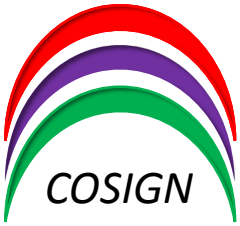

Grant Agreement No. 619572

COSIGN

Combining Optics and SDN In next Generation data centre Networks

Programme: Information and Communication Technologies

Funding scheme: Collaborative Project – Large-Scale Integrating Project

Deliverable D4.2

COSIGN orchestrator low level architecture and prototype design

Due date of deliverable: June 2015
Actual submission date: 5. August 2015

Start date of project: January 1, 2014 Duration: 36 months

Lead contractor for this deliverable: I2CAT, José Aznar

Project co-funded by the European Commission within the Seventh Framework Programme		
Dissemination Level		
PU	Public	X
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Executive Summary

The current document comprises the second important achievement of WP4 in the design of the COSIGN orchestrator layer. More specifically, this deliverable focusses on the specification of the orchestrator low level architecture and design, based on the final refinements and inputs provided from the architecture design (WP1), the underlying SDN network control chosen by WP3 partners, and the identification of OpenStack [3] as the reference platform for implementing the required functionality through extending OpenStack data models, interfaces, and services to achieve the requirements and objectives stated in previous deliverables.

The document is organized as follows:

Section 2 gives a brief reminder of the COSIGN orchestrator, its purpose and role as part of the overall architecture.

Section 3 surveys the OpenStack cloud management platform and its components relevant to the COSIGN orchestrator; it also provides the details of the orchestrator development framework that will be built to facilitate the prototyping and demonstration, listing the software components and tools that will be used, including native OpenStack services as well as external OSS frameworks and libraries.

Section 4 describes COSIGN use cases from the Orchestrator's perspective, outlines use-case specific data models and requirements, and presents the converged data model to be used in COSIGN Orchestrator, based on the OpenStack data model.

Section 5 presents the layered architecture of the COSIGN orchestrator, mapped to the layered architecture of the OpenStack cloud management stack. Focusing on COSIGN orchestrator's needs, we distinguish three major architectural layers – the Infrastructure Control and Monitoring Clients' Layer, the Data Store and Algorithms Layer, and the Orchestrator APIs Layer. Each layer is presented in its own subsection, showing also internal components of each layer.

Section 6 specifies the interfaces between the three major architectural layers of COSIGN orchestrator, identified in Section 5. In addition, this section contains flow diagrams showing how major user facing interactions required by the COSIGN use cases are realized through the internal architectural components and interfaces between them.

Section 7 specifies the external interfaces of the COSIGN Orchestrator. These interfaces are: the southbound interfaces towards the Infrastructure controllers, most importantly the COSIGN SDN Controller, and the northbound interfaces where users and higher level tools can interact with the Orchestrator, realizing scenarios allowed by COSIGN use cases.

Finally, Section 8 concludes the deliverable by summarizing its goals as part of WP4 progress and as a contribution to the overall project outcomes.

Document Information

Status and Version:	v15 - final	
Date of Issue:	August 5, 2015	
Dissemination level:	Public	
Author(s):	**Name**	**Partner**
	José Soler	DTU
	Cosmin Caba	DTU
	Albert Pagès	UPC
	Fernando Agraz	UPC
	Salvatore Spadaro	UPC
	Katherine Barabash	IBM
	Yaniv Ben-Itzhak	IBM
	Amaia Legarrea	I2CAT
	Eduard Escalona	I2CAT
	Alessandro Predieri	IRT
	Matteo Biancani	IRT
	Giacomo Bernini	NXW
	Giada Landi	NXW
	Bingli Guo	UNIVBRIS
	Shuping Peng	UNIVBRIS
	Reza Nejabati	UNIVBRIS
	Dimitra Simeonidou	UNIVBRIS
Edited by:	Jose Aznar	I2CAT
Reviewed by:	Domenico Gallico	IRT
	Bingli Guo	UNIVBRIS
	Anna Levin	IBM
Checked by :	Sarah Renée Ruepp	DTU
	Helene Udsen	DTU

Table of Contents

1 Introduction

1.1 Reference Material

1.1.1 Reference Documents

[D1.1]	COSIGN Deliverable D1.1: Requirements for Next Generation intra DCN Design
[D1.4]	COSIGN Deliverable D1.4: Architecture design
[D4.1]	COSIGN Deliverable D4.1: COSIGN orchestrator requirements and high level architecture
[D3.1]	COSIGN Deliverable D3.1: SDN framework functional architecture
[D3.2]	COSIGN Deliverable D3.2: SDN framework northbound and southbound interfaces specification

1.1.2 Acronyms and Abbreviations

Most frequently used acronyms in the Deliverable are listed below. Additional acronyms can be specified and used throughout the text.

ACI	Application Centric Infrastructure
ADC	Application Delivery Controller
API	Application Programming Interface
AWS	Amazon Web Services
BSS	Business Support System
CLI	Command Line Interface
CMS	Cloud Management Software
CRUD	Create, Read, Update and Delete
DC	Data Centre
DCN	Data Centre Network
DOVE	Distributed Overlay Virtual Network
FW	Firewall
GUI	Graphical User Interface
HA	High Availability
HOT	Heat Orchestration Template
IaaS	Infrastructure as a Service
KVM	Kernel-based Virtual Machine
LB	Load Balancing
NFV	Network Function Virtualization
OF	OpenFlow
O&M	Operations and Management
OPNFV	Open Platform for Network Function Virtualization
OSS	Operations Support System
OVS	Open Virtual Switch (also called Open vSwitch)
PaaS	Platform as a Service
QoS	Quality of Service
REST	REpresentational State Transfer
SDN	Software Defined Networking
ToR	Top of the Rack (switch)
TOSCA	Topology and Orchestration Specification for Cloud Applications
UC	Use Case
UML	Unified Modelling Language
vApp	Virtual (Cloud) Application
VDC	Virtual Data Centre
VIM	Virtual Infrastructure Manager

VN	Virtual Network
vNET	Virtual Network (in VDC lingo)
vOS	Virtual Optical Switch
VPC	Virtual Private Cloud
vToR	Virtual Top of the Rack (switch)
WDM	Wavelength-Division Multiplexing
YAML	Yet Another Markup Language

1.2 Document History

Version	Date	Authors	Comment
1.0	06/05/2015	José Aznar	Load initial ToC into the template
3.5	18/05/2015	José Aznar, K. Barabash	Final ToC version of the document with assignments.
4.1	20/05/2015	José Aznar	First integrated version.
5.0	22/06/2015	José Aznar	Second integrated version.
6.0	12/07/2015	K. Barabash	IBM contribution and change of template following up a series of team-wide discussions
7.0	13/07/2015	K. Barabash	More contributions to section 3
8.0	16/07/2015	José Aznar, Cosmin Caba, K. Barabash,	Use cases elaboration and intermediate review
8.5	17/07/2015	José Aznar Albert Pagès	Integrated partners' contributions
8.8	20/07/2015	K.Barabash	Reformatted section 4, integrated UPC updates
9.0	24/07/2015	José Aznar	Merged contributions
9.1	26/07/2015	K. Barabash	Finalized and cleaned up Section 4
9.2	26/07/2015	K. Barabash	Finalized and cleaned up Sections 5 and 6, reviewed section 7
9.5	27/07/2015	José Aznar	Merged contributions to section 7
9.6	28/07/2015	José Aznar	Address IRT review, merge last round contributions
9.7	29/07/2015	K. Barabash	Address remaining IRT review comments
9.8	29/07/2015	José Aznar	Review and combine all the outstanding contributions
9.9	30/07/2015	K. Barabash	Address IRT review comments for the not yet reviewed sections
10.0-final	31/07/2015	K.Barabash	Address UNIVBRIST comments, incorporate inputs from i2CAT, DTU, UPC
11.0-final	04/08/2015	K.Barabash	Address the quality check comments and edits
12.0-final	05/08/2015	S. Ruepp	Final version for submission

2 Overview – COSIGN Orchestrator

The ultimate goal of the COSIGN project is to develop the Next Generation Data Centre Network (DCN) that can adequately serve modern cloud-scale environment [D1.1]. Although the physical DCN characteristics such as the topology, the path redundancy, the total supported bandwidth, the communication latency, the cost, the power consumption, etc., are of the outmost importance, additional properties are gaining increasingly more attention as the scale, the heterogeneity, and the dynamicity of the usage increase. One such property, made possible by the SDN approach to network control, is the flexibility and the ability to take advantage of the underlying resources in a dynamic and programmable way. An additional property is for the network to become an integral part of the overall resource pool, so that a richer set of use cases can be implemented, automatically harnessing all the DC resources required to satisfy users' needs in a way most optimal both to the user and to the provider.

While satisfying the infrastructural DCN requirements with novel optical technologies is COSIGN WP2's goal and achieving programmatic control over the optical DCN is COSIGN WP3's goal, the goal of the COSIGN orchestrator is to coordinate the IT and DCN resource provisioning for different use cases automatically, efficiently, and optimally. In addition, we aim at leveraging the benefits of the dynamically integrated overlay and underlay connectivity capabilities. Following the COSIGN architecture [D1.4], Figure 1 illustrates the architectural significance of the orchestrator layer as the layer located above the DC resource controllers' layer and below the layer of DC resource and services consumers.

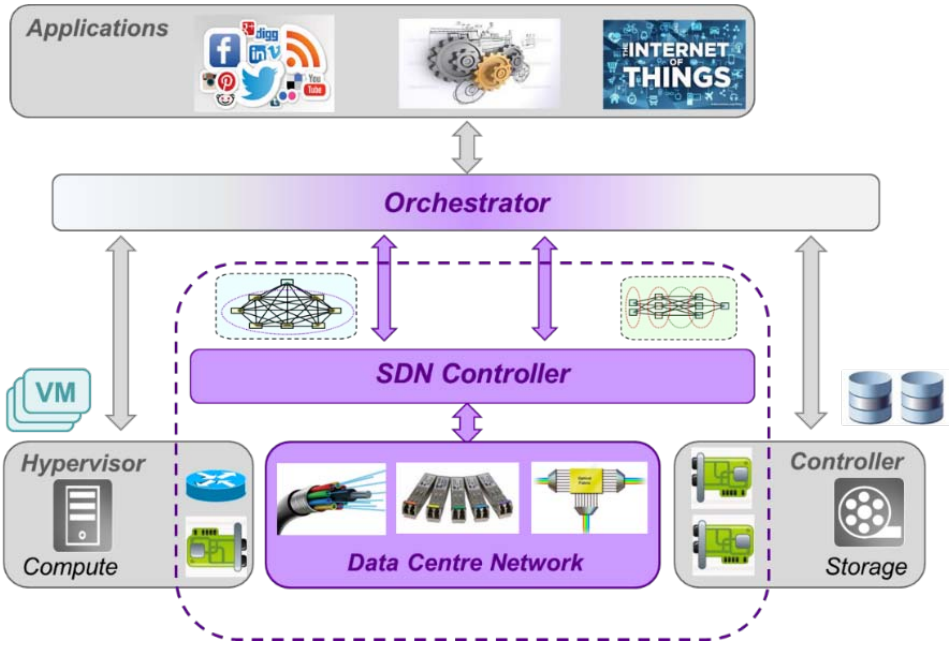

Figure 1 – DC Orchestration layer – position and functionality [D1.4]

According to the software architecture adopted by COSIGN, the orchestrator is the component where all the COSIGN use cases are realized. The orchestrator interacts with all the DC resource controllers to achieve the required automation and includes algorithmic engines and feedback loops to achieve optimal resource usage across the different resource groups and over time. The DCN focus of the project makes COSIGN orchestrator network centric, with an accent on coordinating the networking capabilities and on extending the known orchestration scenarios, e.g., load balancing, application elasticity, failover, to become more network aware than it is possible with today's static approach to networking.

In this document, the internal design and the interfaces of the COSIGN orchestrator are defined, focusing on realizing the use cases outlined in WP1 and on leveraging the capabilities provided by

WP2 and harnessed by WP3. Immediate interfaces are, therefore with the DC users on the north and with the DCN controller on the south. Internal algorithms include the network-aware resource management, realizing advanced application virtualization and infrastructure slicing scenarios. For more details regarding orchestrator role, requirements, and high level architecture, see COSIGN Deliverable [D4.1].

3 Orchestration Platform and Development Framework

Two of the COSIGN use cases brought up by the industrial partners are the Virtual Data Centre (VDC) by Interoute and the Virtual Applications Cloud (vAPP) by IBM. As both the above use cases refer to Cloud as a reference deployment environment, COSIGN DCN must be integrated into the cloud management stack. According to a recent Gartner report [1], providing standard Cloud Management Software (CMS) APIs is of the outmost importance to the success of any cloud-related architecture or product. Therefore, the COSIGN team has chosen to base its orchestrator on a well-known standard cloud management platform. As reported by Forrester [2], as of 2014 OpenStack [3] has become a world-wide de-facto standard CSM for infrastructure centric deployments, confirming it as a best choice for integration of the COSIGN DCN and basis for the COSIGN orchestrator. Also, as reported by Infonetics research [4], OpenStack is a top choice for service providers who are deploying orchestration software in their data centre networks, "nabbing a first-place finish in evaluations with 18 percent of service providers".

Below we describe the current status of OpenStack development, focusing on projects and services most relevant to the COSIGN orchestrator and pointing out the extensions planned as part of COSIGN architecture development. In subsection 3.2, we outline the details of COSIGN orchestrator's development environment based on OpenStack.

3.1 OpenStack as a Reference Platform

COSIGN has adopted OpenStack as the reference DC management platform. This choice was already made and justified in the previous deliverable [D4.1], driven both by the technological motivations and by the considerations for COSIGN impact in a well-established open-source community. Since [D4.1] submission, OpenStack has continued gaining momentum and today constitutes the de-facto reference standard for cloud environments, confirmed by multiple sources, most recent being the Forrester report on state of Cloud Platform Standards [2]. For example, in the Network Function Virtualization (NFV) area, one of the most promising exploitation fields for cloud computing with enhanced network services, the recent first release of the open-source platform for Network Function Virtualization (NFV), OPNFV Arno [5], is strongly based on OpenStack, in combination with OpenDayLight [6], regarding the NFV Infrastructure and Virtual Infrastructure Manager (VIM) components.

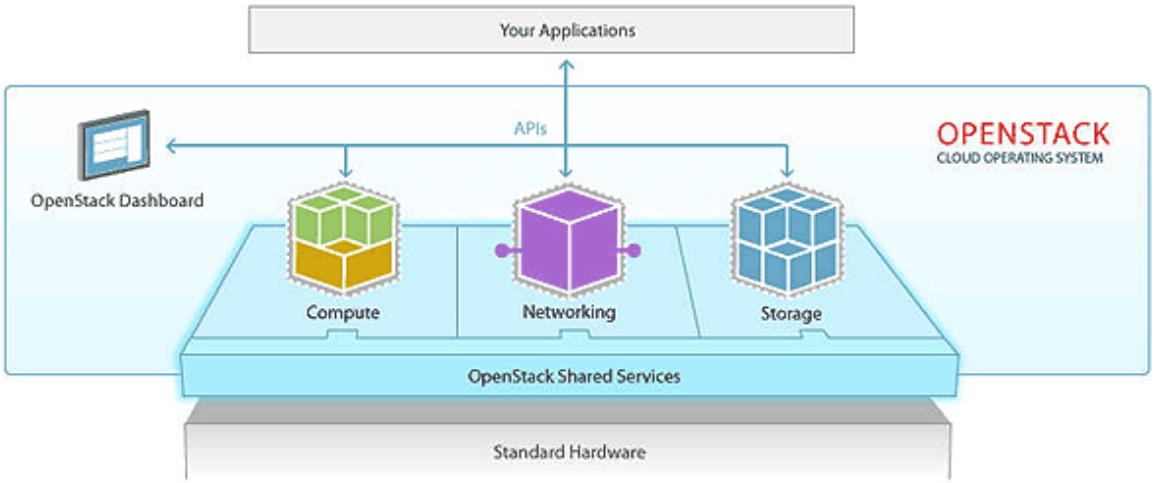

Figure 2 – OpenStack Modular Architecture [3]

3.1.1 OpenStack Core Infrastructural Services – Nova, Neutron, Swift

From a technical point of view, the modularity of OpenStack as shown in Figure 2, is well suited to meet COSIGN requirements in terms of heterogeneous DC infrastructure management and dynamic provisioning and orchestration of enhanced cloud services. The usage of a dedicated and flexible

component for the operation of the DC network, OpenStack Neutron [7], is particularly desirable for COSIGN, since it allows to fully exploit the bi-directional integration between the cloud management platform and the programmable DC network through the mediation of the SDN controllers, without tightly coupling the different architectural layers. Neutron adopts an approach based on a powerful abstraction of the virtual networking concepts and allows isolating the applications from the specifics and the complexity of DC network resources available at the data plane.

The simple OpenStack architecture presented in Figure 2 allows applications to consume DC resources through the abstracted Shared Services, Compute, Networking, and Storage, deployed over the common shared Standard Hardware layer. Additional services, not shown in Figure 2 for simplicity, are less important for COSIGN and comprise common services, e.g., the identity and authentication service Keystone, the image repository service Glance, the message queue service, etc. Overall, there are today multiple core, established, or experimental OpenStack services that can be accessed through the OpenStack REST APIs, OpenStack CLI, and OpenStack GUI – Horizon Dashboard.

The simplicity and the extensibility of the OpenStack approach to managing the infrastructure resources through the Shared Services layer is appealing and lends itself easily to automation through scripting and workflow-based management. It is not sufficient, however, to provide the orchestration platform that, in order to go beyond simple automation, must provide a way to react to events, be aware of dependencies between the resources, and encompass non-trivial multi-step scenarios. Instead of manipulating infrastructure elements one by one, through single commands or with scripts, the orchestrator must work with a declarative model for representing the infrastructure resources and the relationships between them and to be capable of carrying out the correct sequence of actions for realizing the model's intent through the underlying infrastructure controls.

3.1.2 OpenStack Orchestration – Heat

OpenStack Orchestration is a template-driven service for describing and automating deployment of compute, storage and networking resources, created and officially released as part of Openstack Havana [8]. This service, called OpenStack Heat [9], initially mimicked the CloudFormation by the then-standard Amazon Web Services (AWS) Cloud [11]. Today, the main interface to Heat is an OpenStack-native Rest API. As shown in Figure 3, Heat service is located between the OpenStack user and the OpenStack Shared Services layer, directly interacting with the Shared Services and exposed through REST API, CLI, and Horizon.

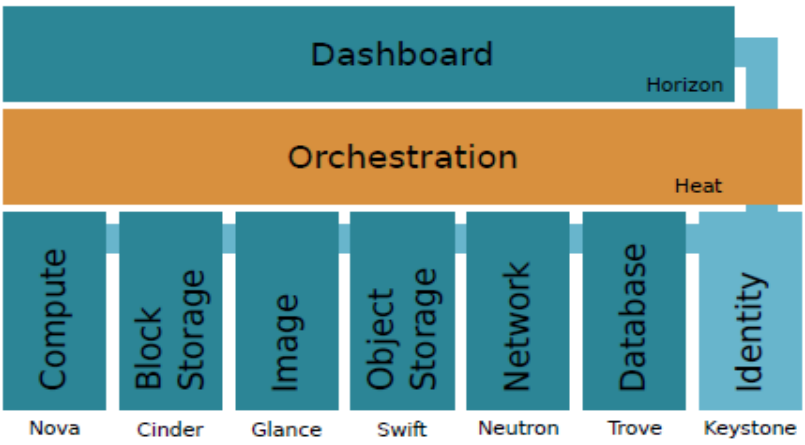

Figure 3 – OpenStack Orchestration Service (Heat) and its interaction with other services [10]

Heat declarative models take form of simple YAML documents. According to a Heat Orchestration Template (HOT) language [12] [13] [14], heat templates include the following four key elements:

- An optional *Parameters* section, which allows user-definable inputs to be specified
- An optional *Mappings* section, which allows key/value lookup of predefined constants

- A mandatory *Resources* section, which describes the resources and relationships defining the application, configuration and infrastructure
- An optional *Outputs* section, which describes the output values to be returned to the user

When the Heat template is deployed, a collection of infrastructure resources, known collectively as a *stack*, is instantiated, in the correct order and using the correctly inferred parameters. *Heat Resources* are native representations of the data types of the underlying OpenStack Services. All Heat resources have a common interface:

- A number of optional or mandatory *Properties* which specify inputs that affect how the resource is configured
- A number of *Output Attributes*, which may be referenced elsewhere in the template using the built-in reference function

In addition, a resource may reference any other resource or its attributes, whereby Heat infers dependencies between them, e.g., instantiation ordering. The Heat Stack is also a Heat resource, allowing the creation of hierarchical structures by nesting stacks. Importantly for COSIGN, Heat exposes two features for easy extensibility – a *Providers* feature that allows defining new resource types out of native resources; and an *Environments* feature that allows overriding parts of the template for specialized types of environments.

The basic architecture of Heat service is presented in Figure 4. Like most other OpenStack services, Heat architecture is multicomponent, with inter-component communications happening over the asynchronous message queue service. The Heat APIs exposed to the user provide template and stack management functionality, while the Heat engine is responsible for carrying out the API requests through communicating with the infrastructural OpenStack services, like Nova, Neutron, Swift, etc.

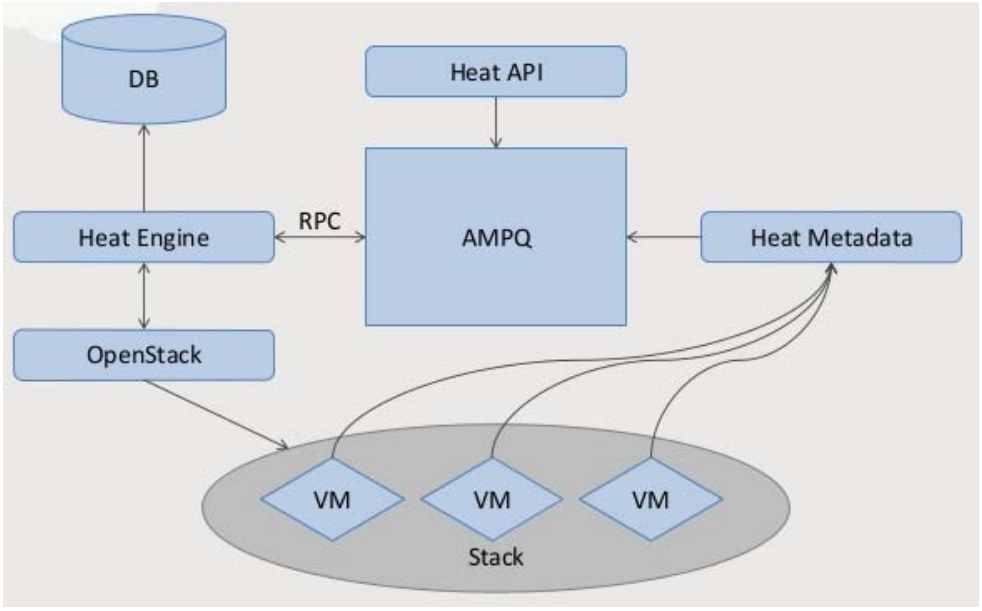

Figure 4 – OpenStack Orchestration Service (Heat) – Basic Architecture [15]

Zooming deeper into the Heat architecture, Figure 5 shows the breakdown of Heat into software components and the interfaces between them, highlighting the components most relevant to the COSIGN orchestrator. The highlighted components are the targets for COSIGN extensions, for supported network-aware orchestration and advanced capabilities of the COSIGN DCN.

It can be seen that Heat APIs consist of three separate libraries: the proper *heat-api* exposing Heat services to the OpenStack users through native REST APIs; the *heat-api-cfn* exposing Heat services through AWS CloudFoundry compatible Query APIs; and the *heat-api-cloudwatch* exposing AWS CloudWatch compatible service for publishing, monitoring, and managing various metrics, as well as for configuring alarm actions based on collected data. All the API libraries dispatch Heat requests through the asynchronous message passing mechanism so that they can be accessed and served by the

Heat engine(s). Heat engines parse the requests, maintain the internal database and administer resource creation and configuration through a layer of Service Clients that in turn call the REST APIs of the OpenStack services. Heat supports built-in resources for all the Core OpenStack services and, starting from the Icehouse release, can natively support all the AWS templates through native OpenStack resources and APIs. In addition, Heat allows extending the standard resources model by means of Resource Plugins that can represent new resources or override built-in resource implementations by supplying another implementation for the same resource type operators. Heat engines can include the so called watcher tasks that manage and configure monitoring engines, e.g., OpenStack Ceilometer [17] agents, as part of their resource management flows as dictated by the templates.

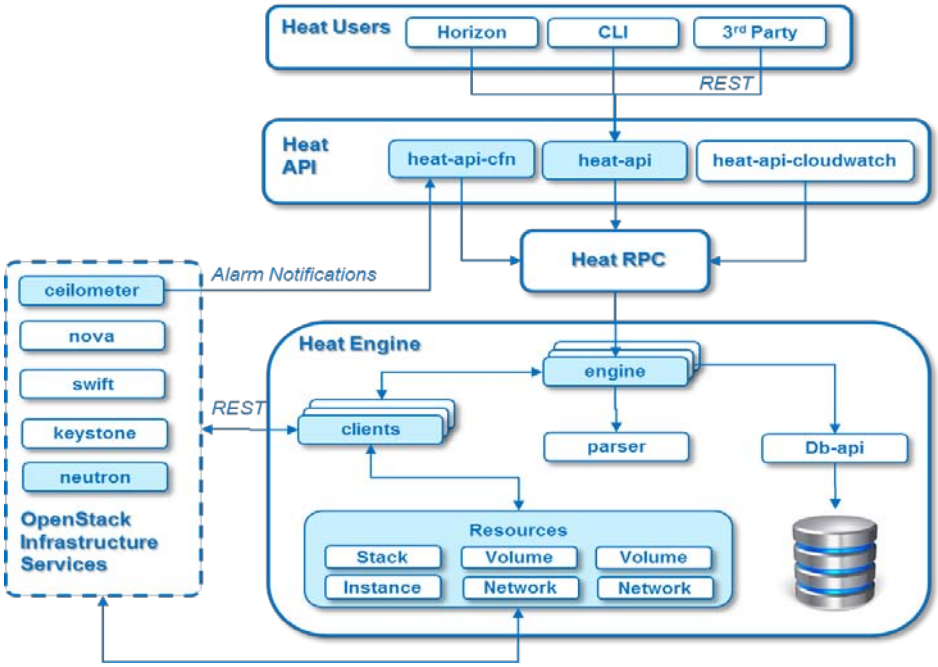

Figure 5 – OpenStack Orchestration Service (Heat) – Major Software Components and Interfaces

It must be noted that at first Heat was not used as a generic orchestration service but as a tool to achieve several specific goals, more or less related to orchestration and not available elsewhere then. For example, one of the first usages was automating the software configuration of the compute instances, as required for the application deployment. Today these capabilities, separated into cloud-init and other software configuration tools, are supported by Heat only for compatibility reasons and no longer form its major focus or responsibility. As software configuration is out of the direct scope of COSIGN, we'll not be focusing on these aspects of Heat throughout development, using preconfigured images for demonstration purposes.

An additional example of early Heat usage is autoscaling support, e.g., for implementing High Availability (HA), cluster elasticity, and Load Balancing (LB) scenarios. These scenarios can be of interest to the COSIGN team because, despite of focusing entirely on compute resources, their outcome is somewhat similar to what COSIGN plans to achieve by harnessing dynamic networking capabilities of the optical domain through the programmable SDN controllers. Although autoscaling support is planned to be taken out of the Heat and exposed as a separate service, Senlin [16], it is instructional to look at its architecture as it bears similarity to what is planned in COSIGN for the entirely different domain, networking. When Senlin is ready for prime time, it will expose its own resources for orchestration by Heat, helping to achieve higher levels of abstraction through composing more advanced groups of resources, or stacks.

Proper future developments in the Heat project include advances in HOT Language, enhancements of the dependency inference engine, improvements of scaling properties through parallel execution, evolving of the stack lifecycle, etc. All these, while very important for operational viability of the orchestrator service, bear little significance to COSIGN, as our major focus is on more adequate

network abstractions and services as well as on adding new networking capabilities enabled by the underlying optical technologies.

A major goal of the COSIGN orchestrator is to allow users to specify declarative models describing the intended connectivity situations and to process these models into resource allocation and configuration steps that will correctly reflect the modeled intention. In addition, a monitoring-based feedback loop must be implemented to trigger resource re-allocation and re-configuration for the already activated stacks. The above goals sound like a proper extension of Heat into the networking domain, above and beyond the simple management of individual resources exposed by today's Neutron. In addition, these goals sound similar to what was done in Heat to provide elasticity in compute resources, namely Heat autoscaling support. This observation makes Heat an obvious, well-justified choice on which to base the COSIGN orchestrator development. Adding to that, Heat is a well-supported project and is ubiquitously deployed as part of most OpenStack installations and vendor packages.

3.1.3 OpenStack Networking – Neutron

Although OpenStack Neutron is one of the core infrastructure services, already covered as such in Section 3.1.1, it is of special importance to COSIGN as a means to expose advanced networking capabilities of the optical data plane. According to architectural decisions made by the project team [D3.1], the COSIGN data plane is built with the virtualization overlay at the logical level and with the optical underlay at the physical layer, as shown in Figure 6. In COSIGN DCN, both the logical and the physical networks are operated through SDN controllers; the overlay controller for the logical networks and the OpenFlow (OF) controller for the physical network and its optical capabilities. To ensure seamless multilayer control, both SDN controllers are managed by a single extended Neutron service. This enables the OpenStack-based COSIGN orchestrator to coordinate the combined configuration of network overlay at the network edges and underlay at the optical DC data plane.

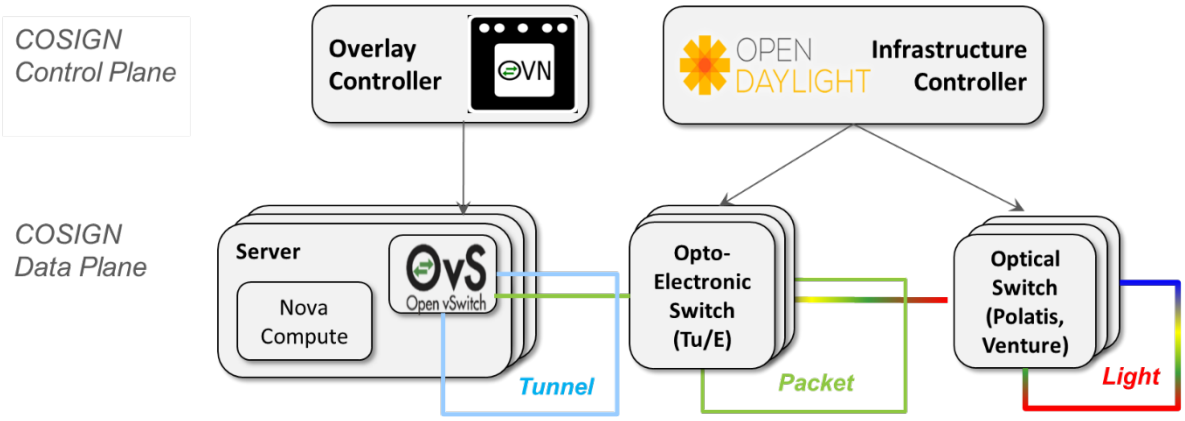

Figure 6 – COSIGN DCN architecture – the multi-layer data plane and the SDN control plane

OpenStack Neutron is designed to manage the logical networks and to completely isolate the cloud user from the physical network considerations. The core Neutron API, at L2 level, manipulates logical constructs representing virtual networks, virtual network subnets, and virtual network ports whereby compute instances can be interconnected. Specific implementations of networking functionality are realized through plugins, some produced by the community and some provided by vendors or third party organizations. Some plugins, e.g., NSX plugin by VMware, Ryu Controller plugin, Midonet plugin by Midokura, implement the logical networking in a way independent of the physical network, typically with edge-based overlays. Other plugins, e.g., CISCO, Juniper, and Arista plugins, implement logical networking through direct manipulation of the underlying physical network, through infrastructure virtualization with VLAN, VxLAN, etc. Both approaches are valid and yield working environments, each with their pros and cons, however it does not seem possible to combine the two approaches, tapping into the benefits of both simultaneously.

In COSIGN, such a combination is highly desirable. On the one hand, edge-based overlays are needed to achieve multi-tenant isolation at scale and independence between the logical domains. On the other

hand, tapping into the dynamic capabilities of the optical underlay is needed to improve resource utilization and provide differentiated services to applications. It is our goal, therefore, to extend Neutron by integrating the logical network controller and the physical network controller, for active configuration and for monitoring of the underlay network, in addition to and not instead of its regular role of the logical network controller.

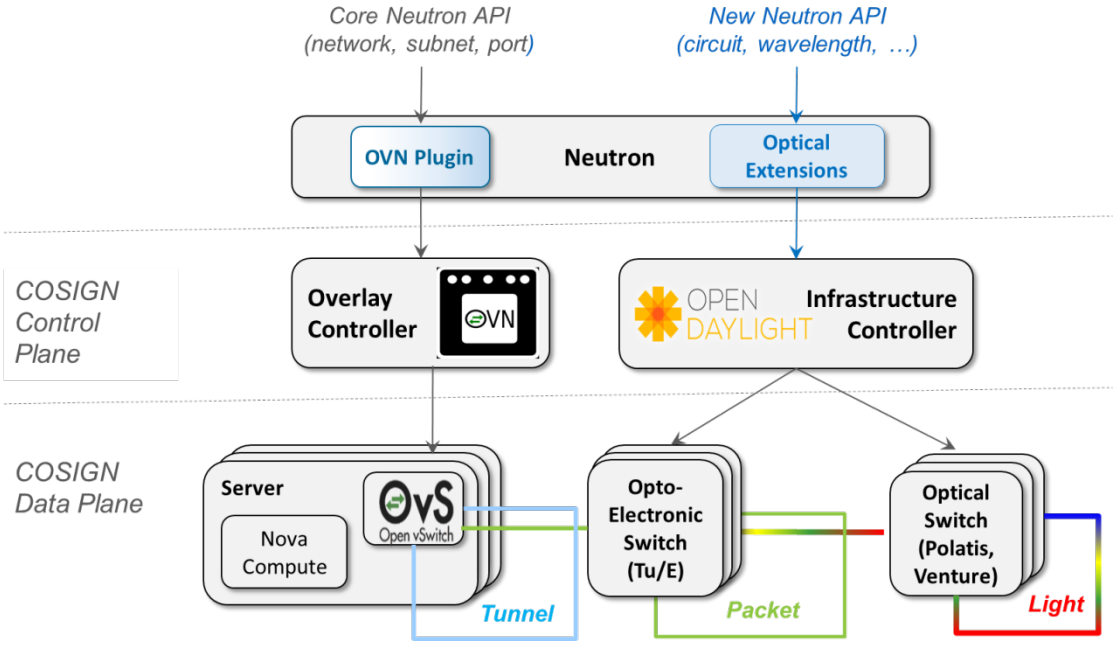

Figure 7 – COSIGN approach to extending OpenStack Neutron by adding an Optical Extensions to Neutron API

One way of extending Neutron, shown in Figure 7, is to allow the core networking API, logical in their essence, to be served through the overlay controller, by implementing the plugin suitable to the chosen overlay controller. In this case, the physical network control will be implemented through Neutron extensions, e.g., by using the Provider Network extension. This approach can be beneficially used to represent the optical capabilities of the COSIGN underlay. With the technologies represented in the COSIGN consortium, these capabilities are restricted to dynamic allocations of optical circuits that can provide fast links in an on-demand fashion and to managing optical TDM connections with the Venture switch. Future extensions can include support for additional capabilities, like bandwidth scheduling through dynamic WDM-based or TDM-based allocations.

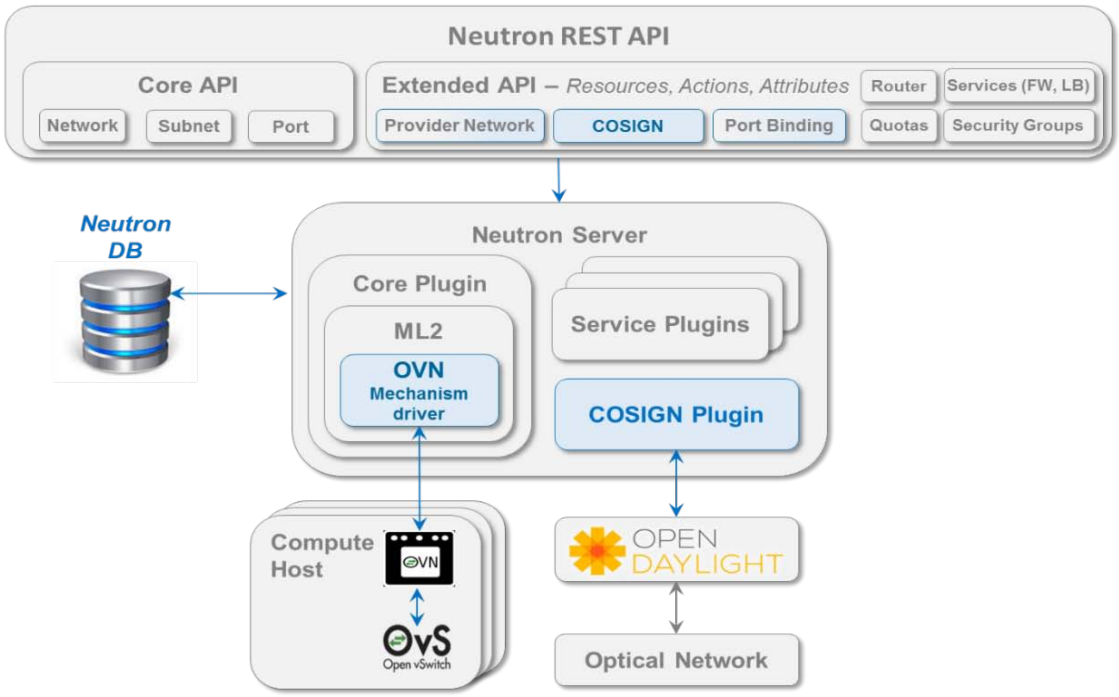

Figure 8 – OpenStack Networking Service (Neutron) – Major Software Components and Interfaces

Zooming deeper into the Neutron architecture, Figure 8 shows the breakdown of Neutron into software components and the interfaces between them, highlighting the components most relevant to the COSIGN orchestrator.

3.1.4 OpenStack Telemetry – Ceilometer

The OpenStack Ceilometer [17] project was designed for implementing cloud billing services. At first, only measurements useful for billing purposes were implemented. Later on, more and more measurements were collected across multiple OpenStack projects with no relation to billing, but for a broader set of purposes, e.g., monitoring, debugging and data visualization.

Data is collected by multiple agents of two basic types – *Polling Agents* that call APIs of other OpenStack services and communicate with external tools to obtain data and *Notification Agents* that listen to notifications sent over the message queue bus, both by the OpenStack components and by the external tools. Both types of agents publish the information to the message queue in a form of *Meters*, *Events*, and *Samples*, as shown in *Figure 9*.

Data collected by the Agents can be further processed in several ways – transformed, stored, and published. A combination of processing steps can be customized by creating processing pipelines from different transformers and publishers. Publisher's target can be the Ceilometer *Collector* service, the message queue, or an external system. The Collector service can store data in a configurable Ceilometer database which can also be directly fed from the external sources. The Ceilometer API allows accessing the monitored data, setting and reading meters, configuring alarms and events, etc. Ceilometer is configurable in several ways some of which are of relevance to COSIGN, namely creating new measurements, defining new Agent Plugins and integrating with external tools.

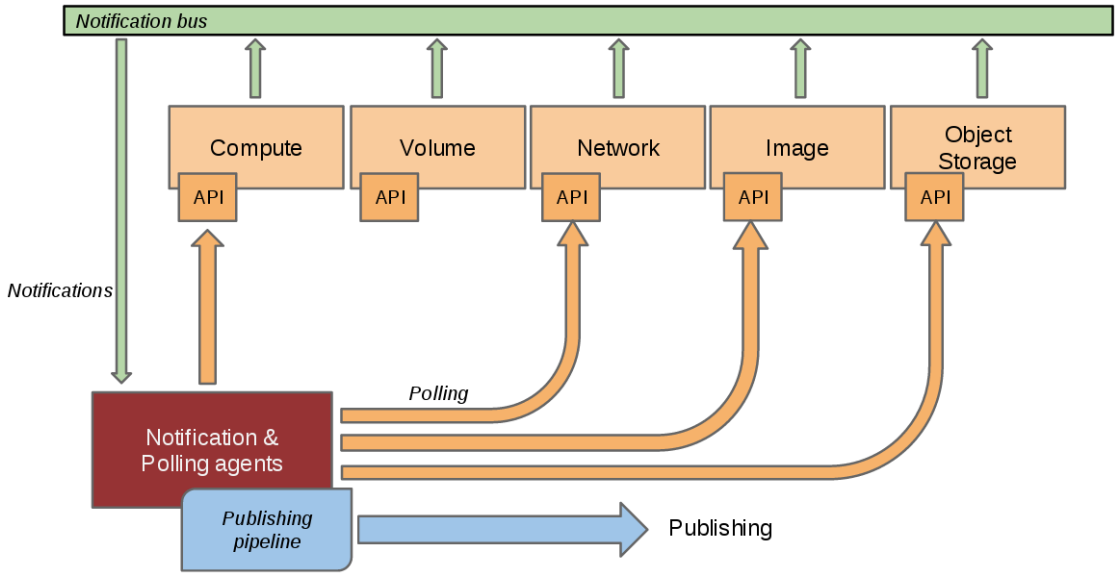

Figure 9 – OpenStack Telemetry Service (Ceilometer) – Data Collection by Agents [18]

To maintain the intended level of networking service, monitoring services must be deployed both in the overlay and in the underlay, to feed the decision making algorithms, which in turn can generate further resource reconfiguration requests. Understanding the measurements currently available for collection in both domains is therefore of outmost importance to COSIGN as a first step to figuring out what new measurements will have to be added as part of the project development, to support the decision making algorithms. Below we list currently available network related measurements provided by different OpenStack services that may potentially be used in the COSIGN orchestrator[1].

OpenStack Ceilometer collects measurements either by polling the infrastructure elements or by consuming the notifications emitted by other OpenStack services. Measurement points are called *meters* and can be of the following types:

- *Cumulative* meters are used for quantities increasing over time
- *Delta* meters are used for quantities changing over time
- *Gauge* meters are used for discrete and fluctuating values

Currently available meters relevant to COSIGN are organised in tables below according to their source.

Table 1 shows meters provided by OpenStack Compute, available for Libvirt, Hyper-V, vSphere and XenAPI hypervisors.

Meter	**Type**	**Unit**	**Resource**	**Description**
network.incoming.bytes.rate	Gauge	Bytes/sec	interface ID	Average rate of incoming bytes.
network.outgoing.bytes.rate	Gauge	Bytes/sec	interface ID	Average rate of outgoing bytes.
network.incoming.packets.rate	Gauge	packet/sec	interface ID	Average rate of incoming packets.
network.outgoing.packets.rate	Gauge	packet/sec	interface ID	Average rate of outgoing packets.

Table 1 – OpenStack Telemetry Service (Ceilometer) – Compute Measurements [19]

[1] Additional measurements, e.g., compute and storage related, can also be used by the COSIGN Orchestrator. There are no plans, however, to extend the currently available set of measurements outside the networking domain.

Table 2 shows SNMP based generic meters provided by hosts running `smpd`, measured per host interface.

Meter (prefix: hardware.network)	**Type**	**Unit**	**Resource**	**Description**
prefix.incoming.bytes	Cumulative	Byte	interface ID	Bytes received by network interface.
prefix.outgoing.bytes	Cumulative	Byte	interface ID	Bytes sent by network interface.
prefix.outgoing.errors	Cumulative	packet	interface ID	Sending error of network interface.
prefix.ip.incoming.datagrams	Cumulative	datagrams	host ID	Number of received datagrams.
prefix.ip.outgoing.datagrams	Cumulative	datagrams	host ID	Number of sent datagrams.

Table 2 – OpenStack Telemetry Service (Ceilometer) – SNMP Measurements [19]

Table 3 shows the only relevant measurement currently provided by Neutron.

Meter	**Type**	**Unit**	**Resource**	**Description**
bandwidth	Delta	Bytes	label ID	Notification

Table 3 – OpenStack Telemetry Service (Ceilometer) – Neutron Measurements [19]

Finally, *Table 4* shows many interesting per port and per flow measurements available through SDN controllers that control OpenFlow switches. These measurements are currently supported by OpenDayLight [6] and OpenContrail [20] plugins.

Meter (prefix: switch.port)	**Type**	**Unit**	**Resource**	**Description**
prefix.receive.packets	Cumulative	packet	switch ID	Packets received on port.
prefix.transmit.packets	Cumulative	packet	switch ID	Packets transmitted on port.
prefix.receive.bytes	Cumulative	Bytes	switch ID	Bytes received on port.
prefix.transmit.bytes	Cumulative	Bytes	switch ID	Bytes transmitted on port.
prefix.receive.drops	Cumulative	packet	switch ID	Drops received on port.
prefix.transmit.drops	Cumulative	packet	switch ID	Drops transmitted on port.
prefix.receive.errors	Cumulative	packet	switch ID	Errors received on port.
prefix.transmit.errors	Cumulative	packet	switch ID	Errors transmitted on port.
prefix.receive.frame_error	Cumulative	packet	switch ID	Frame alignment errors received on port.
prefix.receive.overrun_error	Cumulative	packet	switch ID	Overrun errors received on port.
prefix.receive.crc_error	Cumulative	packet	switch ID	CRC errors received on port.
prefix.collision.count	Cumulative	count	switch ID	Collisions on port.
prefix: switch.table				
prefix.active.entries	Gauge	entry	switch ID	Active entries in table.
prefix.lookup.packets	Gauge	packet	switch ID	Lookup packets for table.
prefix.matched.packets	Gauge	packet	switch ID	Packets matches for table.
prefix: switch.flow.duration				
prefix.seconds	Gauge	sec	switch ID	Duration of flow in seconds.
prefix.nanoseconds	Gauge	nsec	switch ID	Duration of flow in nanoseconds.
prefix: switch.flow				

prefix.packets	Cumulative	packet	switch ID	Packets received.
prefix.bytes	Cumulative	Bytes	switch ID	Bytes received.

Table 4 – OpenStack Telemetry Service (Ceilometer) – Neutron Measurements [19]

To summarize, *Figure 10* zooms into the Ceilometer architecture, showing its major components and interfaces between them, and highlighting the components most relevant to the COSIGN orchestrator. While the OpenStack Ceilometer was successfully deployed to achieve Compute autoscaling with Heat, there is evidence that Ceilometer may not in itself be sufficient for monitoring all the required networking aspects. External tools and protocols for network monitoring and event processing are considered by the COSIGN team for inclusion at later stages of the project development, according to the exact needs, tool availability, and further analysis of this landscape. Examples of such tools are: Open Flow based statistics collection from edge switches, e.g., ToRs and virtual hypervisor switches, sFlow collectors, Riemann Network Monitoring Framework [21], etc.

In the context of the COSIGN orchestrator it is required to collect network related information both from the overlay and from the infrastructure and to deliver it to the orchestration layer. This information, e.g., network topology, network capabilities and costs, performance and failure monitoring data, is a necessary input to the DCN orchestration and resource scheduling algorithms and can be used to make network-aware decisions about the provisioning and the management of other cloud resources (i.e., computing and storage) through management and operation actions (e.g., instantiation of VMs in specific servers, migration, etc.). There are several possibilities for collecting and delivering this network specific data: 1) from the device specific agents to OpenDayLight to Neutron and, from here, to the entire cloud management platform through the mediation of the OpenStack Ceilometer monitoring component; 2) from the device specific agents to OpenDayLight and from the OpenDayLight client directly to Ceilometer, without the Neutron mediation [22].

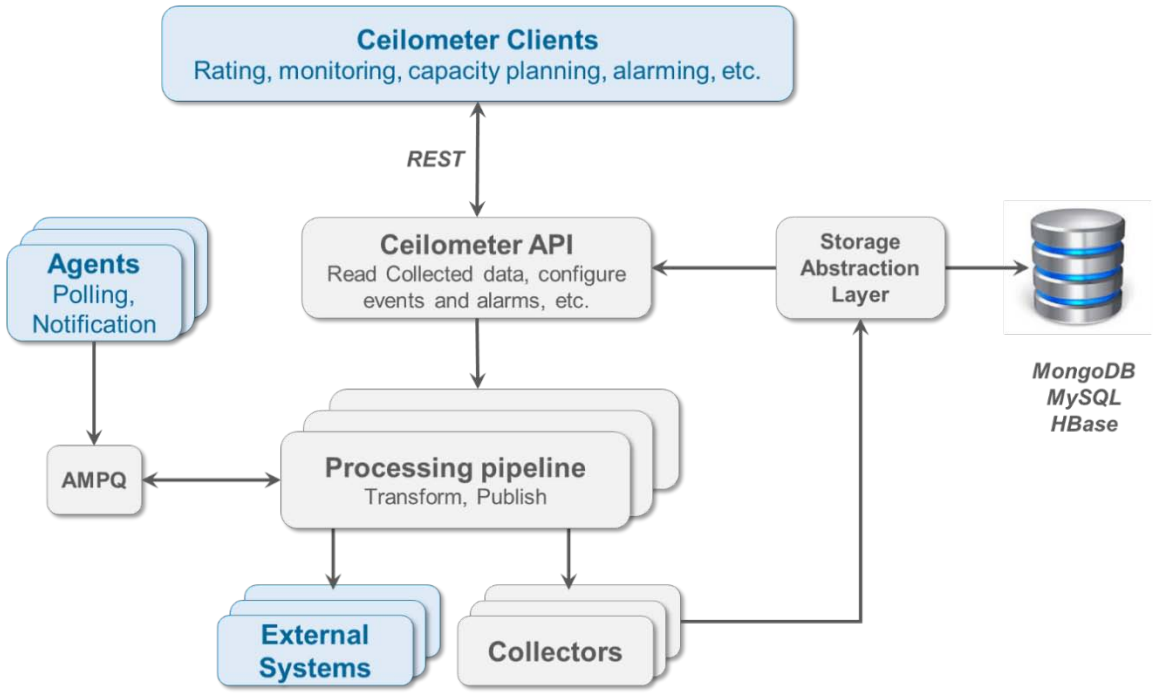

Figure 10 – OpenStack Telemetry Service (Ceilometer) – Major Software Components and Interfaces

3.1.5 OpenStack Group Based Policy (GBP)

The Group Based Policy (GBP) service of OpenStack [23] provides declarative abstractions for achieving scalable, intent-based cloud infrastructure automation. While conceptually referring to all the DC resources, GBP was conceived and first realized specifically for networking. There are several reasons for this.

The first reason lies in the properties of the network as a very special type of resource, inherently different for the rest of the IT resources, e.g., from the compute and storage, and traditionally more difficult to virtualize. For example, as opposed to the compute and storage, network resources are not located in a single physical node or a pool of nodes, but are distributed between devices of different types – forwarding devices, links, network cards, connectors, service appliances, etc. Network links, paths, and nodes are shared more often than not, so that common best practices call for isolation tools and not for increased consolidation and sharing.

The second reason is that the BGP project and its policy-based approach to infrastructure provisioning were originated by a major networking vendor, Cisco [24], more specifically by its Application Centric Infrastructure (ACI) project, where GBP is successfully and efficiently deployed in practice. Integration of Cisco's ACI, through OpenDayLight GBP [25] and OpenStack GBP [26] projects, by users and third parties has shown the validity of the approach and its potential to bring benefits to a broader networking and cloud industry.

In the present OpenStack release, GBP complements the OpenStack networking model with the notion of *Policy* that can be applied between groups of network endpoints. As users look beyond basic connectivity, richer network services with diverse implementations and network properties are naturally expressed as policies. Examples of advanced policies include service chaining, QoS, path properties, access control, etc. GBP is currently realized as Neutron extension although it will probably be spawned into a separate service very soon and might even be extended to allow declarative management of additional IT resources in future releases.

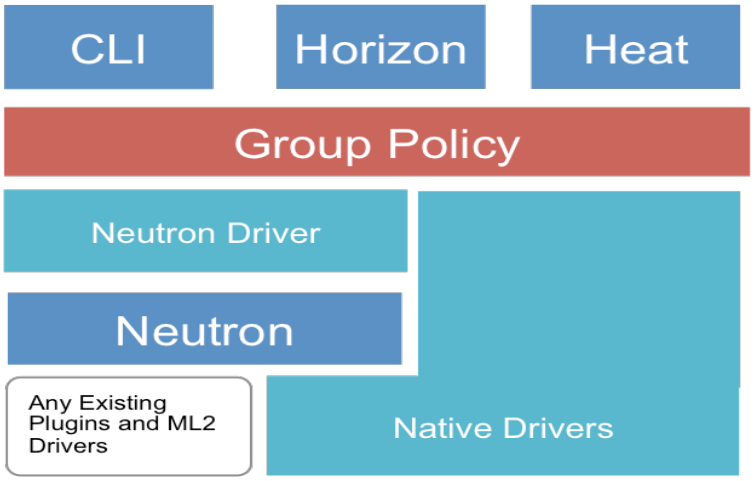

Figure 11 – OpenStack Policy Service (GBP) [27]

As OpenStack Neutron extension, GBP is an API framework offering an intent-driven model for describing application requirements in a way that is independent of underlying infrastructure. Rather than offering network-centric constructs like L2 domains, subnets, and ports of Neutron, GBP introduces generic primitives of Policies and Policy Groups to describe connectivity, security, and network services between groups of network endpoints. GBP thus allows application administrators to express their networking requirements using declarative abstract model, leaving policy rendering to the specifics of the pluggable policy driver which translates the declarative model into the imperative operations on the underlying networking model of OpenStack – Neutron, as shown in *Figure 11*.

Located directly below the OpenStack Orchestration (Heat) and above the OpenStack Networking (Neutron and its extensions), the Policy service allows formulating more abstract and high-level requests towards the infrastructure than is enabled by Neutron. The COSIGN orchestrator, having similar goals, can benefit from the GBP, either by using and extending it into directions of our interest, e.g., tapping into the optical data plane benefits, or by building a new component, specializing in the optical extensions, but having a similar architectural role as the GBP.

Going forward, COSIGN envisions extension of the GBP or the GBP-like Policy service into additional IT resources above the networking, to yield the benefits shown in *Figure 12*.

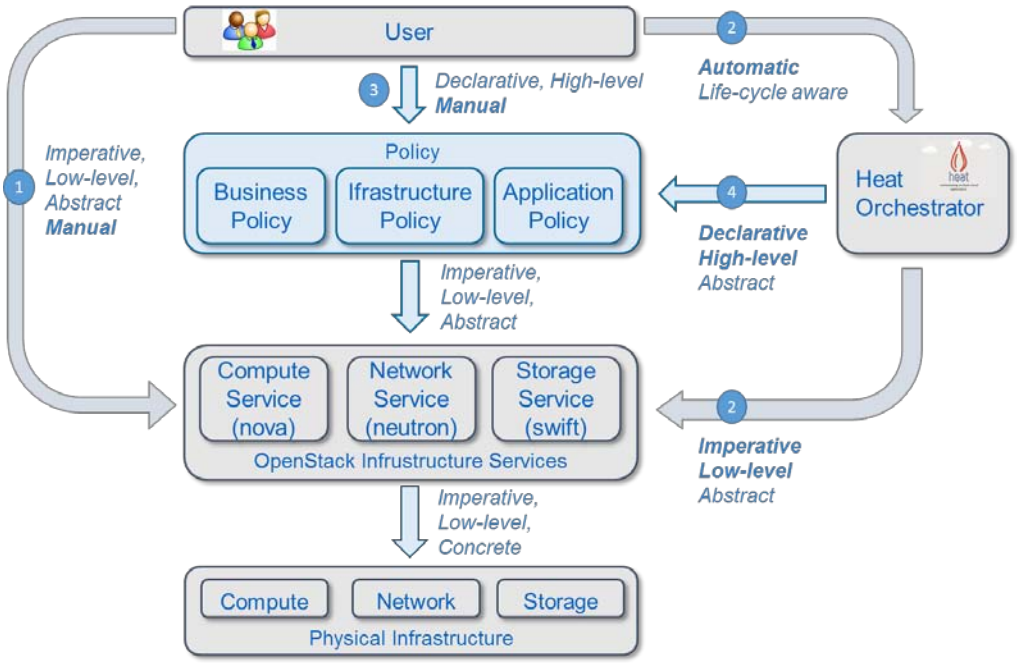

Figure 12 – Benefits of the Extended Policy Service

It can be seen that although the existing OpenStack Shared Infrastructure services (nova, neutron, swift) abstract and virtualize the physical infrastructure resources, the interfaces they provide are still very low level and imperative, requiring knowing and specifying minute level of details. A user, accessing the infrastructure through these services, acting as indicated by the route marked (1) in *Figure 12*, is freed from the platform and technology specific details due to virtualization, but still has to issue imperative commands, either manually or with scripts, exerting low-level controls onto the abstracted physical infrastructure. In the absence of a Policy service, a user who has chosen to operate the infrastructure through Heat, acting as shown by the route marked (2) in *Figure 12*, enjoys smart orchestrated automation not available to a user directly calling the APIs of the Infrastructure Services now called by Heat on his behalf. Far more useful than option (1), this option requires low level Infrastructure control from Heat, complicating it with unneeded domain specific details. It is therefore hard to implement advanced cross-domain scenarios such as those envisioned in COSIGN with this architecture.

A Policy service enables declarative high level definition of the way resources should be provisioned and used. Policies can come from the workload, from the infrastructure operator, and from the business owner, each defining rules to be followed and actions to be executed. A user of the policy service, acting via route (3) in *Figure 12*, enjoys the ability to express the intended structure of the workload, the intended way of resource usage, or the intended way of compliance enforcement. Not going through Heat, however, makes the operations manual, missing the benefits of the lifecycle management and feedback loops provided by Heat. This can be resolved by integrating Heat with the Policy service, whereby the User, acting as marked by routes (2)+(4, can enjoy an automated, orchestrated, intent-based, high-level, and abstract way of infrastructure resource management. Of course, in this case, the template provided by the user to the Heat service in route (2) must include higher level GBP-borne constructs. There is a community driven initiative aiming to integrate Heat with GBP. This work is only beginning, and it is not clear yet what artifacts will be available for reuse, and what will have to be created from scratch if we choose to include GBP as part of the COSIGN orchestrator.

The COSIGN team recognizes the benefits of attaining the level of service provided by GBP and aims to include it at the architecture level of the orchestrator. At the design and implementation level, the COSIGN orchestrator will be mostly network focused as we are building the Next Generation Optical DCN and not the entire DC. We envision that COSIGN use cases involving combined orchestration of networking and compute resources are possible without extending the network policy service to

additional types of resources, and that the use cases can be supported through flexing the interactions between the existing components to enable cross-domain decision making.

3.2 Orchestration Development Framework

In this section we first outline the OpenStack based architecture of the COSIGN orchestrator and then provide the details of the OpenStack based collaborative development framework.

3.2.1 COSIGN Orchestrator Architectural Summary

Figure 13 presents a high level overview of the COSIGN orchestrator to be built from the OpenStack components described in Section 3.1, and its placement in the overall COSIGN architecture – above the data plane and control plane layers and directly below the DCN client layer.

The COSIGN orchestrator will consist of three major components:

- OpenStack Heat, extended with new templates and resource plugins;
- OpenStack Neutron extended to allow access to the optical infrastructure controller in addition to providing logical network abstractions of core Neutron API through edge based overlays;
- A new OpenStack component, shown here as *COSIGN DCN Orchestrator*, that will enclose the smarts of COSIGN resource allocation and management on a higher level of abstraction than is possible with Neutron;

To facilitate understanding, Figure 13 omits several details, partially for simplicity and partially because this first interpretation of the orchestrator will be further refined and extended as development progresses.

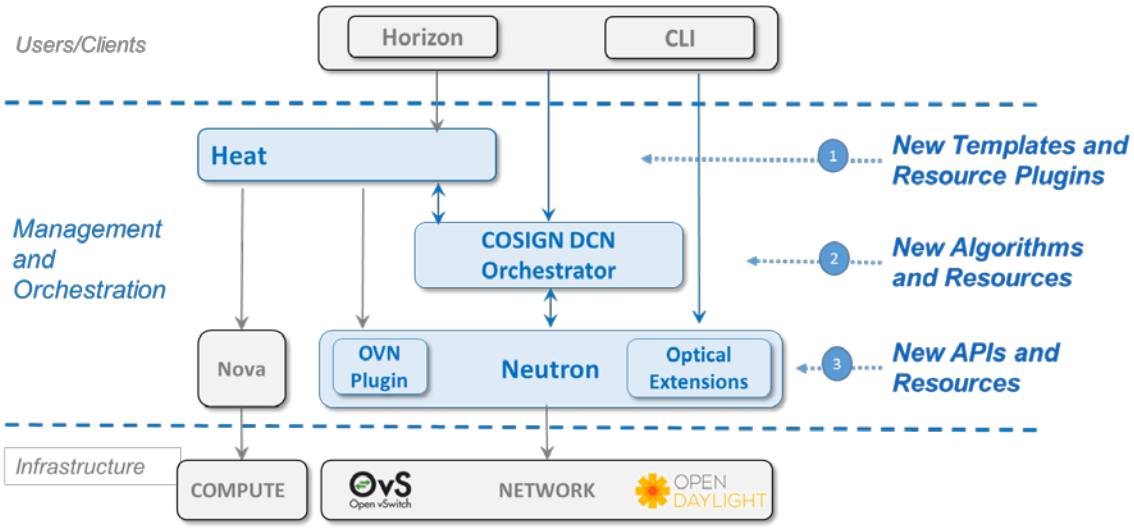

Figure 13 – COSIGN DC Orchestrator – Major Software Components, Interfaces, and OpenStack Extensions

First, we show here only the networking and the compute resource controllers, aiming to coordinate between them at the initial stage and to add storage considerations later, as the more complex use cases are considered. Technically, coordinating with storage is not overly different from coordinating with the compute resource controller, justifying the plan to have it as advanced add-on feature. We find it beneficial to learn from combining the networking and compute controllers before adding yet another type of resource, instead of starting with a far more complex environment upfront.

Second, Figure 13 does not show any metering components and how they fit into the picture and the flows. This is for simplicity of the figure only; monitoring flows is covered later on in this document where more detailed views on specific components and interactions are presented. That said, and taking into account the OpenStack Ceilometer survey in Section 3.1.4, incremental extensions and add-ons to the metering functionality will be introduced as needed throughout the implementation stages.

Third, Figure 13 is not providing exact interfaces where the combined orchestration of networking and compute will happen. At the moment several alternative approaches exist for addressing this most advanced challenge from the software architecture standpoint, namely providing advanced support for dynamic workloads simultaneously, from the perspectives of networking, compute and storage. One possibility is to extend the 'COSIGN DCN orchestrator' to include multi-resource intent models and to communicate with additional resource controllers, similarly to one direction proposed for GBP project evolution. Another approach is to preserve the decoupling between the resource groups to the extent possible, and contain all the dependencies inside the Heat template definitions. The COSIGN orchestrator team plans hands-on experimentation with both these approaches. The decision will be made at a more advanced stage of the design and implementation process, when exact specifications of all the use cases have been completed and combined into a unified architecture.

3.2.2 Orchestrator Development Setup

The COSIGN software architecture is very complex and requires coordination between the different teams, across partner organizations and Work Packages. The Orchestrator development team must have a sufficiently stable and reproducible environment, where different features can be developed, unit tested, and integrated. Task 4.3 calls for integration testing of the WP4 DC Orchestrator with the WP3 SDN controller layer. For initial validation of the use cases, suitable front end interfaces must be developed. Although full end-to-end integration will take place in WP5, we will be adding scaffolding to emulate components and interfaces external to WP4 in order to ensure integration-ready WP4 outcomes. As the integration efforts and full test bed integration proceed, the scaffolding will be replaced with the real functional components, both software and hardware. In addition, the development teams will need to devise synthetic tests to induce load into the system, for functional and rudimentary performance testing, in anticipation of testing under more realistic workloads in WP5. Last, but not least, it is desirable to be able to assess scaling properties of the entire software architecture during development.

Taking into account the above requirements towards the development framework, we have performed a study and experimentation with different ways to deploy OpenStack for development purposes. Although the idea of fully automated roll-out of OpenStack installations is highly attractive, the decision was not to go to the higher end of devOps complexity because of our need to remain agile with respect to the included components, their configuration, extensions, hooks to external systems, etc. We do not want to induce premature hardening of the environment on the one hand, while on the other hand taking precautions not to end up with a fragile or non-reproducible setup. At the moment, we consider Devstack [28] to be the most suitable tool for OpenStack roll-out.

DevStack is an opinionated script to quickly create a working OpenStack development environment with a minimal set of the required components, with a reasonable set of defaults allowing newcomers to experiment with OpenStack on a small scale, even on a single laptop. DevStack has tooling to manage the OpenStack services from a convenient set of screens and to access them with pre-configured authentication tokens. In addition to a main script, *stack.sh*, that builds and initiates the OpenStack environments according to provided configuration parameters, DevStack includes examples of using OpenStack services from a command line and complete scripts intended as exercises but useful as a quick sanity check for the OpenStack installation. Since its inception, DevStack has evolved to support a large number of configuration options, alternative platforms, and combinations of services. As OpenStack becomes more complex, with new services and usage patterns added on a regular basis, using DevStack to create a specifically needed installation requires more skills and experience. Therefore, building a COSIGN-specific flavour of DevStack to be used as a basis for the community development has been defined to be the first tasks to follow after finalization of the High Level Design.

Initially, the COSIGN DevStack should include at least the following OpenStack services: Nova, Keystone, Horizon, Glance, Heat, and Neutron. On the one hand, including optional services that we might need, e.g., GBP, Ceilometer, one of the storage services (Cinder or Swift), from the beginning is desirable, as adding services to existing DevStack installation can be rather traumatic and might even require starting almost from scratch. On the other hand, the more components we add upfront, the more complex the task of creating a working COSIGN DevStack becomes. In addition to installing the

right set of OpenStack services, the script should figure out the dependencies and be accompanied by instructions regarding hardware and software requirements and prerequisites, and by a sanity test script that will validate the installation. The COSIGN flavour of DevStack must support different sizes of OpenStack installation, ranging from all-in-one installation in a single node running the control services and the compute host[2] to a multi-node installation with a separate control node and multiple compute hosts arranged in several subsets for emulating multi-rack environments. For development and functional testing, the installation can run in a virtualized setting without being connected to real DC fabric. For performance and scale testing, bigger installation on dedicated hardware must be supported, with an access to emulated DC fabric. This also can be used to a standalone demonstration of the orchestrator capabilities, as well for as for demonstrations integrating the orchestrator and the controller layer (WP3). For end-to-end demonstration purposes the environment will be transferred into a test bed setting (in WP5) and connected to a real DC fabric (from WP2).

The COSIGN DevStack is already under development, taking into account all the considerations, requirements, and trade-offs described above. Upon completion, every team member will be able to download COSIGN DevStack, prepare a development machine as prescribed in the instructions, run the main script on a suitably prepared development machine, and validate the installation using helper scripts. An additional possibility will be distributing a Virtual Machine that includes the entire environment enclosed in containers, for development purposes only. Figure 14 presents an example of the COSIGN orchestrator environment for development and demonstration purposes that is intended to run on a dedicated hardware host and include all the required components – OpenStack services and SDN controllers – as virtual machines.

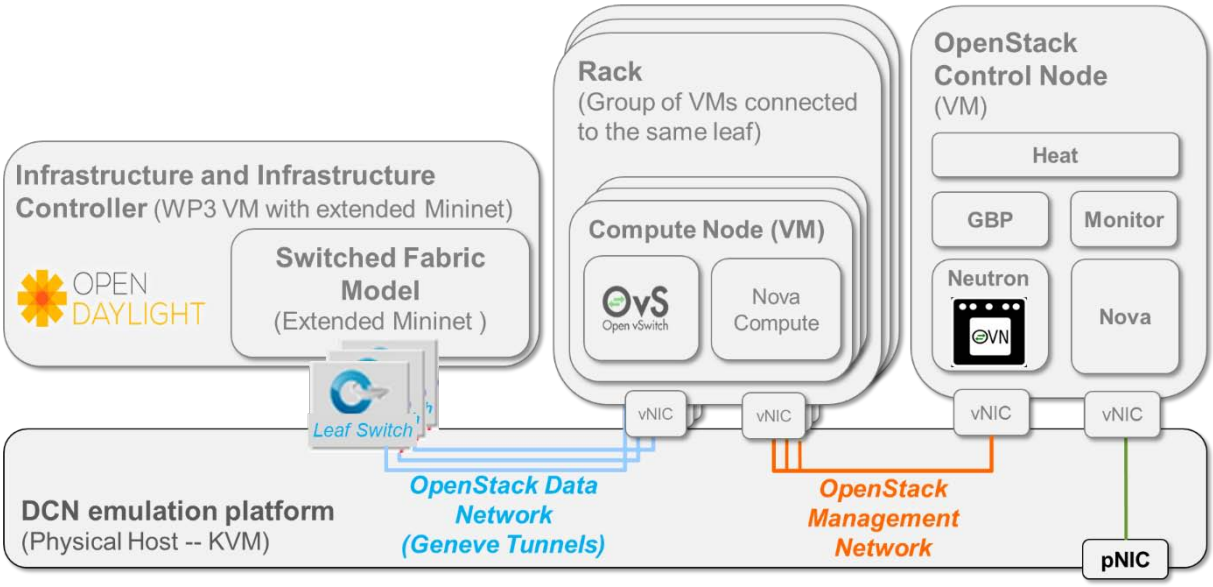

Figure 14 – COSIGN Orchestrator – Example Development and Demonstration Environment

The physical host shown in Figure 14 represents an emulation platform for the DCN fabric. It runs hypervisor, e.g., Kernel-based Virtual Machine (KVM [29]) and Linux networking tools, e.g., Linux Bridge [30] or Open Virtual Switch (OVS or Open vSwitch [31]), to emulate OpenStack networks. Figure 14 shows two networks – one OpenStack management network and one OpenStack guest's data network. In the future, additional networks can be implemented, e.g., external network and storage network. The DCN emulation platform runs several VMs – 1) the OpenStack controller VM with the services, connected to the management and to the external networks; 2) a set of VMs running OpenStack Compute hosts, connected to the management and to the guest data networks; and 3) VM running the COSIGN SDN controller and emulating the DCN fabric with mininet [32].

[2] The smallest installations such as the all-in-one virtualized setups are useful for several purposes – for building and debugging the environment creation process itself; for learning purposes such as educating team members new to the OpenStack; for developing and testing changes of higher level components that do not touch multi-host networking, e.g. REST clients or Horizon extensions.

4 Orchestrator's Role in COSIGN Use Cases Realization

Architecturally, the COSIGN Orchestrator is the top part of the software stack, and as such directly responsible for realizing all the COSIGN use cases through interaction with the underlying software and hardware layers. Three major groups of usage patterns were defined for COSIGN in earlier deliverables [D1.1] [D4.1]: 1) the Virtual Data Centre (**VDC**) use case where the User is a DC or Cloud operator requesting to maintain a sufficient level of control over physical aspects of his part of a large-scale infrastructure of an IaaS provider; 2) the Virtual Application (**vApp**) use case where the User is an application developer and/or operator who cares only about the operational aspects of his workload and not about the infrastructure; 3) the Data Centre Operations (**DC-Ops**) use case where the User is the infrastructure owner and/or operator who cares about the operational and the business aspects of the cloud itself, aiming at reducing costs and increasing revenues through smart infrastructure management and control.

It must be understood that the use cases above include a multitude of aspects above and beyond the immediate COSIGN focus – the networking. In the following sections, we describe these use cases from the networking perspective. We show specific scenarios where the COSIGN Orchestrator brings value to Users of different types, by tapping into unique capabilities of the COSIGN network – SDN-driven programmability and optics-powered forwarding.

4.1 Virtual Data Centre Use Case

This section details the Virtual Data Centre (VDC) use case, expanding the description already found in previous deliverable of WP4, [D4.1], and providing the orchestrator's requirement both from/towards the SDN controller and from/towards the User layer. Additionally, a full description of the operational model and workflows for VDC provisioning is provided.

4.1.1 Use Case Description from the Orchestrator Perspective

Virtual Data Centre (**VDC**) is a case of Infrastructure as a Service (IaaS) concerned with providing advanced infrastructure solutions to multiple **VDC owners** over a single shared physical infrastructure. Essentially, a VDC tries to replicate the capabilities of a physical DC, providing a fully manageable and operable infrastructure that can grow or shrink according to the needs of the VDC owner and **VDC clients**, but without the associated maintenance costs. Hence, a VDC is mainly designed to allow VDC owners to design, deploy and manage cloud services, intended to be offered to VDC clients, through a dedicated infrastructure composed of compute, storage, network, and security appliances that fulfil specific service requirements.

From a point of view of DC physical infrastructure provider, called for the sake of this use case **VDC provider**, VDC provisioning involves joint orchestration of compute and network resources so as to guarantee the fulfilment of the VDC Owners' requests. A key point of the whole provisioning process relates to the isolation of different **VDC instances**, both from a logical and from a physical perspective so as to guarantee that VDC instances do not interfere with each other and fully independent control of them can be exerted. From the compute side, such constraint involves the placement of **VM**s in a way that their operation does not tax the server in which they are deployed, hence, no performance degradation can be perceived. On the other hand, from the network perspective, since VDC instances must be provided to enforce QoS guarantees (bandwidth, latency, etc.) for connectivity between VMs, this results in a partitioning of the physical network infrastructure, e.g. by means of slicing, where each partition is reserved for the sole use of a particular VDC instance. From the orchestrator's perspective, the control and optimization of the routing and allocation of resources is a key element to guarantee that the underlying physical infrastructure is utilized efficiently while satisfying the requirements posed by multiple tenants.

The **VDC** s**ervice** identified in COSIGN, inspired by already existing commercial services such as Interoute VDC [34] or Amazon Virtual Private Cloud (VPC) [35], consists of several VDC instances deployed on top of the same physical infrastructure, offering fully operable and manageable virtual infrastructures, with strict QoS parameters, security and reliability guarantees, which will be utilized

for serving applications oriented to be consumed by end users. Such virtual infrastructure is then mapped to the corresponding physical resources. The structure of a VDC instance mainly consists of:

- VMs, characterized by CPU, memory and local storage requirements. They are the virtual homonyms of physical servers. Such virtual machines will be loaded into the servers with a bare Operating System (OS) image, equipped with the desired resources. Then, users' applications can be installed on top of the base image. Additionally, VMs may be organized/deployed following specific rules set up by affinity groups which define collocation/anti-collocation relationships between VMs so as to keep an optimized performance of the whole VDC.
- A virtual network (**vNet**), representing the communication needs between the different pools of VMs. Hence, a vNet is characterized by a topology, which consists of virtual nodes (**VM pools**) and virtual links (**vLinks**), which may state the bandwidth required between virtual nodes as well as other parameters such as communication latency requirements. Besides the topology representation, a vNet comprises a set of virtual firewalls (**vFW**) and virtual load balancers (**vLB**) that implement the specific security and traffic handling policies inside the VDC instance.

Figure 15 depicts a multi-tenant VDC hosting scenario and the mapping of the virtual resources to the physical networks. Take in mind that, since COSIGN network scenario is based on optical technologies, it may be the case that several VDC instances, managed by different VDC owners, can share the same physical path for mapping a virtual link. Nevertheless, VDC instances can be fully isolated by reserving independent wavelength/port/optical-TDM spaces to the multiple co-existing VDC instances, so traffic from different VDC owners does not mix.

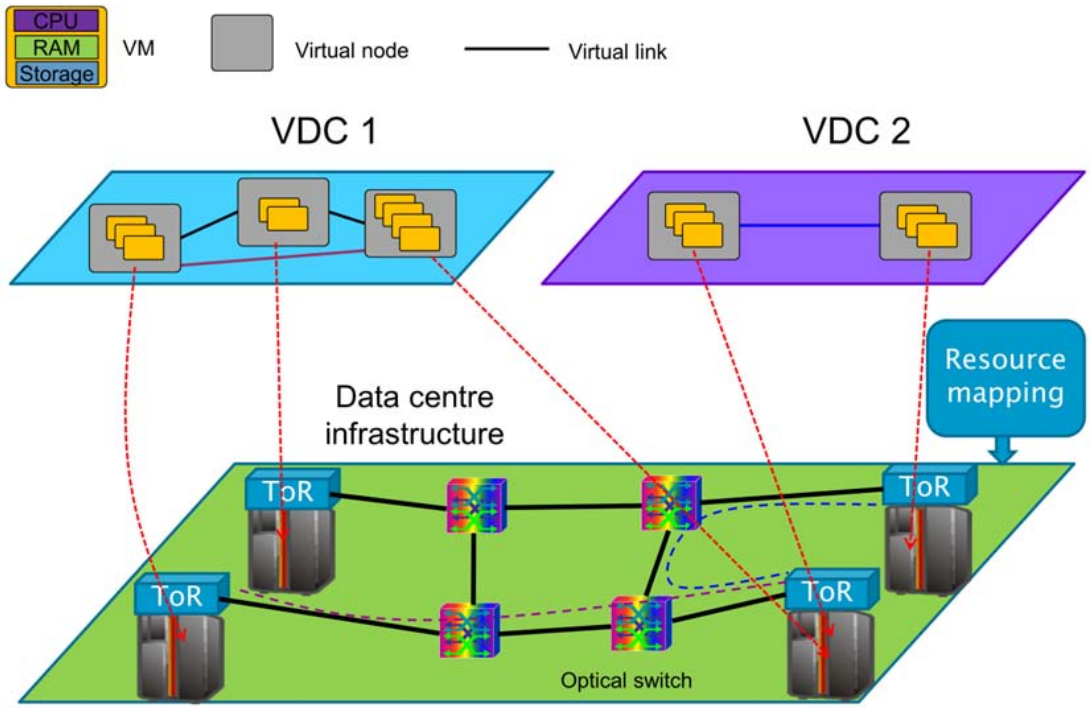

Figure 15 – Example of multi-tenant VDC service scenario.

The deployment process of a VDC instance starts with an infrastructure request from a VDC owner. A VDC owner will specify the desired VDC instance through the VDC dashboard service, which allows for the specification of the characteristics (topology, VM templates, security, etc.) of the VDC. The VDC dashboard service will be the main access point for VDC owners to the VDC environment, allowing for the instantiation of the resources of the VDC. An additional dashboard service will allow the monitoring of the deployed infrastructure in terms of health and performance as well as its operation. This second dashboard, dedicated to each VDC instance, may be customized for each VDC owner, providing specific tools (i.e. programmable APIs) that allow for the management and the

operation of the VDC. Once the VDC owner has fully specified the characteristics of the VDC request, such request will then be issued to the DC orchestrator. The orchestrator will compute the optimal resource mapping of the VDC request according to the specifications of the VDC owner, the available resources of the underlying physical infrastructure and internal policies of the DC operator. After deciding the optimal mapping of the virtual resources into physical ones, the orchestrator will begin with the deployment and the configuration of the VDC. For this, the orchestrator will coordinate different services, one for each type of resources (mainly, compute and network), passing to them the desired parameters of the resources to be configured. Once the VDC has been properly configured, a representation of the configured infrastructure will be sent to the VDC dashboard. From this moment, the VDC owner can start to operate and manage its VDC instance. For this, proper tools will be exposed towards the VDC owner for accessing, controlling and monitoring each type of resources, e.g. public IPs for accessing the VMs and programmable APIs to give control over all the components of the VDC, both compute and network.

Besides the described VDC service, in COSIGN we aim to provide more advanced functionalities in VDC instances in order to provide an enhanced virtual network environment which will allow for the optimization of the VDC owner resources. With this purpose, COSIGN optionally envisions the possibility to extend the previously described virtual network, which only consists of virtual links connecting VM pools, with virtual Top of the Rack (**vToR**) switches and virtual Optical Switches (**vOS**). With such an extension, a VDC instance becomes the virtual replica of the physical infrastructure of a DC (see Figure 16), providing the possibility to perform switching operations of the interconnections between VM pools, that is, between pairs of vToRs. In this case, the orchestrator will also have to expose to upper layers an abstraction of the switching capabilities (vToRs and vOS) and provide the right tools to operate these objects. All these functionalities will be provided thanks to the SDN controller, which will be responsible for providing the orchestrator with the right abstraction of the underlying reserved physical resources that match the requirements of the VDC request. Nevertheless, all these functionalities constitute an enhancement of the basic VDC service and will be treated as optional parameters in COSIGN, leaving the door opened to more long-term scenarios. Such enhanced VDC service, with advanced switching capabilities, is designed to cover the necessities of research and development customers (e.g. experimenters, academia, etc.), which require high control of the VDC resources in order to perform advanced experimentation over the requested virtual infrastructure. With the combination of both the basic and the enhanced VDC services, COSIGN aims at covering a large market segment, spanning from enterprises requiring advanced cloud infrastructure solutions, service operators to more niche customers such as R&D entities.

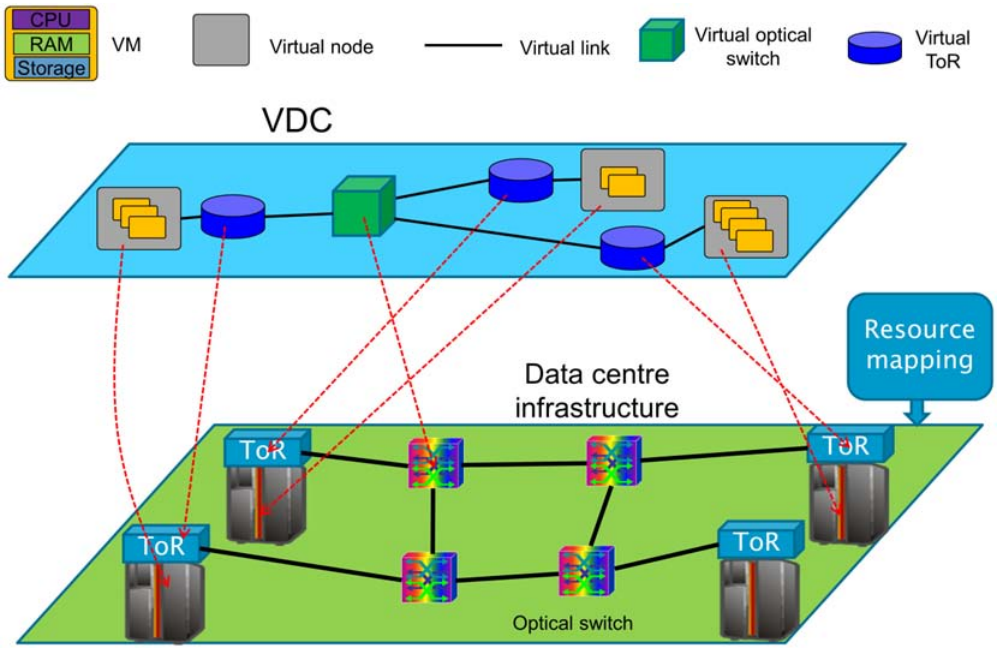

Figure 16 – Extended VDC service example

4.1.2 Use Case Specific Data Model

Table 5 presents the most significant objects in the VDC-induced data model and their most significant attributes. To realize VDC use case, COSIGN orchestrator's data model must be capable of representing these objects and operating on them.

Entity	Description	Most Significant Attributes
VDC service	A service whereby a **VDC** (namely, a faithful reproduction of a physical Data Centre) presented by a **VDC provider** to a **VDC owner**, billable with a pay-per-use or flat model.	VDC service description
VDC provider	Owner or subscriber of the physical Data Centre who publishes **VDC Service** to multiple independent users.	Resource inventory (devices, software, properties, status) for the physical infrastructure
VDC owner	**VDC service** client, consuming the **VDC service** from a single **VDC provider** or from multiple VDC Providers. **VDC owner** can own the VDC workload, choosing to run and consume his own business services on a VDC instance instead of acquiring and maintaining a physical DC. Depending on a business model, the **VDC owner** can by itself be either a DC service provider or a Cloud service provider, so that his clients own, operate, and consume the VDCs or services deployed on VDCs.	VDC owner ID VDC owner's account information VDC owner's credentials
VDC instance	A specific deployment of a VDC created by a **VDC provider** as requested by a **VDC owner,** in premises owned or hired by the **VDC provider.**	VDC owner ID VDC instance ID VDC resources inventory (components, properties, status, access information and credentials)
VDC client	End user of services deployed on top of the VDC. Can be the same entity as **VDC owner** or an external entity that consumes services remotely through entry points provided by **VDC owner** without knowledge of the infrastructure properties and without knowing how the infrastructure is virtualized.	VDC owner ID VDC client ID Client-specific accounting data might be collected by the VDC provider on behalf of VDC Client as a service to VDC owner that in turn can use this data to bill his customer by use.
VM	A unit of compute delivered to **VDC owner** by **VDC provider.**	VDC owner ID VDC instance ID VM ID VM required (min/max) resources – Memory, CPU, disk, storage, network connectivity VM performance metrics and meters VM status, resource usage stats, performance stats
VM pool	A group of VMs delivered to **VDC owner** by **VDC provider**, having some common properties.	VDC owner ID VDC instance ID VM pool ID Colocation/anti-collocation requirements Elasticity requirements Connectivity inside the pool requirements Connectivity with other VM Pools requirements External connectivity requirements VM pool status and resource usage stats – number of VMs, consumed bandwidth, etc.
vNet	A virtual network fabric delivered to **VDC owner** by **VDC provider**, allowing for the communication between **VM pools**. It is	VDC owner ID VDC instance ID vNet ID

	composed by a set of virtual nodes, i.e. **VM pools**, interconnected through virtual links (**vLinks**) that state communication requirements between virtual nodes.	Network topology and capacity requirements Security and traffic handling capabilities requirements
vLink	A virtual link element, interconnecting two virtual nodes inside **vNet**, delivered to **VDC owner** by **VDC provider**.	VDC owner ID VDC instance ID vNet ID vLink ID Bandwidth capacity requirements Latency requirements Source, destination ID
vFW	A network security system delivered to **VDC owner** by **VDC provider**, stating traffic acceptance or rejection inside **vNet** based on an applied rule set.	VDC owner ID VDC instance ID vNet ID vFW ID Network interfaces requirements Security rules requirements
vLB	A load balancing system delivered to **VDC owner** by **VDC provider**, stating traffic engineering and balancing rules inside **vNet**	VDC owner ID VDC instance ID vNet ID vLB ID Balancing rules requirements
vOS (optional)	A network node element that allows to perform circuit-based traffic switching operations inside **vNet**, delivered to **VDC owner** by **VDC provider**	VDC owner ID VDC instance ID vNet ID vOS ID Switching capacity requirements
vToR (optional)	A network node element that allows for traffic aggregation, mutiplexation and demultiplexation from/to a **VM pool** inside a **vNet**, delivered to **VDC owner** by **VDC provider**	VDC owner ID VDC instance ID vNet ID VM pool ID Electrical and/or optical port capacity requirements

Table 5 – Use Case Specific Data Model for Virtual Data Centre (VDC) Use Case

In the data model presented in *Table 5* actors are the VDC provider, the VDC owner, and the VDC client, but only the first two actors are of immediate importance to the orchestrator. It should be clear that other actors fulfilling roles in VDC provider organization or in VDC owner organization might exist, each with specific responsibilities and authorities. Modelling these different roles is out of the COSIGN scope and belongs to higher level Business Support and Operations Support Logics. Figure 17 represents high level actors of the VDC use case and their relationships.

4.1.3 User-driven Interactions and Requirements

In what follows we describe the main user interactions that have to be enabled between service consumer and service provider of the VDC service, namely the VDC owner and the VDC provider according to Figure 17. In COSIGN these interactions must be supported by the orchestrator, so that the VDC service can be either provided by the COSIGN UI or realized by the third parties by calling the COSIGN Orchestrator's API. For example, VDC portal may be an external web application, different from the OpenStack dashboard.

4.1.3.1 Publishing Service Portfolio

When requesting a VDC instance, a potential VDC owner should be aware of the detailed service portfolio offered by the VDC provider, in particular the types and the properties of available resources and their costs. To satisfy this need, the VDC Provider should publish a portfolio related to the configuration options for both compute and network resources, e.g. in a VDC service dashboard. In the case of VMs, user should be presented with a choice of the candidate operating systems as well as the potential hardware configurations (CPU, memory, storage, network capabilities). One option could be to display VM flavours, which are predefined templates of VM configurations, for which the VDC

owner could choose the preferred option. Additionally, the possibility of configuring its own VM template from a given set of parameters should be allowed. For the network side, the maximum available bit-rate per network link should be advertised. It is also necessary to allow the specification of additional network parameters, such as maximum tolerated latency, switching capacity (in the case of vToRs/vOS), etc. Finally, security and traffic handling options (firewalls and load balancers) should also be included and with the possibility to select the desired configuration.

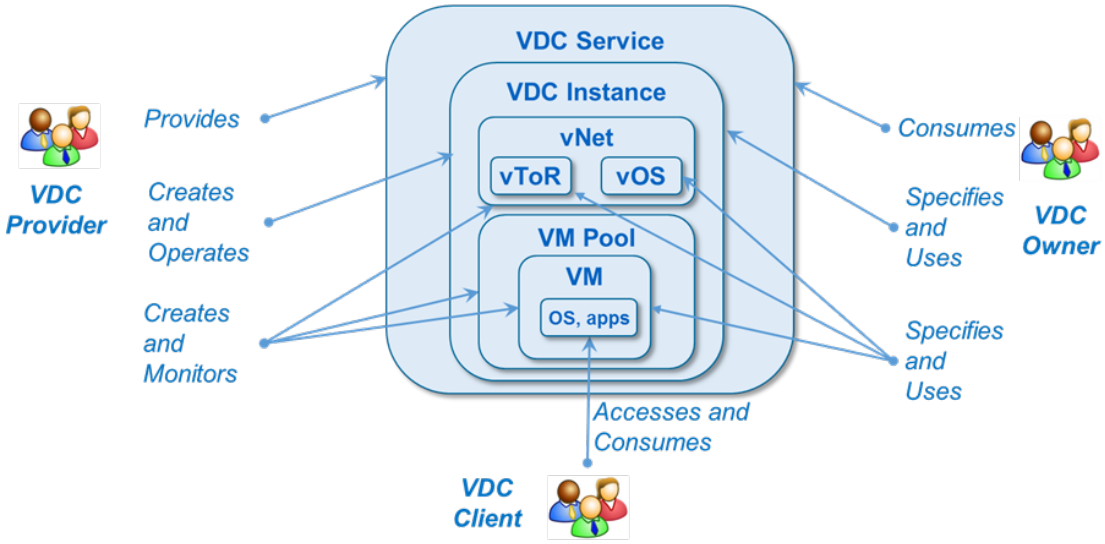

Figure 17 – VDC Use Case – Actors and Relationships

4.1.3.2 VDC Instance Operations

The VDC provider must allow the VDC owner to Create, Read, Update and Delete (CRUD) VDC instances as it sees fit, bounded by the VDC service description. Such feature is fundamental to ensure a dynamic and flexible service while guaranteeing the maximum Quality of Experience (QoE) to tenants. Such operations will be allowed through the COSIGN orchestrator.

4.1.3.3 VDC Resource Operation

Besides provisioning and operating VDC instances, one essential part of the VDC service is the possibility to access the resources of the VDC instance and operate them at will. For example, it should be possible to install any desired application on the configured VMs or allow for establishing any desired traffic flow through the virtual network. For this, proper tools to access and operate the VDC must be exposed to the VDC owner. For the access to the VMs, one possibility is to provide the public IP address. As for the network resources, custom APIs will be provided to the VDC owner.

4.1.4 Infrastructure-driven Interactions and Requirements towards SDN Controller

In what follows we describe the main interactions between the VDC service provider and the physical infrastructure required to implement the VDC use case according to Figure 17. It must be understood that in COSIGN these interactions must be provided by the infrastructure controllers layer, namely by the compute, the storage, and the network controllers. COSIGN orchestrator implementing the VDC Service is therefore a client to these controllers. In COSIGN scope, the main focus is on interactions with the network controller, represented by the COSIGN SDN controller layer (WP3).

4.1.4.1 Resources Utilization and Availability

In order to perform the joint compute and network provisioning for the VDC use case, the underlying physical resource utilization must be exposed towards COSIGN orchestrator. More specifically, the corresponding provisioning algorithms must have access to resource availability information regarding the utilization of the servers as well as the utilization of the intra-DCN. For this, one possibility could be to enable the orchestrator with the capability of fetching compute resources utilization from the

compute controller. As for the network, the orchestrator could fetch the desired information, both from the topological and the resource availability perspective, from the COSIGN SDN controller. In order to do so, the COSIGN SDN controller should provide the capability to query at will any kind of information regarding the DCN. For this, and depending on the necessary information, the COSIGN SDN controller will expose the totality of the DCN and its characteristics or an abstracted view of the network, summarizing the important traits that are relevant for the provisioning decision.

4.1.4.2 Continuous Monitoring and Updates

Both compute and network resources, as well as the already deployed VDC instances, should be constantly monitored to track changes on their utilization. Such changes have to be reflected in the orchestrator database, which will be queried periodically by the orchestrator algorithms. If changes are detected and if they impact on the current DC operator's internal policies, a re-planning operation will be triggered, finding the new mapping of the affected VDC instances. Then, with such a new mapping, the underlying physical resources will be updated.

4.1.4.3 Explicit Resource Allocation and Slicing

A key point of COSIGN orchestrator is the joint optimization of compute and network resources when allocating a new VDC instance or when applying re-planning operations to existing VDC instances. For this, orchestrator algorithms will define very specific resources mappings for the requests, stating both the mapping of the VMs as well as the virtual network elements (vLink, vToR, vOS, etc). Such mappings will be compliant with the current allocation policy of the DC operator and the requested resources for the VDC instance. Once the mapping has been defined, the underlying physical infrastructure must be reserved and configured. For this, both the compute and the network controllers must allow for the allocation of the explicit resources decided during the mapping phase.

4.1.4.4 VDC Network Isolation and QoS Guarantees

In order to operate independently, VDCs network resources must be isolated physically so as they do not interfere with each other. Nevertheless, VMs belonging to different VDCs may be allocated onto the same physical servers as long as they are logically isolated, i.e. their operation does not affect the other deployed VMs inside the server. Additionally, for the virtual network, in order to enforce strict QoS guarantees, the allocated resources for a VDC owner must be reserved for his sole use and operation. Hence, COSIGN SDN controller should ensure that resources reserved for a VDC owner cannot be accessed by others. Additionally, they should track which physical resources have been reserved for which VDC so when a new provisioning operation is required, only the appropriate resources are exposed towards the orchestrator to make the mapping decision.

4.2 Virtual Cloud Application Use Case

This section details the Virtual Application (vApp) use case, expanding the description already found in earlier deliverable of COSIGN WP4 [D4.1]. In addition, the section provides the orchestrator's requirement both from/towards the SDN controller and from/towards the User layer, as well as a full description of the operational model and workflows for vApp provisioning is provided, along with detailed data models and methods.

4.2.1 Use Case Description from the Orchestrator Perspective

Virtual Application Cloud (vApp[3]) is an infrastructural building block towards Platform as a Service (PaaS) solution, concerned with providing advanced infrastructure independent platforms to multiple application developers, operators, and clients, over a single shared physical infrastructure. While each virtual cloud application consumes physical infrastructure resources, a set of resources provided to such application does not attempt to replicate the capabilities of a physical DC or to provide configuration-level access to the physical infrastructure. Instead, in this use case cloud provider allows

[3] Previous deliverables have referred to this use case by different names, e.g. multi-tenant application cloud. Currently used name, vApp, was chosen to better differentiate this use case from the VDC use case.

its users to develop, deploy, and operate complex dynamic applications without considering or even being aware of the intrinsic details of the underlying DC technologies. In addition, smooth migration of applications deployed elsewhere to the vApp cloud must be supported with as little changes to the application and its components as possible. Figure 18 presents an example of a typical use request that is usually formulated in natural language using application level terminology. The request includes the type and the number of application components to be deployed, the connectivity pattern required to interconnect the components, and a set of business driven or compliance driven rules that govern the components and the connectivity. The request does not include technology specific details related to the underlying platform, e.g. what hypervisor to use, whether to segment the network with VLANs or with routing domains, whether to deploy a Firewall (FW) appliance or to employ a set of IP tables rules, etc.

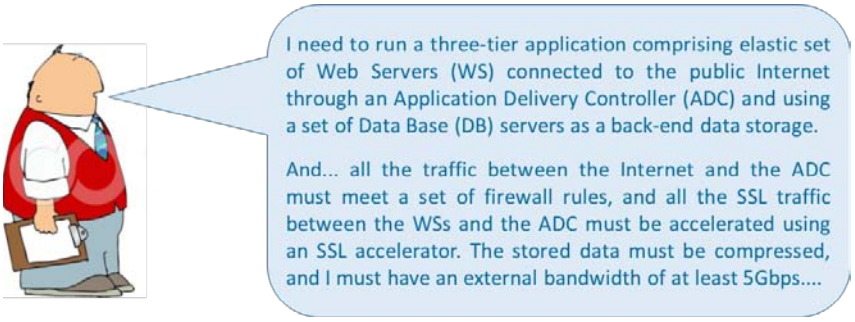

Figure 18 – Virtual Cloud Application – User Request Example [36]

From a point of view of a physical infrastructure provider, vApp provisioning involves joint orchestration of compute and network resources so as to guarantee the fulfilment of the request in a context of concrete technologies and capabilities existing in the DC. A key point of the whole provisioning process relates to a highly dynamic nature of modern cloud application. Depending on the application load from the client side, a time of a day or a month of a year, or on external or internal failures, applications can dynamically grow or shrink and application components can come up, go down, or move to new locations. It is not possible for the application owner to be involved in all these reconfigurations so this becomes the responsibility of the cloud operator and part of the service he provides. In addition to supporting dynamicity, cloud provider has to run multiple instances of different virtual applications belonging to different users and having different demands and usage patterns, see Figure 19. Sufficient level of isolation between the different applications is mandatory – functional isolation, performance isolation, security isolation, and, last but very important – management isolation, i.e. providing self-service access to all the operational properties of the application and its components.

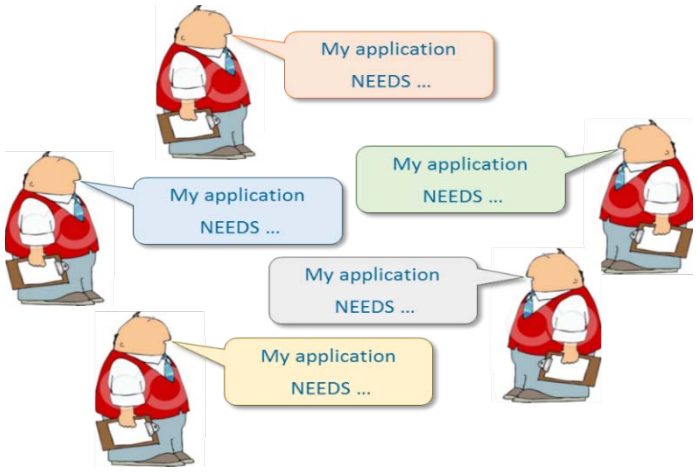

Figure 19 – Virtual Cloud Application – Multiple User Requests

The vApp service identified in COSIGN is inspired by already existing commercial services such as IBM Cloud Orchestrator [37] or CISCO Application Centric Infrastructure [38]. Use case consists of

large number of virtual application instances deployed on top of the same physical infrastructure so that each instance is independently provisioned and operated by its owner. Application instances are described by their users in high level terms, specifying the intended usage, connectivity patterns, and properties, while the provider is responsible for realizing the intent through commanding the infrastructure. Following the IBM Distributed Overlay Virtual Network (DOVE) concepts, networking intent is described as connectivity between endpoints along with the policies associated to the connectivity. Network intent is then modeled by grouping network endpoints that share policy criteria and assigning policy rules and actions to pairs of policy groups, the source policy group and the target policy group.

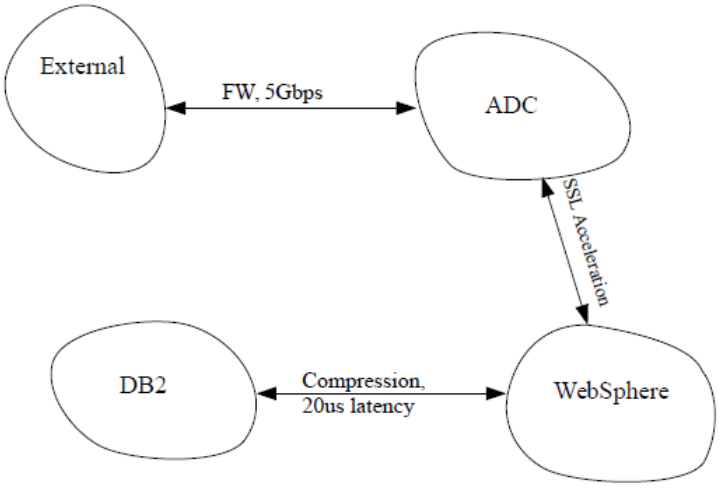

Figure 20 – Virtual Cloud Application – Modeling Network Intent

Figure 20 shows the intent based model DOVE yields for the application described by *Figure 18* as application blueprint or pattern. Application patterns are logical entities and do not have any physical resource associated with them. Application patterns are extensible and composable and can be either provided as part of the cloud service or created by the application owner from basic building blocks – groups, policies, policy rules and actions of different types.

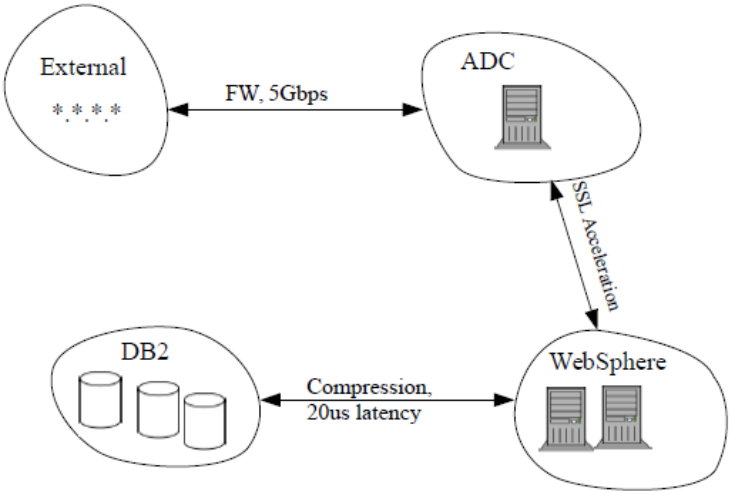

Figure 21 – Virtual Cloud Application – Instantiated Application Blueprint

When an application pattern is instantiated to become a deployed application instance, some resources might be allocated or even reserved for its sole use, depending on the request encoded in the blueprint. During application lifetime resource allocation can be adjusted based on the infrastructure status and application level meters, to maintain the functionality and the service levels prescribed by the blueprint (see Figure 21).

4.2.2 Use Case Specific Data Model

Entity	Description	Most Significant Attributes
vApp Service	A service whereby the ability to deploy and operate virtual cloud application by specifying its high level properties, i.e. connectivity patterns, type and number of compute components, lifecycle events, is presented by a **vAPP Cloud Provider** to a **vApp Owner**, billable by use.	vApp Service description
vApp Cloud Provider	Owner or subscriber of the physical Data Centre who publishes **vApp Service** to multiple independent users.	Virtual Resource Inventory (supported resource types, supported event types, etc.)
vApp Owner	**vApp Service** client, consuming the **vApp Service** from a single **vApp Cloud Provider** or from multiple Providers. **vApp Owner** can be application developer needing devOps and CI services. **vApp Owner** can be also application operator needing support for monitoring and periodic adjustment of the production environment.	vApp Owner ID vApp Owner's account information vApp Owner's credentials
vApp Pattern	A blueprint describing the virtual cloud application, specifying its compute needs, lifecycle events, and connectivity requirements. **vApp Patterns** can either be predefined by the **vApp Cloud Provider** or created by the **vApp Owner.** In addition, **vApp Patterns** should be composable.	vApp Pattern ID vApp blueprint description with types and numbers of vApp Components, Policy Groups, Policies between the Policy Groups
vApp Instance	A specific deployment of a virtual cloud application created by a **VDC Provider** as specified by a **vApp Owner** by providing a **vApp Pattern,** in premises owned or hired by the **VDC Provider.**	vApp Owner ID vApp Instance ID Virtual Resources Inventory (allocated resources, requested evens, requested stats, etc) Physical Resources Inventory and their mapping to the Virtual Resources
vApp Client	End user of services provided by virtual cloud application. Can be the same entity as **vApp Owner** or external entity that consumes services remotely through entry points provided by **vApp Owner,** through **vApp Cloud Provider. vApp Owner** is not exposed knowledge of the infrastructure properties and without knowing how infrastructure is virtualized.	vApp Client ID Client-specific accounting data might be collected by the vApp Cloud Provider on behalf of vApp Client as a service to vApp Owner that in turn can use this data to bill his customer by use.
vApp Component	Virtual Application building block delivering value during the application's lifetime. Can be compute node (VM, container, bare-metal server, etc.) or storage node.	vApp Instance ID vApp Component ID
vApp Policy Group	A logical entity grouping **vApp Components** for purposes of assigning common properties, e.g. connectivity policy.	vApp Policy Group ID list of vApp Component IDs currently populating the group
vApp Policy	A logical description of intended behaviour or property. For networking, means connectivity properties like bandwidth, latency, path redundancy, what in-network services are applied and in what order, etc.	vApp Policy ID vApp Policy Target ID (vApp Policy Group ID) vApp Policy Source ID (vApp Policy Group ID) vApp Policy rules – conditions and actions to be applied to the communication between the source and the target policy groups

Table 6 – Use Case Specific Data Model for Virtual Application (vApp) Use Case

Table 6 presents the most significant objects in the vApp-induced data model and their most significant attributes. To realize the use case, COSIGN Orchestrator's data model must be capable of representing all these objects and operating on them.

In data model presented in *Table 6* actors are the vApp Cloud Provider, the vApp Owner, and the vApp Client, but only the first two actors are of immediate importance to the orchestrator. It should be clear that other actors fulfilling roles in vApp Cloud Provider organization or in vApp Owner organization might exist, each with specific responsibilities and authorities. Modelling these different roles is out of the COSIGN scope and belongs to higher level Business Support and Operations Support Logics. In addition, Policies can account to more than connectivity and describe failover and high availability patterns, allocation and anti-collocation rules, and more. These are also out of COSIGN scope if they do not involve network-related events and/or actions. Figure 22 represents high level actors of the vApp Use Case and their relationships.

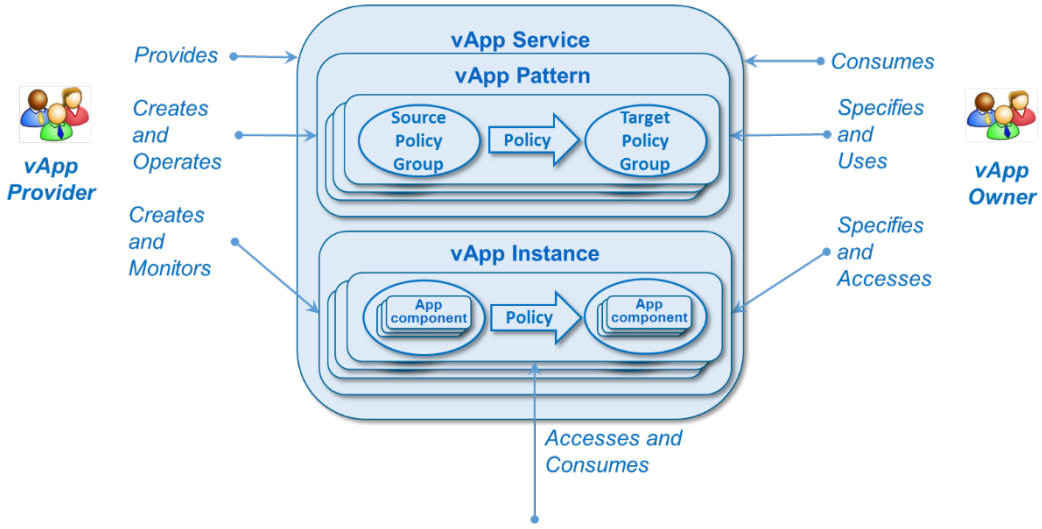

Figure 22 – Virtual Cloud Application – Actors and Relationships

4.2.3 User-Driven Interactions and Requirements

In what follows we describe the main User interactions that have to be enabled between the main User of the vApp Service, namely the vApp Owner, and the vApp Cloud Provider according to Figure 22. In COSIGN these interactions must be supported by the orchestrator, so that the vApp service can be realized by calling the COSIGN Orchestrator's API. Such realizations can be either implemented as part of the COSIGN UI or realized by the third parties.

4.2.3.1 Isolation and Self-Service Requirements

One of the most important user-driven features required from the virtual application cloud is isolation from the other users in terms of the resource usage, performance, failure, and security. From the networking perspective it means that for security reasons traffic, data, or operational patterns of one user must not be seen by others. In addition, failures induced by others should not disturb user's operations and performance should not suffer from some users utilizing the capacity unproportionally. Isolation from the physical network is of the outmost importance as well because, in this use case, the user is not expected to be networking expert and even if he is, exposing low level networking details and let them leak into the virtual application abstraction can yield the application not-portable and very hard to manage, especially under frequent changes.

Self-service is the next feature most often demanded by the application owner moving to a cloud environment. Users want to retail freedom to decide upon all the aspects of application design, lifecycle, and resource utilization. From the networking perspective, this means ability to create application specific non-trivial communication patterns, specifying their needs in terms of bandwidth, latency, in-network services (e.g. FW, LB), etc. In addition, users want to be able to control the

addressing of their application components so, for example, applications deployed elsewhere can be migrated to the cloud without IP renumbering.

4.2.3.2 vApp Patterns Operations

Virtual applications can be created to run on a cloud (cloud-born applications) or adopted to run on a cloud (legacy applications). For both application types, there are several patterns known in advance that can be beneficially reused when modelled as logical composable objects. Examples are: cluster, multi-tier web application, Map-reduce engine, etc.

The user should be presented with a library of the existing patterns that can be immediately instantiated into application blueprints or extended into a more evolved scenario specific patterns modelling the application at hand. A library of building blocks and operators for creating user-specific patterns from scratch must be available as well. Upon creating patterns can be named, renamed, saved, updated, etc. Patterns can belong to a user context or to be available to all the users or all the users of the specific type or filling a specific role.

4.2.3.3 vApp Instance Operations

When an application pattern is instantiated, an application instance is created comprising all the virtual components of the application, their status, statistics, lifetime event watches, etc. Application lifecycle can be customized or generic, depending on the pattern and on what can be supported by the cloud. In the most generic case, a running application can be stopped, paused, updated, undeployed, and, eventually, taken out of service. New application components can be created and existing components can be removed if the application pattern permits this. In general, we do not consider the possibility to migrate a running application to a new application pattern or to a new version of its application pattern. This can be valuable feature but is far beyond the COSIGN project goals.

4.2.3.4 vApp Components Operations

Application components are compute or storage nodes where application data is stored and the application-defined computation occurs. Networking entities do not normally belong to applications but are provided as a service to them by the cloud provider. This includes, network forwarding functionality required for connectivity which can be subdivided into L2 (broadcasting and switching) and L3 (routing) if it is required by the application component's networking stack. For example, users might insist to have their network modelled around L2 and L3 notions, reflected in the OS networking stack configuration – routing table, defaults gateway, etc. Network services are also provided by the cloud provider as a service, either in a form of appliances (shared or dedicated, physical or virtual, centralized or distributed), or implemented under the hood of the generic connectivity services (implicit routing, edge-based filtering and rate limiting, etc.).

Network-specific operations on app components are, therefore, connecting the component's network interface to a connectivity service as specified by the application pattern. In the policy-based abstraction we plan to follow, the operation is as simple as to making a component to belong to a suitable policy group. Upon creation, the component is connected to its group, or, to be more exact, is created in a policy group, so resource allocation to a component is done as prescribed by the policy group. Upon destruction, the component is removed from the group so all the associated resources are freed up and available to the application or to other, depending on type of resource and its sharing properties. We do not consider the possibility to move an existing active component from one policy group to another.

Additional operations on application components are not directly related to networking and, therefore, to COSIGN. These operations are part of the component lifecycle and are defined and governed by other resource controllers (e.g. the storage and the compute). Software consideration, such as installed operating system application stack, credentials, software configuration and updates, and more, are out of the COSIGN scope.

4.2.4 Infrastructure-Driven Interactions and Requirements towards SDN Controller

In what follows we describe the main interactions between the vApp Service Provider and the physical infrastructure required to implement the vApp Use Case according to Figure 22. It must be understood that in COSIGN these interactions must be provided by the infrastructure controllers layer, namely by the compute, the storage, and the network controllers. COSIGN Orchestrator implementing the vApp Service is therefore a client to these controllers. In COSIGN scope, the main focus is on interactions with the network controller, represented by the COSIGN SDN Controller Layer (WP3).

4.2.4.1 Resources Allocation

By its nature of being infrastructure independent, vApp use case should not involve direct manipulation of the underlying infrastructural entities like, for example port or link reservation or dynamic circuit creation. It is, however, desirable to enable path differentiation and specialization to control the service level or quality of application experience. We choose to implement most of this functionality by providing infrastructure independent hints, inserted by the virtualization edges (vSwitches) into data packets, to designate the desired infrastructure-level choices, like on which of available paths to send the flow, or what flows must get preferential treatment as compared to others.

4.2.4.2 Resources Availability and Utilization

In order to provide the vApp service, the orchestrator must have up to date information about the availability of the DC resources. This type of information exists at the infrastructure controllers' layer. The resources availability information is needed in order to correctly reserve resources for the vApp service provisioning. Depending on the accuracy of this information, the orchestrator can avoid cases where the physical infrastructure becomes overbooked or underutilized. For this, the orchestrator polls the SDN controller, and requests for the status of the underlying resources, including the topology, the capacity of the links, link utilization, etc.

4.2.4.3 Continuous Monitoring and Updates

Network infrastructure information available to the orchestrator, such as the topology, capacity, and utilization must be continuously updated through the controller. This will ensure that the status of the resources is always consistent between the SDN controller and the orchestrator. For example, the orchestrator must receive updates regarding the DC infrastructure events so that it can react to changes, e.g. to failure of physical entities, congestion on links, etc. Upon being informed of these changes, orchestrator may re-apply optimization algorithms to obtain better resource utilization, avoid the congestion, and improve applications' experience. It is necessary that the orchestrator gets specific monitoring information regarding the provisioned connectivity for a vApp instance, to enable the vApp owner to monitor the provisioned service.

The continuous monitoring of the DCN infrastructure can be done through continuous polling of the SDN controller, or through notifications, depending on the type of resource and the protocols used at the orchestrator-controller (SDN northbound) and the controller-DCN (southbound) interfaces.

4.3 Data Centre Operations and Management Use Case

4.3.1 Use Case Description from the Orchestrator Perspective

In addition to the other identified COSIGN use cases and services, the COSIGN DC infrastructure provider requires the necessary tools to administrate the infrastructure, monitor and optimize its utilization among other actions, as well as ensuring a correct ICT infrastructure operation and management. The Data Centre Operations and Management (O&M) Use Case (UC) is complementary to the other two use cases defined in COSIGN (i.e. VDC and vApp). While the other COSIGN UCs define the services and operations that are to be provided to the customers, the DC O&M UC defines additional operations, targeted towards the data centre operator. The architecture and framework defined by COSIGN would support the two previously defined UCs as well as the DC O&M, compiling a strong solution for DC management for future proofing and enhancement of DC and its networks.

D1.1 [D1.1] deliverable stated the convenience of having a separated use case exclusively for ICT operations purposes in order to integrate management functionalities such as the network service provisioning and automation within the COSIGN orchestrator. The high level data model for the DC O&M UC and two potential operational workflows (i.e. path provisioning with QoS and provisioning of VMs to already existing virtual tenant networks) were specified in D4.1 [D4.1].

4.3.2 Use Case Specific Data Models

The Data Centre Operations and Management Use Case includes two separated data models, each with its typical usage and its ultimate goal:

1. A data model representing the services deployed on the DC infrastructure, along with the resources allocated to these services, information on their users, etc. Scenarios around this data model are related to the needs of hosted cloud tenants (e.g. vApp owners and VDC owners) and aim to provide support for, for example, the cloud customer billing services. Network related entities of this data model is described in Section 4.3.2.1.
2. A data model representing the entire pool of DC resources, along with their health, utilization, interdependencies, power consumption, need for SW upgrades, etc. Scenarios around this data model are related to the needs of the DC/Cloud provider itself and aim to provide support for, for example, DC manageability, sustainability, profitability, etc. Network related entities of this data model is described in Section 4.3.2.2.

Although the former data model could be considered a special case of the latter, the ultimate goal, the target scenarios, and the beneficiaries of the two are different enough (see above) to be divided into two separated data models, to provide a deeper description level of each.

4.3.2.1 Data Model Supporting Deployed Services and Resources Allocated to them

Table 7 includes a description of the most relevant network related entities for the operation and management of the data model that captures the resources allocated to specific services and the most significant attributes associated to these entities.

Entity	Description	Most Significant Attributes
DC O&M service	A service which enables to efficiently operate and manage the complex ICT infrastructure of a datacentre.	DC O&M service description
DC cloud provider	Owner of the physical Data Centre who publishes **DC O&M service** to the **DCN admin** (for data model 2) or **DCN service owner** (for data model 1)	Virtual Resource Inventory – supported resource types, supported event types, etc.
DCN admin	Administrator of the physical Data Centre which operates and manages the entire pool of DCN resources.	DCN admin credentials
DC service owner	Can be the VDC service (**VDC owner**), the vApp service (**vApp Owner)** or any other provided service (use case) owner.	VDC/vApp owner ID VDC/vApp owner's account information VDC/vApp owner's credentials
VDC service	A service whereby a VDC (namely, a faithful reproduction of a physical Data Centre) presented by a VDC provider to a VDC owner, billable by use. The detailed data model for this resource is described in section 4.2.2.	VDC service description
vApp service	A service whereby the ability to deploy and operate virtual cloud application by specifying its high level properties. The detailed data model for this resource is described in section 4.3.2.	vApp service description

Other services	Any other DCN service that can be operated and managed as previous use cases.	Service description.
Service instance	A specific deployment of a service owned or hired by the **Service owner**	Service instance ID Service instance resources inventory (components, properties, status, access, information and credentials).

Table 7 - Specific Data Model for Operation and Management of Deployed Cloud Services and their Allocated Resources

The data model presented in the previous table involved the following main actors: the *DCN cloud provider* (IaaS provider), the *DCN administrator* (DCN admin in Table 7) and the COSIGN *DCN-based service owners* (DCN service owner in Table 7). Figure 23 represents the high level actors of the DC O&M Use Case and their relationships.

While a DCN service owner will create and operate a particular instance of a service, the DCN admin will manage the overall services and service instances provisioned to the customers. Moreover, the DCN admin will manage the operations that each DCN service owner can execute on the service instances that belong to him.

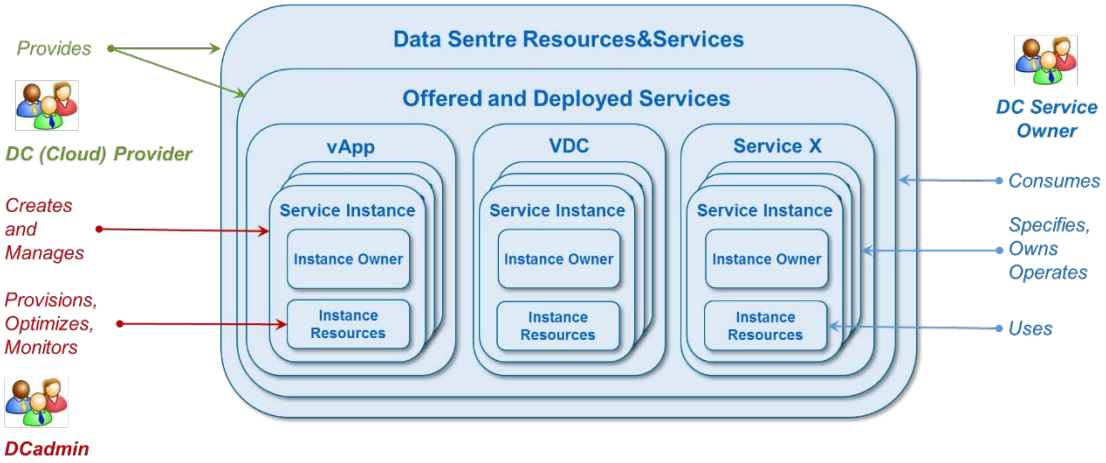

Figure 23 – DC Operation and Management of Deployed Cloud Services – Actors and Relationships

4.3.2.2 Data Model Representing the DC Resources and Services

The second particular situation that this use case presents is the full operation and management of the entire pool of DCN resources, DCN-based provided services and the clients.

Entity	Description	Most Significant Attributes
DC O&M service	A service which enables to efficiently operate the complex ICT infrastructure.	DC O&M service description
DC cloud provider	Owner of the physical Data Centre who publishes **DC O&M service** to the **DCN admin** (for data model 2) or **DCN service owner** (for data model 1)	Virtual Resource Inventory (supported resource types, supported event types, etc.)
DCN admin	Administrator of the physical Data Centre which operates and manages the entire pool of DCN Resources, services and DCN service clients.	DCN admin credentials
DCN service owner	DCN service client, consuming the VDC service (**VDC owner**), the vApp service (**vApp Owner)** or any other provided service (use case).	VDC/vApp owner ID VDC/vApp owner's account information VDC/vApp owner's credentials
Service	A specific deployment of a service owned or hired by the **Service owner**	Service instance ID Service instance resources inventory

instance		(components, properties, status, access, information and credentials).
NetPool	Complete set of network resources including vNets, **Network Resources**, and **vLink**.	Service owner ID Service instance ID NetPool ID Network Connectivity inside the pool requirements External connectivity requirements NetPool status and virtual resource usage stats – number of virtual network resources, links, ports, Bandwidth, etc.
vNet	A virtual network fabric delivered to **service owner** by **DCN Cloud Provider**. It is composed by a set of **network devices**, i.e. **switches**, interconnected through virtual links (**vLinks**) that state communication requirements between **network resources**.	Service owner ID Service instance ID vNet ID Network topology , virtual resources and capacity requirements
vLink	A virtual link element, interconnecting two **virtual devices** inside **vNet**, delivered to **VDC owner** by **VDC provider**.	VDC owner ID VDC instance ID vNet ID vLink ID Bandwidth capacity requirements Latency requirements Source, destination ID
Network resource	Network devices are the abstracted representation of physical network resources and virtual network resources assigned for different services, e.g. **vSwitch, vToR, vPorts**, etc.	Network resource Type Network resource ID Network resource specific attributes.

Table 8 - Specific Data Model for the operation and management of the entire pool of DCN resources, services and tenants.

In the data model presented in the previous table the main actors involved are the *DCN cloud provider* (IaaS provider), the *DCN administrator* and the COSIGN *DCN service owners*. Figure 24 depicts the high level actors of the DC O&M Use Case and their relationships.

In this case, the DCN admin executes management operations on the components of a service instance, where each service instance is materialized into a set of network resources and its associated attributes (e.g. connectivity, bandwidth).

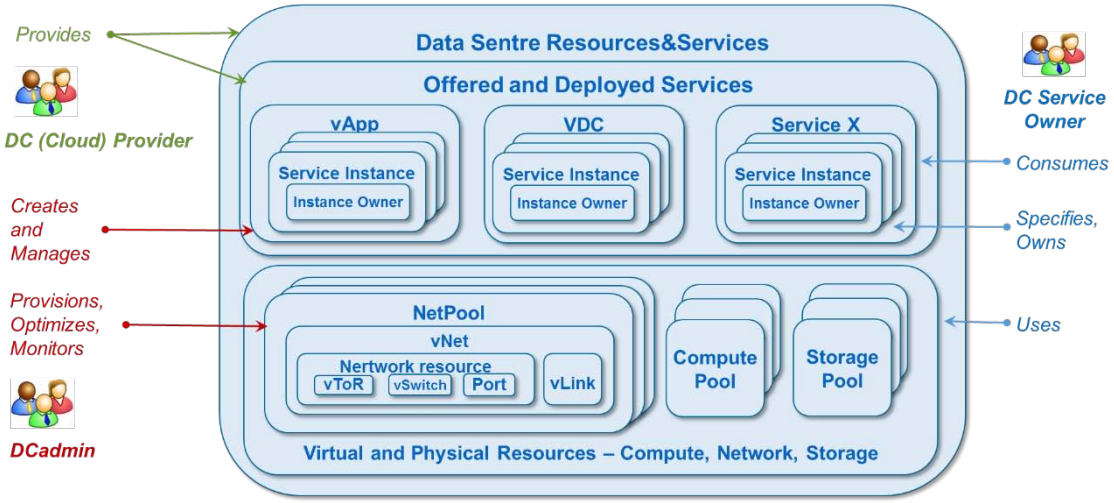

Figure 24 - DCN Operation and management of the entire pool of DCN resources - Actors and Relationships.

4.3.3 User-Driven Interactions and Requirements

This section describes the specific interactions and requirements that the main tenants of the DC O&M use case (i.e. DCN admin and Service owner) need from the orchestrator. The interactions that should be enabled through the DC O&M use case are at least the following ones:

4.3.3.1 Multi-Tenant Access

The current use case considers two types of tenants (Figure 24):

1. DCN administrator who may have access to the entire stack of resources
2. DCN service instance owner – a client that only can operate and manage the virtual resources allocated to the requested services.

Thus, it must be enabled separated access for the different types of tenants, enabling the management options which correspond to the tenants' consumed services. The COSIGN orchestrator must include in its upper REST API several types of access permission.

4.3.3.2 Device Inventory, Monitoring and Configuration

COSIGN DCN administrator may be enabled with options to visualize the available devices at the data plane, monitor their status and trigger configurations from the orchestration layer which exert they effect at the underlying SDN controller and physical data plane layers. The configuration options at the orchestrator layer are simplified and hide the optical HW layer complexity. It is a task of the SDN control layer to ensure a proper matching between the configuration triggered from the orchestrator and the reflection such configuration entails on the data plane devices.

4.3.3.3 Separation between the DCN Admin and the Service Owner

Owners of DC Service Instances should be able to monitor and to configure the DC resources associated to the services they own and operate. For the vApp service, only virtual resources are visible to the service instance owner and lend themselves for his monitoring and operational control. For the VDC Service, physical resources may be reserved to the exclusive use of the service instance and can be thus monitored and operated by the service instance owner. In all cases, however, only the DCN admin may have full access rights to all the physical and the virtual resources.

In the case of COSIGN, the focus is on DCN resources – the physical, e.g. ports, links, devices, etc. and virtual, e.g. virtual switches, overlay virtual networks, security groups, etc. Both the DCN admin and the service instance owners will be granted with permissions to monitor utilization and availability for the resources in their authorization domain. Whereas in the case of the COSIGN customer, he/she will be able to check the utilization and availability of the virtual resources assigned to the consumed services, the administrator will have the same rights but covering all customers' resources present in the system.

4.3.3.4 Support for Maintenance Tasks

The possibility to Create, Read, Update and Delete (CRUD) while *provisioning connectivity across DCN resources* based on the use cases is fundamental to ensure a proper Quality of Experience (QoE) to users while making use of their services. Since it is more a resource-oriented feature, it directly applies to consumers of COSIGN services and it does not apply so much to the COSIGN DCN admin.

Moreover, the administrator should be able to perform the following tasks to support various maintenance operations:

- Provision connectivity inside the DCN;
- Migrate VMs;
- Trigger back-up and replication procedures.

4.3.3.5 Trigger and Control the Resource Optimization

The administrator is able to trigger the execution of various algorithms which have as a goal the optimization of resource utilization in the DC. These algorithms can be a part of the other two use cases, or they can be specific algorithms that apply only for the DC O&M use case. The algorithms are applied to a set of resources and are given an objective function. The target set of resources can be created based on various rules such as:

- The resources belonging to a VDC or virtual network
- The resources of the same type: optical resources

Some of the objectives considered for the optimization process are:

- Increase the perceived QoS (QoE) for the customer
- Reduce energy consumption
- Reduce resource consumption

The features generally described above lead to specific requirements towards the constituent layers in the COSIGN architecture. Figure 25 depicts the interactions between the logical entity which enables the DC O&M use case and the other COSIGN components. The DC O&M logical function offers a graphical user interface (GUI) through which the DC operator (i.e. admin) can access the functionality enabled through this use case. The GUI can be implemented as an extension to the OS Horizon dashboard, as a standalone interface or a suitable combination of the two.

Two types of interfaces are identified:

1. Interfaces that enable the administrator to access the DC O&M capabilities (1 and 2 in the figure).
2. Interfaces that enable the DC O&M to fulfil the requests made by the administrator (3-7 in the figure, further explained in the next section).

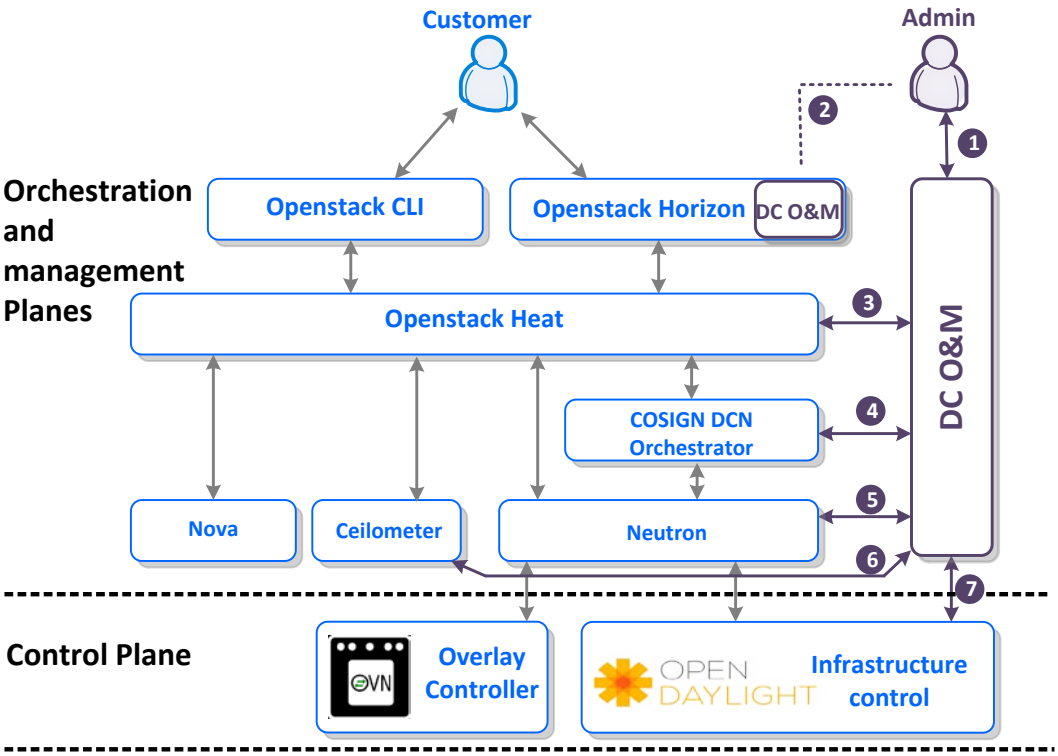

Figure 25 – Generic interactions between the DC management and other COSIGN components.

4.3.4 Infrastructure-Driven Interactions and Requirements towards SDN Controller

It is also required to identify and describe the main interactions between the Operation and Management user case service provider and the physical infrastructure required to implement the use case. The main focus is on the interactions between the DC O&M component and the network controller, represented by the COSIGN SDN controller layer (WP3). In order to fulfil the operations defined for this use case, it is necessary to gather information and enforce configuration from and towards the lower layers of the COSIGN architecture. To this end, the DC O&M block interacts with several other components, illustrated in Figure 25 through interfaces 3-7. The DC O&M requires access to physical and logical resources at various abstraction levels:

4.3.4.1 SDN Infrastructure Controller Level

The optical DCN fabric should be exposed through interface 7. This means that the DC O&M entity has access to monitoring information regarding the optical devices. Moreover, it should be possible to enforce configuration on the devices and set up connectivity in the optical DCN. The operations supported on interface 7 can also be implemented over interface 5. However, this requires extending Neutron in order to support configuration of optical devices. This implementation detail will be decided during a later phase.

4.3.4.2 Overlay Controller Level

The DC O&M must have access to the virtual switches inside the servers that implement the overlay virtual networks. This enables monitoring and controlling the overlays. This requirement can be realized through interface 5, by interacting with Neutron which implements the extensions for OVN.

4.3.4.3 VM Specific Monitoring Information

It is also required to retrieve VM information in order to properly operate the networking aspects of the DCN services considering the VMs' load and behaviour.

4.3.4.4 DCN Orchestrator

Interface 4 allows the DC O&M to have control over the DCN optimization algorithms that are implemented within the DCN Orchestrator. Through this interface the execution of such an algorithm on a set of resources can be triggered.

4.3.4.5 Neutron Level Network Abstractions

DC O&M must have access to higher level network abstractions that exist and are exposed by Neutron (also interface 5). Such abstractions can be Virtual Networks, Subnets, and Ports of the base Neutron API, as well as the abstractions of the extended Neutron API, e.g. Security Groups, Floating IPs, etc.

4.3.4.6 Enable Overall Access to the Cloud Management Platform

Since a service (i.e. VDC or vApp) is defined as a stack inside Heat, in order for the administrator to be able to easily manage such services, it is necessary to enable access through interface 3. Through this interface the DC O&M has access to the services provisioned to the customers.

4.4 Use Cases Summary and the Converged Data Model

The three use cases described above bear sufficient level of similarity to converge on a common data model. The use cases operate at different abstraction levels: vApp provides the most abstract view of the infrastructure that is viewed as an abstracted platform where an application runs enjoying the infrastructure capabilities without being exposed to its minute details; VDC, in addition to operating at an abstraction level similar to vApp, provides access to specific low level aspects of the underlying capabilities through a virtualization layer but does not allow direct access to physical device configuration. O&M use case allows direct access to physical device configuration for controlling and configuring the infrastructure by its owner. An O&M user, in addition to low level controls, can enjoy VDC operations, e.g. for creating experimentation sandbox for testing new technologies and

capabilities before they are rolled out in the production DC. An O&M user can also enjoy vApp operations by running virtualized application, e.g. for automation of routing tasks or for optimizing resource usage from the operator's perspective. Some of these tasks are network-specific, i.e. specialized for administering, managing, and optimizing the DCN domain, while some can involve the DC-wide orchestration. The application can be developed and operated by the DC operator itself, or acquired form the third parties. So, in some sense, the O&M use case can be seen as the most generic of all, encompassing the others as special cases. We have decided to treat them separately to demonstrate the business value of the COSIGN orchestrator, realized through different realistic use cases, stakeholders, and scenarios.

Each use case, being targeted at a different audience, with different level of familiarities of the underlying technologies, and with different levels of access and control rights over the infrastructure, has its own data model including objects of business value to the user. These use case specific models are outlined in Sections 4.1, 4.2, and 4.3 above. To be implemented as part of a unified architecture, the use-case specific data models must converge to a unified data model of the chosen architecture. As was detailed in Section 3.1, OpenStack is going to serve as reference architecture and a unified implementation platform for use cases. Therefore, use-case specific data models must converge to the models of the OpenStack and its services whenever possible and extensions must be implemented where there entities of value to one or more use cases cannot be satisfactorily modelled in OpenStack today.

In this section we outline the relevant details of the OpenStack data model showing how use-case specific entities can be mapped. It must be understood that, as the project develops, different mapping can be used according to changes in the project needs or in new developments in the very fast-going OpenStack community.

4.4.1 OpenStack Data Models

4.4.1.1 Keystone

Keystone fulfils several roles in OpenStack and is implemented as several services. Some roles are not of particular relevance to the COSIGN orchestrator, e.g. management of OpenStack services, service endpoints, tokens, etc. We describe the part of Keystone Data Model relevant to managing identities of users and resources and relationships between them.

In the area of the user management, Keystone has evolved from a simple model comprised of *Tenants* and *Users*, to a more evolved model briefly described below. The current model underlies the Keystone API V3.0 **[**39] and comprises the following data types:

- **User**: represents a human; has name, email, and account credentials; associated with one or more Projects by being granted Roles on a Project.
- **Project**: represents a unit of OpenStack resource ownership and service API invocation.
- **Domain**: represents the/an authority domain and can enclose multiple Projects; Domain's administrator is a User granted administrative role on a Domain that can manage other Users and Projects inside the authority domain; each Domain defines a namespace, scoping objects and attributes, so that entities global in a domain must have unique names.
- **Role**: represents a role in Role Based Access Control (RBAC) metadata; each Role can be associated with many User-Project and/or Group-Project pairs; Role captures the operations that a user can perform in a given tenant. Role's meaning can prescribed through Rules and Extras while Role object itself is only a name. Openstack Services have roles that have assigned specialized meaning already.
- **Rule**: represents a part of RBAC metadata, describing a set of requirements for performing an action.
- **Extras**: represents customizable bucket of key-value entries for RBAC metadata, for extending the Role description.
- **Token**: represents credentials associated with a User or a User-Project pair.

- **Group**: represents a collection of Users, convenience type that plays role similar to a User; when Role is granted to a Group on a Project, all Users in a Group are granted that Role on that Project.

Keystone V3.0 Data Model allows many-to-many relationships between Users and Groups to Projects and Domains, while backend implementation adds data model hardening suitable for their needs. Figure 26 shows Data Model types relevant to COSIGN Orchestrator and their relationships. Default credentials store (local MySQL database) will be used to simplify the installation. Also, COSIGN will not require defining the User and the Group entities – either of them alone is sufficient for COSIGN demonstrator's needs. Choosing the more convenient data type to use will be done at a later stage of the project.

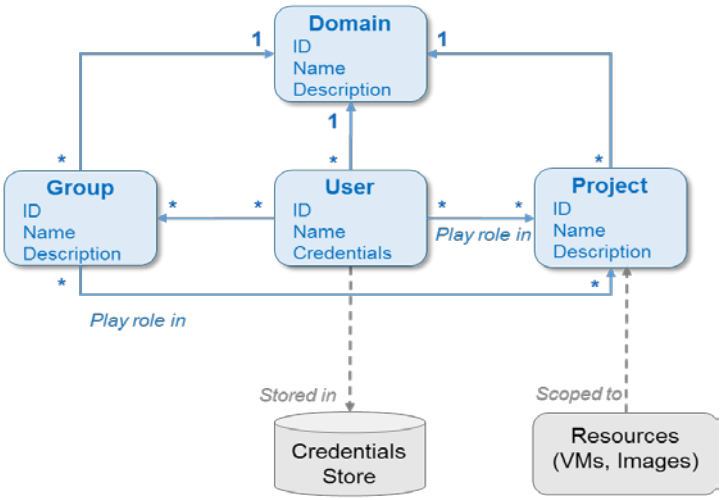

Figure 26 – OpenStack Keystone – Data Model Entities relevant to the COSIGN Orchestrator

4.4.1.2 Nova

Nova is the OpenStack service that provides representation and low level API control for the compute resources [40] with a mission statement to implement services and associated libraries to provide massively scalable, on demand, self-service access to compute resources [41]. At first, Nova was dealing with a single type of compute resource – the Virtual Machine (VM), focused purely on providing access to VMs running on a variety of different hypervisors. Nova API was developed to manage VMs and had drivers to support different hypervisors and different ways of network connectivity to VMs. Today additional types of compute resources are being added to Nova's consideration – containers and bare-metal servers, aiming to have a common API supported by a framework of drivers and plugins to support diversity aiming to allow different types of compute resources to be supported by the same Nova service. Handling differences and providing type specific support is planned to be provided by upper-level projects that will build upon the simple low level infrastructure API of Nova – the Magnum project for containers and the Ironic project for bare-metal servers.
This separation of duties allows the COSIGN orchestrator to work directly over the Nova API to support VM orchestration in concert with the DCN orchestration. Extending COSIGN orchestrator to other types of compute resources is out of the project scope.
For VM management, Nova data model comprises the following two major data types [42]:

- **Server** – A virtual machine (VM) instance
- **Flavor** – An available hardware configuration for a Server. Each flavor has a unique combination of disk space, memory capacity and priority for CPU time.

To provide useful computation, a VM must be booted with a suitable operating system and application stack customized to the intended usage. While earlier Nova releases took care of software configuration on the VMs, this is delegated to other services today and for all practical purposes, Nova

can use pre-configured images to activate VMs. An image is a data type from the Glance service data model, closely related to Nova and used in Nova API. Image creation and customization is out of the scope of COSIGN project.
A Nova Server object (that will probably be renamed into a more generic Instance) has a rich set of lifecycle events, all part of Nova service APIs:

- Create – builds a new VM instance with a given name, Flavor, Image, and meta-data; when created, the VM has been assigned to a specific Nova compute host and assigned with specific network addresses for its interfaces.
- Reboot – restarts the operating system or takes a VM through a complete power cycle.
- Rebuild – replaces the Image used to create the VM.
- Resize – converts to a different flavor, scaling the VM up or down.
- Stop – shuts off the VM.
- Start – starts the VM that was stopped.

Administrative users can have access to more lifecycle events:

- Pause/Unpause – stores the VM's state in host's RAM; the VM stays active and consumes resources without progressing in its states.
- Suspend/Resume – stores the VM's state and memory to disk; the VM is stopped and does not consume resources (other than disk space and management attention).
- Back-up – copies the VM's state or data to a persistent storage.
- Migrate/LiveMigrate/Evacuate – moves the VM to another Nova compute host.
- Update – includes operations like changing password, injecting new networking state, resetting the networking for a VM, resetting other aspects of VM's state, etc.

To manage physical compute resources, Nova has the following data types:

- **Host** – represents a physical compute host, its total resources (e. g. CPU, memory, and disk), free resources, resources allocated to projects. Nova API allows disabling/enabling Hosts, power cycling them, and pinging them for availability.
- **Hypervisor** – represents hypervisor specific statistical and monitoring data on the Host.

Additional objects are data model extensions used for different purposes not of direct relevance to COSIGN: 1) Host Aggregates and Availability Zones are used to influence the scheduler's placement decisions; 2) Quotas are used to cap resource usage of Projects, Domains, and Users; network related objects used for a legacy nova-networking operations, like Network, Floating IP and related entities (Floating IP Pool, Floating IP DNS Record, Floating IP Bulk), Security Group, Project Network. One relevant data model extension that can be used in COSIGN is Server Group, representing a named collection of VMs with a common policy attribute. *Figure 27* shows Data Model types relevant to the COSIGN Orchestrator and their relationships; Server Group object is not shown here for simplicity.

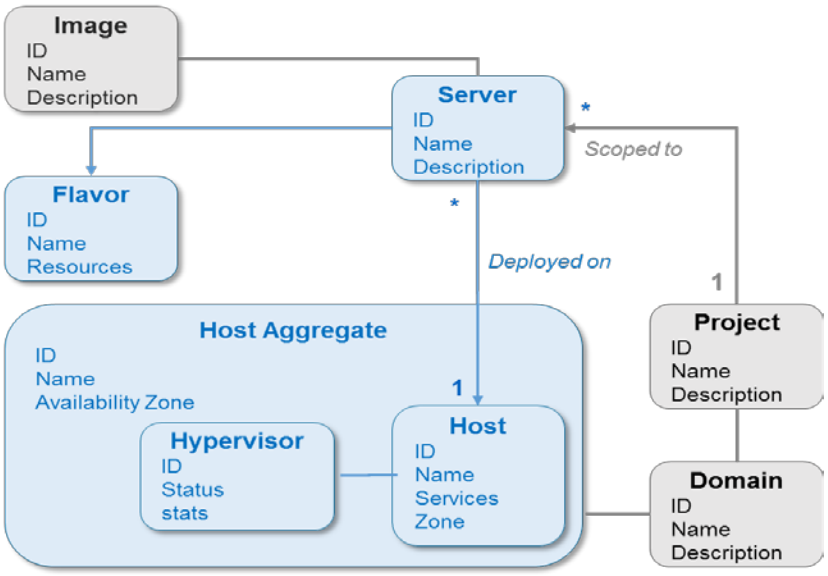

Figure 27 – OpenStack Nova – Data Model Entities relevant to the COSIGN Orchestrator

4.4.1.3 Neutron

Neutron is an OpenStack project providing network connectivity as a service between interface devices, e.g. virtual network interfaces, managed by other OpenStack services, e.g., Nova. The Neutron core data model used by the current v2.0 API is restricted to a very small amount of objects required for providing L2 connectivity:

- **Network –** represents L2 broadcast domain
- **Subnet –** represents an IP subnet and can be associated with a Network
- **Port –** represents a point of attachment of a network client, e.g. a VM's virtual interface, to a Network, using up an IP address from one of the Subnets associated with the Network

Extensions exist for additional networking aspects, such as external connectivity, L3 connectivity or routing, network services such as load balancer, firewall, VPN, security groups, etc. Extended data model types of special potential relevance to COSIGN are:

Networks provider extended attributes extension – enables administrative users to specify how Network objects map to the underlying networking infrastructure, by adding the following attributes to a Network data type:

- `provider:network_type` – specifies the type of a physical network mapped to the OpenStack Network at hand; types can be, for example, Flat, VLAN, VxLAN, or GRE.
- `provider:physical_network` – identifies the physical network on top of which the OpenStack Network at hand is being implemented; note that OpenStack does not expose any facility for retrieving the list of available physical networks, therefor COSIGN will have to implement such an API if it will be decided to use Provider extension.
- `provider:segmentation_id` - identifies an isolated segment on the physical network; the nature of the segment depends on the segmentation model defined by `provider:network_type`; for example, if `provider:network_type` is VLAN, then `provider:segmentation_id` will hold a VLAN identifier

The actual semantics of these attributes depend on the technology back end of the particular plugin and these attribute values are completely hidden from regular tenants.

The **Networks multiple provider** extension enables administrative users to define multiple physical bindings for an OpenStack Network and to list or show details for Networks with multiple physical bindings. Multiple physical bindings for a network are defined by including a segments list in the request body of a Network creation request. Each element in the segments list has the same structure as the provider network attributes. These attributes are `provider:network_type`, `provider:physical_network`, and `provider:segmentation_id`.

Note that it is not possible to use both the Provider and the Multiple Providers extensions at the same time and that the ML2 Neutron plugin supports both extensions.

4.4.1.4 Ceilometer

The Ceilometer project [17] aims to deliver a unique point of contact for billing systems to acquire all of the measurements they need to establish customer billing, across all current OpenStack core components and with a goal to support future OpenStack components.

Ceilometer enables Users to store samples, events, and alarm definitions in supported database back ends. Current OpenStack Telemetry API v2 [43] comprises the following data types:

- **Alarm** – represents actions to be performed when specified conditions are met; Alarms are scoped to Project or to User-Project pair and can be create/deleted and updated; their details and history data can be read.

- **Meter** – represents a measurement point such as counter keeping track of a specified value, typically collected from a specific resource; Meters can be created by their suppliers so that Telemetry service starts harvesting their data.
- **Sample** – represents data harvested from a Meter; read only.
- **Resource** – represents data collection point with Telemetry metadata; read only.
- **Capability** – represents the supported API capabilities, as well as the representation and the storage options for the Telemetry data; consists of a set of parameters, mostly Boolean, and their values – true or false.

4.4.1.5 Heat

Heat is the OpenStack service that provides a template based orchestration [9]. Heat allows Users to specify a cloud application using the Heat Orchestration Template (HOT) language [12], [13], [14] and deploys the application according to the provided template by executing appropriate OpenStack API calls. Currently, templates allow creation of most OpenStack resource types, such as Nova Server instances, Volumes, Security Groups, Users, etc.

Current OpenStack Orchestration API v1 [44] comprises the following data types:

- **Template** – represents a description of cloud application complete with the Resources that have to be deployed to run the application, parameters needed to figure out dependencies and invoke the relevant APIs.
- **Stack** – represents an instance of a cloud application deployed as prescribed by the associated Template; Stacks can be created and deleted, updated, suspended and resumed, abandoned and adopted; Stacks provide API access to their resources, resource lifecycle events, and the associated template; Stacks' snapshots can be requested and manipulated as stand-alone entities.
- **Resource** – represents an OpenStack entity that can be manipulated as part of the orchestration workflows; Resource has properties and attributes, as well as lifecycle event handlers that allow Heat engine to figure out when to act upon the Resource and what OpenStack API to invoke; supported lifecycle events are *create, delete, update, suspend, resume*

The COSIGN orchestrator will reuse the existing Heat resources for the relevant OpenStack entities and will develop new resource plugins for entities created as part of COSIGN development. Fully supported Heat Resources of relevance to COSIGN are listed below:

- **Heat**
  - `OS::Heat::Stack`
  - `OS::Heat::AutoScalingGroup`
  - `OS::Heat::ScalingPolicy`
  - `OS::Heat::InstanceGroup`
  - `OS::Heat::ResourceGroup`
  - `OS::Heat::WaitCondition`
  - `OS::Heat::WaitConditionHandle`
  - `OS::Heat::UpdateWaitConditionHandle`
- **Keystone**
  - `OS::Keystone::Group`
  - `OS::Keystone::Project`
  - `OS::Keystone::Role`
  - `OS::Keystone::User`
- **Nova**
  - `OS::Nova::Flavor`
  - `OS::Nova::Server`
  - `OS::Nova::ServerGroup`
- **Neutron**
  - `OS::Neutron::MeteringLabel`

  - OS::Neutron::MeteringRule
  - OS::Neutron::Net
  - OS::Neutron::Port
  - OS::Neutron::ProviderNet
  - OS::Neutron::Subnet
- **Ceilometer**
  - OS::Ceilometer::Alarm
  - OS::Ceilometer::CombinationAlarm
  - OS::Ceilometer::GnocchiAggregationByMetricsAlarm
  - OS::Ceilometer::GnocchiAggregationByResourcesAlarm
  - OS::Ceilometer::GnocchiResourcesAlarm

In addition to the above there are several blueprints in different stages of maturity in the OpenStack community, e.g. Orchestration for Group Based Policy Resources Blueprint [45]. The COSIGN team will monitor developments in the orchestration community and decide whether to adopt the community resource plugins or to develop new plugins for the already existing resources that are not currently supported by the Orchestrator.

4.4.1.6 Group Based Policy

GBP OpenStack project [27] introduces a policy model to describe the relationships between different logical groups or tiers of an application. The primitives have been chosen in a way that separates their semantics from the underlying infrastructure capabilities. Resources may be public or local to a specific tenant. The key primitives are:

- **Policy Target** – represents an individual network endpoint, e.g. a NIC; Policy Target is a basic addressable unit in the architecture.
- **Policy Group** – represents a set of Policy Targets, an infrastructure agnostic grouping construct without specifying any network semantics; each Policy Group models its dependencies by declaring Rule Sets it provides to other Policy Groups as well as Rule Sets it consumes when communicating to entities in other Policy Groups.
- **Policy Classifier** – represents a means of filtering network traffic by matching packet header fields such as protocol and port range, as well as the flow direction, e.g. into a Policy Group, out of a Policy Group, or both.
- **Policy Action** – represents an action to take when a particular rule is applied; currently supported types include “allow”, “deny”, and “redirect” although additional action types will be offered in the future.
- **Policy Rule** – represents a pair of a Policy Classifier and a Policy Action.
- **Policy Rule Set** – represents a set of Policy Rules that may be nested through parent child relationships.

Additional feature of the OpenStack GBP service is that it enables intent-based network service insertion and chaining not easily attainable otherwise. For this, basic data model above is extended with the following entities:

- **Service Chain Node** – represents a logical device capable of providing network services of a particular type, e.g. load balancer appliance, firewall appliance, etc.
- **Service Chain Spec** – represents an ordered grouping of Service Chain Nodes; Service Chain Spec is one possible value of Policy Action attribute for a “redirect” Policy Action of a basic GBP model.
- **Service Chain Instance** – represents a specific instantiation of a Service Chain Spec as Policy Action between Policy Groups; Service Chain Instances are created automatically when a service chain is activated as part of a Policy Rule Set.

Figure 28 represents the currently supported OpenStack GBP data model graphically. The COSIGN team will weigh the pros and the cons of adopting the BGP data model as opposed to implementing new DCN orchestration component built specifically to support and expose COSIGN capabilities – optical forwarding in the dataplane and a programmable OF-based control above it.

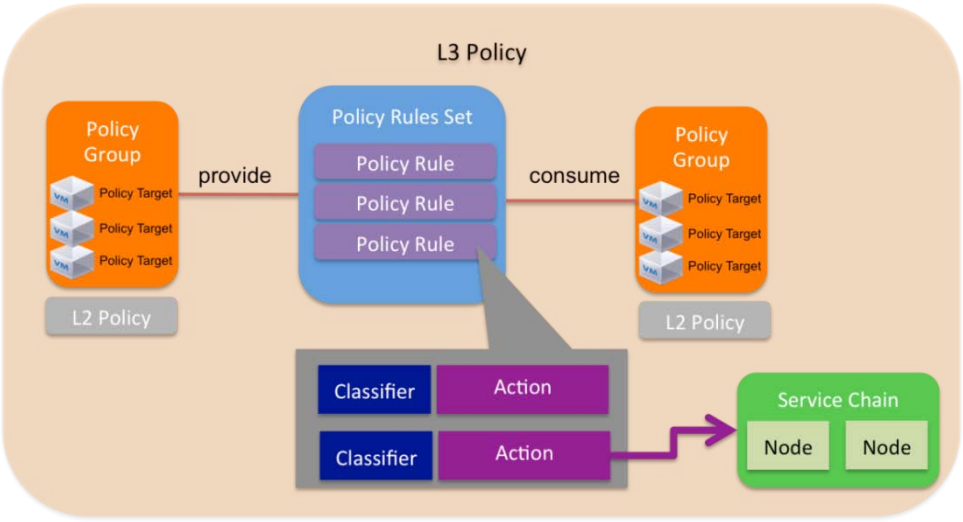

Figure 28 – OpenStack GBP – Data Model [27].

4.4.2 Mapping the VDC Data Model to OpenStack

In this section, the VDC data model that has been outlined in [D4.1] and refined in Section 4.1.2 of this document is mapped to data types existing in the OpenStack data models. *Table 9* summarizes the mapping, giving for each data type in the VDC data model its candidate representative in OpenStack. Note that for some data types, several candidates exist, while for some other types no mapping can be found in the current OpenStack model.

VDC Data Model Entity	OpenStack Mapping	Comments
VDC Service	-	Not an entity type, represents a new type of Service. Implementing new service is out of the COSIGN scope.
VDC Provider	-	Not an entity type; represents the owner of the DC.
VDC Owner	Keystone Project	Can also be Keystone Domain
VDC Instance	Heat Stack	Can also be Keystone Project (if VDC owner mapped as Keystone Domain)
VDC Client	-	No OpenStack object exists, requires a new COSIGN specific extension if per-client resource allocation and/or accounting is implemented
VM	Nova Server	Represents compute Instance
VM Pool	-	No exact mapping, can be mapped as elasticity group, a policy group, or as a new COSIGN specific extension
vNet	Neutron Network	
vOS	-	No OpenStack object exists. It requires a new COSIGN specific extension. This is an optional module.
vToR	-	No OpenStack object exists. It requires a new COSIGN specific extension. This is an optional module.
vLink	-	No OpenStack object exists, requires a new COSIGN specific extension
vFW	Neutron Services Extention – FWaaS	
vLB	Neutron Services Extention – LBaaS	

Table 9 – Mapping VDC Data Model to OpenStack

For concrete implementation, the choice will have to be made whether to use one or another of the candidate mappings and whether to create COSIGN specific extensions. The COSIGN team will make this choice after experimenting with the possible options and according to the evolving status of OpenStack.

Figure 29 depicts the data model for the VDC request structure at the orchestrator layer. The UML representation shows the main entities as well the relations among them. A *VDC* instance can be deployed onto several *VDC Zones*, which represent a physical geographical area location (i.e. a DC) where the cloud resources associated to the VDC must be allocated. The *VDC Customer* (i.e. the tenant) is the consumer of the VDC service and the main actor who interacts with the VDC service dashboard in order to:

- Request a new VDC instance, selecting its main parameters (e.g. VM templates, VDC zone, network configuration, etc.).
- Start, stop and access the resources of the VDC instance.
- Monitor the status and the alarms or logs of the VDC instance and its virtual resources.
- Request the modification of existing resources in the VDC instance or the instantiation of additional resources within the same VDC instance.
- Remove the VDC instance.

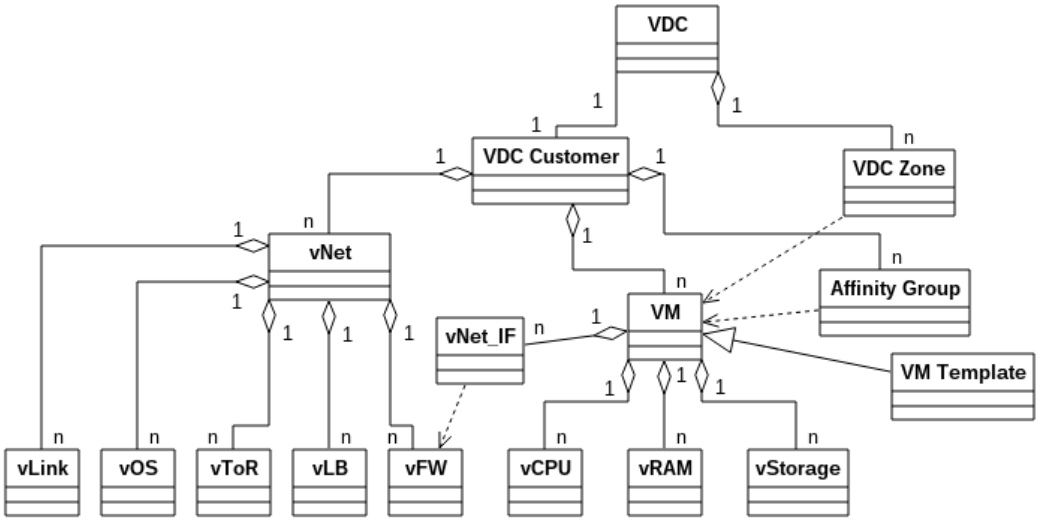

Figure 29 – Data model of the VDC service

A VDC Customer can be represented through the *tenant* concept used in OpenStack, while a VDC instance can be somehow related to a *stack* resource in the Heat terminology. In general, a VDC service must be able to support multiple tenants concurrently and, for each of them, it should be able to support multiple VDC instances of different size and complexity. All the VDC instances deployed through the VDC service must guarantee their full isolation from the others.

The VDC instance is mainly constituted from a set of *VMs* and virtual networks (*vNet*), which are utilized to implement the VDC Customer IaaS service. The allocation of VMs is regulated through sets of *Affinity Groups*, which state a set of rules on how to select and allocate virtual resources on top of physical servers. Such rules are understood as collocation/anti-collocation constraints for the orchestrator provisioning operations.

Each VM is composed by a number of virtual CPU elements (*vCPU*), virtual RAM (*vRAM*), virtual local storage (*vStorage*) as well as a number of virtual network interfaces (*vNet_IF*), which are utilized to enable the communication between virtual machines within the VDC instance. The resources of the VM are allocated according to a specific *VM Template*, which corresponds to a specific hardware configuration according to predefined application/performance profiles.

As for the VDC network side, *vNet* represents the virtual network interconnecting the pools of VMs. Essentially, a vNet comprises a set of virtual links (*vLink*), which are mapped to physical paths and wavelengths in the underlying physical infrastructure in order to match the specified QoS requirements, namely, bandwidth and latency parameters. Moreover, the vNet also comprises a set of virtual firewalls (*vFW*) and virtual load balancers (*vLB*), which are utilized to implement the security and traffic handling policies of the particular VDC instance. Optionally, in the aforementioned enhanced VDC service, the vNet also comprises a set of virtual ToRs (*vToR*) and virtual optical switches (*vOS*). Such elements will be utilized to perform advanced traffic handling (e.g. connection switching) for the traffic flows in the VDC and are defined as optional.

All the main VDC type of resources (VMs and vNet) are managed in COSIGN through different components of the OpenStack cloud platform and their joint validation, coordination and composition in a global cloud service is handled through the/an orchestrator.

4.4.3 Mapping the vApp Data Model to OpenStack

In this section, the vApp data model that has been outlined in [D4.1] and refined in Section 4.2.2 of this document, is mapped to data types existing in the OpenStack data models. *Table 10* summarizes the mapping, giving for each data type in vApp data model its candidate representative in the OpenStack. Note that for some data types, several candidates exist, while for some other types no mapping can be found in the current OpenStack model.

VDC Data Model Entity	OpenStack Mapping or Reference	Comments
vApp Service	-	Not an entity type, represents a new type of Service. Implementing new service is out of the COSIGN scope.
vApp Cloud Provider	-	Not an entity type; represents the owner of the DC.
vApp Owner	Keystone Project	Can also be Keystone Domain
vApp Pattern	-	No OpenStack object exists; if GBP will be used, map to GBP blueprint, otherwise COSIGN specific extension is needed
vApp Instance	Heat Stack	Can also be Keystone Project (if VDC owner mapped as Keystone Domain)
vApp Client	-	No OpenStack object exists, requires a new COSIGN specific extension if per-client resource allocation and/or accounting is implemented
vApp Component	Nova Server	Represents compute Instance
vApp Policy Group	-	No OpenStack object exists; if GBP will be used, map to GBP Policy Group, otherwise COSIGN specific extension is needed
vApp Policy	-	No OpenStack object exists; if GBP will be used, map to GBP Policy, otherwise COSIGN specific extension is needed

Table 10 – Mapping vApp Data Model to OpenStack

For concrete implementation, the choice will have to be made whether to use one or another of the candidate mappings and whether to create COSIGN specific extensions. The COSIGN team will make this choice after experimenting with the possible options and according to the evolving status of OpenStack.

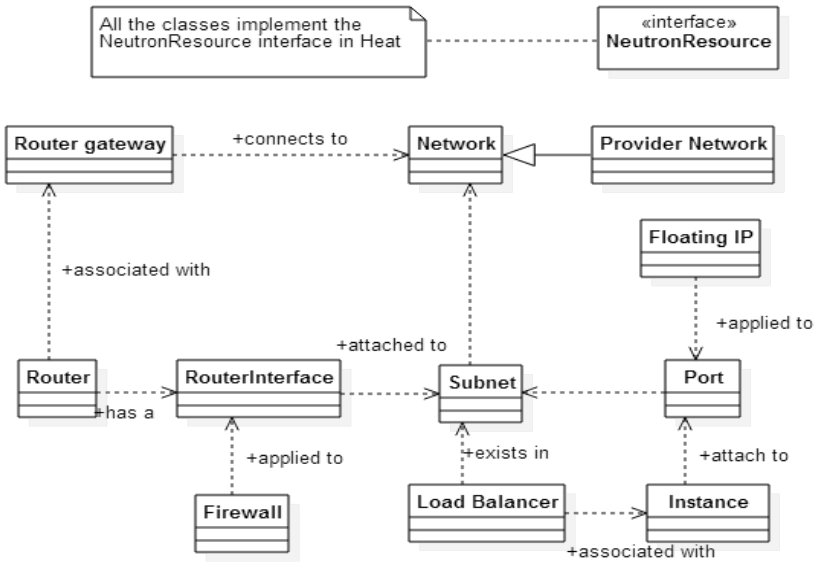

Figure 30 – Data model for virtual networks at the orchestrator

Figure 30 shows the data model for virtual networks at the orchestrator layer. The UML representation depicts the main logical entities that constitute a virtual network and the relations among them.

At the orchestrator level, the differentiation between a virtual network that is based on overlay and a virtual network that maps to a physical substrate lies in a configuration option. This configuration option is delivered by the *Provider Network* extensions, on top of the regular Network features. On a regular basis, OpenStack creates the networks as overlay networks, using various tunnelling technologies (e.g. GRE, VxLAN, or Geneve). However, by specifying the *provider* network attribute when creating a new virtual network, the user can request that a new virtual network is mapped to a specified segment of the physical network so that this network's ports are connected directly the specified segment of the physical network.

4.4.4 Mapping the O&M Data Model to OpenStack

Since the DC O&M entity (Figure 25) interacts with other components at different levels in the architecture, it will therefore need to be aware of multiple data models, which are contained by the components it interacts with. More specifically, the DC O&M functions are active at the same time with any of the VDC of Cloud Virtual Application (App) use cases. Hence, the DC O&M interacts with data models specific to these use cases. For instance, it displays monitoring information regarding the abstractions specific to each use case, a VDC of a vApp, and it enables the administrator to also manage these abstractions. In conclusion, the DC O&M function only takes advantages of existing data models defined by other components and unifies them to enable a consistent set of management operations for the data centre operator.

In this section, all the data models relevant to the O&M use case, i.e. data models that are accessed through the interfaces numbered from 3 to 7 in Figure 25, are presented from the point of view of the DC O&M entity. The focus is on operational aspects of the data models, showing how the required operations are enabled. The actual data models can, however, be more complex in each of the COSIGN components that maintain them (e.g. OS Heat, Network Orchestrator, Neutron, ODL).

At the OS Heat component the following data model (Figure 31) is available for the DC O&M entity, through interface 3 (Figure 25). The data model is a generic representation, which takes the form of a *stack*, of the two COSIGN services: VDC and vApp. The administrator is able through the DC O&M to visualize (inventory, monitoring) and perform operations for the services provisioned to the customers. At the highest level, the services are represented as a *stack.*

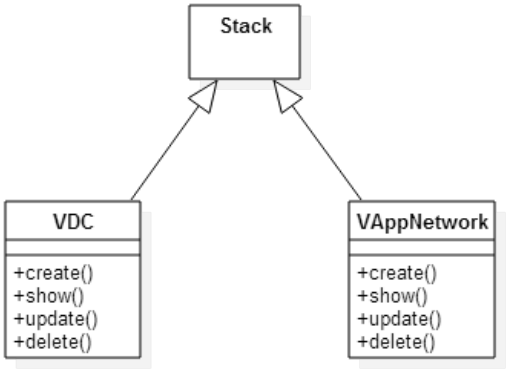

Figure 31 Generic data model at Openstack Heat

The data model depicted in Figure 32 is available to the DC O&M through interface number 4 (Figure 25). The DCN orchestrator contains a set of algorithms, aiming at optimizing service provisioning and the DCN resources consumption. The algorithms are generally executed when a new service is requested. However, the administrator is able, through the DC O&M, to execute some algorithms independently of the service provisioning. Therefore, the management plane is able to configure the parameters of any of the algorithms and execute them with a set of arguments.

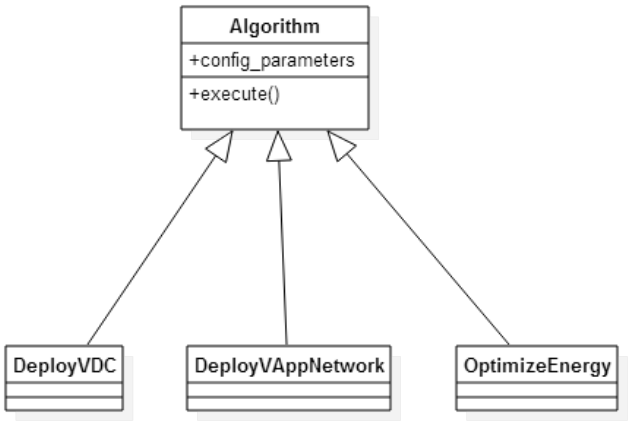

Figure 32 Data model at the Network orchestration entity

The Neutron component comprises various network abstractions since it interacts with multiple network providers (OVN, OpenDayLight). Figure 33 presents Neutron abstractions such as network overlays (through the OVN integration), higher level abstractions (virtual networks, virtual subnets). The DC O&M allows the admin to visualize (inventory, monitor) these abstractions (e.g. overlay).

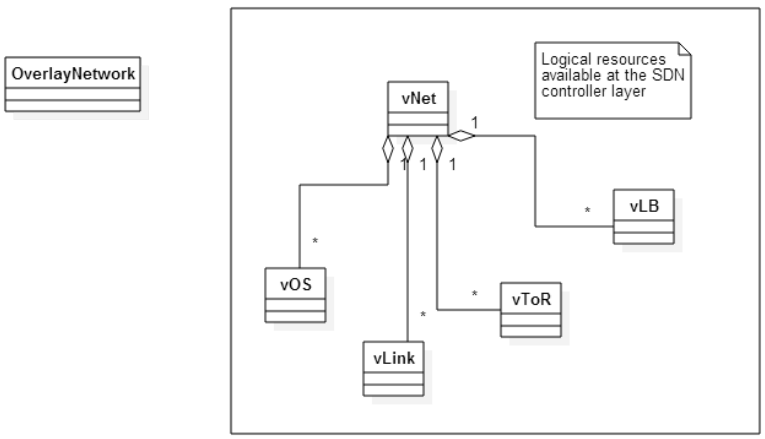

Figure 33 Data model at Neutron

In Figure 34 the data model available through interface 6 is presented. This data model is maintained by Ceilometer components and it is related to network monitoring. Through the DC O&M, the admin can visualize and configure alarms in the DC.

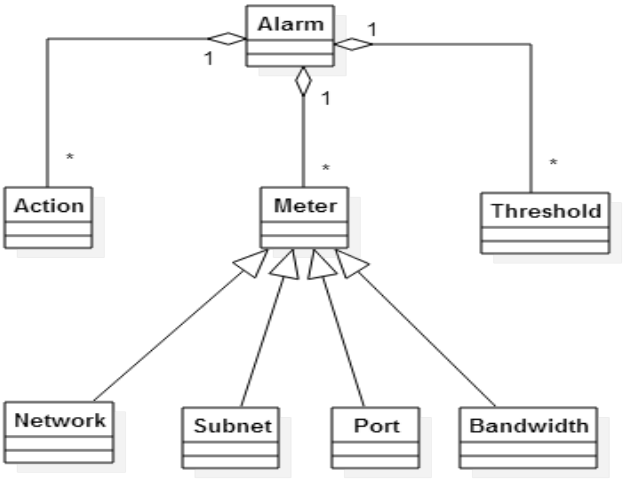

Figure 34 Data model at Ceilometer

The DC O&M interacts with ODL through interface 7 (Figure 25) in order to get detailed information about the DCN. This information is related to the physical topology and is shown in Figure 35.

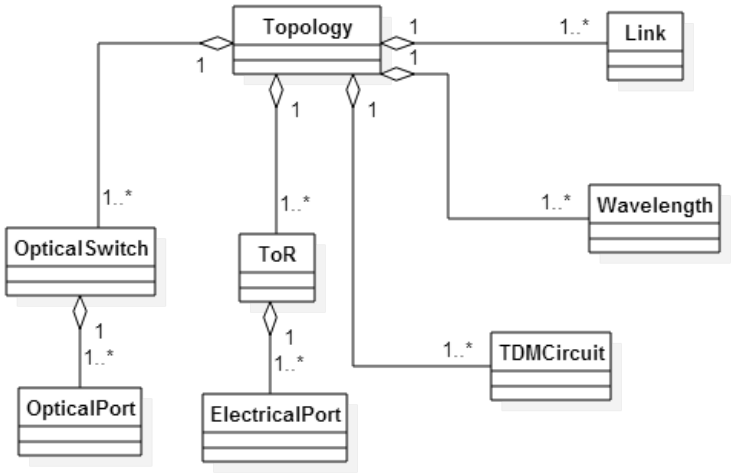

Figure 35 Data model at the SDN Controller (OpenDaylight)

As a general remark, in order to support all the complex management operations the DC O&M entity must gather the necessary information from various sources, present it to the administrator in a meaningful and consistent way, take administrator operations and enforce them at the correct component/level in the architecture. This requires a certain degree of orchestration. However, the features already implemented for the other two use cases will be exploited as much as possible. For instance, to modify a VDC service, the features already implemented in OpenStack Heat will be re-used.

5 Layered Architecture – Full Functional Specification

The COSIGN orchestrator is based on OpenStack. Thus, the following sections are presented with respect to the architecture of OpenStack Heat, as a core of the COSIGN orchestrator, and various other OpenStack components such as resource controllers (e.g. Nova, Neutron) and monitoring plugins (e.g. Ceilometer). Figure 36 shows the OpenStack architecture mapping to the COSIGN orchestrator layers. Apart from the OpenStack Heat internal abstractions and utilities (e.g. parser, service, etc.), three main functional layers have been identified with respect to the COSIGN orchestrator:

1) The *Infrastructure control and monitoring clients'* layer (orange boxes in the figure): serves as the main point of contact with underlying resource controllers and plugins (Nova, Neutron, Ceilometer). The orchestrator is able to monitor and perform control operations on the underlying infrastructure through the capabilities of the various clients belonging to this layer.

2) The *Data stores and algorithms* layer (green boxes in the figure): this layer contains the main logic for deploying the COSIGN services and also the necessary data stores to keep the information about the underlying physical infrastructure and the deployed virtual services.

3) *The Orchestrator's API* layer (blue boxes in the figure): this layer consists of the APIs needed by an external entity to communicate with the orchestrator in order to request new services or configure existing ones.

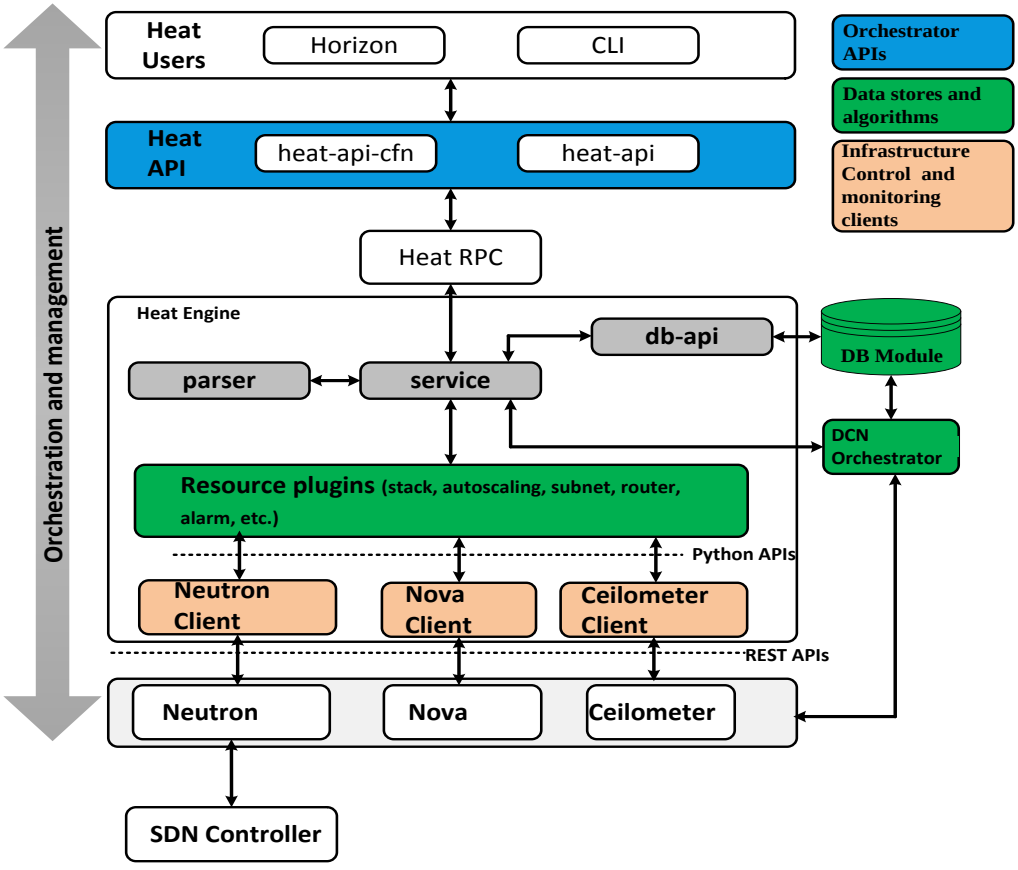

Figure 36 – OpenStack Layered Architecture

5.1 Infrastructure Control and Monitoring Clients' Layer

The lower layer of the orchestrator comprises the clients for the resources controllers. As seen in Figure 36, the infrastructure control and monitoring client's layer serves as an adaptation layer between the OpenStack Heat core components and each individual resource controller (e.g. Nova, Neutron, etc.). In general, the role fulfilled by the various components within this layer is to translate

between python API calls, received from Heat core components, and the REST API calls towards the resource controllers. Hence, the components act as clients for the resource controllers.

This layer comprises several client components such as:

- Nova client: is responsible for realizing the communication with the Nova controller. This client exposes python APIs inside the orchestrator, and maps them to Nova REST APIs.
- Neutron Client: is responsible for realizing the communication with the Neutron plugin (network controller). This component translates the python API calls received from other internal components of the orchestrator into REST APIs calls towards Neutron.
- Ceilometer Client: responsible for realizing the communication with Ceilometer module. Ceilometer is responsible for monitoring and alarm configuration.

Only the Nova, Neutron and Ceilometer client will be detailed in the following since they are the important ones for COSIGN.

5.1.1 Components of the Infrastructure Control and Monitoring Clients' Layer

5.1.1.1 Nova Client

Figure 37 illustrates the Nova client component and its interaction with other entities belonging to the orchestration layer. The most important resources that are supported by Nova are: Servers, also referred to as 'Instances' for historical reasons, Server Groups, and Flavors (see Section 4.4.1). When the Heat engine performs an operation on any of the Nova resources, the resource gets a handle of the Nova client and delegates the operation to the client through the available Python API. Further the Nova Client translates the method call to a REST API request and sends it to the Nova Controller.

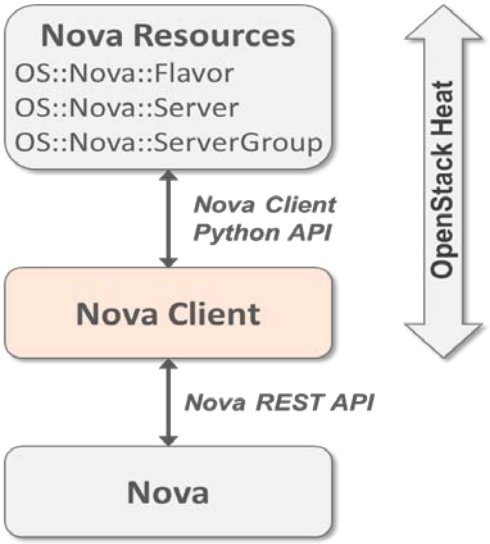

Figure 37 – Nova Client Component Interaction Diagram

5.1.1.2 Neutron Client

The Neutron client interacts with the Neutron controller service on the southbound, and various network resources on the northbound. The interaction is captured in Figure 38.

When Heat instantiates a *stack* for a tenant, it invokes the instantiation of all the resources composing that stack (e.g. network, subnet, router, firewall, etc.). In this way, each resources implements a set of handlers (e.g. `handle_create()`, `handle_delete()`, etc.) that are called by Heat to perform CRUD operations for that specific resource. Each Neutron resource has a reference of the Neutron client, and implements the logic by executing API calls against the Neutron client.

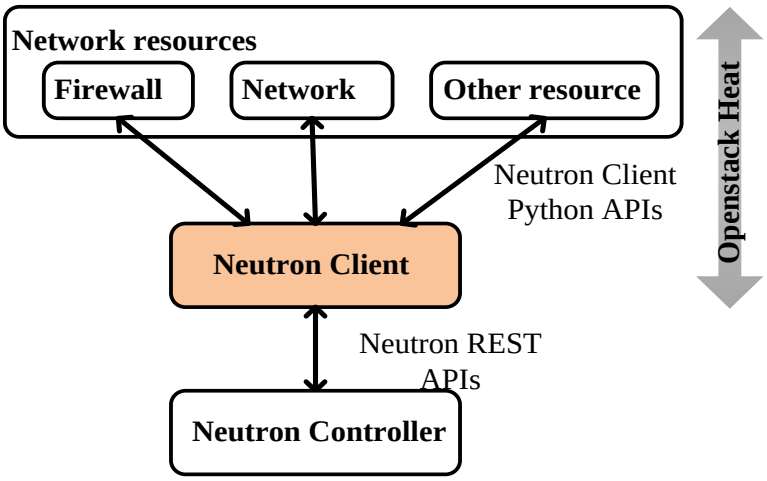

Figure 38 – Neutron Client component interaction diagram

For example, when a firewall is created a request towards the Neutron client is made, including the properties of the firewall to be created. The following Python method is called:

```
neutron_client.create_firewall(firewall_props)
```

In conclusion, the Neutron client receives requests to create, read, update and delete network resources, as Python API calls, and transforms these requests into REST requests towards the Neutron service.

5.1.1.3 Ceilometer Client

The Ceilometer client interacts with the Ceilometer service at southbound, through the Ceilometer REST APIs [43]. The resource that interacts with the Ceilometer client is the a*larm* resource (Figure 39). The alarm resource represents a rule for what measurements to perform in the infrastructure and what actions to take when the measurements do not comply with the specified thresholds. By configuring alarms, the Heat component can implement auto-scaling capabilities: update (scale) a set of resources when an alarm is triggered.

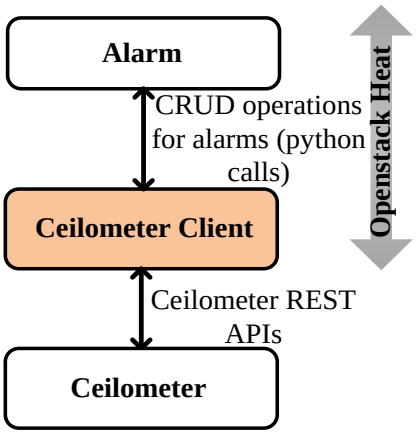

Figure 39 – Ceilometer Client component interaction diagram

5.2 Data Store and Algorithms Layer

The data store and algorithms layer fulfils mainly two tasks. On one hand, it performs the orchestration of the services to be deployed, providing the Heat engine module, which is the core of the orchestration service, with a collection of provisioning algorithms that take decisions on the resource allocation for user requests' coming from the orchestrator APIs layer. Moreover, it provides specific algorithms to apply re-planning/re-provisioning decisions for already deployed services. All of these algorithms are periodically fed with the status of the deployed services and the physical infrastructure in order to do some adjustments to improve the performance/utilization of the data

centre and the deployed services. On the other hand, this layer also contains the data store of the orchestrator, which stores information regarding the currently deployed services and applications. Moreover, it also stores information regarding the infrastructure resources inventory received from the infrastructure clients (Nova, Neutron, etc.) in order to feed the provisioning algorithms with the latest information concerning the status of the physical infrastructure. Figure 40 depicts the generic block diagram of the layer. It essentially consists of three differentiated parts:

- Heat Engine: it contains the OpenStack Heat service, which takes the output of the provisioning algorithms and communicates with each one of the infrastructure clients in order to orchestrate the service/application petition.
- DCN Orchestrator: it provides several algorithms to perform the selection of the physical resources to deploy a service/application request according to its characteristics and the status of the physical infrastructure.
- DB Module: it contains the databases to store information regarding the deployed applications and the resource inventory.

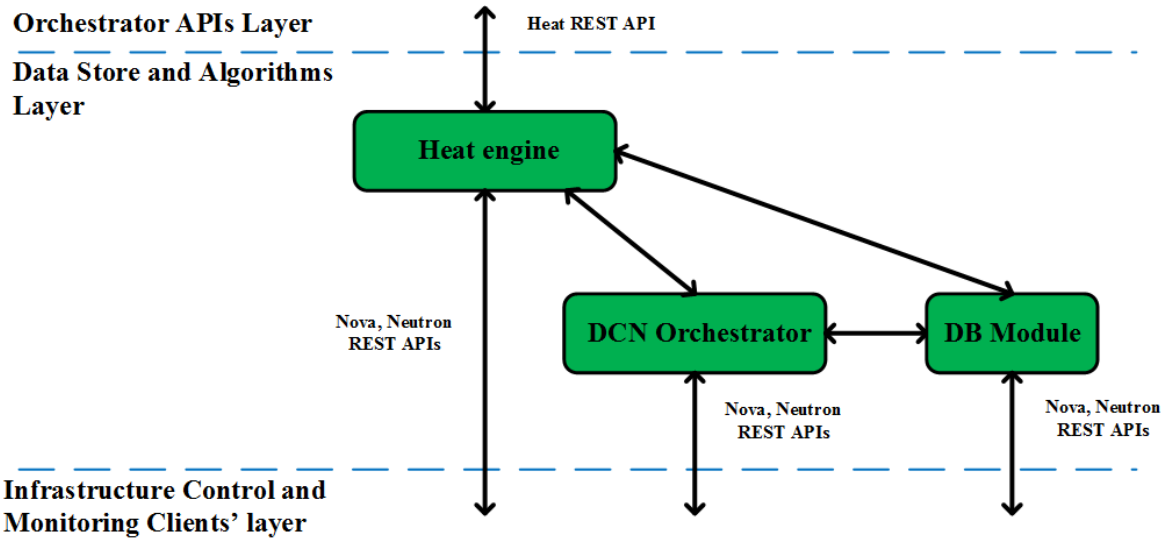

Figure 40 – Block diagram of the data store and algorithms layer

5.2.1 Components of the Data Store and Algorithms Layer

5.2.1.1 Heat Engine

The Heat engine module contains the Heat service, which is responsible for interaction with the associated resource client services (e.g. Neutron, Nova, etc,) in order to deploy the infrastructure specified in the YAML template representation of the user request. Upon validation of the template, the Heat service will decide if the current request requires the computation of an algorithm in order to find the optimal mapping of the request. This being the case, it then communicates with the DCN orchestrator, which after performing the calculations will return a modified YAML template stating the details of the mapping. In this regard, COSIGN will provide modifications to the Heat service and the YAML templates in order to handle such interaction. Once the modifications to the template have been applied, the Heat service will continue with the orchestration process as normal. For this, it is also the responsible for the creation and management of the stack, communicating with several resource plugins, one for each type of resources: compute, network and storage. The mission of these plugins is to allow operators of OpenStack environments to provide custom resource handlers to configure and reserve the desired infrastructure. Hence, the Heat service module also defines these plugins. The Heat service module communicates with the control and monitoring clients' layer through the associated APIs at the northbound interface. Finally, the Heat service module communicates with the DB module in order to store information regarding the data models of the deployed services/applications into the deployed applications database. Figure 41 provides a detailed view of the Heat engine module.

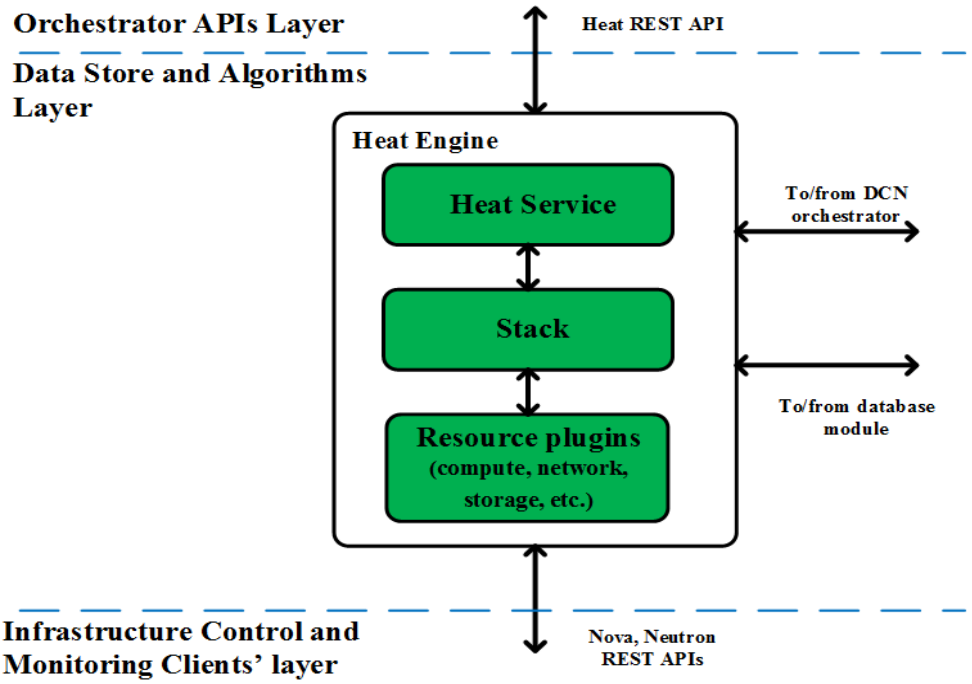

Figure 41 – Block diagram of the Heat engine module

5.2.1.2 DCN Orchestrator

The DCN orchestrator module is composed of several sub-modules, which provide provisioning and mapping decisions for each one of the COSIGN use cases, namely, virtual data centre (VDC) provisioning, virtual cloud application (vApp) and data centre operation and management (O&M). The DCN orchestrator module receives a petition for a mapping computation from the Heat module through the Heat REST API interface. Then, the algorithm broker module decides, depending on the request, which algorithm should be called. Once the algorithm has been selected, the corresponding sub-module fetches resource availability information from the DB module, which contains the most updated information regarding the resource availability at the infrastructure layer. With this information, along with the characteristics of the request, it calculates the necessary resource mapping. Once the calculation has ended, it informs back the Heat engine, through the algorithms broker, with a modified YAML template containing the details of the mapping.

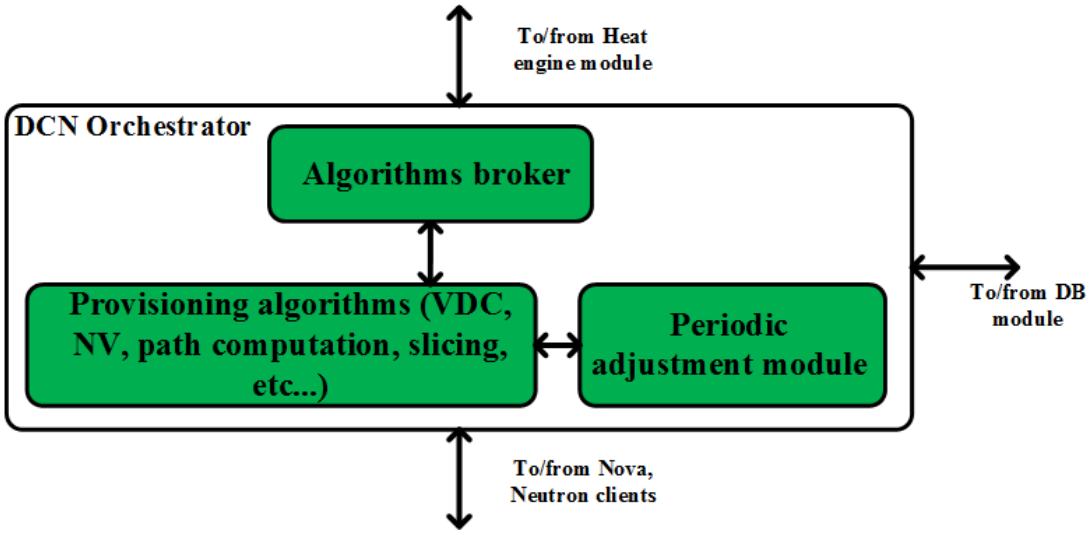

Figure 42 – Block diagram of the DCN orchestrator module

Besides the specific algorithms, there is a special sub-module called, periodic adjustments, whose goal is to allow for changes/adjustments to already deployed services/applications. For this, it constantly polls the database and, once it detects that a resource or service status has changed, it checks if the

current allocation policy of the DC operator has been violated. This being the case, it issues a petition for re-planning/re-mapping to the corresponding algorithm, stating which deployed services should be modified. Keep in mind that such a process is internal to the DC operator, so the tenants/clients should not be affected, i.e. no service disruption should be experienced. Figure 42 summarizes the structure of the algorithms module.

Additionally, the DCN orchestrator module has a direct interface towards the Nova and Neutron clients at the southbound. The mission of such interface is to listen to critical infrastructure changes, i.e. due to a failure. In this way, the algorithms can react more rapidly and decide which should be the new mapping of the deployed services in order to overcome the detected failure. Nevertheless, for regular mapping decisions, the algorithms still rely on the information stored at the DB module.

5.2.1.3 DB Module

This module contains two databases, which are utilized to store information regarding the deployed services and the infrastructure status. The Heat service allows for the possibility to link a MySQL database to the heat engine. Such database, here named deployed applications DB, is utilized for the heat service to store the data models of the infrastructures deployed once the corresponding YAML templates have been processed and the related services contacted. Additionally, another database, named resource inventory DB, is defined and utilized to keep track of the available resources and their status at the infrastructure layer. The information on this database is accessed through simple SQL queries from the DCN orchestrator module as status information is needed during the provisioning decisions. Moreover, this database is periodically updated according to the current status of the infrastructure. The updates to that database are provided by the infrastructure management clients' modules at the control and monitoring clients' layer through the associated APIs. Figure 43 depicts a detailed view of the DB module.

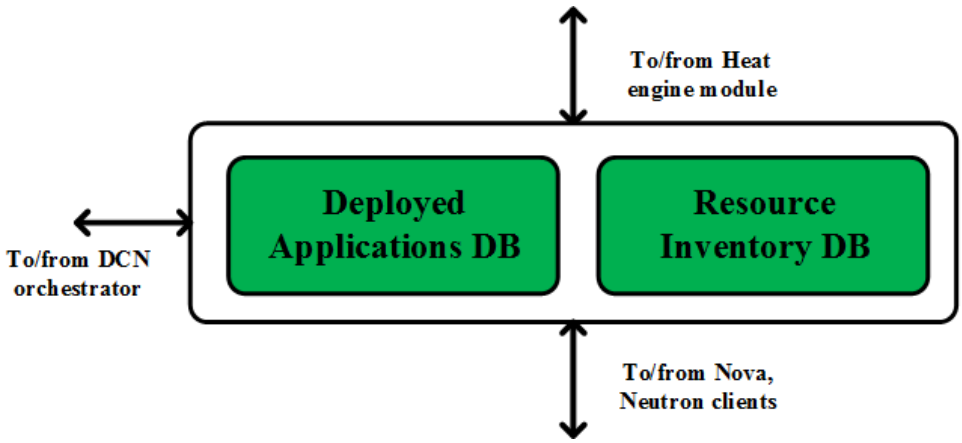

Figure 43 – Block diagram of the DB module

5.3 Orchestrator APIs Layer

The orchestrator APIs layer constitutes the outermost layer of the COSIGN orchestrator and is concerned with the presentation of COSIGN bounded services through APIs. Depending on the use case and/or scenario as well as the type of tenant (Data Centre users and administrators), the system is capable to expose underlying abstracted network information and deliver CRUD methods to manage and configure specific network capabilities and handle requests. The APIs are available based on REST calls, which more specifically aim to:

- Enable COSIGN tenants to specify through the REST calls:
  - Request of COSIGN services, e.g. VDC and/or vApp.
  - Request of information on the services or applications performance, e.g. monitoring, resources utilization, etc.

- Enable application developers to accelerate the application delivery by providing flexible access to COSIGN system by defining and easily exposing their own application APIs for testing, development, configuration etc.
- Enforce authentication, access control, and authorization policies while accessing to COSIGN systems and services.
- Integrate with upper level components, e.g. for exposing the COSIGN DCN capabilities and services to Operations Support System (OSS) and Business Support System (BSS) tools.

5.3.1 Components of the Orchestrator APIs Layer

Figure 44 presents the component level structure of the relevant OpenStack Client Interfaces. It can be seen that all the OpenStack services expose REST APIs for manipulating their respective data models. To ease the API access and refactor common components, OpenStack provides language bindings that implement packages or libraries around the services' APIs. These bindings are called REST API Clients and currently exist for Java and Python while other bindings can be added. Note that only python bindings are included in the official OpenStack releases and only these libraries are shown in *Figure 44*.

Above the REST API clients level there are shell utilities wrapping REST calls into executable CLI commands. Initially, each service and its respective REST API client were exposing its own executable for this purposes. While convenient for developers, this practice did not favour users as each service's CLI was designed differently encouraging ambiguity and inducing frustration among users and operators. At some point, unified REST API client and its wrapper executable were created in a project called OpenStackClient. While not all the currently existing OpenStack APIs are unified under this new common project yet, it is believed to be a preferable way to go so that individual services' CLI start to be deprecated. *Figure 44* presents these soon-to-become legacy CLIs in faded colour and the new unified components of OpenStackClient in a bright colour, to emphasize the contrast.

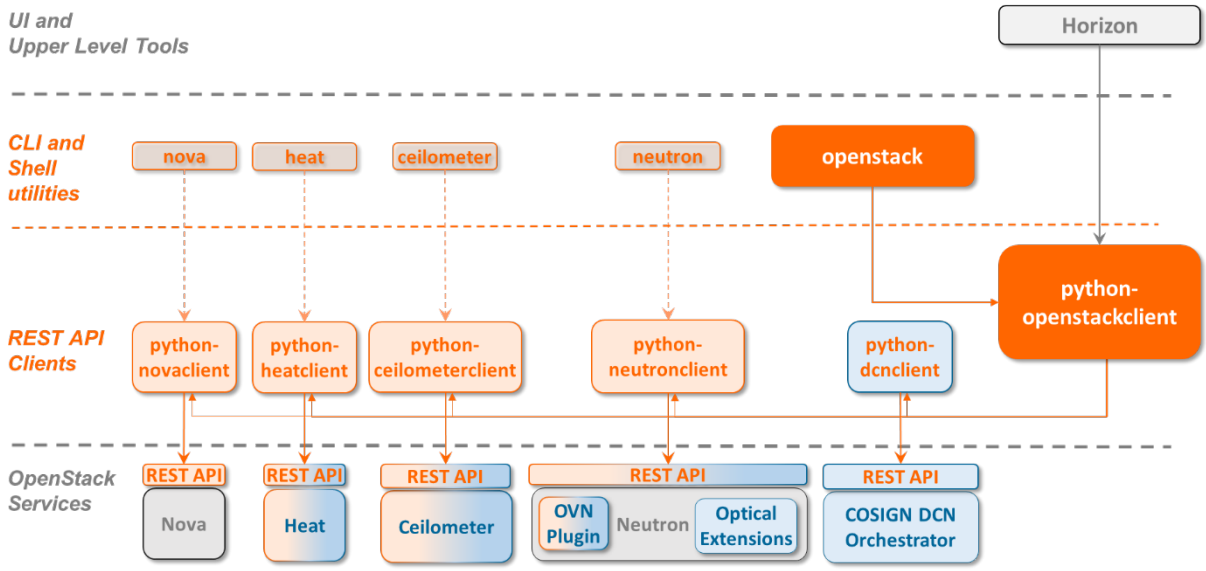

Figure 44 – Components of the OpenStack Client Interfaces

In addition to showing the existing components, *Figure 44* shows where COSIGN orchestrator's extensions might be needed, using blue hues. OpenStack UI service, Horizon, currently binds the service-specific REST API clients, while in the future will probably converge on using mostly the unified OpenStackClient binding, as shown in *Figure 44*, at least for the established projects and services. To improve COSIGN visibility, it is highly desirable to extend the OpenStackClient library and Horizon, in addition to what's shown in *Figure 44*. These extensions are however out of scope of the COSIGN orchestrator and might be considered as part of the demonstrator effort (WP5).

6 Internal Interfaces and Operational Flows

Section 5 has presented the architectural layers of the COSIGN Orchestrator, namely the Infrastructure Control and Monitoring Clients' Layer, the Data Store and Algorithm Layer, and the Orchestrator API Layer. In this section, the internal interfaces between these layers are defined, based on the functional specification of Section 5. In addition, this section shows how these interfaces support the user facing operational flows realising the COSIGN use cases presented in Section 4.

6.1 Inter-Layer Interfaces

6.1.1 Infrastructure Control and Monitoring to/from Data Store and Algorithms

In COSIGN, service requests from the u*Orchestrator API layer* will be elaborated by *Data Store and Algorithms layer* and then, based on the resource availability information, these requests will be implemented with proper network, storage and server configurations in the *Infrastructure Control and Monitoring layer*. Also, for infrastructure/service management and optimization purposes, through these interfaces, the *Data Store and Algorithms layer* should be able to collect monitoring information from the infrastructure layer and send out reconfiguration/modification requests. So, the main functions of the interfaces between the *Infrastructure Control & Monitoring layer* and the *Data Store & Algorithm layer* are:

- Retrieve information from *Infrastructure Control & Monitoring layer*, including general resource availability information, monitoring information, application, service user and service provider specific configuration data, as well as abstracted capabilities and service patterns.
- Execute configuration at the *Infrastructure Control & Monitoring layer*, including IT resource allocation and network configuration commands for VDC and vApp use case, as well as maintenance tasks and other miscellaneous operations for management and re-optimization purpose.

In the following section, more detailed interface specification will be defined.

As presented in Figure 40, the *Data Store and Algorithms layer* includes three functional modules – i.e. Deployed Application & Resource Database, Heat Engine and DCN Orchestrator. These modules and their functions are defined in Section 5.2.1. To support these functions, the detailed interface specifications are defined as following:

- Interface between *Infrastructure Control & Monitoring layer* and *Deployed Application & Resource Database*:
  - Retrieve Uniform Topology information with abstracted network and IT elements, e.g., server, links, virtual/physical network devices (L1 (i.e., optical switches) or L2/L3 devices), as well as their ports. More specifically, for first time, topology with integrated electronic and optical devices is introduced in COSIGN's vision.
  - The resource utilization/availability information should be abstracted by functional modules in the *Infrastructure Control & Monitoring layer* (i.e., Nova, Neutron, Cinder and the SDN controller [D3.2]) and exposed to the *Resource Inventory & Deployed Application Database*.
  - Collection of network infrastructure and service monitoring data, e.g. collecting abstractions for statistics about the network and traffic load (delay, packet errors, packet drops, etc.) through Ceilometer and the SDN controller.
- Interface between *Infrastructure control & monitoring layer* and *Heat Engine*:
  - Supported service patterns by Infrastructure layer needs to be exposed, e.g., virtual resources template (VM, vToR, vOS and vLink specification), group based policy model.

- Abstract Quality of Service (QoS) resources need to be exposed to serve QoS-enabled applications' requests, including bandwidth, latency, or other electrical-/optical-related QoS resources.

- Interface between *Infrastructure Control & Monitoring layer* and *DCN Orchestrator*:
  - Listen/track the service/application violations and resources status change events (e.g., physical layer failure) reported by Infrastructure layer.

As a summary, Table 11 shows all the required functions which should be supported by the interfaces between functional modules of the Infrastructure Control and Monitoring Clients' Layer and the Data Store and Algorithms Layer.

Modules in Orchestrator	**Modules/APIs in Infrastructure control and monitoring layer**		**Extension**	**Supported Operations**
Deployed Application & Resource Database	COSIGN SDN controller	network-topology	Integrated topology with optical and electronic switches	GET
		network-statistics	Add optical layer performance features	
		device-capability	Add abstracted optical switch capability	
	Nova	Servers/Flavors/Limits	not expected	
	Network (Extended Neutron)	Subnets/Ports	Report abstracted optical link/switch capability and QoS features (e.g., delay)	
	Storage (Cinder)	Volume/QoS specification/Limits	not expected	
	Ceilometer	Resources/Meters	Add optical layer monitoring features	
Heat Engine	COSIGN SDN controller	Overlay virtual networks Virtual optical slices Optical connectivity	Extend to support CRUD operation for all the COSIGN use cases [D3.2]	GET PUT DELETE
	Compute (Nova)	Servers/Flavors	Not expected	
	Network (Extended Neutron)	Subnets and vLink mapping Optical connectivity provisioning	Be able to parse vLink bandwidth requirements to end-to-end path, including optical connectivity	
	Storage (Cinder)	Volume/QoS specification	Not expected	
	Ceilometer	Meters	Not expected	
DCN Orchestrator	COSIGN SDN controller	DCN information service	Add network resource status and service violation monitoring capability	GET PUT DELETE
	Compute (Nova)	Process Monitoring/Resource Alerting	Not expected	
	Network (Extended Neutron)	Resource Alerting	Not expected	
	Storage (Cinder)	Resource Alerting	Not expected	
	Ceilometer	Meters	Not expected	

Table 11 – Interface between the Infrastructure Control and Monitoring Clients' and the Data Store and Algorithms layers

6.1.2 Orchestrator APIs to/from Data Store and Algorithms

In this section, the interface between the *Data Store and Algorithms layer* and the *Orchestrator APIs layer* is defined. The main purpose of such an interface is to allow for issuing of service requests to the orchestrator Heat engine module and allow the CRUD operations of the different services identified in COSIGN.

In COSIGN, we will utilize the OpenStack Horizon project as a point of entry for petitions. Horizon is a web-based dashboard which gives the possibility to provide a customized graphical front-end for easy access to orchestrated service petitions. Once a user, or the data centre administrator, has issued a petition through Horizon, it will then communicate through the Heat REST API with the Heat engine module at the *Data Store and Algorithms layer*, providing the high level characteristics of the request. Then, depending on the type of request, the DCN orchestrator module will compute the resource allocation, which will be fed to the Heat service in a YAML template form, in order to orchestrate the overall infrastructure.

The main tasks that this interface must support are:

- Virtual network (VN) use case:
    1. Deploy VN as described in the template.
    2. Destroy VN.
    3. Update VN.
    4. Health/performance reports for the deployed VN.
- Virtual Data Centre (VDC) use case:
    1. Deploy VDC as described in the template.
    2. Destroy VDC.
    3. Update VDC.
    4. Health/performance reports for the deployed VDC.
- Data centre operation, management and orchestration:
    1. Direct infrastructure management (start/stop VM, establish a communication flow between VMs, perform traffic switching in a VDC, etc…).
    2. Indirect infrastructure management – react to failure or congestion, optimize operational tasks, e.g. image loading on boot, back up, replication.

Basically, the *Data Store and Algorithms layer* to *Orchestrator APIs layer* interface consists of the information comprising the characteristics of the requests, either service requests or management operations. The information of the request is passed to the *Data Store and Algorithms layer* through the Heat REST API, which also provide with the reply to the request initiator.

Modules in Orchestrator	**Modules in Orchestrator APIs layer and related service**		**Extension**	**Supported Operations**
Heat engine	Horizon	VDC service	Add the possibility to create vToRs and vOS (optional)	GET/PUT/POST/ DELETE
		NV service		GET/PUT/POST/ DELETE
		Data center operation and management	Allow the possibility to perform optical switching (enhanced VDC service; optional)	GET/PUT/POST/ DELETE

Table 12 – Interface between the Orchestrator APIs and the Data Store and Algorithms layers

To facilitate the comprehension of the information exchange between the Orchestrator APIs and the *Data Store and Algorithms layer*, Table 12 summarizes the involved modules, as well as the supported functions for the three use cases identified in COSIGN, namely VDC, vApp, and O&M. Essentially, the interface has to support CRUD operations (GET, PUT, POST, DELETE) for all three identified services. The content and parameters of the operation are specific to the operation itself, e.g. in the case of the VDC service, when applying a modification to a current deployed VDC, a PUT function will be called, passing as parameters the identifier of the VDC along with the YAML template description of the modified VDC.

6.2 Operational Flows Realizing COSIGN Use Cases

6.2.1 VDC Use Case

6.2.1.1 Create VDC

The first operation described in this section is the creation of a VDC instance, like the ones presented in Figure 15 and Figure 16. For completeness, we will illustrate the creation process for the enhanced VDC service, since it includes all the potential elements of the defined VDC instance. The creation workflow is executed through the coordinated invocation of different entities and services at the COSIGN cloud platform, mainly through the exchange of REST messages, as shown in Figure 45.

The tenant specifies the characteristics of the desired VDC request through the VDC dashboard service. Once the VDC has been fully specified, the tenant validates the request and a YAML template is elaborated and sent to the Heat service module at the orchestrator through Heat REST API. There, the template is parsed and validated. Upon validation, the heat service requests to the algorithms module to find the optimal mapping (according to the DC operator policies) of the VDC request.

For this, the algorithms module first fetches the resource availability, both compute and network, from the database, which stores the information regarding resource availability. Such information is actualized by Nova and Neutron clients every time the status of the available resources at the infrastructure layer change. With the fetched information and the characteristics of the VDC request, the algorithms module computes the VDC mapping and returns the specifications of the mapping to Heat (e.g. through a modified YAML template or utilizing the internal Heat classes).

After receiving the VDC mapping details, the Heat service module creates the stack object, which is the request representation inside OpenStack. Once the stack has been created, it is invoked recursively with the *create()* method in order to instantiate the VDC resources through the proper resource plugins (Instance, vNet, vToR, etc.). Depending on the type of resource, the Nova service (for compute resources) or the Neutron service (for network resources) are contacted through their clients, which will then trigger the creation and configuration of the requested resources.

Once all the resources have been successfully deployed and configured, the Heat service answers back to the VDC dashboard with the characteristics of the deployed VDC instance. From this moment, the tenant can start operating his VDC infrastructure.

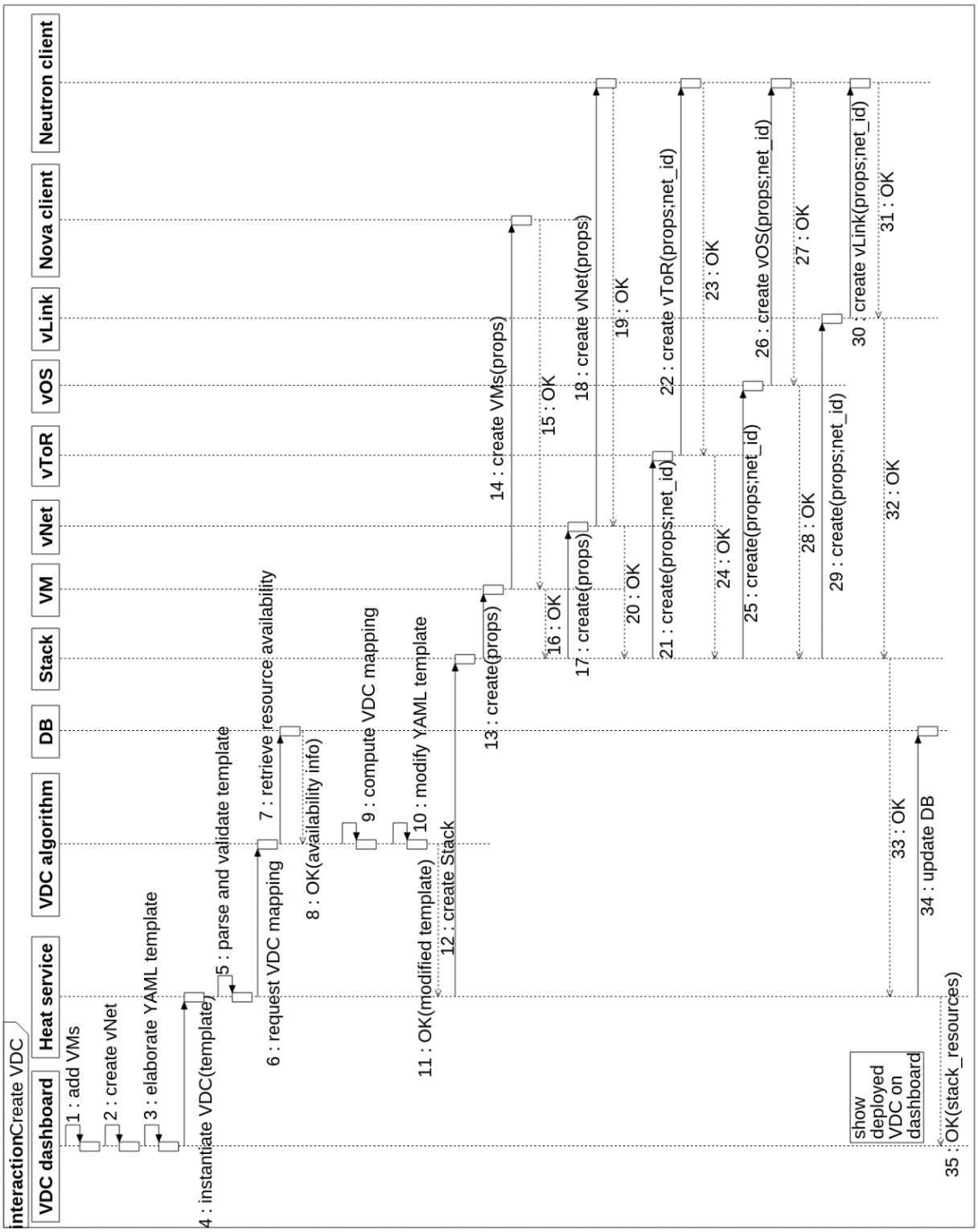

Figure 45 – Message sequence chart for VDC creation

6.2.1.2 Retrieve VDC Information (read)

The read operation, that is, retrieving the information of the deployed VDC instance, its resources and their characteristics, mainly consists of fetching the corresponding stack object from the database. Then, the stack can be invoked to list the resources of the stack through the Heat command *resource-list*. After that, the specific information of all the stack resources can be fetched by recursively

invoking the *resource-show* command. Once all the information has been gathered, a description of the resources can be returned to the request initiator. Figure 46 depicts the whole message exchange.

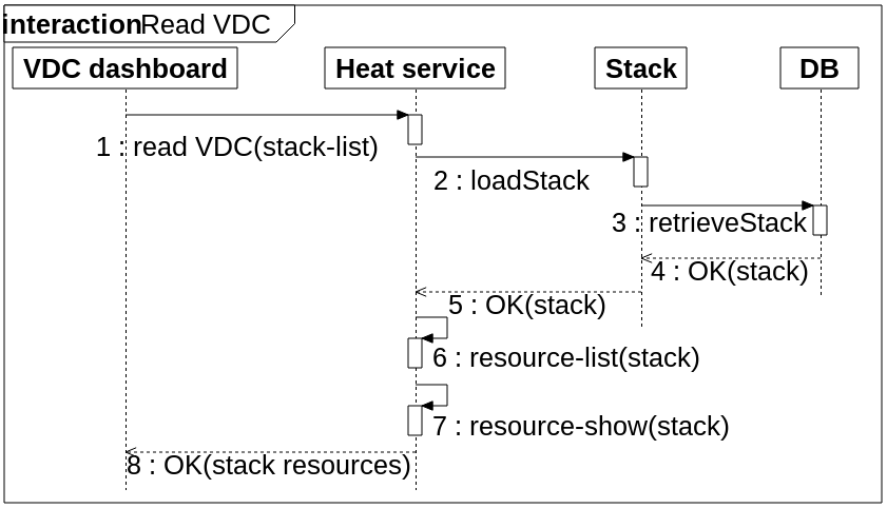

Figure 46 – Message sequence chart for reading a VDC

6.2.1.3 Update VDC

The process of updating the already deployed VDC to another VDC consists mainly on updating the corresponding stack through a modification of the original template. For this, first the tenant has to edit the VDC through the VDC dashboard service. Once it has been edited, a new YAML template representation of the request will be sent to the Heat service module. There, the service will retrieve the old stack from the database. Next, the stack invokes the update method on itself, passing as argument the new stack. The update of the stack resources depends largely on the type of resources and the wanted update. Some resources are updated in-place, while others require to be replaced by new ones, thus, the old ones must be destroyed. Once the stack has triggered the update command, it is applied recursively on all the resources that constitute the stack. After all of them have been updated, the new stack is returned to the heat service, which stores it into the DB and informs back the request initiator. Figure 47 depicts the whole process.

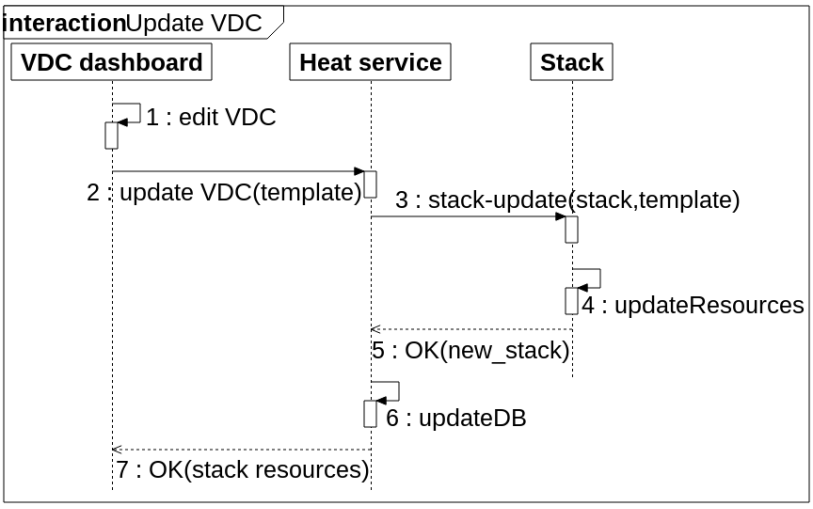

Figure 47 – Message sequence chart for VDC update

6.2.1.4 Delete VDC

The last detailed operation is how to delete an already deployed VDC instance (Figure 48). First of all, the tenant triggers the delete operation through the VDC dashboard service, which is then sent to the Heat service module through the Heat REST API. Then, the service module retrieves the corresponding stack from the database. Once the stack object has been fetched, the delete method is invoked on the stack, which, at its turn, triggers recursively the deletion of all the resources of the

stack, calling iteratively the delete method on all of them. Hence, each resource must know how to delete itself, and it is mainly done by delegating the operation to the specific resource controller (e.g. Neutron, Nova). Then, the corresponding controller destroys the virtual resources, releasing the reserved underlying resources. Once all the resources have been correctly liberated, the tenant is informed of the success of the operation.

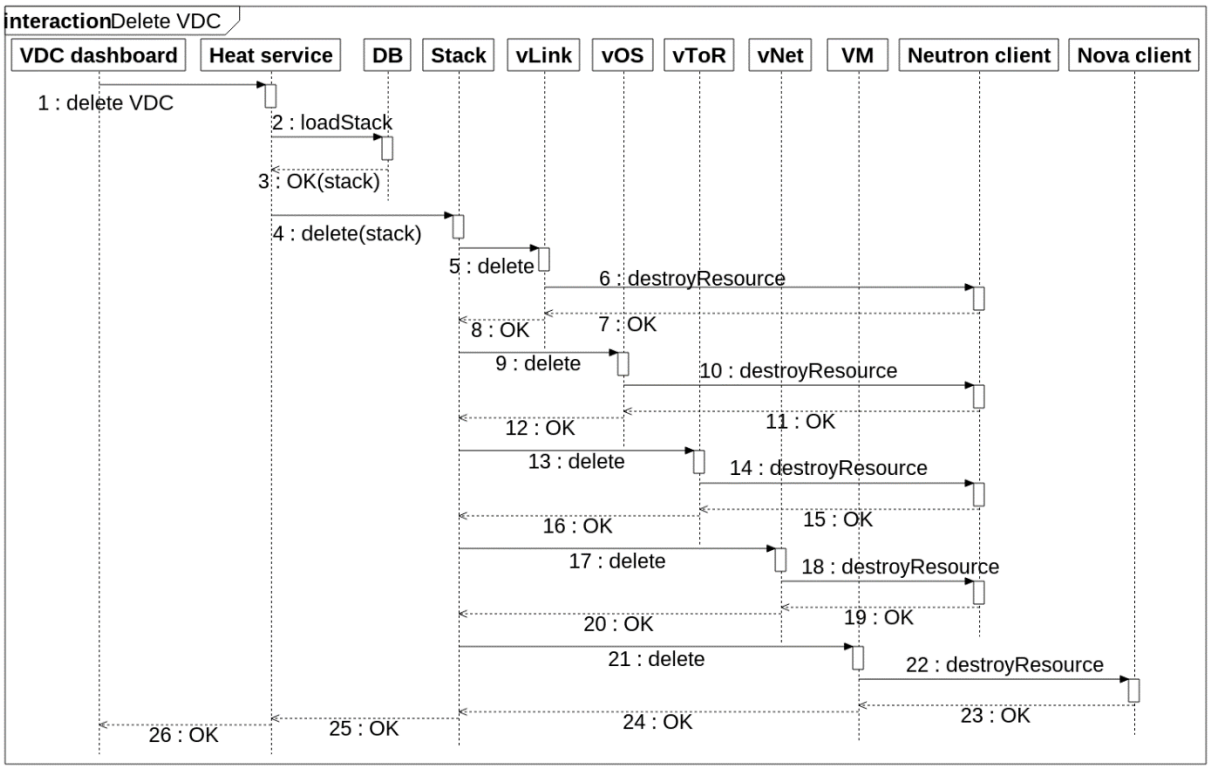

Figure 48 – Message sequence chart for VDC deletion

6.2.2 vApp Use Case

6.2.2.1 Create vApp

The first operation described in this section is the vApp creation.

Figure 49 shows the orchestrator layer message sequence chart for the creation of the virtual network depicted in *Figure 30*. Since at the orchestrator level the definition of services is done through templates, the virtual network definition is included in a template. Further, the virtual network is part of a *Stack*, which is a set of resources provisioned to a tenant. For illustration purposes a VM (i.e. instance) is also included in the diagram.

After the request for virtual network creation (i.e. template) has arrived at the *Service* component, which is the main coordinator in the orchestrator, it is processed by a parser, to get the *stack* object. Then the stack is invoked for creation, which recursively invokes the *create()* method on all the resources belonging to the stack (network and compute resources). After the resources have been created, the stack is saved to the database and a description is returned to the request initiator.

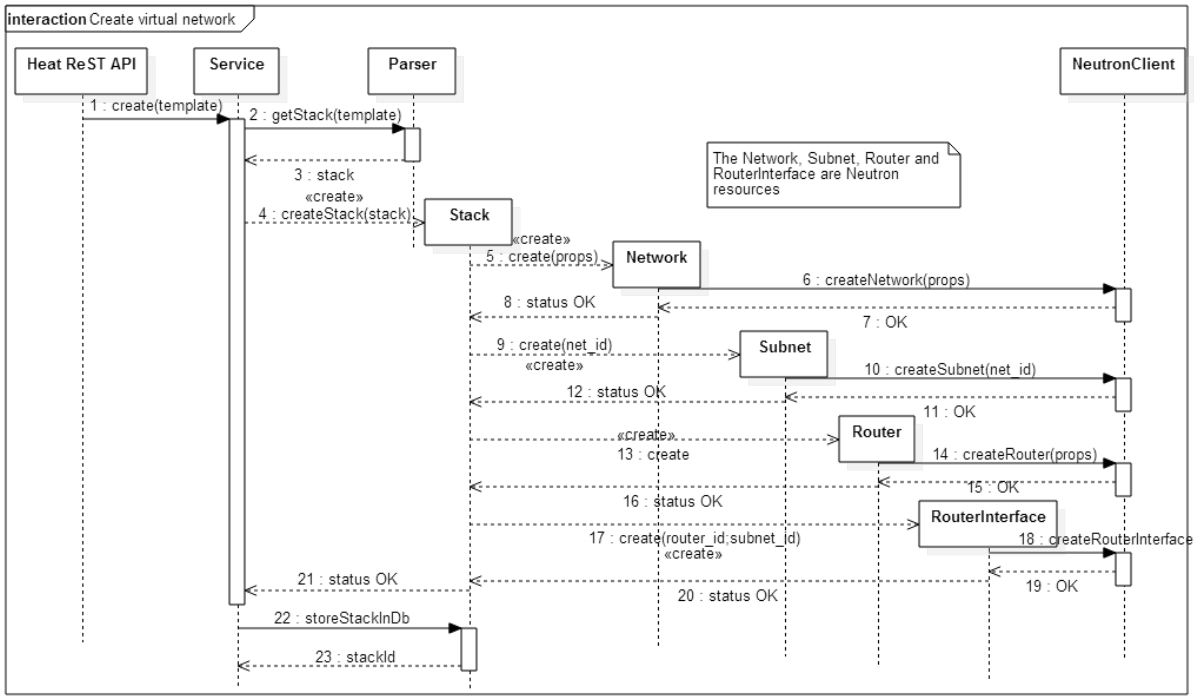

Figure 49 – Message sequence chart for virtual network creation

6.2.2.2 Retrieve vApp Information (read)

The process of retrieving a virtual network is as simple as fetching the corresponding stack from the database, looping through its resources, and returning a description of those resources as a string (Figure 13).

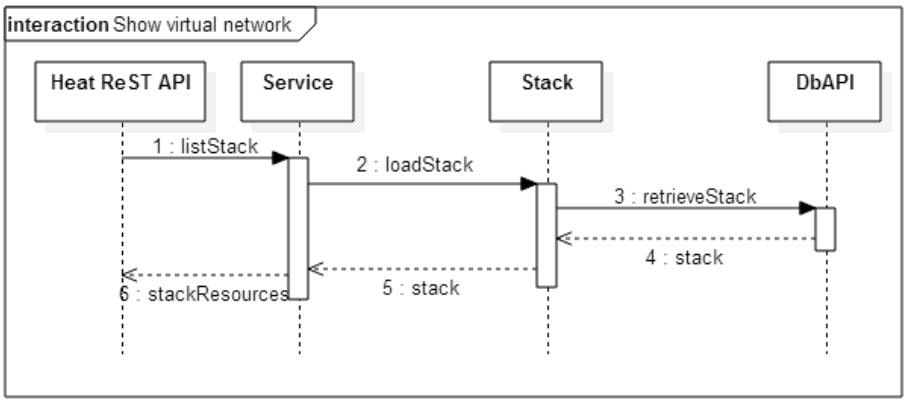

Figure 50 – Message sequence chart to show virtual network

6.2.2.3 Update vApp

To update a virtual network to another virtual network, the tenant must specify a new stack configuration in its request. The service component retrieves the old stack from the database and invokes an update on itself, passing as argument the new stack. The rest of the process largely varies on the type of update wanted for the resources. Some resources can be easily updated, some have to be destroyed and others created in place. The algorithm for update in included in the *Stack*, and recursively triggered on the constituent resources, such that all resources must know how to update themselves.

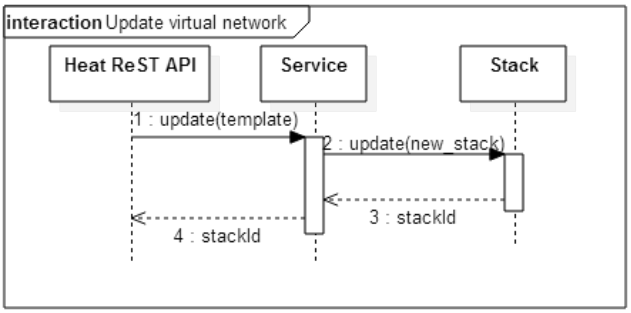

Figure 51 – Message sequence chart for virtual network update

6.2.2.4 Delete vApp

The process of deleting a virtual network is represented in *Figure 52*. The first step is to fetch the corresponding stack (which includes the virtual network) from the database. Then the delete method is called on the stack, i.e. a method which is recursively called on all the resources belonging to the stack. Hence, each resource must know how to delete itself, and it is mainly done by delegating the operation to the specific resource controller (e.g. Neutron).

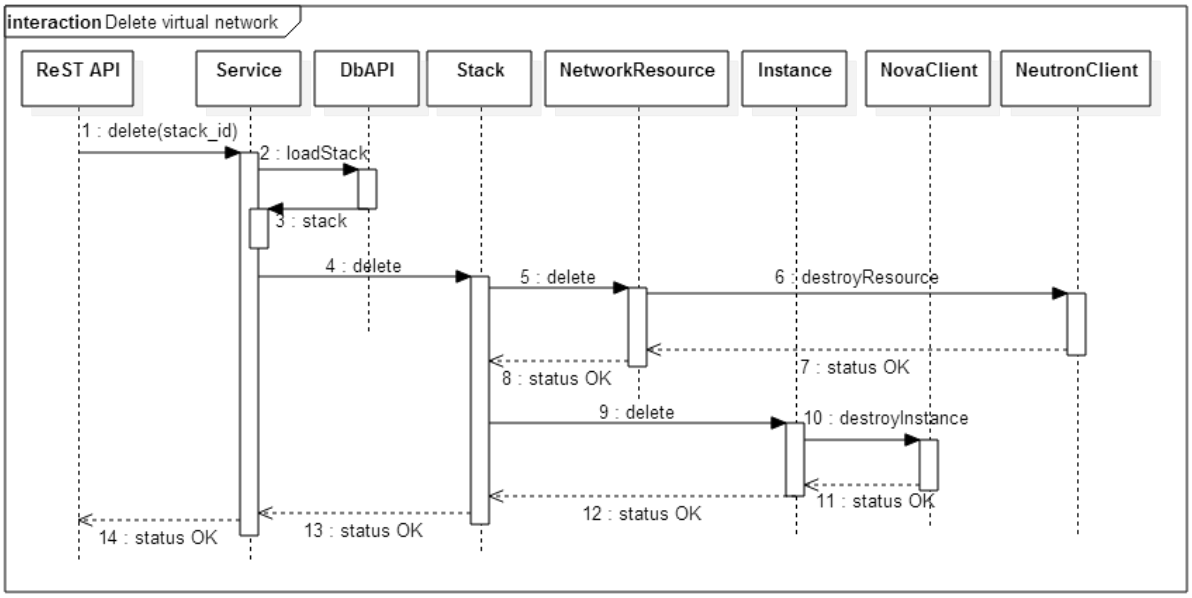

Figure 52 – Message sequence chart for deleting a virtual network

6.2.3 Operations and Management Use Case

Based on the requirements previously defined, this section describes the specific workflows for the following management operations. While [D4.1] describes two of the management tasks (i.e. create new VM and provide connectivity), this section describes the workflows for other management operations such as inventory and monitoring of resources and execution of optimization algorithms.

Figure 53 shows the message sequence chart for various tasks that are related to visualization of resources, and also configuration of alarms. Here, the administrator visualizes the resources (devices, services, other components). This means that there is an inventory of the resources (at different abstraction layers) and that the monitoring information about the resources is fetched and made available at the DC O&M dashboard.

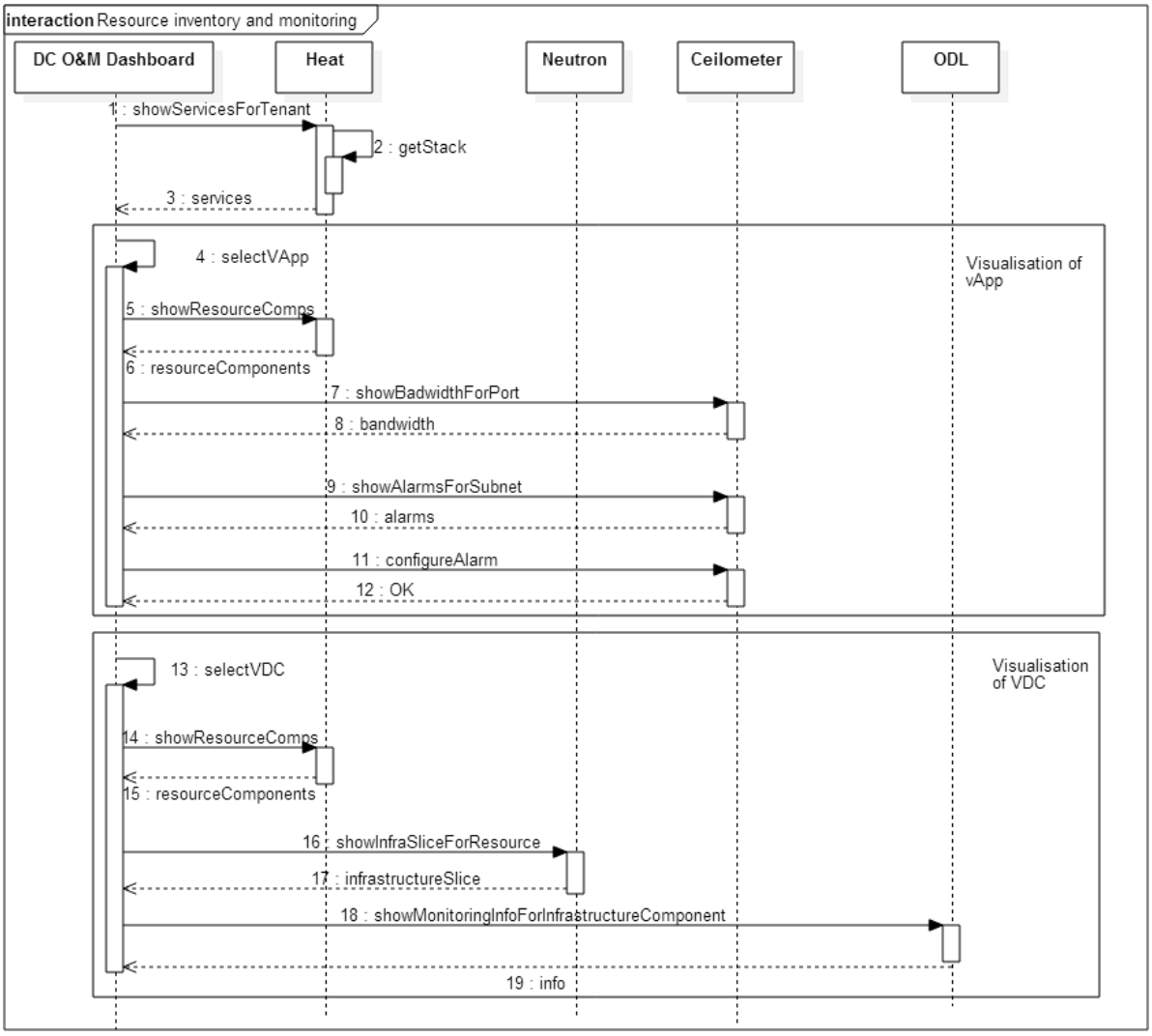

Figure 53 – Inventory and Monitoring of resources

Initially the admin issues a command to get the services for a tenant. The Heat entity fetches the corresponding stacks from the DB and replies to the DC O&M entity, where the resources are displayed (1-3). The admin choses a service (i.e. a vApp), and further initiates a request to Heat to see the components of the selected vApp (5-6). The admin initiates a request to the DC O&M to see monitoring info regarding a port belonging to the vApp. The request is sent to Ceilometer which replies with the needed info (7-8). The admin request for alarm information and then configures a new alarm (9-12).

In the second part, from all the services belonging to the tenant (requested initially 1-3), the admin selects a VDC and initiates a request to visualize the components of this service. The request is handled by Heat which returns the needed info to the DC O&M (14-15). A VDC is mapped into a physical infrastructure, according to the output given by the DCN orchestration entity which contains the resource optimization algorithms. The admin wishes to visualize the mapping of the provisioned virtual resources belonging to the VDC, onto the physical infrastructure. To this end, a request is sent to Neutron to get the infrastructure slice that lies under the VDC (16-17). Further, another request is sent to ODL to get low level monitoring info regarding a resource in the DCN. The admin now monitors the physical infrastructure that support a VDC service provisioned to one of the customers.

Figure 54 shows the message sequence chart for executing optimizations of the resource mapping onto the physical infrastructure. The first part of the diagram in Figure 54 has been explained above. After the admin selects a VDC for a tenant, he triggers a re-optimization of resources by using a specific algorithm and objective. The DC O&M first constructs a template (6-7) then initiates a request to the DCN Orchestrator for a new mapping of resources. Following, the DCN Orchestrator executes the

algorithm and returns a new template capturing the new mapping (9-10). The DC O&M triggers a stack update procedure to Heat, passing the new template. Further, Heat updates the resources according to the new mapping (12-15). In the end, the admin has re-optimized the resource allocation for a particular VDC.

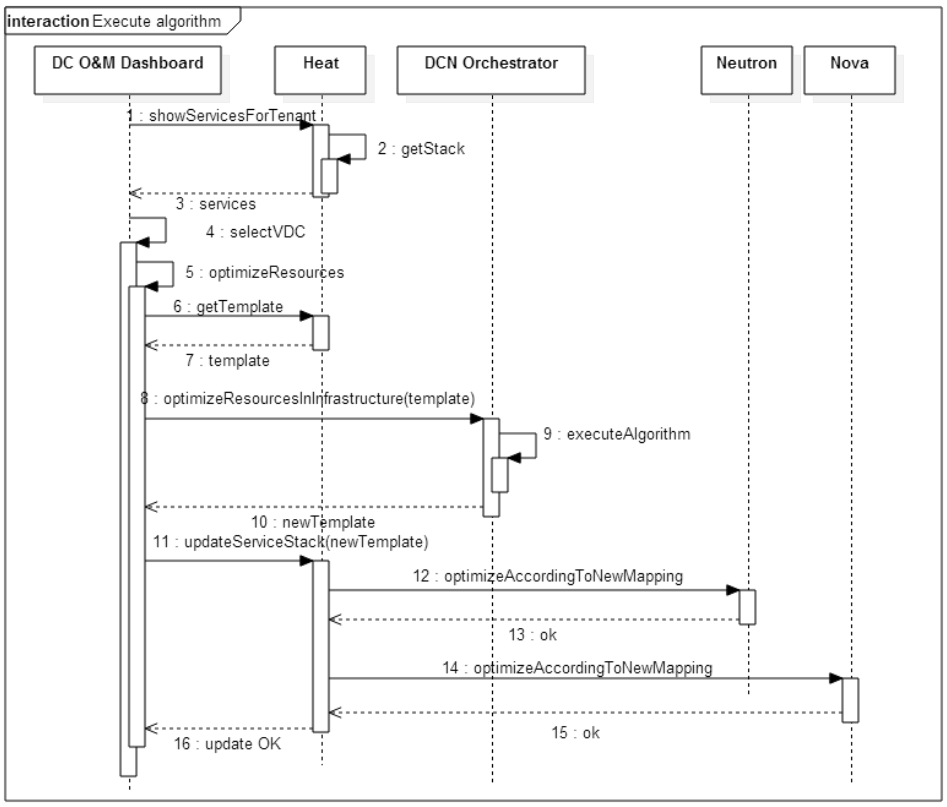

Figure 54 – Execution of optimization algorithm

7 Orchestrator's External Interfaces

This section specifies the external interfaces of the COSIGN Orchestrator. These interfaces are: the southbound interfaces towards to Infrastructure controllers and the northbound interfaces towards the clients realizing COSIGN use cases.

7.1 Northbound Interfaces

Orchestrator's northbound interfaces are interfaces towards the clients and upper level management tools layer. These are the interfaces whereby COSIGN use cases will be triggered.

In this section, we provide the Application Programming Interfaces (APIs). The APIs can be invoked by human users manually. API calls can also be included in the shell scripts that automate the commonly occurring steps or groups of steps. APIs can be wrapped by the UI workflows for scenarios where visualization can add value. Last but not the least, APIs or their wrappers (e.g. language bindings) can become part of higher level workflows in OSS and BSS tools. Defining all these upper layer interactions and workflows are out of the Orchestrator's scope and will be considered as part of the COSIGN demonstrator effort, in WP5.

7.1.1 Interfaces Specification

As it was pointed in section 5.3.1, and shown in Figure 44, the COSIGN Orchestrator APIs Layer relies on the OpenStack REST APIs which enable to manipulate the respective service data models. The REST API Clients binding libraries [46] around the services' APIs enable to ease the API access and the executable CLI commands wrapping was created in the OpenStackClient project for users and developers convenience.

The end user/application is able to request for the service by means of the CLI interface or making use of the Horizon dashboard. In the first case, it will be used by the CLI commands that wrap the OpenStack interface. In the latter case, use case operations (create, delete, show, edit, etc.) will be specified by means of an adapted use case Heat template. In both cases, REST APIs of each of the involved OpenStack services will be called making use of corresponding the REST API binding libraries (clients).

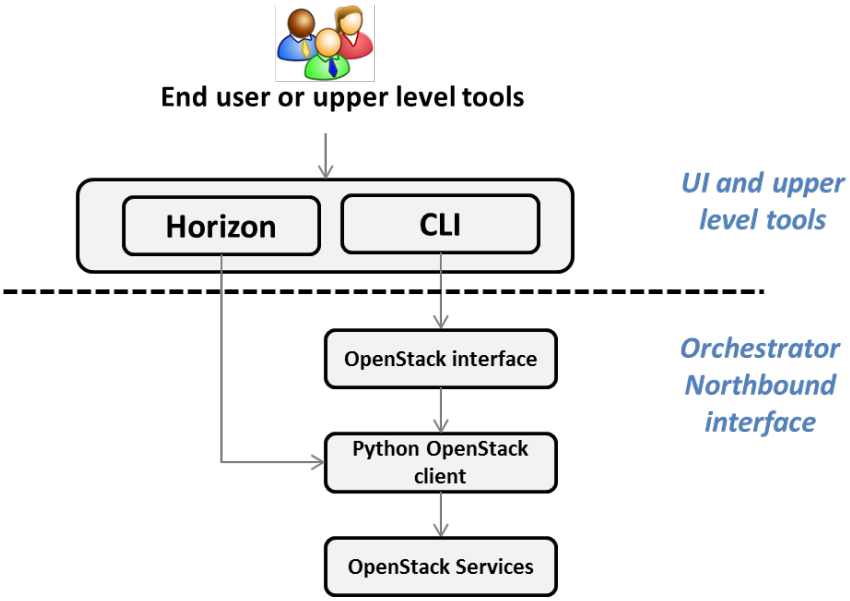

Figure 55 – Interfaces between the orchestrator and the user/application layers

7.1.1.1 Interface Definition for Provisioning the VDC Use Case

In order to request for VDC operations, the following orchestrator OpenStack services (REST API clients or CLI and shell utilities unified under the OpenStack interface) are involved. The NOVA REST API is required for VMs handling, meanwhile the Neutron plugin handles the part related to the network that has been specified in the Heat template.

Table 13 shows the involved API layer modules while requesting a VDC instance and. the required APIs extension, while Table 14 shows operations to satisfy VDC instance related operations requests.

Infrastructure control layer Module and their APIs		**Extension**	**Function Supported**
OpenStack Client interface	Python-Heatclient / Heat REST API	Template extensions to support the VDC request	GET, PUT, POST, DELETE
	Python-Novaclient / Nova REST API	No extensions are needed	GET, PUT, POST, DELETE
	Python-Neutronclient / Neutron REST API	Optical extensions needed	GET, PUT, POST, DELETE

Table 13 – High level description of the VDC interface

Operation	**URI**	**Description**
POST	Create VDC template	Creates a network and add VMs to the VDC instance.
GET	List VDC stack	Show Deployed VDC on Dashboard VDC instance information.
PUT	Update VDC template	Edit current deployed VDC instance, updating the template.
DELETE	Delete VDC ID	Deletes a specified VDC instance corresponding to a certain ID.

Table 14 – List of required APIs for requesting a VDC instance

7.1.1.2 Interface definition for provisioning the vApp use case

The operations related to the vApp use case are similar to previous VDC ones except for the Nova part, which is not required. The Horizon module needs to be integrated with Heat so that the End user or northbound application is able to specify the use case request.

Infrastructure control layer Module and their APIs		**Extension**	**Function Supported**
OpenStack Client interface	Python-Heatclient / Heat REST API	Template extensions to support the vApp request	GET, PUT, POST, DELETE
	Python-Neutronclient / Neutron REST API	Policy extensions needed	GET, PUT, POST, DELETE

Table 15 – High level description of the vApp interface

Operation	**URI**	**Description**

POST vApp	Create vApp template	Creates a vApp instance. The vApp definition is included in the template.
GET	List vApp stack	Show Deployed vApp on Dashboard vApp instance information.
PUT	Update vApp template	Edit current deployed vApp instance, updating the template.
DELETE	Delete vApp ID	Deletes a specified vApp instance corresponding to a certain ID.

Table 16 – List of required APIs for the operation of a vApp instance

The Operations and Management Use case has been designed to enable the *DCN service owner* to operate the assigned resources, and for the *DCNadmin* to operate the entire pool of network resources (see Figure 23 and Figure 24). The operational aspects of the use case instances under DCN service ownership are handled by means of previous described interfaces (GET, POST, PUT, DELETE) whereas the operation and management of the whole set of network resources is carried out directly over the infrastructure control platform, i.e. ODL (explained in section 7.2).

7.2 Southbound Interfaces

This section describes the interfaces between the COSIGN Orchestrator and the control plane, from the network point of view. The interaction between the two planes is captured in Figure 56. There are two entities in the control plane that provide services towards the orchestrator: the OVN controller and the ODL controller. Each of these entities interacts with the orchestrator through a set of interfaces for the purpose of provisioning of services which support the COSIGN use cases. These interfaces are mostly based on REST APIs. Following, a description of each of these interfaces is given, in relation to the services that are needed from the orchestrator.

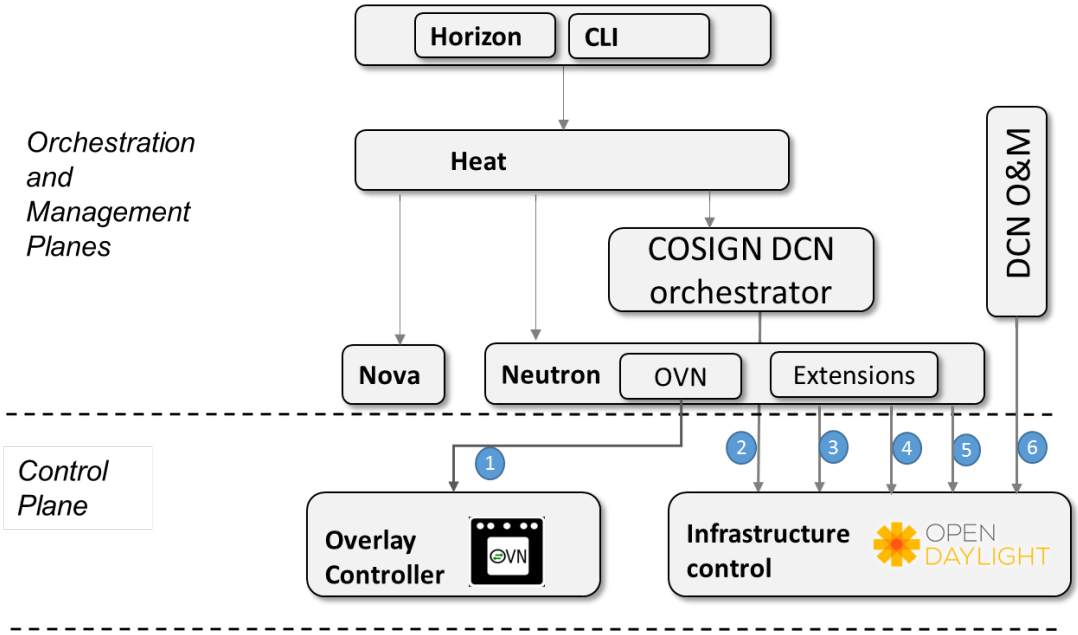

Figure 56 – Interfaces between the orchestrator and the control plane

7.2.1 Interface Definition for Provisioning of Overlay Networks

In order to provision overlay networks the orchestrator requires support from the OVN controller for the creation and managing of such overlays (interface 1 in Figure 56). Table 17 captures a description of this interface, while Table 18 gives a more detailed view of the methods needed for the interface.

The only two entities involved for this interface are the OVN Neutron plugin at the orchestrator, and the OVN controller which implements the overlay network service. The OVN controller contains a database which keeps information regarding the virtual network overlays created. In order to create a new overlay or manage an existing one, the OVN Neutron plugin performs a set of operations on the OVN controller database.

Modules in Orchestrator	**Infrastructure control layer Module and their APIs**		**Extension**	**Function Supported**
Neutron OVN plugin	OVN Controller	Overlay Service	No extensions are needed on top of the OVN functionality	GET, PUT, POST, DELETE

Table 17 – High level description of the interface

All the methods needed for managing the overlay networks are described in Table 18.

Operation	**URI**	**Description**
GET	List networks	Lists networks to which the specified tenant has access.
POST	Create network	Creates a network.
POST networks	Bulk create networks	Creates multiple networks in a single request.
GET	Show network	Shows information for a specified network.
PUT	Update network	Updates a specified network.
DELETE	Delete network	Deletes a specified network and its associated resources.

Table 18 – List of APIs for provisioning of overlay networks

7.2.2 Interfaces for Provisioning of Optical Connectivity

The current interface, used for provisioning of optical connectivity in the DCN, is exposed by the ODL controller towards the orchestrator. Optical connectivity is required as the baseline service to support other higher level services in the data centre or to support various management operations. Optical connectivity is always established by the COSIGN SDN controller, through the ODL optical drives. In some cases, the decision to create an optical circuit can be made by the SDN controller itself, e.g. as a response to congestion or a failure it detects. In other cases, this decision is made by the COSIGN Orchestrator and communicated to the controller through the API described here.

Modules in Orchestrator	**Infrastructure control layer Module and their APIs**		**Extension**	**Function Supported**
Neutron (ODL plugin)	COSIGN SDN controller (ODL)	Optical connectivity provisioning module	Configuration of optical switches along the path between the ingress and the egress endpoints.	POST, PUT, GET, DELETE
DC O&M	COSIGN SDN Controller (ODL)	Optical connectivity provisioning module		

Table 19 – High level description of the interface

Table 19 gives a high level description of the interface for provisioning of optical connectivity. On the orchestrator side, it is the Neutron ODL plugin which initiates the requests (interface 5 in Figure 56).

Additionally, requests for optical connectivity in the DCN can arrive from the DC Operations & Management (O&M) entity (interface 6). Hence there are two logical interfaces that leverage the optical connectivity provisioning service. The ODL controller contains a module for provisioning of optical connectivity which receives and fulfils the requests. A request contains the two endpoints (ingress and egress) for the connection and other attributes (e.g. QoS, bandwidth).

Table 20 contains the REST APIs for provisioning of optical connectivity. The APIs are exposed by COSIGN SDN Controller and are based on the RESTCONF protocol.

Operation	URI	Description
POST	/restconf/config/optical-provisioning-manager:connection	Creates a new optical connection
PUT	/restconf/config/optical-provisioning-manager:connection/connectionID	Modifies some parameters of an established optical connection with identifier "connectionID"
GET	/restconf/config/optical-provisioning-manager:connection/connectionID	Retrieves the details of an established optical connection with identifier "connectionID"
GET	/restconf/config/optical-provisioning-manager:connections	Lists all the established optical connections
DELETE	/restconf/config/optical-provisioning-manager:connection/connectionID	Tears-down and removes the optical connection with identifier "connectionID"

Table 20 – List of APIs for provisioning of optical connectivity

7.2.3 Interfaces to the network topology service

This interface is used by the COSIGN SDN Infrastructure Controller to provide information about the physical topology of the datacentre network, with particular reference to:

- The datacentre network nodes, including both end-points, i.e. the opto-electronic switches used to interconnect the datacentre servers, and the core optical devices.
- The connectivity between the datacentre network nodes, through links which interconnect ports on different devices. For the VDC use case, where the algorithms need to jointly allocate network and computing resource, the characteristics and capabilities of the optical links must be also exposed to the orchestrator (e.g. wavelength availability).
- The hosts connected to the network end-points.

Modules in Orchestrator	Infrastructure control layer Module and their APIs		Extension	Function Supported
Neutron	COSIGN SDN controller	network-topology	none	GET
COSIGN DCN Orchestrator (VDC algorithms)	COSIGN SDN controller	network-topology	Optical characteristics and capabilities	GET

Table 21: Interface between orchestrator and SDN controller for network topology

The service is consumed by two entities at the orchestrator level (two logical interfaces exploiting this service), as shown in Table 21:

- Neutron, to collect information about DCN endpoints and interconnected hosts (interface 3);
- The VDC algorithms within the COSIGN DCN orchestrator, to collect information about the entire DCN topology, including it optical characteristics and capabilities (interface 2).

The topology service can be accessed in read-only mode, via REST APIs, as described in *Table 22*.

Operation	URI	Description
GET	GET/restconf/operational/network-topology:network-topology	Shows the physical topology of the datacentre network, including network nodes, their ports (called termination points), hosts (including MAC and IP addresses), and links (defined through source and destination nodes and ports).

Table 22 – List of APIs for DCN topology service

7.2.4 Interface for provisioning of virtual optical slices

This interface is used by the COSIGN orchestrator to request the creation of multi-tenant, isolated virtual network slices including virtual optical resources (interface 4 in Figure 56). This service is implemented by the COSIGN SDN infrastructure controller through the OpenDayLight Virtual Tenant Network (VTN) application, properly extended to support virtual optical resources. This interface has been documented in D3.2, section 3.2.2, and it does not require major modifications.

In particular, a slice is characterized through a topology of virtual nodes and links, with optional parameters for QoS guarantees, monitoring options and delay constraints. The brief summary of the REST APIs messages is reported in *Table 23*. The detailed parameters are available in D3.2. The service is consumed by Neutron (see *Table 24*).

Modules in Orchestrator	**Infrastructure control layer Module and their APIs**		**Extension**	**Function Supported**
Neutron	COSIGN SDN controller	VTN application	Specification of virtual optical resources	GET, POST, PUT, DELETE

Table 23 – Interface between orchestrator and SDN controller for virtual optical slices

Operation	URI	Description
POST	/restconf/config/virtual-infrastructure-manager:slice	Creates a new virtual optical slice.
GET	/restconf/config/virtual-infrastructure-manager:slice/<TenantID>/<SliceID>	Shows the information associated to the virtual optical slice.
GET	/restconf/config/virtual-infrastructure-manager:slice/<TenantID>	Shows the information associated to the virtual optical slices owned by a given tenant
PUT	/restconf/config/virtual-infrastructure-manager:slice/<TenantID>/<SliceID>	Updates or modifies the configuration of the specified virtual optical slice.

DELETE	/restconf/config/virtual-infrastructure-manager:slice/<TenantID>/<SliceID>	Removes the virtual optical slice and releases the occupied resources.

Table 24 – List of APIs for provisioning of virtual optical slices

7.2.5 Interfaces for Supporting Virtual Cloud Applications (vApp)

As described in Section 4.2.1, the vApp use case involves defining application blueprints in high level terms and deploying multi-component cloud applications interconnected as prescribed by the blueprint. In many existing solutions, such a use case is approached in completely infrastructure agnostic terms, employing dynamically created overlays on top of statically provisioned underlays (the DCN infrastructure). Although satisfying the functional requirements as well as the requirements for multi-tenancy, scalability, and isolation, these solutions typically cannot be dynamic in properties depending on physical infrastructure. For example, without relying on continuous feedback between the overlay and the underlay, it is impossible to differentiate between the alternative paths the overlay traffic can be tunneled over in the underlay infrastructure. In COSIGN, we aim to address this limitation and to create such a feedback loop whereby dynamic forwarding decisions can be made in the underlay depending on the requirements specified in the overlay. For this, we devised the combined approach of the Infrastructure DCN Controller and the Overlay DCN Controller, governed under the same Orchestration plane, as presented from Figure 6 onwards.

In addition to combining the two network forwarding planes – the underlay and the overlay, we aim to surface dynamic capabilities of the COSIGN undelay (enabled by the optical technologies) to the virtual cloud application interconnected by overlay tunnels. This will allow, for example, dynamic creation of optical circuits to support service levels defined by the vApp blueprint.

This extension for the vApp Use Case is presented in Figure 57, showing how to employ dynamic circuit establishment according to the traffic in the data-plane. To that end, applications' flows are tagged by the OVSs with DSCP markers. These DSCP markers identify both the QoS of the flow and the type of the flow according to a predefined classification, e.g., mice, elephant, short, long, etc. The flows are monitored at the Opto-Electronic switches which are connected directly to the optical switch, by an SDN Controller plane component designated here Physical Observer. This component is owned and specified in WP3 [D3.2]. Exact interfaces and their co-existence with the rest of interfaces presented in this section will be further refined as T4.3 progresses towards the full integration between the COSIGN Orchestrator and the COSIGN SDN Controller.

In what follows we describe the conceptual approach and outline the major interfaces without fully specifying the details, using Figure 57 for illustration.

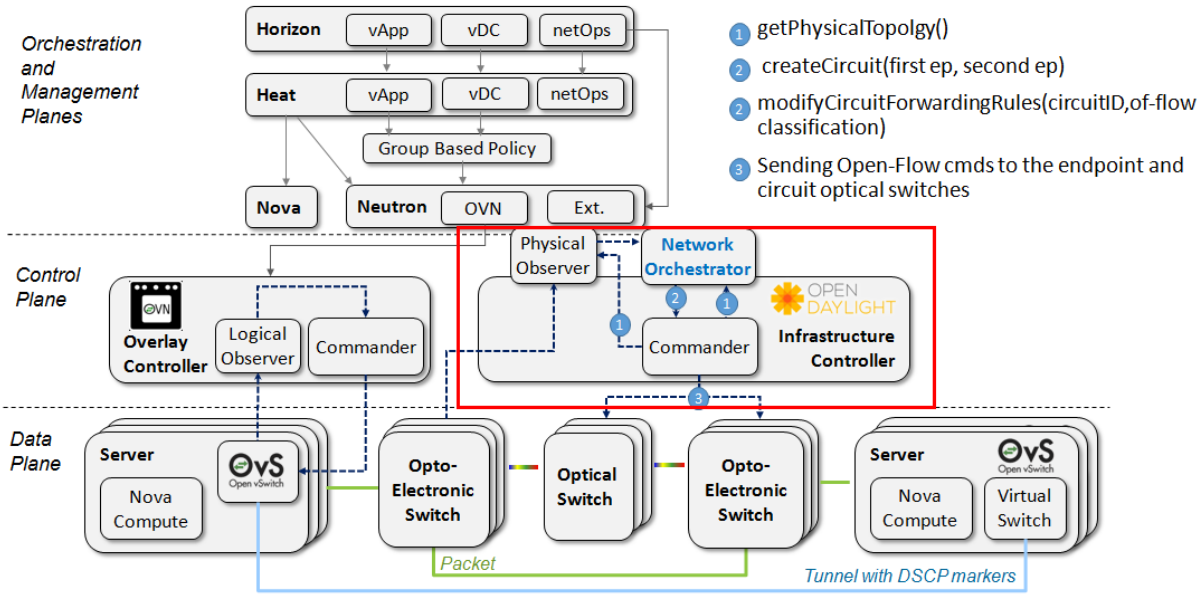

Figure 57 - APIs flow between ODL and network orchestrator

Step 1 in Figure 57 shows how the Physical Observer and the Network Orchestrator obtain the physical topology information from the Infrastructure Commander (ODL). The Physical Observer can identify and inform the Network Orchestrator about the need to dynamically create an optical circuit over the optical switch for sending aggregated mice flows, elephant flow, etc. The Network Orchestrator decides which circuit to establish depending on the current optical circuits, traffic demands, QoS and more. Then, in Step 2 of Figure 57, the Network Orchestrator asks the Infrastructure Controller (ODL) to create a new circuit. Once, it gets ACK and connection ID from the Infrastructure Controller, the Network Orchestrator can modify, through the Infrastructure Controller's API, forwarding tables of the corresponding opto-electronic switches to forward the desired flows through the newly established circuit, as shown in Step 3 of Figure 57. At any time, the Network Orchestrator can use currently existing circuits for sending additional flows by modifying the forwarding rules for a given circuit over the corresponding opto-electronic switches, also initiating the redirection through the Infrastructure Controller's API.

8 Conclusions

This deliverable follows up on the previous one of this Work Package [D4.1] by providing a more complete specification of the COSIGN Orchestrator blueprint outlined in D.4.1.

The base orchestration platform identified in D4.1, OpenStack, is analysed here in high level of details showing what OpenStack components will be included as part of the COSIGN Orchestrator, which shall be extended, and what additional components must be created. In addition, COSIGN specific data models, identified in D4.1 and refined here, are mapped to the data models of OpenStack components. Here as well, we point out what entities of significance to COSIGN can be mapped to existing OpenStack object types, which need to be extended, e.g. by adding attributes of methods, and which ones need to be created afresh. In some cases, where multiple possibilities exists, we point out the options to be considered throughout the prototyping and integration efforts, leaving the decision for later when some experience is gained. The WP4 team will continue to monitor the OpenStack developments over the remaining period of the project duration, in order to best use possible options existing in the core OpenStack community and in surrounding open source efforts.

In addition to specifying the OpenStack interlocks, the layered architecture of the COSIGN architecture is refined, covering its data models, operational flows, and interfaces. The architecture specification is continuously validated against the three major groups of COSIGN use cases – the Virtual Data Centre (VDC), the Virtual Application Cloud (vApp), and the DC Operations and Management (O&M).

Interlocks with the COSIGN SDN Controller (WP3) layer are specified and will be further refined as both the layers, the Orchestrator, and the SDN Controller, get fleshed out as working code and the integration process evolves as part of T4.3.

Interlocks with the COSIGN demonstrator (WP5) as just outlined and will be articulated when concrete demonstration scenarios are picked up by WP5, also depending on the DCN topologies chosen for the demonstrator (WP5 and WP2).

REFERENCES

[1] Gartner Technical Professional Advice: In-Depth Assessment of IBM SoftLayer, an IBM company. March 2015. (Private access only).
[2] Forrester report. The State Of Cloud Platform Standards: Q2 2015. May, 2015.
[3] OpenStack. Open source software for creating private and public clouds. https://www.openstack.org/.
[4] Infonetics Research. Cloud Service Providers Reveal Preferences for Data Center Technologies and Vendors. June 2015.
[5] OPNFV. Open Platform for Network Function Virtualization. Link: https://www.opnfv.org/software.
[6] OpenDaylight, A Linux Foundation Collaborative Project. http://www.opendaylight.org/
[7] OpenStack Neutron. https://wiki.openstack.org/wiki/Neutron.
[8] OpenStack Havana Release. https://www.openstack.org/software/havana/.
[9] OpenStack Heat. https://wiki.openstack.org/wiki/Heat.
[10] Openstack meetup heat-nov23, slides by Swapnil Kulkarni. Link.
[11] AWS CloudFormation. Documentation link.
[12] Heat Orchestration Template (HOT) Guide. Link.
[13] Heat Orchestration Template (HOT) specification. Link.
[14] Heat Orchestration Template reference. http://docs.openstack.org/hot-reference/content/
[15] Heat. Orchestrating Multiple Cloud applications. Slides by Dina Belova: Link.
[16] OpenStack Senlin. https://wiki.openstack.org/wiki/Senlin.
[17] OpenStack Ceilometer. https://wiki.openstack.org/wiki/Ceilometer.
[18] Ceilometer developer documentation. http://docs.openstack.org/developer/ceilometer/index.html
[19] Measurements page of Ceilometer in the Cloud Administrator Guide.
[20] OpnContrail, An open-source network virtualization platform for the cloud. Link.
[21] Riemann. A Network Monitoring System. http://riemann.io/.
[22] OpenDaylight Host Tracker REST API Client. Link to module documentation
[23] Croup Based Policy for OpenStack Whitepaper
[24] Group-Based Policy for OpenStack. Link.
[25] Group Based Policy (GBP) OpenDaylight project. Link.
[26] GroupBasedPolicy in OpenStack wiki. https://wiki.openstack.org/wiki/GroupBasedPolicy
[27] Group Based Policy (GBP) Documentation. Link.
[28] DevStack - an OpenStack Community Production. http://docs.openstack.org/developer/devstack/
[29] KVM. Kernel Virtual Machine. http://www.linux-kvm.org/page/Main_Page
[30] Linux Bridge. http://www.linuxfoundation.org/collaborate/workgroups/networking/bridge
[31] Open vSwitch (OVS). Production Quality, Multilayer Open Virtual Switch. http://openvswitch.org/
[32] Mininet. An Instant Virtual Network on your Laptop (or other PC). http://mininet.org/
[33] OVN, Bringing Native Virtual Networking to OVS. Link.
[34] INTEROUTE VIRTUAL DATA CENTRE Application Programming Interface. Link.
[35] Amazon Virtual Private Cloud Documentation. http://aws.amazon.com/documentation/vpc/
[36] Network Technologies for Next-generation Data Centers. Rami Cohen, IBM Haifa Research Lab. September 2013. ECOC2013 Workshop on SDM: How to migrate from point-to-point transmission to full optical networking? Link.
[37] White Paper: Simplifying cloud management and data center automation. Link.
[38] What Is Cisco ACI? SDxCentral summary and resources. Link.
[39] OpenStack API Reference. Identity API v3. http://developer.openstack.org/api-ref-identity-v3.html
[40] OpenStack developer documentation. Scope of the Nova project. Link.
[41] OpenStack Governance – Nova Mission Statement. Link.
[42] OpenStack API Reference. Compute API v2.1. Link.
[43] OpenStack API Reference. Telemetry API v2. http://developer.openstack.org/api-ref-telemetry-v2.html
[44] OpenStack API Reference. Orchestration API v1. Link.
[45] Orchestration for Group Based Policy Resources. Link.
[46] OpenStack Python API bindings. Link.