Open-Source GIS Sandbox & Data Stories - New tools for workshop interactivity

Marc Folini

May 2022

GIS courses benefit from supplementing theoretical concepts with exercises for participants to become an active part in the learning experience. This requires participants to have access to an environment that provides the necessary software and data - ideally with minimal setup effort, independent of the system they use and easy to uninstall without leaving traces after the workshop. Different approaches were evaluated, and container technology was found to be a good fit to abstract away the setup hassle. A proof of concept of what will be referred to as *Sandbox* was implemented and made publicly available free to use under MIT license. The Sandbox is based on Docker Compose and includes a spatial database, an OGC data server, GDAL, and Python scripting capabilities with access to libraries from the geo-ecosystem. It requires only minimal setup and a single command to be started and stopped respectively.

In order to help participants making sense of the Sandbox components and their interactions the concept of *data stories* was explored. Designed as data-driven cross-application tutorials, they allow participants to fully focus on how different applications interact to transform raw data into meaningful results. This is in contrast to many other tutorials, to which data is secondary to teach application-centered functionalities in a narrow scope. Built upon the Sandbox capabilities, an example data story was implemented to create a bike suitability map from raw open data from the city of Zürich. Participants learn about the GDAL command line tools to inspect datasets and load them into PostGIS, where spatial processing and aggregation is performed. The result is made available as a new dataset to be consumed and styled in QGIS.

Contents

1 Introduction

The GIS[1] ecosystem today is extensive and fast evolving with multiple technologies addressing similar use cases with varying overlap in functionality. The motivation to employ GIS technology is often data-driven, i.e. a company usually wants to gain insights from spatial data or process it to be used downstream in products. This requires an end-to-end understanding how different components of the GIS ecosystem interact to transform the raw data into the desired product. For somebody new to the field, the vast amount of possibilities might be overwhelming at best and frustrating at worst when trying to develop a holistic mental model about what technologies to use in what way to tackle real world use cases.

Many tutorials exist, but these are usually centered around a specific tool or technology, whereby data is of secondary nature to explain the functionalities in a narrow scope. Workshops and lectures are another common vehicle to acquire knowledge about a field, whereby it is desirable for participants to experience theoretical concepts hands-on for themselves. The challenge lies in providing all participants with access to the necessary software and data with minimal setup efforts. This is exacerbated when it comes to more complex setups with multiple components that need to interact.

It is hypothesized that there is a gap between what people new to the field, lecturers and workshops organizers need and what is widely available. This thesis was written as part of the CAS RIS 2021/22[2] course and aims to address this gap on two levels:

1. **Infrastructure** In order to provide participants access to a variety of components from the GIS ecosystem with minimal setup effort the concept of a *Sandbox* environment was explored. The requirements and components to be implemented were chosen to support the CAS RIS course curriculum.

2. **Content** In order to help participants making sense of the Sandbox components and their interactions, the concept of a *data story* was explored. Each data story is a data-driven cross-application tutorial. It has a tangible data driven use case and provides guidance on how available technologies and tools interact with the data to achieve the desired result. An example data story was conceptualized and implemented to create a bike suitability map from raw open data from the city of Zürich.

[1] Geographic Information System (GIS).
[2] Certificate of advanced studies (CAS) in GIS (in German RIS) offered by ETH Zurich.

2 Sandbox

2.1 Requirement analysis

The content of the CAS RIS course was analyzed for potential hands-on exercises. From these findings functional and non-functional requirements[3] for a Sandbox were derived.

2.1.1 Content analysis of CAS RIS curriculum

At the time of writing the CAS RIS course curriculum consisted of four weeks of content modules on a wide variety of topics. Each module was screened with a focus on where a Sandbox could facilitate hands-on experiences without substantially altering the module structure or material. Modules focusing primarily on usage of proprietary software like ArcGIS were ignored. Table 1 summarizes the findings.

2.1.2 Functional requirements

The following functional requirements were identified for the Sandbox to be useful in realizing the potential identified in the curriculum analysis:

- A spatial database is available together with utilities to import and query data.

- An OGC[4] server implementation is available. There is a way to read predefined example data, read data from the user's system or connect to the Sandbox spatial database.

- An OGC client implementation is available, which can interact with the Sandbox OGC server or any other OGC service publicly available on the internet.

- GDAL[1] command line utilities such as gdalinfo, ogrinfo and ogr2ogr are available. There is a way to read either predefined example data or read data from the user's system.

[3] Functional requirements specify the tasks an application must be able to perform. Non-functional requirements specify additional aspects about the application performing the task, for example usability, look and feel or performance.

[4] The Open Geospatial Consortium (OGC) is a worldwide community to improve the access to geospatial data by various means such as developing standards. Such standards for the web include Web Map Service (WMS), Web Feature Service (WFS) and Web Coverage Service (WCS) to request styled map images, simple geometry features and raster data via HTTP respectively.

Table 1: Content analysis of existing lectures for Sandbox potential.

Module	**Sandbox potential**
Interoperabilität	- Inspection of and conversion between various data formats. - Read Shapefile into a spatial database.
SQL	Write and run basic SQL queries on sample data.
Geodatenbanken	- Run basic spatial queries on sample data. - Perform spatial aggregation on sample data.
Geometrische Methoden	Run simple examples on sample data.
Topologische Methoden	
Mengenmethoden	
Statistische Methoden	
Einführung in Python	Self-study introduction of Python basics.
Internet und GIS I & II	- Load data from OGC web services WMS/WFS/WCS. - Consume WMS with an OGC compliant client system.
Projekt Internet & GIS	- Use geodatabase as data source for OGC server. - Create style and serve data as WMS. - Consume WMS with an OGC compliant client system.

- An environment is available to write and run Python scripts which make use of the Python geo-ecosystem. A use case would be to provide illustrative examples for the concepts of geometry, topology and spatial statistics.

2.1.3 Non-Functional requirements

- Users can create content which is persisted between restarts of the Sandbox.

- There is a possibility to distribute example scripts and data as part of the Sandbox.

- Creation and distribution of data stories that are compatible with the tools of the Sandbox should be possible without in-depth technological knowledge.

- The Sandbox can be installed effortlessly and in a uniform way across the most common operating systems, namely Apple OS, Linux and Windows.

- The Sandbox can be uninstalled without leaving traces on the operating system.

- Turning the Sandbox on/off and accessing components should not require any programming knowledge.

- A partial or total reset of applications and data is possible in case something gets messed up.

- Usage is not coupled to the duration of the CAS RIS course or otherwise restricted, e.g. via externally controlled credentials.

2.2 Existing projects

Google search engine was used to look for already existing similar projects using various combinations of the terms *gis*, *geo*, *sandbox*, *workshop*, *playground* and *spatial*. No project was found that satisfied the requirements outlined in section 2.1, but some noteworthy findings are listed below.

- Joeyklee's Geosandbox[2] consists of tutorials that focus on JavaScript client side mapping libraries such as Leaflet. It is not suitable for our purpose but mentioned here to avoid confusion due to the similar name.

- IDRE Sandbox[3] seems to be a collection of workshops and tutorials. It is mentioned here because the interesting goals and presentation of some workshops

served as inspiration for the data story of this thesis.

- Digital Earth Australia (DEA) Sandbox[4] was a wonderful discovery. After creating a free account a cloud hosted JupyterLab[5] platform with extensive Python Jupyter Notebooks provides an interactive engaging experience to learn about the analysis of satellite imagery and much more. The DEA Sandbox was a major inspiration to investigate the capabilities of JupyterLab more closely.

- Geopython Workshop Repository[6] is an extensive knowledge hub about a broad range of geospatial topics. The workshop is built on Python Jupyter Notebooks using a containerized JupyterLab environment with pre-installed libraries. Docker Compose[7] is used as container orchestration technology. Even though the setup is too limited for the use case of this thesis (a spatial database is missing for example) the idea of using container technology to abstract away dependency management provided major inspiration for this thesis.

- The GeoStack Project[8] describes itself as "A Self-Study Beginner Course in Open Source Geospatial Programming for Data Scientists". It is a massive repository of knowledge consisting of workshops and cookbooks, some 180+ pages heavy. While the content is amazing, its structure and presentation was found to be overwhelming and bloated. The tasks aim to solve real-world use cases in a cross-application manner which provided major inspiration for the data story of this work. The software setup is not trivial as it requires a virtual machine and running several scripts.

2.3 Technology evaluation

2.3.1 High level architectural approaches

The research on existing projects and further exploration of potential technologies led to three distinct groups of architectural approaches to be evaluated: Sandbox as direct installation on user system, Sandbox as local containerized application and Sandbox as cloud service.

Sandbox as direct installation on user system

It was found that only four established open-source projects (QGIS[9], PostGIS[10], pgAdmin[11] and GeoServer[12]) would be enough to satisfy the major part of the functional requirements. While the roles of PostGIS as spatial database and GeoServer as

OGC data server are well-defined, the QGIS installation is particularly interesting because it comes bundled with versions of GDAL and Python that work together well. The community has produced installers for all major operating systems, which makes the installation relatively straight-forward.

When it comes to non-functional requirements this approach falls short. A major shortcoming is the intrusive setup stemming from the lack of separation of the Sandbox software from the rest of the user's system. Depending on already existing software and specific configurations this approach is at best error-prone and at worst capable to permanently alter the user's system. Because only the tools themselves are installed, tutorials and data stories would need to be provided in a separate way.

Sandbox as local containerized application

Container technology has been around for decades but took off in 2013 with the release of Docker, which provided the tools that made container technology accessible to the broader mass. The basic idea is to package applications, so they can be run in isolation from other applications. Compared to virtual machines containers are very lightweight and can be started and stopped within seconds.

This approach takes the direct installation approach one step further by running a spatial database and an OGC data server as containerized applications. A difference from the direct installation approach is that QGIS was found to be not suitable because of extra configuration required for graphical output on the computer the containerized applications run on, hereafter called *host system*. On a positive note, complex installation procedures for containerized applications can be fully abstracted away from the user during the packaging process. This allows to fulfill the functional requirements without QGIS by setting up a JupyterLab environment with system dependencies like GDAL in order to have access to GDAL terminal commands and a wide variety of Python packages from Python's geo-ecosystem. Example scripts and data stories can be distributed as part of the container and by using virtual volumes they can be persisted across Sandbox restarts or selectively reset.

An initial setup complexity is the installation of a container runtime, here Docker[5], on the user's computer. However, this is an easy and mature process for all major operating systems at this point.

[5] Any other software that provides a container runtime will do to run containerized applications. Up to this day Docker remains one of the most popular tools for that purpose to the point where it is often used synonymously for everything to do with containers. Docker is chosen for its maturity and excellent documentation.

Sandbox as cloud service

The advances of container technology went hand in hand with the maturation of the publicly available cloud. It's possible today even for a layman to run containerized applications in the cloud at scale using a variety of managed services ranging from authentication to load balancing.

This approach takes the local containerized approach one step further into the cloud. Instead of bothering with the installation of Docker on the user's computer and running the containers locally the Sandbox is made available as a service with a simple web interface and optionally some kind of login. Some functional requirements such as the availability of a spatial database could even be satisfied with cloud native components such as fully managed database services.

The advantage of a cloud service is the low entry barrier because there is no installation required. Example data and data stories can be made readily available as part of the container setup and resetting is as simple as starting a new Sandbox cloud instance. But this convenience comes with a cost and complexity overhead at the cloud side, because for persisting data across sessions some form of authentication must be in place and every cloud session uses resources which are billed by the second. Assuming the cloud Sandbox is publicly available to fulfill the requirement that the Sandbox' usage should not be coupled to the duration of the CAS RIS course, further complexity arises from implementing and maintaining security best practices against malicious attacks. Data management is another topic which can prove challenging when it comes to cloud services in the geospatial domain. Data usually needs to be uploaded before being accessible and spatial data can quickly become rather large for standard internet connection upload speeds leading to waiting times and connection timeouts.

Conclusion: Sandbox as local containerized application

All three approaches satisfy the functional requirements, so the non-functional requirements will be used to argue for the final decision. The direct installation approach falls short here. The local containerization approach has some minor shortcomings when it comes to the initial installation, but shines in almost all other areas. The cloud service approach is the opposite, shining with a zero-effort setup but falling behind the local containerization approach in terms of maintainability, cost, and data management.

The approach *Sandbox as local containerized application* was eventually chosen for further implementation, because the simplicity of a local setup combined with the advantages of container technologies was found to outweigh the cost of the initial setup.

2.3.2 Technology evaluation for local containerized application

The technology has to be able to run multiple containerized applications which can communicate among each other as well as with the host system and store data which is persisted across Sandbox restarts. Among many possibilities, three technologies were considered more closely due to their popularity and the availability of detailed documentation: Docker Compose, Docker Swarm and Minikube.

- **Docker Compose** orchestrates multiple containerized applications on a single host to act like a single application, whereby a simple text file specifies details about the components and their interactions.

- **Docker Swarm**[13] shares many concepts with Docker Compose and can be considered an evolution towards orchestrating multiple containerized applications across multiple host machines. It is also possible to run Docker Swarm on a single host machine.

- **Minikube**[14] is a tool that lets you run a local version of a Kubernetes Cluster. Kubernetes is one of the most popular container orchestration software and at a high level similar to Docker Swarm. It provides its own approach towards managing containers, networking and storage which is generally more geared towards large-scale enterprise applications.

All three technologies were evaluated on a two container setup of a spatial database (PostGIS) and an associated administrator application (pgAdmin). While all three technologies provided extensive documentation that allowed for successful testing, Docker Compose was found to be the simplest solution in terms of initial setup and ease of use, due to its single configuration file. To that end *Docker Compose* was chosen as the technology to implement the *Sandbox as local containerized application*.

2.4 Architecture

The Sandbox was designed as a modular multi-container setup to be run locally by Docker Compose. The architecture is outlined in Figure 1 and will be explained in the following sections.

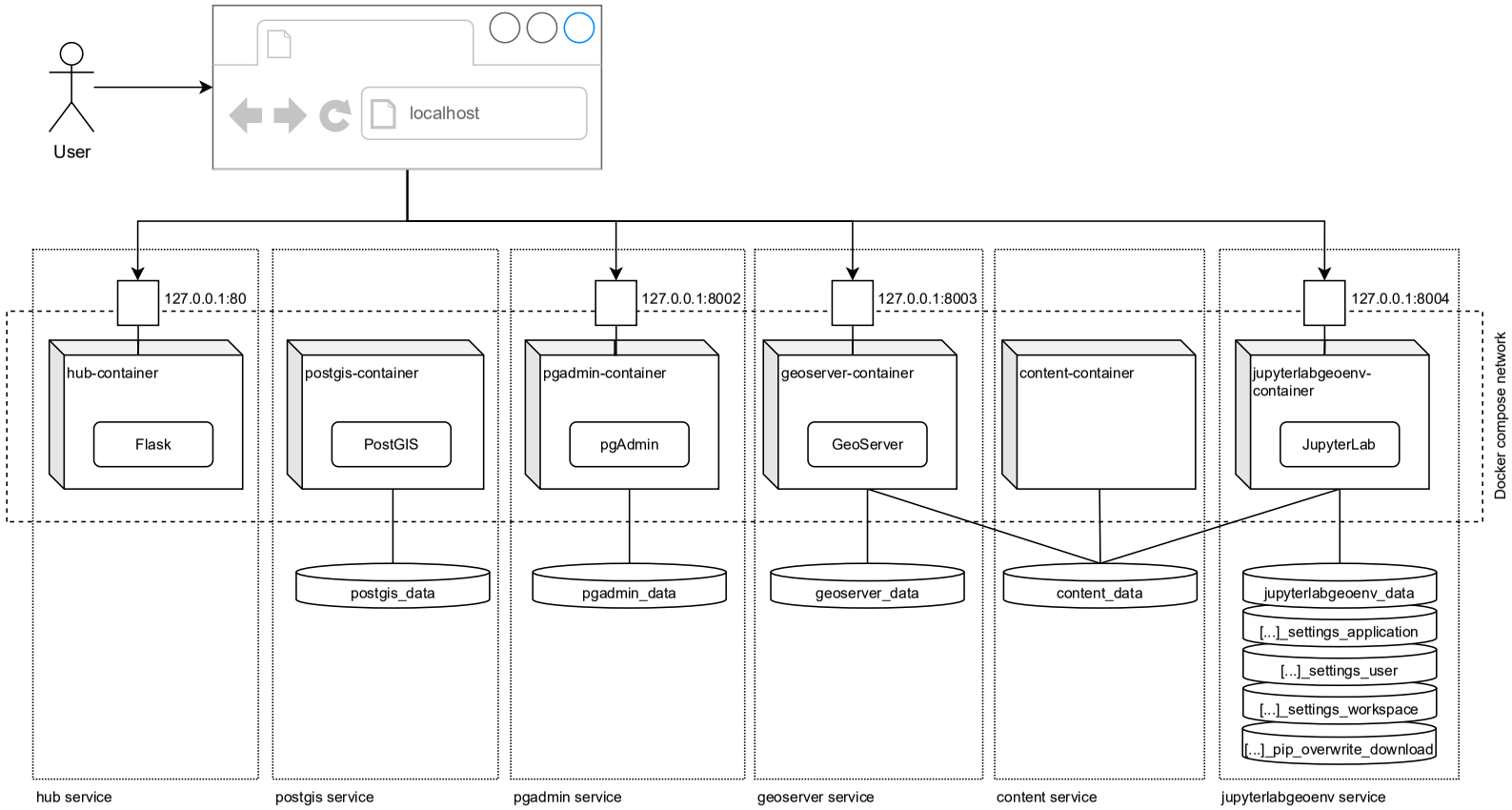

Figure 1: Sandbox Docker Compose architecture

Terminology of Docker and Docker Compose

In Docker terminology, the first step towards packaging an application is to create a *Dockerfile*, a text file which describes the installation procedure and configuration of the desired application package in a standardized way. A Dockerfile is used to *build an image*, a lightweight, standalone package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings[15]. An image is a template which can be easily distributed and from which one or more *containers* can be created within which the containerized application runs in its isolated environment.

Docker Compose orchestrates multiple containerized applications on a single host to act like a single application. At the heart of a Docker Compose application is a configuration file, called *compose-file* which describes the components of the multi-container setup and how they interact. Each component is called a *service*, which consists of an image that specifies the containerized application at the heart of the service and additional configuration about how to interact with other services and the host system. When a Docker Compose application is started, the images of each service are used to create containers within which the application of interest then runs. A compose-file can also declare *volumes*, which can be used by services to save data which should be persisted across restarts of a service and *network settings*, which define how containers can interact with each other and the host network.

Containerized applications

At its heart the Sandbox consists of six services, each defining a container with a distinct purpose:

- **hub-container** runs a web application that serves as an entry point and central hub for the Sandbox. Here a user finds links to access other Sandbox components and the relevant connection information. This *Sandbox Hub* is the first thing a user sees when typing *localhost* in the browser after starting the Sandbox.

- **postgis-container** is a containerized version of PostGIS, a spatial database.

- **pgadmin-container** is a containerized version of pgAdmin4, a graphical user interface to facilitate the administration of PostgreSQL based databases, such as PostGIS.

- **geoserver-container** is a containerized version of GeoServer, which is an OGC data server and a reference implementation of the OGC web services.

- **content-container** serves solely as a data storage without any containerized application. It contains the data story related narrative in form of Jupyter Notebooks as well as the sample data.

- **jupyterlabgeoenv-container** is a containerized version of JupyterLab, a web-based interactive development environment for Python scripts and more. It should come bundled with various pre-installed Python packages from the geo-ecosystem and fundamental libraries such as GDAL and GEOS[16]. This setup allows users to explore the Python geo-ecosystem as well as the various command line utilities that GDAL offers. The expressive nature of Jupyter Notebooks is also used to visualize the data stories.

Usage of volumes for persistent data storage

Some containers make use of virtual volumes as shown in Figure 1 to persist data across restarts of the Sandbox. Examples include the data stored in PostGIS, the database connections set up in pgAdmin or the data and configuration files used by GeoServer. A special case is content-data, which can be accessed by other containers such as jupyterlabgeoenv-container to access the data story Jupyter Notebooks.

Networking

All containers are part of a virtual network managed by Docker Compose which is isolated from the host system. In this network every container can connect to every other container by its container name and the port exposed by the container's application. As shown in Figure 1 some containers also expose selected ports to the host network, which allows to establish a connection to the application in the container from outside the Docker Compose network through the host network's localhost interface.

Access to host file system

Even though not indicated in Figure 1, it is also possible to grant containers access to the file system of the host machine. This seems to contradict the basic idea of isolation but proves to be useful in certain cases. The current Sandbox implementation creates such a link to the host file system for the jupyterlabgeoenv-container and geoserver-container with the intent to make it easy for users to let these components make use of data saved on their local machine.

2.5 Implementation

This section covers the implementation of the Sandbox using Docker Compose. The resulting implementation is provided in this thesis' repository[17] and is free to use under MIT license.

Containerized applications

In a first step, a search was performed on DockerHub[18], an online repository for image distribution. For the following containerized applications of our architecture suitable third-party images were available and the image names with appropriate versions were added to the compose-file:

- For postgis-container the official PostGIS image was used[19].

- For pgadmin-container the official pgAdmin4 image was used[20].

- For geoserver-container a GeoServer image provided by Kartoza was used[21]. This image has configuration options to include community extensions, plugins and sample data.

For each of the other containerized applications a Dockerfile was created describing how to build an image containing the desired application and data:

- hub-container was based on a slim version of the official Python Image[22]. It was extended to run a web application using a Python web-framework called Flask[23] to serve as an entry point to users linking to other Sandbox components with a short description and connection information. Effort was put into designing the user interface to be minimalistic, functional and responsive to different screen sizes. The Sandbox Hub was implemented as a static website using HTML and CSS[6] leveraging Bootstrap 5[24] and Masonry[25] for look and functionality.

- content-container was based on a minimalistic Linux distribution called Alpine[26]. Designed as a data container, it acts solely as a vehicle to distribute data stories and tutorials. The Dockerfile only copies the data to the image together with a script that runs at container startup to copy the content from the image to a shared volume called content-data.

[6] HTML (HyperText Markup Language) defines the structure of a website, i.e. what is considered a header, a title or a paragraph. CSS (Cascading Style Sheets) defines the styling (look) of a website.

- jupyterlabgeoenv-container was based on the official Ubuntu Linux distribution[27]. It was primarily chosen to leverage the UbuntuGIS package respoitory[28] that allowed to install GDAL and other GIS related libraries in a straight-forward way. Using this foundation, JupyterLab and a variety of Python packages from the GIS-ecosystem were compiled into a standard Python requirements file[29] and installed. Lastly, the PostgreSQL command line utility psql was added to allow connecting to and interacting with databases from a JupyterLab terminal.

Upon completing development milestones, semantic versioning[7] was used to mark new releases for each custom component. GitHub Actions[30] were used to create automated workflows that trigger upon each release. These workflows build images from the Dockerfiles and upload them to GitHub Container Registry[31], GitHub's image distribution platform. The image name with appropriate version was then added to the compose-file.

Configurable compose-file

It was deemed important to let users easily configure parts of the Sandbox. For example, the default ports exposed by the Sandbox for external connections might clash with those of other software installed on the host. The Sandbox implements a standard Docker Compose pattern, whereby variables with default values are used in the compose-file, which can be selectively overwritten by creating an environment file[32]. A complete list of variables can be found in the Sandbox repository top level README[17]. In order to reflect changes that impact how to connect to a container in the Sandbox Hub, its content is almost entirely specified in the compose-file using the same variables.

Configurable Python environment in jupyterlabgeoenv-container

The Python environment in jupyterlabgeoenv-container comes with a broad range of packages pre-installed in the image, which results in a fast startup but does not allow persisting user changes such as package additions or modifications. Persisting the whole Python environment on a volume would allow the user to keep all modifications when restarting the Sandbox, but with the downside of loosing them when moving to a new version of jupyterlabgeoenv-container with updated base environment. A compromise solution was implemented, whereby the user can specify additional packages in a dedicated user override requirements file. At each Sandbox shutdown the user modifications are lost because the Python environment is not persisted on a volume. Then at Sandbox startup, packages specified in the user override requirements file are installed on top of

[7] A commonly used specification whereby a version consists of a major, minor and patch number. Increases in major versioning number are not backwards compatible while minor number releases are. Patch releases are usually bug fixes.

the base environment. These user-defined package archives are downloaded and persisted on a volume on first installation, which allows subsequent installations from these archives even without internet connection. One downside with this approach is that each user package increases the startup time of jupyterlabgeoenv-container, because it is only available after the package installation completed. This sequential design was a deliberate choice, because the alternative where the installation happens in the background could easily lead to strange behavior when the user tries to run code that requires a user specified package that has not yet been installed in the background.

Networking and user experience challenge

Containers interacting with each other use the Docker Compose network, while all connections from outside the Sandbox are routed via the host's localhost interface. For the user this means that the way to connect to a Sandbox container differs based on whether the connection happens from another Sandbox container (e.g. Sandbox pgAdmin to Sandbox PostGIS) or from outside the Sandbox (e.g. QGIS to Sandbox PostGIS). Hours were spent researching how to homogenize the user experience without success and in the current implementation the Sandbox Hub makes it as explicit as possible which connection information to use in which case.

2.6 Setup and usage

For a more detailed documentation on how to set up and use the Sandbox please refer to the top level README page of this thesis' repository[17].

Initial setup

The only prerequisite to use the Sandbox is a working installation of Docker, which has become a straight-forward task across all common operating systems in recent years with the maturation of Docker Desktop[33].

Starting and stopping the Sandbox

With Docker Desktop running in the background, only the compose-file[34] needs to be available on the host system. The Sandbox can then be started by running the following

command in a terminal in same folder as the compose-file[8]:

```
docker compose -p sandbox up -d
```

Once the Sandbox started, the Sandbox Hub is available by typing *localhost* in a web browser.

The Sandbox can be shut down by running the following command in a terminal in same folder as the compose-file:

```
docker compose -p sandbox down
```

The -p flag used in the commands above specifies the Docker Compose project name, which is used as a prefix for many Docker Compose components such as networks and volumes. By choosing different project names, it is possible for multiple instances of the Sandbox to coexist with separate volumes and hence separate persisted state.

Removing the Sandbox from your system

To completely clean up all traces from the steps above, simply uninstall Docker Desktop, which will remove all images and volumes. Selective resets of images or volumes are possible from Docker Desktop's Images and Volumes tabs.

3 Data story

3.1 Narrative and presentation

A data story is a data-driven cross-application tutorial. It has a tangible data driven use case and provides guidance on how available technologies and tools interact with the data to achieve the desired result. As such a data story needs a *narrative* that provides a motivating *scenario* to guide the user and *data* that can be transformed by *applications* into the desired result.

While it was clear that the applications will be provided by the Sandbox, several approaches were evaluated how to present the narrative of data stories to the user, ranging

[8] At startup, Docker Compose checks the availability of the images on the host machine and downloads them from DockerHub or other provided image repositories if necessary. These initial downloads will take some time, so make sure to be connected to a fast and reliable internet connection. Further startups use the downloaded images and will only take seconds and work even without internet connection.

from complete decoupling via external static websites or file transfer to full integration into existing containers. It was found that the expressive nature of Jupyter Notebooks was well suited for a data story's narrative, offering full support for text styling through Markdown language and support for images, GIFs and in-line code execution. More so, the creation of new content using Jupyter Notebooks is possible without technical background, which is a big plus compared to an evaluated alternative that framed data stories as static websites made up of HTML and CSS.

3.2 Integration into Sandbox

The decision where to store the Jupyter Notebooks and the associated data in the context of the Sandbox architecture was mainly driven by the requirement to allow for distributing updates and new data story content with minimal impact on the users. To that end a separate *content service* was introduced to the Sandbox as outlined in Figure 1, which consists of a data container called content-container that holds the data story Jupyter Notebooks and associated data. When the Sandbox starts, the container's data gets copied to a volume that is accessible by jupyterlabgeoenv-container. Instead of reinventing the wheel, jupyterlabgeoenv-container is used as the primary way of interacting with the data stories. This pattern has several nice properties:

- Updating a data story is as simple as updating the Jupyter Notebooks and associated data in the Sandbox repository, followed by releasing a new version of the content-container component.

- From the user perspective, obtaining new versions of data stories at the next Sandbox start is as easy as updating the content-container component version in the compose-file (or downloading the updated compose-file from the repository). Assuming only the content-container component version changed, the update will be very quick because all other components do not need to be downloaded again.

3.3 Example data story - Zürich bike suitability map

An example data story was conceptualized based on topics from the CAS RIS curriculum and implemented using the capabilities of the Sandbox. Open data from the city of Zürich is used to derive a visualization showing what percentage of roads in each district are suitable for biking. Participants learn about GDAL's command line tools to inspect datasets and load them into PostGIS, where spatial processing and aggregation is performed. The result is made available as a new dataset to be consumed and styled as an informative map in QGIS. The following section demonstrates the

data story scenario as presented to the user. The full data story can be found under *_content/datastories/bike_suitability_zurich* in jupyterlabgeoenv-container.

Scenario

Imagine you are working at a small company that is specialized in city micromobility. Your company allows people to easily rent bikes on an ad-hoc basis to get around town. You recently participated in a brainstorming session on how to increase public awareness about the fact that still many roads are not suitable to be used by bikes.

At that meeting the idea was born to visualize the ratio of streets which are suitable for biking for each district in Zürich. This ratio, or "biking suitability indicator", could then be turned into beautiful maps to highlight leader and laggard districts. This would surely spark a public debate on the topic! Your team even had the idea to make this not only a one-time thing, but to update this indicator and downstream visualizations on a monthly basis to incentivize the government to take action.

Excited by this idea, your friend checked the publicly available geodata of the city of Zürich and sent you two Shapefiles containing the city districts and the road network with indication which roads are suitable for biking. As the GIS expert the joy of making this work is now on you...

You plan to make use of a spatial database which can be accessed by your coworkers to reduce the number of files being sent around. Furthermore, you plan to create this new dataset in a way which is easy to update when the underlying data changes - thus making the monthly update as painless as possible.

You plan your work in steps:

1. Explore the road and district data your coworker sent you.
2. Load the Shapefile data into PostGIS.
3. Calculate the ratio of roads suitable for bikes per district.
4. Create a new dataset to share with coworkers.
5. Visualize the results as a map in QGIS.

Let's get to work! The full data story can be found under *_content/datastories/bike_suitability_zurich* in jupyterlabgeoenv-container.

4 Conclusion

4.1 Results

During this thesis a GIS Sandbox was created and made publicly available free to use under MIT license[17]. The Sandbox is based on Docker Compose and includes a spatial database, an OGC data server, GDAL and Python scripting capabilities with access to libraries from the geo-ecosystem. It requires only minimal setup and a single command to be started and stopped respectively.

The functional requirements for the Sandbox were derived by screening the existing CAS RIS course modules for potential where a Sandbox could be leveraged to provide participants with hand-on exercises to become an active part in the learning process. With that in mind the concept of data stories was established and implemented in a way that seamlessly integrates with the Sandbox and is easily extensible with further content.

Built upon the capabilities of the Sandbox an example data story was conceptualized and implemented to create a bike suitability map in QGIS from raw open data from the city of Zürich.

4.2 Outlook

The Sandbox potential identified in table 1 could be transformed into hands-on learning experiences with the new capabilities provided by the Sandbox. This could either happen by leveraging the data stories framework, by creating Jupyter Notebooks with exercises that are distributed independently and just accessed via the Sandbox, or by simply giving participants time to experiment on existing examples from slides within the Sandbox.

The project could be presented to the community to adopt or modify it for their own purposes. A potential next step could include a lightning talk at a GIS conference.

Finally, it is the hope of the author that the easy access to a range of powerful open-source software could be an inspiration and potential backbone for a dedicated course module on the nature, importance and challenges of the GIS open-source ecosystem. Such a module could provide insights beyond treating open-source as free-but-hardly-usable alternative to commercial products as it is the case in some of today's GIS courses.

References

[1] *GDAL*. Apr. 21, 2022. URL: https://gdal.org/.

[2] Joey Lee. *Geosandbox*. Apr. 21, 2022. URL: https://joeyklee.github.io/geosandbox/.

[3] *IDRE sandbox*. Apr. 21, 2022. URL: https://sandbox.idre.ucla.edu/sandbox/category/projects/.

[4] Geoscience Australia. *Digital Earth Australia (DEA) sandbox*. Apr. 21, 2022. URL: https://www.dea.ga.gov.au/developers/sandbox.

[5] *JupyterLab*. Apr. 21, 2022. URL: https://jupyterlab.readthedocs.io/en/stable/.

[6] *Geopython workshop*. Apr. 21, 2022. URL: https://github.com/geopython/geopython-workshop/.

[7] *Docker Compose*. Apr. 21, 2022. URL: https://docs.docker.com/compose/.

[8] *The GeoStack Project*. May 10, 2022. URL: https://the-geostack-project.github.io/.

[9] *QGIS*. Apr. 21, 2022. URL: https://qgis.org/de/site/.

[10] *PostGIS*. Apr. 21, 2022. URL: https://postgis.net/.

[11] *pgAdmin*. Apr. 21, 2022. URL: https://www.pgadmin.org/.

[12] *GeoServer*. Apr. 21, 2022. URL: https://geoserver.org/.

[13] *Docker Swarm*. Apr. 21, 2022. URL: https://docs.docker.com/engine/swarm/.

[14] *Minikube*. Apr. 21, 2022. URL: https://minikube.sigs.k8s.io/docs/.

[15] *Docker Image definition*. Apr. 24, 2022. URL: https://www.docker.com/resources/what-container/.

[16] *GEOS*. Apr. 21, 2022. URL: https://libgeos.org/.

[17] Marc Folini. *OS GIS Sandbox*. Apr. 21, 2022. URL: https://github.com/laiskasiili/os_gis_sandbox/.

[18] *Docker Hub*. Apr. 21, 2022. URL: https://hub.docker.com/.

[19] *Official postgis/postgis image*. Apr. 21, 2022. URL: https://registry.hub.docker.com/r/postgis/postgis/.

[20] *Official dpage/pgadmin4 image*. Apr. 21, 2022. URL: https://hub.docker.com/r/dpage/pgadmin4/.

[21] *Kartoza Geoserver Docker Image*. Apr. 21, 2022. URL: https://hub.docker.com/r/kartoza/geoserver/.

[22] *Official Python Image*. May 9, 2022. URL: https://hub.docker.com/_/python/.

[23] *Python Flask*. Apr. 26, 2022. URL: https://flask.palletsprojects.com/en/2.1.x/.

[24] *Bootstrap 5*. May 9, 2022. URL: https://getbootstrap.com/docs/5.0/getting-started/introduction/.

[25] *Masonry*. May 9, 2022. URL: https://masonry.desandro.com/.

[26] *Official Alpine Linux Image*. May 9, 2022. URL: https://hub.docker.com/_/alpine/.

[27] *Official Ubuntu Linux Image*. May 9, 2022. URL: https://hub.docker.com/_/ubuntu/.

[28] *UbuntuGIS PPA*. May 9, 2022. URL: https://launchpad.net/~ubuntugis/+archive/ubuntu/ppa/.

[29] *PIP requirements file*. May 4, 2022. URL: https://pip.pypa.io/en/stable/reference/requirements-file-format/.

[30] *Github Actions*. Apr. 23, 2022. URL: https://docs.github.com/en/actions/learn-github-actions/.

[31] *Github Container Registry (GCR)*. Apr. 23, 2022. URL: https://docs.github.com/en/packages/working-with-a-github-packages-registry/working-with-the-container-registry/.

[32] *Docker Compose Env file*. Apr. 26, 2022. URL: https://docs.docker.com/compose/env-file/.

[33] *Docker Desktop*. Apr. 23, 2022. URL: https://www.docker.com/products/docker-desktop/.

[34] *OS GIS Sandbox Docker Compose File*. Apr. 23, 2022. URL: https://raw.githubusercontent.com/laiskasiili/os_gis_sandbox/main/docker-compose.yml/.