



PUBLICAMUNDI
SCALABLE
REUSABLE
OPEN
GEOSPATIAL
DATA

REPORT FOR

DELIVERABLE D3.2

I INTRODUCTION

This report provides an overview of the technical characteristics and functionality of deliverable D3.2 “Scalable mapping”. Its purpose is to provide a short introduction to the software developed and present the major functionalities and improvements introduced by the PublicaMundi project.

The reader is encouraged to visit the software’s repository (<https://github.com/PublicaMundi>) to receive:

- Up-to-date versions of the software, along with documentation targeted to developers
- Detailed information regarding all development effort (commits, activity, issues)
- Instructions regarding the installation of the software and its dependencies



2 GEOSPATIAL ANALYTICS

2.1 DESCRIPTION OF TASK

The aim of this task was to provide granular geospatial analytics to CKAN that can shed light on the actual use of the geospatial services and data that is being accessed. A design document has been prepared to detail the functionality of this component and how the services will access and make full use of this analytics. In addition, a collection of analytics to be kept, analyzed, and presented to system administrators has been performed. Further, our review of current analytic services/software has highlighted the extremely limited offerings of usage analytics for web mapping frameworks. As a result, we developed the geospatial analytics component as a reusable software/service, suitable not only for PublicaMundi, but also for several other OGC-based services.

Collecting statistics about the usage of the offered services is an important aspect of the project, as it offers insights regarding possible optimizations. Specifically, efficient allocation of resources, needed for achieving a high degree of scalability, as well as internal optimizations of lower level components - such as the data processors in Publicamundi – benefit from an analytics module capable of identifying and maintaining information about the services that are accessed, as well as the access patterns on the targeted datasets.

2.2 DESCRIPTION OF WORK

The work done for implementing the analytics module consisted in building four components:

- A *parser component*, which reads periodically information about the incoming requests from the log files generated by the proxy module (HAProxy server). The parser offers a framework for programmatically defining methods of extracting relevant data. A set of methods for extracting currently relevant data (*accessed service, accessed datasets and the targeted area of the datasets*) has been already implemented, while further extraction methods can be easily implemented in the future, in case more information becomes relevant for optimizations (*e.g. semantic analysis of the operations applied on datasets could be done*). The parser passes the extracted information to the next component.



- An information maintenance component which receives new data from the parser and merges it with the previously acquired data. Each parsing step generates a new entry in a relational database table designated to store all statistics. At the same time, targeted aggregated views - one for each display widget and API component - of the information are kept separately, to provide a quick response to the requestors. For example, during a day several parsing steps are executed. For counting the number of coverages / layers that have been accessed during that day, a separate view (which is updated at each parsing step) is used.
- A web-based display for the harvested information, consisting graph-based widgets for displaying service access information over time, and map-based widgets, displaying the most accessed areas of the hosted datasets colored according to their respective access frequency. For example, Figure 1 shows a graph widget displaying the access counts of W*S services, Figure 2 shows a widget displaying the most accessed areas in the hosted datasets, Figure 3 shows a widget displaying the most accessed coverages/layers, with date filtering capabilities, while Figure 4 shows a widget displaying the access statistics to coverage c0001, by band.
- An external API, through which other services can access the information maintained by the analytics module. Methods are available for acquiring information about each individual service access, each individual dataset, as well as aggregated information by component (e.g. rasdaman or Geoserver).

Services access statistics

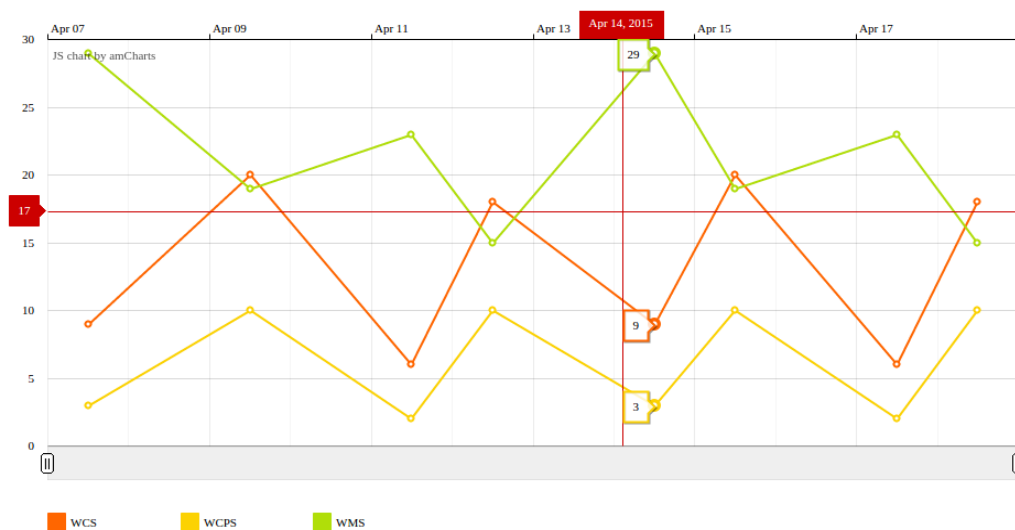


Figure 1: Access counts to W*S services, by date, displayed in a graph widget



Most requested areas

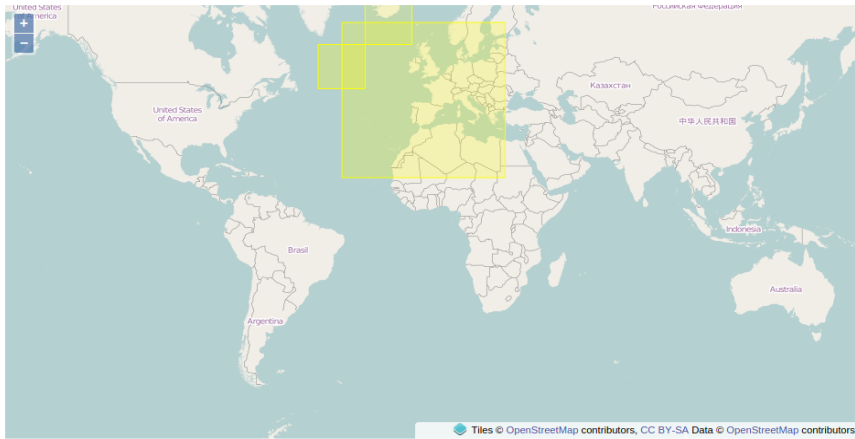


Figure 2: Map widget displaying the most accessed areas of the hosted datasets

Most requested coverages / layers

























Coverage Name	Access Count	Operations
c0001	46	 
poi	38	 
tiger	32	 
states	30	 
img_sample	8	 
mosaic	8	 
roads	8	 
tasmania_cities	8	 
tasmania_roads	8	 
archsites	6	 
bugsites	6	 
poly_landmarks	6	 

Figure 3: Widget displaying the most accessed coverages / layers, with filtering capabilities

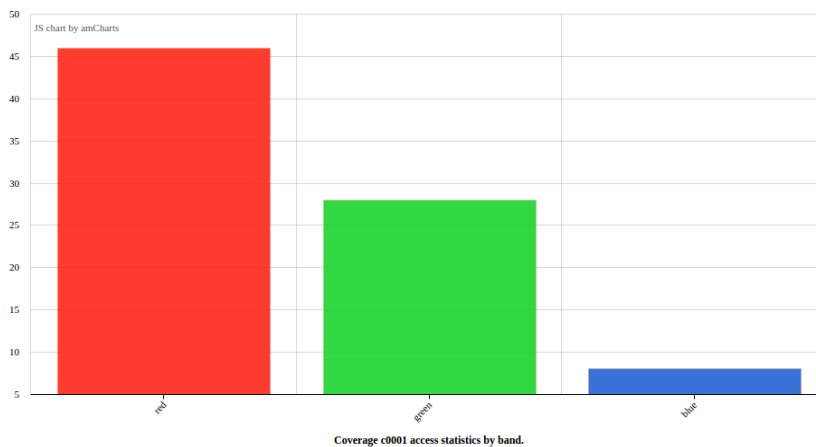


Figure 4: Widget displaying access statistics by band to coverage c0001



3 DEMAND-BASED MAP PROVISION

3.1 DESCRIPTION OF TASK

In this task we aimed at developing a JavaScript web mapping framework for scalable, on-demand provision of maps, which automatically takes into account user-demand, as monitored through the geospatial analytics component. In particular, we developed: a portal/client developer toolkit extending OpenLayers/Leaflet, automatic production of WMS/WFS/WCS service endpoints for geospatial data in the catalogue, automatic optimization techniques for map generation, dynamic adjustment of maps to client capabilities, and RESTful APIs for serving maps by multiple map/tile servers deployed in low cost virtual machines.

3.2 DESCRIPTION OF WORK

After extending of CKAN to store, publish and manage geospatial datasets (*through raster and vector storers - Tasks 2.2 and 3.1*), visualization of spatial datasets was a significant part of this task, structure in two main directions. First, the development of a unified map component, integrated into the CKAN catalog, to serve as a generic visualization tool for spatial datasets (*with enhanced discovery and download capabilities*). Second, the development of a Web Mapping API, supporting multiple platforms (desktop, mobile, tablet) in order to provide tools for open data re-use to developers, directly from the catalogue's API.

Another important sub-task was to address scalability and automatic deployment issues for the OGC services, as automatically exposed from the PublicaMundi catalog. For this purpose, we developed an automated system to deploy WMS/WFS/WCS service endpoints in a cloud environment and performed several scalability enhancements to the tiling strategies based on analytics.

The developed components are described in more detailed in the following sections. The Map Client application is now deployed on geodata.gov.gr and labs.geodata.gov.gr. All the new datasets that are published through geodata.gov.gr and are ingested from vector storer are automatically available through OGC services and from the Mapping API. The production system has the provision to manually spin off new virtual machines to handle increased user demand. At the same time, the raster server



(rasdaman) is optimized with tiling strategies based on the analytics module that gives us insight on the partitioning strategies needed for the data based on previous requests on the same coverage types. For vector data, the already supports dynamic caching of requested tiles with pre-fixed strategy (disk quotas per dataset per CRS assigned by system administrator).

3.2.1. PublicaMundi Map Component

The PublicaMundi Map Client component (Figure 5) is a web based application that augments the CKAN catalog with advanced mapping features not present in the current spatial extension. It is installed as a stand alone application and integrates tightly with any CKAN catalog installation through the CKAN action API. Moreover, the component provides extended support for the PublicaMundi Vector Storer and Data API by exploiting the extra metadata information and the query capabilities offered by the aforementioned extensions respectively.

The PublicaMundi Map Client allows users to browse, search and preview CKAN datasets and resources in a spatial oriented perspective. The main features of the map application include:

- Browsing spatial resources organized hierarchically by group, organization and package in a tree-like structure
- Browsing spatial resources organized hierarchically by organization, group and package in a tree-like structure
- Search the catalog for spatial resources by using text and spatial filtering (Figures 6 and 7)
- Dynamically load WMS metadata from resources and discover layers
- Preview one or more WMS layers, including layers from different datasets
- Controlling the z-ordering of overlapping layers
- Changing the opacity of each layer

In addition to data previewing, the component offers integration with the PublicaMundi Data API through a set of tools. The users, for example, can export spatial data that have been ingested to CKAN by the Vector Storer. The export process consists of two steps. Initially, the user draws a polygon for declaring an area of interest (Figure 8). Then the Data API is used for



selecting the data from all visible and queryable layers which overlap with the selected polygon.

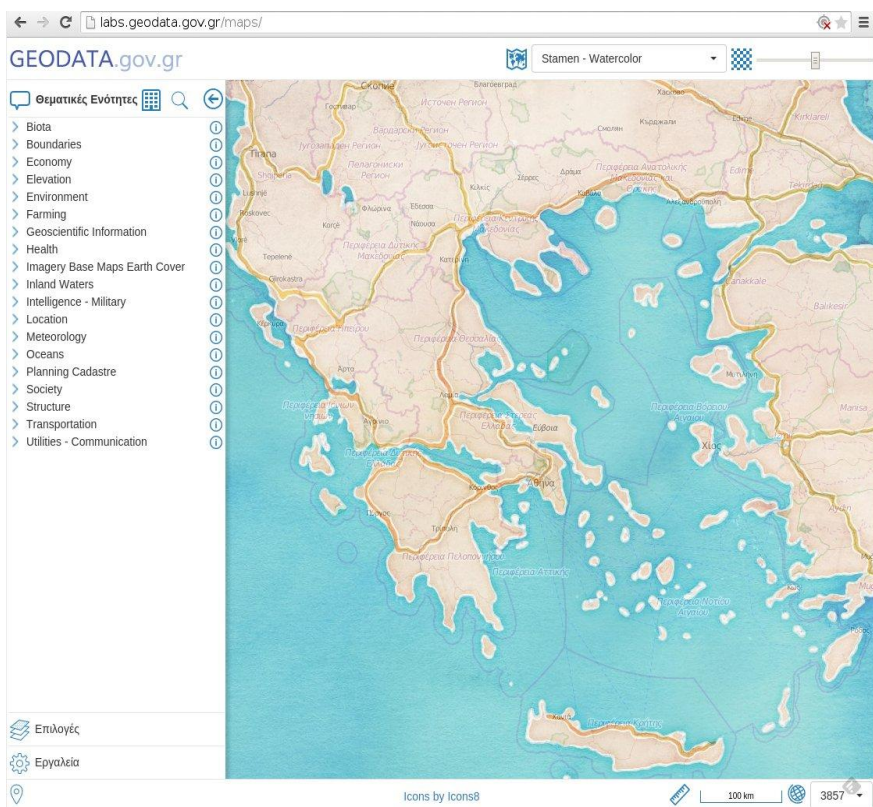


Figure 5: The PublicaMundi Map Client application. Grouping of datasets by topic categories

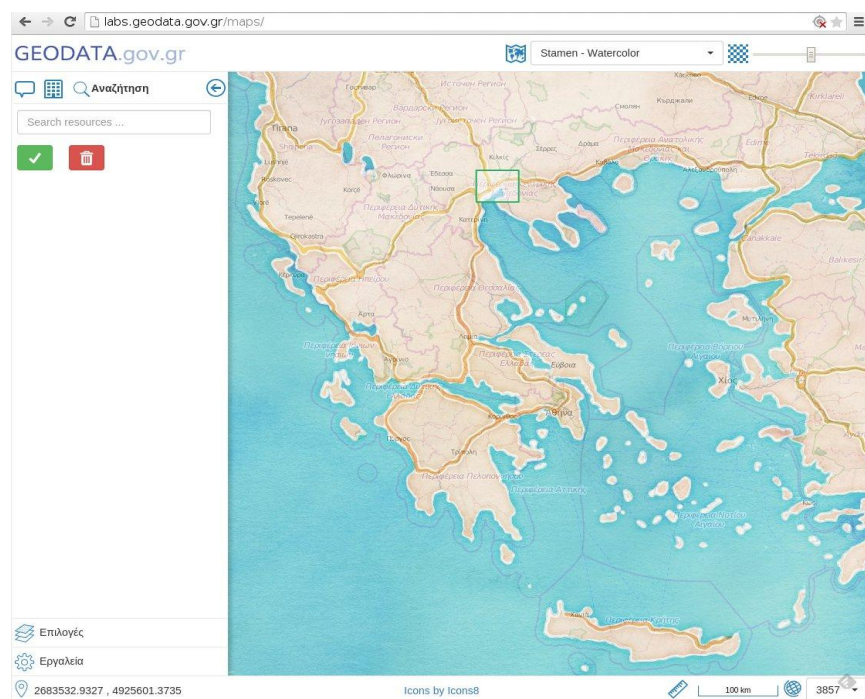


Figure 6: Discovery of datasets through spatial queries - Selection of area



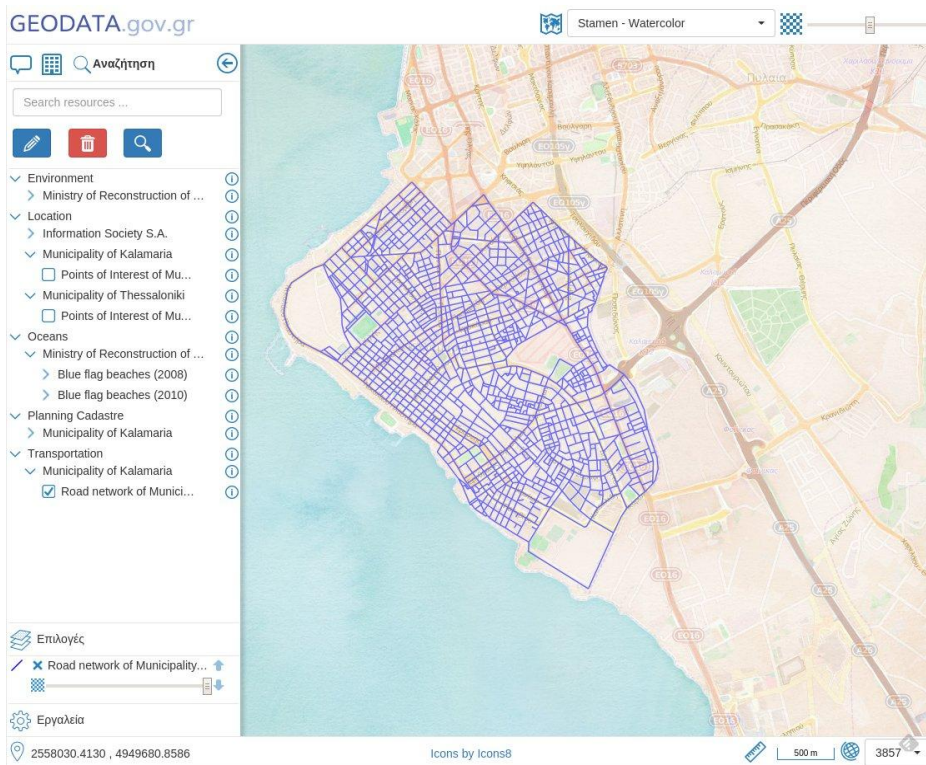


Figure 7: Discovery of datasets through spatial queries - Display of results

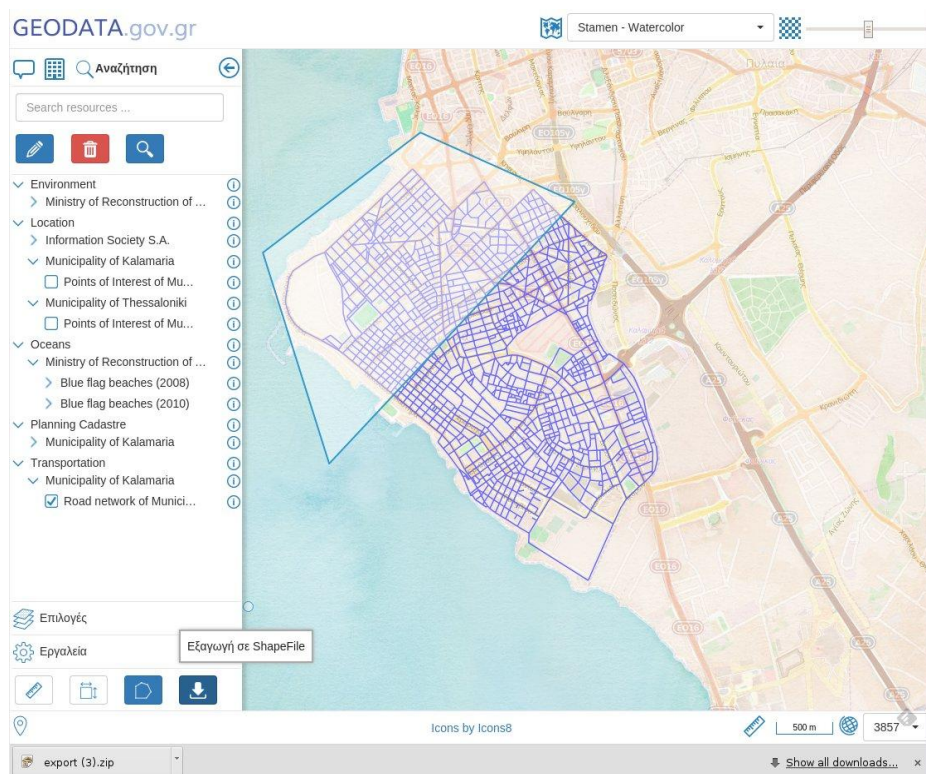


Figure 8: Download part of dataset through Data API integration

3.2.2. Mapping API

The PublicaMundi Mapping API (Figure 9) is a simple JavaScript API for enabling spatial data visualization and re-use of open data as provided by



the PublicaMundi catalog. Some of the key features implemented for the Mapping API include:

- Support for multiple platforms (desktop, mobile, tablet). The PublicaMundi API tries to detect client and load the appropriate framework, thus supports dynamic adjustment of maps to client capabilities.
- Supports dynamic framework loading e.g. by using browsing detection prefer leaflet for devices with small viewports and OpenLayers for desktop clients.
- Supports vector and raster data through OGC WMS, WFS, WCS services, TMS, map tile sources as well as file formats like GeoJSON and KML (Figures 10 and 11).
- Implements lightweight class wrappers for existing mapping frameworks (OpenLayers, Leaflet). It was chosen not to implement another mapping library but to build on existing ones. The goal was to achieve simplicity of implementation through normalization of method calls for most usable features both frameworks have to offer e.g. set/get center, set/get zoom, show/hide layer.
- Mapping framework independence. PublicaMundi Mapping API abstracts framework specific classes like Leaflet, OpenLayers. The new API gets best features from both frameworks e.g. using utility static methods like map (Leaflet) but also JSON inline configuration (OpenLayers)
- Supports CKAN DataStore API and allows users to preview spatial data directly in CKAN.

The Mapping API also supports per-layer styling in order to better visualize displayed layers. The supported styling variables are the same as in Leaflet for simplicity and the necessary transformations for compatibility with OpenLayers 3 are handled by the library.



GEODATA.gov.gr

Mapping API Demo

Leaflet OpenLayers

```

1 <!-- PublicaMundi API script -->
2 <script src="lib/PublicaMundi/build/publicamundi.js"
3 data-library="leaflet"></script>
4
5 PublicaMundi.noConflict();
6
7 //Popup handling with Bootstrap
8 var onFeatureClick = function(layer, features,
9 coordinate) {
10   if (features) {
11     feature = features [0];
12   }
13   if (popup) {
14     map.setOverlayPosition(popup, coordinate);
15   }
16   $(document.getElementById('popup')).popover('destroy');
17
18   var text;
19   if (feature['name']) {
20     text = feature['name'];
21   }
22   else if (layer._options.name ===
23     beaches._options.name) {
24     text = feature['PRELEV'];
25   }
26   else {
27     text = JSON.stringify(feature);
28   }
29 }

```

GEODATA.gov.gr

Mapping API Demo

Leaflet OpenLayers

```

1 <!-- PublicaMundi API script -->
2 <script src="lib/PublicaMundi/build/publicamundi.js"
3 data-library="ol"></script>
4
5 PublicaMundi.noConflict();
6
7 //Popup handling with Bootstrap
8 var onFeatureClick = function(layer, features,
9 coordinate) {
10   if (features) {
11     feature = features [0];
12   }
13   if (popup) {
14     map.setOverlayPosition(popup, coordinate);
15   }
16   $(document.getElementById('popup')).popover('destroy');
17
18   var text;
19   if (feature['name']) {
20     text = feature['name'];
21   }
22   else if (layer._options.name ===
23     beaches._options.name) {
24     text = feature['PRELEV'];
25   }
26   else {
27     text = JSON.stringify(feature);
28   }
29 }

```

Figure 9: Mapping API demonstration; similar results are shown for both Leaflet and OpenLayers JavaScript libraries

GEODATA.gov.gr Datasets Topics Organizations Maps API News About admin English

Organizations // landsat_thessaloniki (WMS)

landsat_thessaloniki (WMS) View in Maps Manage

URL: http://rasdaman.dev.publicamundi.eu:8080/rasdaman/ows/wms13?service=WMS&version=1.3.0&request=GetCapabilities

A WMS layer resource generated from landsat_thessaloniki

Resources

- landsat_thessaloniki
- landsat_thessaloniki ...
- landsat_thessaloniki ...

Additional Information

Last updated	December 13, 2014
Created	December 13, 2014
Format	wms

Figure 10: Support of WMS previews in CKAN through PublicaMundi Mapping API



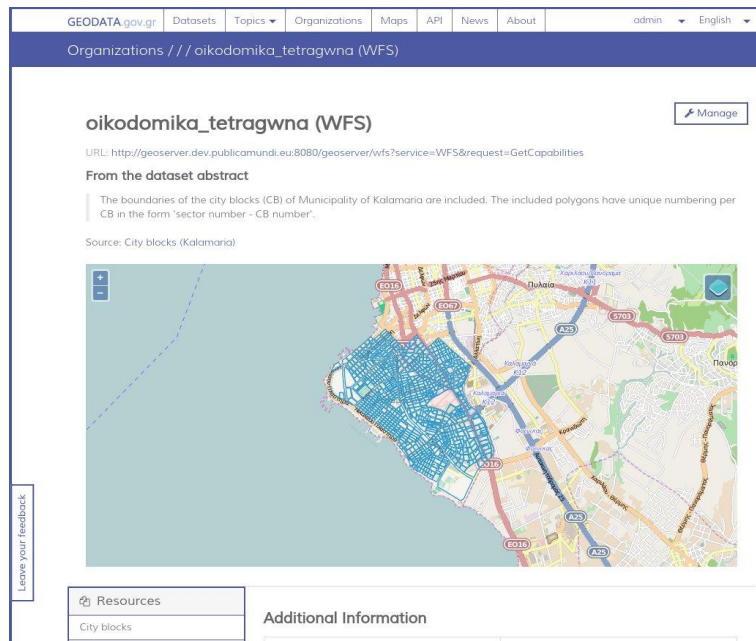


Figure 11: Support of WFS previews in CKAN through PublicaMundi Mapping API

3.2.3. Scalability and Optimization

As described in D1.2, PublicaMundi uses the Synnefo Cloud Infrastructure (and its reference implementation Okeanos) as an integration environment and management tool for Virtual Machines (VMs). As part of Task 3.3, several base virtual machines were designed and deployed based on Debian GNU/Linux operating system, with predefined configurations for OGC service endpoints. The configuration of such virtual machines included OSGeo servers like rasdaman, ZOO WPS, pycsw, GeoServer and MapProxy. All those services were pre-optimized for cloud performance. The virtual machines are available to the PublicaMundi system for automatic deployment of OGC web services. Once the system administrator needs extra CPU/RAM power, new virtual machines can be started on the cloud to handle the extra requests. The production software components of the production system were deployed into 8 virtual clusters, with the provision of spinning up more virtual machines into each cluster if necessary. Moreover, installation scripts have been developed, using the Ansible library, so that creation of base virtual machines is reproducible and extendable.

Besides provision for new virtual machines, and thus more servers available for the PublicaMundi system to scale, much development effort has focused to demand-based map provision, exploiting the statistics gathered by the analytics module, but also internal strategies of the map servers used in the system.



Specifically for the rasdaman system and raster data, several heuristic improvements were developed. One of these is the pyramid levels optimization where for each map layer a set of scale levels are precomputed and stored in the database allowing for instant retrieval time for clients requesting these layers at specific scale factors. As each WMS request is translated into a rasql query to retrieve the resulting data, several enhancements were made on the translation process, especially when applying styles over existing layers. The two are now always combined into a single query, eliminating the processing step done on the petascope level, thus enhancing the overall performance of the system. Furthermore, as the rasdman server manager was rewritten based on a new communication protocol, we can now support the creation and destruction of worker server processes on the fly, delivering a more scalable approach to resource allocation. Lastly, the rasdaman instances in the cloud are using a file based storage that can map directly on a network file storage, which is inherently scalable, avoiding the problems of optimizing relational databases to support data partitioning across nodes.

For the vector-based services (based on GeoServer), the MapProxy server has been deployed as a proxy/tiling service. The current configuration enables the automatic caching of all tiles for all datasets in production and supports the Greek Reference System (GGRS 87) as well as Web Mercator (EPSG:3857) and WGS84. Until now, the storage space of the production system can easily host all tiles for all datasets published, but in the future, optimizations based on gathered analytics will be applied. These optimizations will be location-based since MapProxy can apply fine-grained control of seeded/cached tiles using PostGIS or WKT geometries

