Assignment 02 Case Study:
Setting up an Open Source Enterprise Level GIS Homelab

Patrick Harrison (000733057)

GEOS 540: Applied GIS Directed Field Studies

Southern Alberta Institute of Technology

October 8, 2025

Total Pages: 20

Contents

Executive Summary

With the adoption of Geographic Information Systems (GIS), organizations need to manage their data and workflows on an enterprise level from a central location. While proprietary solutions for this are very robust, this is not feasible for every use case due to costs or technical accessibility. There is a need an for a enterprise level GIS that is accessible to use for individuals and small businesses so that they can take advantage of efficient workflows that a centralized workflow system offers.

A free and open source software (FOSS) solution that remains lean, low-cost, and scalable is implemented for a homelab environment. Using very limited hardware, a robust GIS enterprise tech stack built upon a GNU+Linux server, using Geoserver and PostgreSQL with PostGIS containers. This showed that a solid foundation can provide most of the functionality that proprietary systems provide.

QGIS was used as the primary analysis and cartographic software as QGIS projects can be stored directly in a PostGIS database. Keeping project files in a central location is a feature that proprietary systems like ArcGIS don't have. This allows for greater collaboration and standardization of workflows.

The setup demonstrated the technical feasibility for an individual to have an enterprise level system, provided a testing environment for configuration, and created a professional development platform for acquiring enterprise GIS administration skills. Implementing an enterprise-level GIS involves time investment (though Docker containers streamline deployment), along with hardware and networking expenses. Deploying this system in a homelab setting offers benefits beyond cost savings: it keeps skills current with GIS technologies outside the ESRI ArcGIS ecosystem while establishing a robust framework for organizing and standardizing geospatial workflows, for individual users or organizations.

Introduction

Benefits of Enterprise level GIS

Geographic Information Systems (GIS) provides a tool for enhanced decision making with regards to spatial problems. It has become a common business tool and many government agencies see it as a critical part of their operation (Babinski, 2009). Enterprise level GIS solutions need to be developed to help organizations leverage their information more effectively. Gaudet in 2012 said that “information is knowledge and how a company can leverage it within an organization or across business assets, will impact how successful the company operates”.

Enterprise GIS platforms have traditionally been dominated by proprietary solutions such as Esri’s ArcGIS Enterprise (Shaharudin, van Loenen, & Janssen, 2025). These high upfront cost solutions are not always feasible to individuals or small organizations wanting to use the benefits of enterprise systems. Due to monetary or accessibility costs, the benefits of the proprietary solution can be drastically outweighed by the costs. There is a need for lower cost solutions for smaller businesses and individuals to both learn, and to use enterprise level GIS professionally.

Proprietary enterprise GIS systems often require specialized training and certification programs that are costly and time-intensive. Additionally, system administration and configuration typically demand dedicated IT staff with vendor-specific expertise. These technical accessibility problems can be mitigated by using non-proprietary options along with Docker containers.

FOSS Alternatives

When deciding on a GIS toolset, cost and functionality are two big constraints. The cost of a toolset is not just the up-front licensing or purchase of the software but also the training, deployment and hardware that comes along with it. Free and Open Source Software (FOSS), means that the user has ‘freedom’ to procure, provision, modify and redistribute said software (Sutton, 2023). The freedom granted can be thought of as something to consider as extra functionality. As such, FOSS is not always cost free but can be a much cheaper option with the right skill set. Large communities of support exist to extend FOSS software to suite a variety of needs and although the learning curve can be steep, the information and training required is more accessible.

It can be generally considered that FOSS software is not as feature rich as proprietary options (Sutton, 2023). This may be true, however, by the nature of FOSS software, it is more modular by design, and combining multiple different open-source projects was found to perform resonably well compared to proprietary solutions such as ArcGIS (Donnelly, 2010).

The Homelab Context

The homelab has emerged as a valuable environment for individuals to experiment with technologies that might otherwise be inaccessible. At its core, a homelab is a personal infrastructure setup that allows individuals to host software (Helder, 2024). For GIS professionals, homelab like setups offer an opportunity to use enterprise level systems with a low barrier of entry.

The homelab appraoch is well-suited for enterprise GIS implementations as it mirrors those constraints by small organizations: limited hardware budgets, minimal IT support and the need for future scalability without being sucked into a proprietary subscription based model. By deliberately working within these constraints, a homelab implementation makes this achievable with modest resources while still maintaining professional standards. Modern containerization technologies like Docker have made enterprise-grade deployments even more accessible in homelab environments, allowing complex multi-service architectures to run efficiently on hardware that might otherwise sit idle.

The homelab approach also addresses a disconnect in the GIS profession: while job postings for GIS positions increasingly require knowledge of enterprise systems, databases, and web services, individual practitioners often lack access to these technologies for skill development. By deploying an enterprise-grade tech stack on minimal hardware, a homelab demonstrates that these capabilities are not exclusive to large organizations with substantial IT budgets. This accessibility empowers individuals to maintain technical currency with modern GIS practices, experiment with open source alternatives to proprietary solutions, and develop a deeper understanding of how enterprise GIS systems function beneath their user interfaces.

This case study documented a FOSS enterprise GIS stack using GeoServer, PostgreSQL/PostGIS, and QGIS in a homelab environment. The study focused on core enterprise capabilities including data storage, web services and desktop analysis. It demonstrated the technical feasibility for a small organization to have an enterprise system, provided a testing environment for configuration, and created a professional development platform for acquiring enterprise GIS administration skills.

Literature Review

Enterprise GIS Solutions

The expansion of geographic information system (GIS) processes and services to both professional GIS users and non GIS users throughout an organization requires a centralized infrastructure (Gaudet, 2012). For this infrastructure to be useful, it must at minimum comprise of a data storage solution as well as a method for retrieval, creation, and manipulation of the data. Babinski in 2009 looks at "Framework GIS data" such as transportation, property ownership, and boundaries for municipal systems. These datasets require constant retrieval and updates. They also cross multiple areas of expertise and as such are expensive to maintain. A data sharing approach can minimize costs to make this data useful to the entire organization. This concept extends beyond the municipal sector, as "framework GIS data" can be understood more generally as any foundational dataset that serves as a common, essential resource for a wide range of GIS applications. As an individual this centralization of common data is still very useful.

GIS, being data heavy processes, face the problem of data accessibility. Interviews with Exploration Geologists showed that they spent upwards of 4 hours a day looking for information due to data silos (Gaudet, 2012). This time cost can be very expensive in any type of organization, but can be reduced by using a centralized source of truth. With data being centralized, it can reduce data silos, duplicate data/analysis, as well as promote data information sharing among departments across an organization (Alissa, 2002). How best to implement the centralization for an organization is now the important question.

Sunyaev et al. conclude in 2023 that "an important aspect of future [enterprise information systems] will be their ability to master heterogeneity". In other words, when implementing an enterprise GIS solution, managing how to deal with data and processes that are not standardized will make or break it. Sunyaev et al. also states that a more decentralized enterprise system will create a better end fit for a specific need, however, it comes at the cost of greater complexity and heterogeneity. For a GIS use case, this could look like the projection of the GIS data. A standard solution is keeping hosted data in WGS84 committing to more homogeneous data standards but is not the best fit for all end users.

A more precise definition of an enterprise level GIS solution is given by Woodard wherein they list conditions that must be met for a GIS to be considered an enterprise level GIS.

A GIS transitions into an enterprise system when the following conditions have been met: 1.

> It is based and operates from a centralized spatial database. 2. The system connects multiple departments with multiple users. 3. The organization's executives, managers, supervisors, and staff support and feel they have an ownership stake in the system. 4. It provides these departments and users with a standard set of tools for the creation, editing, analyzing, mapping and distribution of information. (Woodard, 2020)

An interesting requirement here is the users must feel they have an ownership and stake in the system. This feeling would be enhanced by using FOSS software where the user is not licensing software from a vendor but having complete control and freedom to do what they want with it. Another point that Woodard makes is the connection between departments and users. For a homelab setup, the need for multiple users and departments could look like using the principle of least privilege (giving users and systems only the minimum access rights necessary to perform their tasks), and variations in personal projects respectively.

Software Stack

Schumacher in their 2025 paper, aptly named 'Honey, I Shrunk the GIS', provided a tech stack for using containerized software to build out a portable and scalable framework for mapping radiological measurement data. They used containerization and microservices can make the development and deployment much simpler as well as providing a robust base that can be easily built upon for more complex data pipelines. This paper also used a FOSS tech stack including Geoserver, PostgreSQL with PostGIS along with some peripheral software such as PGAdmin for database administration, and Node.js for a custom frontend. Although Schumacher used the software as a single-purpose deployment for a specific solution, the software used can be generalized and deployed to preform more general use cases.

A key part of an enterprise level GIS is the web map server. Owusu-Banahene and Coetzee in 2012 compared three open source web map servers: Map Server, GeoServer and QGIS Server. Although since 2012 many things may have changed in how these products work and their modern viability, this will give a decent base line of their differences that we can cross reference with their respective documentations. What will be important for a Map Server in this case study is ease of use and adherence to web mapping service OGC standards. MapServer and GeoServer ranked higher than QGIS Server in terms of documentation support, and Geoserver ranked higher in terms of WMS standards (Owusu-Banahene & Coetzee, 2012).

The other key part of an enterprise level GIS is the data store. A relational database management system (RDBMS) is ideal as it can provide multi-user editing, user and role management as well as transactional control, and enforce data consistency and integrity. Deepika, Chirag, and Darshit found that PostgreSQL with PostGIS extension out-performed the Oracle's proprietary solution in terms of time but was more computationally expensive in 2016. In terms of Not Only SQL (NoSQL) databases like MongoDB, PostGIS started to fall behind. It was found that MongoDB performed 10 times better on average in certain situations (Agarwal & Rajan, 2017). The findings by Agarwal and Rajan suggested that these databases will outperform pure relational SQL databases in high read-write environments. There are limits to the spatial operations that can be performed in MongoDB however, and PostGIS provides more comprehensive geo-analysis abilities at the database level (Agarwal & Rajan, 2017).

Methodology

System Architecture

The enterprise GIS implementation utilizes a Debian server operating system as its foundation, providing a stable and well-supported GNU+Linux environment. The Host machine has 16Gb of disk space and 4Gb of memory. Due to these limitations, it was crucial to set up disk partitions on the host machine.

Table 1. Server machine disk partitions

Partition	Size (Gb)	Description
`/`	7.0	The root partition where the OS applications live
`/srv`	8.0	Partition to host data.
`swap`	1.0	Swap space to avoid memory overload.

As seen in Table 1 the system is very sparse. The system employs a dedicated `\srv` partition for service data, following Linux Filesystem Hierarchy Standard best practices for separating application data from system files. If the server is trying to store too much data, the operating system will not be effected. This partition strategy enhances data management, backup procedures, and system maintenance.

The core GIS services, GeoServer and PostgreSQL with PostGIS, are deployed as Docker containers, enabling simplified deployment, version control, and service isolation. These containers communicate through a Docker bridge network, which provides internal networking between services while maintaining security boundaries from the host system. The containers inside the docker network have a their data vol-

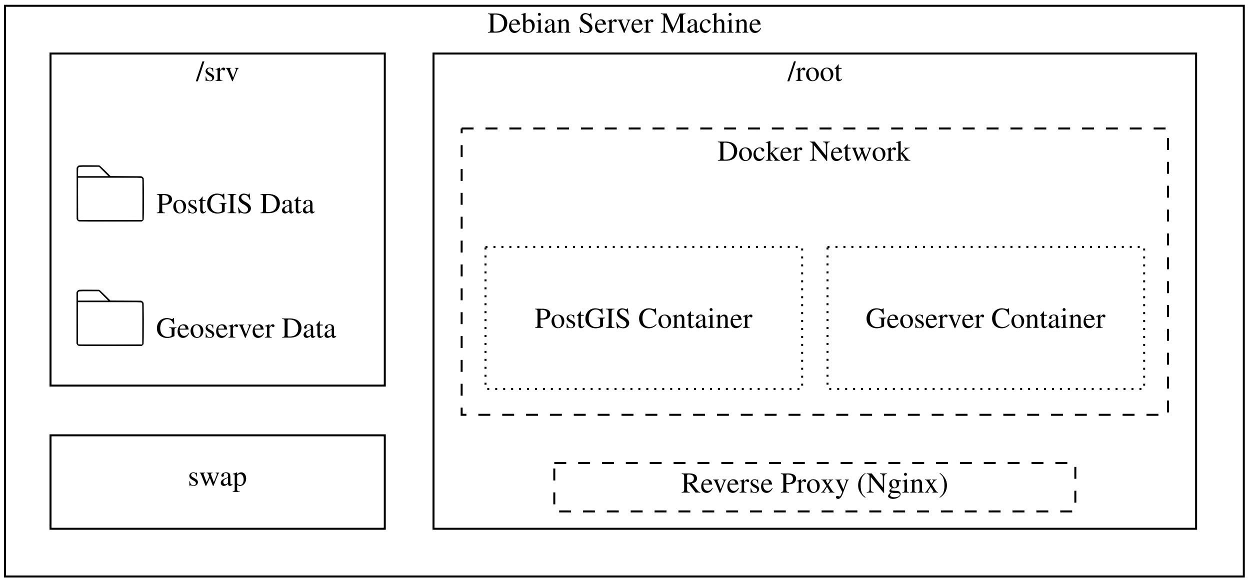


Figure 1. Server architecture showing disk partitions and important components.

umes mapped to specific directories on the host machine in the `\srv` partition. This setup can be seen in Figure 1.

A docker compose script is used to quickly spin up the containers and modify the configurations. This is also where the docker network is defined. By using this, it is reproducable, and easy to see exactly what is going into spinning up the system. It is used in conjuction with a `.env` file so secrets and other variables are stored as environment variables and are not exposed if sharing the configuration. A full `docker-compose.yml` file can be seen in Appendix A.

Part of setting up a centralized server involves a certain knowledge of networking. As this setup is for a homelab implementation, on the same server machine, there is a software called AdGuardHome which does a couple of helpful things for the local network but in this usecase, it allows for a DNS rewrite where the server can be accessed via `domain.local` and in conjunction with the Nginx reverse proxy, `geoserver.domain.local` can be used to access the geoserver UI. Following this, it is important for the machine to have a static ip address on the network. This is achieved by modifying the `/etc/networks/intefaces` file to have the Debian server request a static IP from the DHCP server of your local network. An example can be seen in the listing 1.

Listing 1. /etc/network/interfaces label

```
source /etc/network/interfaces.d/*

# The loopback network interface
```

```
auto lo
iface lo inet loopback

# The primary network interface
allow-hotplug enp1s0
iface enp1s0 inet static # changed to static from dynamic
    # Below lines are needed after changing to static from dynamic ip
    address 192.168.68.169 # Chosen Static IP
    netmask 255.255.255.0  # network masking
    gateway 192.168.68.1   # local network router
    dns-domain local
    dns-nameservers 192.168.68.1
```

The setup of a reverse proxy is fairly simple using Nginx, however it can get complicated fast the more moving parts there is. While this is out of scope for this case study, encrypted connections can be set up so connections are made through the https protocol opposed to http. This process can be done by generating the SSL certificates and setting up the reverse proxy to handle the https connections against the certs.

Database Configuration

This implementation deploys a containerized PostgreSQL database with the PostGIS spatial extension using Docker Compose. PostGIS extends PostgreSQL's capabilities by adding support for geographic objects, enabling location-based queries and spatial data analysis.

The service utilizes the official `postgis/postgis:latest` Docker image, which provides a production-ready PostgreSQL installation with the PostGIS extension pre-configured. This approach eliminates manual installation complexity and ensures consistency across deployment environments..

Database initialization parameters are externalized through environment variables. This way, quick changes can be made to some the configuration that make it less risky to break something. The environment variables here are:

- Database name, username, and password are sourced from environment variables (`POSTGRES_DB`, `POSTGRES_USER`, `POSTGRES_PASSWORD`)
- External port configuration is parameterized via `POSTGRES_EXTERNAL_PORT`

The implementation also employs a bind mount volume strategy, mapping the host directory `/srv /postgis-data` to the container's internal data directory (`/var/lib/postgresql/data`). This architectural decision ensures data persistence across container lifecycle events (restarts, updates, recreations), direct host-level access to database files for backup and recovery operations, and separation of application state from data.

Following some workflows set out by Paterson from Luna Geospatial Inc., multiple databases inside the PostGIS container can be used for different ‘departments’ or projects, while schemas can be used either as the equivalent to the ESRI Geodatabase dataset, or to isolate data for the projects themselves. One big advantage to having complete ownership and control over the backend database is the flexibility it offers in this respect [1].

Geoserver Configuration

This implementation deploys a containerized GeoServer instance using Docker in the same docker bridge network with PostGIS. GeoServer implements Open Geospatial Consortium (OGC) web services, enabling interoperable access to spatial data through industry-standard protocols including Web Map Service (WMS), Web Feature Service (WFS), and Web Coverage Service (WCS).

The deployment utilizes the Kartoza GeoServer image (`kartoza/geoserver:latest`). This is not the official Docker image for Geoserver, however it was found to provide more ‘sane’ defaults, and more environment variables exposed to Docker at startup.

The implementation establishes a direct connection to the PostGIS database service through the database configuration parameters: Because of this, geoserver now depends on the PostGIS service being healthy.

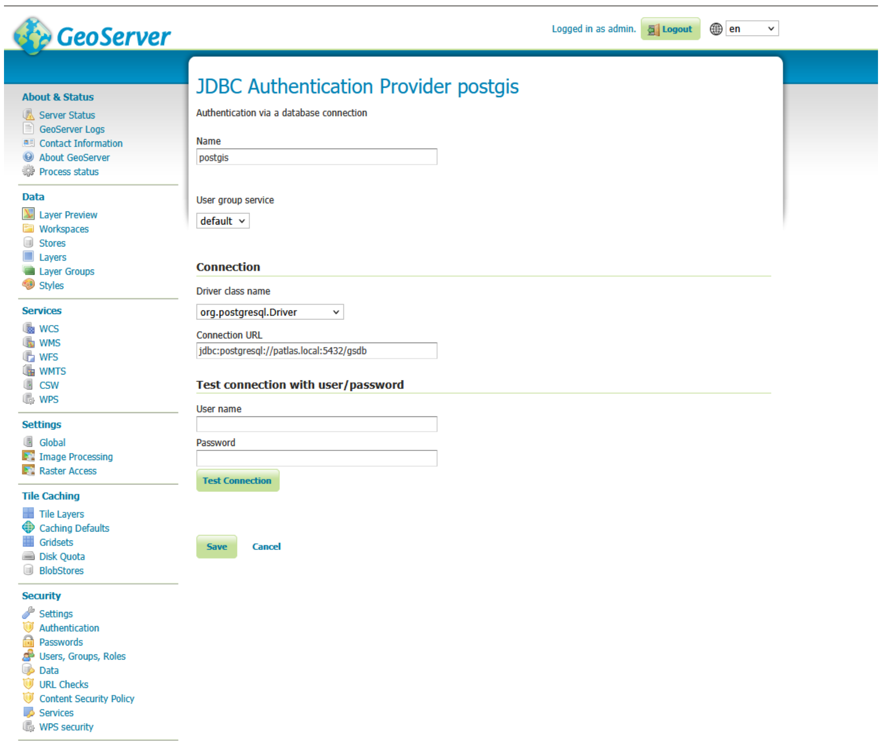
- `DB_BACKEND: POSTGRES`: Specifies PostgreSQL as the primary data store
- `POSTGRES_HOST: postgis`: References the PostGIS service by the Docker Network DNS name
- `POSTGRES_PORT: 5432`: Standard PostgreSQL port
- Database credentials (`POSTGRES_DB`, `POSTGRES_USER`, `POSTGRES_PASSWORD`) are synchronized with the PostGIS service configuration

Additionally `SSL_MODE: disable` disables SSL encryption for internal Docker network communication, which is appropriate for isolated network environments.

[1] I came to suspect that the reason ArcGIS oneline and ArcGIS enterprise limit their ’folders’ to one level deep is due to the fact that they are really just PostgreSQL schemas on the backend

Through the Java Naming and Directory Interface (JNDI) configuration (`POSTGRES_JNDI: true` and `POSTGRES_JNDI_NAME: jdbc/postgres`), GeoServer maintains a pooled connection to the PostGIS database that serves dual purposes: accessing geospatial data and validating user credentials. This architecture enables GeoServer to store user accounts, roles, and permissions directly within the PostgreSQL database schema, allowing for centralized identity management across the geospatial infrastructure. Rather than maintaining separate authentication stores, this JDBC-based (Java Database Connectivity) approach centralizes user management within the database. The configuration supports role-based authorization for layer and service access, and simplifies credential management by maintaining a single source of truth for authentication data. See the setup for this in Geoserver in Figure 2

Figure 2. Manual setup of authenticating geoserver users with the PostGIS server



JDBC Authentication Provider postgis

Authentication via a database connection

Name: postgis

User group service: default

Connection

Driver class name: org.postgresql.Driver

Connection URL: jdbc:postgresql://patlas.local:5432/gsdb

Test connection with user/password

User name:

Password:

Because this system is very light weight, memory allocation parameters are externally configured to support performance tuning. In this case, an initial 2Gb `INITIAL_MEMORY` and 3Gb `MAXIMUM_MEMORY` was set. This parameterization enables environment-specific resource optimization based on workload characteristics and available system resources.

The implementation employs a dual-volume strategy for operational data segregation. The logs are also stored on the `/srv` partition to not overloaded the root partition.

- `/srv/geoserver-data:/opt/geoserver/data_dir`: GeoServer configuration, layer definitions, styles, and metadata
- `/srv/geoserver-logs:/opt/geoserver/logs`: Isolates application logs for centralized monitoring and analysis

The Geoserver UI is mapped to port 8081 of the host machine as per the docker compose configuration (see Appendix A). Thus when accessing `geoserver.domain.local` from a machine on the same network, you will get the Geoserver UI. Geoserver was not straight forward to set up the proxy for. The port redirects and Geoserver needs to know what the base URL is so it can redirect properly.

Listing 2. nginx configuration for Geoserver: /etc/nginx/available-sites/domain.local.conf

```
# GeoServer subdomain
server {
    listen 80;
    server_name geoserver.domain.local;

    client_max_body_size 100M;

    location = / {
        return 301 /geoserver/;
    }

    # Root goes directly to geoserver
    location ^~ /geoserver/ {
        proxy_pass http://localhost:8081/geoserver/;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header Host $http_host;
        proxy_set_header X-Forwarded-Proto http;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

    # fix redirects:
        proxy_set_header X-Forwarded-Host $http_host;
        proxy_set_header X-Forwarded-Server $http_host;
    }
}
```

Results

To measure the feasibility of the setup, a simple project that involves many aspects of common GIS workflows and data requirements is performed. Given a study area or study point, create a site base map to aid in field operations that shows the Alberta Township System and base features [2]. The project

[2]This was Assignment 03 in GEOS410 when I did it.

involves spatial queries that can be done at the database layer for filtering, as well as providing an example of standardized styling that can be applied to layers in the enterprise system.

Setting up the database and granting permissions to users so that the principle of least privilege is followed is quite simple.

Listing 3. Database Creation language

```
CREATE DATABASE field_operation_support
    WITH
    OWNER = gis_admin
    TEMPLATE = postgres
    ENCODING = 'UTF8'
    LOCALE_PROVIDER = 'libc'
    TABLESPACE = pg_default
    CONNECTION LIMIT = -1
    IS_TEMPLATE = False;
GRANT ALL ON DATABASE field_operation_support TO gis_editor;
GRANT CONNECT ON DATABASE field_operation_support TO gis_readonly;
GRANT ALL ON DATABASE field_operation_support TO gis_admin;
```

Three separate schemas were then made, one to hold all the raw ATS grid data, one to hold the base feature data, and one schema to hold data for the specific project. This way, table views can be created for the project schema and there is no duplication of data but a simple filtering at the database level. The raw base and ATS grid data was loaded into the database using the QGIS DB Manager.

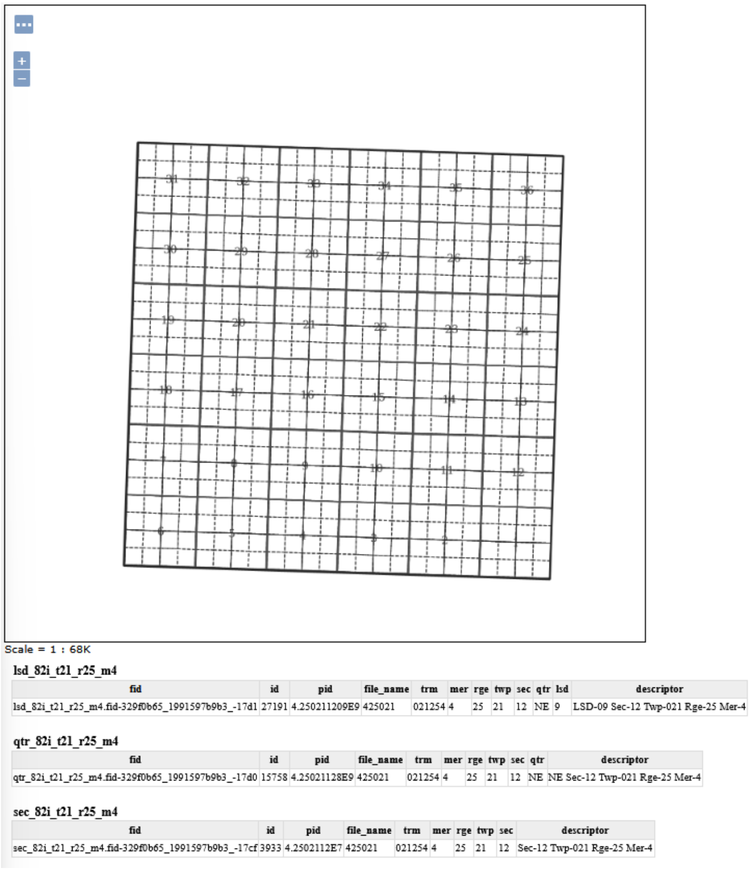
Table views can then be used in the `caresland_project` schema to select the needed ATS township where the study point is contained within the ATS section. An example SQL to create this view:

```
CREATE VIEW carseland_project.twp_82i_t21_r25_m4 AS
SELECT *
FROM ats.twp_82i
WHERE twp = 21
  AND rge = 25
  AND mer = 4;
```

The views can then be published as their own web feature service (WFS) or web mapping service (WMS) on geoserver, and using styles exported from QGIS, geoserver provides an OpenLayers web viewing. See Figure 3 for an example Group Layer viewing in Open Layers on geoserver. The layers can then be added as a WMS as seen in figure 4 while keeping the symbology set in Geoserver.

Here we have set up a complete workflow for a GIS system. The system operates from a centralized spatial data base on a server, including the project file. The system also knows how to deal with multiple users with different permissions that are managed centrally on the database. The users of the system have

Figure 3. OpenLayers view from Geoserver of ATS Grid



Scale = 1 : 68K

lsd_82i_t21_r25_m4

fid	id	pid	file_name	trm	mer	rge	twp	sec	qtr	lsd	descriptor
lsd_82i_t21_r25_m4.fid-329f0b65_1991597b9b3_-17d1	27191	4.250211209E9	425021	021254	4	25	21	12	NE	9	LSD-09 Sec-12 Twp-021 Rge-25 Mer-4

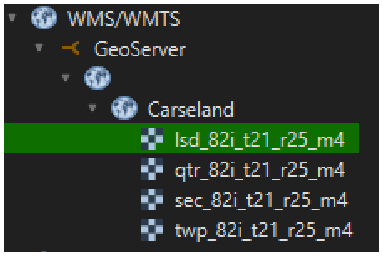
qtr_82i_t21_r25_m4

fid	id	pid	file_name	trm	mer	rge	twp	sec	qtr	descriptor
qtr_82i_t21_r25_m4.fid-329f0b65_1991597b9b3_-17d0	15758	4.25021128E9	425021	021254	4	25	21	12	NE	NE Sec-12 Twp-021 Rge-25 Mer-4

sec_82i_t21_r25_m4

fid	id	pid	file_name	trm	mer	rge	twp	sec	descriptor
sec_82i_t21_r25_m4.fid-329f0b65_1991597b9b3_-17cf	3933	4.2502112E7	425021	021254	4	25	21	12	Sec-12 Twp-021 Rge-25 Mer-4

Figure 4. Adding WMS from the Geoserver in QGIS



complete control and knowledge over system as well giving ownership directly to the users. The system also provides the standard set of tools for the reation, editing, analyzing and mapping/publishing of the information. According to (Woodard, 2020), we have a fully fledged enterprise GIS comprised soley of FOSS software.

Conclusion

The containerized approach offers several advantages: rapid deployment and teardown for testing configurations, easy scaling of resources, and the ability to maintain multiple versions or instances simultaneously. Combined with the structured storage approach using the `/srv` partition, this architecture creates a lightweight yet robust enterprise GIS platform capable of supporting individual users or small organizational workflows while remaining extensible for future growth at a low cost.

While the current implementation serves single-user or small organizational deployments effectively, the architectural foundation supports several expansion pathways without requiring a complete redesign. The Docker networking model enables straightforward integration of additional services such as tile caching layers (MapProxy, GeoWebCache) and authentication proxies (OAuth2 Proxy, OpenLDAP) OpenLDAP was briefly considered to centralize the authentication of users, however adding this additional, admittedly super confusing and over engineered approach, was found to be to be unreasonable for this small of a system.

The standardized OGC service interfaces exposed by GeoServer ensure compatibility with a broad ecosystem of GIS clients and spatial analysis tools, from desktop applications like QGIS to web mapping libraries including OpenLayers and Leaflet. Resource allocation parameters for GeoServer's JVM can be adjusted dynamically to accommodate increasing concurrent user loads or larger dataset processing requirements. For organizations experiencing growth, this architecture can transition to orchestrated container platforms such as Kubernetes, leveraging the existing container definitions while gaining advanced scheduling, load balancing, and high availability capabilities. The implementation represents not only a functional GIS deployment, but a foundation for growing geospatial infrastructure requirements.

Future work would need to continue trying to stream-line some of the processes such as publishing to Geoserver. The publishing to Geoserver is seemed slightly clunky comparatively to ArcGIS, however there are plugin solution in QGIS such as GeoCat Bridge and Geoserver Explorer (though Geoserver Explorer is not supported for QGIS>3.0/Python3) to help ease this process. Currently GeoCat Bridge is a good solution, but is has a bug that stops the user from publishing layers that have a PostGIS data source. Overall, if the proprietary solutions for enterprise level GIS is out of reach, FOSS solutions can be a good alternative with the right knowledge base and the time to piece together a cohesive system.

References

Agarwal, S., & Rajan, K. (2017, 9). Analyzing the performance of nosql vs. sql databases for spatial and aggregate queries. *FOSS4G*. Retrieved from `https://doi.org/10.7275/R5736P26` doi: 10.7275/R5736P26

Alissa, B. (2002, 2). Needs analysis improves enterprise gis. *American City & County*, *117*, 10.

Babinski, G. (2009, 2). Gis as an enterprise municipal system. *Government Finance Review*, *25*, 22-30. Retrieved from `http://libresources.sait.ab.ca/login?url=https://www.proquest.com/trade-journals/gis-as-enterprise-municipal-system/docview/229723743/se-2`

Deepika, S., Chirag, S., & Darshit, S. (2016, 9). Comparing oracle spatial and postgres postgis. *IJCSC*, *7*, 95-100. Retrieved from `www.csjournalss.com`

Donnelly, F. P. (2010). Evaluating open source gis for libraries. *Library Hi Tech*, *28*, 131-151. doi: 10.1108/07378831011026742

Gaudet, N. (2012). *Enterprise gis & architecture* (Doctoral dissertation, University of Denver). doi: 10.56902/ETDCRP.2012.2

Helder. (2024, Apr). *What is a homelab and why should you have one?* Linux Handbook. Retrieved from `https://linuxhandbook.com/homelab/`

Owusu-Banahene, W., & Coetzee, S. (2012, 10). An evaluation of the suitability of three open source map servers for setting up a geospatial thematic web service. In *Proceedings: Gissa ukubuzana 2012 conference, johannesburg, october* (pp. 2–4).

Paterson, C. (2022). *Qgis postgis and geoserver in an enterprise environment.* Luna Geospatial. Retrieved from `https://youtu.be/Myv6WghrItE?si=mNEw90hUCFOmNVXB`

Schumacher, A. (2025, 7). Honey, i shrunk the gis: Developing scalable and lightweight geospatial software applications with microservices and containerization. In (Vol. 48, p. 225-231). International Society for Photogrammetry and Remote Sensing. doi: 10.5194/isprs-Archives-XLVIII-4-W13-2025-225-2025

Shaharudin, A., van Loenen, B., & Janssen, M. (2025, 2). Developing an open data intermediation business model: Insights from the case of esri. *Transactions in GIS*, *29*. doi: 10.1111/tgis.13304

Sunyaev, A., Dehling, T., Strahringer, S., Xu, L. D., Heinig, M., Perscheid, M., . . . Rossi, M. (2023, 12).

The future of enterprise information systems. *Business and Information Systems Engineering*, *65*, 731-751. doi: 10.1007/s12599-023-00839-2

Sutton, T. (2023, 6). *Deciding between fossgis and proprietary software in the enterprise.* GIM International. Retrieved from `https://www.gim-international.com/content/article/deciding-between-fossgis-and-proprietary-software-in-the-enterprise`

Woodard, J. (2020). *Enterprise gis : concepts and applications*. CRC Press. (Includes bibliographical references and index.)

A Full docker-compose.yml

```
services:
  # PostgreSQL with PostGIS extension
  postgis:
    image: postgis/postgis:latest
    container_name: postgis
    environment:
      POSTGRES_DB: ${POSTGRES_DB}
      POSTGRES_USER: ${POSTGRES_USER}
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
    volumes:
      - /srv/postgis-data:/var/lib/postgresql/data
    ports:
      - "${POSTGRES_EXTERNAL_PORT}:5432"
    networks:
      - mapgard
    restart: unless-stopped
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U ${POSTGRES_USER} -d ${POSTGRES_DB}"]
      interval: 30s
      timeout: 10s
      retries: 5

  # GeoServer
  geoserver:
    image: kartoza/geoserver:latest
    container_name: geoserver
    environment:
      GEOSERVER_ADMIN_USER: ${GEOSERVER_ADMIN_USER}
      GEOSERVER_ADMIN_PASSWORD: ${GEOSERVER_ADMIN_PASSWORD}

      DB_BACKEND: POSTGRES
      POSTGRES_HOST: postgis
      POSTGRES_PORT: 5432
      POSTGRES_DB: ${POSTGRES_DB}
      POSTGRES_USER: ${POSTGRES_USER}
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
      SSL_MODE: disable
      POSTGRES_SCHEMA: public

      POSTGRES_JNDI: true
      POSTGRES_JNDI_NAME: jdbc/postgres

      INITIAL_MEMORY: ${GEOSERVER_INITIAL_MEMORY}
      MAXIMUM_MEMORY: ${GEOSERVER_MAXIMUM_MEMORY}

      SAMPLE_DATA: "false" # No GS sample data

      STABLE_EXTENSIONS: importer-plugin # Needed for GeoCat Bridge publishing

    volumes:
      - /srv/geoserver-data:/opt/geoserver/data_dir
      - /srv/geoserver-logs:/opt/geoserver/logs

    ports:
      - "${GEOSERVER_EXTERNAL_PORT}:8080"
    depends_on:
      postgis:
        condition: service_healthy
    networks:
      - mapgard
    restart: unless-stopped
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:8080/geoserver/web"]
      interval: 30s
      timeout: 10s
      retries: 5

networks:
  mapgard:
    driver: bridge
    ipam:
      config:
        - subnet: ${NETWORK_SUBNET}
```