OSGS

Release 1.0

Tim Sutton

Dec 09, 2021

CONTENTS:

Welcome to the Open Source Geospatial Stack!

In the open source geospatial world, there are different interesting technologies that have been developed to allow one to manipulate, visualize, publish, manage, edit, etc. geospatial data. These technologies are often in disparate projects. Programs like the QGIS project, are often ensemble applications that take some of these different technologies as components and use them to provide some higher level functionality. In some cases the integration is in silos or vertical applications, for example QGIS, PostGIS, Geoserver, etc. The aim of this project is to take a number of those different silos and incoporate them into a single platform, therefore making it really easy to access and publish the different types of services that are easily available on their own but not so easily available in a consolidated platform.

CHAPTER

ONE

INTRODUCTION

The Open Source GIS Stack by Kartoza is maintained in the kartoza / osgs repository.

1.1 What is the Open Source GIS Stack?

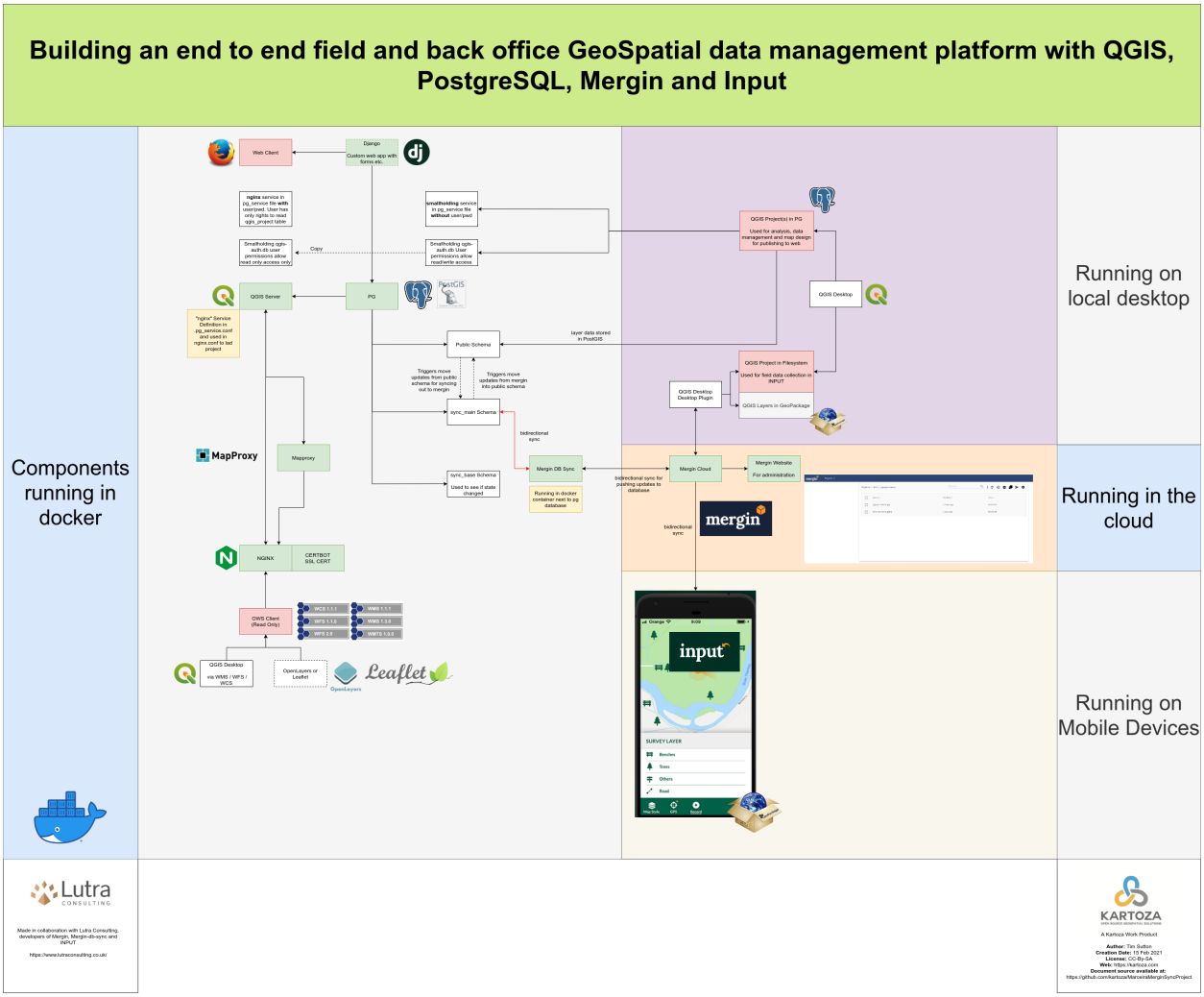
In the open source geospatial world, there are different interesting technologies that have been developed to allow one to manipulate, visualize, publish, manage, edit, etc. geospatial data. These technologies are often in disparate projects. Programs like the QGIS project, are often ensemble applications that take some of these different technologies as components and use them to provide some higher level functionality. In some cases the integration is in silos or vertical applications, for example QGIS, PostGIS, Geoserver, etc. The aim of this project is to take a number of those different silos and incoporate them into a single platform, therefore making it really easy to access and publish the different types of services that are easily available on their own but not so easily available in a consolidated platform.

The Open Source GIS Stack (OSGS) is a platform based on Docker, which is a container deployment environment. Containers are isolated application runtimes. They typically contain a minimal version of an operating system, typically Linux. In that minimal version of the operating system the containers will typically have one single task they are responsible for doing, for example running PostGIS or running Geoserver and so on. Each container has a very specific task and the design intent that you do not deploy mutliple services in a single container. You must deploy each service in its own container then orcherstrate those containers together. This is all managed using Docker and Docker Compose which is the orchestration environment for Docker. OSGS is simply a set of Docker orchestration routines to bring up different services in different containers, make them all talk together and present them as one cohesive architecture.

The Open Source GIS Stack (OSGS) is a platform based on Docker, which is a container deployment environment. Containers are isolated application runtimes. They typically contain a minimal version of an operating system, typically Linux. In that minimal version of the operating system the containers will typically have one single task they are responsible for doing, for example running PostGIS or running Geoserver and so on. Each container has a very specific task and the design intent that you do not deploy mutliple services in a single container. You must deploy each service in its own container then orcherstrate those containers together. This is all managed using Docker and Docker Compose which is the orchestration environmnt for Docker. OSGS is simply a set of Docker orchestration routines to bring up different services in different containers, make them all talk together and present them as one cohesive architecture.

The other important thing that OSGS does, is it provides some scripts to do various management commands and tasks, for example, bringing up services or down services, copying data, making backups, etc. These scripts are provided using a utility called “make” which is typically used by programmers to compile software, but can also be used to automate tasks. There is a Makefile which does all the high level tasks on the orchestrated architecture that OSGS provides.

1.2 Overview Diagram



Note: Anybody can take open source software and package it as Docker services. Therefore, when you are choosing which Docker service and containers to run, you can go and have a look at the various ways different people have packaged up a particular software and find one that works the best for you.

CHAPTER

TWO

INSTALLATION

2.1 Preparing the server

2.1.1 Basic Security

Unattended upgrades

This will automatically install only security fixes on a continual basis on your server.

```
sudo apt install unattended-upgrades
```

ssh

Disable password authentication for SSH

```
sudo vim /etc/ssh/sshd_config
```

Set this:

```
PasswordAuthentication no
Then do
sudo systemctl restart sshd.service
```

Crowdsec

https://crowdsec.net/

```
wget -qO - https://s3-eu-west-1.amazonaws.com/crowdsec.debian.pragmatic/crowdsec.asc␣
↪|sudo apt-key add - && echo "deb https://s3-eu-west-1.amazonaws.com/crowdsec.debian.
↪pragmatic/$(lsb_release -cs) $(lsb_release -cs) main" | sudo tee /etc/apt/sources.list.
↪d/crowdsec.list > /dev/null;
sudo apt-get update
sudo apt-get install crowdsec
```

Fail2ban

```
sudo apt install fail2ban
```

See: https://www.fail2ban.org/wiki/index.php/Main_Page

Firewall

```
sudo ufw allow ssh
sudo ufw enable
sudo ufw status
```

Should show something like this:

```
Status: active

To             Action   From
--             ------   ----
22/tcp           ALLOW    Anywhere
22/tcp (v6)        ALLOW    Anywhere (v6)
```

We will open more ports as they are needed.

Status monitoring

gotop is a great console based dashboard for monitoring your server.

Ubuntu:

```
sudo apt-get install golang
cd
go get github.com/cjbassi/gotop
chmod +x go/bin/gotop
sudo cp go/bin/gotop /usr/local/bin/
```

Fedora:

```
sudo dnf install golang
cd
go get github.com/cjbassi/gotop
chmod +x go/bin/gotop
sudo cp go/bin/gotop /usr/local/bin/
```

Now just type gotop whenever you want to see your terminal system monitor.

2.1.2 Additional Software

Docker

```
sudo apt install docker.io
sudo apt-get -y install python3-pip
sudo pip3 install docker-compose
```

Git, rpl, pwgen, Make and openssl

Needed for checking out our docker project and running the various make commands we provide.

```
sudo apt install git make rpl pwgen openssl apache2-utils
```

or fedora:

```
sudo dnf install openssl rpl git pwgen
```

2.1.3 Firewall

If you are using ufw, open port 80 and 443 as minimum. After the initial setup, you can again close port 80.

```
sudo ufw allow 80
sudo ufw allow 443
```

Move on to OSGS Installation

Ok we are ready to install OSGS! Go ahead to the initial configuration page now.

2.2 Wireguard VPN

You can further enhance the security of communications between clients and the OSGS server by configuring a Wireguard VPN. The goal here will be to tunnel all non-web based requests to the server through the VPN. This can include Postgres and Mosquitto connections, and others if they are published on a port. You can also move the actual management over SSH of the OSGS server itself into the VPN so that the only public facing ports of the server are https and the vpn service.

In this document we describe a scenario where the OSGS server is also a VPN server, but there are other potential scenarios such as using a second server and the VPN server and the OSGS as a VPN client.

2.2.1 Setting up the VPN Server

Note: There is NO public facing ssh port open on this host, you can only access ssh via the VPN. So for management when the VPN is down or you don't have your client set up yet, use your hosting provider's virtual console.

Note 2: Use these instructions at your own risk. Misconfiguring your server could lock you out of it permanently, so proceed with caution!

Note 3: We assume you have already carried out the *server_preparation* steps before following this guide.

Log in to your server

```
ssh yourhost
```

or use the hosting provider's console to log in if you do not already have SSH access.

Note: When logging in take note that the keyboard layout in the console is US if you are using hetzner for your provider, so if you have a different layout e.g. Portuguese you need to type as if you are on a US keyboard.

Initial Install

Assuming an ubuntu based server here...

```
sudo su -
apt update
apt upgrade
apt install wireguard
cd /etc/wireguard/
umask 077; wg genkey | tee privatekey | wg pubkey > publickey
ls -l privatekey publickey
```

Should show:

```
-rw------- 1 root root 45 Oct 31 23:33 privatekey
-rw------- 1 root root 45 Oct 31 23:33 publickey
```

Make a note of the private key and public key:

```
cat privatekey
cat publickey
```

Now configure the conf file:

```
vim /etc/wireguard/wg-osgs.conf
```

Note: You can run multiple Wireguard VPN clients and servers on the same host so we include osgs in the conf file name to make it clear what this VPN is to be used for.

Add this content:

```

## Set Up WireGuard VPN on Ubuntu By Editing/Creating wg0.conf File ##
[Interface]
## My VPN server private IP address ##
```

(continues on next page)

(continued from previous page)

```
Address = 192.168.7.1/24

## My VPN server port ##
ListenPort = 41194

## VPN server's private key i.e. /etc/wireguard/privatekey ##
PrivateKey = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

# Also enable NAT routing of all traffic through the VPN
# This is needed for VPN clients to be able to conenct to each other's services and ports
PostUp = iptables -A FORWARD -i wg0 -j ACCEPT; iptables -t nat -A POSTROUTING -o eth0 -j␣
↪MASQUERADE
PostDown = iptables -D FORWARD -i wg0 -j ACCEPT; iptables -t nat -D POSTROUTING -o eth0 -
↪j MASQUERADE

##
## All osgs peers (clients) should be added here
##

[Peer]
## Client VPN public key ##
PublicKey = XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

## client VPN IP address (note  the /32 subnet) ##
AllowedIPs = 192.168.7.2/32
```

Server Networking and Firewall Configuration

Note: Only do this for cases when you want to support peer to peer access in the VPN. In normal circumstances it is better to have an architecture where peers can talk to the server only and not to each other.

This section copied from https://linuxize.com/post/how-to-set-up-wireguard-vpn-on-ubuntu-20-04/#server-networking-and-firewall-configuration

IP forwarding must be enabled for NAT to work. Open the /etc/sysctl.conf file and add or uncomment the following line:

```
sudo vim /etc/sysctl.conf
```

Add this line:

```
net.ipv4.ip_forward=1
```

Save the file and apply the change:

```
sudo sysctl -p
```

Should show:

```
net.ipv4.ip_forward = 1
```

Firewall config

```
ufw allow 41194/udp
ufw status
```

Should show:

```
Status: inactive
```

Allow ssh connections originating from within the VPN

```
ufw allow from 192.168.7.0/24 to any port 22 proto tcp
```

For the same for each port that you want to be accessible only via the VPN e.g.

```
ufw allow from 192.168.7.0/24 to any port 5432 proto tcp
ufw allow from 192.168.7.0/24 to any port 8883 proto tcp
```

Would allow Postgres connections (5432) and Mosquitto (8883) over the VPN only.

On my test system the set of final UFW rules looks like this:

```
root@osgs:/etc/wireguard# ufw status numbered
Status: active

     To                         Action      From
     --                         ------      ----
[ 1] 80                         ALLOW IN    Anywhere
[ 2] 443                        ALLOW IN    Anywhere
[ 3] 41194/udp                  ALLOW IN    Anywhere
[ 4] 22/tcp                     ALLOW IN    192.168.7.0/24
[ 5] 5432/tcp                   ALLOW IN    192.168.7.0/24
[ 6] 8883/tcp                   ALLOW IN    192.168.7.0/24
```

i.e. only web and vpn traffic are publicly accessible and other services and ports need to be accessed from within the VPN.

Enable wireguard service in systemd

```
systemctl enable wg-quick@wg-osgs
systemctl start wg-quick@wg-osgs
systemctl status wg-quick@wg-osgs
```

A convenient way to do status checks is with the wg command. It will show all connected hosts.

```
wg
```

2.2.2 Allow incoming SSH only from the VPN

As indicated above, do:

```
ufw allow from 192.168.6.0/24 to any port 22 proto tcp
```

Also update your sshd to listen only on the wg interface

```
# Changed by Tim to listen on wireguard interface only
# Change XXX to your IP
ListenAddress 192.168.7.XXX
# Disabled by Tim
PermitRootLogin no
PasswordAuthentication no
```

2.2.3 Connecting with your client

For each client you need to create public/private keys on the client and a [peer] entry on the server, then restart client and server wireguard instances.

Install Ubuntu

```
apt install wireguard
sudo apt install openresolv
sudo apt install resolvconf
```

or (for fedora):

```
dnf install wireguard wireguard-tools
```

Configure client keys

```
cd /etc/wireguard/
umask 077; wg genkey | tee privatekey | wg pubkey > publickey
ls -l privatekey publickey
cat publickey
cat privatekey
```

Add client to the server

Log in to the server:

```
ssh vpn
sudo vim /etc/wireguard/wg-osgs.conf
```

Note: Steps below already partly done for the first host if you have followed the server setup notes above. You need to repeat this process for each host that needs to connect to the server via VPN.

Now add a new peer, using your public key from your client machine. Make sure to allocate a unique IP for it:

```
[Peer]
## Desktop/client VPN public key ##
PublicKey = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

## client VPN IP address (note  the /32 subnet) ##
AllowedIPs = 192.168.7.2/32
```

2.2.4 Configure on the client

sudo vim /etc/wireguard/wg-osgs.conf

```
[Interface]
## This Desktop/client's private key ##
PrivateKey = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx=

## Client ip address ##

# Replace .2 with your allocated ip address!
Address = 192.168.7.2/24


[Peer]
## Server public key ##
PublicKey = XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

## set ACL ##
AllowedIPs = 192.168.7.0/24

## Server's public IPv4/IPv6 address and port ##
Endpoint = XX.YY.ZZ.AA:41194

##  Key connection alive ##
PersistentKeepalive = 15
```

Enable and start wireguard

```
sudo systemctl enable wg-quick@wg-osgs
sudo systemctl start wg-quick@wg-osgs
sudo systemctl status wg-quick@wg-osgs
```

To bring it down again in the future you can do:

```
sudo wg-quick down wg-osgs
```

Or to restart:

```
sudo systemctl restart wg-quick@wg-osgs
```

And to check status:

```
sudo wg
```

Which should show something like this:

```
interface: wg-osgs
  public key: Kimironw1r21/RH3jbd97HTwtXjlUJFJAOXmBKuB+hg=
  private key: (hidden)
  listening port: 58616

peer: IdLyVLa0htZ82EKYWM9sXjB5ST15O3U5yYbNW+3XOQY=
  endpoint: XX.YY.ZZ.AA:41194
  allowed ips: 192.168.7.0/24
  transfer: 0 B received, 296 B sent
  persistent keepalive: every 15 seconds
```

2.2.5 Further configuration notes

Note: Only once you have verified that you can connect with a client.

Stopping Wireguard

```
systemctl stop wg-quick@wg-osgs
```

Configure ssh

For additional security you can set the SSH listenaddress to your VPN network so that it will not even respond to conneciton attempts from the wider internet.

```
vim /etc/ssh/sshd_config
```

Changed these (add them to the end of the file):

```
# Changed by Tim to listen on wireguard interface only
ListenAddress 192.168.7.1
# Disabled by Tim (should already be done in server_preparation step)
PermitRootLogin no
PasswordAuthentication no
```

Again, check carefully at the end of the file, some providers add a `PasswordAuthentication yes` to the bottom of the file which you want to override.

Enable the VPN

```
systemctl enable wg-quick@wg-osgs
```

Enable the firewall

Now enable the firewall (you probably want to be logged in at the service provider's virtual console here as your WAN connection will be dropped until you access via VPN).

```
ufw enable
ufw status
systemctl restart sshd
```

Should show:

```
Status: active

To                         Action      From
--                         ------      ----
41194/udp                  ALLOW       Anywhere
22/tcp                     ALLOW       192.168.7.0/24
41194/udp (v6)             ALLOW       Anywhere (v6)
```

Now go to https://github.com/kartoza/kartoza/wiki/WireGuard-Client-Configuration for client configs. . . .

2.3 Initial Configuration

2.3.1 User Group

Add yourself to the user group of docker so you don't need to sudo docker commands.

```
sudo usermod -a -G docker $USER
```

On linux you can run:

```
newgrp docker
```

To become part of the docker group. On other Operating Systems you should log out and in again to assume the upgraded permissions.

2.3.2 Project Checkout

Note you can check out the project anywhere, but for our examples we will use /home/web/osgs.

```
cd /home
sudo mkdir web
sudo chown timlinux.timlinux web
cd web
```

(continues on next page)

(continued from previous page)

```
git clone https://github.com/kartoza/osgs
cd osgs
```

2.3.3 Configuration

If you are going to use a self-signed certificate on a localhost (for testing):

```
make configure-ssl-self-signed
```

If you are going to use a letsencrypt signed certificate on a name host (for production):

```
make configure-letsencrypt-ssl
```

2.3.4 Fetching Docker Images

You can optionally prefetch all the docker images that are used in the stack.

```
[timlinux@fedora OpenSource-GIS-Stack]$ docker image ls
REPOSITORY                      TAG                 IMAGE ID       CREATED         SIZE
apache/superset                 latest-dev          dae66d8ef922   13 hours ago    1.9GB
pbf                             stable              24a6ed68e359   14 hours ago    252MB
swaggerapi/swagger-ui           latest              be6494c7edf4   32 hours ago    98.7MB
lutraconsulting/mergin-db-sync  latest              2f479ae94f5d   2 days ago      630MB
opendronemap/odm                latest              223a72b0d4e5   3 days ago      1.62GB
byrnedo/alpine-curl             latest              5208e5295614   9 days ago      6.9MB
redis                           latest              739b59b96069   10 days ago     105MB
nginx                           latest              62d49f9bab67   2 weeks ago     133MB
node                            14                  d6602e31594f   2 weeks ago     943MB
kartoza/docker-osm              imposm-latest       3d8739408ed0   3 weeks ago     1.08GB
kartoza/docker-osm              osmupdate-latest    bb1457cbd311   3 weeks ago     250MB
certbot/certbot                 latest              c8eac9ed295e   3 weeks ago     357MB
silexlabs/silex                 latest              be2ad993c076   4 weeks ago     1.38GB
klakegg/hugo                    0.82.0              2feec974b2db   5 weeks ago     46.1MB
openquake/qgis-server           stable              45d3c20d6088   2 months ago    1.26GB
kartoza/mapproxy                latest              d6df8e43db21   3 months ago    1.37GB
kartoza/postgis                 13.0                a85a969d604b   5 months ago    1.34GB
kartoza/geoserver               2.18.0              f6774787e652   6 months ago    1.88GB
postgrest/postgrest             latest              befe6cd943da   11 months ago   84.6MB
kartoza/docker-osm              osmenrich-latest    ccab39593491   22 months ago   949MB
jguyomard/hugo-builder          latest              650ec6415cde   2 years ago     57.2MB
quay.io/lkiesow/docker-scp      latest              b99cf24fe937   2 years ago     12.5MB
bytemark/webdav                 latest              c124350447bb   2 years ago     95.6MB
geodata/gdal                    latest              1e1929f80f44   3 years ago     1.29GB
```

```
docker-compose pull
```

2.4 Obtaining free fonts for your projects

There are two great sources of free fonts that you can use for your projects:

http://ftp.gnu.org/gnu/freefont/freefont-ttf-20120503.zip https://fonts.google.com/

There is a makefile target that can be run with:

```
make get-fonts
```

That will fetch all of these fonts for you. If you also fetch these fonts on your local machine and place them in your `~/.fonts` folder then you can use them in your local QGIS projects, know they will also be available to QGIS server if you publish that project as a web map.

Note: This and other makefile targets assume that you have not changed the `COMPOSE_PROJECT_NAME=osgisstack` environment variable in .env.

Note: The above make command fetches a rather large download!

2.5 Production Stack

2.5.1 Overview

In this section we will bring up the full production stack, but to do that we first need to get an SSL certificate issued. To facilitate this, there is a special, simplified, version of Nginx which has no reverse proxies in place and not docker dependencies. Here is an overview of the process:

1. Replace the domain name in your letsencrypt init script
2. Replace the email address in your letsencrypt init script
3. Replace the domain name in the certbot init nginx config file
4. Open up ports 80 and 443 on your firewall
5. Run the init script, ensuring it completed successfully
6. Shut down the minimal nginx
7. Replace the domain name in the production nginx config file
8. Generate passwords for geoserver, postgres, postgrest and update .env
9. Copy over the mapproxy template files
10. Run the production profile in docker compose

At the end of the process you should have a fully running production stack with these services:

IMAGE	PORTS	NAMES
nginx:alpine	0.0.0.0:80->80/tcp, 0.0.0.0:443->443/tcp	osgisstack_nginx_1
quay.io/lkiesow/docker-scp	22/tcp	osgisstack_scp_1
kartoza/hugo-watcher:main	1313/tcp	osgisstack_scp_1

The following ports will be accessible on the host to the docker services. You can, on a case by case basis, allow these through your firewall using ufw (uncomplicated firewall) to make them publicly accessible:

1. 80 - http:Only really needed during initial setup of your letsencrypt certificate

2. 443 - https: All web based services run through this port so that they are encrypted
3. 5432 - postgres: Only expose this publicly if you intend to allow remote clients to access the postgres database.
4. 2222 - scp: The is an scp/sftp upload mechanism to mobilise data and resources to the web site

For those services that are not exposed to the host, they are generally made available over 443/SSL via reverse proxy in the Nginx configuration.

Some things should still be configured manually and deployed after the initial deployment:

1. Mapproxy configuration
2. setup.sql (especially needed if you are planning to use postgrest)
3. Hugo content management
4. Landing page static HTML

And some services are not intended to be used as long running services. especially the ODM related services.

2.5.2 Configuration

We have written 2 make targets that automate the steps 1-10 described in the overview above. Either target will ask you for your domain name, legitimate email address and then go ahead and copy the templates over, replace placeholder domain names and email address, generate passwords for postgres etc. and then run the production stack. Remember you need to have ufw, rpl, make and pwgen installed before running either of the commands below.

If you are going to use a self-signed certificate on a localhost (for testing) run:

```
make configure-ssl-self-signed
```

If you are going to use a letsencrypt signed certificate on a name host (for production) run:

```
make configure-letsencrypt-ssl
```

2.6 Production Stack installation instructions for Windows Users

The Open Source GIS full production stack was developed for use on machines running a Linux distribution. Hence for Windows users, it is recommended that you download and install the following for a smooth user experience when using the Open Source GIS Stack (OSGS).

1. Windows Subsytem for Linux (WSL) and your Linux distribution of choice
2. Docker
3. Bash and Git for Windows

Note: The OSGS stack was tested using the Ubuntu on the Windows Subsystem for Linux (WSL)

For Ubuntu create the .ssh directory and the authorized_keys file. `cd ~` then follow these directions.

Next, you need to install the make utility for your Linux distribution. The make commands available for the OSGS platform are in the `Makefile`. You also need to install the following dependencies: rpl, pwgen, pip using:

```
sudo apt-get update
sudo apt install make
sudo apt-get install rpl
sudo apt-get install pwgen
```

(continues on next page)

(continued from previous page)

```
sudo apt-get install apache2-utils (#should solve your problem with missing htpasswd␣
↪binary)
sudo apt install python3-pip
```

Sign up for a Github account if you do not have one and fork the kartoza / osgs repository. Using the Git Bash terminal clone the forked repository onto your local machine. Start up your Linux distribution terminal and navigate to the location of the cloned repository.

CHAPTER

THREE

SERVICES

Here follows the list of services that are available in the OSGS stack.

- Items with a (PR) suffix can be considered production ready.
- Items with a (WIP) suffix can be considered a work in progress.

3.1 Nginx - PR

Nginx is a lightweight web server acting as a proxy in front of QGIS server, and as a server for the static HTML content.

Service name: nginx

Project Website: NGINX

Project Source Repository: nginx / nginx

Project Technical Documentation: nginx documentation

Docker Repository: nginx

Docker Source Repository: nginxinc / docker-nginx

3.1.1 Configuration

3.1.2 Deployment

3.1.3 Enabling

3.1.4 Disabling

3.1.5 Accessing the running services

3.1.6 Additional Notes

3.2 File Browser

3.3 Hugo Watcher - PR

Hugo is a static site generator. Hugo builds pages when you create or update your content. Since websites are viewed far more often than they are edited, Hugo is designed to provide an optimal viewing experience for the website's end

users and an ideal writing experience for website authors. *[1]*

The Hugo Watcher service watches for changes in the static content source files of the hugo site and rebuilds the site whenever a source file is changed.

Service name: hugo-watcher

Project Website: HUGO

Project Source Repository: gohugoio / hugo

Project Project Technical Documentation: Hugo Documentation

Docker Repository: kartoza/hugo-watcher

Docker Source Repository: kartoza / hugo-watcher

3.3.1 Deployment

```
make deploy-hugo
```

3.3.2 Enabling

```
make enable-hugo
```

3.3.3 Starting

```
make start-hugo
```

3.3.4 Stopping

```
make stop-hugo
```

3.3.5 Disabling

```
make disable-hugo
```

3.3.6 Logs

```
make hugo-logs
```

3.3.7 Shell

```
make hugo-shell
```

3.3.8 Backing up data

```
make backup-hugo
```

3.3.9 Restoring data

```
make restore-hugo
```

3.3.10 Accessing the running services

3.3.11 Additional Notes

3.3.12 References

[1] Hugo Authors. (2020, June 3). About Hugo. Hugo. https://gohugo.io/about/what-is-hugo/

3.4 Postgres and PostGIS - PR

PostgreSQL is a powerful, open source object-relational database management system (ORDBMS) that uses and extends the SQL language combined with many features that safely store and scale the most complicated data workloads.*[1][2]*

PostGIS is a spatial extension to for PostgreSQL object-relational database. This allows for GIS (Geographic Information Systems) objects to be stored in the database.*[3]*

The heart of any stack like the OpenSource GIS Stack would be PostGIS and PostgreSQL. This is because many of the other tools provided in the OSGS stack, for example Goeserver, QGIS Server and Mergin, all rely or can make use of PostGIS as a data storage platform and in some cases as a data anlysis platform. The OSGS stack provides PostgreSQL version 13 along with PostGIS which are provided by the the Kartoza PostGIS Docker container. There are a few considerations that were made when deploying this container. One, which is actually a general design consideration of the OSGS stack, is to try and make everything as secure as possible out of the box, rather than leaving security as an afterthought. To fulfill this consideration, what the OSGS stack does, is it spins up a PostgreSQL instance with PostGIS deployed in it and creates a separate user account for connecting to this instance. It also enables SSL encription by default, as required to the PostgreSQL database. This is so that if you choose to open the port as a public service, firstly, we are not using some well documented default password and username combination and secondly, all the traffic between your client application and the server is encrypted.

Service name: db

Project Website: PostgreSQL and PostGIS

Project Source Repository: postgres / postgres and postgis / postgis

Project Technical Documentation: PostgreSQL Documentation and PostGIS Documentation

Docker Repository: kartoza/postgis

Docker Source Repository: kartoza / docker-postgis

3.4.1 Configuration

The project includes detailed documentation so this section only contains details relevant to the Open Source GIS Stack configuration.

Database password:

Generate a strong password:

```
pwgen 20 1
```

Replace the default docker password for the postgres user with the strong password:

```
rpl "POSTGRES_PASSWORD=docker" "POSTGRES_PASSWORD=<strong password>" .env
```

Service file configuration:

Service files entries serve two scenarios:

1. They are needed for opening QGIS projects stored in postgres with PG connection URI because at the project URI you cannot use QGIS authdb. If you prefer to store your projects on the file system, you should rather remove these lines (whole nginx section) since the authentication from pg_conf/pg_service.conf can be done more securely by QGIS authdb.
2. Used by your QGIS Server projects to connect to the database once the project is opened from either the file system of the database. You can either specify your password and username in service file or for more advanced configuration you can store user / password credentials in a QGIS authdb file. Refer to the authdb section and in qgis_conf/qgis-auth.db and the readme in that folder.

On your local machine you should create your own service file with the same service name but connection details that make sense when using the database from your local machine. When you upload your projects into the stack they will connect using the settings from the server hosted service file below assuming you used the same service name.

To carry out the service file configuration, copy, rename then edit the pg_service file in pg_config as per the example below (note that we also substitute in the database password created in the steps above).

```
cp pg_conf/pg_service.conf.example \ pg_conf/pg_service.confpassword=docker
rpl password=<your password> pg_conf/pg_service.conf
```

Deployment

```
docker-compose --profile=postgres up -d
```

Note that the default configuration opens the postgresql service to all hosts. This is a potential security hole. If you open the port on the firewall e.g.

```
ufw allow 5432 tcp
```

Then be sure to connect from pg clients like psql or QGIS with SSL enabled so that passwords and data are not transmitted in clear text.

Validation

Create a local pg_service.conf file like the example below and save it in `~/.pg_service.conf` or similar as appropriate to your operating system (see https://www.postgresql.org/docs/12/libpq-pgservice.html for details on configuration options).

```
[os-gis-stack]
dbname=gis
port=5432
host=<hostname of your server>
user=<your password>
password=docker
```

Now pass the server parameter to psql and list the databases as per the example below:

```
[timlinux@fedora ~]$ psql service=os-gis-stack -l
List of databases
```

Name	Owner	Encoding	Collate	Ctype	Access privileges
gis	docker	UTF8	C.UTF-8	C.UTF-8	
postgres	postgres	UTF8	C.UTF-8	C.UTF-8	
template0	postgres	UTF8	C.UTF-8	C.UTF-8	=c/postgres
template1	postgres	UTF8	C.UTF-8	C.UTF-8	=c/postgres

```
(4 rows)
```

Test from QGIS is similar:

XXXXXXXXXXXXXX

Note that there was no need to supply any credentials other than the service file name.

3.4.2 Deployment

```
make deploy-postgres
```

3.4.3 Enabling

```
make enable-postgres
```

3.4.4 Configuration

```
make configure-postgres
```

3.4.5 Starting

```
make start-postgres
```

3.4.6 Stopping

```
make stop-postgres
```

3.4.7 Disabling

```
make  disable-postgres
```

3.4.8 Polling the service logs

```
make db-logs
```

3.4.9 Creating the service shell

```
make db-shell
```

3.4.10 Reinitializing the service

```
make reinitialise-postgres
```

3.4.11 Backing up data

To back up a QGIS project stored in db, run:

```
make db-qgis-project-backup
```

To back up the entire GIS postgres db, run:

```
make db-backup
```

To back up all postgres databases, run:

```
make db-backupall
```

To back up the mergin base schema from postgres db, run:

```
make db-backup-mergin-base-schema
```

3.4.12 Restoring data

To restore a previously backed up QGIS project to db, run:

```
make db-qgis-project-restore
```

3.4.13 Accessing the running services

The Postgres service can be accessed by creating a connection using the Postgres user and password provided in the .env file.

3.4.14 Additional Notes

3.4.15 References

[1] The PostgreSQL Global Development Group. (n.d.). PostgreSQL: About. PostgreSQL: The World's Most Advanced Open Source Relational Database. Retrieved August 22, 2021, from https://www.postgresql.org/about/

[2] The PostgreSQL Global Development Group. (2021, August 12). 1. What Is PostgreSQL? PostgreSQL Documentation. https://www.postgresql.org/docs/current/intro-whatis.html

[3] The PostGIS Development Group. (2021, August 20). PostGIS 3.1.4dev Manual. PostGIS - Documentation. https://postgis.net/docs/manual-3.1/

3.5 QGIS Server - PR

With the QGIS Server service you can publish one or more QGIS projects including: 1. Projects stored in-database in PostgreSQL 2. Projects stored in the file system

For the QGIS Server, we have chosen the OpenQuake build of QGIS Server because it has a few interesting characteristics. One, is that you can deploy QGIS server-side extensions easily with it and two, is that it supports things like the QGIS Authentication System. The QGIS Authentication System is an authentication database that provides more advanced security options, provides pg_service support, and provides some special features for URL rerouting so that your project paths are hidden away from the user (which is both a security and a convenience concern).

The OpenQuake QGIS Server is used for the QGIS Server instance. The OSGS also provides a couple of sample plugins like a demonstrater plugin and a plugin for handling atlas reports. The demonstrater plugin is a modified version of the GetFeatureInfo handler and will return some html back and in a nicely formatted table. The plugin for handling atlas reports, written by Lizmap extends the QGIS getPrint support to allow you to request a specific page from an atlas print. This is pretty handy if you, for example, click on a feature and you want to get then from an atlas report the one page for that feature in the atlas.

Another feature that Docker provides for applications such as QGIS Server is the ability to horizontally scale them. Our platform has some key configuration examples showing you how you can, for example, scale up the QGIS Server instance to have ten concurrently running instances. This is useful for handling increased or high load on the server. Scaling will create a round robin request handler, so that as the requests come in, it will pass each successive request over to the next running instance, and those requests will be handled by that instance, passed back and then that instance will stand by and wait for the next request to come in.

The QGIS Server works in orchestration with many of the other containers, including the PostGIS container. It also works pretty well in conjunction with the SCP (secure copy) container which allows the users of the OSGS architecture to easily move data from their local machine onto the server, either manually by copying and pasting files using an application such as Onescp or using built into Linux file browsers. For example, if you are one the GNOME desktop it has built into SFTP support.

Project Website: QGIS.org

Project Source Repository: QGIS on GitHub

Project Project Technical Documentation: QGIS on GitHub

Docker Repository: openquake/qgis-server:stable

Docker Source Repository: gem / oq-qgis-server

3.5.1 Configuration

3.5.2 Deployment

3.5.3 Enabling

3.5.4 Disabling

3.5.5 Accessing the running services

Every project you publish will be available at `/ogc/project_name` which makes it very simple to discover where the projects are deployed on the server.

3.5.6 Additional Notes

3.5.7 Further Reading

You should read the QGIS Server documentation on QGIS.org. It is well written and covers a lot of background explanation which is not provided here. Also you should familiarise yourself with the Environment Variables.

Alesandro Passoti has made a number of great resources available for QGIS Server. See his workshop slide deck and his server side plugin examples, and more examples here.

3.5.8 QGIS Server Atlas Print Plugin

See the project documentation for supported request parameters for QGIS Atlas prints.

3.5.9 QGIS Server Scaling

If your server has the needed resources, you can dramatically improve response times for concurrent QGIS server requests by scaling the QGIS server:

```
docker-compose --profile=qgis-server up -d --scale qgis-server=10 --remove-orphans
```

To take advantage of this, the locations/upstreams/qgis-server.conf should have one server reference per instance e.g.

```
    upstream qgis-fcgi {
        # When not using 'host' network these must reflect the number
        # of containers spawned by docker-compose and must also have
        # names generated by it (including the name of the stack)
        server osgisstack_qgis-server_1:9993;
        server osgisstack_qgis-server_2:9993;
        server osgisstack_qgis-server_3:9993;
        server osgisstack_qgis-server_4:9993;
        server osgisstack_qgis-server_5:9993;
        server osgisstack_qgis-server_6:9993;
        server osgisstack_qgis-server_7:9993;
        server osgisstack_qgis-server_8:9993;
        server osgisstack_qgis-server_9:9993;
        server osgisstack_qgis-server_10:9993;
    }
```

Then restart Nginx too:

```
docker-compose --profile=production restart nginx
```

Note that if you do an Nginx up it may bring down your scaled QGIS containers so take care.

Finally check the logs of Nginx to make sure things are running right:

```
docker-compose --profile=production logs nginx
```

3.6 GeoServer - PR

GeoServer is a Java-based server, built on GeoTools, that allows users to view and edit geospatial data, using the open standards set forth by the Open Geospatial Consortium (OGC). By implementing the Web Map Service (WMS) standard, GeoServer can create maps in a variety of output formats. GeoServer also conforms to the Web Feature Service (WFS) standard, and Web Coverage Service (WCS) standard which permits the sharing and editing of the data that is used to generate the maps. GeoServer also uses the Web Map Tile Service standard to split your published maps into tiles for ease of use by web mapping and mobile applications. OpenLayers, a free mapping library, is integrated into GeoServer, making map generation quick and easy. GeoServer is a modular application with additional functionality added via extensions.*[1]*

Service name: geoserver

Project Website: GeoServer

Project Source Repository: geoserver / geoserver

Project Technical Documentation: GeoServer Documentation

Docker Repository: kartoza/geoserver

Docker Source Repository: kartoza / docker-geoserver

3.6.1 Deployment

```
make deploy-geoserver
```

3.6.2 Enabling

```
make enable-geoserver
```

3.6.3 Configuration

```
make configure-geoserver-passwd
```

3.6.4 Starting

```
make start-geoserver
```

3.6.5 Stopping

```
make stop-geoserver
```

3.6.6 Disabling

```
make disable-geoserver
```

3.6.7 Logs

```
make geoserver-logs
```

3.6.8 Shell

```
make geoserver-shell
```

3.6.9 Accessing the running services

The services can be accessed on /geoserver/ e.g. https://localhost/geoserver.

Look in the .env file for the administrator password.

3.6.10 Additional Notes

After configuring, you should remember to set the master password as per https://docs.geoserver.geo-solutions.it/edu/en/security/security_overview.html#the-master-password (which is different to the admin password).

3.6.11 References

[1] Open Source Geospatial Foundation. (n.d.). About - GeoServer. GeoServer. Retrieved August 21, 2021, from http://geoserver.org/about/

3.7 Docker OSM mirror - PR

OpenStreetMap (OSM) is a digital map database of the world built through crowdsourced volunteered geographic information. The data from OSM is freely available for visualization, query, download, and modification under open licenses. *[1]* OSM can also be described as a free, editable map of the whole world *[2]*.

The Docker OSM mirror service is a docker compose project to setup an OSM PostGIS database with automatic updates from OSM periodically. The only files you need is a PBF file, geojson (if you intend to restrict data download to a smaller extent than the one specified by the PBF) and run the docker compose project.*[3]*

The Docker OSM mirror service is composed of Docker ImpOSM3, Docker OSM Update and Docker OSM enrich. Docker ImpOSM3 takes the PBF file and imports it into the PostGIS OSM database. It will also apply any new diff file that arrives to the database. Docker OSM update runs every few minutes and regualarly fetches any new diff file for all the changes that have happened over the update interval from OpenStreetMap and applies any new features to your existing PostGIS OSM database. OSM enrich goes to the OSM API and gets the username and last change timestamp for each feature.*[3]*

Service name: osm-mirror

Project Website: OpenStreetMap

Project Source Repository: openstreetmap / openstreetmap-website

Project Technical Documentation: OpenStreetMap Getting Help

Docker Repository: kartoza/docker-osm

Docker Source Repository: kartoza / docker-osm

3.7.1 Deployment

```
make deploy-osm-mirror
```

3.7.2 Enabling

```
make enable-osm-mirror
```

3.7.3 Configuration

```
make configure-osm-mirror
```

3.7.4 Starting

```
make start-osm-mirror
```

3.7.5 Stopping

```
make stop-osm-mirror
```

3.7.6 Disabling

```
make disable-osm-mirror
```

3.7.7 Reinitialising

```
make reinitialise-osm-mirror
```

3.7.8 Creating a vector tiles store from the docker osm schema

```
make osm-to-mbtiles
```

3.7.9 Logs

```
make osm-mirror-logs
```

3.7.10 Shell

```
make osm-mirror-osmupdate-shell

make osm-mirror-imposm-shell
```

3.7.11 Accessing the running services

3.7.12 Additional Notes

The OSM mirror uses the kartoza/docker-osm tool to create an in-database mirror of a designated geographical area in the designated postgres databse schema (set to: osm). The OSM mirror tool is described in the project README here:

https://github.com/kartoza/docker-osm

To deploy the docker-mirror you need to follow the steps descibed below. First a process overview:

1. Create the PBF feature container passing it a URL to a PBF file
2. Create a clip file that will be used to constrain any retrieved / imported data to a specific geographic area.
3. Tweak the mappings.yml file (advanced users)
4. Run the docker-osm service
5. Optionally include these data in published services via QGIS projects, GeoServer etc.

PBF Container

During the `make configure` process, the script will ask for the URL to an OSM .pbf file e.g.:

```
------------------------------------------------------------------------
Fetching pbf if not cached and then copying to settings dir
------------------------------------------------------------------------
URL For Country PBF File: https://download.geofabrik.de/central-america-latest.osm.pbf
```

You can enter any valid URL for an OSM .PBF file at this point. A docker container will be built that fetches the PBD and stores it on the host file system under osm_config.

Clip Area

Create the clip area to constrain the geographical region that data will be harvested for. For best performance, a simple rectangle is best, but any complext polygon can be used. The clip area must be saved as `osm_config/clip.geojson`. The format for the clip area must be GeoJSON. You can easily create this using QGIS.

Mappings

For advanced users, you can tweak the `osm_config/mapping.yml`

You can see how the imposm3 mapping syntax works here:

https://imposm.org/docs/imposm3/latest/mapping.html

Note that you cannot alter the mappings after the service is running without clearing the databse and restarting the import.

Run services

To run the OSM services do:

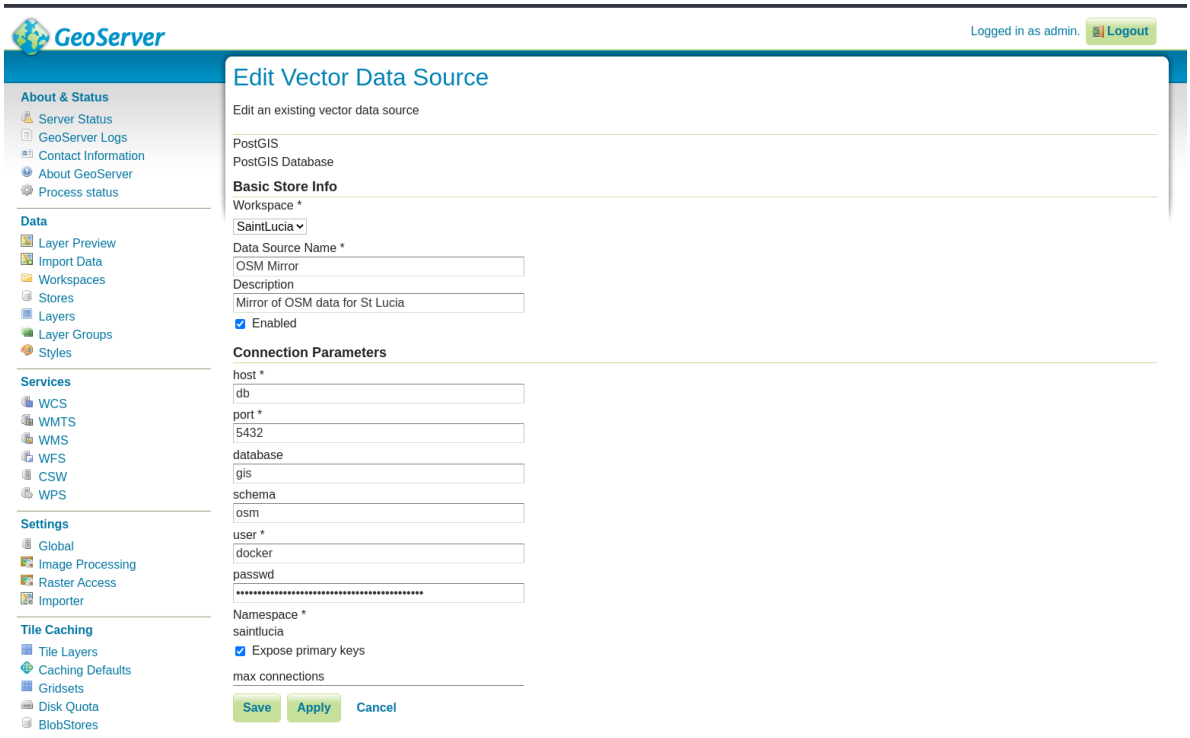
```
docker-compose --profile=osm up -d
```

Publishing with GeoServer

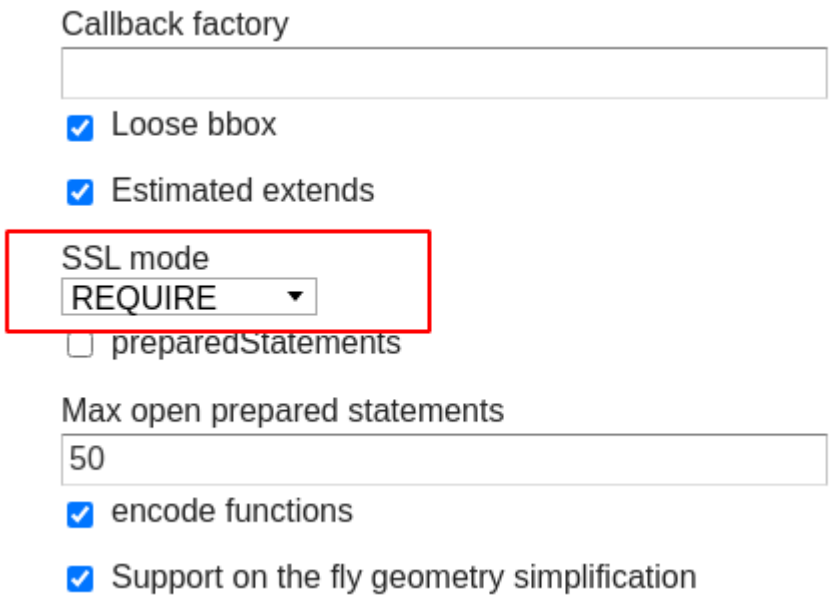
You can publish the data in the osm schema using GeoServer or by publishing a QGIS project that references the data layers in the OSM schema.

The steps for publishing with GeoServer are quite simple:

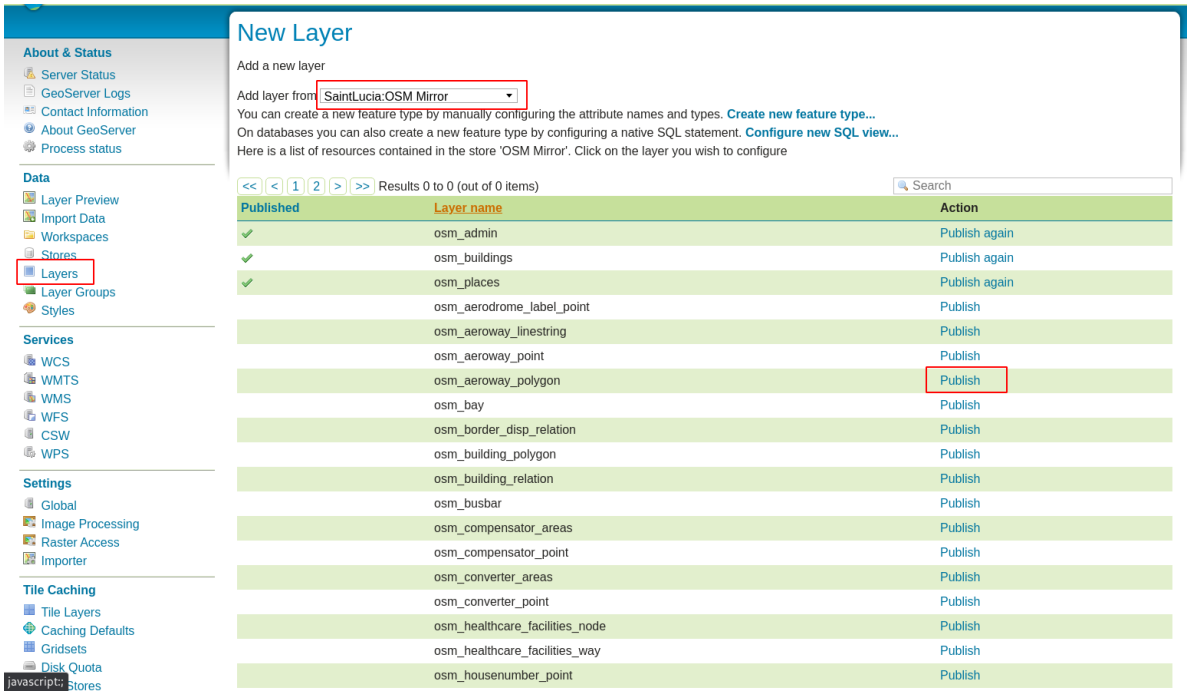
1. Log in to GeoServer using the 'admin' user and the password in .env.
2. Create a new store of type 'Postgis' and configure it as per the screenshot below, replacing the password with the Postgres password stored in .env:



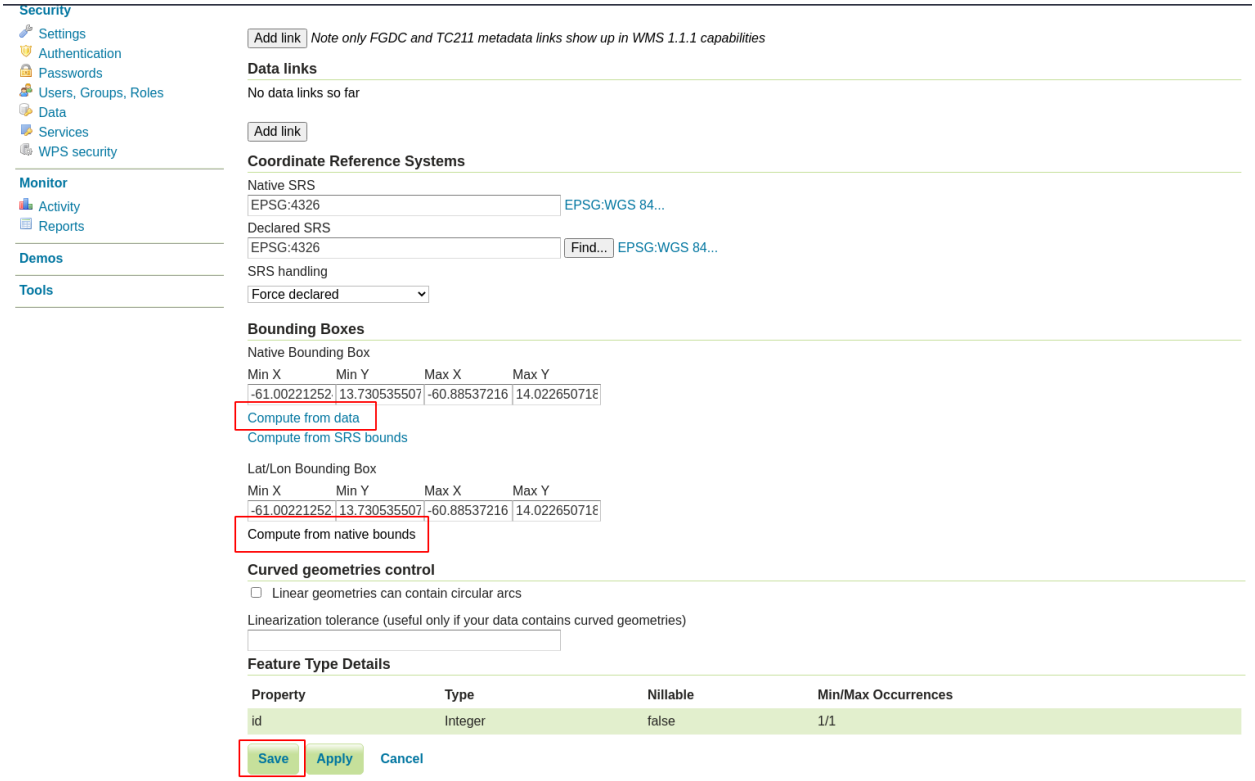
Also, be sure to scroll down and set SSL mode to Required:



3. Register one or more layers from that store as per the image below:



4.Complete the layer details as appropriate and make sure to click the options highlighted in red in the screenshot below:

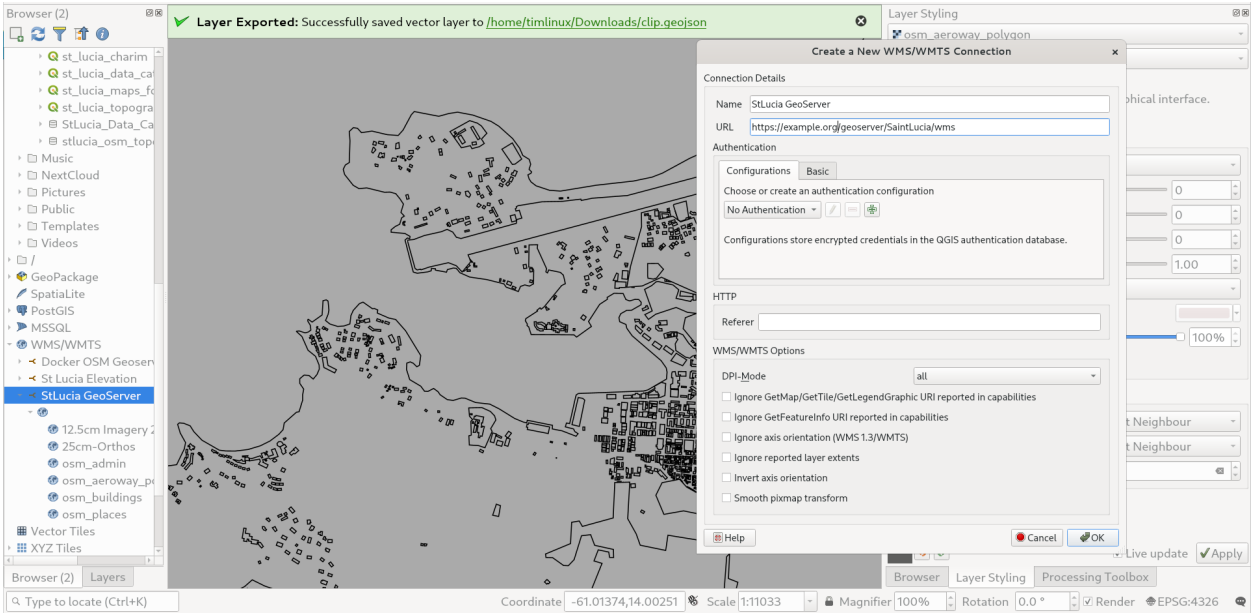


1. Connect to the GeoServer from a client e.g. QGIS using WFS or WMS using the scheme:

https://example.org/geoserver/SaintLucia/wfs

or

https://example.org/geoserver/SaintLucia/wms



Publishing with QGIS Server

The workflows described in the section on working with the PostgreSQL database below are basically all you need to know, so we don't repeat that here, other than to remind you that hte OSM mirrored data is by default stored in a schema called 'osm'.

OSM Attribution

Note that whenever you publish a map containing OSM data, be careful to adhere to the license and credit the OSM Project as per:

https://www.openstreetmap.org/copyright

3.7.13 References

[1] Quinn, S., & Dutton, J. A. (n.d.). OpenStreetMap and its use as open data | GEOG 585: Web Mapping. GEOG 585 Open Web Mapping. Retrieved August 30, 2021, from https://www.e-education.psu.edu/geog585/node/738

[2] About OpenStreetMap - OpenStreetMap Wiki. (n.d.). OpenStreetMap Wiki. Retrieved August 30, 2021, from https://wiki.openstreetmap.org/wiki/About_OpenStreetMap

[3] Kartoza. (n.d.). GitHub - kartoza/docker-osm: A docker compose project to setup an OSM PostGIS database with automatic updates from OSM periodically. GitHub. Retrieved August 30, 2021, from https://github.com/kartoza/docker-osm#readme

3.8 Node Red - PR

3.9 Jupyter Notebook - PR

3.10 PostgREST - PR

PostgREST is a standalone web server that turns your PostgreSQL database directly into a RESTful API. The structural constraints and permissions in the database determine the API endpoints and operations. *[1]* The PostgREST service ion the OSGS platform is used for pushing readings from IoT devices to our database.

Service name: postgrest

Project Website: PostgREST Documentation

Project Source Repository: PostgREST / postgrest

Project Technical Documentation: PostgREST Documentation

Docker Repository: postgrest/postgrest

Docker Source Repository: PostgREST / postgrest

3.10.1 Configuration

3.10.2 Deployment

3.10.3 Enabling

3.10.4 Disabling

3.10.5 Accessing the running services

3.10.6 Additional Notes

3.10.7 References

[1] Nelson, J., & Chavez, S. (n.d.). PostgREST Documentation — PostgREST 8.0.0 documentation. PostgREST Documentation. Retrieved August 26, 2021, from https://postgrest.org/en/v8.0/

3.11 Swagger - PR

3.12 SCP - PR

The SCP (secure copy) containers have been arranged so that there are some standard containers out of the box. Each container has its data stored in its own docker volume as well. The data is somewhat isolated and there are containers for QGIS projects, fonts, SVGs that your QGIS projects might reference, general file sharing, uploading data to ODM, etc. The SCP service is designed to only support connections with SSH public-private key encryption and password based authentication. The way that you provision users into it is that you copy the SSH public key into a file in the configuration folder for SCP and then that user will be allowed to make the connection to whichever SCP share that you have created for them. The SCP container can be used to copy a QGIS project file from your desktop up to the server

with all the QGIS resources that it needs such as shapefiles. The QGIS Server instance can then be used to access the project from the OGC web services.

Project Website:

Project Source Repository:

Project Project Technical Documentation:

Docker Repository:

Docker Source Repository:

3.12.1 Configuration

3.12.2 Deployment

3.12.3 Enabling

3.12.4 Disabling

3.12.5 Accessing the running services

3.12.6 Additional Notes

3.12.7 SCP File Drop Shares

This is a container intended for users to upload files for publication on the server. It runs on port 2222 so we need to expose that through the firewall:

```
sudo ufw allow 2222
```

You can add your public keys from the host e.g.

```
cat ~/.ssh/authorized_keys > scp_conf/gis_projects
```

Or copy them in by other means. Each file you create in scp_conf will be a user name when the scp container runs, with it's own directory in the storage volume, unless an explicit storage volume has been pre-defined (see list of these below). Each file should contain a list of public keys. If you add a key at some point, or a new user file, you may need to restart the container:

```
docker-compose profile=scp restart
```

The following scp shares are made for the various purposes listed below. You need to follow the same pattern of creating a config file for each. These shares each have a dedicated volume associated with it which is also mounted into the associated server container.

- **User:** geoserver_data
- **Named Volume:** scp_geoserver_data
- **Volume Mounted To:** scp, geoserver
- **Notes:** Copy vector and raster datasets here for publishing in GeoServer.
- **Example Use:** `sftp://geoserver_data@<hostname>:2222/home/geoserver_data`

- **User:** qgis_projects
- **Named Volume:** scp_qgis_projects
- **Volume Mounted To:** scp, qgis-server
- **Notes:** Copy QGIS projects and data here for publishing with QGIS Server. See notes on directory layout below.
- **Example Use:** `sftp://qgis_projects@<hostname>:2222/home/qgis_projects`

- **User:** qgis_svgs
- **Named Volume:** scp_qgis_svgs
- **Volume Mounted To:** scp, qgis-server
- **Notes:** Embed SVGs in styles by preference in QGIS. Use this drop if you have no way to use embeded SVGS.
- **Example Use:** ``sftp://qgis_svgs@:2222/home/qgis_svgs`

- **User:** qgis_fonts
- **Named Volume:** scp_qgis_fonts
- **Volume Mounted To:** scp, qgis-server
- **Notes:** Copy fonts directly into the root folder.
- **Example Use:** `sftp://qgis_fonts@<hostname>:2222/home/qgis_fonts`

- **User:** hugo_data
- **Named Volume:** scp_hugo_data
- **Volume Mounted To:** scp, hugo*
- **Notes:** Upload markdown files for static site generation with Hugo.
- **Example Use:** `sftp://hugo_data@<hostname>:2222/home/hugo_data`

- **User:** odm_data
- **Named Volume:** scp_odm_data
- **Volume Mounted To:** scp, odm *
- **Notes:** Upload imagery data for processing with ODM
- **Example Use:** `sftp://odm_data@<hostname>:2222/home/odm_data`

- **User:** general_data
- **Named Volume:** scp_general_data
- **Volume Mounted To:** scp
- **Notes:** General sharing directory. Later we will publish this under nginx for public downloads. Don't put any sensitive data in here.

- **Example Use:** `sftp://general_data@<hostname>:2222/home/general_data`

Note: Any user connecting to any of these shares will be able to see all other files from all other users. They will only have write access to the folder they are connecting to, for all other shares their access will be read only. If you want to further partition the access to files you can create multiple scp services, each with one of the mount points listed above. In so doing users would not be able to see the other mount points listed above.

3.12.8 Directory layout for the QGIS projects folder

When adding projects to the qgis_projects folder, you need to follow this convention strictly for the projects to be recognised by QGIS Server:

`qgis_projects/<project_name>/<project_name>.qgs`

For example:

`qgis_projects/terrain/terrain.qgs`

There is a convenience Make target that will copy your .ssh/authorized_keys file contents into each of the scp_config user files listed in the table above.

```
make setup-scp
```

3.12.9 Starting the container

```
docker-compose --profile=scp up -d scp
```

Example copying of data into the container from the command line:

```
scp -P 2222 sample-document.txt localhost:/data//gis_projects/gis_projects/gis_projects
```

In Nautilus (file manager in Linux Gnome Desktop) you can test by connecting

`sftp://<hostname>:2222/data/gis_projects`

into the red highlighted box below:

XXXXXXXXXXXXXXXXXXXX

After that open a second window and you can drag and drop files too and from the folder. Windows users can use the free WinSCP application to copy files to the server.

3.12.10 FAQ

Q: When connecting I get "Host key validation failure" or similar **A:** Remove the entry for the server in your ~/.ssh/known_hosts

3.13 Lizmap - WIP

Lizmap is an open source software designed by 3Liz that allows QGIS® Desktop to create Web map applications. *[1]*

Project Website: Lizmap

Project Source Repository: 3liz / lizmap-web-client

Project Technical Documentation: Lizmap - Documentation

Docker Repository: 3liz/lizmap-web-client

Docker Source Repository: 3liz / docker-lizmap-web-client

3.13.1 Deployment

```
make deploy-lizmap
```

3.13.2 Enabling

```
make enable-lizmap
```

3.13.3 Configuration

```
make configure-lizmap
```

3.13.4 Starting

```
make start-lizmap
```

3.13.5 Stopping

```
make stop-lizmap
```

3.13.6 Disabling

```
make disable-lizmap
```

3.13.7 Logs

```
make lizmap-logs
```

3.13.8 Shell

```
make lizmap-shell
```

3.13.9 Accessing the running services

3.13.10 Additional Notes

3.13.11 References

[1] 3Liz. (n.d.). Lizmap. Retrieved August 26, 2021, from https://www.3liz.com/en/lizmap.html

3.14 Mergin Server - WIP

Mergin is a web platform for storage and synchronisation of geospatial projects across multiple users and devices (desktop and mobile).The platform is especially useful when you need:

- **Mobile data collection**. If you need to capture location of assets (and their attributes) or update an existing database.
- **Data sharing**. No complicated setup of access by IT admins to get your data to colleagues or clients. Set up permissions and send invites with few clicks.
- **Offline access**. Work with data with no interruption even without constant Internet connection - sync any changes when you are back online.
- **Collaborative editing**. No more problems dealing with multiple copies of the same dataset in different versions - all changes are automatically consolidated in one place.
- **Audit changes**. Knowing who has changed what and when in a database is often important - Mergin keeps track of the history and allows to go back if needed.
- **No coding required**. Everything can be set up with no knowledge of programming. *[1]*

Project Website: Mergin

Project Source Repository: lutraconsulting / mergin

Project Technical Documentation: Mergin Help

Docker Repository: lutraconsulting/mergin

Docker Source Repository: lutraconsulting / mergin

3.14.1 Deployment

```
make deploy-mergin-server
```

3.14.2 Enabling

```
make enable-mergin-server
```

3.14.3 Configuration

```
make configure-mergin-server
```

3.14.4 Starting

```
make start-mergin-server
```

3.14.5 Stopping

```
make stop-mergin-server
```

3.14.6 Disabling

```
make disable-mergin-server
```

3.14.7 Logs

```
make mergin-server-logs
```

3.14.8 Shell

```
make mergin-server-shell
```

3.14.9 Restoring data

```
make restore-mergin-server-sql
```

3.14.10 Accessing the running services

3.14.11 Additional Notes

3.14.12 References

[1] Lutra Consulting. (n.d.). GitHub - lutraconsulting/mergin: Store and track changes to your geo-data. GitHub. Retrieved August 26, 2021, from https://github.com/lutraconsulting/mergin

3.15 Mergin Client - WIP

This tool will synchronise a Mergin cloud project into a PostgreSQL project.

There are two modalities in which you can work with Mergin projects:

1. ***mergin-db-sync***: A Mergin project which is synchronised into a PostgreSQL database and supports bidirectional syncing and editing.
2. ***mergin-client*** **(covered here):** A folder containing multiple mergin projects (all of the projects shared with a mergin user). These projects are synchronised into the filesystem and published via QGIS Server as web mapping services.

Service name: mergin-client

Project Website: Mergin

Project Source Repository: lutraconsulting / mergin-py-client

Project Technical Documentation: Mergin Python Client

Docker Repository: kartoza/mergin-client

Docker Source Repository: kartoza /mergin-client

3.15.1 Configuration

```
make configure-mergin-client
```

3.15.2 Reinitialising

```
make reinitialise-mergin-client
```

3.15.3 Redeploying

```
make redeploy-mergin-client
```

3.15.4 Accessing the running services

3.15.5 Additional Notes

For field data collection and project synchronisation support see the following:

1. The Mergin platform for cloud storage of projects
2. The INPUT mobile data collection platform

You use the Mergin client to clone one or more Mergin projects to the host running docker-compose and these projects are made available through QGIS Server.

One critical note is that the Project directory and the Project File names must be the same, otherwise QGIS Server will not recognize the project as being valid. For example:

- Valid: `FooProject/FooProject.qgz`
- Not Valid: `FooProject/BarProject.qgz`

Once published in this way, valid projects will be accessible from any OGC compliant client (e.g. QGIS Desktop, OpenLayers, Leaflet) using the following URL Scheme:

`https://yourhost.com/ogc/yourproject`

For example, here is one we published for a client (domain name changed):

`https://example.org/ogc/Elevation`

You can read more about the mergin-client at the separate git repo here.

3.16 Mergin-db-sync - PR

There are two modalities in which you can work with Mergin projects:

1. ***mergin-db-sync*** **(covered here)**: A Mergin project which is synchronised into a PostgreSQL database and supports bidirectional syncing and editing.
2. ***mergin-client*****:** A folder containing multiple mergin projects (all of the projects shared with a mergin user). These projects are synchronised into the filesystem and published via QGIS Server as web mapping services.

Service name: mergin-sync

Project Website: Mergin

Project Source Repository: lutraconsulting / mergin-db-sync

Project Technical Documentation: DB Sync Script

Docker Repository: lutraconsulting/mergin-db-sync

Docker Source Repository: lutraconsulting / mergin-db-sync

3.16.1 Configuration

3.16.2 Deployment

3.16.3 Enabling

3.16.4 Disabling

3.16.5 Accessing the running services

3.16.6 Additional Notes

In the .env file you should specify these options:

- **MERGIN_USER:** This is the user account that will be used to log in and pull/push updates to the Mergin project.
- **MERGIN_PASSWORD:** This is the user password for the above account.
- **MERGIN_PROJECT_NAME:** Specified in the form of `user/project` this is the Mergin project that will be synchronised into the database.
- **MERGIN_SYNC_FILE:** This is the name of a GeoPackage `yourgeopackage.gpkg` in the Mergin project whose schema will be replicated into a PostGIS schema as described below.
- **DB_SCHEMA_MODIFIED:** This is a PostgreSQL schema (schemas can be thought of as 'folders' within your database within which tables are found) that will contain the synchronised data form mergin. The content of the tables are editable via INSERT/UPDATE/DELETE operations, bit the structure of these tables (via ALTER commands) should not be attempted. Note that the replication is bidirectional, so changes made in the database are synchronised to all mergin clients and changes made in the distributed clients will make their way back into your database.
- **DB_SCHEMA_BASE:** This is a 'hands off" copy of the content in the MERGIN_SYNC_FILE that is stored in PostgreSQL to act as a reference when mergin calculates the changeset between the MODIFIED schema content and the remote copies of the data. DO NOT USE THIS and definately DO NOT EDIT THIS..
- **MERGIN_URL:** This is the public server where your mergin project is hosted. By default it would be "https://public.cloudmergin.com" unless you are self hosting the Mergin backend, or using an alternative hosted instance.

Note that in the docker-compose file, the assumption is made that the database being used for Mergin syncing is called 'gis' and the hostname (in the private docker network) is called 'db'. The username and password are taken from the following keys in the .env file:

- POSTGRES_USER
- POSTGRES_PASSWORD

CHAPTER

FOUR

UTILITIES

4.1 make ps

4.1.1 Synopsis

Displays the actively running services in the OSGS platform.

4.1.2 Usage

Make Target: ps

Arguments: none

Example usage: `make ps`

Example output:

```
-----------------------------------------------------------------
Current status
-----------------------------------------------------------------
          Name                        Command                  State                 ␣
↪                  Ports
-------------------------------------------------------------------------------------------------
↪-----------------------------------------------------------
osgisstack_db_1             /bin/sh -c /scripts/docker ...   Up (healthy)   0.0.0.0:5432-
↪>5432/tcp,:::5432->5432/tcp
osgisstack_hugo-watcher_1   python3 /hugo_watcher.py         Up             1313/tcp
osgisstack_nginx_1          /docker-entrypoint.sh /bin ...   Up             0.0.0.0:443->
↪443/tcp,:::443->443/tcp, 0.0.0.0:80->80/tcp,:::80->80/tcp
osgisstack_scp_1            /bin/sh -c /run.sh               Up             0.0.0.0:2222-
↪>22/tcp,:::2222->22/tcp
```

4.1.3 Notes

Make ps only shows active containers.

4.2 make kill

4.2.1 Synopsis

Kills all actively running services in the OSGS platform.

4.2.2 Usage

Make Target: kill

Arguments: none

Example usage: `make kill`

Example output:

```
----------------------------------------------------------------------
Killing all containers
----------------------------------------------------------------------
Killing osgisstack_db_1           ... done
Killing osgisstack_nginx_1        ... done
Killing osgisstack_hugo-watcher_1 ... done
Killing osgisstack_scp_1          ... done
          0.0.0.0:2222->22/tcp,:::2222->22/tcp
```

4.2.3 Notes

Make kill only shows active containers from the OSGS platform, other containers should continue unaffected.

4.3 make nuke

4.3.1 Synopsis

Kills all actively running services in the OSGS platform, deletes all configurations, deletes all docker volumes, removes letsencrypt certificate. Generally has the effect of resetting your system to the state it was at when you performed the initial git checkout.

4.3.2 Usage

Make Target: nuke

Arguments: none

Example usage: `make nuke`

Example output:

```
------------------------------------------------------------------
Disabling services
This command will delete all your configuration and data permanently.
Are you sure? [y/N] y
Please type CONFIRM to proceed CONFIRM
------------------------------------------------------------------
Nuking Everything!
------------------------------------------------------------------
Going to remove osgisstack_db_1, osgisstack_nginx_1, osgisstack_hugo-watcher_1,␣
↪osgisstack_scp_1
Removing osgisstack_db_1           ... done
Removing osgisstack_nginx_1        ... done
Removing osgisstack_hugo-watcher_1 ... done
Removing osgisstack_scp_1          ... done

[sudo] password for timlinux:
make[1]: Entering directory '/home/timlinux/dev/docker/OpenSource-GIS-Stack'

------------------------------------------------------------------
Reset site configuration to default values
This will replace any local configuration changes you have made
------------------------------------------------------------------
Are you sure you want to continue? [y/N] y
make[1]: Leaving directory '/home/timlinux/dev/docker/OpenSource-GIS-Stack'
make[1]: Entering directory '/home/timlinux/dev/docker/OpenSource-GIS-Stack'

------------------------------------------------------------------
Disabling services
This will remove any symlinks in conf/nginx_conf/locations and conf/nginx_conf/upstreams
effectively disabling all services exposed by nginx
------------------------------------------------------------------
Are you sure? [y/N] y
make[1]: Leaving directory '/home/timlinux/dev/docker/OpenSource-GIS-Stack'
```

4.3.3 Notes

Again we warn you here that make nuke is extremely destructive and should only be used if you want to completely reset your installation!

4.4 make docs

4.4.1 Synopsis

Builds the system documentation.

4.4.2 Usage

Make Target: docs

Arguments: none

Example usage: `make docs`

Example output:

```
```

See also: *building docs*

4.4.3 Notes

None

4.5 Building Documentation

4.5.1 HTML Docs

```
sudo dnf install sphinx
pip3 install --upgrade myst-parser
pip install sphinx-sizzle-theme
```

See https://sphinx-themes.org/sample-sites/sphinx-sizzle-theme/ for theme specific info.

See https://www.sphinx-doc.org/en/master/usage/markdown.html for Markdown in Sphinx support notes (and the docs at https://myst-parser.readthedocs.io/en/latest/syntax/optional.html). Especially, note the admonitions docs which are used to make little alert etc boxes in the docs: https://myst-parser.readthedocs.io/en/latest/syntax/optional.html#html-admonitions

4.5.2 PDF Docs

On fedora I just install the huge, full texlive package:

```
sudo dnf install texlive-scheme-full
```

Then build:

```
make latexpdf
```

Sometimes you need to run it a second time if it is a fresh build.

After the build is done, the PDF will be at:

```
docs/build/latex/osgs.pdf
```

CHAPTER

FIVE

WORKFLOWS

5.1 Publishing changes to the static website

This workflow shows how to publish changes to the static website using the SCP containers and the Hugo content SCP folder labelled hugo_data, by editing a document locally then pushing the changes up.

5.2 Uploading a QGIS project

The steps to be demonstrated in this workflow are:

- Creating a shapefile
- Creating a QGIS project file
- Putting the QGIS project file in a named directory that matches the name of QGIS project file
- Uploading the folder containing the QGIS project to the SCP folder for QGIS projects
- Accessing the QGIS project on the OGC link

5.3 Accessing PostGIS from QGIS using a pg_service file

This workflow will demonstrate the following:

- Setting up your Postgres connection
- Loading data into the Postgres database
- Creating the connection
- Creating the pg_service file
- Using the pg_service file on the client side of the stack (where it references the external port) and the server side (where it references the internal port)

5.4 Authentication of Postgres using a pg_service file and a qgis-auth.db file

This workflow will demonstrate how to access Postgres in QGIS using a pg_service file and a qgis-auth.db file.

5.5 Publishing a QGIS project

This workflow will demonstrate how to publish a QGIS project where the layers are inside of Postgres. This involves uploading a QGIS project file through the pg_service file and or qgis-auth.db for authentication then publishing the map.

5.6 Publishing layers using Geoserver

This workflow will demonstrate the following:

- Basic configuration of Geoserver using a shapefile store (a directory of shapefiles)
- Uploading one or more shapefiles to the server using the scp drop zone for Geoserver
- Registering the layers
- Publishing the layers as basic WMS layers

5.7 Connecting to Geoserver and QGIS Server in Hugo

This workflow covers connecting to QGIS server and Geoserver in Hugo through the following steps:

- Creating a new map file pointing to the service URL
- Specifying the layers you want
- Adding any surrounding text and information for the service

5.8 QGIS Desktop as a Web Services Client

This workflow will cover how to use QGIS Desktop as a web service client for both the QGIS Server and Geoserver published layers.

5.9 Node-red workflow that accesses data in Postgres

This is a simple workflow where you connect to a table in Postgres, select all the rows in the table and put them into a table in a dashboard in node red.

5.10 PostgREST API

This workflow will demonstrate how to:

- Publish layers from a Postgres schema into the PostgREST API
- Use swagger in the web browser to go and discover the published services there

5.11 Node-red connect to the PostgREST API

This workflow will demonstrate how to connect to the PostgREST API, pull data out of table and present it in a node-red dahsboard.

5.12 Node-red point layer to world map

This workflow demonstrates how to take a point layer and connect to it using either the PostgREST API or via Postgres connection, get the points and some data for each point and push them through to the world map in node-red, to create a web map based on live data in the database.

5.13 Set up process for pg notify

Pg notify is a notification system that Postgres has that can let a client know that a change has occurred in the database. This workflow will demonstrate how to:

- Register the notification function
- Push out the message to a client (we will start with QGIS as the client)
- Have QGIS automatically update and refresh the canvas when a notification happens

5.14 Mergin-db-sync client workflow 1

This workflow will demonstrate how to:

- Create a project in QGIS
- Publish it to Mergin
- Synchronize the published project into your Postgres database schema

5.15 Mergin-db-sync client Workflow 2

This workflow will demonstrate how to:

- Create a QGIS project with data in Postgres
- Tell Mergin to generate a geopackage and push the geopackage out to the Mergin server

5.16 Creating an Open Street Map mirror into your database

5.16.1 Preparing the Country PBF file and the clip area document

The PBF files for the country or region of interest can be downloaded from GeoFabrik. The PBF file used in this workflow was for Kenya and the URL for the country PBF file is https://download.geofabrik.de/africa/kenya-latest.osm.pbf.

The clip area allows you to limit the features being imported into the PostGIS OSM database to a small area you can work with. You will need to save the clip area document as `conf/osm_conf/clip.geojson`. The `clip.geojson` can be the same extent of the administrative area for your country or region specified in the PBF file, or it can be a smaller extent. The CRS of the geojson should always be EPSG:4326.*[1]*

You can easily create such a clip document at https://geojson.io or by using QGIS. For this workflow the clip area document for the country Kenya, was obtained using QGIS. The Kenya country boundary data was obtained from the Kenya- Subnational Administrative Boundaries data.

5.16.2 Deploying the OSM mirror service.

To deploy the initial stack, which includes the Nginx, Hugo watcher and SCP services, please run either `make configure-ssl-self-signed` or `make configure-letsencrypt-ssl`.

Next deploy the Postgres service using `make deploy-postgres`. If you have PostgreSQL already installed outside of the stack (on your local machine) ensure that you specify a different Postgres public port number other than the default 5432. For example, you can use the port number 5434 for the public port.

To deploy the OSM mirror service, run the `make deploy-osm-mirror` command and follow the subsequent instructions. You can view the logs for the OSM mirror service using the command `make osm-mirror-logs`.

5.16.3 Loading the OSM Mirror Layers into QGIS

To use the layers in the OSM Postgis database in QGIS, use the Postgres public port number, username and password contained in the `.env` file to create a connection to the PostGIS OSM database `gis` in QGIS. Make sure to the set the SSL mode to require.

The imported Open Street Map layers for the clip area specified are present in the `osm` schema of the database.

To load a layer from the `osm` schema onto the QGIS Map View, double click on the table or drag and drop the table on the Map View.

5.16.4 Saving a QGIS project into the OSM database

In the **Menu Toolbar** click on **Project**. From the drop down menu select **Save To PostgreSQL**.

Save the project in the `public` schema and name it `qgis projects` to allow the `make backup-db-qgis-project` command to be able to create a back up of the project.

5.16.5 Backing up and restoring a QGIS project into the database

To back up the QGIS project created in the previous section, run the command `make backup-db-qgis-project`. This backs up the `qgis_projects` table in the `public` schema as a `.sql` file named `QGISProject.sql`.

To restore a backed up QGIS project, name the `.sql` file `QGISProject.sql` and place the file in the `backups` folder then run the command `restore-db-qgis-project`.

5.16.6 Saving QGIS layer styles into the database

To save the style of a layer into the database, right click on the layer in the **Layers Panel** and select **Properties**.

In the Symbology section of the Layer Properties, click on **Style** > **Save style**.

In the Save Layer Style dialogue select **In Database (postgres)** and name the style file. You can add an optional description of the style and also set the style to be the default style for the layer.

The saved style is added as an entry in the `layer_styles` table in the `public` schema of the PostGIS OSM database.

5.16.7 Backing up and restoring the QGIS styles into the database

To back up the QGIS styles created in the previous section, run the command `make backup-db-qgis-styles`. This backs up the `layer_styles` table in the `public` schema as a `.sql` file named `QGISStyles.sql`.

To restore a back up of the QGIS styles, name the back up file `QGISStyles.sql` and place it in the `backups` folder then run the command `make restore-db-qgis-styles`.

5.16.8 References

[1] Kartoza. (n.d.). GitHub - kartoza/docker-osm: A docker compose project to setup an OSM PostGIS database with automatic updates from OSM periodically. GitHub. Retrieved August 30, 2021, from https://github.com/kartoza/docker-osm#readme

5.17 OSM enrich workflow

This workflow will demonstrate how to use osm-enrich to add a timestamp and the user name for every feature being pulled from Open Street Map.

5.18 PostgreSQL Workflows

There are a number of different workflows we will explore here:

- Connecting to the PostgreSQL database from your QGIS Desktop application
- Connecting to the PostgreSQL database from QGIS Server
- Connecting to the PostgreSQL database from GeoServer
- Connecting from other applications

Before we dive into the details here, let us quickly examine some basic use cases:

- You collect data in the field using Input and use the mergin-db-sync workflow to synchronised field collected data into the database, then publish maps for this data in QGIS Server.

- You want to publish a layer in GeoServer, so you connect to the database from your desktop, drag and drop a local file system layer into the database using QGIS, then publish the layer from GeoServer.
- You want to create a project in QGIS desktop that uses PostgreSQL for data storage, then publish that data as a QGIS Project.

There are many other workflows that the Open GIS Stack supports, it is really up to your imagination! What is key to understand are the mechanisms that you can use to connect to the database in different contexts. And so we will focus this section on the different connection modalities.

5.18.1 Direct connection

Direct connection to the server over the standard PostgreSQL port exposed to the public internet is probably the least secure but most convenient approach. We do not recommend this approach, and if you do follow it be sure to use the option to force remote clients to connect using SSL (and expect a performance penalty in the process).

Note also that connecting to a PostgreSQL database using QGIS from a remote host may often be slow and irritating to use on a daily basis, especially if your internet connection is not very fast and your datasets are large.

By default the docker-compose.yaml

Publishing with QGIS Server

The workflows described in this section apply equally to any database hosted

Further reading for understanding authentication with PostgreSQL using cert based authentication here:

https://joelonsql.com/2013/04/27/securing-postgresql-using-hostssl-cert-clientcert1/

CHAPTER

SIX

OSGS ROADMAP

Like most Open Source Software, the Open Source GIS Stack is an ongoing work in progress, however the project is anticipating some architectural changes which are likely to be breaking in nature.

This document outlines the various ongoing activities and critical changes expected to be introduced.

6.1 Components

The current roadmap and update strategy consists of various components, some of which are being developed in parallel.

The core components currently under development include:

- Platform and application extension
- Blueprint framework development
- Ongoing documentation updates
- Deployment guidelines and strategies
- Contribution guidelines
- Administration Interface
- Cloud-native architecture migration

6.1.1 Considerations

The current *main* branch is under heavy development and should be considered unstable.

Use the *deployment* branch for production deployments that avoid breaking changes.

Once the architecture changes, it is possible that a *compose-deployment* branch may be maintained for legacy environments, but a more sophisticared release management strategy is likely to be implemented.

6.1.2 Architectural Changes

The current architecture and deployment strategy rely heavily on make targets and a relatively complex docker compose configuration to provide a reasonably simple interface for running a "setup wizard" and declarative infrastructure management.

The intention going forward is to develop a cloud-native infrastructure design using kubernetes with an integrated local deployment strategy for single-server deploys. It is expected that this migration will introduce breaking changes.

In-between the migration to k8s there may possibly be additional functionality introduced as a part of a python-based commandline management solution which leverages docker-compose rather than kubernetes.

6.1.3 Blueprints

The concept for blueprints is to provide default configurations and end to end solutions for application specific purposes or to provide a scaffold for building solutions tailored to a particular domain vertical. These blueprints will be designed to be may include various boilerplate projects, sample data, and other components intended to lower the barrier for entry into location intelligence for various applications.

Currently, the intention is to identify key components critical to solution development along with their locations or endpoints, and develop a structured process or expose an API for merging or introducing additional components which are made available from a remote git repository, along with various hooks or bootstrapping operations.

6.1.4 Administration

The osgs-admin UI is an ongoing effort to provide a "click to run" experience for stack deployment that includes a front-end management console. The application is designed as a unified configuration management and monitoring solution which is currently focussed on managing the stack via docker-compose.

The admin ui is under development at https://github.com/zacharlie/osgs-admin and once it reaches functional beta will likely be migrated to the kartoza github platform. A stripped-down version of the admin-ui application will likely be made available for providing end-users a method of managing docker-compose environments for application specific purposes, whilst the core application will be developed to support the cloud native architecture.