Project Acronym: STORM CLOUDS

Grant Agreement number: 621089

Project Title: STORM CLOUDS – Surfing Towards the Opportunity of Real Migration to CLOUD-based public Services

D2.3.3

Storm Clouds Platform Implementation Status Report

Workpackage: WP2

Version: 0.3

Date: 23/09/2015

Status: WP leader accepted

Dissemination Level: PUBLIC

Nature: REPORT

Editor: Marco Consonni (Hewlett Packard Italiana S.r.l.)

Authors: Marco Consonni (Hewlett Packard Italiana S.r.l), Andrea Milani (Hewlett Packard Italiana S.r.l)

Reviewed by: Alkiviadis Giannakoulias (European Dynamics)

Version Control

Modified by	Date	Version	Comments
Marco Consonni	15/8/2015	0.1	First draft
Alkiviadis Giannakoulias	21/9/2015	0.2	WP Internal Review
Marco Consonni Andrea Milani	23/9/2015	0.3	Ready for Project Coordinator Review

Executive Summary

Surfing Towards the Opportunity of Real Migration to Cloud-based public Services (STORM CLOUDS) is a project partially funded by the European Commission within the 7th Framework Program in the context of the Capital Improvement Plan (CIP) project (Grant Agreement No. 621089). The project has the objective of exploring the shift to a cloud-based paradigm for providing services that Public Authorities (PAs) currently implement with more traditional Information Technology (IT) deployment models. The defines guidelines on moving application to cloud computing and is based on direct experimentation with pilot projects conducted in, at least, the cities participating to the consortium [1].

Work Package 2 (WP2) delivers Storm Clouds Platform (SCP), a cloud computing platform for hosting application services. SCP implements Infrastructure as a Service (IaaS) functions, Platform as a Service (PaaS) functions as well as other features (Database Layer, Monitoring Layer, etc.) that significantly facilitate the deployment of applications in a cloud environment.

This document is the third issue of the iterative deliverable "Storm Clouds Platform - Implementation Status Report" that describes the current status of the SCP providing technical details on the actual implementation with particular emphasis on implementation of custom components, on top of off-the-shelf products, for the SCP automated deployment.

Table of Contents

Abbreviations

Acronym	Description
CLI	Command Line Interface
CIDR	Classless Inter-Domain Routing
CIP	Capital Improvement Plan
CIP-PSP	*See CIP and PSP*
DB	Data Base
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
DOW	Description of Work
FP7	Framework Program 7
GB	Gigabyte
GUI	Graphical User Interface
HOT	Heat Template Language
IaaS	Infrastructure as a Service
IT	Information Technology
N/A	Not Available or Not Applicable
PA	Public Authority
PaaS	Platform as a Service
PDF	Portable Document Format
PSP	Policy Support Program
SCP	STORM Cloud Platform
SME	Small and Medium Enterprise
TBD	To Be Defined
TBW	To Be Written
URL	Uniform Resource Locator
WP	Work Package
YAML	YAML Ain't Markup Language

1 Current Implementation Status

Storm Cloud Platform (SCP) is deployed in two forms: SCP@HP and SCP@Enter. The former is hosted on physical infrastructure housed at HP's premises and is mainly used for development and testing purposes; the latter is implemented using services made available by a public cloud operator (Enter S.r.l. http://www.enterpoint.it) that provides IaaS services based on OpenStack.

The following table briefly summarizes the evolution of the SCP throughout the project milestones:

MILESTONE	TITLE	NOTES
M6	IaaS and Basic DbaaS	SCP@HP • IaaS services, Object Storage Excluded • DB Layer - Basic Functionality (no HA) SCP@Enter • IaaS Fully Implemented
M12	Backup and Basic Monitoring	SCP@HP • IaaS - Object Storage Implemented • Monitoring – Basic Functionality (no HA) SCP@Enter • IaaS Fully Implemented • Monitoring – Basic Functionality (no HA)
M20	High Availability DB Layer, Monitorig Services, PaaS	SCP@HP • IaaS Fully Implemented • DB Layer Fully Implemented • Monitoring Fully Implemented • PaaS Fully Implemented • Jump Start Station Fully Implemented SCP@Enter • IaaS Fully Implemented • DB Layer Fully Implemented • Monitoring Fully Implemented • PaaS Fully Implemented • Jump Start Station Fully Implemented

able 1-1 Storm Clouds Platform Evolution

The table reads as follows:

- **MILESTONE**: delivery date;
- **TITLE**: brief title summarizing the implementation stage
- **NOTES**: more detailed description of the implementation status for the two SCP instances.

As shown above, the first two delivery phases were mostly dedicated to the implementation of the IaaS platform services on SCP@HP; for this reason the two previous issues of this document [2], [3] describe the hardware selected for the implementation and how OpenStack has been deployed on that. This information will not be reported here but the interested readers are invited to refer to these versions.

In the latest phase the activities have been focused on implementing SCP services like DB Layer, Monitoring Layer, PaaS functions, etc. As explained in [4], these services are deployed "on-top" of the IaaS Layer that provides the computational resources (e.g. VMs, virtual volumes, virtual network objects, etc.) for running the software selected for the implementation. These services can be collectively called as **over-cloud services** for emphasizing the fact that they are implemented and run using "objects" made available by the Infrastructure as a Service (IaaS) Layer.

This concept is summarized by the picture showing the logical view of the SCP Overall Architecture [4][1]:

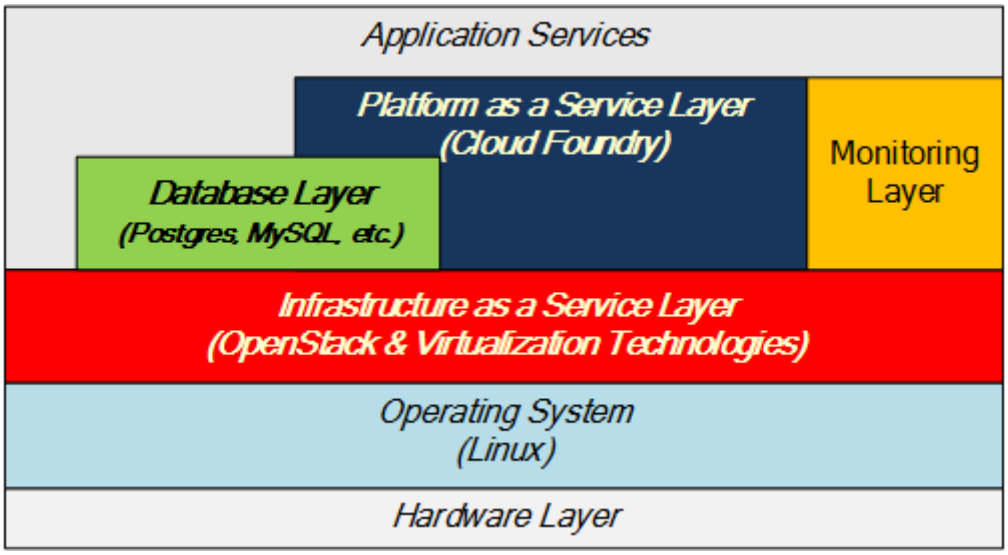


Figure 1 1 - Storm Clouds Platform - Overall Architecture

The creation of infrastructural objects on OpenStack has been automated using Heat, the OpenStack orchestration engine [5], used for deploying the over-cloud services. This practice, which has already been applied for the creation of second version of the "Cloud Application Catalogue" [6], results in the production of software scripts that, in addition to being **tools** used for activating SCP services in the IaaS platform, are also project deliverables documenting how to arrange IaaS objects as well as install and configure software packages on the virtual machines (VMs) in order to deliver the required functions.

[1] The logical view of the Overall SCP Architecture has been slightly modified for reflecting the current state of the art.

2 SCP Deployment Tools

This chapter describes the tools for deploying SCP over-cloud components.

For each tool, it provides the following information:

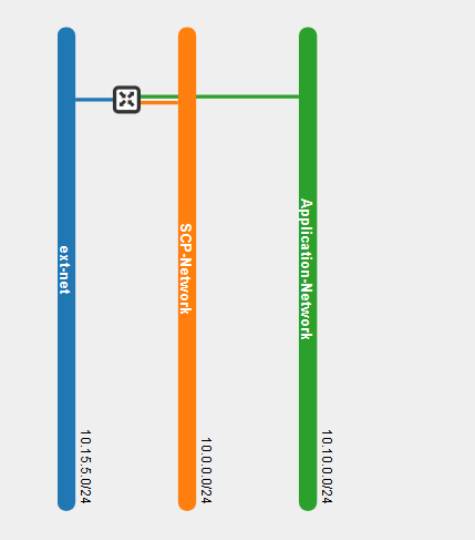
- **Name**: the name of the tool;
- **Languages**: the programming language(s) used for implementing the tool;
- **Description**: a description of the tool;
- **Input**: input parameters;
- **Output**: output values.

The tools are implemented with the following scripting languages:

- **YAML (Heat)**: used for creating IaaS objects using OpenStack Heat;
- **Bash Shell**: used both for configuring SCP@HP and for configuring the VMs created through YAML (heat) scripts.

2.1 SCP Stack

This section describes the Heat script for creating the SCP over-cloud services.

Name	SCP.yaml
Languages	Yaml (heat script)
Description	This script creates all the IaaS objects required for hosting and supporting the STORM cloud applications: - Network Objects; - DB Layer Objects; - Monitoring Layer Objects; - Platform as a Service Objects; - Jump-Start Station Objects. As a general rule, all the Virtual Machine (VM) objects install Ubuntu Server 14.04.3 LTS, Trusty Tahr [7]. **Network Objects** These objects represent the networking environment where both the SCP over-cloud services and the application services run[2]. The following picture shows the network objects created by the script: 

[2] The network layout created by this script supersede the one in [6].

SCP-Network (represented by the orange thick line) is the private network to which all SCP over-cloud services are connected. It supports addresses belonging to subnet 10.0.0.0/24 (IP address range reserved for private networks [8]) and provides DHCP service. An IP address subrange (by default addresses from 10.0.0.1 to 10.0.0.127) is reserved for static IP allocation. The reason for reserving a range of IP addresses for static allocation is that some SCP services are themselves implemented as VMs (for example the DB layer) and assigning them a static IP address prevents the need for a local DNS service. Access to those services is through 'well-known' IP addresses.
Application-Network (represented by the green thick line) is the private network to which VMs hosting the application services are connected. It supports addresses belonging to subnet 10.10.0.0/24 and provides DHCP service. An IP address subrange (by default addresses from 10.10.0.1 to 10.10.0.127) is reserved for static IP allocation.
SCP-Router (represented by the small black square) connects the two networks to each other and to the external provider network (represented by the blue thick line).
The provider network, being the representation of the network for accessing Internet, is not created by the script and must be made available by the OpenStack administrator.

DB Layer Objects
This is a set of objects implementing the DB Layer services. It is an active-passive cluster composed of three VMs arranged as described in [4] (see Appendix A - Active-Passive Cluster on Linux Platform). Two VMs – namely the active and the stand-by nodes - host the DB engine; the third VM – the quorum node – is used for cluster voting purposes only. In order to minimize the resources used for the implementation, both MySQL and PostgreSQL DB engines are hosted in the three-node cluster. In case of workload increase, the DB Layer can be easily adapted for deploying MySQL and PostgreSQL on different three-node clusters as well as for creating multiple copies of three-node clusters hosting MySQL, PostgreSQL or both.

DBNode1 is one of the two nodes in the cluster hosting the DB engines. When the cluster is deployed, this VM is configured as the active (or master) node, actually providing the database service for the applications.
The VM installs the following software packages:
- DRBD Ver. 8.4.3
- Corosync Ver. 2.3.3
- Pacemaker Ver. 1.1.10
- MySQL Server Ver. 5.5.44 (with InnoDB)
- PostgreSQL Server Ver. 9.3.9
- Zabbix Agent Ver. 2.4.6 (see Monitoring Layer)

The two DB engines are configured for storing data on DBVolume1 (see below).

DBNode2 is similar to DBNode1 but, when the cluster is deployed, the VM is configured as the stand-by (or slave) node. Should DBNode1 fail, the cluster software automatically promotes DBNode2 to master. In this way, DB services are delivered without interruption.
The node is equipped with the same software installed on DBNode1 and stores database data on DBVolume2 (see below).

DBNodeQ is the quorum node of the cluster.
It is equipped with the same software installed on DBNode1 and DBNode2 albeit the DB services are disabled. In addition, the quorum node hosts DB administration software packages providing a web-based GUI:
- phpmyadmin Ver. 4.0.10 (for administering MySQL);
- phppgadmin Ver. 5.1 (for administering PostgreSQL);

DBVolume1 is the external virtual volume where DBNode1 stores the database data.
The reason for using an external volume instead of the VM internal volumes are:
- the size of the volume is independent on the 'flavor[3]' of the VM allowing more flexibility during the deployment;

[3] OpenStack flavors specify the amount of resources (e.g. RAM size, boot disk size, number of virtual CPUs, etc.) allocated when a VM (or instance) is created

- the lifecycle of external volumes is independent of the lifecycle of the VM the volume is connected to (e.g. the VM can be destroyed while the volume is kept)
- the virtualization technology used for implementing virtual volumes at IaaS layer level (i.e. OpenStack) usually optimizes the usage of resources with benefits both in terms of performance and cost (e.g. the price per GB of a IaaS block storage service is cheaper than the corresponding price per GB allocated in the VM).

DBVolume2 is similar to DBVolume1 but used by DBNode2.

Notes:

- The DB services are available at the well-known IP address 10.0.0.10.
- DBNode1and DBNode2 should be deployed on two different physical servers in order to prevent a failure at the physical level from compromising the cluster functionality. In this perspective they can be deployed on two different availability zones[4].
- DBVolume1 and DBVolume2 should be deployed on two different availability zones.
- The DB administration web interfaces for the databases are available at the well-known URLs:
  - http://10.0.0.13/phpmyadmin (MySQL web administration);
  - http://10.0.0.13/phppgadmin (PostgreSQL web administration);

Monitoring Layer Objects

This is the set of objects implementing the Monitoring Layer services.
It is an active-standby cluster composed of three VMs: two of them host the Zabbix services – namely Zabbix server and Zabbix Web Front End – the third is the quorum node.
Zabbix requires a database engine for storing data.
In SCP, the database of the Monitoring Layer is hosted on the DB Layer.
The following paragraphs describe the most relevant objects in the cluster.

ZabbixNode1 is one of the two nodes of the active-standby Zabbix cluster. When SCP is deployed, it is configured as the active node. The VM installs the following software packages:

- Zabbix Server MySQL Ver. 2.4.6
- Zabbix Frontend php Ver. 2.4.6
- OpenStack Swift CLI Ver. 2.0.3
- Duplicity Ver. 0.6.23

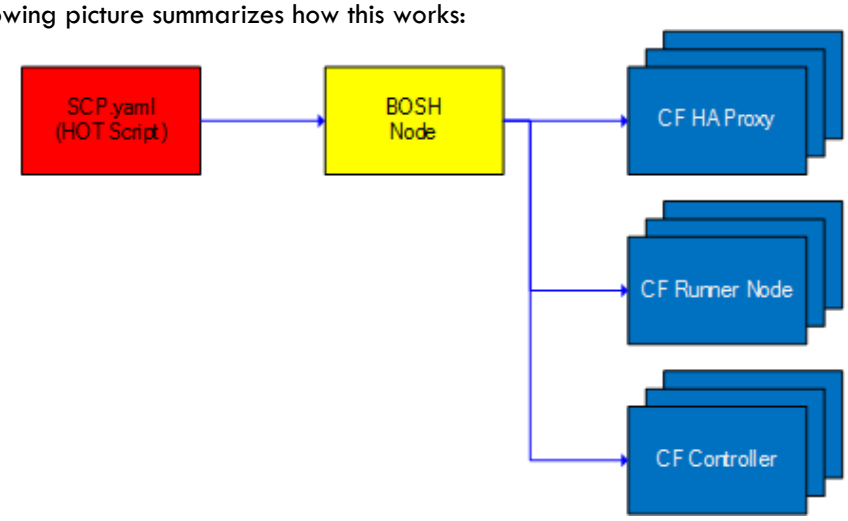
ZabbixNode2 is similar to ZabbixNode1 but configured as the standby node when SCP is deployed.

ZabbixNodeQ is the quorum node of the cluster; it is equipped with the same software installed on ZabbixNode1 and ZabbixNode2 albeit the monitoring services are disabled.

Notes:

- ZabbixNode1 and ZabbixNode2 should be deployed on two different physical servers in order to preserve the cluster functionality in case of hardware failure. For this reason they should be deployed on two different availability zones (see also DB Layer description).
- Zabbix nodes install Swift CLI and Duplicity for implementing the database backup according to the guidelines defined in [4] (see par. "2.4 Data Back-Up Using the IaaS Platform Services").
- The Zabbix server as well as the Zabbix Frontend are both available at well-known IP address 10.0.0.20.
- Albeit Zabbix is able to monitor nodes without necessarily requiring the installation of the Zabbix agent, using the agent is strongly recommended. For this reason, the other nodes in the SCP are equipped with such a piece of software.

[4] For a brief description of OpenStack availability zones, see [4] at paragraph "2.3.1.2 Availability Zones".

	Platform as a Service Objects These are the objects for implementing the Platform as a Service component based on CloudFoundry [9]. As described in the SCP architecture [4], the deployment of CloudFoundry is based on BOSH [10], a free software deployment and lifecycle engine that can deploy complex systems on several IaaS platforms, including OpenStack. BOSH creates IaaS objects directly calling the IaaS API and orchestrating the deployment of a system. In this perspective, the object created by the HOT YAML script described here is only a node where BOSH is installed, configured and run; the creation of the other IaaS objects required for deploying CloudFoundry is actually performed by the BOSH node and not by the SCP.yaml script. The following picture summarizes how this works:  **BOSHNode** is the VM for orchestrating the deployment of CloudFoundry. The VM installs the following software packages: - BOSH CLI Ver. 1.3072.0 - CloudFoundry Software Distribution Package Ver. 2.10 - Zabbix Agent Ver. 2.4.6 **CF-HAProxyNode, CF-RunnerNode, CF-ControllerNode**: these nodes collectively implement the CloudFoundry platform. They are created by BOSHNode that also installs the required software components taken from the CloudFoundry Software Distribution Package. ***Notes:*** - BOSHNode is configured for connecting CF-HAProxyNode and CF-RunnerNode to Application-Network because they are used for accessing and hosting application services. - CF-ControllerNode is connected to SCP-Network - The CloudFoundry API terminator, deployed on CF-HAProxy, is made available at the well-known IP address 10.10.0.10. **Jump-Start Station** It is a single Virtual Machine designed for managing and administering both the over-cloud SCP services and the applications hosted by the SCP. It is equipped with the OpenStack CLI packages for interoperating with OpenStack.
Input	The script receives several input parameters for facilitating the deployment of the SCP over-cloud services in different IaaS platforms (e.g. SCP@HP and SCP@Enter). The parameters are organized in groups gathering functional related information. **Public_Network_Parameters** - **Public_Network**: name of the public network provided by the OpenStack administrator - **Public_DNS_IP**: IP address of the DNS service provided by the OpenStack administrator **SCP_Network_Parameters** - **SCP_Subnet_1stIP**: first IP address - **SCP_Subnet_CIDR**: IP address range in CIDR notation

	- **SCP_Gateway_IP**: IP address of the network gateway - **SCP_1stDHCPIP**: first IP address allocated via DHCP - **SCP_LastDHCPIP**: last IP allocated via DHCP **Application_Network_Parameters** - **Application_Subnet_1stIP**: first IP address - **Application_Subnet_CIDR**: IP address range in CIDR notation - **Application_Gateway_IP**: IP address of the network gateway - **Application_1stDHCPIP**: first IP address allocated via DHCP - **Application_LastDHCPIP**: last IP allocated via DHCP **DB_Server_Parameters** - **DB_Virtual_IP**: well-known IP address of the DB service - **DB_Node1_IP**: fixed IP address of DBNode1 - **DB_Node2_IP**: fixed IP address of DBNode2 - **DB_NodeQ_IP**: fixed IP address of DBNodeQ - **DB_Node_Image**: name of the VM image to boot from - **DB_Node_Flavor**: OpenStack flavor for creating DBNode instances - **DB_Quorum_Node_Flavor**: OpenStack flavor for creating DBNodeQ instance - **DBNode1_Availability_Zone**: availability zone of DBNode1 - **DBNode2_Availability_Zone**: availability zone of DBNode2 - **DBNodeQ_Availability_Zone**: availability zone of DBNodeQ - **DB_Volume_Size**: size in GB of the DB volumes - **DBVolume1_Availability_Zone**: availability zone of DBVolume1 - **DBVolume2_Availability_Zone**: availability zone of DBVolume2 - **DB_Admin_User**: database administrator's username - **DB_Admin_Pass**: database administrator's password **Zabbix_Parameters** - **Zabbix_Virtual_IP**: well-known IP address of the Zabbix service - **Zabbix_Node1_IP**: fixed IP address of ZabbixNode1 - **Zabbix_Node2_IP**: fixed IP address of ZabbixNode2 - **Zabbix_NodeQ_IP**: fixed IP address of ZabbixNodeQ - **Zabbix_Node_Image**: name of the VM image to boot from - **Zabbix_Node_Flavor**: OpenStack flavor for creating ZabbixNode instances - **Zabbix_DB_Name**: name or IP address where MySQL database service is available - **ZabbixNode1_Availability_Zone**: availability zone of ZabbixNode1 - **ZabbixNode2_Availability_Zone**: availability zone of ZabbixNode2 - **ZabbixNodeQ_Availability_Zone**: availability zone of ZabbixNodeQ **CloudFoundry_Parameters** - **CloudFoundry_Virtual_IP**: well-known IP address of the CloudFoundry service **OpenStack_Parameters** - **Key_Name**: name of the private/public keypair used to boot/access the created instances - **OS_USERNAME**: OpenStack account username - **OS_PASSWORD**: OpenStack account password - **OS_TENANT_NAME**: OpenStack tenant where the stack is deployed - **OS_AUTH_URL**: OpenStack authentication service URL
Output	None

The SCP Heat Template[5] described above calls several bash shell scripts for installing and configuring the software packages on the nodes. These scripts, often called configuration **fragments**, are organized in functionally related groups and described in the following sections.

[5] Heat Template is synonym for HOT script; a script, written in HOT (Heat Orchestration Template) language, interpreted by OpenStack Heat.

2.2 DRBD

The following are the scripts for installing and configuring DRBD.

They work with the assumption that the synchronized volume is provided as an external OpenStack virtual volume.

Name	drbd_setup.sh
Languages	Bash Shell
Description	It installs and configures DRBD software for the active-standby cluster
Input	__Node1_Name__: name of node 1 __Node2_Name__: name of node 2 __NodeQ_Name__: name of the quorum node __ DRBD_Resource__: name of the DRBD resource to activate (external volume) __ Volume_Mountpoint__: name of the mount point for the DRBD resource
Output	None

Name	drbd_master_configure.sh
Languages	Bash Shell
Description	It configures the node as DRBD master and performs the initial synchronization of DRBD managed volumes (master and slave).
Input	__DB_DRBD_Resource__: name of the DRBD resource __DB_DRBD_Device__: name of the volume __ Volume_Mountpoint__: name of the mount point for the DRBD resource
Output	None

2.3 PostgreSQL

The following are the scripts for installing and configuring PostgreSQL database engine and phppgadmin, the web based administration GUI.

Name	postgresql_setup.sh
Languages	Bash Shell
Description	It installs and configures PostgreSQL software package
Input	__DB_Directory__: name of the DB directory
Output	None

Name	postgresql_master_finalize.sh
Languages	Bash Shell
Description	It creates the PostgreSQL database directory and configures PostgreSQL engine accordingly (note that the DB directory needs to be created on the master only because the slave disk is automatically synchronized by DRBD)
Input	__DB_Directory__: name of the DB directory __DB_Admin_User: username for the DB administrator __DB_Admin_Password: password for the DB administrator
Output	None

Name	phppgadmin_setup.sh
Languages	Bash Shell
Description	It installs and configures phppgadmin software package.
Input	__Virtual_IP__: IP address where the PostgreSQL service is made available __DB_Admin_User: username for the DB administrator __DB_Admin_Password: password for the DB administrator
Output	None

2.4 MySQL

The following are the scripts for installing and configuring MySQL database engine and phpmyadmin, the web based administration GUI.

Name	mysql_setup.sh
Languages	Bash Shell
Description	It installs and configures MySQL database engine
Input	__DB_Directory__: name of the DB directory __DB_Root_Password__: the password for 'root' user (DB super-user)
Output	None

Name	mysql_master_finalize.sh
Languages	Bash Shell
Description	It finalizes the MySQL database installation on the master node (note that the finalization is required for the master node only because the slave disk is automatically synchronized by DRBD)
Input	__DB_Admin_User: username for the DB administrator __DB_Admin_Password: password for the DB administrator
Output	None

Name	phpmyadmin_setup.sh
Languages	Bash Shell
Description	It installs and configures phpmyadmin software package.
Input	__Virtual_IP__: IP address where the MySQL service is made available __DB_Admin_User: username for the DB administrator __DB_Admin_Password: password for the DB administrator
Output	None

2.5 Zabbix

The following are the scripts for installing Zabbix software package components.

The database creation, the server installation and configuration, the front-end installation and configuration are all implemented as different scripts for improving flexibility and allowing deployments different from the one implemented here.

Script zabbix-agent_setup.sh is supposed to be used on nodes that are under Zabbix monitoring.

Name	zabbix-update.sh
Languages	Bash Shell
Description	It adds the Zabbix code repository to the list of code repositories to get code from when submitting "apt-get install" commands.
Input	None
Output	None

Name	zabbix-database_setup.sh
Languages	Bash Shell
Description	It creates the MySQL database for storing Zabbix data
Input	__DB_Host__: IP address of the MySQL database service __DB_Database__: name of the MySQL database __DB_User__: username of the MySQL account __DB_Password__: password of the MySQL account
Output	None

Name	zabbix-server_setup.sh
Languages	Bash Shell

Description	It installs and configures the Zabbix server software package. It requires the credentials for accessing an already deployed and configured MySQL service
Input	__DB_Host__: IP address of the MySQL database service __DB_Database__: name of the MySQL database __DB_User__: username of the MySQL account __DB_Password__: password of the MySQL account
Output	None

Name	zabbix-frontend_setup.sh
Languages	Bash Shell
Description	It installs and configures the Zabbix web front-end software package. It requires the credentials for accessing an already deployed and configured MySQL service
Input	__DB_Host__: IP address of the MySQL database service __DB_Database__: name of the MySQL database __DB_User__: username of the MySQL account __DB_Password__: password of the MySQL account
Output	None

Name	zabbix-agent_setup.sh
Languages	Bash Shell
Description	It installs and configures Zabbix agent for allowing Zabbix server to monitor a VM
Input	__Zabbix_Server_IP__: IP address of the Zabbix server
Output	None

2.6 CloudFoundry

This section describes the scripts for installing and configuring CloudFoundry.

As described above, CloudFoundry is actually deployed by BOSH, therefore we need just a single script for preparing the BOSHNode.

Name	cloudfoundry_boshsetup.sh
Languages	Bash Shell
Description	It installs, configures and runs the BOSH software packages. After that, it calls the scripts for deploying CloudFoundry. As BOSH directly interacts with OpenStack API, the script requires the credentials for accessing OpenStack IaaS services.
Input	__Service_IP__: IP address where the CloudFoundry services are available __OS_USERNAME__: OpenStack account username __OS_PASSWORD__: OpenStack account password __OS_TENANT_NAME__: OpenStack tenant name __OS_AUTH_URL__: OpenStack authentication service URL
Output	None

2.7 Cluster

The following scripts install and configure the software for implementing an active-standby cluster implemented using Pacemaker and Corosync.

Some scripts (namely cluster-node_initialize.sh, cluster_install.sh and cluster_quorum_standby.sh) are reusable independently of the services hosted by the cluster; others (dbcluster_master_configure.sh and zabbixcluster_master_configure.sh) are specifically designed for the function (i.e. services) delivered by the cluster.

The scripts presented in this section have been implemented adapting examples found on the Internet ([11], [12] and [13]).

Name	cluster-node_initialize.sh
Languages	Bash Shell
Description	It performs a basic active-standby cluster node initialization.

Input	__Node1_IP__: IP address of node 1 (master node) __Node2_IP__: IP address of node 2 (slave node) __NodeQ_IP__: IP address of quorum node __Virtual_IP__: Virtual IP address where the cluster services are available __Node1_Name__: name of node 1 __Node2_Name__: name of node 2 __NodeQ_Name__: name of quorum node
Output	None

Name	cluster_install.sh
Languages	Bash Shell
Description	It installs and configures the cluster management software (Pacemaker and Corosync).
Input	__Cluster_Subnet__: the subnet
Output	None

Name	cluster_quorum_standby.sh
Languages	Bash Shell
Description	It switches the quorum node to standby mode
Input	None
Output	None

Name	db_cluster_master_configure.sh
Languages	Bash Shell
Description	It configures the database active-standby cluster defining the controlled resources (services), the dependencies and the colocation. It also promotes the calling node to active state.
Input	__Virtual_IP__: Virtual IP address where the cluster services are available __DB_DRBD_Device__: name of the volume __DB_DRBD_Resource__: name of the DRBD resource __DB_Directory__: name of the DB directory
Output	None

Name	zabbix_cluster_master_configure.sh
Languages	Bash Shell
Description	It configures the Zabbix active-standby cluster defining the controlled resources (services), the dependencies and the colocation. It also promotes the calling node to active (or master).
Input	__Virtual_IP__: Virtual IP address where the cluster services are available
Output	None

2.8 Miscellaneous Scripts

The following scripts implement some general purpose functions and can be reused for the deployment of the applications hosted by SCP.

Name	aptget-update.sh
Languages	Bash Shell
Description	It downloads the package lists from the repositories and "updates" them to get information on the newest versions of packages and their dependencies.
Input	None
Output	None

Name	openstack-cli_install.sh
Languages	Bash Shell
Description	It installs all the OpenStack CLI packages for allowing the VM to interoperate with the hosting OpenStack cloud, if needed. As an example, Duplicity uses “python-swiftclient” package for saving back-up files in the Object Storage Service (see below)

Input	None
Output	None

Name	duplicity_setup.sh
Languages	Bash Shell
Description	It installs all the software components required for performing back-up using Duplicity and store the files in the OpenStack Object Storage. It requires the credentials for accessing the underlying OpenStack IaaS cloud in order to access the Object Storage Service and save the back-up files.
Input	__OS_USERNAME__: OpenStack account username __OS_PASSWORD__: OpenStack account password __OS_TENANT_NAME__: OpenStack tenant name __OS_AUTH_URL__: OpenStack authentication service URL
Output	None

3 Summary and Conclusions

This document has summarized the current state of the art of the Storm Cloud Platform (SCP) and described, mostly from a technical point of view, the implementation of the services built on top of OpenStack IaaS, collectively named over-cloud services. Over-cloud services are shared functions used for facilitating the deployment of application services in a production-ready environment where non-functional yet fundamental aspects like high availability, scalability and application management in general need to be addressed.

Database Layer implements database engines in high availability configuration, Monitoring Layer provides services for monitoring applications, PaaS platform facilitates the deployment of web-based applications (e.g. the programmer is not required to deal with infrastructural details such as VMs and is free to focuses on the implementation of the application software.

Most of the effort has been focused on automating the deployment of the over-cloud services and implementing reusable software for that purpose (HOT and Bash shell scripts). This document describes the technical details of the software to date but, as is common practice in software development, we expect to modify the software components in order to add new (unplanned but useful) features or refactor the code for improving the modularity and the reusability. For this reason, this document will be updated accordingly in case of significant modifications to the software resulting in new issues. Although new issues of the document were not originally planned [1], we think that this document should always reflect the state-of-art of the platform, show the technical evolution and capture the lessons learnt in the implementation.

References

[1] "Surfing Towards the Opportunity of Real Migration to CLOUD-based public Services," STORM CLOUDS Consortium, November 2013.

[2] "D2.3.1 Storm Clouds Platform Implementation Status Report," STORM CLOUDS Consortium, 2014.

[3] "D2.3.2 Storm Clouds Platform Implementation Status Report," STORM CLOUDS Consortium, 2015.

[4] "D2.2.2 - Storm Clouds Platform Architectural Design," STORM CLOUDS Consortium, 2015.

[5] "OpenStack Heat - Wiki Page," [Online]. Available: https://wiki.openstack.org/wiki/Heat. [Accessed Jan 2015].

[6] "D2.4.2 Cloud Application Template Catalogue," STORM CLOUDS Consortium, 2015.

[7] "Ubuntu 14.04.2 LTS (Trusty Tahr)," [Online]. Available: http://releases.ubuntu.com/14.04/. [Accessed June 2015].

[8] "RFC1918 - Address Allocation for Private Internets," The Internet Engineering Task Force (IETF®), Feb 1996. [Online]. Available: https://www.ietf.org/rfc/rfc1918.txt. [Accessed June 2015].

[9] "Cloud Foundry - Foundation Page," [Online]. Available: http://www.cloudfoundry.org/cloud-foundry-foundation-launch.html. [Accessed Jan 2015].

[10] "Cloud Foundry - BOSH Page," [Online]. Available: http://docs.cloudfoundry.org/bosh/. [Accessed Jan 2015].

[11] T. Shaun , "High Availability with PostgreSQL and Pacemaker," 2012. [Online]. Available: https://wiki.postgresql.org/images/0/07/Ha_postgres.pdf. [Accessed May 2015].

[12] "MySQL High Availability: Configuration and Deployment Guide," Oracle, September 2012. [Online].

[13] "High Availability setup for Zabbix 1.8 - 2.4," [Online]. Available: https://www.zabbix.org/wiki/Docs/howto/high_availability. [Accessed June 2015].