.8 -m venv /home/geonode/dev/.venvs/geonode
$ source /home/geonode/dev/.venvs/geonode/bin/activate
```

4- Download/Clone GeoNode from Github

To download the latest geonode version from github, the command “git 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

---

1.36. How to Develop  583
$ git clone https://github.com/GeoNode/geonode.git -b 4.1.x

Install Nodejs 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 apt-get install nodejs npm
$ cd geonode/geonode/static
$ npm install --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’.

Warning: Starting from the following step, you have to make sure that you installed GDAL correctly according to the documentation page “Install GDAL for Development”

Install GeoNode in the new active local virtualenv

$ cd /home/geonode/dev  # or to the directory containing your cloned GeoNode
$ pip install -e geonode
$ cd geonode/geonode

Create local_settings.py
Copy the sample file /home/geonode/dev/geonode/geonode/local_settings.py.geoserver.sample and rename it to be local_settings.py

$ cd /home/geonode/dev/geonode
$ cp geonode/local_settings.py.geoserver.sample geonode/local_settings.py
$ gedit geonode/local_settings.py

In the local_settings.py file, add the following line after the import statements:

SITEURL = "http://localhost:8000/"

In the DATABASES dictionary under the ‘default’ key, change only the values for the keys NAME, USER and PASSWORD to be as follows:

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'geonode_dev',
        'USER': 'geonode_dev',
        'PASSWORD': 'geonode_dev',
    
    .......
    .......
    ........
    ......
In the DATABASES dictionary under the ‘datastore’ key, change only the values for the keys NAME, USER and PASSWORD to be as follows:

```python
# vector datastore for uploads
'datastore': {
    # ENGINE: 'django.contrib.gis.db.backends.postgis',
    # 'ENGINE': '', # Empty ENGINE name disables
    'ENGINE': 'geonode_dev-imports',
    'USER': 'geonode_dev',
    'PASSWORD': 'geonode_dev',
    .......
    .......
    .......
    .......
}
```

In the CATALOGUE dictionary under the ‘default’ key, uncomment the USER and PASSWORD keys to activate the credentials for GeoNetwork as follows:

```python
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': urljoin(SITEURL, '/catalogue/csw'),
        # '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',
        # 'ALTERNATES_ONLY': True,
    }
}
```

5- Install and Start Geoserver

From the virtual environment, first you need to align the database structure using the following command:

```bash
$ cd /home/geonode/dev/geonode
$ python manage.py migrate
```
then setup GeoServer using the following command:

```
$ paver setup
$ paver sync
```

6- Now we can start our geonode instance

```
service apache2 stop
service tomcat7 stop

$ paver start
```

The starting process will take around 20 seconds (depends on your machine) and at the end it shows the following message:

```
Performing system checks...
System check identified no issues (1 silenced).
January 21, 2020 - 22:49:50
Django version 1.11.27, using settings 'geonode.settings'
Starting development server at http://0.0.0.0:8000/
Quit the server with CONTROL-C.
GeoNode is now available.
```

Now you can visit the geonode site by typing `http://0.0.0.0:8000` into your browser window.
Install GeoNode-Project for development after installing GeoNode-Core

GeoNode-Project gives the user flexibility to customize the installation of the GeoNode. GeoNode itself will be installed as a requirement of your project. Inside the project structure it is possible to extend, replace or modify all geonode components (e.g. css and other static files, templates, models..) and even register new django apps without touching the original GeoNode code. In order to install GeoNode-Project, the following steps need to be executed alongside the previous GeoNode-Core installation steps.

1- Use django-admin.py to create a project “my_geonode” from a GeoNode-Project template as follows:

Note: Before running the following command, make sure that you are currently working on the virtual environment and just outside geonode directory. The command will create a new project called “my_geonode” which should be located at the level of geonode-core installation directory “inside /home/geonode/dev”

```
$ django-admin.py startproject my_geonode --template=https://github.com/GeoNode/geonode-project/archive/master.zip -e py,rst,json,yml,ini,env,sample -n Dockerfile
```

Note: Although the following command might show that the majority of requirements are already satisfied “because GeoNode-Core was already installed”, it is recommended to still execute it as it might update or install any missing package.

```
$ ls /home/geonode/dev  # should output: geonode  my_geonode
```

2- Install all the required packages/tools for GeoNode-Project as follows:

```
$ pip install -e my_geonode
```

Note: As mentioned earlier, GeoNode will be installed as requirement for the GeoNode-Project in order to be able to extend it
Install GeoNode-Project directly from scratch

If you didn’t install GeoNode-Core earlier and you wanted to install GeoNode-Project directly, please follow these steps:

1. Create a virtual environment as follows:

   $ vim ~/.bashrc
   # add the following line to the bottom
   $ source /usr/share/virtualenvwrapper/virtualenvwrapper.sh

   $ source /usr/share/virtualenvwrapper/virtualenvwrapper.sh
   $ mkvirtualenv --python=/usr/bin/python3 my_geonode

   Alternatively you can also create the virtual env like below
   $ python3.8 -m venv /home/geonode/dev/.venvs/my_geonode
   $ source /home/geonode/dev/.venvs/my_geonode/bin/activate

2. Clone the geonode-project repo from Github

   $ git clone https://github.com/GeoNode/geonode-project.git -b 4.1.x

3. Install Django framework as follows

   $ pip install Django==3.2.13

4. Use django-admin.py to create a project “my_geonode” from a GeoNode-Project template as follows:

   $ django-admin startproject --template=./geonode-project -e py,sh,md,rst,json,yml,ini,
   → env,sample,properties -n monitoring-cron -n Dockerfile my_geonode

5. Install all the requirements for the GeoNode-Project and install the GeoNode-Project using pip

   $ cd my_geonode
   $ pip install -r requirements.txt --upgrade
   $ pip install -e . --upgrade

6. Install GDAL Utilities for Python

   $ pip install pygdal="\gdal-config --version\".*" # or refer to the link <Install GDAL
   → for Development <https://training.geonode.geo-solutions.it/005_dev_workshop/004_devel_
   → env/gdal_install.html>

7. Install GeoServer and Tomcat using paver

   $ paver setup
   $ paver sync
   $ paver start

8- Visit http://localhost:8000/
1.36.3 How to run GeoNode Core for development

In order to start GeoNode Core for development, you need to make sure that no GeoNode instance is running first. This can be done by running the following commands:

```bash
$ cd /home/user/geonode
$ paver stop_geoserver
$ paver stop_django
```

Then you need to start both geoserver and django services as follows:

```bash
$ paver start_geoserver
$ paver start_django
```

Now you can visit your GeoNode GUI by typing `http://localhost:8000` into your browser window.

1.36.4 How to run GeoNode Project for development

In order to run a project for development, the following steps have to be followed:

1. Make sure there is no running instance of GeoNode first by running the following command:

```bash
$ cd /home/user/my_geonode
$ paver stop
```

The above command will stop all services related to GeoNode if running.

2. Start the servers by running `paver start` as follows:

```bash
$ paver start
```

Now you can visit your geonode project site by typing `http://localhost:8000` into your browser window.

1.36.5 Start MapStore2 client in development mode

Pre-requisites

1. You need a running instance of GeoNode somewhere; in this specific example we assume GeoNode is running on `http://localhost:8000`
Install needed packages

```
sudo apt install nodejs npm
```

Prepare the source code

```
git clone --recursive https://github.com/GeoNode/geonode-mapstore-client.git geonode-mapstore-client-dev
```

Compile MapStore2 Client

```
cd geonode-mapstore-client/geonode_mapstore_client/client/
npm update
npm install
npm run compile
```

Edit the file `env.json`

```
vim env.json
```

```
{
  "DEV_SERVER_HOST": "localhost:8000",
  "DEV_SERVER_HOST_PROTOCOL": "http"
}
```

Run MapStore2 in Development mode

```
npm run start
```

Connect to `http://localhost:8081`

This is a proxied version of GeoNode form MapStore2 client. **To upload new layers user the original GeoNode.**

Everytime you render a map, from GeoNode layers details page or map creation, you will access to the MapStore2 dev mode runnig code.

You can now update the code on the fly.
Example 1: Disable the PrintPlugin from the Layer Details small map

```javascript
BurgerMenuPlugin: require('../MapStore2/web/client/plugins/BurgerMenu'),
ScaleBoxPlugin: require('../MapStore2/web/client/plugins/ScaleBox'),
MapFooterPlugin: require('../MapStore2/web/client/plugins/MapFooter'),
// PrintPlugin: require('../MapStore2/web/client/plugins/Print'),
TimelinePlugin: require('../MapStore2/web/client/plugins/Timeline'),
PlaybackPlugin: require('../MapStore2/web/client/plugins/Playback'),
```

Example 2: Disable the MousePositionPlugin from the big maps

```javascript
SaveAsPlugin: require('../MapStore2/web/client/plugins/SaveAs').default,
MetadataExplorerPlugin: require('../MapStore2/web/client/plugins/MetadataExplorer'),
GridContainerPlugin: require('../MapStore2/web/client/plugins/GridContainer'),
StyleEditorPlugin: require('../MapStore2/web/client/plugins/StyleEditor'),
TimelinePlugin: require('../MapStore2/web/client/plugins/Timeline'),
PlaybackPlugin: require('../MapStore2/web/client/plugins/Playback'),
// MousePositionPlugin: require('../MapStore2/web/client/plugins/MousePosition'),
SearchPlugin: require('../MapStore2/web/client/plugins/Search'),
SearchServicesConfigPlugin: require('../MapStore2/web/client/plugins/SearchServicesConfig'),
```

1.36.6 Workshops

The workshops documentation demonstrates few examples on how to utilize GeoNode-Project in order to extend/customize GeoNode’s functionalities according to your business. The covered topics include the following:

1- Customize your GeoNode with the geonode-project
2- Customize the look and feel
3- Create your ResourceBase Metadata
4- Create your own django app
5- Add a custom model
6- Permissions and APIs
7- Deploy your GeoNode
1- Customise your GeoNode with the geonode-project

In this example, GeoNode-Project is cloned to create a template instance in which the rest of the examples will be building on top of it.

1- Assuming you already installed GeoNode-Core, firstly we need to create a GeoNode-Project template and this can be achieved from the following command:

```bash
django-admin.py startproject my_geonode --template=https://github.com/GeoNode/geonode-project/archive/master.zip -e py,rst,json,yml,ini,env,sample -n Dockerfile
```

Here, django-admin is used with startproject option to create my_geonode project copying the template which is passed as GeoNode-project Github repo. It also includes “py,rst,json,yml,ini,env,sample” extensions

2- Once the cloning finished, the next step is to install the GeoNode-Project we just downloaded as follows:

```bash
pip install -e my_geonode
```

3- Install geoserver using paver as follows

```bash
cd /home/geonode/my_geonode/src
paver setup
```

4- Note the GeoNode database connection parameters mentioned in the .env.sample.py file. Rename it to .env then use `psql` to create the required user and grant the required privileges as follows:

```bash
su postgres
createdb geonode
psql
```

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

**Warning:** Don’t forget to exit from postgres user before executing the following commands

5- Run GeoNode using paver

```bash
cd /home/geonode/my_geonode/src
paver start
```

**Note:** You may find this warning message: You have 132 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): account, actstream, admin, agen_ratings, announcements, auth, avatar, base, contenttypes, dialogos, documents, favorite, geonode_client, geonode_themes, groups, guardian, invitations, layers, maps, mapstore2_adapter, monitoring, oauth2_provider, people, pinax_notifications, services, sessions, sites, socialaccount, taggit, tastypie, upload, user_messages. Which means you have some sql statements not executed yet and you need to run the “migrate” to sync your database first then “paver start” again as follows:

```bash
python manage.py migrate
paver start
```
Warning: If encountered this message: (Invalid HTTP_HOST header: '0.0.0.0:8000'). You may need to add `u'0.0.0.0'` to ALLOWED_HOSTS. It can be fixed in the settings.py file. You will need to add: `ALLOWED_HOSTS = ['0.0.0.0']` in settings.py

6- Once the previous step is done, you can visit 0.0.0.0:8000 to view the GUI of GeoNode. However, we still don’t have an account in order to login from the GUI. This can be done using “paver sync”. The command will create sync with latest fixtures and also creates a superuser “admin” with default password “admin”

7- Use the created account to login from the GUI through localhost:8000 or 0.0.0.0:8000

2- Customize the look and feel

In this section we will change the look and feel of GeoNode, in particular we will do some customization to help understanding how the template inheritance works and how to add new stuff to your GeoNode. The changes will include the home page, the top menu, the footer and a generic GeoNode page.

Homepage:

The geonode-project provides some predefined templates to change the home page and the general site content.

In the “my_geonode/src/my_geonode/templates/geonode-mapstore-client/snippets” directory we can files with similar names as the geonode-mapstore-client. This way we can override the different parts of the site eg the header, menu, body content and the footer.

Create a file name hero.html and add the following.

The theme:
To change the theme of our geonode-project we can act on the site_base.css file available in the “my_geonode/my_geonode/static/css” folder.

The file is empty so we can inspect elements of the home page with the browser’s developer tools and define css rules in there.

For example, if we want to change the background of the jumbotron, in this file we can add

```css
.msgapi .gn-hero .jumbotron { background: red }
```

Then once we refreshed the browser, we should see the change as follows:

![My GeoNode](image)

The top menu:

Now we can make some changes that will apply to the whole site. We can add an item to both the left and right side of the top menu bar.

This can be done by creating a `get_menu_json.py` under `templatetags` folder to override GeoNodes default menu.

```python
@register.simple_tag(takes_context=True)
def get_base_right_topbar_menu(context):
    is_mobile = _is_mobile_device(context)

    if is_mobile:
        return []

    return [
        {
            "type": "link",
            "href": "/",
            "label": "Custom 3"
        },
        {
            "type": "link",
            "href": "/",
            "label": "Custom 4"
        },
    ]

@register.simple_tag(takes_context=True)
def get_base_left_topbar_menu(context):
    is_mobile = _is_mobile_device(context)
```

(continues on next page)
On browser refresh you will see a new entry in the nav bar which is persistent to the whole site.

Modify functionality

In this section, we will patch the ResourceBase of GeoNode and update the Templates in order to add one more field to the Metadata Schema.

We will add a DOI field to the ResourceBase model and modify the Templates in order to show the new field both into the Metadata Wizard page.

**Note:** Make sure to be inside “my_geonode” directory to execute the following commands

Customizing metadata can be achieved from the model which is defined in the core at “geonode/geonode/base/models.py” as follows:

```python
# internal fields
uuid = models.CharField(max_length=36)
owner = models.ForeignKey(
    settings.AUTH_USER_MODEL,
    blank=True,
    null=True,
    related_name='owned_resource',
    verbose_name=_('Owner'))
contacts = models.ManyToManyField(
    settings.AUTH_USER_MODEL,
    through='ContactRole')
```
title = models.CharField(_('title'), max_length=255, help_text=_('
  name by which the cited resource is known'))
alternate = models.CharField(max_length=128, null=True, blank=True)
date = models.DateTimeField(_('
date'),
  default=now,
  help_text=date_help_text)
date_type = models.CharField(_('
date type'),
  max_length=255,
  choices=VALID_DATE_TYPES,
  default='publication',
  help_text=date_type_help_text)
edition = models.CharField(_('
edition'),
  max_length=255,
  blank=True,
  null=True,
  help_text=edition_help_text)
abstract = models.TextField(_('
abstract'),
  max_length=2000,
  blank=True,
  help_text=abstract_help_text)
purpose = models.TextField(_('
purpose'),
  max_length=500,
  null=True,
  blank=True,
  help_text=purpose_help_text)
maintenance_frequency = models.CharField(_('
maintenance frequency'),
  max_length=255,
  choices=UPDATE_FREQUENCIES,
  blank=True,
  null=True,
  help_text=maintenance_frequency_help_text)

To add fields directly to the ResourceBase Class without actually modifying it, this can be done from “my_geonode/src/my_geonode/apps.py” file

The “ready” method is invoked at initialization time and can be currently used to tweak your app in several ways

```python
class AppConfig(BaseAppConfig):

    name = "my_geonode"
    label = "my_geonode"

    def ready(self):
        super(AppConfig, self).ready()
        run_setup_hooks()
```

Now we will add the “patch_resource_base” method to the AppConfig and execute it from the ready method as follows:
from django.db import models
from django.utils.translation import ugettext_lazy as _

class AppConfig(BaseAppConfig):
    name = "my_geonode"
    label = "my_geonode"

    def _get_logger(self):
        import logging
        return logging.getLogger(self.__class__.__module__)

    def patch_resource_base(self, cls):
        self._get_logger().info("Patching Resource Base")
        doi_help_text = _("A DOI will be added by Admin before publication.")
        doi = models.TextField(
            _('DOI'),
            blank=True,
            null=True,
            help_text=doi_help_text)
        cls.add_to_class('doi', doi)

    def ready(self):
        super(AppConfig, self).ready()
        run_setup_hooks()

        from geonode.base.models import ResourceBase
        self.patch_resource_base(ResourceBase)

Note: you will need to perform migrations as follows: - Add field doi to resourcebase

Once you run python manage.py migrate:

Running migrations:
Applying announcements.0002_auto_20200119_1257... OK
Applying base.0031_resourcebase_doi... OK
Applying people.0027_auto_20200119_1257... OK

Till now, we have patched the DB, however, it is not yet sufficient as we still need to display the added field.

Let’s extend the default templates so that we can show the newly added field
3- Create your own django app

In this section, we will demonstrate how to create and setup the skeleton of a custom app using the django facilities. The app will add a geocollections functionality to our GeoNode.

The Geocollections app allows to present in a single page, resources and users grouped by a GeoNode Group. We can assign arbitrary resources to a Geocollection, a Group and a name that will be also used to build a dedicated URL.

*Note:* Make sure to be inside “my_geonode” directory to execute the following commands.

Create the django app

Django gives us an handy command to create apps. We already used startproject to create our geonode-project, now we can use startapp to create the app.

```
python manage.py startapp geocollections
```

This will create a folder named geocollections that contains empty models and views.

We need to add the new app to the INSTALLED_APPS of our project. inside “my_geonode/src/my_geonode/settings.py”:

```
INSTALLED_APPS += (PROJECT_NAME,) to be: INSTALLED_APPS += (PROJECT_NAME, 'geocollections',)
```

Add a custom model

In this section, we will add a custom model and the related logic as follows:

- Add a new model
- Add urls and views
- Add admin panel
- Add the template

```
vim geocollections/models.py
```

```python
from django.db import models
from geonode.base.models import ResourceBase
from geonode.groups.models import GroupProfile

class Geocollection(models.Model):
    """
    A collection is a set of resources linked to a GeoNode group
    """
    group = models.ForeignKey(GroupProfile, related_name='group_collections')
    resources = models.ManyToManyField(ResourceBase, related_name='resource_collections')
    name = models.CharField(max_length=128, unique=True)
    slug = models.SlugField(max_length=128, unique=True)

    def __unicode__(self):
        return self.name
```
At this point we need to ask django to create the database table. Django since version 1.8 has embedded migrations mechanism and we need to use them in order to change the state of the db.

**Note:** Make sure to be inside “my_geonode” directory to execute the following commands

```
python manage.py makemigrations
# the above command informs you with the migrations to be executed on the database
python manage.py migrate
```

Next we will use django generic view to show the collections detail. Add the following code in the views.py file:

```
vim geocollections/views.py

```from django.views.generic import DetailView
from .models import Geocollection
class GeocollectionDetail(DetailView):
    model = Geocollection
```

Add url configuration

In order to access the created view we also need url mapping. We can create a urls.py file containing a url mapping to our generic view:

```
vim geocollections/urls.py

```from django.conf.urls import url
from .views import GeocollectionDetail
urlpatterns = [
    url(r'^(?P<slug>[-\w]+)/$', GeocollectionDetail.as_view(), name='geocollection-detail'),
]
```

We also need to register the app urls in the project urls. So let’s modify the “my_geonode” urls.py file adding the following:

```
vim my_geonode/src/my_geonode/urls.py

```...
urlpatterns += [
    url(r'geocollections/', include('geocollections.urls')),
]
...```
We need a user interface where we can create geocollections. Django makes this very easy, we just need the admin.py file as follows:

```
vim geocollections/admin.py
```

```python
from django.contrib import admin
from .models import Geocollection

class GeocollectionAdmin(admin.ModelAdmin):
    prepopulated_fields = {'slug': ('name',)}
    filter_horizontal = ('resources',)

admin.site.register(Geocollection, GeocollectionAdmin)
```

Now we can visit the admin page and create a geocollection from there as follows:

![Image of admin page]

Adding the template

Now we need the template where the geocollection detail will be rendered. Let's create a geocollections directory inside the “my_geonode/templates” directory with a file named geocollection_detail.html:

```
mkdir -p my_geonode/templates/geocollections/
vim my_geonode/templates/geocollections/geocollection_detail.html
```
To check the results, create a group in the geonode ui interface and load one or more layers/documents
login into the admin panel -> geocollections and create a geocollections
Visit http://localhost:8000/geocollections/<the-name-of-the-created-geocollection> and view the results.
Now that you know how to customize an html template, you can tune the page as you prefer.

Permissions and APIs

In this section we will add some more advanced logic like permissions and APIs. The permissions in GeoNode are
managed with django-guardian, a library which allow to set object level permissions (django has table level authorization).
The APIs are implemented through django-tastypie.
The topics to be covered include:
• Permissions on who can view the geocollection
• How to add templated and js to embed a permission ui in our geocollection detail page
• API to serve json serialized searchable endpoint

Permissions logic (permissions objects)
We need to add the permissions object to the database. We can do this by adding the following meta class to our
Geocollection model, guardian will take care of creating the objects for us.

```python
class Meta:
    permissions = (           ('view_geocollection', 'Can view geocollection'),
                     )
```

Then run “python manage.py makemigrations” and “python manage.py migrate” to install them

Permissions logic (set_default)
Let’s add a method that will be used to set the default permissions on the Geocollections. We can add this logic to the
Geocollection model but could also be a generic Mix-in similar to how it is implemented in GeoNode.

```python
from django.contrib.auth.models import Group
from django.contrib.auth import get_user_model
```

(continues on next page)
from django.contrib.contenttypes.models import ContentType
from django.conf import settings
from guardian.shortcuts import assign_perm

def set_default_permissions(self):
    """
    Set default permissions.
    """

    self.remove_object_permissions()

    # default permissions for anonymous users
    anonymous_group, created = Group.objects.get_or_create(name='anonymous')

    if settings.DEFAULT_ANONYMOUS_VIEW_PERMISSION:
        assign_perm('view_geocollection', anonymous_group, self)

    # default permissions for group members
    assign_perm('view_geocollection', self.group, self)

Permissions logic (methods)

Now we need a method to add generic permissions, we want to be able to assign view permissions to groups and single users. We can add this to our Geocollection model

def set_permissions(self, perm_spec):
    anonymous_group = Group.objects.get(name='anonymous')
    self.remove_object_permissions()

    if 'users' in perm_spec and "AnonymousUser" in perm_spec['users']:
        assign_perm('view_geocollection', anonymous_group, self)

    if 'users' in perm_spec:
        for user, perms in perm_spec['users'].items():
            user = get_user_model().objects.get(username=user)
            assign_perm('view_geocollection', user, self)

    if 'groups' in perm_spec:
        for group, perms in perm_spec['groups'].items():
            group = Group.objects.get(name=group)
            assign_perm('view_geocollection', group, self)

def remove_object_permissions(self):
    from guardian.models import UserObjectPermission, GroupObjectPermission
    UserObjectPermission.objects.filter(content_type=ContentType.objects.get_for_model(self),
                                         object_pk=self.id).delete()
    GroupObjectPermission.objects.filter(content_type=ContentType.objects.get_for_model(self),
                                          object_pk=self.id).delete()

Permissions logic (views.py)

We can add now a view to receive and set our permissions, in views.py:
import json
from django.core.exceptions import PermissionDenied
from django.http import HttpResponse
from django.contrib.auth import get_user_model

User = get_user_model()

def geocollection_permissions(request, collection_id):
    collection = Geocollection.objects.get(id=collection_id)
    user = User.objects.get(id=request.user.id)

    if user.has_perm('view_geocollection', collection):
        return HttpResponse(
            'You have the permission to view. please customize a template for this view',
            content_type='text/plain')

    if request.method == 'POST':
        success = True
        message = "Permissions successfully updated!"
        try:
            permission_spec = json.loads(request.body)
            collection.set_permissions(permission_spec)

            return HttpResponse(
                json.dumps({'success': success, 'message': message}),
                status=200,
                content_type='text/plain')
        except:
            success = False
            message = "Error updating permissions :(
        return HttpResponse(
            json.dumps({'success': success, 'message': message}),
            status=500,
            content_type='text/plain')

Permissions logic (url)
Lastly we need a url to map our client to our view, in urls.py

from django.conf.urls import url
from .views import GeocollectionDetail, geocollection_permissions

urlpatterns = [
    url(r'^(?P<slug>[-\w]+)/$',
        GeocollectionDetail.as_view(),
        name='geocollection-detail'),
    url(r'^permissions/$',
        geocollection_permissions,
        name='geocollection-permissions'),
]
This url will be called with the id of the geocollection, the id will be passed to the view in order to get the permissions.

**Warning:** A note on the client part, the server side logic is just one part necessary to implement permissions.

A checklist of what is necessary:

- A template snippet that can be embedded in the geocollection_detail.html, you can copy and simplify: _permissions_form.html and _permissions.html (in geonode/templates)
- A javascript file that will collect permissions settings and send them to the server, you can copy and simplify: _permissions_form_js.html (in geonode/templates)

**API**

The GeoNode API system easily allows to plug in new APIs. This section demonstrates the required steps:

We need first to create an api.py file in our geocollection app.

```python
import json
from tastypie.resources import ModelResource
from tastypie import fields
from tastypie.constants import ALL_WITH_RELATIONS, ALL

from geonode.api.api import ProfileResource, GroupResource
from geonode.api.resourcebase_api import ResourceBaseResource

from .models import Geocollection

class GeocollectionResource(ModelResource):
    users = fields.ToManyField(ProfileResource, attribute=lambda bundle: bundle.obj._group.group.user_set.all(), full=True)
    group = fields.ToOneField(GroupResource, 'group', full=True)
    resources = fields.ToManyField(ResourceBaseResource, 'resources', full=True)

    class Meta:
        # (continues on next page)
```
queryset = Geocollection.objects.all().order_by('-group')
ordering = ['group']
allowed_methods = ['get']
resource_name = 'geocollections'
filtering = {
    'group': ALL_WITH_RELATIONS,
    'id': ALL
}

API authorization

We want the API to respect our custom permissions, we can easily achieve this by adding the following to the beginning of api.py:

```python
from tastypie.authorization import DjangoAuthorization
from guardian.shortcuts import get_objects_for_user

class GeocollectionAuth(DjangoAuthorization):
    def read_list(self, object_list, bundle):
        permitted_ids = get_objects_for_user(
            bundle.request.user, 'geocollections.view_geocollection').values('id')

        return object_list.filter(id__in=permitted_ids)

    def read_detail(self, object_list, bundle):
        return bundle.request.user.has_perm(
            'view_geocollection', bundle.obj)
```

And this to the GeocollectionResource Meta class:

```python
authorization = GeocollectionAuth()
```

Add a url for our API

In order to publish our API we need a url and we want that url to appear under the GeoNode’s /api domain.

The final url for our API has to be /api/geocollections.

We can inject the url into the GeoNode API by adding the following lines to “my_geonode/urls.py” file:

```python
from geonode.api.urls import api
from geocollections.api import GeocollectionResource
api.register(GeocollectionResource())
```

And add the following in the urlpatterns:
url(r'', include(api.urls)),

The final result will be:

```python
from django.conf.urls import url, include
from django.views.generic import TemplateView
from geonode.urls import urlpatterns
from geonode.api.urls import api
from geocollections.api import GeocollectionResource

api.register(GeocollectionResource())

urlpatterns += [
    # include your urls here
    url(r'', include(api.urls)),
    url(r'^geocollections/', include('geocollections.urls')),
]
```

Let’s test permissions on API

We can test the permissions on API by manually set a permission from the command line and check that the API respects it.

With running `python manage.py shell` from inside our “my_geonode” folder, it opens a geonode shell.

A perm spec could look like this:

```python
perms = {
    'users': {
        'AnonymousUser': ['view_geocollection'],
        'alessio': ['view_geocollection']
    }
}
```

and we can assign the permissions with:

```python
from geocollections.models import Geocollection
Geocollection.objects.first().set_permissions(perms)
```

our http://localhost:8000/api/geocollections should now list the geocollection.

If you remove the ‘AnonymousUser’ line from perms and assign again the permissions it will disappear.

```python
perms = {
    'users': {
        'alessio': ['view_geocollection']
    }
}
```

**Deploy your GeoNode**

So far we demonstrated how to modify, extend and style our GeoNode in dev mode but now it’s time to go on production. In this section we will clarify how to:

- commit your work on GitHub
- setup your server
• setup your GeoNode for production

Push to GitHub

It is always a good practice to keep your code in a remote repository, GitHub is one of the options and is indeed the most used.

It is assumed that you already have a GitHub account and that you have git installed and configured with your name and email.

We will push only the my_geonode folder to GitHub and as we knew earlier, GeoNode for us is a dependency and we’ll just reinstall it as it is on the server.

Steps to push your code to GitHub:

• Create an empty repository in GitHub and copy its address
• In my_geonode, run git init to initialize an empty repository
• Add your remote repository address with `git remote add yourname yourremoteaddress`
• edit `.gitignore` adding all `*.pyc` files
• `git add *` to add all content of my_geonode
• `git commit -m 'initial import'` to make the initial commit
• `git push yourname master` to push the code to the GitHub repository

Setup the server

There are several options for deploying GeoNode projects on servers. In this section, we explain how to deploy it on Ubuntu server 18.04 using system-wide installation

Note: For quick installation, follow the INSTALLING documentation at http://docs.geonode.org/en/master/install/core/index.html

Setup our my_geonode

We need now to install the developed “my_geonode” project following these steps:

• `git clone from your repository (in the folder of your preference)`
• `sudo pip install -e my_geonode`
• edit the settings where needed
• edit `/etc/apache2/sites-enabled/geonode.conf` replacing the wsgi path to the `my_geonode/my_geonode/wsgi.py` file
• add the apache rights to the “my_geonode” folder with a directory like

```
<Directory "/path/to/my_geonode/"
    Order allow,deny
    Require all granted
</Directory>
```

• Test your server.

This documentation helps developers to install GeoNode-Core and GeoNode-Project from different scenarios. GeoNode-Project can be installed on top of GeoNode-Core if already installed. Also GeoNode-Project can be installed from scratch as it has GeoNode-Core as a prerequisite.
HTTP ROUTING TABLE

/api
GET /api/v2/, 525
GET /api/v2/categories/, 525
GET /api/v2/categories/{id}/, 526
GET /api/v2/datasets/, 526
GET /api/v2/datasets/{id}/, 526
GET /api/v2/datasets/{id}/maplayers/, 528
GET /api/v2/datasets/{id}/maps/, 528
GET /api/v2/datasets/{id}/{field_name}/, 528
GET /api/v2/documents/, 529
GET /api/v2/documents/{id}/, 529
GET /api/v2/documents/{id}/linked_resources/, 530
GET /api/v2/geoapps/, 531
GET /api/v2/geoapps/{id}/, 532
GET /api/v2/geoapps/{id}/{field_name}/, 533
GET /api/v2/groups/, 533
GET /api/v2/groups/{id}/, 535
GET /api/v2/groups/{id}/managers/, 536
GET /api/v2/groups/{id}/members/, 537
GET /api/v2/groups/{id}/resources/, 537
GET /api/v2/harvesters/, 537
GET /api/v2/harvesters/{harvester_id}/harvestable_resources/, 538
GET /api/v2/harvesters/{id}/, 539
GET /api/v2/harvesting-sessions/, 540
GET /api/v2/harvesting-sessions/{id}/, 541
GET /api/v2/keywords/, 541
GET /api/v2/keywords/{id}/, 541
GET /api/v2/management/commands/, 541
GET /api/v2/management/commands/{cmd_name}/, 542
GET /api/v2/management/commands/{cmd_name}/jobs/, 542
GET /api/v2/management/commands/{cmd_name}/jobs/{id}/, 543
GET /api/v2/management/commands/{cmd_name}/status/, 543
GET /api/v2/management/jobs/, 544
GET /api/v2/management/jobs/{id}/, 544
GET /api/v2/management/jobs/{id}/status/, 544
GET /api/v2/maps/, 545
GET /api/v2/maps/{id}/, 545
GET /api/v2/maps/{id}/datasets/, 546
GET /api/v2/maps/{id}/maplayers/, 546
GET /api/v2/maps/{id}/{field_name}/, 546
GET /api/v2/owners/, 547
GET /api/v2/owners/{id}/, 547
GET /api/v2/regions/, 547
GET /api/v2/regions/{id}/, 547
GET /api/v2/resource-service/execution-status/{ execution_id }, 548
GET /api/v2/resources/, 548
GET /api/v2/resources/approved/, 555
GET /api/v2/resources/favorites/, 556
GET /api/v2/resources/featured/, 556
GET /api/v2/resources/published/, 556
GET /api/v2/resources/resource_types/, 556
GET /api/v2/resources/{id}/, 549
GET /api/v2/resources/{id}/extra_metadata/, 551
GET /api/v2/resources/{id}/permissions/, 553
GET /api/v2/resources/{id}/ratings/, 554
GET /api/v2/resources/{id}/{field_name}/, 551
GET /api/v2/schema/, 557
GET /api/v2/tkeywords/, 557
GET /api/v2/tkeywords/{id}/, 557
GET /api/v2/upload-parallelism-limits/, 558
GET /api/v2/upload-parallelism-limits/{slug}/, 559
GET /api/v2/upload-size-limits/, 559
GET /api/v2/upload-size-limits/{slug}/, 560
GET /api/v2/uploads/, 560
GET /api/v2/uploads/{id}/, 561
GET /api/v2/users/, 561
GET /api/v2/users/{id}/, 563
GET /api/v2/users/{id}/groups/, 564
GET /api/v2/users/{id}/resources/, 565
POST /api/v2/geoapps/, 531
POST /api/v2/groups/, 534
POST /api/v2/harvesters/, 538
POST /api/v2/management/commands/{cmd_name}/, 542
<table>
<thead>
<tr>
<th>Method</th>
<th>Route</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST</td>
<td>/api/v2/management/commands/{cmd_name}/jobs/</td>
<td>543</td>
</tr>
<tr>
<td>PATCH</td>
<td>/api/v2/geoapps/{id}/</td>
<td>533</td>
</tr>
<tr>
<td>PATCH</td>
<td>/api/v2/groups/</td>
<td>534</td>
</tr>
<tr>
<td>PATCH</td>
<td>/api/v2/groups/{id}/</td>
<td>536</td>
</tr>
<tr>
<td>PATCH</td>
<td>/api/v2/harvesters/{harvester_id}/harvestable-resources/</td>
<td>538</td>
</tr>
<tr>
<td>POST</td>
<td>/api/v2/maps/</td>
<td>545</td>
</tr>
<tr>
<td>POST</td>
<td>/api/v2/resources/</td>
<td>548</td>
</tr>
<tr>
<td>POST</td>
<td>/api/v2/resources/create/{resource_type}/</td>
<td>555</td>
</tr>
<tr>
<td>PATCH</td>
<td>/api/v2/harvesters/{id}/</td>
<td>540</td>
</tr>
<tr>
<td>PATCH</td>
<td>/api/v2/management/commands/{cmd_name}/jobs/{id}/start/</td>
<td>543</td>
</tr>
<tr>
<td>PATCH</td>
<td>/api/v2/management/commands/{cmd_name}/jobs/{id}/stop/</td>
<td>543</td>
</tr>
<tr>
<td>PATCH</td>
<td>/api/v2/management/jobs/{id}/start/</td>
<td>544</td>
</tr>
<tr>
<td>PATCH</td>
<td>/api/v2/management/jobs/{id}/stop/</td>
<td>545</td>
</tr>
<tr>
<td>POST</td>
<td>/api/v2/resources/{id}/extra_metadata/</td>
<td>552</td>
</tr>
<tr>
<td>PATCH</td>
<td>/api/v2/harvesters/{id}/</td>
<td>540</td>
</tr>
<tr>
<td>PATCH</td>
<td>/api/v2/management/commands/{cmd_name}/jobs/{id}/start/</td>
<td>543</td>
</tr>
<tr>
<td>PATCH</td>
<td>/api/v2/management/commands/{cmd_name}/jobs/{id}/stop/</td>
<td>543</td>
</tr>
<tr>
<td>PATCH</td>
<td>/api/v2/management/jobs/{id}/start/</td>
<td>544</td>
</tr>
<tr>
<td>PATCH</td>
<td>/api/v2/management/jobs/{id}/stop/</td>
<td>545</td>
</tr>
<tr>
<td>POST</td>
<td>/api/v2/upload-parallelism-limits/</td>
<td>558</td>
</tr>
<tr>
<td>POST</td>
<td>/api/v2/upload-size-limits/</td>
<td>559</td>
</tr>
<tr>
<td>POST</td>
<td>/api/v2/uploads/</td>
<td>560</td>
</tr>
<tr>
<td>POST</td>
<td>/api/v2/uploads/upload/</td>
<td>561</td>
</tr>
<tr>
<td>POST</td>
<td>/api/v2/users/</td>
<td>562</td>
</tr>
<tr>
<td>PUT</td>
<td>/api/v2/datasets/id/</td>
<td>526</td>
</tr>
<tr>
<td>PUT</td>
<td>/api/v2/datasets/id/metadata/</td>
<td>529</td>
</tr>
<tr>
<td>PUT</td>
<td>/api/v2/documents/id/</td>
<td>529</td>
</tr>
<tr>
<td>PUT</td>
<td>/api/v2/geoapps/id/</td>
<td>532</td>
</tr>
<tr>
<td>PUT</td>
<td>/api/v2/groups/id/</td>
<td>535</td>
</tr>
<tr>
<td>PUT</td>
<td>/api/v2/harvesters/id/</td>
<td>539</td>
</tr>
<tr>
<td>PUT</td>
<td>/api/v2/maps/id/</td>
<td>545</td>
</tr>
<tr>
<td>PUT</td>
<td>/api/v2/resources/id/</td>
<td>549</td>
</tr>
<tr>
<td>PUT</td>
<td>/api/v2/resources/id/copy/</td>
<td>551</td>
</tr>
<tr>
<td>PUT</td>
<td>/api/v2/resources/id/extra_metadata/</td>
<td>552</td>
</tr>
<tr>
<td>PUT</td>
<td>/api/v2/resources/id/permissions/</td>
<td>553</td>
</tr>
<tr>
<td>PUT</td>
<td>/api/v2/resources/id/set_thumbnail/</td>
<td>555</td>
</tr>
<tr>
<td>PUT</td>
<td>/api/v2/resources/id/update/</td>
<td>555</td>
</tr>
<tr>
<td>PUT</td>
<td>/api/v2/users/id/</td>
<td>563</td>
</tr>
<tr>
<td>DELETE</td>
<td>/api/v2/groups/</td>
<td>534</td>
</tr>
<tr>
<td>DELETE</td>
<td>/api/v2/groups/{id}/</td>
<td>536</td>
</tr>
<tr>
<td>DELETE</td>
<td>/api/v2/harvesters/{id}/</td>
<td>540</td>
</tr>
<tr>
<td>DELETE</td>
<td>/api/v2/resources/</td>
<td>549</td>
</tr>
<tr>
<td>DELETE</td>
<td>/api/v2/resources/{id}/delete/</td>
<td>551</td>
</tr>
<tr>
<td>DELETE</td>
<td>/api/v2/resources/{id}/extra_metadata/</td>
<td>552</td>
</tr>
<tr>
<td>DELETE</td>
<td>/api/v2/resources/id/favorite/</td>
<td>552</td>
</tr>
<tr>
<td>DELETE</td>
<td>/api/v2/resources/id/permissions/</td>
<td>554</td>
</tr>
<tr>
<td>DELETE</td>
<td>/api/v2/users/</td>
<td>562</td>
</tr>
<tr>
<td>DELETE</td>
<td>/api/v2/users/id/</td>
<td>564</td>
</tr>
<tr>
<td>PATCH</td>
<td>/api/v2/datasets/</td>
<td>526</td>
</tr>
<tr>
<td>PATCH</td>
<td>/api/v2/datasets/id/</td>
<td>527</td>
</tr>
<tr>
<td>PATCH</td>
<td>/api/v2/documents/</td>
<td>529</td>
</tr>
<tr>
<td>PATCH</td>
<td>/api/v2/documents/id/</td>
<td>530</td>
</tr>
<tr>
<td>PATCH</td>
<td>/api/v2/geoapps/</td>
<td>532</td>
</tr>
</tbody>
</table>