eMOTIONAL Cities

Mapping the cities through the senses of those who make them

DELIVERABLE 3.2

Architecture definition and code for the generic SDI

AUGUST 2022

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement n°945307. This document reflects only the author's view and the Commission is not responsible for any use that may be made of the information it contains.

Project Title	**eMOTIONAL Cities**
Deliverable	**D3.2**
Work package	**WP3**
Task	**Architecture definition and code for the generic SDI**
Number of pages	**14**
Dissemination level	**Public**
Leader	**ByteRoad**
Contributors and peer-reviewers	**DTU**
Date	**31/08/2022**
File name	**eMC_2022.08_D3.2_Architecture_definition_and_code_for_the_generic_SDI.pdf**
Version	**1.1**
Main authors	**Antonio Cerciello (ant@byteroad.net), Joana Simoes (jo@byteroad.net)**
Contributor	**Carlos Lima Azevedo (climaz@dtu.dk)**

Acronym List

API	Advanced Programming Interface
AWS	Amazon Web Services
ELK	ElasticSearch-Logstash-Kibana
FOSS	Free and Open Source Software
IAM	Identity Access and Management
JSON	JavaScript Object Notation
NoSQL	Not Only SQL
OWS	OGC Web Services
OGC	Open Geospatial Consortium
OSGeo	Open Source Geospatial Foundation
REST	Representational State Transfer
SOAP	Simple Object Access Protocol
SDI	Spatial Data Infrastructure
WCS	Web Coverage Service
WFS	Web Feature Service
WMS	Web Map Service
WMTS	Web Map Tile Service

Index

Acronym List 3

Summary **5**

Introduction **6**

Components of the SDI 6

pygeoapi 7

Elasticsearch 7

Geoserver 8

PostGIS 8

Architecture **10**

The OGC API Stack 10

The Legacy Stack 11

Source code **13**

How to build the SDI from source code 13

Docker Images 13

Zenodo 13

References **14**

Index of Figures

Figure 1 - Diagram of the docker-compose file… 10

Figure 2 - Architecture of the OGC API Stack 11

Figure 3 - Architecture of the Legacy Stack 12

Summary

The purpose of this document is to describe the architecture of the Spatial Data Infrastructure (SDI) of the eMOTIONAL Cities project and its components. Also, it lists the repositories where the source code is stored and instructions for building the software and setting up the infrastructure.

The solution is based on industry standards but was designed starting from the requirements described in the “Data Management Plan” (deliverable D3.1).

In the document "Description of the SDI" (deliverable D3.3) it will be detailed how the various use cases can be implemented with the technological solution described here.

1. Introduction

At the core of the eMOTIONAL Cities project lies a Spatial Data Infrastructure (SDI), which assembles datasets that characterise the emotional landscape and built environment in different Cities across Europe and the US. The SDI is a key tool not only to make the research data available within the project consortium but also to allow cross-fertilisation with other ongoing projects and, later on, to reach a broader public audience.

The design of the SDI is based on the data management plan (deliverable D3.1). Specifically, we tried to find tools that respond to the needs expressed by the project partners during the interviews/surveys that took place in the first months of the project.

For more than twenty years, SDIs have adopted the OGC OWS service interfaces (like WMS [1], WFS [2], CSW [3]), which are based on SOAP, the Simple Object Access Protocol. In recent years a new “family” of APIs has emerged within OGC, which is more aligned with modern web practices [4]. In this project, we leveraged the advantages of this new approach and compiled a stack to implement an SDI based on OGC APIs [5].

While it is technically possible to have a fully OGC API-based SDI, there are some practical reflections that we considered. First of all, the limited support from external tools. Many tools ensure excellent support for the previous generation of OGC standards (OGC Web Services aka OWS) and progressively improve support for the newest standards, but this still requires some time. Furthermore, the specifications of some OGC API standards needed for the project are still in draft. Hence, the inability to offer an OGC API solution completely based on officially approved standards.

To get around the problem, we added additional OGC API tools to allow support for OWS standards to circumvent the limitations and, at the same time, expand the number of standards supported. The rationale behind this choice is expanded in a published paper [6]. To structure the architecture description, the rest of the document refers to OGC API tools as *OGC API Stack*, while OWS tools as *Legacy Stack*. However, everything is part of a single infrastructure, offering different tools and support for multiple OGC standards.

In section 2, we describe the architecture of both stacks.

1.1 Components of the SDI

Once we selected the standards for the SDI, we set out to select our software stack from existing server-side implementations, favor choosing software from the Open Geospatial Foundation (OSGeo), as we would like to benefit from all the advantages of using Free and Open-Source Software (FOSS), including the ability to contribute changes back if deemed relevant. In the interest of simplicity, we would also favour software projects that implement more than one of the standards we selected. Below is a description of the main components of the eMOTIONAL Cities SDI.

All the data collected in the project will first be stored in the project's Data Lake, a cloud store accessible only to data owners and those responsible for the project's data management (WP3).

From a software point of view, the Data Lake is a private S3 Bucket, with a folder for each project participant, hosted in the Amazon AWS infrastructure. It is secured using the Identity Access and Management (IAM) system, which enables fine-grained permissions at the folder level. The Data lake is a versatile solution to facilitate the sharing of datasets, which may require further processing before being ingested into the actual storage of the SDI (Elasticsearch, PostGIS, FileSystem, etc ...)

Several pipelines, developed by categories of datasets, deal with transferring the data from this component to the *storage/data providers* directly used in the SDI.

1.1.1 pygeoapi

pygeoapi is a Python server implementing the OGC API suite of standards, which “emerged as part of the next generation OGC API efforts in 2018”. It implements several OGC APIs, including OGC API - Tiles and OGC API - Features, and it is certified OGC Compliant and an OGC Reference Implementation for OGC API - Features - Part 1: Core 1.0. Rather than being a facade for existing W*s implementations, pygeoapi implements all the OGC APIs from scratch, leveraging all the benefits of modern web practices. For instance, it makes use of HTTP verbs (e.g.: \textit{GET/PUT/POST/DELETE}) and codes (e.g.: 200, 201, 400, etc) and relies on content negotiation to retrieve the relevant media types. It also adopts JSON (JavaScript Object Notation) encodings, which are very popular among web developers

and a first-class citizen in RESTfull (REST- REpresentational State Transfer) web services.

In pygeoapi several “data providers” are supported for publishing vector data as OGC API - features. Elasticsearch, along with SensorThingsAPI, is the data provider which provides the full range of functionality, including support for DateTime, which is an important aspect of neuroscience data.

1.1.2 Elasticsearch

is a Lucene-based index server, with Full-Text capability, with support for distributed architectures. All the functionalities are natively exposed through the RESTful interface, while the information is managed as JSON documents. In the SDI, Elasticsearch is used as NoSQL storage for the OGC API stack and as the main data provider for pygeoapi.

ElasticSearch is part of the ELK stack [7], which includes Logstash, a server-side data processing pipeline that ingests data from multiple sources simultaneously, transforms it, and then sends it to Elasticsearch and Kibana, a tool to visualise ElasticSearch data with maps, charts, and graphs. By adopting ElasticSearch as a vector data provider for pygeoapi, we can also leverage the complete ELK stack to ingest and visualise our data assets.

1.1.3 Geoserver

The legacy stack is built around the free and open-source GeoServer, which implements industry-standard OGC protocols such as Web Feature Service (WFS), Web Map Service (WMS), and Web Coverage Service (WCS). Additional formats and publication options are available as extensions, including Web Processing Service (WPS) and Web Map Tile Service (WMTS).

Administrators can configure GeoServer to collect data from different sources, like filesystems and databases (spatial or not).

As part of the project, GeoServer will exclusively share data, which will be gathered in the Data Lake, the same used in the OGC API stack, and ingested in GeoServer’s stores by suitably configured data pipelines. A wide range of GIS clients can consume

data from GeoServer, including MapStore. MapStore is a highly modular Open Source WebGIS framework.

1.1.4 PostGIS

is an extension of the PostgreSQL relational database to enable GIS functionalities. It's the main storage for vector datasets in the Legacy stack.

2. Architecture

The Spatial Data Infrastructure is presented as a Service Oriented Architecture (SOA) composed of FOSS (Free and Open Source Software) components. These components are virtualized in docker containers and integrated via a docker-compose file (see figure 1).

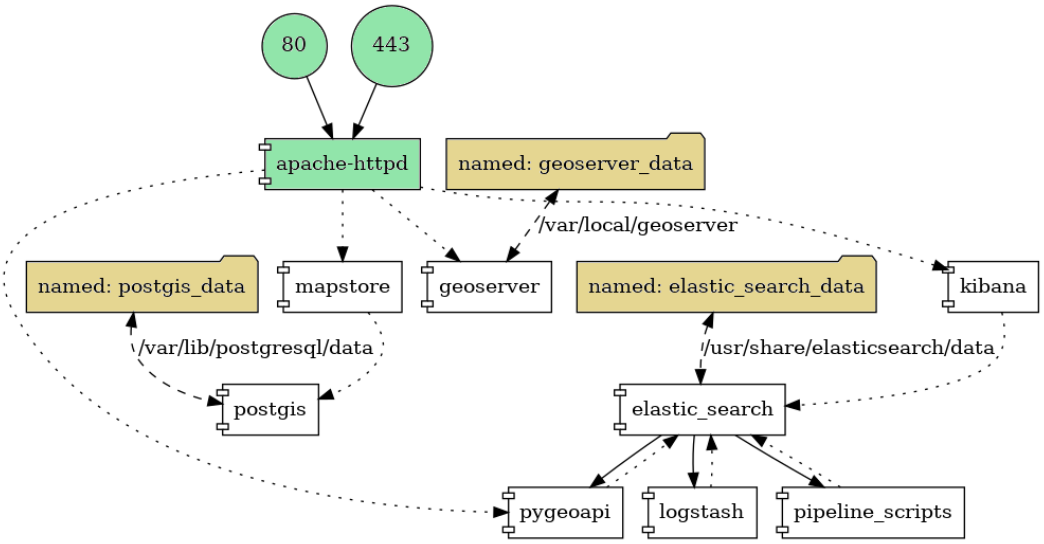

Figure 1 - Diagram of the docker-compose file, showing how the components fit together.

All the tools that are exposed to the web (pygeoapi, GeoServer, Mapstore and Kibana) are via HTTPS with the Apache HTTP web server.

All the data that is not static-configuration is persisted in named volumes defined in the docker-compose.

The following separation between the OGC stack and Legacy is mainly used for explanatory purposes because, from a technical point of view, the different components of the SDI are deployed together through the same docker-compose to allow the reuse of components such as the web server and facilitate the integration where possible.

The OGC API Stack

In this architecture, both data and metadata are ingested from the data lake (see figure 2). In the vector tiles pipeline, vector tiles are rendered using Tippecanoe and stored in MinIO (Tile repo), scalable storage compatible with AWS S3. The vector tiles are served from MinIO as OGC API - Tiles, using pygeoapi. In the other pipelines, both feature data and metadata are ingested in Elasticsearch and published as OGC API - Features and OGC API - Records. They can be consumed by any OGC API-aware clients, like QGIS, python OWSLib or ArcGIS. This architecture includes Kibana (see 1.1.3), a dashboard that displays the data from Elasticseach in different widgets, including a map.

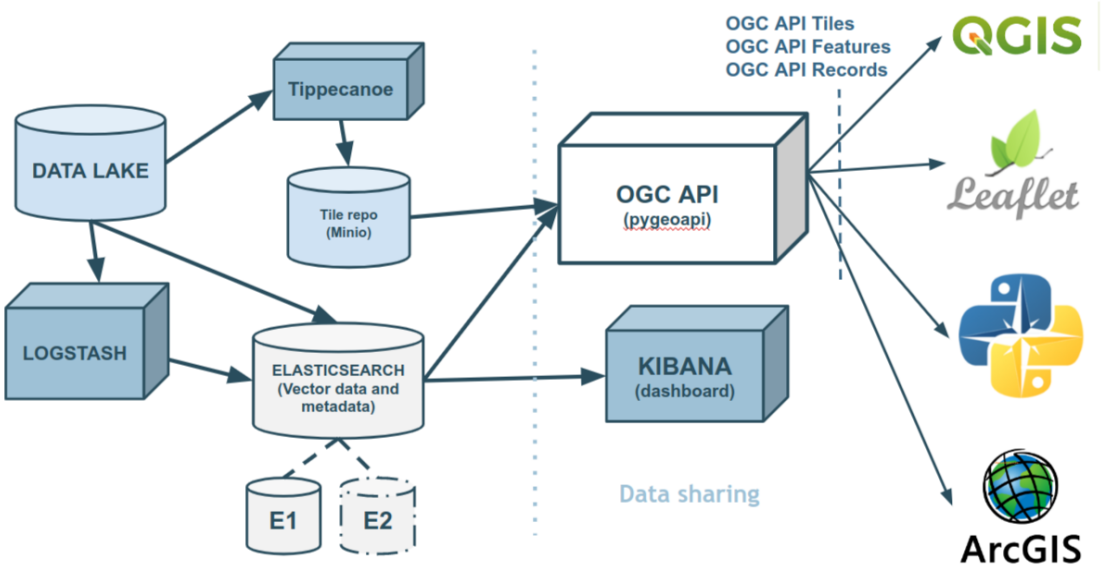

Figure 2 - Architecture of the OGC API Stack

The Legacy Stack

In this architecture (see Figure 3), the data is ingested from the data lake into a relational database (vector data) or the filesystem (raster data). Geoserver publishes the datasets using OWS standards (e.g., WMS, WMTS, WFS), which can be consumed by any OGC OWS-aware clients like QGIS, ArcGIS, OpenLayers, etc. In this architecture, we have included Mapstore, a WebGIS framework that enables the creation of maps based on datasets published using these standards. Instead of

exposing the data directly from Geoserver, we use a proxy. Geowebcache stores tiles as they are requested, eliminating redundant request processing and thus saving large amounts of processing time.

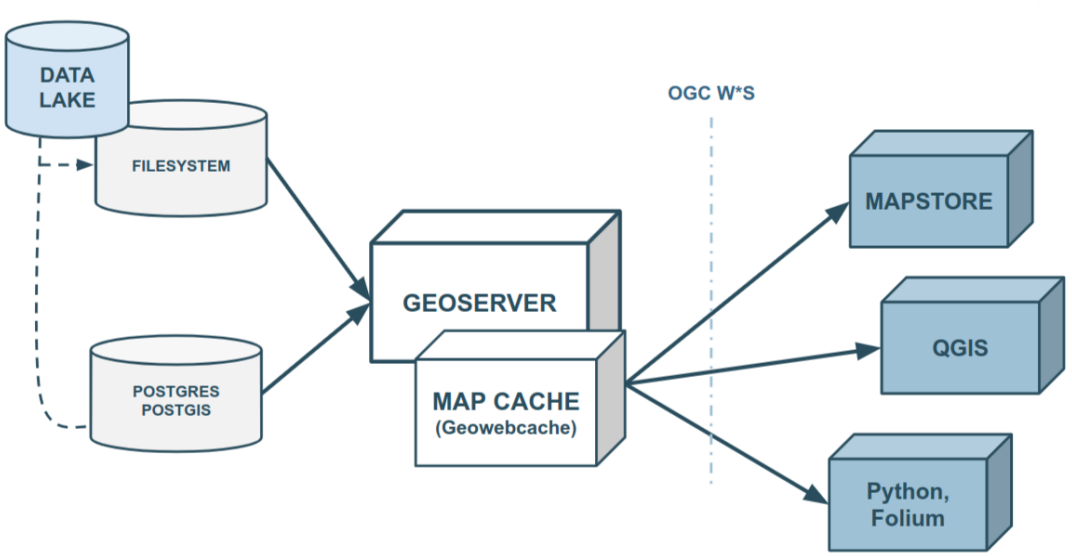

Figure 3 - Architecture of the Legacy Stack

3. Source code

All the code produced within the WP3 of the eMOTIONAL Cities project is stored in repositories under the emotional-cities organisation on Github [7].

The code of the SDI is published on the master branch of the *openapi-sdi* Github repository [10], along with instructions to start the SDI locally or on a production server. With docker composition, some test data will be automatically loaded into containers. The data reproduces some examples from the project. The code is publicly accessible and shared under a FOSS licence (e.g. MIT) [10].

3.1 How to build the SDI from source code

To build the SDI, please refer to the updated documentation on the project README [11].

3.2 Docker Images

Public docker images with the SDI software are available on Docker Hub, under the emotionalcities organization [12].

3.3 Zenodo

An initial project release was published on Zenodo: 10.5281/zenodo.6591180 [13].

4. References

[1] https://en.wikipedia.org/wiki/Web_Map_Service

[2] https://en.wikipedia.org/wiki/Web_Feature_Service

[3] https://en.wikipedia.org/wiki/Catalogue_Service_for_the_Web

[4] https://github.com/OAI/OpenAPI-Specification/blob/main/versions/3.0.2.md

[5] https://ogcapi.ogc.org/

[6] Simoes, J. and Cerciello, A., “Serving Geospatial Data Using Modern and Legacy Standards: a Case Study from the Urban Health Domain”, ISPRS - International Archives of the Photogrammetry, *Remote Sensing and Spatial Information Sciences*, vol. 48W1, pp. 419–425, 2022. doi:10.5194/isprs-archives-XLVIII-4-W1-2022-419-2022.

[7] https://www.elastic.co/what-is/elk-stack

[8] https://github.com/emotional-cities

[9] https://github.com/emotional-cities/openapi-sdi

[10] https://github.com/emotional-cities/openapi-sdi/blob/master/LICENSE

[11] https://github.com/emotional-cities/openapi-sdi#quick-setup

[12] https://hub.docker.com/orgs/emotionalcities

[13] https://zenodo.org/record/6620894#.YvLCvmHMJgE