RESEARCH ARTICLE

Storm Clouds Platform: a cloud computing platform for smart city applications

Marco Battarra, Marco Consonni*, Samuele De Domenico and Andrea Milani

Hewlett Packard Italiana S.R.L., 9, Via Di Vittorio Giuseppe – 20063 Cernusco Sul Naviglio, Italy

Abstract: This paper describes our work on STORM CLOUDS[1], a project with the main objective of migrating smart-city services, that Public Authorities (PAs) currently provided using traditional Information Technology, to a cloud-based environment. Our organization was in charge of finding the technical solutions, so we designed and implemented a cloud computing solution called **Storm Clouds Platform** (SCP), for that purpose. In principle, the applications we ported could run on a public-cloud service, like Amazon Web Services™[2] or Microsoft® Azure[3], that provide computational resources on a pay-per-use paradigm. However, these solutions have disadvantages due to their proprietary nature: vendor lock-in is one of the issues but other serious problems are related to the lack of full control on how data and applications are processed in the cloud. As an example, when using a public cloud, the users of the cloud services have very little control on the location where applications run and data are stored, if there is any. This is identified as one of the most important obstacles in cloud computing adoption, particularly in applications manage personal data and the application provider has legal obligation of preserving end user privacy[4]. This paper explains how we faced the problem and the solutions we found. We designed a cloud computing platform — completely based on open-software components — that can be used for either implementing private clouds or for porting applications to public clouds.

Keywords: smart city, cloud computing, infrastructure as service, OpenStack

*Correspondence to: Marco Consonni, Hewlett Packard Italiana S.R.L., 9, Via Di Vittorio Giuseppe – 20063 Cernusco Sul Naviglio, Italy; Email: marco.consonni@hpe.com

Received: February 3, 2016; **Accepted:** April 12, 2016; **Published Online:** May 9, 2016

Citation: Battarra M, Consonni M, De Domenico S, *et al*. 2016, Storm Clouds Platform: a cloud computing platform for smart city applications. *Journal of Smart Cities*, vol.2(1): 14–25. http://dx.doi.org/10.18063/JSC.2016.01.003.

1. Introduction

Cloud computing is a delivery model for technology enabled services that drives greater agility, speed and cost savings. It provides on-demand access, via a network, to an elastic pool of shared computing resources (e.g., services, applications, frameworks, platforms, servers, storage, and networks) that can be rapidly provisioned and released with minimal service provider interaction and scaled as needed according to a pay-per-use paradigm.

We recently participated in STORM CLOUDS, a project partially funded by the European Commission within the 7th Framework Program (Grant Agreement No. 621089) with the main objective of deploying smart-cities application services to a cloud-based environment. In the project, we were in charge of designing and implementing the technical solutions taking into account some general but fundamental requirements:

- the tools used to port and/or deploy the applications should be based on open source technology in order to avoid vendor lock-in issues,
- the solution was required to support different deployment models ranging from situations in which the applications run on data-centers owned

by PAs, to scenarios in which they run on a public cloud, made available by some cloud service provider,
- the solution should support the implementation of a catalogue of easily deployable applications: the idea was that any ported application could be reused by PAs not directly participating in the project.

This article briefly describes the experience gained while working on the project, illustrating the problem and the solution found. Storm Clouds Platform (SCP) was the platform we implemented and this article briefly describes the functions it implements and the architecture.

2. The Problem

In order to find a solution for the generic problem of "porting applications to cloud", first we tried to understand the problem better, focusing on two aspects: the project context and the applications to port.

2.1 Project Context

We started by identifying the "entities" (e.g., organizations, users, other systems, etc.) participating in the project and/or interacting with applications to port. Figure 1 summarizes what we found out.

As reported[1], we needed a 'digital space' for running application services and, as required by the cloud computing paradigm, the resources for running them should be activated on-demand. We called this digital space Storm Clouds Platform (SCP).

Analyzing the applications, we found that they could inter-operate with each-other and/or with External Digital Services in order to achieve the required functionality. For instance, several applications used Google Maps[5], a service available on Internet through a web service interface, for rendering maps on HTML pages. Citizens and/or public servants were identified as the End Users of the applications while Public Administrations (e.g., municipalities) were the Application Providers, responsible for providing services to citizens. Application Creators were the organizations in charge of implementing the applications. We found that usually Independent Software Vendors (ISVs) cover this role but, in some cases, the Application Provider is also the Application Creator (e.g., a municipality might also be the creator of the application). Generally, it was important to distinguish these two roles because, while application creators are usually interested in the technical details for porting applications to a cloud environment, application providers give more importance to aspects like the cost of the solution, the economic benefits, whether data are processed according to regulations (e.g., if and how security/privacy problems are addressed), etc.

In the project, we played the role of the Platform Provider, in charge of designing, implementing and

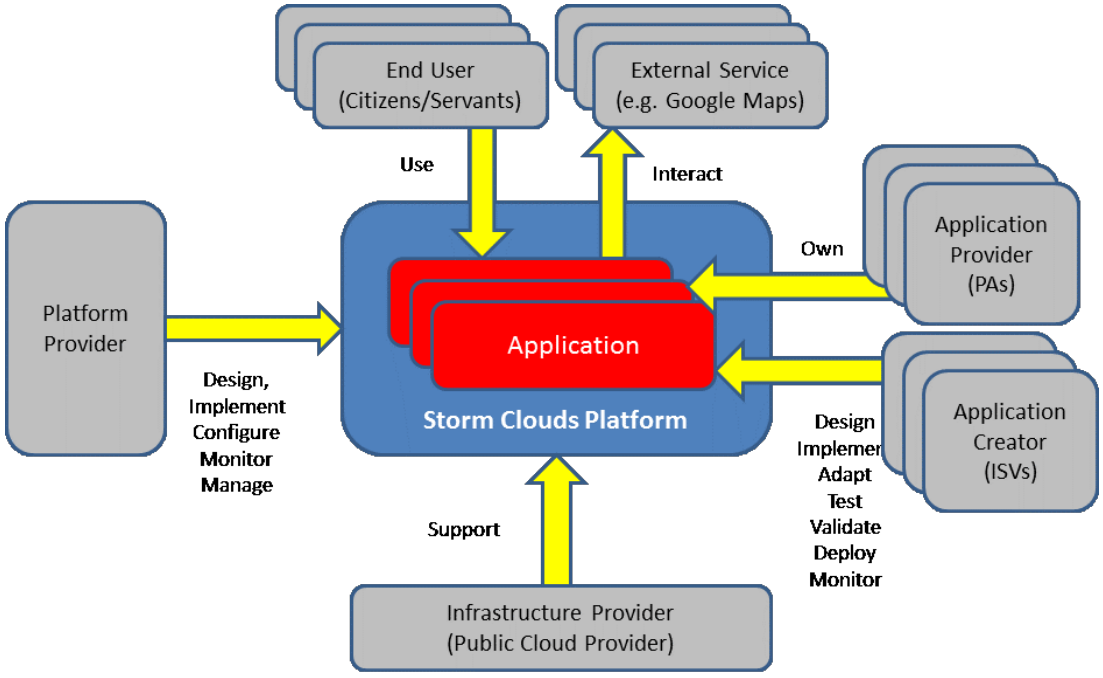

Figure 1. STORM CLOUDS project context.

operating the SCP. We decided to engage an Infrastructure Provider for supplying the physical resources (e.g., physical servers, disk storage and network connectivity) for hosting the platform and the applications. The distinction between these two roles, in addition to being useful for the project because it was not necessary to make an up-front investment for the physical resources, was also aligned with the general requirement of providing a solution that could be deployed on-premises (i.e., on the municipality's data centre), off-premises (i.e., a data centre of a third party) or even using services of a public cloud operator. In fact, after the project termination, PAs could decide to implement the solution at their own data centres, use a data centre of a third party or even deploy the applications on a public cloud.

2.2 Application Services

The applications were selected from a set of completely implemented smart-city services that the partners developed in previous projects. We conducted an assessment for gathering information that served as requirements for the platform. We asked the application proponents to give technical details on their applications in order to characterize the kind of workload we should support. Table 1 summarizes the results of the assessment.

Most of the applications were implemented using open source products at different levels (e.g., Linux operating system; MySQL/MariaDB or PostgreSQL databases; PHP, Python, Java and JavaScript programming languages, etc.), whilst few were based

Table 1. Application assessment results

Application	Operating System					Language								Database					
	Ubuntu	CentOS	Linux (*)	OpenWrt	Windows	Javascript	PHP	Java	Perl	C#	Python	PL/SQL	Ruby	PostgreSQL	MySQL	SQLite	MSSQL	Mongo	Oracle
Blue Parking Valladolid		1			1	1		1			1	1						1	1
City Navigator											1								
Co-Labora			1		1		1								1				
Crowd Dunding													1	1					
Emissão Plantas de Localização	1					1	1	1						1					
Honolulu Answers	1												1	1					
HotSpot CMA	1			1		1	1								1				
Ideol – Innobarómetro	1					1	1	1			1			1	1				
Improve My City	1	1	1			1	1								1				
LimeSurvey		1	1			1	1		1					1			1		
LocalGIS	1	1			1	1		1						1					
LocalWiki	1	1									1			1					
OpenCivic	1	1	1			1	1								1				
OpenTripPlanner								1											
Participação Pública (PPGIS)	1	1	1			1								1					
Plano Director Municipal	1					1	1	1						1					
SEDOC	1						1	1		1					1	1			
Sense the City	1	1	1			1	1								1				
SIMPLEXT	1							1											
Urbanismo en Red - UeR	1							1						1					
Virtual City Marketplace	1	1	1			1	1								1				
Virtual City Tour	1	1	1			1	1								1				
Vive Valladolid	1	1	1			1	1								1				
We The People Petitions	1					1	1								1			1	
Total	**18**	**11**	**9**	**1**	**3**	**15**	**14**	**9**	**1**	**1**	**4**	**1**	**2**	**10**	**11**	**1**	**1**	**2**	**1**

Note: (*) Linux stands for any Linux distribution.

on proprietary technologies such as Oracle DBMS, Windows Server 2008, etc. The steering committee of the project decided to port only applications based on open source software packages, in order to be sure that the act of "porting the application to cloud" did not constitute an infringement of the licensing rights when the application was activated.

During the assessment we also found out that some applications managed sensitive information like, for example, citizens' personal data. This was a very important aspect with implications both on the applications to migrate during the project and the solution we designed. In fact, the Application Providers had the legal responsibility of preserving citizens' privacy[4] and — although during the project some activities were addressed to verify the application security — they preferred to migrate services that did not manage sensitive information. Application Providers were not in the position of fully controlling the way data was managed because, as described above, all the applications were to run on servers made available by the Infrastructure Provider, an organization external to the project. However, the Application Owners required that the solution could be hosted on any data centre so that, after the project termination, there would have the possibility of creating a Storm Clouds Platform instance at their own premise. As a result, we had to fulfill the requirement of defining a solution that could be implemented both on-premises and off-premises.

2.3 High-level Requirements

The analysis of the project context and the results of the application assessment can be briefly summarized in a list of high-level requirements.

- **Open Source Technology**: the solution should be implemented using open source software packages for avoiding vendor lock-in issues and for controlling the cost of the porting. This was also in accordance with the indications of the European Commission (EC) that "*supports Free/Open Source Software (FOSS) as a development model since it is a very effective way to collaboratively develop software with fast take-up and improvement cycles*" [6].
- **LAMP Workload**: the solution should support and facilitate the porting of LAMP applications. According to a broadly accepted definition[7], "*The acronym LAMP refers to first letters of the four components of a solution stack, composed entirely of free and open-source software, suitable for building high-availability heavy-duty dynamic web sites*". The meaning of the LAMP acronym depends on which specific components are used as parts of the actual bundle. In the case of the project, "L" stands for Linux; "A" for Apache HTTP Web Server; "M" identifies MySQL, MariaDB or MongoDB, the database management system; "P" is for PHP or Python, the programming languages used for dynamic web pages and web development[1].
- **Deployment Models**: the solution should support various cloud computing deployment models like *Private Cloud* in which "*the cloud infrastructure is provisioned for exclusive use by a single organization*"[8] (this is the case in which a municipality has its own data centre, *Community Cloud* in which "*the cloud infrastructure is provisioned for exclusive use by a specific community of consumers from organizations that have shared concerns*"[8] (when, for example, all the municipalities use a single cloud provided at the National level) and *Public Cloud* in which "*the cloud infrastructure is provisioned for open use by the general public*"[8].
- **Management**: the solution should implement tools for managing the applications and the components of the solution itself. In this area, the requirement was directed to tools for administering, monitoring and automating the deployment of the services.

3. The Solution

Figure 2 shows the logical architecture of the Storm Clouds Platform (SCP), i.e., the solution we designed for the project.

SCP is a layered architecture in which the Infrastructure as a Service Layer (IaaS Layer) works as the foundation of the whole solution. IaaS Layer provides basic IT capabilities as compute services (e.g., Virtual Machines), storage services (e.g., Virtual Volumes) and networking services (e.g., Virtual Networks) for the implementation of the upper layers.

Definitely the IaaS layer should be run on some physical resources, such as servers, disks and network equipment that are represented by the Hardware Layer. This layer is implemented by the hardware of the data

[1] As described in the following pages, the architecture we designed also supports PostgreSQL database management system and other programming languages like Java, Perl and Ruby.

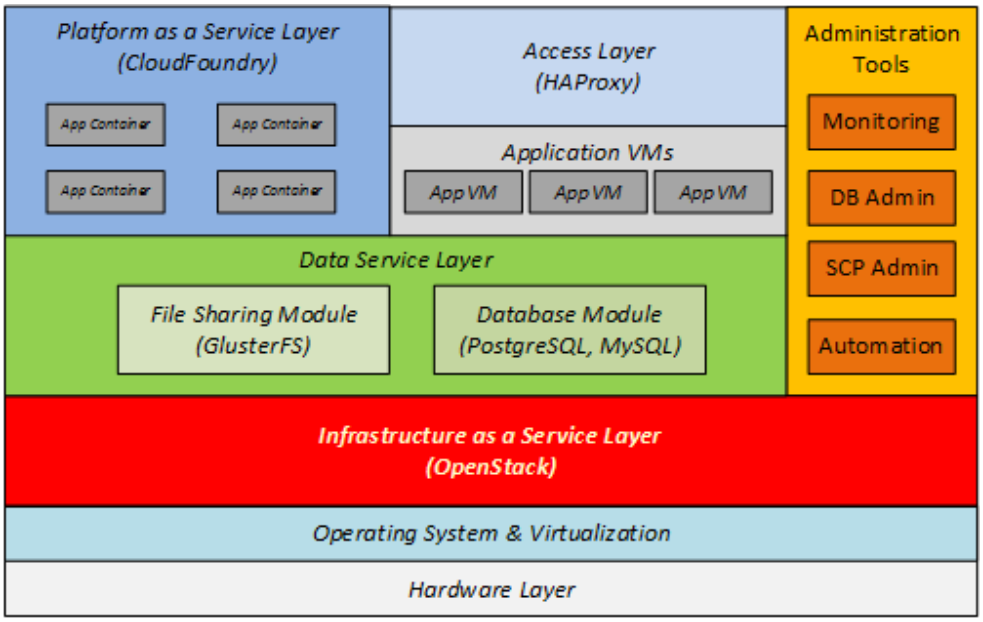

Figure 2. Storm Clouds Platform architecture.

centre(s) where the platform is hosted and was considered out of scope. The only requirement for the Hardware Layer is that the servers are equipped with an Operating System supporting OpenStack [2], the software solution we selected for implementing the IaaS Layer.

The platform supports the Applications in two alternative ways: as Application VMs activated at the IaaS Layer for running the application logic or as Application Containers hosted by the platform as a Service Layer (PaaS Layer). There are pros and cons for these two alternative approaches, as described in following sections of this article; we decided to implement both in order to have a more flexible solution.

While analysing the Applications we observed common functions that we 'factored-out' and support at the platform level. The Data Service Layer implements database and file sharing functionality, while the Access Layer manages the requests coming from the end users of the applications.

Finally, SCP provides Administration Tools for monitoring and managing the resources (i.e., services and applications) activated in the platform.

3.1 Infrastructure as a Service Layer

The Infrastructure as a Service Layer provides services for creating virtual resources (like virtual machines, virtual disks and virtual networks) used instead of their physical counterparts, for deploying and running application software. Resources are provided as services, meaning that they are 'created' when needed, used to run applications, and 'removed' when the application is not needed anymore. The actual computation happen at the physical level but physical resources and applications are not tightly bound together. This makes it easier to reuse the physical infrastructure for several purposes, usually at different times. This is obtained with the extensive use of virtualization technology[9] that is part of the IaaS Layer.

For the implementation of the IaaS Layer, we selected OpenStack[10], an open source technology (all source code is freely available under the Apache 2.0 license). The main reason for selecting this technology was because it was the most popular and most adopted open source IaaS solution[11].

Figure 3 shows the OpenStack high-level logical architecture. OpenStack is composed of the following modules mapping the fundamental IaaS services:

- Nova provides computation services (Virtual Machines);
- Neutron provides networking services (Virtual Networks);
- Cinder provides block storage services (Virtual Disks);
- Swift implements object storage services (Files);
- Horizon provides a web front-end for managing and controlling the resources allocated in the cloud;

[2] Possible distributions are Debian 7.0, openSUSE, SUSE Linux Enterprise Server, Red Hat Enterprise Linux, CentOS, Fedora and Ubuntu 12.04/14.04 (LTS).

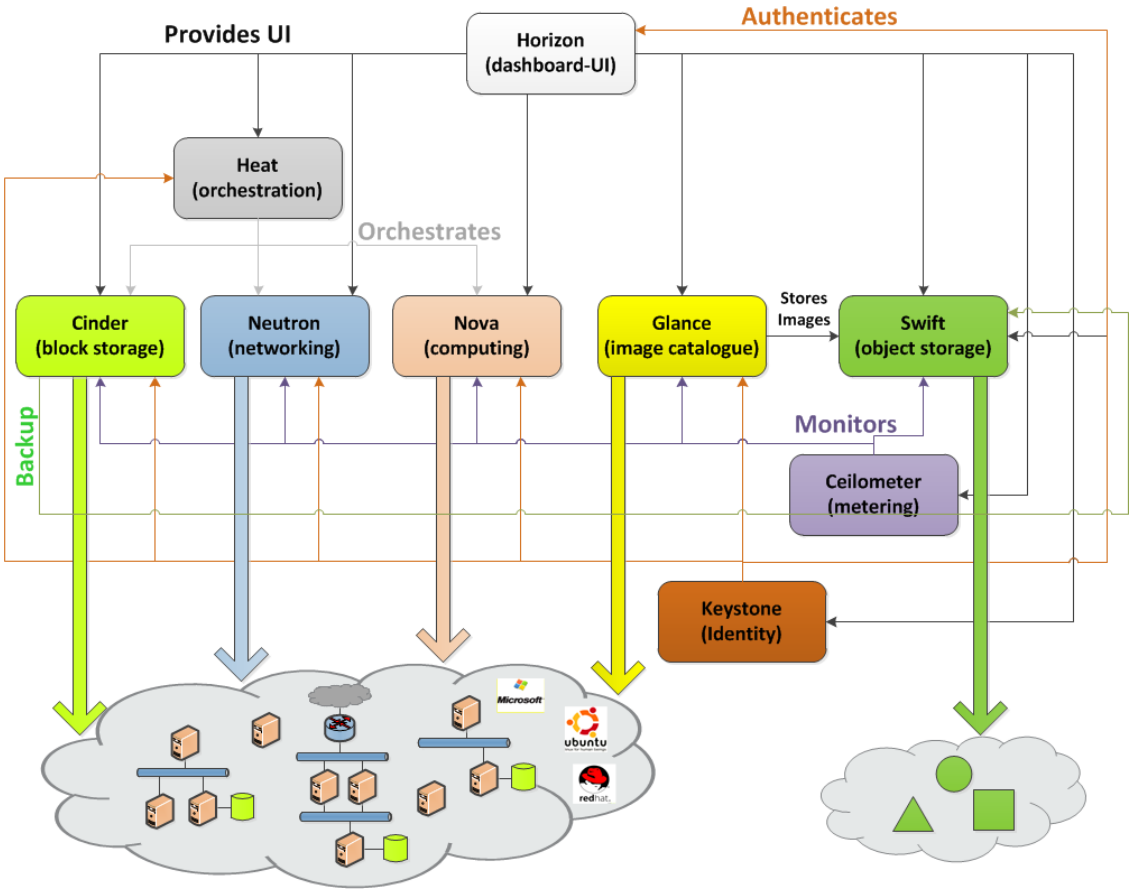

Figure 3. OpenStack logical architecture.

- Glance implements a catalogue for storing virtual machine images;
- Keystone implements authentication and authorization functions;
- Heat uses the other components for orchestrating the creation/deletion of virtual resource groups described by script files called “stacks”;
- Ceilometer monitors the usage of resources for metering and accounting purposes.

In principle, the IaaS Layer could be sufficient for deploying any application we assessed. We could activate a virtual machine, install all the required software packages and run the application. This situation is described by Figure 4.

This scenario, albeit supported by the platform, does not address some important issues of a production-ready situation. What happens if a VM is switched off accidentally or voluntary (for example for maintenance reasons)? What if a single machine is not enough for handling the requests of an increased number of users? In other words, how do we address high-availability and scalability issues?

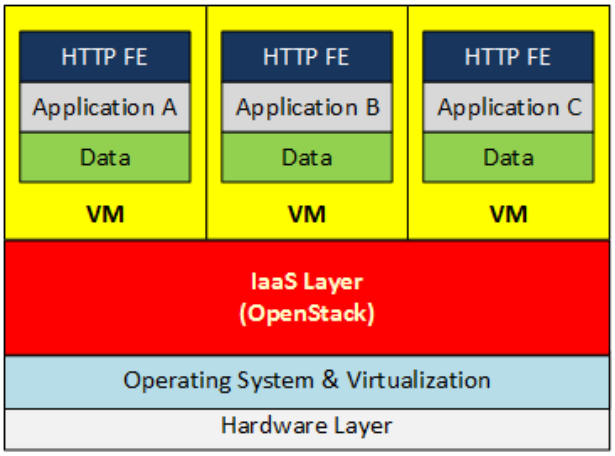

Figure 4. Deploying applications on IaaS.

In addition, from a functional perspective, the essence of an application is represented by the business logic it implements (the part in grey), while functions like HTTP traffic management and data management (e.g., database management) are common to all applications. We decided to implement them at the platform level and make them available as services, in order to

facilitate the application deployment. Following these ideas, we thought of 'decoupling' the three levels of the application stack (namely, the HTTP Front End (HTTP FE), the Application logic and the data management) and provide the HTTP FE functions and the data management functions at SCP level. In addition, we considered that several applications could benefit from Platform-as-a-Service solutions specifically designed for LAMP stacks. As a result, we envisioned the solution shown in Figure 5.

All the layers were implemented using VMs activated at the IaaS Layer level and implement high-availability and scalability, enabling the implementation of similar features for the application services[3].

While working directly at the IaaS layer provides great flexibility and full control, the PaaS layer provides a convenient alternative that facilitates the deployment of applications by 'hiding' the complexity of the underlying infrastructure. In fact, the application developer does not explicitly activate VMs for running her applications; she just uploads software packages to the PaaS Layer that takes care of activating the computing resources on her behalf. Furthermore, the PaaS can handle scalability and high-availability automatically when applications are designed properly.

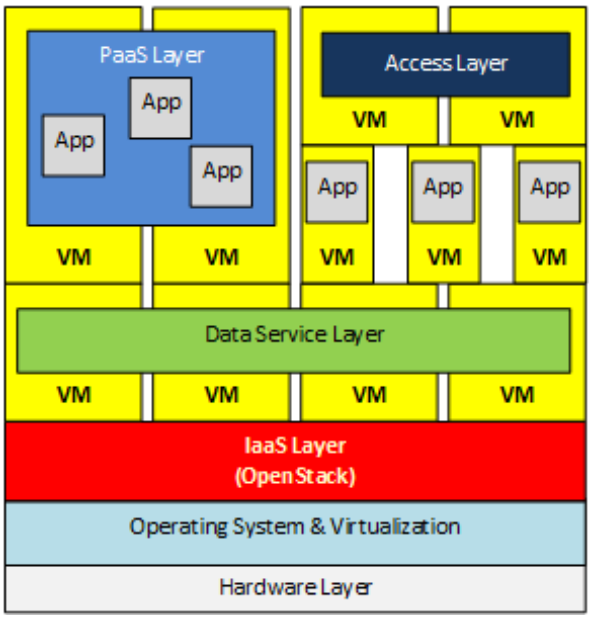

Figure 5. Decoupling the application stack.

[3] High availability and scalability of an application can be obtained only when all the layers support such features, application layer included. For this reason, we cannot claim that their implementation at the Access and the Data Service Layer is sufficient; it also depends on how the application is designed.

In this perspective, SCP enables different application migration options summarized in the Table 2.

The following sections describe the layers built on top of the IaaS Layer in greater details.

3.2 Data Service Layer

The applications that analyzed store structured data in a database and use a file system for unstructured data like images, music and videos. As mentioned above, one of the objectives of the Data Service Layer is to provide mechanisms for implementing H/A; therefore two or more VMs — hosting the business logic of an application — should be able to share data so that, in case one VM becomes unavailable, the other(s) can continue the service.

For these reasons Data Service Layer implements both database and file sharing functions with two subcomponents: the Database Services Module, implementing database functions, and the File Sharing Services Module, providing file server functionality. They are both deployed on VM clusters and implement high-availability and scalability.

At the time of writing, the Database Service Module supports MySQL/MariaDB and PostgreSQL database engines; it is deployed as an active/stand-by VM cluster but the architecture supports also other deployment topologies[4], such as active/active, N+M, N-to-N, etc.[12]

The File Sharing Service Module is implemented as a cluster of VMs hosting Gluster[13], a scale-out network-attached storage file system. Gluster implements a client/server architecture in which the servers are aggregated into one large parallel network file system; while clients, equipped with the Gluster client software package, mount shared volumes that are seen as local file systems.

The Data Service Layer provides each application with a private database and a private volume but multiple VMs of an application can share the same data allowing the implementation of H/A and scalability.

3.3 Access Layer

The Access Layer implements the HTTP Front End for web based services. It receives HTTP requests directed to applications and, having knowledge of what VMs host the required service, dispatches the request accordingly, trying to balance the traffic among all the VMs. In addition, it continuously monitors the availability

[4] The topology of a cluster defines how many nodes are used and/or how the work is distributed among them. For more information see[12].

Table 2. Migration options

Option	Description	Pros	Cons
Full IaaS	All the application components are deployed on VM(s) explicitly managed by the application owner	- No architectural change of the application - Full control on the resources used for the deployment	- Great deployment complexity because the application owner must take care of installing and configuring all the components for H/A and scalability
IaaS + Data Service Layer + Access Layer (optional)	Data management and (optionally) HTTP traffic management are delegated to the platform while the application business logic is still deployed on VM(s)	- No architectural change of the application - Less deployment complexity because the application owner 'leverages' the high-available and scalable features of the platform layers	- Because of the centralized administration of the shared functions (e.g. data service layer), application owners cannot deploy their applications in full autonomy
PaaS + Data Service Layer	Applications are hosted by the PaaS Layer and use the Data Service Layer for storing data	- No infrastructure management required by the user: the platform does it for her	- Applications can require significant changes to comply with PaaS principles

of the VMs and, in case one of them becomes unavailable, redistributes the traffic among the remaining ones.

The Access Layer implements the Load Balancer Module based on the open source HAProxy software[14] and is implemented as a set of VMs deployed 'in front' of the VMs hosting the applications, as described in Figure 6.

The HAProxy VMs receive the HTTP request and dispatch it to the Application VMs distributing the workload according to a load balancing algorithm. In addition, HAProxy VMs periodically monitor the Application VMs and, in case a VM appears unavailable, redistribute the upcoming requests to the remaining ones. As shown in Figure 6, HAProxy can be deployed on several VMs (in the picture we have a two VM cluster) implementing H/A.

3.4 Platform as a Service Layer

The Platform as a Service Layer (PaaS Layer) is a sophisticated solution that allows developers to deploy their web applications to the cloud, without having to take care of the underlying infrastructure. In fact, while IaaS Layer focuses on managing the fundamental

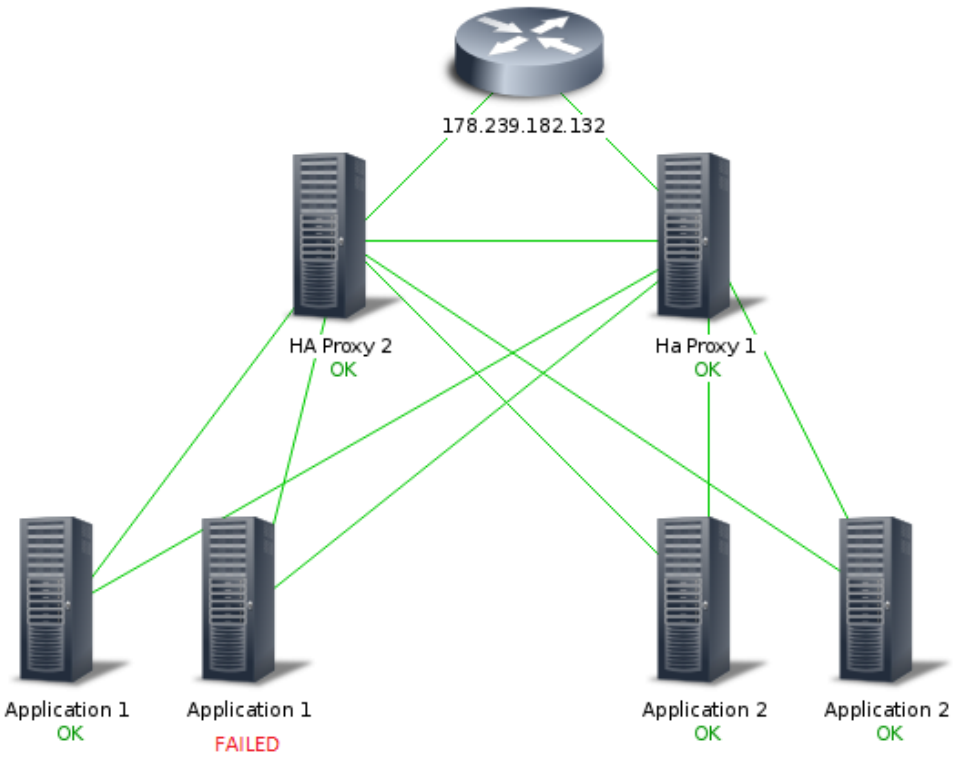

Figure 6. Load balancer module (HAProxy).

infrastructure building blocks in a cloud environment, thus allowing the transfer any existing deployment to the cloud with little or no architectural changes, PaaS Layer goes one step further and focuses on managing applications instead of infrastructure. The developer who can deploy an application to the PaaS and expects it to perform, delegates all infrastructure management tasks to the PaaS and focuses on development work instead. As a consequence, the primary resources involved in deploying an application to a PaaS are not virtual machines, virtual storage and virtual network objects, but application services, configuration and artifacts, as shown in Figure 7.

The picture shows that the PaaS Layer uses resources (mainly VMs) provided by the IaaS Layer 'hiding' the correspondent complexity. Programmers can deploy scalable and highly available applications without requiring advanced infrastructure skills because the PaaS Layer takes care of activating/deactivating VMs for hosting applications on their behalf. The workload is automatically load balanced, similar to what the Access Layer does for IaaS, if the developer chooses to start multiple instances of the application.

Cloud Foundry[15] is the open source solution we selected for implementing the PaaS Layer released under the Apache License 2.0 and supported by the Cloud Foundry Foundation, established in December 2014, with EMC, HP, IBM, Intel, Pivotal, SAP and VMware as platinum members. Cloud Foundry is based on Linux Container (LXC) technology that isolates applications using operating system containers; this feature permits to run several applications on a single machine (virtual or physical) optimizing the resource usage. It is worth mentioning that, in case the programmer requires to activate several copies of an application for H/A, it is up to Cloud Foundry to transparently deploy them on different (virtual) machines.

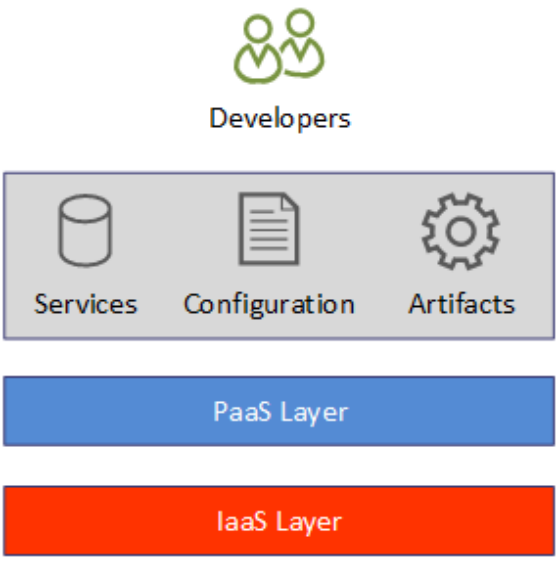

Figure 7. Platform as a service layer.

3.5 Administration Tools

SCP implements functions that both the platform administrator and the application owners are able to use for managing, monitoring and administering the cloud platform components as well as applications running in the cloud. The actions a user can perform depend on her role: the platform administrator has full control on all the objects deployed in the cloud (platform components and applications), whilst application owners have full control on their applications and can perform only some actions on the platform components. For instance, application owners have full control on databases and shared volumes used by their own applications but do not have any control on databases and shared volumes of other application owners.

3.6 Monitoring Module

The Monitoring Module is a component for verifying the working conditions of the resources in the SCP. The cloud administrator can monitor the resources used for implementing the platform services (e.g., the VMs used for the different layers as well as the physical servers of the IaaS) while the application owners can keep under control only the VMs of their own applications.

The Monitoring Module is implemented by Zabbix[16], a tool available under GNU General Public License (GPL) version 2 for monitoring the availability and performance of IT infrastructure components. According to the configuration, Zabbix, continuously gathers information from the servers under control and, in case one or more parameters reach a threshold value, it notifies the operator. Zabbix offers several monitoring options ranging from simple checks for verifying the availability/responsiveness of a server, to sophisticated measurements of parameters like CPU load, disk volume occupation, network traffic, number of processes, etc. Zabbix provides several ways for representing monitoring data in both graphical and textual/tabular format (Figure 8).

Zabbix can inform operators when a problem occurs with a server by sending an e-mail message, an Instant Messages (IM) or an SMS; we used such a feature for

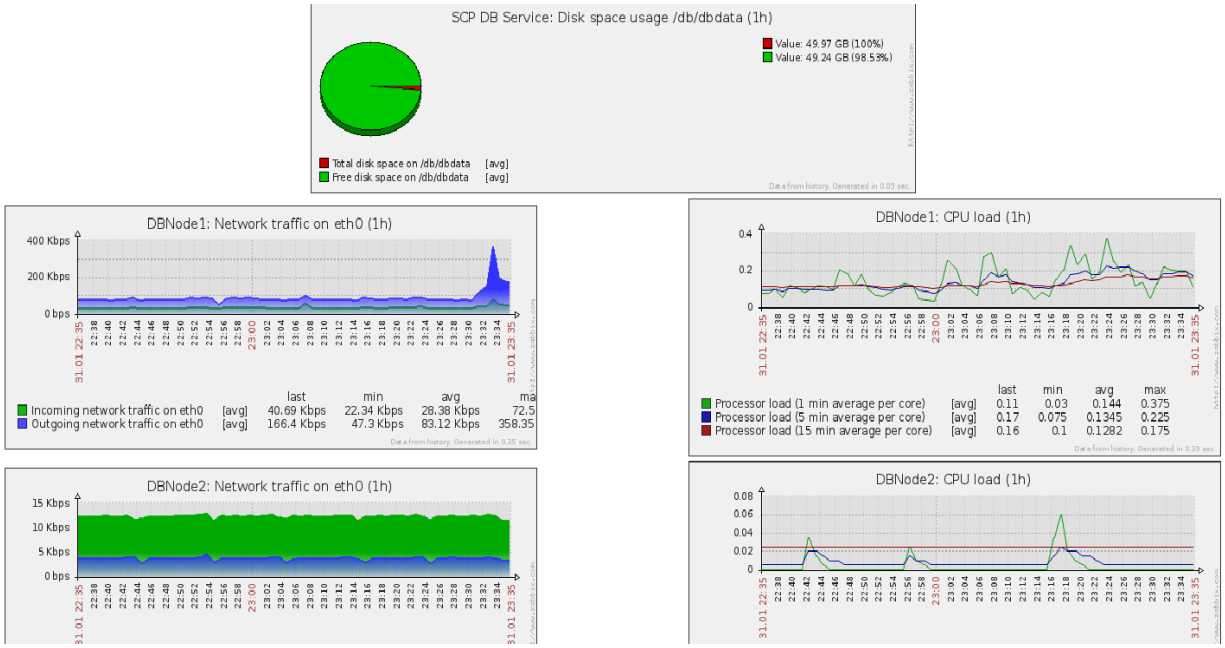

Figure 8. Zabbix monitoring page.

notifying the platform administrator or the application owners in order to request their intervention.

3.7 Database Administration Module

SCP provides web based tools for administering the supported database engines: we selected phpMyAdmin[17], for MySQL administration, phpPgAdmin[18], for PostgreSQL (Figure 9). They implement very similar functions for the corresponding database engine, such as creating, modifying and deleting databases and database objects (e.g., tables, indexes, etc.), submitting queries, importing/exporting data, managing database accounts, etc.

In the SCP context, the cloud administrator has full control on all the objects and configures database accounts for the application owners, giving them the rights of managing only the database objects created for their own applications.

3.8 Platform Administrator's Console

The Platform Administrator's Console is designed exclusively for the SCP administrator who needs to have full control on all the resources in the platform. Through the console, the administrator is able to manage all the elements at any level, IaaS Layer level included.

The console provides a Command Line Interface (CLI) suitable for recurrent tasks that can be automated using CLI scripts. The console is implemented as an Ubuntu Linux Server with the installation of the CLI interfaces of all the other components of the platform:

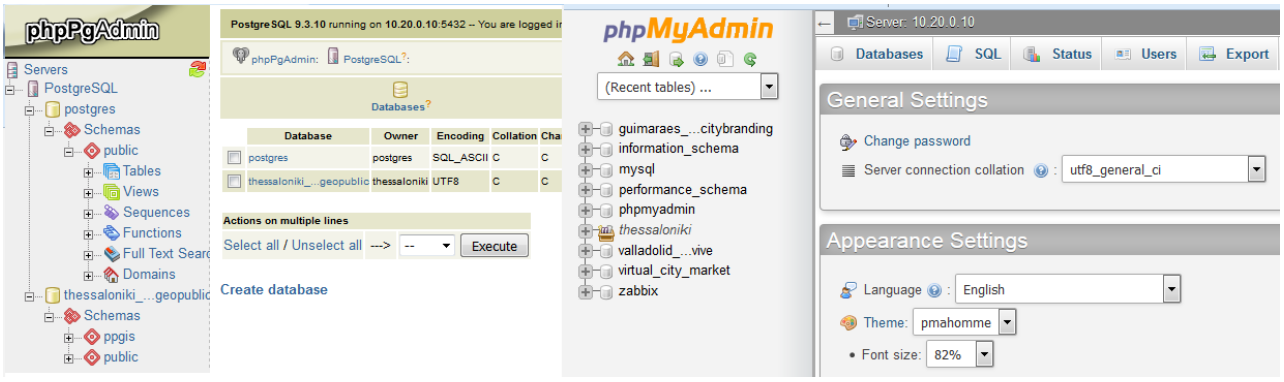

Figure 9. Database Administration — phpPgAdmin and phpMyAdmin.

- MySQL client and PostgreSQL client for managing the corresponding DB engines of the Database Services Module;
- Gluster client, for managing the File Sharing Services Module;
- OpenStack CLI software packages, for managing the IaaS Layer
- Cloud Foundry CLI, for managing the PaaS Layer.

The console is activated as a VM in the IaaS cloud when SCP is hosted in an OpenStack public cloud. However, it can also be deployed on a physical server when SCP is deployed on private cloud.

During the project, we heavily used the console for automating the deployment and the configuration both of the platform components and the application services. For this purpose we extensively used OpenStack Heat that permits the IaaS cloud user to describe all the IaaS objects that are needed for an application in a script — called stack — and to "*control the entire lifecycle of infrastructure and applications within OpenStack clouds*"[19]. In this perspective, the activation and deactivation of the IaaS objects can be simply obtained by 'submitting a stack' to Heat that is in charge of automatically creating/destroying the listed IaaS objects (e.g., VMs, Virtual Disks, etc.).

4. Conclusion

This article described the experience we gained on STORM CLOUDS, a project experimenting the migration of smart-city digital services to a cloud computing paradigm.

After analysing the problem, both from the organizational and the technical point of views, we decided to implement Storm Clouds Platform (SCP), a cloud computing infrastructure designed for hosting the applications selected by the project consortium.

During the project, we implemented two instances of the platform: one at Hewlett Packard Entreprise's premises (SCP@HPE), the other hosted at a public cloud-computing operator (SCP@Operator). We used SCP@HPE mainly for testing purposes and for supporting the 'cloudification process', consisting in the technical activities for porting the selected applications to cloud (e.g., adaptation, configuration, automation, etc.). SCP@Operator, on the other hand, was used as the "production environment", for making the migrated applications available to the end users on Internet.

SCP@HPE is a private cloud providing services for exclusive use of the STORM CLOUDS project partners while SCP@Operator is hosted on a public cloud. This exercise demonstrated that our solution supports both public and private deployment models allowing the project partners (in particular PAs) to decide how to manage their applications once the project terminates. In some cases, they may decide to keep their applications on a public cloud operator or, as an alternative, they can deploy services on equipment at their own sites, for instance for fulfilling privacy and security requirements.

All the software components used for the SCP implementation are available under an Open-Source Software (OSS) license, fulfilling one of the main requirements of the project. We selected broadly adopted software packages in order to guarantee long term support for the solution.

The architecture presented here is a baseline for future extensions and modifications with the objective of improving the way functions are implemented or for adding new functions not currently available. As an example, today — when an application owner uses the Data Service Layer — the platform administrator needs to create the database(s) and the shared volume (s) for the programmer to use. This operational model does not fulfils one of the fundamental requirements for a pure "as-a-service" paradigm in which services (in this case the database and the shared volumes) should be provided in a self-service manner without any intervention of the cloud administrator. OpenStack community is actively working on these aspects that are respectively addressed by Trove[20] and Manila[12] projects. Similar problems affect the monitoring functions we implemented and are addressed by another OpenStack project called Monasca[21]. When the project was started, these solutions were not available or they were in a very primitive state not suitable for a production-ready environment; consequently we decided to implement those functions following a more traditional, yet more proven, approach. Moreover, the solution we provide can be more easily replicated on public clouds based on OpenStack because, according to the OpenStack Market Place web page[22], at the time of writing only one operator offers Database-as-a-Service (implemented with Trove) and none of them implements Manila or Monasca.

Evolutions can be directed to support new programming and deployment paradigms like, for example, Docker[23] that is based on Linux containers (LXC), the same technology used by our Cloud Foundry based PaaS Layer. Actually, Docker can substitute Cloud Foundry but presents similar adoption problems: the

application architecture need to be reviewed in order to exploit containers at their full potential and this requires a strong commitment by the application owners with the related costs. It is worth to remember that this was the main reason why we designed a solution supporting both IaaS-based deployment paradigm (more traditional and with less impact on the applications) and a PaaS deployment paradigm (that would require adaptations).

In conclusion, we think that our experience demonstrates that cloud computing is a viable solution for implementing smart city services. Stakeholders can take great advantage of the inherent delivery model that promotes agility, speed and cost savings. Certain issues such as lack of control on how/where data and applications are managed/deployed and vendor-lock-in are still obstacles for the adoption of public cloud computing models. However, we feel that solutions like the one described in this paper are on the direction of alleviating the problem.

Conflict of Interest and Funding

No conflict of interest was reported by all authors. STORM CLOUDS, short form for "Surfing Towards the Opportunity of Real Migration to Cloud-based public Services", is a project partially funded by the European Commission within the 7th Framework Program (Grant Agreement No. 621089)[24].

References

1. STORM CLOUDS Consortium, 2013, *Surfing Towards the Opportunity of Real Migration to CLOUD-based public Services*, viewed January 30, 2016, <http://stormclouds.asi-soft.com/the-project/>
2. Amazon.com Inc., n.d., *Amazon Web Services — Main, Page*, viewed January 7, 2016, <https://aws.amazon.com>
3. Microsoft Corporation, n.d., *Microsof Azure — Main Page*, viewed January 10, 2016, <https://azure.microsoft.com>
4. *Directive 95/46/EC of the European Parliament and of the Council of 24 October 1995 on the protection of individuals with regard to the processing of personal data and on the free movement of such data*, n.d., viewed January 11, 2016, <http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=urisery:OJ.L_.1995.281.01.0031.01.ENG>
5. Google Inc., n.d., *Google Maps — Main Page*, viewed January 2, 2016, <https://maps.google.com>
6. *CORDIS — Free and open source software activities in European Information Society initiatives*, n.d., viewed January 4, 2016, <http://cordis.europa.eu/fp7/ict/ssai/foss-home_en.html>
7. *LAMP (software bundle) — Wikipedia page*, n.d., viewed January 30, 2016, <https://en.wikipedia.org/wiki/LAMP_(software_bundle)>
8. Mell P and Grance T, 2011, *The NIST definition of cloud computing*, viewed January 29, 2016, <http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf>
9. *Virtualization — Wikipedia page*, n.d., viewed January 20, 2016, <https://en.wikipedia.org/wiki/Virtualization>
10. *OpenStack project*, n.d., viewed February 1, 2016, <https://www.openstack.org>
11. *Top 10 open source projects of 2014*, n.d., viewed February 1, 2016, <https://opensource.com/business/14/12/top-10-open-source-projects-2014>
12. *OpenStack Manila — Wiki page*, n.d., viewed January 22, 2016, <https://wiki.openstack.org/wiki/Manila>
13. *Gluster — main page*, n.d., viewed January 15, 2016, <https://www.gluster.org>
14. *HAProxy — main page*, n.d., viewed January 17, 2016, <http://www.haproxy.org>
15. *Cloud Foundry — main page*, n.d., viewed January 11, 2016, <https://www.cloudfoundry.org>
16. Zabbix Company, n.d., *Zabbix — main page*, viewed January 29, 2016, <http://www.zabbix.com>
17. *phpMyAdmin — main page*, n.d., viewed January 21, 2016, <https://www.phpmyadmin.net>
18. *phpPgAdmin — main page*, n.d., viewed January 21, 2016, <http://phppgadmin.sourceforge.net/doku.php>
19. *OpenStack Heat — Wiki page*, n.d., viewed January 21, 2016, <https://wiki.openstack.org/wiki/Heat>
20. *OpenStack — Trove page*, n.d., viewed January 22, 2016, <https://wiki.openstack.org/wiki/Trove>
21. *OpenStack Monasca — Wiki Page*, n.d., viewed January 22, 2016, <https://wiki.openstack.org/wiki/Monasca>
22. *OpenStack marketplace — web page*, n.d., viewed January 23, 2016, <https://www.openstack.org/marketplace/public-clouds>
23. Docker Inc., n.d., *Docker main page*, viewed January 23, 2016, <https://www.docker.com>
24. *Storm Clouds Project — European Commission project page*, n.d., viewed July 1, 2014, <http://ec.europa.eu/digital-agenda/en/storm-clouds-project-cloud-public-services>
25. *Vendor lock-in — Wikipedia page*, n.d., viewed January 12, 2016, <http://en.wikipedia.org/wiki/Vendor_lock-in>
26. *OpenStack — Neutron/LBaaS page*, n.d., viewed January 15, 2016, <https://wiki.openstack.org/wiki/Neutron/LBaaS>
27. *Puppet open source*, n.d., viewed January 24, 2016, <https://puppet.com/product/open-source-projects>
28. *Zabbix license page*, n.d., viewed January 25, 2016, <http://www.zabbix.com/license.php>
29. *High-availability cluster — Wikipedia page*, n.d., viewed January 26, 2016, <https://en.wikipedia.org/wiki/High-availability_cluster>