Architecture Concept

AUGUST 2014

DELIVERABLE

Project Acronym: **SDI4Apps**
Grant Agreement number: **621129**
Project Full Title: **Uptake of Open Geographic Information Through Innovative Services Based on Linked Data**

D3.1 ARCHITECTURE CONCEPT

Revision no. 07

Authors: Martin Kuba (Masaryk University)
Tomáš Sapák (Masaryk University)

Project co-funded by the European Commission within the ICT Policy Support Programme		
Dissemination Level		
P	Public	X
C	Confidential, only for members of the consortium and the Commission Services	

REVISION HISTORY

Revision	Date	Author	Organisation	Description
01	8/7/2014	Martin Kuba, Tomáš Sapák	MU	Initial draft
02	19/7/2014	Luděk Matyska	MU	Comments and corrections
03	22/7/2014	Tomáš Sapák	MU	Comments and corrections
04	24/7/2014	John O'Flaherty	MAC	Comments and corrections
05	25/7/2014	Peter Bednár	E-PRO	Comments and corrections
06	8/1/2014	Martin Tuchyňa	SAZP	Comments and corrections
07	8/8/2014	Tomáš Sapák	MU	Corrections

Statement of originality:

This deliverable contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation or both.

Disclaimer:

Views expressed in this document are those of the individuals, partners or the consortium and do not represent the opinion of the Community.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ACRONYMS

SaaS – Software as a Service
SDI – Spatial Data Infrastructure
SME – Small and Medium Enterprises
IaaS – Infrastructure as a Service
PaaS – Platform as a Service

EXECUTIVE SUMMARY

This document proposes an architecture suitable for spatial data infrastructure based on computational cloud.

The first chapter introduces computational cloud by providing its definition, followed by an overview of its three different service models demonstrated in examples, and provides an overview of the current state of the art in cloud infrastructures.

The next chapter analyses requirements for spatial data infrastructure. Requirements are based on six planned pilot applications described in Work Package 6 of the Description of Work document and also on deliverables and experiences of previous projects. Analysis identified 3 main use cases common to all spatial data processing – data collection, batch data transformation and data provision. To take advantage of cloud environment it is necessary to be able to scale spatial data storage. Analysis of storage scalability requirements showed, that data collection should scale writing, data provision should scale reading and data transformation should scale both reading and writing, but doesn't necessarily need to use the same storage for reading and writing. The reading part should also support PostGIS technology, because previous projects depend on it. Based on these analysis an architecture is proposed. The architecture is depicted in Figure 4.

The last chapter discusses implementation options for the proposed architecture.

1 INTRODUCTION

The main target of the SDI4Apps project is to build a cloud based framework with open API for geographic data integration, easy access and provision for further use. The aim of the framework is to bridge two separate worlds – the world of government-related top-down managed spatial data infrastructures (SDI), and the world of voluntary individuals, initiatives, and small and medium enterprises (SMEs) developing applications (Apps) based on geographic information – hence the name SDI-for-Apps.

The solution will be validated through six pilot applications that will be deployed on an instance of that framework. This document provides an architecture concept of the framework.

In this document, the idea of computational cloud is first precisely defined and illustrated by examples of its three different service models, followed by an overview of the current state of the art. Then, requirements from applications processing geographical data are analyzed. Based on these requirements, an architecture of the cloud based framework is proposed. The last part then discusses implementation options, which are important for practical deployment of the pilot applications, but from the architectural point of view, they are just implementation details.

Due to the chosen strategy[1] of the SDI4Apps project, this architectural concept predates any work on the pilot applications, and it will be iteratively refined and enhanced using feedback from developers of the pilot applications and users experimenting with them.

[1] described in the section B3.2a of the project's Description of Work document

2 CLOUD

2.1 Definition of Cloud Computing

Cloud computing is a general term for anything that involves delivering hosted services over the Internet.[2]

A detailed definition provided by NIST[3] in the document *The NIST Definition of Cloud Computing*[4] states:

Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.

The NIST definition continues with five essential characteristics: (i) on-demand self-service, (ii) broad network access, (iii) resource pooling, (iv) rapid elasticity and (v) measured service.

The NIST definition also gives the following three service models of cloud:

1. ***Software as a Service (SaaS)**: The capability provided to the consumer is to use the provider's applications running on a cloud infrastructure. The applications are accessible from various client devices through either a thin client interface, such as a web browser (e.g., web-based email), or a program interface. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited user-specific application configuration settings.*

2. ***Platform as a Service (PaaS)**: The capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages, libraries, services, and tools supported by the provider. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly configuration settings for the application-hosting environment.*

3. ***Infrastructure as a Service (IaaS)**: The capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, and deployed applications; and possibly limited control of select networking components (e.g., host firewalls).*

The NIST definition also explains cloud infrastructure as:

A cloud infrastructure is the collection of hardware and software that enables the five essential characteristics of cloud computing. The cloud infrastructure can be viewed as containing both a physical layer and an abstraction layer. The physical layer consists of the hardware resources that are necessary to support the cloud services being provided, and typically includes server, storage and network components. The abstraction layer consists of the software deployed across the physical layer, which manifests the essential cloud characteristics. Conceptually the abstraction layer sits above the physical layer.

2 http://searchcloudcomputing.techtarget.com/definition/cloud-computing

3 National Institute of Standards and Technology, U.S. Department of Commerce

4 http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

2.2 Types of Cloud Computing

Cloud computing has three rather different service models. To get a better understanding of the abstract concepts, we will illustrate them using some well-known examples.

2.2.1 Software as a Service

The Software-as-a-Service (SaaS) cloud service model is the one best known to computer users, because it is the only one that users directly use. It provides **device independence**, its resources can be accessed from any computer connected to the Internet, i.e. a PC, a notebook, a smartphone, a tablet, a smart TV, an e-book reader, an Internet kiosk etc. Some well-known examples of SaaS are:

- web mail – Gmail, Hotmail
- social networking and messaging – Facebook, Google+, Twitter
- on-line office suites – Google Docs, Microsoft Office 365
- file services – Dropbox, Google Drive, Microsoft OneDrive, ownCloud
- image libraries – Picasa, Flickr
- music libraries – Spotify, Google Play, iTunes
- video libraries – YouTube, Vimeo
- communication tools – Adobe Connect, WebEx
- business software – Salesforce, NetSuite
- web content management – Google Sites, Wordpress

SaaS is sometimes referred to as "on-demand software" or "application service providing (ASP)". The SaaS provider provides the software in the form of downloadable executable code which runs in e.g. JavaScript in user's web browser or natively on user's operating system (MS Windows, Linux, OS X, Android, iOS, etc.). The client code calls some platform-independent API (Application Programming Interface) on the provider's servers. Such an API is often based on the HTTP protocol and sometimes it can be called by third party applications. A nice example of such an open API is the Google API[5] which allows third parties to develop applications that can use the functionality provided by Google servers.

2.2.2 Platform as a Sevice

A platform is a software environment used to develop and run applications[6]. The Platform-as-a-Service (PaaS) cloud service model is not visible to end users, it is targeted to application developers and maintainers who want to develop and deliver their SaaS applications.

Some well-known examples are:

- Google App Engine (provides PHP, Python, Java, Go)
- Amazon Elastic Beanstalk (provides Ruby, PHP, Python, .NET, Java, JavaScript)
- Heroku (provides Ruby, PHP, Python, Java, JavaScript, Perl)
- Microsoft Azure Websites (provides PHP, Python, .NET, JavaScript)
- Red Hat OpenShift (provides Ruby, Python, PHP, JavaScript, Perl, Java, Haskell, .NET)

The platforms usually provide a set of programming languages with their standard libraries and a selection of popular frameworks written in those languages, like Django for Python, Ruby on Rails for Ruby, or Node.js for JavaScript. Additionally they provide some means for storing data, usually relational databases such as MySQL, PostgreSQL, Oracle, MS SQL, and some non-SQL databases optimized for high scalability such as Google's BigTable or Amazon's DynamoDB.

5 http://en.wikipedia.org/wiki/Google_APIs
6 http://explainingcomputers.com/cloud.html

2.2.3 Infrastructure as a Service

The Infrastructure-as-a-Service (IaaS) service model provides a virtual data center. Its users do not have to build their own air-conditioned fire-protected computer rooms containing racks filled with computers connected by cables and network equipment, with electricity supplied by uninterruptible power sources (UPS).

An IaaS provider provides so called **virtual machines** (VMs) that provide a complete system platform supporting the execution of a complete operating system. Many VMs can be hosted on a single physical machine running **hypervisor** software (popular choices are KVM, Xen, VMware).

IaaS provider also usually provides:

- **disk images** with preinstalled popular operating systems (various versions of Linux, MS-Windows)
- **networking services** - virtual local area networks, virtual private networks, IP addresses, firewalls, load balancers, domain name service (DNS)
- **storage services** - virtual block storage, file storage, object storage, relational database storage, no-SQL storage, tape archive storage, content delivery network (CDN), triplestore

Large IaaS providers distribute their physical resources into separate geographically dispersed locations, so that disasters such as flooding or earthquakes can disable only a part of their infrastructure.

IaaS consumers create and destroy VMs on demand, either manually or by setting up scaling rules. Such rules can state for example that a new VM should be started when existing VMs are loaded on more than 95%.

Some well-known **public IaaS clouds** are:

- Amazon Elastic Compute Cloud
- Google Compute Engine
- Microsoft Azure
- Rackspace Cloud Servers

It is also possible to create a private cloud using one of many available **software tools**, some of them are:

- OpenNebula
- OpenStack
- Eucalyptus
- VMware vCloud Suite

Resources hired from an IaaS cloud can be either used directly, for example for an on-demand movie rendering, or as a layer under a PaaS or SaaS cloud. For example, the SaaS cloud file hosting service Dropbox operated by Dropbox Inc., runs on top of the Amazon's IaaS cloud, thus a SaaS cloud operated by one company can be run on top of an IaaS cloud operated by another company.

2.2.4 Cloud Types Summary

The cloud has three different service models:

- **Software-as-a-Service** model provides on-demand access to software, either as downloadable code executed on client computers, or through remote API calls to code executed on servers
- **Platform-as-a-Service** model provides on-demand software environment for deploying applications. The environment includes concrete programming languages, their specific libraries, and additional services like SQL and no-SQL storage. PaaS cloud is usually used as a layer under SaaS cloud services.
- **Infrastructure-as-a-Service** model provides on-demand resources from a virtual data center. The resources can be used directly or as a layer under PaaS or SaaS cloud services.

Sometimes the division line is blurry, for example a software service may be used as a part of a platform, or a relational database storage can be provided either as a part of an infrastructure or as a part of a platform.

2.3 State of the Art in Cloud Infrastructures

In this section we provide a description of selected cloud infrastructure providers.

2.3.1 Amazon Cloud

Amazon operates the oldest public cloud, with perhaps the most developed set of features. The cloud provides both IaaS and PaaS services.

The IaaS services provide virtual machines (called EC2 – Elastic Compute Cloud) run in Xen hypervisors, with virtual block devices used as disks (called EBS – Elastic Block Storage), and virtual network services: virtual local area network (called VPC – Virtual Private Cloud), virtual http load balancers (called ELB – Elastic Load Balancer), and virtual firewalls.

The amount of used resources can be governed manually or by setting up scaling rules based on any measured metrics (CPU load, network load, etc.), the rule based service is called Auto Scaling.

The IaaS services also include object file storage (called S3 – Simple Storage Service), magnetic tape archive (called Glacier), relational databases (called RDS – Relational Database Service), and NoSQL key-value database (called DynamoDB).

Amazon offers PaaS services (named Elastic Beanstalk) where the platform is suitable for general web applications. It provides runtimes for several programming languages (Ruby, PHP, Python, .NET, Java, JavaScript).

2.3.2 Google cloud

Google cloud is significantly younger than Amazon cloud, and it provides less features. It also provides both IaaS and PaaS services.

The IaaS services provide virtual machines (called Compute Engine) run in KVM hypervisors, however it does not provide virtual disks, it uses persistent disk images which are copied to the physical machine on VM start and copied back on VM termination.

Google also provides virtual local networks and firewalls. For load balancing, there are 3 options. Google recommends installing an http proxy acting as load balancer into a VM, it also has HTTP load balancing service in Limited Preview as of July 2014, and network load balancing service in General Availability.

For auto scaling, Google does not provide an out-of-the-box solution like Amazon, instead they recommend writing an own application that monitors the used resources and uses Google REST API to start and stop VM instances as needed.

The IaaS services also include object file storage (called Cloud Storage), relational databases (called Cloud SQL), NoSQL row-column-timestamp-value database (called BigTable). Google does not provide a magnetic tape archive.

Google offers PaaS services (named App Engine) where the platform is suitable for highly scaling web applications which has to be designed specifically for the App Engine[7]. Several projects[8] [9] are trying to provide an open source reimplementation of the Google App Engine to avoid vendor lock-in.

There is a significant difference between the Google PaaS offering and other PaaS offerings. The Google App Engine provides more infrastructure to make it easy to write scalable applications, but can only run a limited range of applications designed for that infrastructure. While other services allow users to install and configure nearly any *NIX[10] compatible software, the App Engine requires developers to use only its supported languages, APIs, and frameworks. Existing web applications that require a relational database will not run on an App Engine without modification.

7 http://en.wikipedia.org/wiki/Google_App_Engine#Major_differences

8 http://en.wikipedia.org/wiki/AppScale

9 https://code.google.com/p/typhoonae/

10 http://en.wikipedia.org/wiki/Unix-like

2.3.3 CERIT-SC Cloud

Masaryk University, one of the SDI4Apps participants, operates a private IaaS cloud through its CERIT-SC department in collaboration with CESNET, the association of Czech public universities and the Czech Academy of Sciences. The cloud is available for employees and students of all Czech academic and research institutions.

The cloud is based on **OpenNebula** cloud management software, it uses Xen and KVM hypervisors. It provides disk images with preinstalled Debian Linux, CentOS, SciLinux, and MS-Windows. OpenNebula allows both ways of working with disk images – virtual block devices (like Amazon) and persistent disk images copied on VM start and termination (like Google). For performance reasons, only the copied persistent disks are used in the CERIT-SC cloud now.

OpenNebula supports creation of virtual local networks like Amazon and Google. CERIT-SC cloud does not provide firewalls and load balancers out-of-the-box, they are left to be implemented by the user.

For auto scaling, OpenNebula offers a component named OneFlow which allows setting up scaling policies based on metrics data from monitoring or time schedule.

For object file storage, an S3-compatible service called Cumulus is provided. No relational nor NoSQL databases are provided, but they can be installed by a user into VMs. A tape archive is provided in the form of HSM (Hierarchical Storage Management) where the lowest layer in the hierarchy is implemented by magnetic tapes, and higher layers are implemented using massive arrays of idle disks (MAIDs) and RAID disk arrays.

The CERIT-SC cloud uses a pool of physical machines with the following specifications (as of summer 2014):

- 32 machines with 8 CPUs and 128GB RAM each
- 48 machines with 12 CPUs and 90GB RAM each
- 112 machines with 16 CPUs and 128GB RAM each
- 10 machines with 24 CPUs and 96GB RAM each

In total, machines with 2864 CPUs are collected in the cloud. Should the need arise, more CPUs can be added from CESNET's MetaCentrum computing infrastructure collecting theoretically up to 9640 CPUs in total[11] However these resources are not sitting idle waiting to be used, rather they are used for high performance computing most of the time, so experiments requiring a high number of CPUs must be planned beforehand.

The CERIT-SC cloud infrastructure can be used only for research purposes. Thus it can be used to develop and test the SDI infrastructure, but it cannot be used for production runs of the SDI infrastructure. Hence the developed SDI infrastructure should be independent of the concrete cloud implementation, so that it can be deployed to Amazon, Google or any other public cloud, or on a cloud build specifically for the SDI purposes with appropriate funding.

2.3.4 FI-WARE - Generic and Specific Enablers

The SDI4Apps project's description of work document talks about generic and specific enablers. The terms come from the FI-WARE project[12] (the letters FI mean Future Internet). FI-WARE is an open cloud-based infrastructure[13] provided both in the form of working cloud instance (IaaS, PaaS and SaaS) and in the form of installable packages to enable companies to build their own cloud instance. In the FI-WARE project **generic enablers** are defined as:

> *elements which offer reusable and commonly shared functions serving multiple areas of use across various sectors*

A slightly more concrete explanation is given by IBM[14] as:

11 http://metavo.metacentrum.cz/pbsmon2/nodes/physical

12 http://www.fi-ware.org/developers-entrepreneurs/

13 http://www.fi-ware.org/

14 http://www.research.ibm.com/haifa/projects/services/fi-ware/index.shtml

> *elements which offer reusable and commonly shared functions that make it easier to develop Future Internet Applications in multiple sectors*

But even this definition is still rather vague.

The idea of reusable software parts reappears in computer science again and again since its beginning in 1950s, it was implemented (in historical order) as

- **subroutines**[15] in machine code and assembly languages
- **functions** and **procedures** in structured programming[16] languages
- **objects** in object-oriented programming[17] languages
- **components** in component-based software engineering[18]
- **services** in service-oriented architectures[19]

The reusable parts variants differ in their long-term maintainability, but they always have an **interface** and sometimes they have also an **internal state** which can be changed through the interface. The interface is represented by (locally or remotely) callable subprograms (functions, procedures, methods) which may take some arguments as input and may return some result values. The internal state is kept in computer memory or in a persistent data storage, typically in a filesystem or some kind of database.

Given the vague definition of Generic Enablers (GEs), nearly any reusable software part can match this definition, be it a library of functions in a programming language, or a hosted service accessible over a network.

However the FI-WARE vision document[20] explains that the enablers are identified as functions shared across multiple usage areas, dividing enablers into two classes:

- **generic enablers** are able to serve in multiple usage areas
- **domain-specific common enablers** are common to multiple applications in a very limited set of usage areas

Key goals of the FI-WARE project are the identification and specification of GEs, together with the development of reference implementations of identified GEs.

Please note that neither the term *usage area*, nor *future internet* are defined in the FI-WARE vision document, furthermore the definitions of generic enablers and future internet applications are somewhat circular - a generic enabler should be identified as functions common to multiple future internet applications, but future internet applications are defined as applications based on GEs.

Our interpretation of FI-WARE is that the project tried to identify, through interviewing users and developers of Internet applications, what functions they now think they might need in the future. The thoughts were transformed into specifications of packages of functions, and reference implementations of these specifications were developed. The packages of functions were divided into general ones that may be beneficial to any application, called **generic enablers**, and more specialized ones suitable only for a specific application domain, called **specific enablers**.

In another words, the FI-WARE project tried to specify a platform-as-a-service cloud suitable to any future Internet application, with optional extensions specific to some applications.

The project has identified 62 enablers[21]. They are divided into 6 chapters[22]:

15 http://en.wikipedia.org/wiki/Subroutine
16 http://en.wikipedia.org/wiki/Structured_programming
17 http://en.wikipedia.org/wiki/Object-oriented_programming
18 http://en.wikipedia.org/wiki/Component-based_software_engineering
19 http://en.wikipedia.org/wiki/Service-oriented_architecture
20 http://forge.fi-ware.org/plugins/mediawiki/wiki/fiware/index.php/Overall_FI-WARE_Vision
21 http://catalogue.fi-ware.org/enablers
22 http://www.fi-ware.org/developers-entrepreneurs/

- **Cloud Hosting** – the fundamental layer which provides the computation, storage and network resources in which services are provisioned and managed.
- **Data/Context Management** – the facilities for effective accessing, processing, and analyzing massive data volumes, transforming them into valuable knowledge available to applications.
- **Architecture of Applications / Services Ecosystem and Delivery Framework** – the infrastructure to create, publish, manage and consume services across their life cycle, addressing all technical and business aspects.
- **Internet of Things (IoT) Services Enablement** – the bridge where FI services interface and leverage the ubiquity of heterogeneous, resource-constrained devices in the Internet of Things.
- **Interface to Networks and Devices** – open interfaces to networks and devices, providing the connectivity needs of services delivered across the platform.
- **Security** – the mechanisms which ensure that the delivery and usage of services is trustworthy and meets security and privacy requirements.

The Cloud Hosting chapter defines an IaaS cloud. Its core is the **DataCenter Resource Management generic enabler** (DCRM GE)[23] which extends the open-source OpenStack cloud management software with some additional features like VM high availability or support for placement policies. Its schema is depicted in Figure 1. Additional GEs interact with it as depicted in Figure 2 to form the Cloud Hosting part.

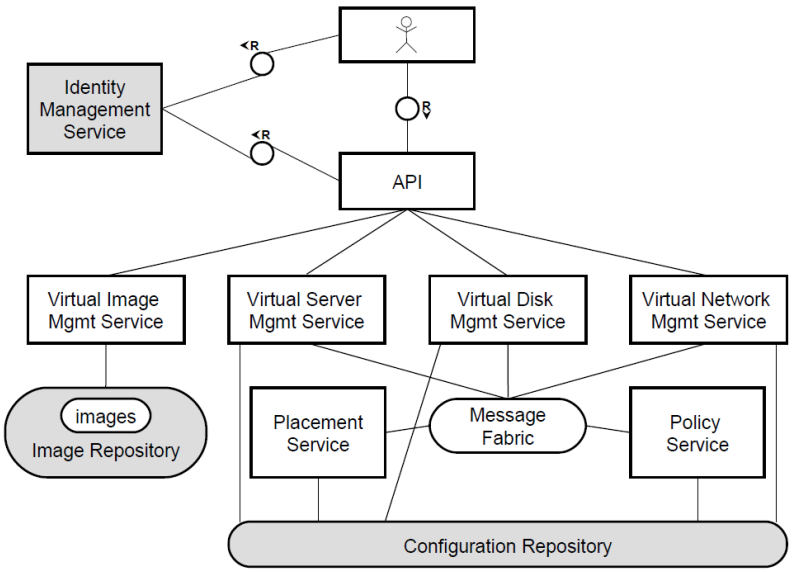

Figure 1: FI-WARE DataCenter Resource Management GE

[23] https://forge.fi-ware.org/plugins/mediawiki/wiki/fiware/index.php/FIWARE.ArchitectureDescription.Cloud.DCRM

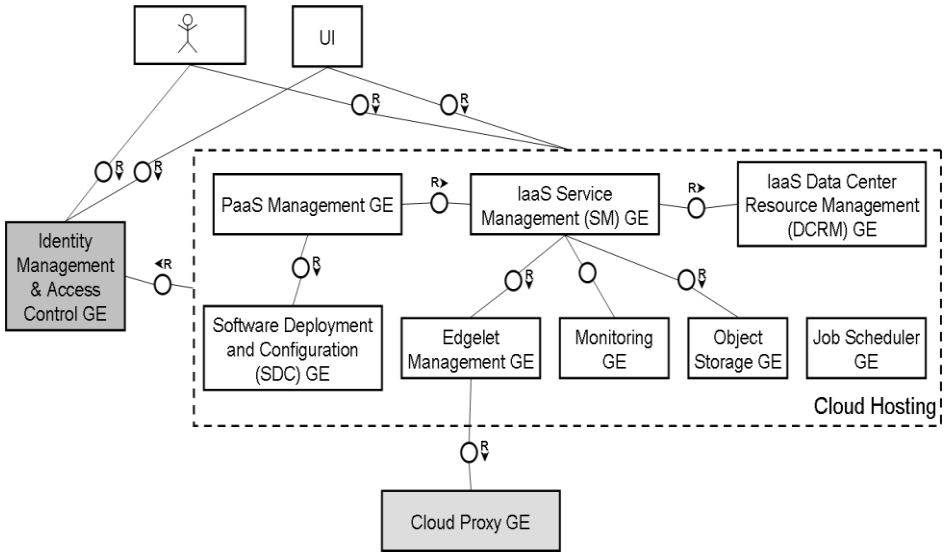

Figure 2: FI-WARE Cloud Hosting Architecture

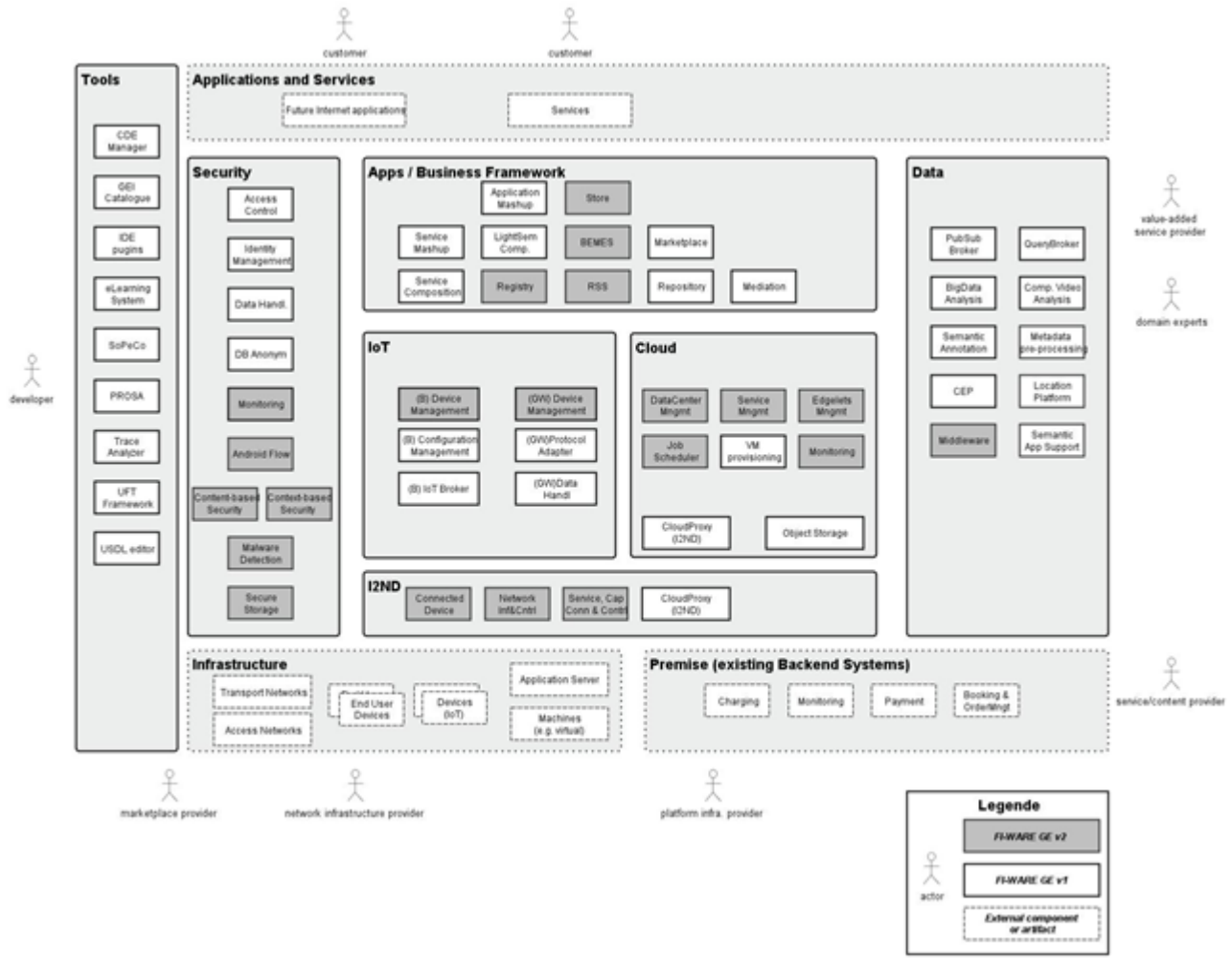

Figure 3: FI-WARE platform with all chapters

The Figure 3 shows the whole FI-WARE platform with all chapters. The images are from the FI-WARE wiki[24].

24 https://forge.fi-ware.org/plugins/mediawiki/wiki/fiware/index.php/FI-WARE_Architecture

The article [1] by Berre et al. in its Chapter 4 identifies the enablers from the Cloud Hosting chapter as IaaS, while the other chapters are related to PaaS and SaaS.

3 CLOUD INFRASTRUCTURE FOR SPATIAL DATA

3.1 Requirements Analysis

The requirements for the planned cloud-based spatial data infrastructure can come from two sources:

- the planned internal pilot applications, as they are described in the Description of Work document for the Work Package 6
- from deliverables and experiences of previous projects

There are six planned internal pilot applications:

1. Easy Data Access
2. Open Smart Tourist Data
3. Open Sensor Network
4. Open Land Use Map Through VGI
5. Open INSPIRE4Youth
6. Ecosystem Services Evaluation

The corresponding tasks 6.1 – 6.6 in WP6 are planned to start in month 12, so as of the time of this writing only the high-level descriptions from the Description of Work document are available, no finer details.

The experiences of the previous projects are more helpful. An analysis provided the following information:

- all spatial data processing can be always assigned into one of 3 main use cases:
  - **data collection** – using crawlers, sensors, crowdsourcing
  - **batch data transformation**
  - **data provision** which can be further divided into
    - **provision of transformed data without further processing**
    - **answering queries** - interactive data transformation
- the **requirements for scalability** of the three data use cases (collection, batch transformation, provision) **are not related** – all combinations are possible, e.g. an easy data collection may be followed by a very computationally intensive transformation and that may be followed by provision of data to a moderate, but not negligible, number of simultaneous users
- the first two use cases and provision of data without processing are not time-constrained, however the use case of **answering queries is limited by the time** a user is willing to wait for an answer to his or her query. There can be two types of scaling to reduce the time:
  - **scaling to more concurrent users**
  - **scaling to process more data in a single query**
- many existing spatial data software tools rely on the **PostGIS** extension of the PostgreSQL relational database management system[25], which adds new data types (geometry, geography, raster and others), functions, operators and index enhancements that apply to these spatial types[26]. Thus to ease the transition of the current generation of spatial data applications to the cloud architecture, this must be taken into account.

A cross-check of these findings based on the previous projects with the descriptions of the planned pilot applications confirmed that the requirements of the pilot applications are in line with these findings.

25 http://en.wikipedia.org/wiki/PostGIS#Users
26 http://postgis.net/features

3.1.1 Spatial Data Storage Scalability Requirements Analysis

The data collection can be performed in one of two ways – push or pull. **Data pull** would be performed by downloaders or crawlers copying data from outside sources. **Data push** would be performed by **hardware sensors** pushing their measured data into a data storage, or by humans creating data using **crowdsourcing** which can be considered as just using human sensors instead of hardware sensors.

In either way, the requirements for scalability of **data collection** are only for concurrent **data writing**.

During **data provision** no spatial data are written, so the scalability requirements are just for concurrent **data reading**.

During the batch data transformation, raw collected data are read, and transformed data are written, thus both scalability for **data reading** and **data writing** are required, however the transformed data do not have to be written to the same data storage system from where raw collected data are read, so it is possible to **scale reading and writing independently** by using two separate spatial data storage systems, one for reading and the other for writing.

During the interactive data transformation performed when answering queries, scaling to more concurrent users requires just scaling of **data reading**, as in that case where data are not written back to the spatial data storage, thus this case is the same as for the provision of transformed data without further processing.

The most difficult case is interactive data transformation that needs to process more data in a single query. This case cannot be solved by simple horizontal scaling (adding more computers), because there is only a single query, and it cannot be solved by vertical scaling (using more powerful computers) because there is a low limit on vertical scaling. The only solution seems to be a radical application redesign, perhaps using the map-reduce technique which maps parts of a single processing onto many computers and then reduces their partial results into a single final result. But this depends on the particular query and cannot be solved in a general case.

Summary:

- data collection requires scalable data writing
- data provision requires scalable data reading
- batch data transformation requires scalable data reading and scalable data writing, but the source and destination storages may be separate
- the scalable storage should be PostGIS -compatible, at least on the reading side

3.1.2 Cloud Service Model Analysis

The previous chapter 2 has defined and described the three cloud service models (IaaS, PaaS and SaaS).

The SDI4Apps infrastructure aims to provide SaaS for services that will not be hosted directly on the infrastructure and PaaS for applications and services that will be hosted directly on it.

The PaaS platform will be tailored to spatial data applications, in this regard it will match the definition of specific enablers as they were discussed in chapter 2.3.4.

Both the SaaS and PaaS services need to be layered on top of IaaS services. While the SaaS and PaaS layers will be specific for spatial data infrastructure (SDI), the IaaS layer should be independent. As discussed in chapter2.3.3, the Masaryk University's cloud cannot be used for production runs, only for research and development, so the created SDI4Apps platform should be deployable to another implementations of IaaS clouds with different APIs, such as Amazon, Google or instances of software stacks like FI-WARE, OpenNebula, OpenStack, VMware vCloud.

To achieve this **separation from specific cloud implementation**, a layer wrapping of the underlying IaaS layer should be created. Such a wrapper could be reimplemented for any IaaS cloud API that the SDI4Aps platform would be subsequently deployed on.

The wrapper may be based on a proposed standard such as OCCI (Open Cloud Computing Interface)[27] or CIMI (Cloud Infrastructure Management Interface)[28], or it may be a specific interface with implementations for OCCI, CIMI, Amazon AWS etc. The wrapper would play a similar role as JDBC plays in the world of relational databases by providing a unified interface for applications that are shielded from differences among SQL database implementations.

The OCCI standard proposed by Open Grid Forum seems very promising in this regard, as it provides a unified API for managing IaaS clouds, and it has already implementations for clouds with Amazon AWS, OpenNebula and OpenStack APIs, and implementations for Google Compute Engine, Microsoft Azure and VMware are planned for near future.

3.2 SDI Architecture Proposal

The requirements analyzed in the previous section 3.1 can be summarized as

- there are 3 main phases of data processing – collection, batch transformation and provision – with independent scaling requirements
- scalable concurrent reading and writing are needed, but the source and destination storages may be separate, and at least the reading side should be compatible with PostGIS

Given these requirements, we propose the architecture depicted in Figure 4. It provides **3 PaaS clouds** targeted to data collection, data transformation and data provision respectively, and a **scalable spatial data storage**, which should have a part that scales for writing, and a part that scales for reading. The reading part would be PostGIS-compatible.

27 http://en.wikipedia.org/wiki/Open_Cloud_Computing_Interface
28 http://cloud-standards.org/

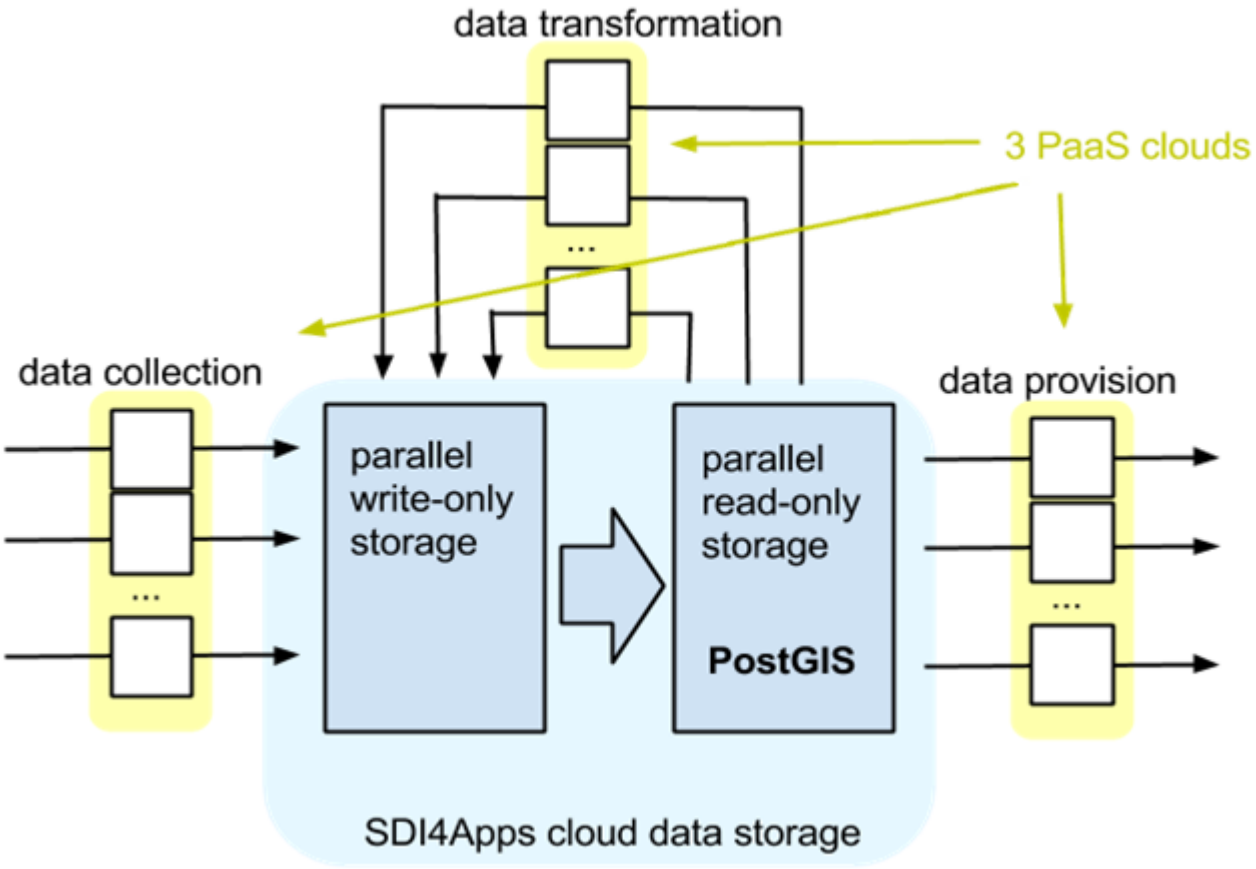

Figure 4: SDI4Apps platform architecture diagram

3.2.1 Scalable Spatial Data Storage

The scalable spatial data storage may be implemented as a single scalable database system, or it can be implemented as two separate systems, where data written to the write-only part are transferred after some delay to the read-only part.

We expect the use of clustered Postgres XC or Postgres XL databases with PostGIS extension, however if that solution appears not to scale well, the write-only part may be implemented using other technologies than relational databases, trading their properties like transactional integrity for higher scalability.

3.2.2 Platform Provided by the PaaS Clouds

The 3 PaaS clouds would provide 3 platforms tailored for the 3 data uses cases – collection, transformation and provision. They may be implemented using a single platform providing features needed for all of them, or they may be implemented separately.

The PaaS will provide API (Application Programming Interface) for the following areas:

- workflow management (e.g. data transformation after collection, incremental updates)
- management of degree of parallelism (how many workers should be allocated to a task, using rules or direct API calls)
- events (finished transformation, partial update of collected data, etc.)

The current best practice is to base APIs on the REST[29] (Representational State Transfer) architectural style.

[29] http://en.wikipedia.org/wiki/REST

The platform will also provide services specific to spatial data applications. The list of these services will be assembled later from the requirements of the pilot applications.

3.2.3 Identified Enablers

The SDI4Apps platform will need generic enablers representing functions provided by a standard IaaS cloud, i.e. VM management and disk storage management. These generic enablers should be independent of the underlying IaaS platform as explained in the section 3.1.2.

The platform will also need security, authentication, authorization, and user rights management of some form. The current most advanced techniques for authentication are based on federated identity, where users reuse their account in some system for accessing other systems even operated by independent providers. The current most advanced techniques for authorization are embodied in the OAuth standard for authorization which enables users to delegate precisely defined rights for working with their data located on cloud servers to third-party applications. The precise security mechanisms will be identified later from requirements of the pilot applications.

For supporting spatial data applications, some specific enablers will be needed. The precise functions needed by spatial data applications will be identified later from the requirements of the pilot applications.

4 IMPLEMENTATION OPTIONS

4.1 PostgreSQL clustering techniques

The proposed SDI4Apps platform should be able to accommodate large numbers of sensors (both hardware and human) on one side and large numbers of consumers on the other. We learned from previous projects, that one of their foundations is the PostgreSQL database with PostGIS extension support. While with current number of users its performance of single instance of database on single virtual machine is satisfactory, with increasing numbers of sensors and consumers, database performance could be a bottleneck for the whole system.

That's where the cloud comes in handy. If we were able to cluster the database, we would be able to take advantage of the key feature of cloud platform – its ability to provision vast numbers of resources on demand.

There are many SQL database implementations and most of the major ones have spatial data support (PostgreSQL PostGIS, Oracle Spatial, MS SQL Server, MySQL …). These spatial extensions are not mutually compatible. SDI4Apps should follow up other projects that use PostGIS. Rewriting all code from old projects would be costly, so we reduced our research to PostgreSQL clustered solutions that supports PostGIS extension.

Native PostgreSQL supports two types of clustering:

- hot standby – asynchronous replication, standbys are read-only
- warm standby – used for HA[30], standbys cannot be queried

Native PostgreSQL could be used for consumers - read operations could be load balanced on top of standbys and thus scale well, on the other hand this solution is not very useful for concurrent writing, because writes can only go to a single node.

Table 1: Third party clustering solutions

Name	Read scaling	Write scaling	Official PostGIS support
Bucardo	yes	yes	no
Pgpool II	yes	no	no
PL/Proxy	yes	yes	no
Postgres-XC	yes	yes	no
Postgres-XL	yes	yes	yes
PostgresForest	yes	yes	no
SkyTools	yes	no	no
Slony	yes	no	no
Stado	yes	yes	yes
Tungsten	yes	no	no

There are only two solutions that supports PostGIS out of the box. We plan to use Postgres-XL, which is more actively developed, but it could be replaced by another solution, if it doesn't prove to be suitable for the SDI4Apps platform.

30 http://en.wikipedia.org/wiki/High_availability

4.1.1 Postgres-XL

Postgres-XL is an open source SQL database solution closely based on Postgres-XC. Postgres-XC key features[31]:

- Write-scalable PostgreSQL cluster – it was observed 3x speedup with five nodes compared with native PostgreSQL
- Synchronous multi-master compared with asynchronous master-slave native PostgreSQL
- Table location transparent – no change in transaction handling
- Tables can be replicated among all nodes for read scalability, or distributed for write scalability
- Same API to Apps as native PostgreSQL
- PostGIS is not supported natively. Postrgres-XC sources need to be patched before compilation.

Postgres-XL is a very fresh project – its first RC was released on 14.5.2014. Key features compared with Postgres-XC[32] are:

- MPP query support
- Performance improvements
- Multi-tenant security
- PostGIS support

31 https://wiki.postgresql.org/wiki/Postgres-XC
32 http://www.translattice.com/postgres_compare.shtml

5 CONCLUSION

The architecture concept described in this document targets an infrastructure for applications delivering geographic data integration, easy access and provision for further use. The concept will be further refined and enhanced using feedback during the development, deployment and user experimentation with the six pilot applications.

REFERENCES

[1] BERRE, Arne J.; USLÄNDER, Thomas; SCHADE, Sven. Identification and Specification of Generic and Specific Enablers of the Future Internet–Illustrated by the Geospatial and Environmental Domain. In: *Towards a Service-Based Internet*. Springer Berlin Heidelberg, 2011. p. 278-289.