Project Acronym: STORM CLOUDS

Grant Agreement number: 621089

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

Deliverable D2.2.1

Storm Clouds Platform Architectural Design

Work Package: WP2

Version: 1.0

Date: 30/07/2014

Status: Project Coordinator Accepted

Dissemination Level: PUBLIC

Authoring

Role	Name	Organisation
Edited by	Marco Consonni	Hewlett Packard Italiana
Author	Marco Consonni	Hewlett Packard Italiana
Author	Andrea Milani	Hewlett Packard Italiana
Reviewed by	Alkiviadis Giannakoulias	European Dynamics
Reviewed by	Agustín González Quel	Ariadna Servicios Informáticos

Version Control

Modified by	Date	Version	Comments
M. Consonni	04/07/2014	0.1	First Draft
M. Consonni, A. Milani	18/07/2014	0.2	Ready for review
M. Consonni	30/07/2014	1.0	Ready for EUC review

Project Presentation

Surfing Towards the Opportunity of Real Migration to Cloud-based public Services (STORM CLOUDS) [1] 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) [2].

The project has the objective of exploring the shift to a cloud-based paradigm for deploying services that Public Authorities (PAs) currently provide using more traditional Information Technology (IT) deployment models. In this context, the term "services" refers to applications, usually made available through Internet, that citizens and/or public servants use for accomplishing some valuable task.

The project aims to define useful guidelines on how to implement the process of moving application to cloud computing and is based on direct experimentation with pilot projects conducted in, at least, the cities participating to the consortium.

STORM CLOUDS will also deliver a consolidated a portfolio of cloud-based services validated by citizens and Public Authorities in different cities and, at the same time, general and interoperable enough to be transferred and deployed in other European cities not taking part in the project. This portfolio will be mainly created from applications and technologies delivered by other CIP Policy Support Program (CIP-PSP) and Framework Program 7 (FP7) projects, as well as resulting from innovation efforts from Small and Medium Enterprises (SMEs).

The project is lead by the following consortium:

Member	Role/Responsibilities	Short Name	Country
Ariadna Servicios Informáticos, S.L.	Co-ordinator	ASI	Spain
Hewlett Packard Italiana S.r.l.	Participant	HP	Italy
EUROPEAN DYNAMICS Advanced Systems of Telecommunications, Informatics and Telematics	Participant	ED	Greece
Research, Technology Development and Innovation, S.L	Participant	RTDI	Spain
Aristotelio Panepistimio Thessaloniki	Participant	AUTH	Greece
Alfamicro Sistemas de Computadores LDA	Participant	Alfamicro	Portugal
Manchester City Council	Participant	Manchester	United Kingdom
Ayuntamiento de Valladolid	Participant	Valladolid	Spain
City of Thessaloniki	Participant	Thessaloniki	Greece
Câmara Municipal de Águeda	Participant	Águeda	Portugal

For more information on the scope and objectives of the project, please refer to the Description of Work (DOW) of the project [3].

Executive Summary

Work Package 2 (WP2) of the Storm Clouds project is aimed at designing and implementing the reference architecture for the Storm Clouds Platform (SCP), the cloud platform infrastructure for hosting application services selected for being ported to cloud. SCP supplies computational resources that are allocated/de-allocated on a on-demand basis following a "as-a-Service" cloud computing paradigm.

This document describes the SCP architecture providing the technical details for the implementation; it shows the main modules, what functions they implement, how they interact and what are the software products selected for the actual realization.

The architecture described here is used for realizing an actual implementation of the platform, made available to the STORM CLOUDS consortium for deploying their applications during the project lifetime. However, SCP architecture can also be used as a reference for implementing similar services on other contexts. Some project participants (e.g. municipalities) might require that applications run in a different, maybe "more controlled", environment implemented in data centres on their premises. This might be preferred and/or strictly required by security and privacy regulations applicable to the information managed by the applications. In such a case, SCP design can be totally or partially reused for the implementation of a private SCP instance.

The description of the architecture is mainly focussed on the software components used for the implementation; the hardware components are intentionally not fully described assuming that the solution can be supported by commodity hardware equipment. In addition, a selection of the hardware necessary for supporting a SCP instance would require an in-depth analysis of the operational, security, financial and performance requirements that are, at the time of writing, out of scope.

According to project objectives defined in the project Statement of Work document [3], "*the project partners declare strong commitment to an open approach, and this will be heavily considered when selecting concrete services to deploy during the project and include in STORM CLOUDS portfolio*"; in this perspective, SCP architecture is based on software solutions and components backed by open software license.

SCP architecture is meant to address the requirements and specification described in a previous STORM CLOUDS project deliverable titled "D 2.1 Storm Clouds Platform – Requirements and Specification"[4].

This document is organized in the following major sections.

Section 1 describes the SCP architecture showing a first-cut decomposition in layers that are detailed in the subsequent sections.

Section 2 describes the software technologies used to implement basic computational resources like virtual machines, virtual disks, virtual networks, etc.

Section 3 details the software technologies used to implement an operating environment for hosting web-based applications.

Section 4 describes the software solutions implementing database services.

Section 5 shows the solutions used for implementing management and maintenance functions for the cloud platform.

Section 6 draws a summary and conclusions.

The document is the first of a document series that will eventually result in the full definition of the SCP architecture. In this first issue the objective is to define the basic services for enabling the porting of the applications; some more advanced aspects regarding a "production ready environment" (e.g. high availability issues, cloud platform management, monitoring, etc.) are partially addressed, only. They will be fully investigated, described and addressed in the next document releases.

Table of Contents

List of Figures

List of Tables

Abbreviations

Acronym	Description
API	Application Programming Interface
CLI	Command Line Interface
DB	Database
DBMS	DataBase Management System
DEA	Droplet Execution Agents
DNS	Domain Name System
DRDB	Distributed Replicated Block Device
DHCP	Dynamic Host Configuration Protocol
ERP	Enterprise Resource Planning
GIS	Geographic Information System
GRE	Generic Routing Encapsulation
GTFS	General Transit Feed Specification
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IP	Internet Protocol
iSCSI	internet SCSI
KML	Keyhole Mark-up Language
KVM	Kernel-based Virtual Machine
IaaS	Infrastructure as a Service
IT	Information Technology
L2	Layer 2 (networking)
L3	Layer 3 (networking)
LXC	LinuX Containers
NAT	Network Address Translation
NIST	National Institute of Standards and Technology
N/A	Not Available
PC	Personal Computer
PaaS	Platform as a Service
QEMU	Quick EMUlator
RAID	Redundant Array of Independent Disks
REST	REpresentational State Transfer
RSS	Really Simple Syndication
SCP	Storm Clouds Platform
SaaS	Software as a Service
SCSI	Small Computer System Interface

Acronym	Description
SSL	Secure Socket Layer
TBC	To Be Confirmed
TBD	To Be Defined
TCP-IP	Transmission Control Protocol – Internet Protocol (suite)
URL	Uniform Resource Locator
VM	Virtual Machine
VPN	Virtual Private Network
WLAN	Wireless Local Area Network

1. Overall Architecture

The following diagram shows the logical view of the Storm Clouds Platform (SCP) architecture:

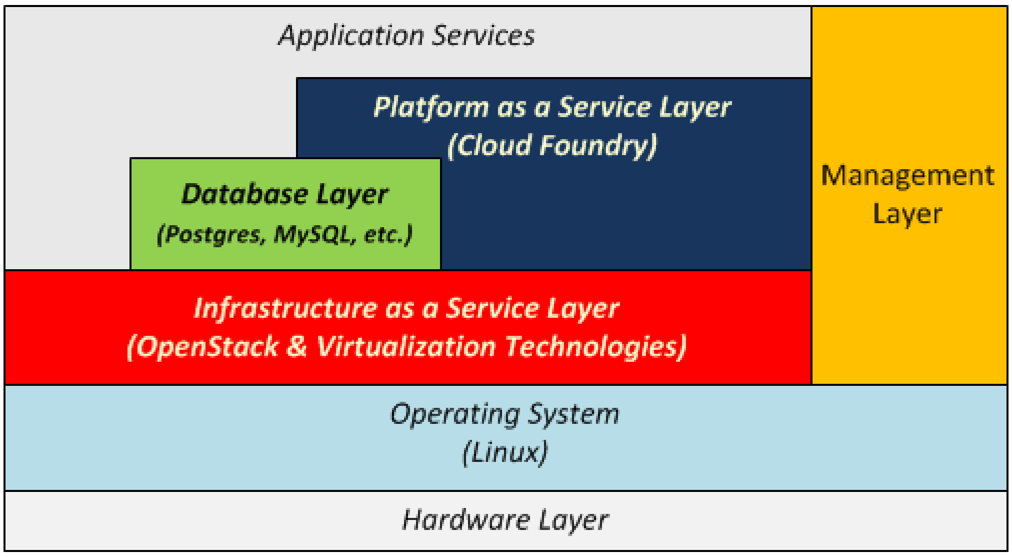
Application Services
Platform as a Service Layer (Cloud Foundry)
Database Layer (Postgres, MySQL, etc.)
Management Layer
Infrastructure as a Service Layer (OpenStack & Virtualization Technologies)
Operating System (Linux)
Hardware Layer

Figure 1-1 - Storm Clouds Platform - Overall Architecture

The Hardware Layer represents the physical equipment where the platform is hosted. It is composed of servers, network connections and equipment, storage devices, etc. This layer is implemented by the hardware of the data centre(s) where the platform is hosted.

Servers are equipped with any Linux distribution supporting OpenStack like Debian 7.0, openSUSE, SUSE Linux Enterprise Server, Red Hat Enterprise Linux, CentOS, Fedora and Ubuntu 12.04/14.04 (LTS).

OpenStack® [5] implements the Infrastructure as a Service Layer that provides services for creating and managing virtual computing resources in the cloud; for such purpose it uses some virtualization technologies that will be described below in this document.

Cloud Foundry™ [6] implements the Platform as a Service Layer. It is worth noting that in SCP architecture the PaaS Layer is built on top of the IaaS Layer. This means that the servers of the PaaS are actually implemented as VMs running in the IaaS cloud. This implies several benefits in terms of flexibility as will be explained subsequently.

The Database Layer provides the applications running in the cloud with installations of database engines; developers can use them for implementing and/or deploying their applications without taking care of the maintenance that is under the cloud provider responsibility. At the time of writing, the Database Layer supports **MySQL™** [7] and **PostgreSQL** [8] database engines.

The Management Layer implements functions for managing and maintaining the cloud platform and is mainly designed for the cloud provider. At the time of writing this component is still to be designed but it will be implemented using open software solutions like **Zabbix** [9] or **Nagios** [10] both available under GNU General Public License (GPL) version 2.

Application Services collectively represents the applications hosted in the SCP. They are not part of the SCP architecture but use the services the architecture provides for running

It is worth noting that the proposed architecture covers several layers, with increasing levels of abstraction, because, in addition to defining the SCP platform, it aims to provide a reference framework that stakeholders may use beyond the STORM project to implement their own cloud platform. For example, organizations that require full control over the infrastructure may implement only the IaaS layer and not use any PaaS technology, while others may hide the IaaS layer completely and expose only the PaaS platform to their developers, in order to standardize deployments and abstract away infrastructure complexity. Others may adopt a hybrid approach and deploy standard applications on the PaaS and application with special requirements on the IaaS. In addition to this, all technologies included in the architecture are open source, which means they can be implemented both on-premises or on public cloud providers, depending on requirements such as security, data protection and cost trade-offs.

The following sections describe the architecture layers in greater details.

2. Infrastructure as a Service Layer

2.1. Concept

The Infrastructure as a Service (IaaS) layer provides basic IT capabilities that can be briefly described as follows:

- **computation services**: the ability to start Virtual Machines (VMs) running an operating system and, optionally, application software;
- **storage services**: ability to create storage elements (virtual disks or files);
- **networking services**: ability to create network elements like Layer 2 (L2) networks, subnets, DHCP services, etc.

Generally speaking, when applications are deployed in a cloud environment, the IaaS layer provides the run-time environment for the execution. The following, figure summarizes the concept:

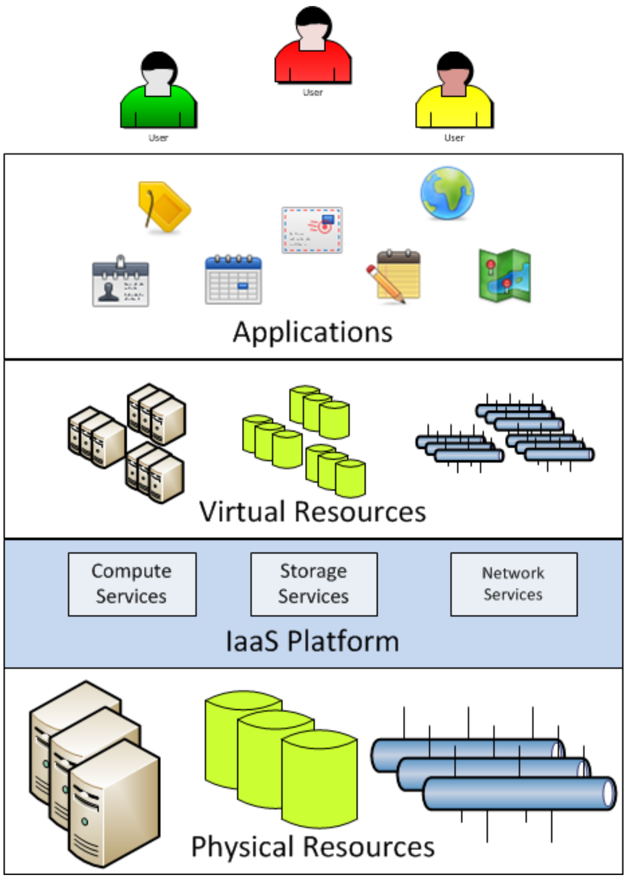

Figure 2-1 - Infrastructure as a Service - Conceptual View

At the bottom of the stack, physical resources like physical machines, physical disks and physical networks provide the actual environment for supporting the computational workload.

The IaaS platform is a software layer providing services for creating virtual resources (virtual machines, virtual disks and virtual networks) that can be used instead of their physical counterparts in order to deploy and run applications. Virtual resources are provided as services; this means that they are created when needed, used to run applications and removed when the application is not anymore needed. The actual computation happens 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.

The IaaS platform provides resources using virtualization, a set of technologies aimed to simulate the existence of a piece of hardware which is “materialized” by a software layer running on top of the physical devices. The idea is that the actual hardware is hidden to the applications and partially or temporarily used for “impersonating” the role of a virtual piece of similar hardware.

The target users of the IaaS Layer are both the cloud provider and the application developers. The cloud provider uses IaaS Layer services for implementing higher layer services: s/he uses virtual machines for running the PaaS Layer software solution or the database engines in the Database Layer.

The application developers, on the other hand, can use the IaaS Platform service in those cases where the applications (or part of them) cannot be hosted in the PaaS Layer.

2.2. Logical View

The following picture shows the logical architecture of the Infrastructure as a Service Layer:

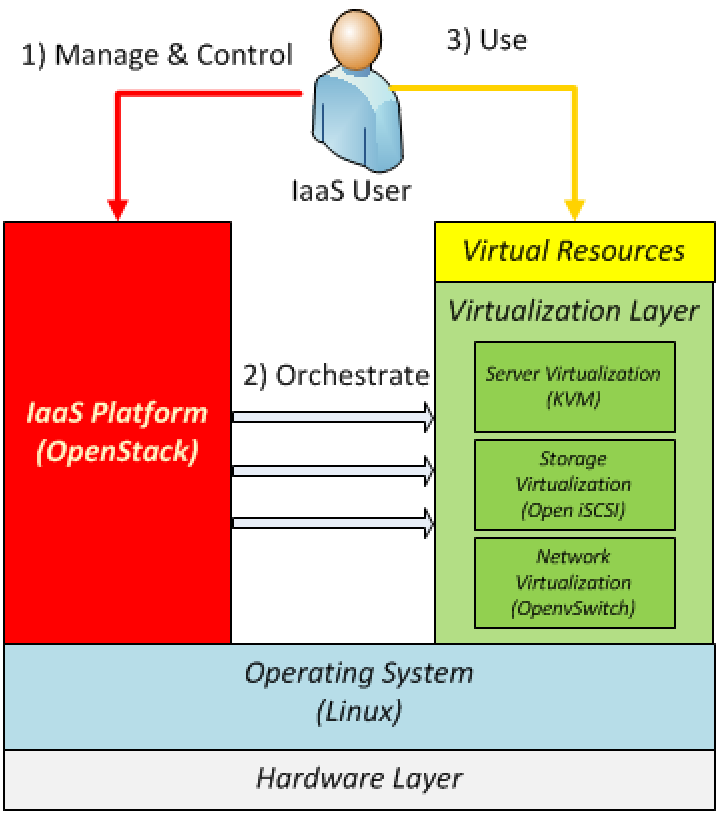
1) Manage & Control
3) Use
IaaS User
Virtual Resources
Virtualization Layer
2) Orchestrate
IaaS Platform (OpenStack)
Server Virtualization (KVM)
Storage Virtualization (Open iSCSI)
Network Virtualization (OpenvSwitch)
Operating System (Linux)
Hardware Layer

Figure 2-2 - Infrastructure as a Service - Logical View

The Hardware Layer represents the physical servers used for deploying the IaaS Layer. They are equipped with a Linux operating system[1], hosting both the Virtualization Layer and the IaaS Platform software packages.

When a user wants to manage and/or control virtual computational resources, s/he interacts with the IaaS Platform requiring a service via an API call (red flow in the picture). The IaaS Platform translates the high level API call into lower level instructions for the Virtualization Layer in order to orchestrate all the actions required for providing the service (grey flow). When the service request is fulfilled, the user can directly interact with the Virtualization Layer for using the virtual resource (yellow flow).

For instance, when a user wants to create a virtual machine, s/he submits a "virtual machine creation request" to the IaaS platform through a suitable API call. The IaaS Platform decides where (i.e. on what server) the virtual machine can be started and interoperates with the Virtualization Layer for creating all the virtual resources for fulfilling the request. It contacts the Storage Virtualization software for retrieving the image of the virtual machine to boot, the Server Virtualization software (i.e. the Hypervisor) for activating a new virtual machine, the Network Virtualization software for connecting the virtual machine to the network, etc.

[1] It can be any Linux distribution supporting both OpenStack and the virtualization technologies mentioned in this document. At the time of writing, the supported platforms are: Debian 7.0, openSUSE and SUSE Linux Enterprise Server, Red Hat Enterprise Linux, CentOS, Fedora and Ubuntu 12.04/14.04 LTS. For more information, see http://docs.openstack.org/

Finally, when the virtual machine is started, the user directly operates on it the same way s/he does with a physical machine.

The Infrastructure as a Service Layer is the combination of the Operating System, the Virtualization Layer and the IaaS Platform.

2.2.1. IaaS Platform

OpenStack is the IaaS Platform selected for the implementation of the SCP.

In addition to being an IaaS platform, OpenStack is also a project and a community backed by an independent foundation supported by several corporate sponsors like Hewlett Packard, IBM, Cisco, Ericsson, Intel, AT&T, Redhat, RackSpace, etc.. All source code is freely available under the Apache 2.0 license (see [5]).

OpenStack provides a documented and open Application Procedure Interface (API). This is a very important aspect in general - and for the project in particular - in order to fulfil the requirements stating that the whole solution shall be implemented with open software products.

The following picture shows the OpenStack high-level logical architecture:

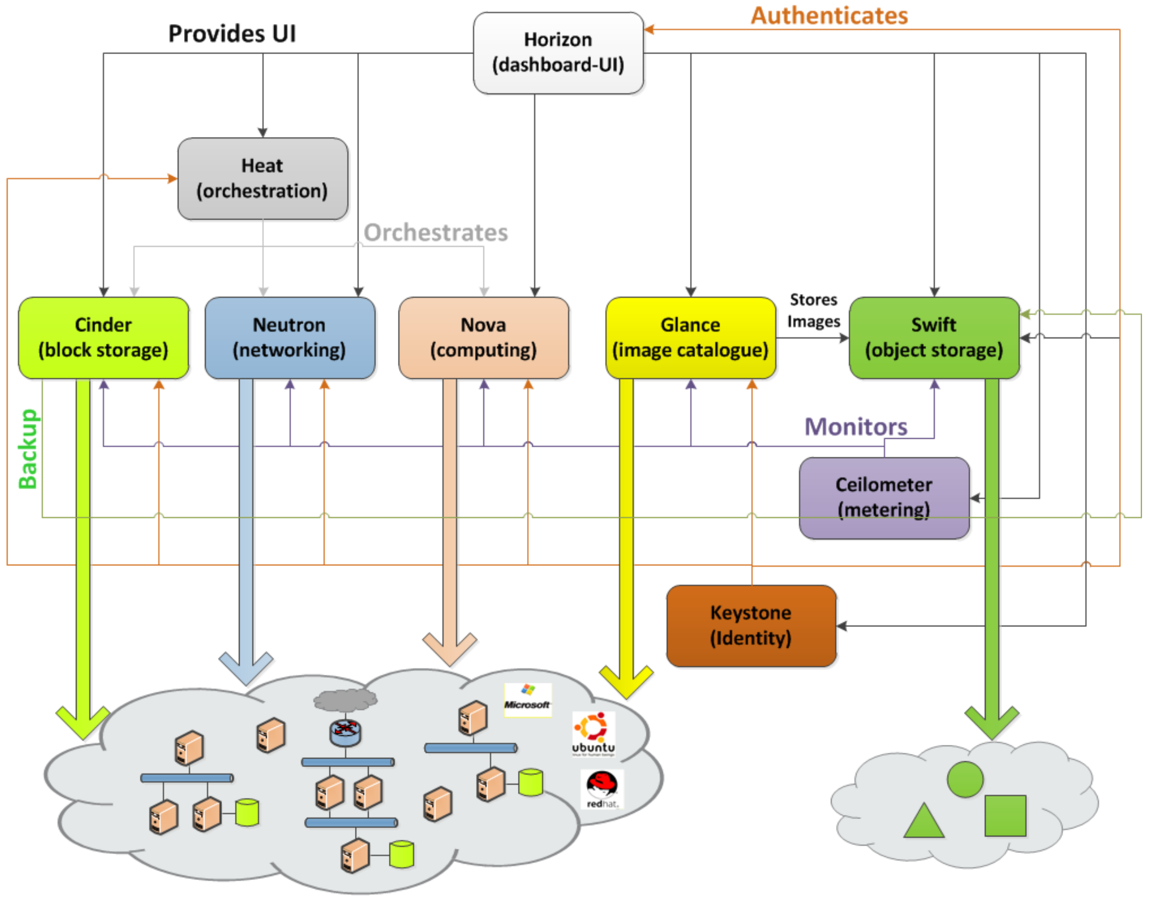

Figure 2-3 - OpenStack Logical Architecture

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;
- **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 aggregations described by script files called “stacks”;
- **Ceilometer** monitors the usage of resources for metering and accounting purposes.

Generally speaking, OpenStack implements storage services for storing data in the cloud but they can be further classified as follows:

- **block storage services**: ability to manage virtual disks to connect to VMs running in the cloud (these services are implemented by Cinder);
- **object storage services**: ability to manage files (i.e. objects) to be stored in the cloud (these services are implemented by Swift).

It is worth noticing that while block storage services are mainly focussed on providing storage resources for the VMs running in the same cloud where the storage is hosted, object storage services implement the ability to store data coming either from VMs in the cloud or by any other application/device running outside the cloud.

More in details block storage services create and manipulate virtual disks used by the VMs running in the cloud: the cloud user can create virtual disks and attach/detach them to VMs for storing data. Object storage services, on the other hand, provide the cloud users with the ability of uploading/downloading files in the cloud. The files can even come from a client application running on a physical device outside the cloud, for example a mobile phone in the hand of the user. In this case, the application running outside the cloud directly interoperates with the object storage services calling a suitable API.

OpenStack modules communicate with each other using a message broker middleware (e.g. RabbitMQ) and store status information in a centralized database (e.g. MySQL); for the sake of brevity, these elements are not shown in the logical architecture.

OpenStack's modules are broken down into sub-modules, implemented as Python programs, usually installed as Linux services (i.e. daemons) on the physical servers of the data centre.

The following table describes the modules and the related sub-modules:

Module	**Sub-Module**	**Type**	**Description**
Keystone	keystone-all	API/Mediator	Implements API end-point and the logic for authenticating cloud users
	keystone	CLI (U)	Submits commands for managing users, tenants, roles, etc.
	keystone-manage	CLI (A)	Submits administrative commands for managing the keystone module
Glance	glance-api	API	Accepts API calls for image management
	glance-registry	Mediator	Manages image metadata (e.g. size, type)
	glance	CLI (U)	Submits commands for managing virtual machines images
	glance-manage	CLI (A)	Submits administrative commands for managing the glance module
Nova	nova-api	API	Implements API for managing computational resources (i.e. VMs)
	nova-scheduler	Mediator	Determines on which node a virtual machine should run
	nova-conductor	Mediator	Mediates interactions between nova-compute and the cloud database
	nova-cert	Mediator	Manages x509 certificates (only needed for EC2 API)
	nova-consoleauth	Mediator	Authorizes tokens that console proxies provide. See nova-novncproxy
	nova-novncproxy	Mediator	Proxy for accessing running instances through a VNC connection
	nova-compute	Agent	Manages VM instances through hypervisor API
	nova	CLI (U)	Submits commands for managing computational resources (VMs)
	nova-manage	CLI (A)	Submits administrative commands for managing the nova module
Cinder	cinder-api	API	Implements API for managing block storage resources (i.e. virtual disks)

Module	Sub-Module	Type	Description
	cinder-scheduler	Mediator	Determines on which node a virtual disk should reside
	cinder-volume	Agent	Manages virtual disks through the API provided by the block-storage virtualization provider
	cinder	CLI (U)	Submits commands for managing virtual disks
	cinder-manage	CLI (A)	Submits administration commands for managing the cinder module
Neutron	neutron-server	API/Mediator	Accepts API calls for virtual network resources management
	neutron-dhcp-agent	Agent	Distributes IP addresses to VMs in collaboration with dnsmasq [11], an external DHCP server
	neutron-metadata-agent	Agent	Provides VMs with a HTTP end-point for retrieving metadata
	neutron-l3-agent	Agent	Implements L3/NAT forwarding to provide external network access for VMs running in the cloud (see also [12])
	neutron-plugin-openvswitch-agent	Agent	Performs configurations for creating virtual L2 trunks and for connecting VMs to them
	neutron	CLI (U)	Submits commands for managing virtual network objects like L2 trunks, ports, subnets, virtual routers, etc.
	neutron-manage	CLI (A)	Submits administrative commands for managing the neutron module
Horizon	openstack-dashboard	GUI	It's a django [13] Python application running as an Apache [14] HTTP server application. It implements a web-based user interface for submitting IaaS requests
Swift	swift-proxy-server	API/Mediator	Implements API for managing object storage resources, like files and containers, and the related metadata
	swift-object swift-object-replicator swift-object-updater swift-object-auditor	Agent/Mediator	Manage actual objects (files) on the storage nodes.
	swift-container swift-container-replicator swift-container-updater swift-container-auditor	Agent/Mediator	Manage a mapping of containers, or folders, within the Object Storage service.
	swift-account swift-account-replicator swift-account-reaper swift-account-auditor	Agent/Mediator	Manages accounts defined with the Object Storage service.
	swift	CLI (U)	Submits commands for managing object storage resources and related metadata
Heat	heat-api	API	Accepts API calls for orchestration services
	heat-engine	Mediator	Create virtual resources described into templates by orchestrating API calls
	heat	CLI (U)	Submits commands for creating virtual resource "stacks"
	heat-manage	CLI (A)	Submits administrative commands for managing the heat module
Ceilometer	ceilometer-api	API	Accepts API calls for telemetry services
	ceilometer-agent-central	Mediator	Polls for resource utilization statistics and for resources not tied to instances or compute nodes

Module	Sub-Module	Type	Description
	ceilometer-alarm-notifier	Mediator	Allows setting alarms based on threshold evaluation for a collection of samples
	ceilometer-collector	Mediator	Monitors the message queues (for notifications and for metering data coming from the agent) and turns them into metering messages
	ceilometer-agent-compute	Agent	Polls for resource utilization statistics on the hosting compute node
	ceilometer	CLI (U)	Submits commands for using telemetry services

Table 2-1- OpenStack Modules and Sub-Modules

The table classifies the sub-modules in types:

- **API** - Application Program Interface front-end provides an API end-point that cloud user can call for submitting IaaS requests. API modules are installed as Linux daemons and implement a RESTful [15] interface.
- **Mediator** – Mediators are daemons coordinating and orchestrating the actions for fulfilling API requests.
- **CLI (U)**: Command Line Interpreters for cloud Users are programs, usually running on users' client machine, for submitting API requests. They are implemented as Python programs callable from the shell of the cloud user's machine. CLI (U) programs submit API requests to API daemons.
- **CLI (A)**: Command Line Intepreters for cloud Administrators are programs, exclusevely running on servers implementing the cloud (cloud nodes), that allow the cloud administrator to perform administrative tasks like showing logged error messages, migrating/synchronizing the cloud database, etc;
- **GUI**: Graphical User Interface provides a point-and-click interface for using infrastructure services (e.g. starting/stopping VMs, creating/attaching/deleting virtual disks, etc.);
- **Agent**: This type of sub-module, implemented as a daemon, is responsible for directly interacting with the virtualization layer and managing the virtualized resources (e.g. VMs). For this reason, agent sub-modules are directly deployed on the nodes that provide the virtual resources and interoperate with the virtualization layer. For instance, nova-compute (the nova module agent), is deployed on the nodes dedicated to run virtual machines and interoperate with the hypervisor installed on such nodes.

The classification above helps the design of the OpenStack deployment; in fact, although in principle every OpenStack sub-module can run on a dedicated node, usually several sub-models are co-located on a single node or set of nodes.

API sub-modules are deployed on nodes that can be accessed by any cloud user. This arrangement makes it possible to require services via API calls. In case OpenStack is used for implementing a public cloud platform, these nodes must be accessible from Internet.

Mediator modules are deployed on nodes that cannot be accessed by cloud users because they implement the internal logics of the IaaS cloud platform.

CLI (U) sub-modules are deployed on any machine that's being used for submitting API calls to OpenStack. These machines can be any node in the cloud but also any client machine running outside the cloud. For instance, a cloud user can deploy these components on her PC in order to being able to submit API calls to OpenStack.

CLI (A) modules are deployed on nodes that cannot be accessed by cloud users but need to be accessed by the cloud administrator(s); they implement administration commands submitted by administrators, only. These sub-modules are usually co-located on the same nodes hosting mediators.

GUI sub-module, being implemented as a web applications running under Apache HTTP server, is deployed on machine accessible by the cloud users.

Agent sub-modules are deployed on nodes that make available the computational resources. For example, nova-compute (the nova agent sub-module) is deployed on all the nodes running the hypervisors and hosting the virtual machines activated in the cloud. Similarly, cinder-volume (the cinder agent) is deployed on any node providing block storage resources (i.e. virtual disks).

Note that some Swift sub-modules are necessarily co-located because they collectively deliver the management of a certain type of object on a node. For instance, swift-object* sub-modules collectively manage files stored into a node of a cloud: swift-object sub-module fulfils object retrieval/store requests, swift-object-replicator replicates objects in the cloud (according to the configured replication policies), swift-object-updater updates objects (e.g. deletes objects) and swift-object-auditor reports information about the received requests in the log file for auditing purposes.

The actual configuration of OpenStack is a quite complex and elaborated process that is out of the scope of this document. OpenStack sub-modules are configured by editing a set of suitable configuration files managed by the cloud administrator. Annex A reports an example. For more information about the OpenStack configuration, please see the OpenStack community documentation page [16].

2.2.2. Virtualization Layer

This section briefly describes the virtualization technologies used for implementing the SCP.

It is worth noticing that OpenStack supports several and, in some cases, alternative virtualization technologies: for instance, for delivering server virtualization services, OpenStack supports various hypervisors like Kernel-based Virtual Machine (KVM), XenServer/XCP, Hyper-V, VMware vSphere, etc.

This document selects a set of virtualization technologies that form an integral part of the SCP architecture. The selection is made according to the following criteria:

- **Licensing**: the technologies must be backed by some open software license;
- **Functionality**: the technology must support most of the functions made available by OpenStack;
- **Documentation:** the technology in itself and the integration with OpenStack must be well documented and the documentation must be freely available;
- **Minimal Software Requirements**: the technology shall not require special or specific additional software and must run on the selected operating system;
- **Minimal Hardware Requirements**: the technology shall not require special or specific hardware features or devices for operating.

2.2.2.1. Server Virtualization

Server virtualization is the technology for creating and managing Virtual Machines: this kind of technology is implemented by hypervisors.

OpenStack supports several hypervisors, as reported in the OpenStack Hypervisor Support Matrix [17].

The hypervisor selected for SCP is Kernel-based Virtual Machine (KVM) [18]; the following table shows how KVM meets the selection criteria:

Criteria	**Notes**
Licensing	According to [19], "*KVM's parts are licensed under various GNU licenses:* • *KVM kernel module: GPL v2* • *KVM user module: LGPL v2* • *QEMU virtual CPU core library (libqemu.a) and QEMU PC system emulator: LGPL* • *Linux user mode QEMU emulator: GPL* • *BIOS files (bios.bin, vgabios.bin and vgabios-cirrus.bin): LGPL v2 or later*"
Functionality	According to [17], KVM and QEMU score the higher level of supported OpenStack functions.
Documentation	KVM documentation is freely available (see [18]). In addition, the integration with OpenStack and the installation procedures are fully documented and freely available (see [16])
Software Requirements	According to [18] (see *Status* section), "*KVM is included in the mainline linux kernel since 2.6.20...*" and "*It is also available as a patch for recent Linux kernel versions and as an external module that can be used with your favourite distro- provided kernel going back up to 2.6.16, therefore including all latest versions for Enterprise Linux Distributions.*" For this reason, KVM meets the minimal software requirements because Linux is the Operating System selected for implementing SCP.
Hardware Requirements	According to [18] (see *Status* section), KVM runs on several hardware architectures like generic Intel®-based hosts and AMD-based hosts. The only requirement for KVM to

Criteria	Notes
	work is that the processors implement the virtualization technology (VT for Intel and SV for AMD). This feature is normally implemented on processors used into typical server and desktop machines.

Table 2-2- KVM Selection Criteria

2.2.2.2. Storage Virtualization

Storage virtualization is the technology for creating and managing block storage elements (i.e. virtual disks) used as persistent disks by the virtual machines.

OpenStack supports several storage virtualization technologies, as reported in the OpenStack Cinder Support Matrix [20].

When selecting a block storage virtualization technology, in addition to the criteria defined in section 3.2.2, we have also to take into account the type of protocol used by the VMs to communicate with the virtual disks. In fact, normally VMs and virtual disks are hosted (or can be hosted) on different nodes; therefore when a VM reads/writes data from/to virtual disks there must be a communication between the node hosting the VM and the node hosting the virtual disk. This communication happens according to a protocol that must be supported both by the virtualization technology managing virtual machines (i.e. the hypervisor) and the virtualization technology managing virtual disks.

The following diagram summarizes the concept:

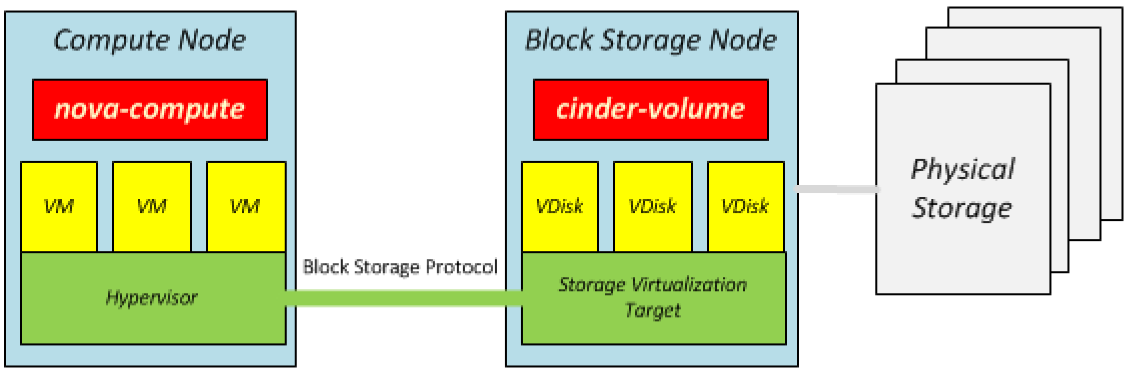

Figure 2-4 - Block Storage Virtualization

The picture also shows that, through virtualization, the VMs are completely "unaware" of what kind of physical storage actually hosts the virtual disks (VDisks, in the picture). Supported solutions range from local disks, directly connected to the Block Storage Node mother board through SCSI bus, to dedicated storage appliances.

The technologies selected for implementing the SCP block storage virtualization are:

- **Storage Virtualization Target**: Linux SCSI target framework [21], known as tgt, is a software package installed on a Linux-based block storage node and works as an iSCSI target providing an interface for accessing virtual disks;
- **Protocol**: Internet SCSI (iSCSI) [22], an IP-based storage networking standard for linking data storage facilities;
- **Physical Storage**: Logical Volume Manage (LVM) [23], providing a volume group local to the block storage nodes.

The following table shows how selected technologies meet the selection criteria:

Criteria	Notes
Licensing	tgt is free software under the terms of the GNU General Public License [24] LVM is available under GNU General Public License [25]
Functionality	According to OpenStack Cinder Support Matrix [20], the selected technologies are considered as the reference for OpenStack development; in this perspective, they fully support all the OpenStack functions
Documentation	tgt documentation is freely available (see [21]) LVM documentation is freely available (see [23]) OpenStack integration and the installation procedures are fully documented and freely available (see [16])

Criteria	Notes
Software Requirements	tgt runs on many widely used Linux distribution like RHEL, CentOS, Fedora, SUSE, Debian, Ubuntu and Gentoo. It requires Linux kernel 2.6.22 or higher [21]. LVM runs on many Linux distributions like CentOS, Debian, Fedora, Gentoo, Mandriva, MontaVista Linux, openSUSE, Pardus, Red Hat Enterprise Linux, Slackware, SLED, SLES, Linux Mint, Kali Linux, and Ubuntu [25].
Hardware Requirements	No specific hardware configurations or devices are required for running the selected technologies. On the other hand, any mass storage device that can be mounted on the selected Linux operating system as a read/write disk is supported.

Table 2-3 - Block Storage Technologies Selection Criteria

2.2.2.3. Network Virtualization

Network virtualization is by far the most complex area in OpenStack. This is due to the inherent complexity of the topic in itself and the large amount of alternative technologies that can be used in OpenStack deployments. For more information on the supported network virtualization technologies, see [26].

In order to fully support the functions required by OpenStack, the underlying networking technologies need to collectively provide services for creating/managing virtual objects or services of various types like L2 network entities (e.g. layer 2 trunks, ports, etc.), L3 entities (e.g. IP addresses NATting, firewall services) and a DHCP service.

The networking virtualization technologies selected for SCP are:

- **Open vSwitch**: implements a distributed virtual multilayer switch . It provides API for creating virtual switches and for connecting virtual machines. Virtual switches are hosted on the same nodes running the VMs that are connected;
- **Linux iptables**: it's a technology, natively supported by many Linux distributions, that allows the system administrators to directly operate on network packet filtering and NATting tables and it's implemented by Linux operating system at kernel level [27]. OpenStack integrates with iptables for implementing firewalling and NATting functions;
- **dnsmasq**: "*Dnsmasq provides network infrastructure for small networks: DNS, DHCP, router advertisement and network boot*" [11]. In OpenStack deployments, dnsmasq mainly implements functions for distributing IP addresses to VMs via DHCP.

The following table shows how the selected technologies meet the selection criteria:

Criteria	Notes
Licensing	Open vSwitch is open source software available under Apache 2.0 license (see [28]). Linux iptables is made available under GNU General Public License (see [29]). Dnsmasq is distributed under GNU General Public License, version 2 and 3 (see [11]).
Functionality	The combination of the selected underlying technologies is considered the reference for OpenStack development; therefore it fully supports the OpenStack functions.
Documentation	The selected technologies are fully described on freely available documentation (see [28], [29] and [11]). In addition, the OpenStack integration and installation procedures are fully documented and freely available (see [16]).
Software Requirements	Open vSwitch is officially available for Debian, Fedora and Ubuntu Linux distributions (see [30]). Linux iptables runs under any Linux distribution fully supporting Linux kernel version 2.4.x and 2.6.x (see [31]) that are, in turn, integral part of most of the Linux distributions like CentOS, Debian, Fedora, openSUSE, Red Hat Enterprise Linux, and Ubuntu. Dnsmasq: according to [11], "*Supported platforms include Linux (with glibc and uclibc), Android, *BSD, and Mac OS X*" and "*Dnsmasq is included in most Linux distributions and the ports systems of FreeBSD, OpenBSD and NetBSD*". As described in OpenStack installation documentation [16], dnsmasq is supported by Debian, openSUSE, SUSE Linux Enterprise Server, Red Hat Enterprise Linux, CentOS, Fedora and Ubuntu Linux distributions.
Hardware Requirements	No specific hardware configurations or devices are required for running the selected technologies; on the other hand, OpenStack deployments can significantly benefit of hardware configurations where nodes are equipped with multiple Network Interface Cards (NICs). For more information, see section **Error! Reference source not found.**.

Table 2-4 - Networking Technologies Selection Criteria

2.3. Deployment View

OpenStack can be deployed in several ways but there are some well-known and accepted good practices for implementing deployment architectures. Usually, the deployment architecture also depends on the scope and the purpose of the cloud and/or the constraints on the physical hardware that is being used (e.g. number and type of physical machines, network connections, etc.).

Options range from single-node all-in-one deployment, where all the modules are installed on a single node that also provides the physical resources for hosting virtual resources (this deployment is suitable for demos and development use cases), to very complex deployments addressing many requirements like high availability of the IaaS platform, nodes distribution in multiple data centres, support to different hardware technologies, etc. (this deployment is suitable for public cloud providers).

In this section, we provide some deployment examples (models) that can be used as a reference when designing an actual cloud platform.

In this phase, the project consortium implements and utilizes the "Medium Complexity Deployment" model.

Basic Deployment

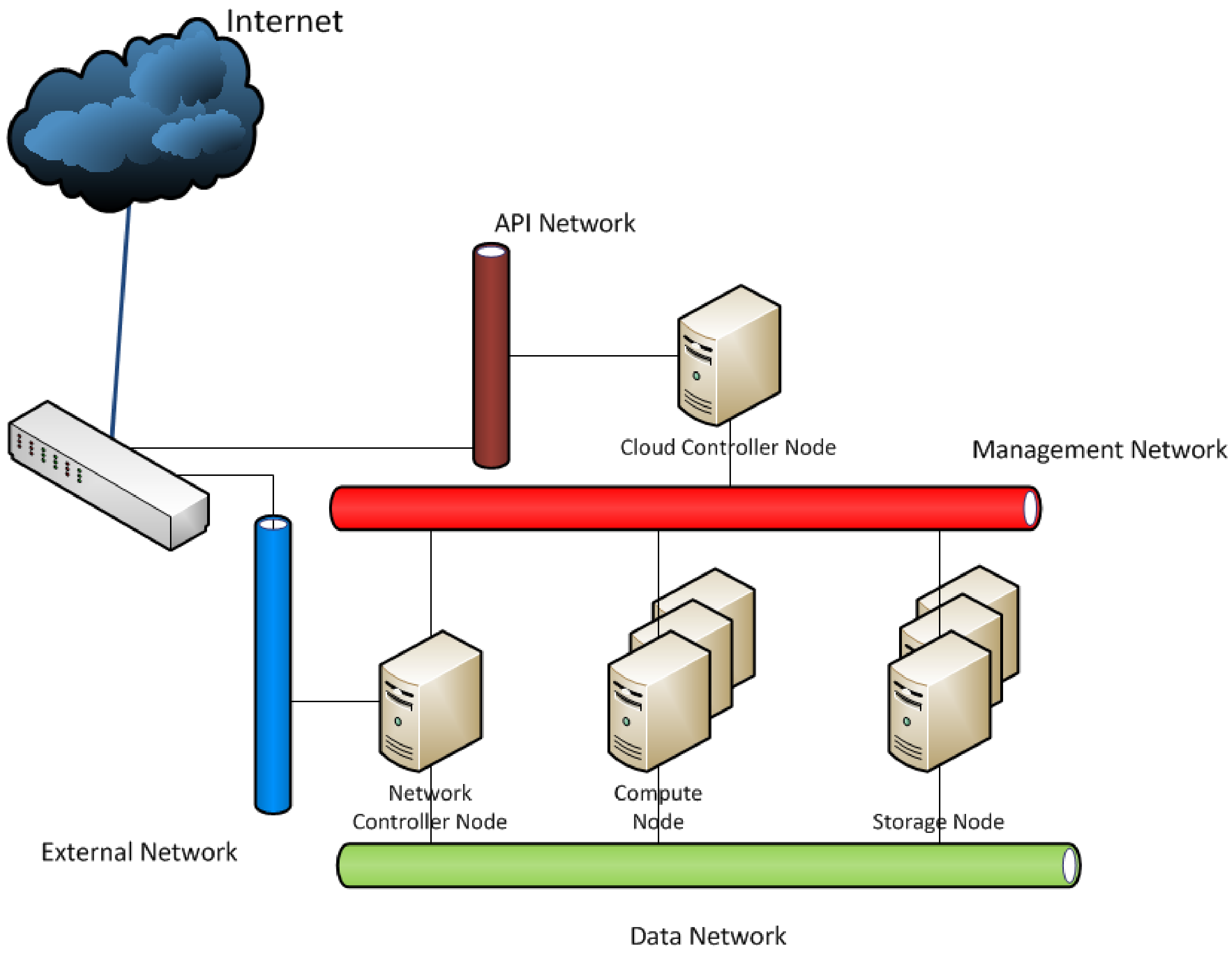

Figure 2-5 - OpenStack Basic Deployment

The nodes in the deployment architecture are:

- **Cloud Controller Node:** it hosts all the centralized functions like the cloud status database, the message broker, the compute and the storage schedulers, API endpoints, authentication services, image catalogue, orchestration engine, monitoring and accounting functions, the web dashboard server, etc.;
- **Network Controller Node:** it hosts some network services like DHCP, layer 2 switching, layer 3 routing and also provides access to VMs from Internet;
- **Storage Node:** it hosts virtual disks;

- **Compute Node**: it hosts the VMs running in the cloud.

The networks in the deployment architecture are:

- **Management Network**: it is used for the communication between the OpenStack elements;
- **Data Network**: it is used for the communication between the VMs running in the cloud and for giving VMs access to Virtual Disks;
- **External Network**: it is used for the communication between the VMs running in the cloud and any other element external to the cloud (e.g. end users on Internet);
- **API Network**: it exposes all OpenStack API endpoints.

A variation of this deployment "collapses" the functions of compute and storage nodes into a single node type providing both the resource types: such a deployment model is described in the OpenStack official documentation [32].

It is worth noticing that the deployment described above does not address some important aspects like high availability and load balancing; on the other hand it can be used for development and testing purposes.

Medium Complexity Deployment

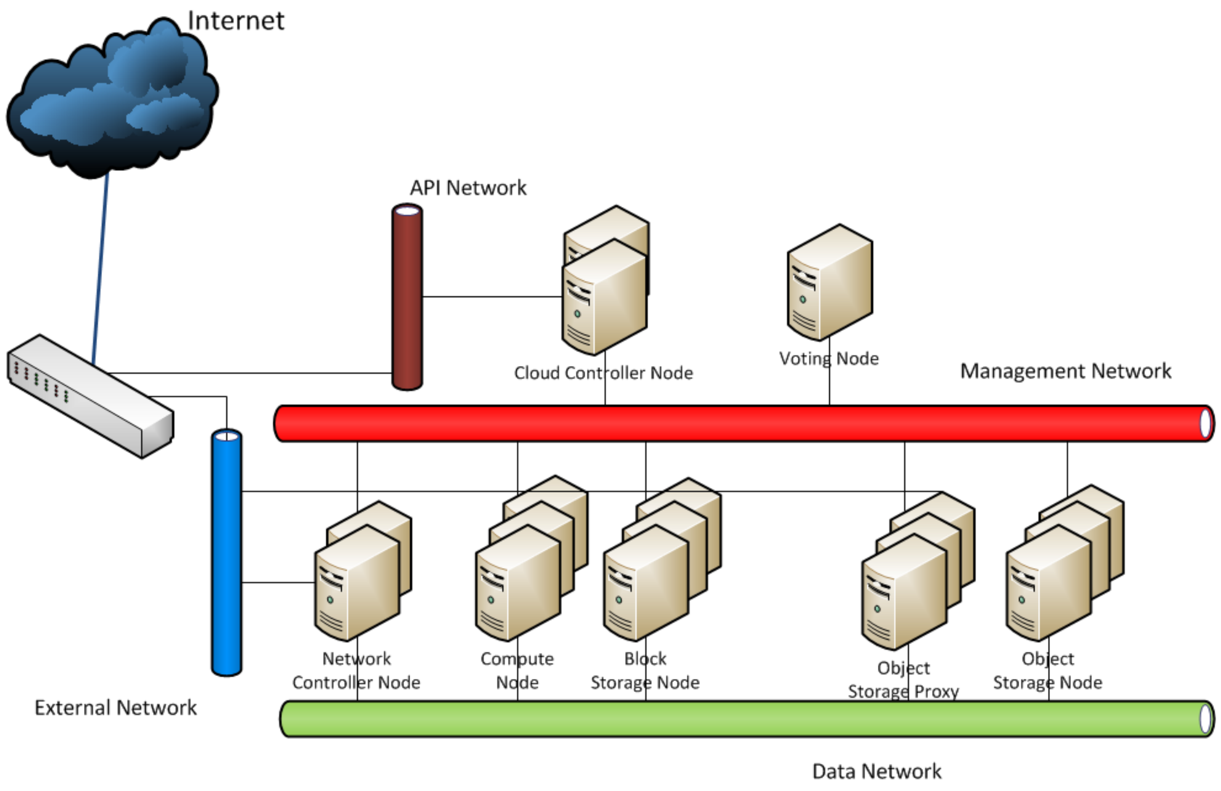

Figure 2-6 - Medium Complexity Deployment

In this deployment model both the controller and the network nodes are replicated and configured as a high availability cluster; a voting node is added for monitoring and controlling the nodes. The reason for the replication is that both the node types are stateful: the controller node keeps the overall status of the cloud (stored in a database) and the message bus. The network controller node stores information about the NATted IP addresses of the VMs and the DHCP server status.

The deployment model also adds two node types for implementing the Object Storage services: Object Storage Node and Object Storage Proxy. The former node type is dedicated to store the user's objects (i.e. files); the OpenStack components running on these nodes are also responsible of replicating the objects. The nodes of the latter type, the Object Storage Proxy, receive the cloud user's requests. They host the Object Storage API front ends (i.e. swift-proxy-server). The reason for dedicating a node type to this software component is that these nodes receive all the requests for uploading/downloading files to/from the cloud resulting in a significant workload and, more important, network traffic. As described in the picture, Object Storage Proxies are directly connected to the External Network.

For more information on high availability / load balancing deployment, see also [33].

3. Platform as a Service Layer

3.1. Concept

While IaaS focuses on managing the fundamental infrastructure building blocks in a cloud environment, thus allowing to transfer any existing deployment to the cloud with little or no architectural changes, Platform as a Service (PaaS) [34] goes one step further and focuses on managing applications instead of infrastructure. The target user in this case is the application developer, who can deploy an application to the PaaS and expects it to just work, delegating all infrastructure management tasks to the PaaS and focusing on development work instead.

As a consequence, the main 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 the figure below:

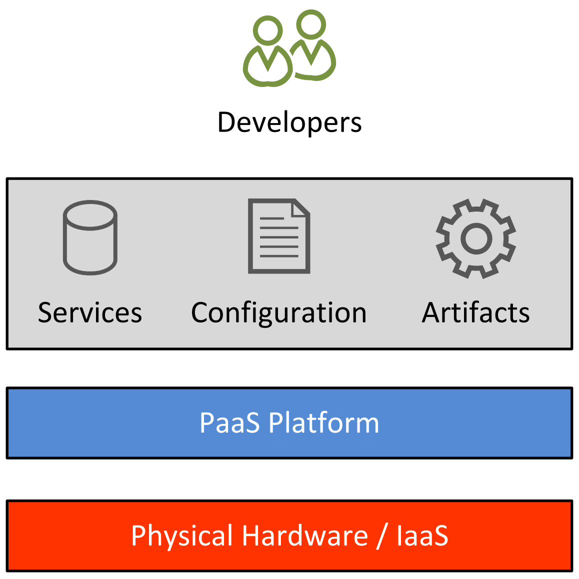

Figure 3-1 – Platform as a Service – Conceptual View

Services are any external component that the application requires in order to run (e.g. a database, a messaging service, etc.). They are provided by the PaaS platform directly and the user can request their use for her own application. For example the user can request the creation of a MySQL database schema and the PaaS will create one on the local MySQL service, without the need for the user to install and manage a private MySQL instance.

Configuration resources include the regular application settings, but also the parameters needed by the PaaS to deploy the application, such as the required RAM and disk size, the required runtime resources (such as the JDK and application server for Java web applications) and the services (such as the database) that must be made available to the application. Depending on the platform, the configuration may also include thresholds for automatically scaling the application as load increases or decreases.

Artifacts represent the application binaries that must be deployed on the PaaS, according to the specified configuration and using the specified services.

Since they work at the application level, PaaS platforms are a great way for developers to deploy scalable and highly available applications without requiring advanced infrastructure skills or a dedicated operations team. On the other hand, this comes at the price of some restrictions in terms of system customization. In fact, since the deployment platform is standardized, developers must take care of using technologies that are supported by the platform. For instance, if the platform supports only Java and PHP, a .NET application will not run on it. Similar considerations apply for application servers and database engines. However typical web applications that run in the cloud use standard components that are supported by major PaaS platforms, so these constraints are not a problem in most cases.

Finally, it is worth noting that IaaS and PaaS are not mutually exclusive, since a user may choose to run some services on a IaaS and use them from an application deployed on a PaaS. The two approaches can also be combined together from the cloud provider point of view. In fact while the provider may run the PaaS on bare metal directly, she may also choose to run it on top of a IaaS in order to take advantage of resource scalability without the need to provision new physical hardware.

3.2. Logical View

The PaaS platform selected for SCP is Cloud Foundry [6].

Cloud Foundry is an open source project, released under the business-friendly Apache License 2.0, started by VMware® in 2011 [35] and subsequently led by Pivotal® (a joint venture between VMware and EMC). As of February 2014, Pivotal announced that the project will be governed by a Cloud Foundry Foundation [36] with strong industry support from EMC, IBM, HP, Pivotal, Rackspace, SAP and others. More members announced the intention to join the Foundation in May 2014 [37].

The sound architecture, industry backing and open governance model all contribute to make Cloud Foundry an attractive choice as a platform that may become the de-facto open-source standard PaaS in the future, similar to what OpenStack represents in the IaaS field.

The figure below shows the logical architecture of Cloud Foundry:

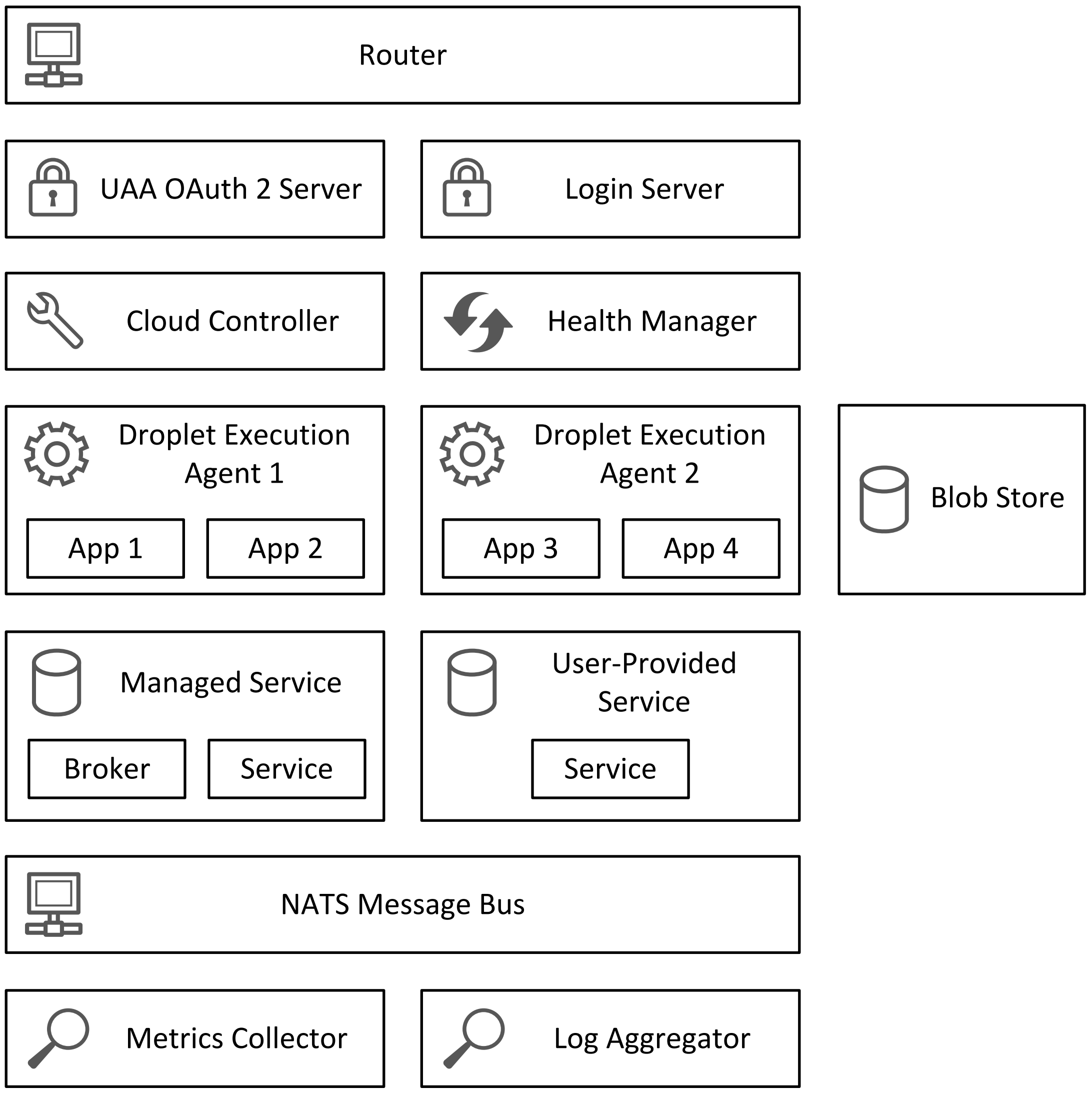

Figure 3-2– Cloud Foundry Logical Architecture

At the top level, the Router component receives all incoming traffic and dispatches it either to the Cloud Controller or to applications, while the OAuth2 Server and Login Server handle authentication of Cloud Foundry users.

The Cloud Controller is the component that manages the lifecycle of applications deployed on the cloud, taking care of starting them, connecting the required services and so on. Application status is monitored by the Health Manager, which can trigger a re-spawn in case an application dies unexpectedly.

The Droplet Execution Agents (DEAs) are the components that actually host applications in Cloud Foundry. While a minimal deployment may have a single DEA, in most cases there will be multiple DEAs each running on a different machine. In order to support multi-tenancy, each application is run in a private environment within the DEA. However, instead of isolating environments using virtual machines as IaaS platforms do, Cloud Foundry uses operating system containers [38] managed via the Warden [39] component. The Warden implementation for Linux is in turn based on Control Groups [40]. Container technologies take advantage of APIs implemented at the operating system level in order to create an isolated environment (i.e. the container) with its own file system, software packages, processes, memory space and network resources. Processes in a container cannot read data in other containers or interfere with processes in different containers and interaction between containers is only possible via traditional inter-machine communication, such as network connections. In other words, containers only share the underlying operating system. With respect to traditional virtualization, this approach requires all applications to be designed to run on a common platform (i.e. a certain version of a single operating system) but also offers several advantages in terms of reduced overhead. In fact there is no need to spawn a new full machine with its operating system copy for each application: all applications share the same operating system installation, thus reducing the per-application RAM and disk footprint dramatically and allowing higher application density on a single physical machine. Also, because there is no virtualization involved, containers offer additional flexibility in terms of deployment. In fact the PaaS can run on bare metal, removing the hypervisor overhead and optimizing performance, but it can also be installed over an IaaS layer, thus maximizing flexibility.

In addition to allocating a container on a DEA, running an application requires a buildpack, which consists in a package that is able to detect the application type, install the required runtime components and execute the application. Cloud Foundry provides official buildpacks for Java, PHP, Python, Ruby, Node.js and Go, while additional buildpacks are provided by the community. It is possible to create custom buildpacks by implementing the required script-based interface. When a developer pushes an application to Cloud Foundry, she can specify a buildpack to use or let the platform auto-detect the application type. For example in case of a Java Web Application, the buildpack may check if the application contains a WEB-INF/web.xml file. Every buildpack must implement a detection script that returns successfully if the buildpack can run the application, so that Cloud Foundry can automatically find a suitable buildpack by invoking that script on every available buildpack. In addition to detection, buildpacks implement a script that installs the required runtime components, such as the JDK and application server for a Java Web Application. These additional software packages may be included in the buildpack itself or downloaded from the Internet, when required. Finally the buildpack implements a third script that Cloud Foundry invokes to start the application.

As we mentioned earlier most applications need to interact with services, such as databases, which are not part of the application and must be provisioned by the platform. Cloud Foundry supports two kinds of services: Managed Services and User-Provided Services. A Managed Service is fully integrated with the platform and can be provisioned on user request by Cloud Foundry. For example, if the service is a database, Cloud Foundry can create a new schema on it when the user requests a new instance of the service. Once a service instance has been created it can be bound (i.e. made available) to a certain application of the user. In order to satisfy different requirements, each service can offer several plans that define the features of the service instance, similar to flavours in the IaaS world. For example a "smalldb" plan may provide a 100 MB database on a single server, while a "largedb" plan may provide a 10 GB database replicated over two servers. Cloud Foundry controls Managed Services via a component called broker that must be provided with the service and implements the Service Broker API. Cloud Foundry includes a broker for MySQL, but brokers for other services are available from the community and of course it is possible to integrate any service by implementing a custom broker. In cases when full integration with Cloud Foundry is not required or desired, it is possible to use User-Provided Services. In this case service instances, such as database schemas, are created outside of Cloud Foundry with traditional mechanisms. Once created, instances must be registered in Cloud Foundry before they can be bound to applications.

Other components in the architecture include the Blob Store, which holds application binaries and buildpacks, and NATS. The latter provides a publish-subscribe system that Cloud Foundry components use to communicate with each other.

Finally, the Metrics Collector can gather various metrics from the platform components in order to monitor system health, while the Log Aggregator collects the logs of applications deployed in DEAs so that users can inspect them. Note that the Metrics Collector allows to write plugins, called "historians", in order to export data to external services. This enables integration with general purpose monitoring tools, such as Zabbix or Nagios, that will be used to consolidate metrics from all components of the whole SCP platform in a single place. Further discussion of the monitoring infrastructure is postponed to D2.2.2.

The main tool for controlling a Cloud Foundry installation is the Command Line Interface, available for Windows, Linux and Mac OS X. Thanks to the CLI it is possible to write scripts to automate tasks such as deploying new versions of applications or scaling horizontally according to the current workload. Additional developer tools include an Eclipse [41] plugin, that allows to configure and deploy applications directly from the IDE, and plugins for Maven [42] and Gradle [43].

3.3. Deployment View

Because Cloud Foundry relies on containers instead of virtual machines in order to run applications, it is possible to deploy the platform both on bare metal hardware and virtual infrastructure. While bare metal may be an interesting option when performance is critical, deploying Cloud Foundry on virtual infrastructure, especially on a IaaS, allows to obtain maximum flexibility in terms of scalability, high-availability and costs. In fact when the PaaS is deployed on a IaaS, it can run with a minimal number of components (e.g. DEAs) when load is light and exploit the autoscaling features of the IaaS to dynamically allocate new instances when the load increases. When the load decreases again, the number of instances can be scaled down back to a minimum. This model of operation is particularly attractive when thinking of a deployment on a IaaS public cloud provider. In fact costs can be minimized by keeping the number of instances low when the load is limited, at the same time achieving optimal performance by increasing the number of instances only when necessary. In addition, using a public IaaS removes the initial investment required to acquire the hardware infrastructure and the subsequent costs for infrastructure upgrades. Regarding high-availabiliy concerns, deploying on a IaaS allows to replicate components over independent availability zones, which may be geographically distributed in case of a public IaaS.

While design choices for the SCP have taken into account scalability, load-balancing and high-availability issues, those themes will be addressed in detail in the second version of the architecture that will be provided in D2.2.2. The architecture described in this deliverable focuses on providing an initial working platform that can be used for migrating and testing the applications selected by the consortium in a cloud environment that is functionally equivalent to the final one.

Based on the above considerations, Figure 3-3 shows the deployment view for the PaaS layer:

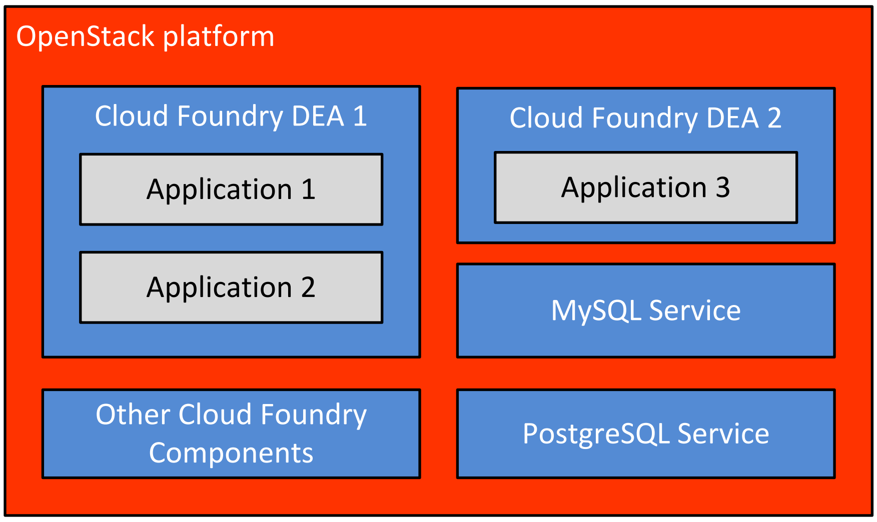

Figure 3-3 – Platform as a Service – Deployment View

Due to the advantages mentioned previously, Cloud Foundry is deployed on top of the IaaS layer, represented by the OpenStack platform. Each Cloud Foundry component runs in its own virtual machine. Since this version of the architecture does not address replication, there is only a single instance for each component and service, with the possible exception of DEAs. In fact depending on the amount of resources required by the applications it may be necessary to allocate multiple DEAs instead of a single very large one, which is expensive on a public cloud and diverges from the horizontal scalability philosophy that privileges using multiple smaller instances in order to increase parallelism and avoid single points of failures.

It is worth noting that thanks to the availability of several OpenStack cloud providers, this architecture can be easily implemented on a public cloud infrastructure, with the associated flexibility and cost benefits. Of course deploying Cloud Foundry over a IaaS is attractive for organizations that require full control over both IaaS and PaaS resources, but this implies the burden of administering the platform. Organizations that wish to benefit from the PaaS abstraction without delving into the details of the underlying infrastructure can rely on one of the public cloud providers that offer ready-to-use Cloud Foundry-based platforms and delegate management of all the underlying layers to the provider.

4. Database Layer

4.1. Concept

Generally speaking, applications use a database for storing persistent data and the SCP Database Layer provides capabilities for supporting this feature.

In principle, any application could run its own database instance and, if necessary, the SCP architecture supports such a scenario. As a matter of fact, an application could be deployed on a single virtual machine that hosts both the application (e.g. the web front end) and the database engine:

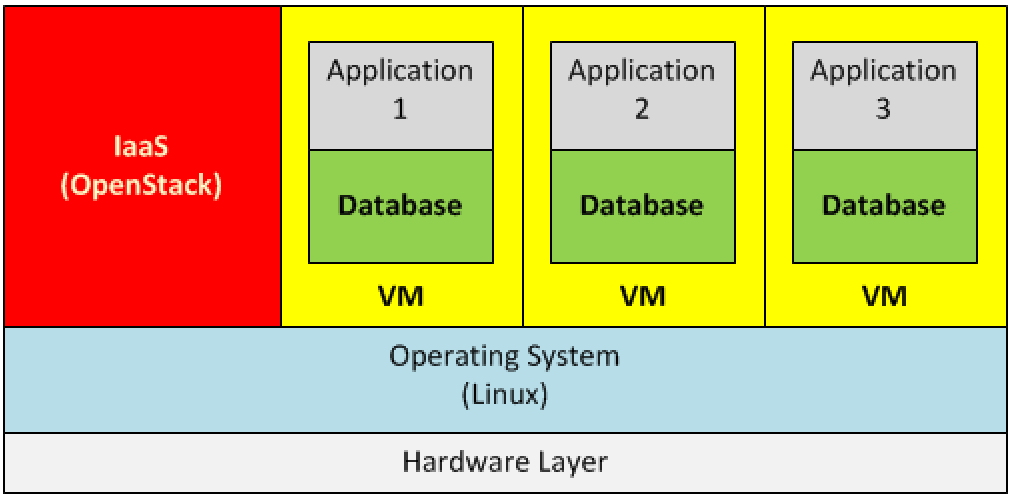

Figure 4-1 - Application Database - Simplistic Scenario

The scenario described above is applicable to very simple use cases where high availability and load balancing features are not strictly required, for instance for development and testing purposes.

In a more complex scenario, the application could be deployed using multiple VMs: some of them hosting web front end instances (in a load balancing configuration), others hosting the database engine (in a high availability configuration).

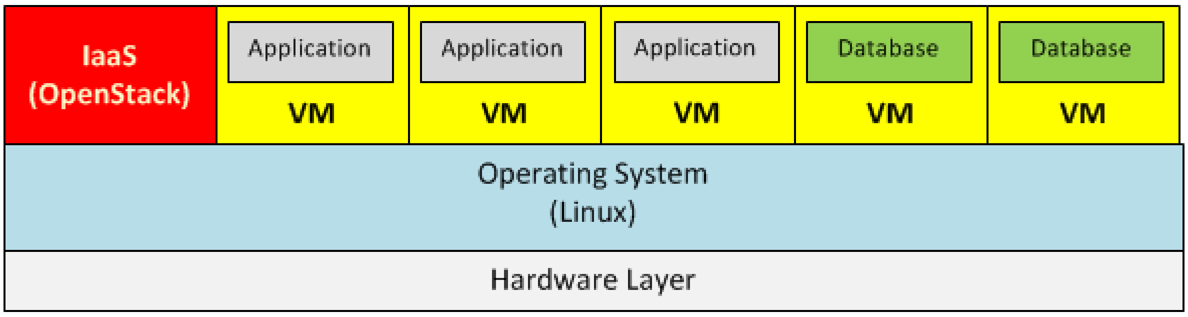

Figure 4-2 – Application Database – Complex Scenario

It's worth noticing that, in order to really address high availability issues, some VMs shall be deployed on different nodes: for instance the VMs hosting the database copies must run on different machines.

The scenarios described above are fully supported by SCP and can be implemented using IaaS capabilities only. In such cases, it is cloud user's responsibility to design the application deployment architecture that is implemented using the building blocks made available by the IaaS Layer. For instance, for implementing a two tier application, the cloud user needs to prepare the VMs for the application tier and the VMs for the database tier. In addition, if high availability is required, the cloud user needs to configure the VMs with the database tier in a high available, maybe clustered, architecture.

SCP architecture implements a specific module, the Database Layer, providing database capabilities and – at the same time – addressing high availability and load balancing issues. In this perspective, the Database Layer alleviates the cloud user's task because s/he can focus on the application development and adaptation without taking care of resolving some deployment issues.

4.2. Logical View

As reported in the overall architecture (see Figure 2-3 - Storm Clouds Platform - Overall Architecture), the Database Layer can be both used by applications running in the IaaS or in the PaaS.

The Database Layer supplies two database engines: MySQL [7], available under GPL V2 license, and PostgreSQL, available under PostgreSQL License (similar to BSD and MIT license [44]).

From the implementation view point, the Database Layer is designed as a set pre-packaged virtual machine images with an installation and configuration of the database engine. They are activated as virtual machines in the SCP taking advantage of services of the IaaS Layer. The deployment of the virtual machines is orchestrated using OpenStack Heat services.

This kind of solution enables different usage models:

- **exclusive**: a database instance is activated for a single application;
- **shared**: a database instance can be used by different applications, provided that they use different database schemes.

The following picture summarizes the concept:

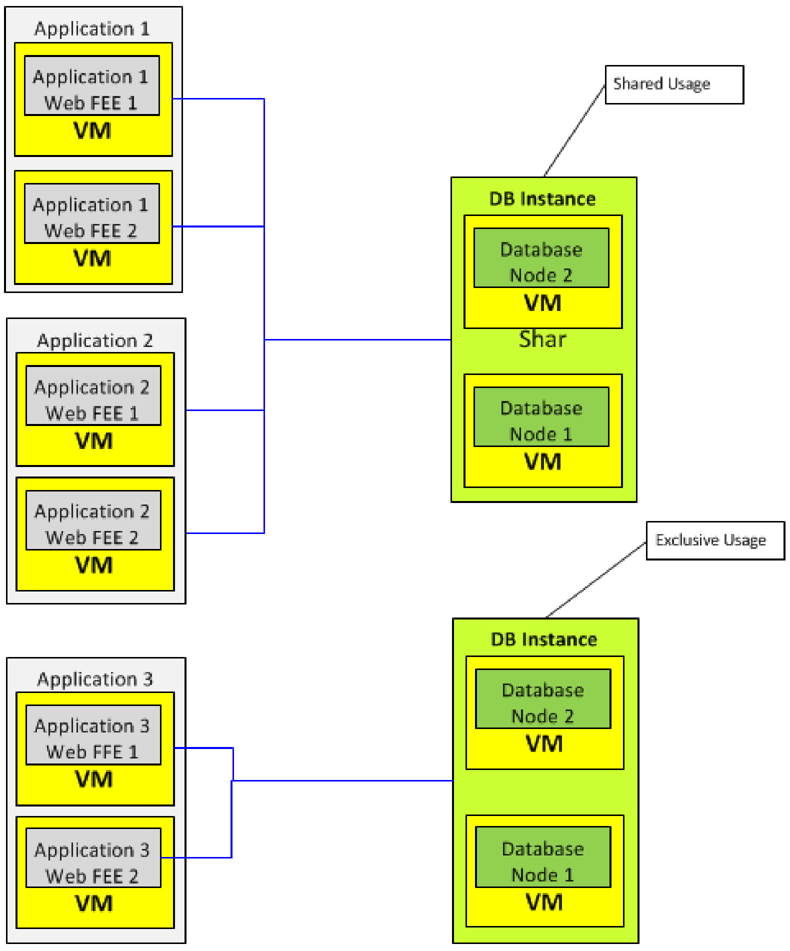

Figure 4-3 - Database Instance Usage Models

The picture shows three applications with two Web Front End nodes each. Application 1 and Application 2 share a single database instance while Application 3 use a database instance in exclusive mode.

4.3. Deployment View

At the time being, for each supported database engine type (e.g. MySQL), a single DB instance is deployed in the SCP and each application uses a database created in the single instance. The cloud administrator is responsible of creating the databases on cloud user's request. The cloud administrator delivers the credentials to cloud users who are, in turn, responsible of creating the schema, accessing the DB, etc.

5. Management Layer

5.1. Concept

The management layer implements functions for managing and maintaining the cloud platform and is designed for the organization in charge of providing the cloud computational services implemented by the SCP, not hosted by the SCP.

In SCP architecture the management layer mainly implements monitoring functions because other functions usually allocated to this layer are actually implemented in other components.

For instance, catalogue management (i.e. functions for managing VM templates and application templates) is implemented by both the OpenStack IaaS Platform and Cloud Foundry PaaS Platform.

Monitoring refers to functions for verifying the availability and the performances of the infrastructure. It is designed for verifying the working conditions of the physical resources (e.g. servers) but can also be useful for monitoring virtual resources (i.e. virtual machines) used for implementing advanced cloud services like PaaS components, virtual machines running DB engines (maybe for implementing DB as a Services components), etc.

At the time of writing, this component is still to be designed but it will be implemented using open software solutions like Zabbix [9] or Nagios [10].

5.2. Logical View

This section is postponed to D2.2.2.

5.3. Deployment View

This section is postponed to D2.2.2.

6. Summary and Conclusions

This document has described the first release of the Storm Clouds Platform (SCP) architecture. SCP is the cloud infrastructure designed for hosting the applications selected by the STORM CLOUDS consortium for experimenting the migration of digital services to a cloud computing paradigm.

SCP is broken down in layers each one implementing a set of highly related functions and services: Infrastructure as a Service Layer provides basic computational objects like virtual machines, virtual disks and virtual networks; Platform as a Service Layer is designed for hosting web-based applications, Database Layer makes it available services for creating, managing and using databases; Management Layer implements functions, typically used by the cloud platform administrator, for controlling, monitoring and administering the whole platform.

For each layer, the architecture describes what the main functions and objectives are, what are the software products selected for the implementation and how they can be deployed on a physical infrastructure.

The document is the first of a two issue series that will eventually result in the full definition of the SCP architecture. In this first issue the objective is to define the basic functions required for porting the applications to a cloud computing environment.

In the next release, the document will detail and develop some aspects regarding a “production ready environment” (e.g. high availability issues, cloud platform management, monitoring, etc.) that are not fully described or developed here.

Appendix A – OpenStack Configuration File Example

This annex reports an excerpt of an OpenStack configuration file:

```
[DEFAULT]
# Default log level is INFO
# verbose and debug has the same result.
# One of them will set DEBUG log level output
# debug = False
# verbose = False

#  Where  to  store  Neutron  state  files.   This  directory  must  be
writable by the
# user executing the agent.
state_path = /var/lib/neutron

# Where to store lock files
lock_path = $state_path/lock

# log_format = %(asctime)s %(levelname)8s [%(name)s] %(message)s
# log_date_format = %Y-%m-%d %H:%M:%S

# use_syslog                           -> syslog
# log_file and log_dir                 -> log_dir/log_file
#   (not   log_file)   and   log_dir                              ->
log_dir/{binary_name}.log
# use_stderr                           -> stderr
# (not user_stderr) and (not log_file) -> stdout
# publish_errors                       -> notification system

# use_syslog = False
# syslog_log_facility = LOG_USER

# use_stderr = True
# log_file =
# log_dir =

# publish_errors = False

# Address to bind the API server
# bind_host = 0.0.0.0

# Port the bind the API server to
# bind_port = 9696
:...
```

References

[1] "Storm Clouds - Project Web Site," [Online]. Available: http://stormclouds.eu/. [Accessed July 2014].

[2] "Storm Clouds Project - European Commission Project Page," [Online]. Available: http://ec.europa.eu/digital-agenda/en/storm-clouds-project-cloud-public-services. [Accessed July 2014].

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

[4] Consonni, Marco;Panuccio, Pasquale, "Storm Clouds Project: D 2.1 - Storm Clouds Platform – Requirements and Specification," STORM CLOUDS Project, 2014.

[5] "OpenStack Project," [Online]. Available: http://openstack.org. [Accessed June 2014].

[6] "Cloud Foundry - Community Main Page," [Online]. Available: http://cloudfoundry.org/index.html. [Accessed July 2014].

[7] "MySQL - Main Page," [Online]. Available: http://www.mysql.com/. [Accessed July 2014].

[8] "PostgreSQL - Main Page," [Online]. Available: http://www.postgresql.org/. [Accessed July 2014].

[9] "Zabbix - Main Page," [Online]. Available: https://www.zabbix.org/wiki/Main_Page. [Accessed July 2014].

[10] "Nagios - Main Page," [Online]. Available: http://www.nagios.org/. [Accessed July 2014].

[11] "Dnsmasq," [Online]. Available: http://www.thekelleys.org.uk/dnsmasq/doc.html. [Accessed July 2014].

[12] "OpenStack Cloud Administrator Guides," [Online]. Available: http://docs.openstack.org/admin-guide-cloud/content/index.html. [Accessed July 2014].

[13] "django project page," [Online]. Available: https://www.djangoproject.com/. [Accessed July 2014].

[14] "Apache HTTP Server Project - Main Page," [Online]. Available: https://httpd.apache.org/. [Accessed July 2014].

[15] "Representational state transfer - Wikipedia Page," [Online]. Available: http://en.wikipedia.org/wiki/Representational_state_transfer. [Accessed July 2014].

[16] "OpeStack Documentation - Main Page," [Online]. Available: http://docs.openstack.org/. [Accessed July 2014].

[17] "OpenStack Hypervisor Support Matrix," [Online]. Available: https://wiki.openstack.org/wiki/HypervisorSupportMatrix. [Accessed July 2014].

[18] "Kernel Based Virtual Machine - Main Page," [Online]. Available: http://www.linux-kvm.org/page/Main_Page. [Accessed July 2014].

[19] "Kernel-based Virtual Machine - Wikipedia Page," [Online]. Available: http://en.wikipedia.org/wiki/Kernel-based_Virtual_Machine#Licensing. [Accessed July 2014].

[20] "OpenStack Cinder Support Mitrix," [Online]. Available: https://wiki.openstack.org/wiki/CinderSupportMatrix. [Accessed July 2014].

[21] "tgt project - Linux SCSI target framework," [Online]. Available: http://stgt.sourceforge.net/. [Accessed July 2014].

[22] "RFC 3720 - Internet Small Computer Systems Interface (iSCSI)," [Online]. Available: http://tools.ietf.org/html/rfc3720. [Accessed July 2014].

[23] "LVM2 Resource Page," [Online]. Available: https://sourceware.org/lvm2/. [Accessed July 2014].

[24] "tgtd Manual Pages," [Online]. Available: http://stgt.sourceforge.net/manpages/tgtd.8.html. [Accessed July 2014].

[25] "Logical Volume Manager (Linux) - Wikipedia Page," [Online]. Available: http://en.wikipedia.org/wiki/Logical_Volume_Manager_%28Linux%29. [Accessed July 2014].

[26] "OpenStack Cloud Administrator Guide - Networking Plug-In Architecture," [Online]. Available: http://docs.openstack.org/admin-guide-cloud/content/section_plugin-arch.html.

[27] "IPTABLES - Linux Manual Page," [Online]. Available: http://ipset.netfilter.org/iptables.man.html. [Accessed July 2014].

[28] "Open vSwitch - An Open Virtual Switch," [Online]. Available: http://openvswitch.org/. [Accessed July 2014].

[29] "iptables - Wikipedia Page," [Online]. Available: http://en.wikipedia.org/wiki/Iptables. [Accessed July 2014].

[30] "Open vSwitch - Wikipedia Page," [Online]. Available: http://en.wikipedia.org/wiki/Open_vSwitch. [Accessed July 2014].

[31] "iptables - free code download page," [Online]. Available: http://freecode.com/projects/iptables/. [Accessed July 2014].

[32] "OpenStack Installation Guide for Ubuntu 12.04/14.04 (LTS)," [Online]. Available: http://docs.openstack.org/icehouse/install-guide/install/apt/content/. [Accessed July 2014].

[33] "OpenStack High Availability Guide," [Online]. Available: http://docs.openstack.org/high-availability-guide/content/index.html. [Accessed July 2014].

[34] "Platform as a Service - Wikipedia Page," [Online]. Available: http://en.wikipedia.org/wiki/Platform_as_a_service. [Accessed July 2014].

[35] "VMware Delivers Cloud Foundry, The Industry's First Open PaaS," VMware, [Online]. Available: http://www.vmware.com/company/news/releases/cloud-foundry-apr2011.html. [Accessed July 2014].

[36] "Pivotal Moves to Establish Open Governance Model for Cloud Foundry," [Online]. Available: https://www.gopivotal.com/platform-as-a-service/cloud-foundry-foundation. [Accessed July 2014].

[37] "Pivotal Names Eight Additional Organizations that Intend to Join the Cloud Foundry Foundation," [Online]. Available: https://www.gopivotal.com/platform-as-a-service/press-release/cloud-foundry-foundation. [Accessed July 2014].

[38] "Operating system–level virtualization," [Online]. Available: http://en.wikipedia.org/wiki/Operating_system-level_virtualization. [Accessed July 2014].

[39] "Warder - Description Page," [Online]. Available: http://docs.cloudfoundry.org/concepts/architecture/warden.html. [Accessed July 2014].

[40] "Control Groups - Description Page," [Online]. Available: https://www.kernel.org/doc/Documentation/cgroups/cgroups.txt. [Accessed July 2014].

[41] "Eclipse Project Web Site," [Online]. Available: http://www.eclipse.org/. [Accessed July 2014].

[42] "Maven Project Web Siste," [Online]. Available: http://maven.apache.org/). [Accessed July 2014].

[43] "Gradle Project Web Site," [Online]. Available: http://www.gradle.org/). [Accessed July 2014].

[44] "PostgreSQL License," [Online]. Available: http://wiki.postgresql.org/wiki/FAQ#What_is_the_license_of_PostgreSQL.3F. [Accessed July 2014].

[45] "STORM CLOUDS Consortium," [Online]. Available: http://stormclouds.eu/?page_id=25. [Accessed July 2014].