Complete Dynamic Multi-cloud Application Management

Project no. 644925

Innovation Action

Co-funded by the Horizon 2020 Framework Programme of the European Union

Call identifier: H2020-ICT-2014-1

Topic: ICT-07-2014 – Advanced Cloud Infrastructures and Services

Start date of project: January 1st, 2015 (36 months duration)

Deliverable D5.3

Cloud federation Network service delivery and integration

Due date:	31/12/2017
Submission date:	29/01/02018
Deliverable leader:	i2CAT
Editors list:	Eduard Escalona (i2CAT), Albert Viñés (i2CAT), Josep Pons (i2CAT)

Dissemination Level

☒	PU:	Public
☐	PP:	Restricted to other programme participants (including the Commission Services)
☐	RE:	Restricted to a group specified by the consortium (including the Commission
☐	CO:	Confidential, only for members of the consortium (including the Commission Services)

List of Contributors

Participant	Short Name	Contributor
Interoute S.P.A.	IRT	Domenico Gallico
Sixsq SARL	SIXSQ	
QSC AG	QSC	
Technische Universitaet Berlin	TUB	Dirk Thatmann
Fundacio Privada I2CAT, Internet I Innovacio Digital A Catalunya	I2CAT	Josep Pons, Albert Viñés, Eduard Escalona
Universiteit Van Amsterdam	UVA	
Centre National De La Recherche Scientifique	CNRS	

Executive Summary

The objective of this document is to provide the final specification of the OpenNaaS CNSMO component as an update to the previous deliverables. It includes the complete set of newly developed services and a list of improvements to already reported services and their integration with SlipStream and the security architecture as part of the final project design. It is also the aim to summarize the list of benefits, improvements and added value that the CNSMO component may bring to the CYCLONE use cases while deploying their applications together with this solution and to provide a brief analysis of competitor projects.

Table of Contents

Figures Summary

1. Introduction

Cyclone Networking Services Manager and Orchestrator (CNSMO) is a refactored and lightweight version of the OpenNaaS platform, designed and developed for the CYCLONE project with the objective to provide multi-cloud networking services to Application Service Providers (ASPs). Previous WP5 deliverables detailed the initial design of CNSMO, with its architecture and functionalities as well as the identified limitations and the refactoring process of OpenNaaS to fulfil the project objectives realistically and integrating cloud networking seamlessly into the application deployment process.

As already described in deliverable D5.2 [CYCLONE-D5.2], some of the initial assumptions made in CYCLONE WP5 have been proven not aligned with the expectations and requirements of application providers, such as the control of network infrastructure owned by cloud providers. CYCLONE constitutes a set of tools aimed at ASPs, with very limited or no access to the physical infrastructure or its management systems. Consequently, CYCLONE has designed a purely overlay solution that is 100% multi-cloud and multi-provider. With CNSMO, application providers are able to configure and control network services in a complete flexible and scalable way while presenting them in a user-friendly control dashboard. To these end, CNSMO has adopted the Software Defined Networking (SDN) and Network Function Virtualization (NFV) concepts in an overlay solution, keeping full control of a virtual infrastructure. This is completely aligned with the initial objective of providing cloud networking to federated multi-cloud environments for complex and distributed application deployments. Thus, the final design of OpenNaaS CNSMO is thought to be used directly by application providers without requiring any access over physical resources owned by the cloud or telecom service providers but giving a similar degree of control and flexibility as if the resources in control would be physical and not virtual.

The main limitation of this solution is that the base performance and physical connectivity depends on the infrastructure service provider (i.e. cloud provider, ISP, etc.). This means that while application providers will have full control over what happens in the virtual network, it works under the assumption that underneath service providers are satisfying agreed SLA guarantees.

Since D5.2, WP5 has focused on the enhancement and extension of CNSMO functionalities, including:

- Optimisation of service deployment
- enhancement of the VPN service
- Implementation of the SDN component
- Implementation of the DNS service
- Implementation of monitoring features
- Implementation of CNSMO service API
- Implementation of CNSMO Graphical User Interface (GUI)
- Extensions for full multi-cloud support
- Integration of CNSMO with other CYCLONE components

This deliverable contains a description of these enhancements, its architecture design and implementation details to overcome the limitations detected in the first reporting period. We have also identified future improvements of CNSMO for implementation roadmap as well as for exploitation.

2. OpenNaaS-CNSMO final specification

2.1. CNSMO cloud networking services final specification

As presented in D5.2 [CYCLONE-D5.2], CNSMO final design implements a lightweight distributed platform that consists in a central server and multiple network services that can run centralised locally or distributed in remote Clouds as part of the application deployment.

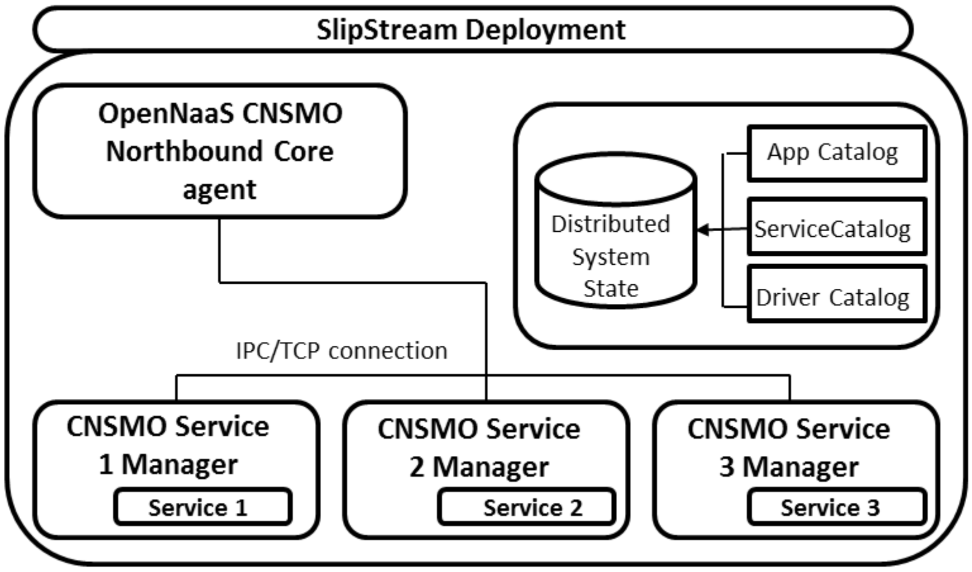


Figure 1: CNSMO architecture

CNSMO's architecture is designed to be modular, flexible, extensible and scalable, so new network services can be added easily and integrated in the overall platform with little effort. This is thanks to the fact that CNSMO is based on network virtualization to implement its services, that can be deployed as containers anywhere in the cloud.

2.1.1. SDN-based cloud networking control and management

Software-Defined Networking (SDN) is an emerging network paradigm. It is dynamic, manageable, cost-effective, and adaptable, making it ideal for the high-bandwidth demands and dynamic nature of today's applications. An SDN architecture decouples the network control and forwarding functions enabling the network control to become directly programmable and the underlying infrastructure to be abstracted for applications and network services. OpenFlow [OpenFlow] is the most accepted protocol used as a mechanism to exchange messages between the SDN Controller and the network devices.

CYCLONE has taken advantage of the SDN capabilities in a unique way. Instead of managing real networking devices, CNSMO uses it to manage and control the inner networking of cloud applications. The large number of features provided by OpenFlow allows the application provider to take any decision on the traffic according to inspected IP packet fields (i.e., when packets match a certain condition, the traffic can be forwarded, dropped, redirected and shaped). Furthermore, it supports multiple tunnelling protocols (GRE, VxLAN, STT, and Geneve, with IPsec support).

In order to provide SDN capabilities to instances that have been deployed with CNSMO features, a new set of services called SDN have been added to CNSMO. According to that, CNSMO-SDN can be installed and deployed automatically with CNSMO if the administrator specifies it in the list of services required for a deployment. In that case, the OpenDaylight (ODL) Controller Software [ODL] is installed and configured in CNSMO Server instance after configuring the CNSMO-VPN service. In the client application instances (where CNSMO Agent is instantiated) the Open vSwitch Bridge software [OVS], is installed and configured in order to allow the server to control their input and output flows. After this step, CNSMO-SDN services can be used to control traffic over the client machines. The concrete architecture and modules implemented and deployed are explained in section 2.2.2.

The implementation done in CYCLONE integrates the SDN capabilities provided by OpenDaylight in CNSMO services as well as some network services required by the selected use cases that are, at same time, examples to see the potential of the CNSMO solution. CNSMO-SDN contains a set of methods that allow retrieving the configuration and status of SDN flows, configuring, configure new flows for blocking traffic targeted to a specific destination IP and TCP Port, deleting and modifying configured flows, as well as services to retrieve information about the number of packets and bytes blocked at some point for a configured filter. Currently, TCP Port Traffic Filters can be configured with CNSMO. All these services details are explained in section 2.2.2.3 . The service for retrieving information (monitoring) about packets blocked by a specific applied flow is mentioned also in the next section.

2.1.2. Monitoring

Control and management requires knowledge about what we are controlling. Thus, if we want to enable and disable capabilities on deployed application instances as well as acting over them or controlling networking elements like traffic, QoS, etc., we must know the status of the different services we have deployed as well as the effects and results achieved at each moment by their application. Therefore, network monitoring is key aspect for CYCLONE and two types of functionalities have been implemented in CNSMO for this purpose.

The first type of functionality is the capability to provide real time information about the status of enabled services. For each set of services: SDN, VPN, FW, DNS, etc. there is a group of calls with the objective of providing status information and allow the administrators to know which are the services enabled, and which problem may prevent them from working correctly. Administrators can log into the CNSMO GUI (see section 2.1.4) and easily be informed about the correctly enabled services over their machines, as well as any possible problem occurred causing the termination of any of them. This information is visually presented with colour light semaphores making very easy the work of administrators.

On the other hand, the other type of implemented monitoring functionality, is related to the capability of obtaining information about the number of packets that are filtered by means of configured SDN rules. CNSMO-SDN allows retrieving the number of packets that have been filtered by a specific flow (which in fact corresponds to an applied filter) from its application. As it is explained in section 2.1.4, the CNSMO GUI uses this service to dynamically show a time series graph with the evolution of the number of filtered packets by each one of the configured filters. In CYCLONE, this functionality has been implemented regarding flow monitoring capabilities, but in the future a wide set of possibilities on this area can be added with little implementation effort, thanks to the flexibility enabled by SDN technologies.

2.1.3. DNS service

CNSMO-DNS is the CNSMO component that deploys and enables a Domain Name System server as part of the application deployment process. It acts as a local DNS in the application's network to resolve host names into IP addresses and vice-versa. New entries in the DNS server can be added dynamically and is automatically integrated with the CNSMO-VPN component in order to configure all VPN clients with the DNS server address.

This service is particularly useful as a decentralized naming system for deploying application components in clouds that do not have internal DNS enabled for private networks.

2.1.4. CNSMO GUI and API

In order to allow the Application administrators (CYCLONE users which deploy a distributed complex application) to easily use all the functionalities and services offered by CNSMO, an API and a Graphical User Interface have been implemented. These components are installed and deployed automatically inside the CNSMO Server instance at deployment time, before installing other services like CNSMO-VPN, CNSMO-SDN, etc. The specification of the CNSMO API can be found in the Annex 1 of this deliverable.

According to that, once the CNSMO-GUI and CNSMO-API have been installed, the administrator owner of the Application Deployment can start using them.

The API is used internally by the GUI which invokes the different API requests in order to get or set the information corresponding to each one of the offered functionalities. To make requests to this API, the privileged users can perform requests from any HTTP REST client (like Postman). The base URL is:

http://<ip of_CNSMO_Server_instance>:8081/api/v1/services/

In order to use the different methods offered by the API, the first request that must be done is the <API_BASE>/*Authenticate* POST request. This request must be invoked providing as parameters the *username* and *password*. As username we have to use the one specified as output parameter in the SlipStream recipe. As password we have to use the randomly generated password that can be visualised also in the SlipStream portal once the CNSMO-GUI installation has finished (first step of the overall deployment of the CNSMO Server).

The list of methods implemented up to now composing the API are the following ones:

Firewall

- POST ' */fw/rules*' → add new CNSMO-FW rule
- GET ' */fw/rules*' → get configured CNSMO-FW rules

Node

- GET *'/sdn/nodes*' → get list of instances composing the deployment as well as their vpn and SDN details (vpn IP address, instance ID , configured services, etc.
- GET *'/sdn/flows*' → get the list of flows (all instances configured SDN filters).

SDN Filters

- GET ' */sdn/nodes/:instanceId/blockedTcpPorts*' → get the list of blocked <destinationIPAddress:TPC ports> for the instance specified as parameter URL (instanceId).
- PUT *'/sdn/blockbyport*' → add new SDN filter. As parameter a JSON is passed containing:
    - Tcp-destination-port
    - Ip4-destination
    - ssinstanceid

It returns the id of the created flow.

- DELETE *'/sdn/nodes/:instanceId/flows/:flowId'* → delete a filter rule (the one represented by flowId passed in the URL parameters) of the instanceId (passed as parameter)

- GET *'/sdn/nodes/:instanceId/flows/:flowId'* → get the information related to an applied filter rule (identified by a *flowId* specified in the URL). This command is used for monitoring purposes. Up to now, it returns the number of packets filtered by this filter since its creation/application.

DNS

- GET *' /dns/records '* → get the list of DNS records configured in the DNS Server (DNS entries indicating the IP address which can be resolved by DNS server when a client instance look for that name).
- POST *'/dns/record'* → add to the list of records a new DNS record (managed by DNS server)

VPN Certs & Config

- POST *'/certs/clients/:name'* → Generates VPN configuration and certificates compatible with the current deployed VPN. With that certificates and configuration (which can be obtained with following methods) an external machine can be connected to the VPN of the current deployment by means of an open-vpn compatible client. *name* parameter must be included in the URL in order to identify the new machine we are going to connect (this should be a non-repeated one).
- GET '/certs/clients/:name/key'
- GET '/certs/clients/:name/cert'
- GET '/certs/clients/:name/config'
- GET '/certs/clients/:name/ca'

In order to easily perform configurations in an intuitive and visual way, the Admin (instead of invoking directly the API by means of a HTTP REST client) can log in to the CNSMO-GUI using the corresponding deployment credentials (the same used for the AUTH method of the CNSMO-API).

The Login page is shown when the user is not already logged in or his token hA expired.

In this page, a simple form with Username and Password inputs is shown as can be seen in the following figure.

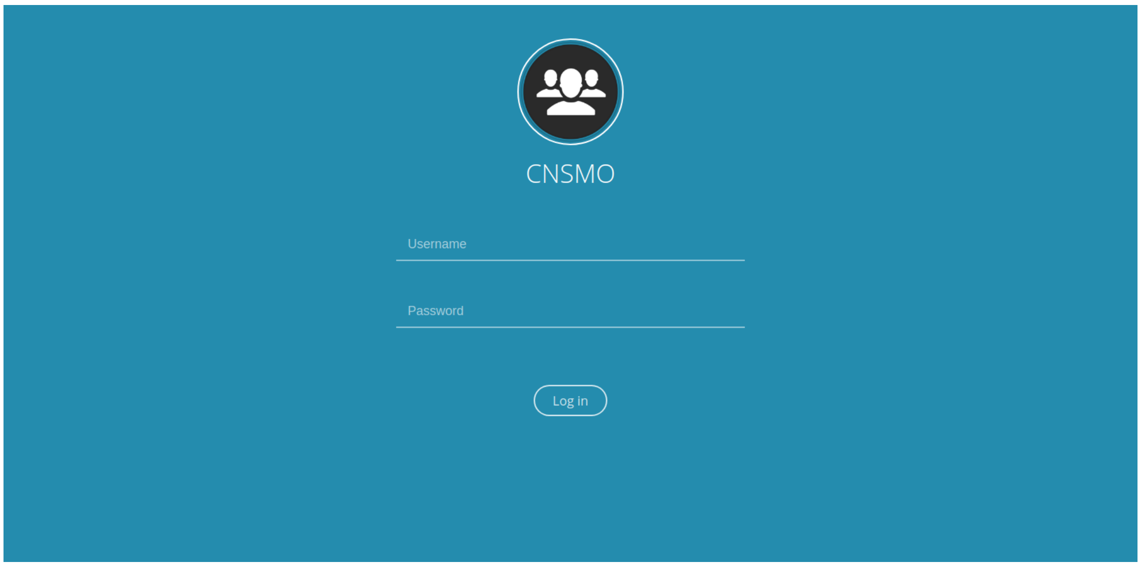


Figure 2: CNSMO Dashboard - Login Screen

If something goes wrong or credentials are incorrect a notification box is shown with the error message response received by CNSMO API. The following image depicts it.

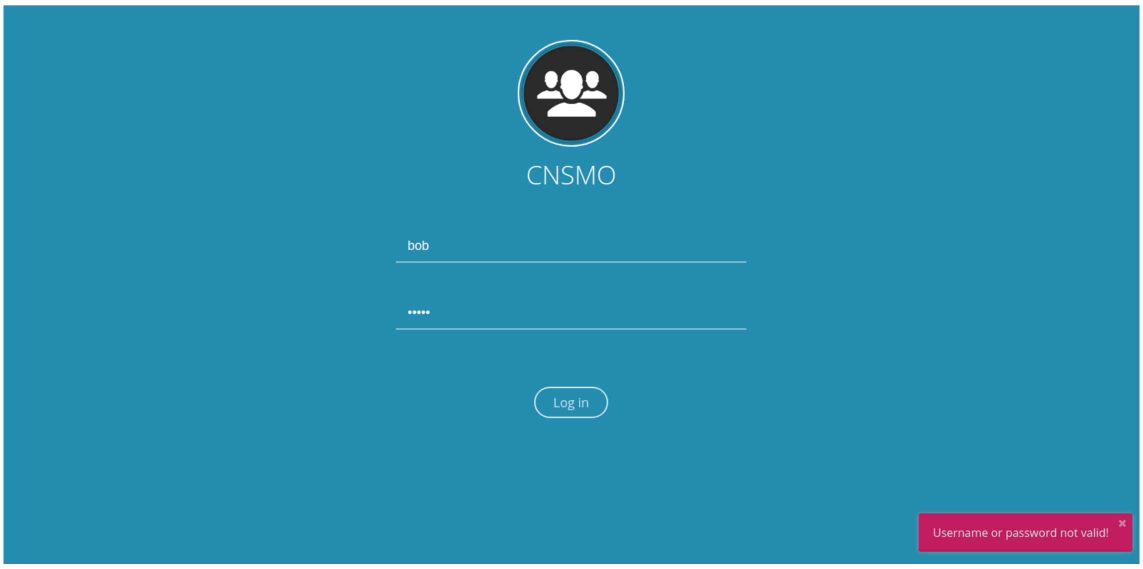


Figure 3: CNSMO Dashboard - Login error

After a successful login, the dashboard page is shown. This page is structured with a fixed toolbar at the top with the title of the page on the left side and the profile clickable badge on the right. The sidebar is on the left of the layout and presents all the categories of implemented features. The content of the layout changes depending on the route, it is situated next to the sidebar and it is scrollable. In next sections each one of the different tab are explained. Each tab corresponds to a different group of high level functionalities.

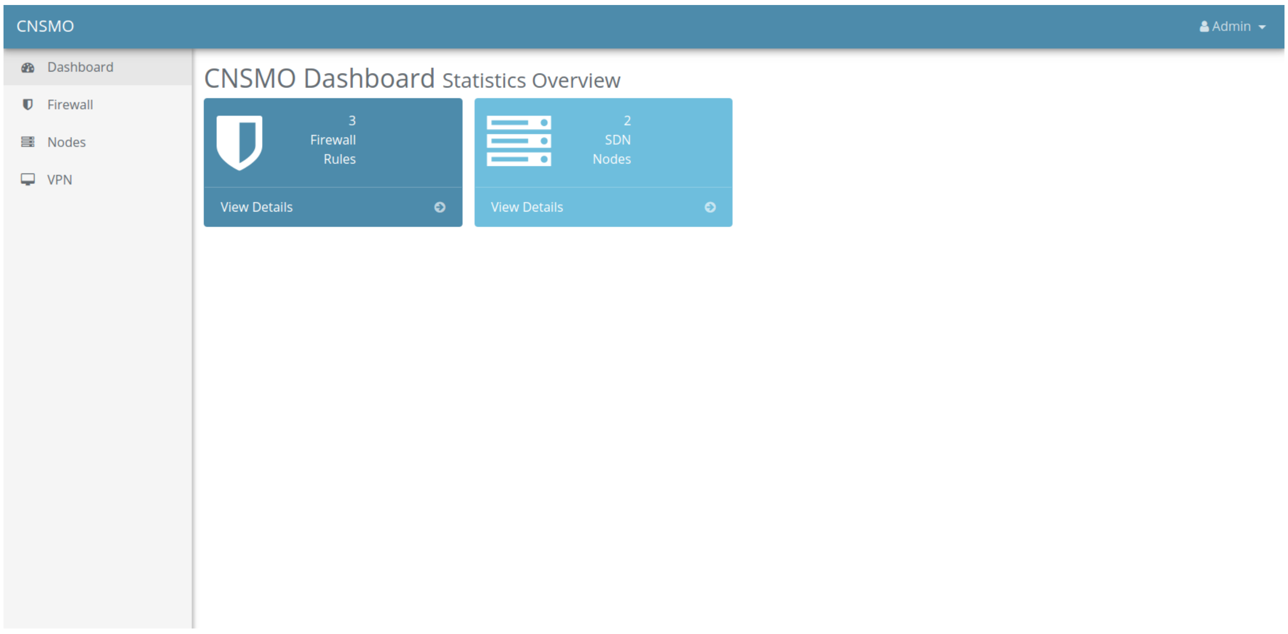


Figure 4: CNSMO Dashboard - Main Screen

The following image appears when clicking on profile badge, showing a Log Out button.

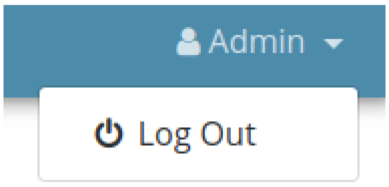
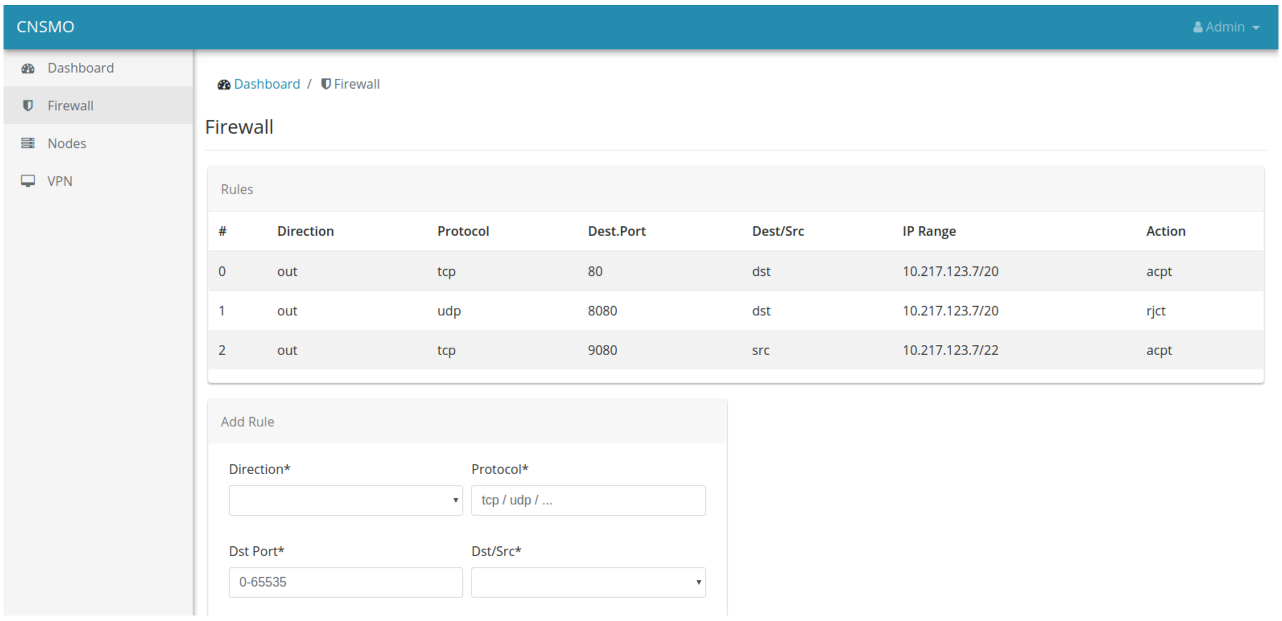


Figure 5: CNSMO Dashboard - Log Out button

2.1.4.1. Firewall

This view allows the management of firewall rules using the CNSMO-FW service. This set of services uses an IP-tables implementation for creating firewall rules as described in D5.2 and D5.1.



Rules

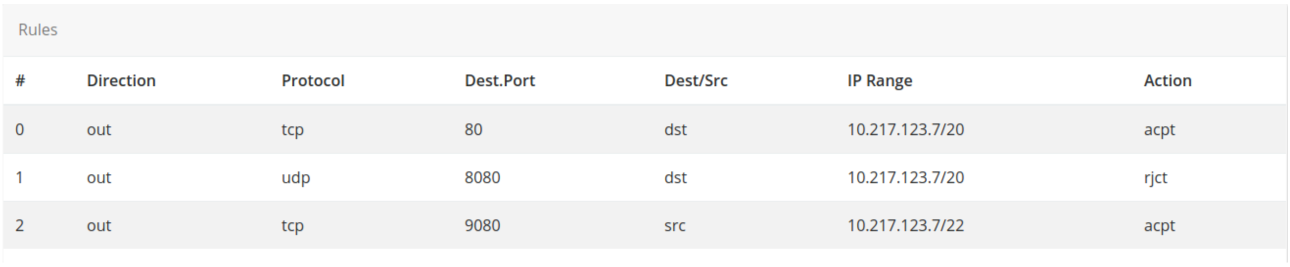
#	Direction	Protocol	Dest.Port	Dest/Src	IP Range	Action
0	out	tcp	80	dst	10.217.123.7/20	acpt
1	out	udp	8080	dst	10.217.123.7/20	rjct
2	out	tcp	9080	src	10.217.123.7/22	acpt

Add Rule

Direction*	Protocol*
	tcp / udp / ...
Dst Port*	Dst/Src*
0-65535	

Figure 6: CNSMO Dashboard - Firewall Tab

The first card box is dedicated to the list of rules that are already present in the firewall.

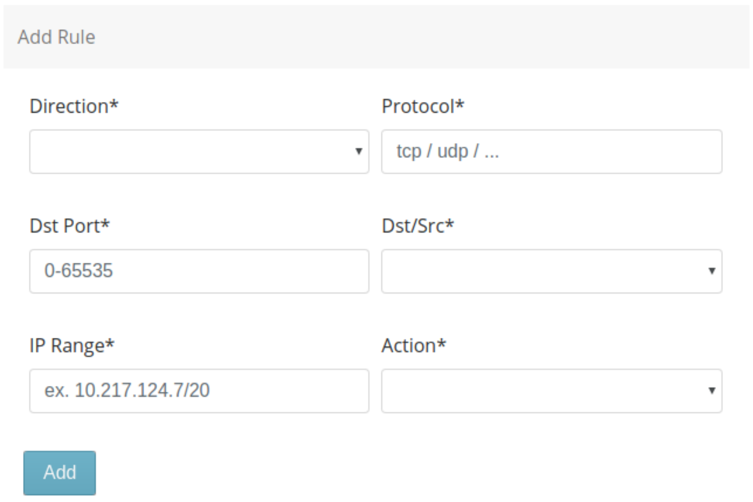


Rules

#	Direction	Protocol	Dest.Port	Dest/Src	IP Range	Action
0	out	tcp	80	dst	10.217.123.7/20	acpt
1	out	udp	8080	dst	10.217.123.7/20	rjct
2	out	tcp	9080	src	10.217.123.7/22	acpt

Figure 7: CNSMO Dashboard - Configured Rules in Firewall

This card box of Firewall allows users to add a new rule in an ACL.

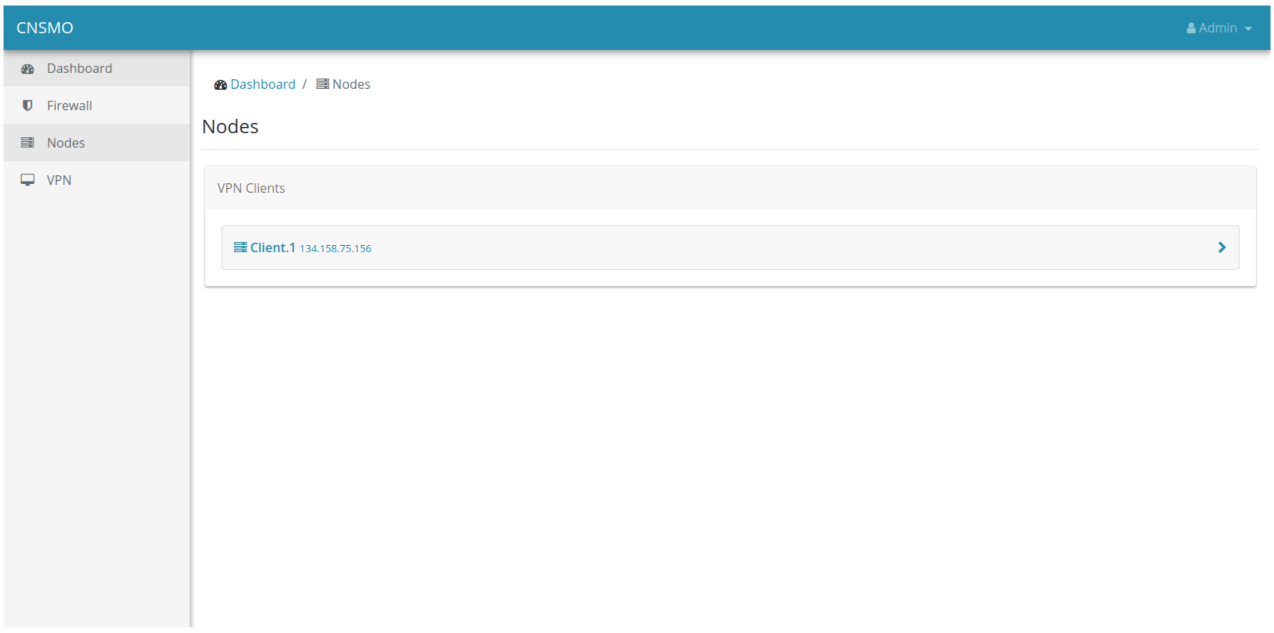

Add Rule

Direction*	Protocol*
	tcp / udp / ...
Dst Port*	Dst/Src*
0-65535	
IP Range*	Action*
ex. 10.217.124.7/20	

Figure 8: CNSMO Dashboard - Adding new FW rule

2.1.4.2. Nodes

This view allows the management of instances deployed in the Application deployment. Mainly this tab is enabled to display an overall view of all the client instances composing the VPN and configure and monitor filters using SDN technology.

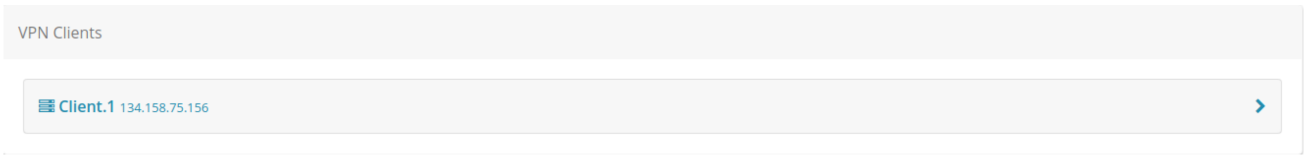


VPN Clients

Client.1 134.158.75.156

Figure 9: CNSMO Dashboard - Nodes tab (list of Instances)

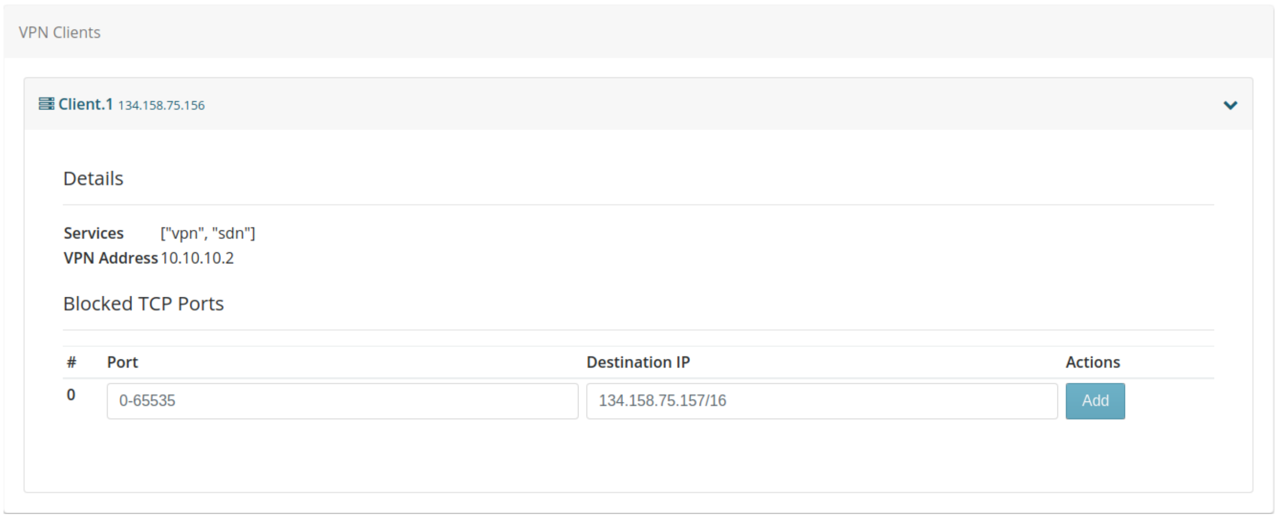
This box shows the list of present client instances in the VPN configured for this deployment.


VPN Clients

Client.1 134.158.75.156

Figure 10: CNSMO Dashboard - One instance in the Instance's List: Client.1

Clicking on the client, the user can see its details: enabled services, VPN address and list of applied FW filters.


VPN Clients

Client.1 134.158.75.156

Details

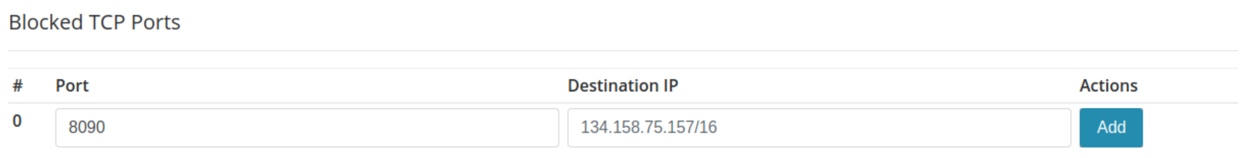
Services	["vpn", "sdn"]
VPN Address	10.10.10.2

Blocked TCP Ports

#	Port	Destination IP	Actions
0	0-65535	134.158.75.157/16	Add

Figure 11: CNSMO Dashboard - Client.1 instance details

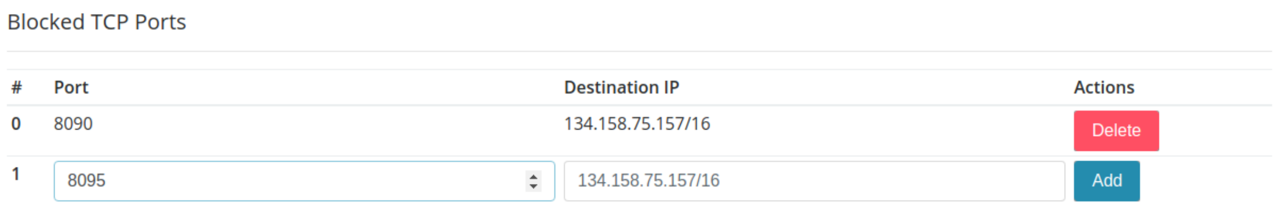
User can also add a new filter for blocking incoming packets to concrete TCP ports by specifying the number of port and the *destination IP / mask* where he wants to apply the filter. The filter is applied to a specific instance, which in fact has a public address, but specifying destination IP makes sense for example in the case the user wants to block packets with private VPN IP as destination.


Blocked TCP Ports

#	Port	Destination IP	Actions
0	8090	134.158.75.157/16	Add

Figure 12: CNSMO Dashboard - Adding new blocking rule (by TCP Port – Destination IP)

It is also possible to delete an entry from the list of blocked TCP ports-Destination IP (disable filter).


Blocked TCP Ports

#	Port	Destination IP	Actions
0	8090	134.158.75.157/16	Delete
1	8095	134.158.75.157/16	Add

Figure 13: CNSMO Dashboard - Deleting blocking rule

Monitoring packets blocked in real time by the configured filters is possible clicking on monitor buttons that will appear on the right of each blocked Port. Clicking on that button a TimeLine Graph appears showing the total number of blocked packets by that filter since the moment of its addition. This graph is constantly refreshed every 5 seconds.

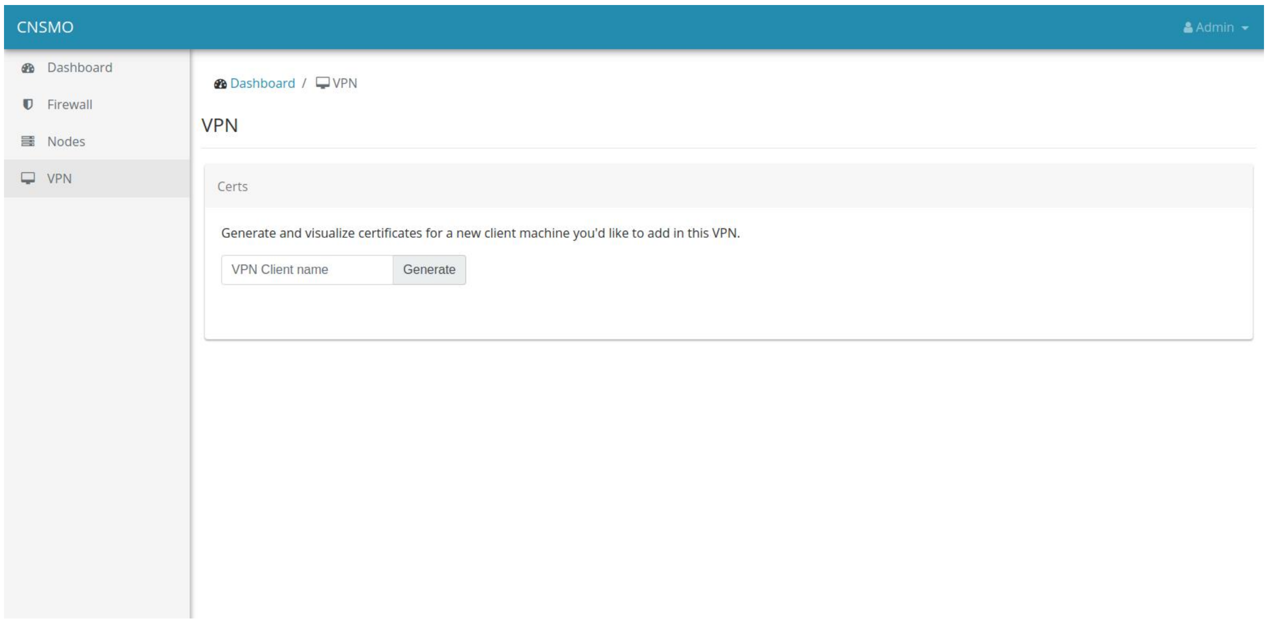
2.1.4.3. DNS

The DNS tab shows in the top part the list of DNS records which the DNS server will use for translating the DNS requests from clients. Each record is used for translating a name of a machine to an IP Address. According to that all the client machines with CNSMO-DNS enabled will be capable of using the names existing in DNS records instead of the specific IPs. These names will be resolved by the DNS server resolving the corresponding IPs which the clients are trying to resolve.

On the bottom half part of the page, the Admin can add new records (name – corresponding IP address) which will be added to the DNS Server Record list.

2.1.4.4. VPN Certificates and Configuration

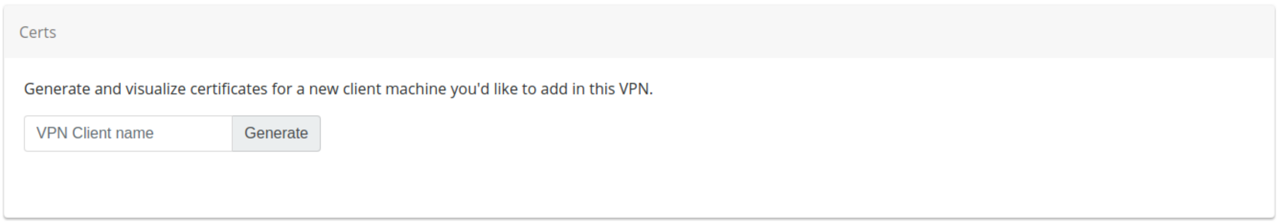
This layout allows administrators to generate and retrieve the required certificates and configuration files in order to add manually another client to the VPN configured on their deployment. This is especially useful when wanting to add VPN clients outside of the cloud application deployment.



Generate and visualize certificates for a new client machine you'd like to add in this VPN.

Figure 14: CNSMO Dashboard - VPN tab

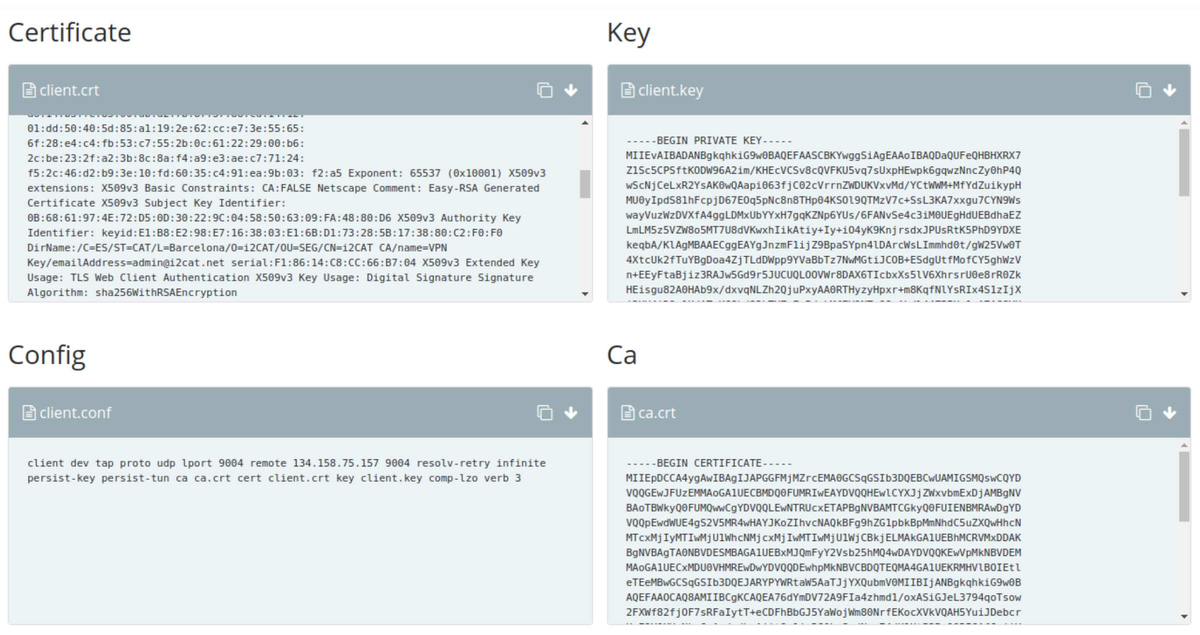
In this box (Figure 15), the user can put the name to be assigned to a new client and generate certificates and configuration files for it by clicking on the Generate button.



Generate and visualize certificates for a new client machine you'd like to add in this VPN.

Figure 15: CNSMO Dashboard - Generating VPN certificates and config for new client

When the user clicks on *Generate* four output files will be generated: *client.crt*, *client.key*, *client.conf* and *ca.crt* (Figure 16).



```
01:dd:50:40:5d:85:a1:19:2e:62:cc:e7:3e:55:65:
6f:28:e4:c4:fb:53:c7:55:2b:0c:61:22:29:00:b6:
2c:be:23:2f:a2:3b:8c:8a:f4:a9:e3:ae:c7:71:24:
f5:2c:46:d2:b9:3e:10:fd:60:35:c4:91:ea:9b:03: f2:a5 Exponent: 65537 (0x10001) X509v3
extensions: X509v3 Basic Constraints: CA:FALSE Netscape Comment: Easy-RSA Generated
Certificate X509v3 Subject Key Identifier:
0B:68:61:97:4E:72:D5:0D:30:22:9C:04:58:50:63:09:FA:48:80:D6 X509v3 Authority Key
Identifier: keyid:E1:B8:E2:98:E7:16:38:03:E1:6B:D1:73:28:5B:17:38:80:C2:F0:F0
DirName:/C=ES/ST=CAT/L=Barcelona/O=i2CAT/OU=SEG/CN=i2CAT CA/name=VPN
Key/emailAddress=admin@i2cat.net serial:F1:86:14:C8:CC:66:B7:04 X509v3 Extended Key
Usage: TLS Web Client Authentication X509v3 Key Usage: Digital Signature Signature
Algorithm: sha256WithRSAEncryption
```

```
-----BEGIN PRIVATE KEY-----
MIIEvAIBADANBgkqhkiG9w0BAQEFAASCBKYwggSiAgEAAoIBAQDaQUFeQHBHXRX7
Z1Sc5CPSftKODW96A2im/KHEcVCSv8cQVFKU5vq7sUxpHEwpk6gqwzNncZy0hP4Q
wScNjCeLxR2YsAK0wQAapi063fjC02cVrrnZWDUKVxvMd/YCtWWM+MfYdZuikypH
MU0yIpdS8lhFcpjD67EOq5pNc8n8THp04KSOl9QTMzV7c+SsL3KA7xxgu7CYN9Ws
wayVuzWzDVXfA4ggLDMxUbYYxH7gqKZNp6YUs/6FANvSe4c3iM0UEgHdUEBdhaEZ
LmLM5z5VZW8o5MT7U8dVKwxhIikAtiy+Iy+iO4yK9KnjrsdxJPUsRtK5PhD9YDXE
keqbA/KlAgMBAAECggEAYgJnzmF1ijZ9BpaSYpn4lDArcWsLImmhd0t/gW25Vw0T
4XtcUk2fTuYBgDoa4ZjTLdDWpp9YVaBbTz7NwMGtiJCOB+ESdgUtfMofCY5ghWzV
n+EEyFtaBjiz3RAJw5Gd9r5JUCUQLOOVWr8DAX6TIcbxXs5lV6XhrsrU0e8rR0Zk
HEisgu82A0HAb9x/dxvqNLZh2QjuPxyAA0RTHyzyHpxr+m8KqfNlYsRIx4S1zIjX
```

```
client dev tap proto udp lport 9004 remote 134.158.75.157 9004 resolv-retry infinite
persist-key persist-tun ca ca.crt cert client.crt key client.key comp-lzo verb 3
```

```
-----BEGIN CERTIFICATE-----
MIIEpDCCA4ygAwIBAgIJAPGGFMjMZrcEMA0GCSqGSIb3DQEBCwUAMIGSMQswCQYD
VQQGEwJFUzEMMAoGA1UECBMDQ0FUMRIwEAYDVQQHEwlCYXJjZWxvbmExDjAMBgNV
BAoTBWkyQ0FUMQwwCgYDVQQLEwNTRUcxETAPBgNVBAMTCGkyQ0FUIENBMRAwDgYD
VQQpEwdWUE4gS2V5MR4wHAYJKoZIhvcNAQkBFg9hZG1pbkBpMmNhdC5uZXQwHhcN
MTcxMjIyMTIwMjU1WhcNMjcxMjIwMTIwMjU1WjCBkjELMAkGA1UEBhMCRVMxDDAK
BgNVBAgTA0NBVDESMBAGA1UEBxMJQmFyY2Vsb25hMQ4wDAYDVQQKEwVpMkNBVDEM
MAoGA1UECxMDU0VHMREwDwYDVQQDEwhpMkNBVCBDQTEQMA4GA1UEKRMHVlBOIEtl
eTEeMBwGCSqGSIb3DQEJARYPYWRtaW5AaTJjYXQubmV0MIIBIjANBgkqhkiG9w0B
AQEFAAOCAQ8AMIIBCgKCAQEA76dYmDV72A9FIa4zhmd1/oxASiGJeL3794qoTsow
2FXWf82fjOF7sRFaIytT+eCDFhBbGJ5YaWojWm80NrfEKocXVkVQAH5YuiJDebcr
```

Figure 16: CNSMO Dashboard - Visualise and Download client VPN certificate and config

By clicking on the arrow buttons on the right top of the toolbar of each code boxes each file can be downloaded or copied in the clipboard. These four files are necessary for manually establishing a new VPN connection from the new client machine we wanted to connect to the VPN by means of an Open-vpn client.

2.1.5.Any-cloud compatibility and inter-operability

The CNSMO deployment is agnostic of the cloud where client or server instances are deployed. We have successfully tested different deployments on combined cloud environments as further described on deliverable D7.4. An important requirement for the multi-cloud inter-operability is having a public IP for each instance or at least a transparent way for Address Translation like AMAZON cloud provider has. According to that, several combinations have been tested, for example deploying the CNSMO Server in Amazon EC2, part of the CNSMO Clients in the CNRS-LAL (France) dedicated testbed for CYCLONE and other part of CNSMO clients in the QSC testbed in Germany or the Exoscale and MS Azure public clouds. The performance of the deployment and operation of the services depends on the cloud as well as on the concrete instance flavours but the successful tests reported in D7.4 clearly show that CNSMO is technology agnostic, which means that it works over any CSP where application components are deployed as long as the generic requirements are satisfied. Eventually, CNSMO has been tested over private clouds based on OpenStack and the following public clouds: Amazon EC2, Microsoft Azure, Exoscale.

2.1.6.Optimisation of network services deployment

CNSMO scalability tests, reported in D7.4, show that one of the bottlenecks of the CNSMO provisioning time is the installation of software packages required for the instantiation of CNSMO Services. For instance, the installation of OpenDaylight, since it has to be downloaded from the repository, increases substantially the time required for the deployment of an application where CNSMO-SDN service is enabled. A solution to cut significantly the deployment time is the possibility of building images in SlipStream. With this feature, a new image can be created with pre-installed packages, avoiding performing the step of packages installation and most part of the post-install steps.

2.1.7. CNSMO as a standalone service

CNSMO architecture has been designed since the beginning to be easily decoupled from the other modules of CYCLONE. The way how the different components of CNSMO are deployed is by means of bash scripts as start point of their initialization. These bash scripts then call corresponding python code that contains most of the logic and algorithms required for the deployment of CNSMO core and Services.

On the other hand, during last period of CYCLONE, the integration with SlipStream has been more intense, and some dependencies have been added in order to increase the performance for this integrated version. An example of that are some of the mechanisms for getting and setting the values of the recipe’s parameters, which are invoked also from CNSMO code in order to efficiently manage the synchronization between the different components and instances of CNSMO.

Thus, we can state that the latest version of CNSMO is fully integrated with SlipStream composing the CYCLONE stack, but CNSMO could also be used/integrated with other cloud orchestration platforms or even standalone.

2.1.8. Scalability and elasticity

Scalability and elasticity are key features of CYCLONE, and CNSMO services have been implemented in a way that can be deployed in scalable and elastic way.

Regarding scalability, a battery of tests has been performed in order to evaluate the behaviour of the CNSMO services deployment with different deployment sizes. These tests are reported in Deliverable D7.4 with successful results, showing that the provisioning time increases linearly with the deployment size, proving the scalability of CNSMO.

Regarding elasticity, CYCLONE allows adding new instances to an application deployment, as an addition to the initially specified instances at deployment time. Consequently, CNSMO services must also allow the possibility of enabling network services in these newly added instances. When a new machine is provisioned, all the selected CNSMO services are automatically installed and deployed on the application VM. With this modification of the CNSMO-VPN implementation, the VPN server can continue with its deployment without waiting for each of the VPN clients to be ready. This way, connection responsibility has been given to VPN clients and allows any instance to be connected to the VPN in any moment as long as the server is up. CNSMO-SDN also supports dynamically addition of new instances, so that, once a new client instance is added to the VPN it connects automatically to the SDN server and appears in the “network Map” as a new machine manageable by the SDN functionalities.

2.2. CNSMO final modules

This section details the technical implementation aspects of the different components and services that compose the CNSMO service platform.

2.2.1. CNSMO server and CNSMO agent

From a logical point of view, we can differentiate two types of components: **CNSMO Server** and **CNSMO Agent**.

The CNSMO Server acts currently as a ***VPN server***, ***SDN controller***, ***DNS server***, ***Load Balancer*** as well as hosting the ***CNSMO WEB components***. CYCLONE’s implementation of the CNSMO server is centralized, deploying the network functions in the same VM, to ease configuration of the SlipStream deployment recipes. In this case, since this component is mandatory, it should be unique and it requires high performance due the amount of functions it has to handle. However, the different network functions could also be instantiated anywhere in the cloud with reachability with the CNSMO server central engine and the CNSMO agents.

The CNSMO Agent is the component from which all the client instances inherit from. The multiplicity of this component is one for each client of the deployment. Basically, the client instances are the ones where specific third-party applications will be installed and are the instances that Administrators want under a controlled network.

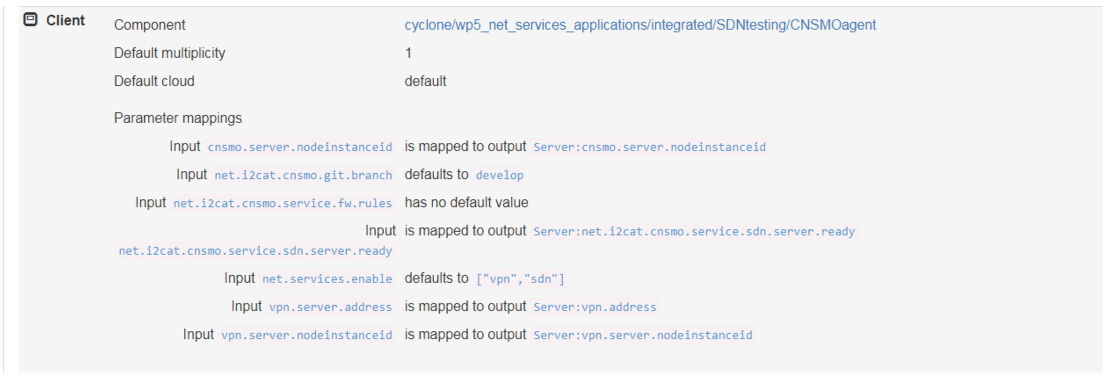
For deploying both components two *python* scripts for each component must be executed. Since we are providing an integrated solution for CYCLONE, these scripts are executed by the *SlipStream* recipe post-install and deployment scripts (*bash* script), but with some adaptations these python scripts could be executed standalone on independent already provisioned machines.

The way how the integrated version works depends on the deployment we are performing and on the Application recipe used by the deployment, a specific number of CNSMO Agent + a CNSMO Server instances are initialised and provisioned under SlipStream orchestration. Two recipes *post-install* and *deployment* are executed after the provisioning time and after packages installation. In addition to that, a list of parameters is defined in the recipes for the correct execution of the deployment.

The Input/output parameters are used for the correct synchronization between client, server and orchestrator, and they are also used in order to exchange important information between them. Depending on which services we want to deploy on the machines, we will need to configure different parameters so that we make visible the private information needed for the services to work properly. For example, a VPN server should make its IP address visible to the clients so that they can stablish a VPN with the server. Here, you can see the list of parameters required:

Agent

- Input parameter *cnsmo.server.nodeinstanceid* is mapped to output *Server:cnsmo.server.nodeinstanceid*
- Input parameter *net.i2cat.cnsmo.git.branch* has to be set to the git branch from where the code has to be downloaded
- Input parameter *net.services.enable* parameter is used to indicate which network service has to be deployed following the format *["vpn","fw","lb","sdn","dns"]*
- Output parameter *net.i2cat.cnsmo.core.ready* set to default
- Output parameter *net.i2cat.cnsmo.dss.address* set to default
- Output parameter *net.services.enabled* set to default



Client	Component	cyclone/wp5_net_services_applications/integrated/SDNtesting/CNSMOagent
	Default multiplicity	1
	Default cloud	default

Parameter mappings

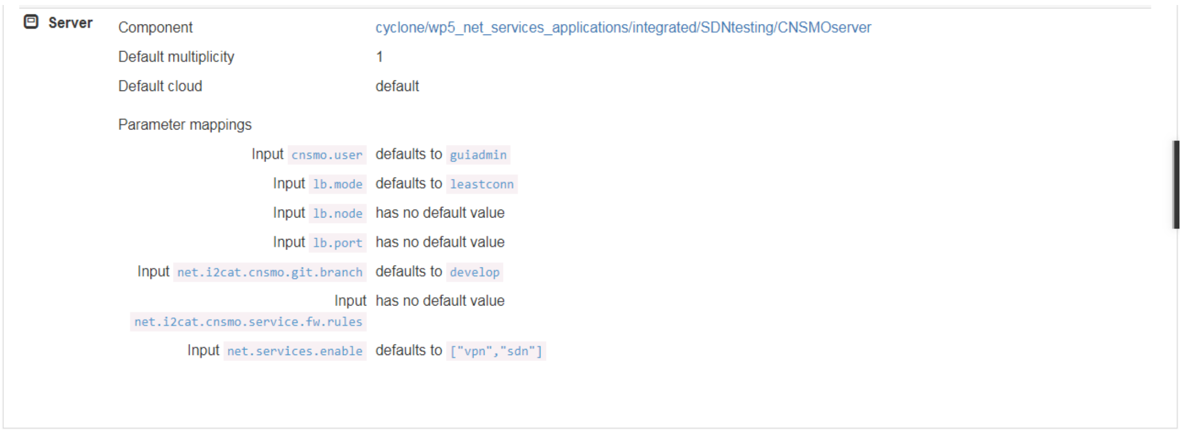
Input `cnsmo.server.nodeinstanceid`	is mapped to output `Server:cnsmo.server.nodeinstanceid`
Input `net.i2cat.cnsmo.git.branch`	defaults to `develop`
Input `net.i2cat.cnsmo.service.fw.rules`	has no default value
Input `net.i2cat.cnsmo.service.sdn.server.ready`	is mapped to output `Server:net.i2cat.cnsmo.service.sdn.server.ready`
Input `net.services.enable`	defaults to `["vpn","sdn"]`
Input `vpn.server.address`	is mapped to output `Server:vpn.address`
Input `vpn.server.nodeinstanceid`	is mapped to output `Server:vpn.server.nodeinstanceid`

Figure 17: CNSMO Agent SlipStream recipe parameters

Server

- Input parameter *net.i2cat.cnsmo.git.branch* has to be set to the git branch from where the code has to be downloaded.
- Input parameter *net.services.enable* parameter is used to indicate which network service has to be deployed following the format *["vpn","fw","lb","sdn","dns"]*
- Output parameter *cnsmo.server.nodeinstanceid* set to default
- Output parameter *net.i2cat.cnsmo.core.ready* set to default
- Output parameter *net.i2cat.cnsmo.dss.address* set to default
- Output parameter *net.services.enabled* set to default

Some services might need additional parameters to be configured



Server

Component	cyclone/wp5_net_services_applications/integrated/SDNtesting/CNSMOserver
Default multiplicity	1
Default cloud	default

Parameter mappings

Input `cnsmo.user`	defaults to `guiadmin`
Input `lb.mode`	defaults to `leastconn`
Input `lb.node`	has no default value
Input `lb.port`	has no default value
Input `net.i2cat.cnsmo.git.branch`	defaults to `develop`
Input `net.i2cat.cnsmo.service.fw.rules`	has no default value
Input `net.services.enable`	defaults to `["vpn","sdn"]`

Figure 18: CNSMO Server SlipStream recipe parameters

After provisioning and after required packages are installed, the *SlipStream* post-install recipe is executed in each one of the instances (CNSMO Server and CNSMO Agents). This recipe performs basically two things, downloading the CNSMO Services code from the repository of CNSMO (branch is indicated as input parameter) and executing a python file contained in the following repository folder:

cnsmo/src/main/python/net/i2cat/cnsmoservices/integrated/run/SlipStream/

In case of CNSMO Server, the *netservicesserverpostinstall.py* will be executed.

In case of CNSMO Agent, *netservicesclientpostinstall.py* will be executed.

In general, these scripts perform the installation of all CNSMO dependencies as well as calling all the corresponding *post-install* scripts for each of the CNSMO services that must be enabled (the ones specified in the deployment as input parameter). In that way, each of the services (VPN, SDN, DNS, LB, etc.) has its own *postinstall.py* file that will be executed from that point in a sequential way.

There is a different folder for each one of the services in the following path of the git repository:

cnsmo/src/main/python/net/i2cat/cnsmoservices/

In addition to executing the *post-install* scripts for the services listed in the input *net.services.enable* parameter, the *netservicesserverpostinstall.py* downloads the repository of CNSMO WEB (GUI + API) and installs and executes it. From that moment the CNSMO-API and CNSMO-GUI are accessible.

The *post-install* step will finish when all the *postinstall.py* files from all the services to enable will finish its execution.

After the *post-install* step, the *netservicesserverdeployment.py* and *netservicesclientdeployment.py* are respectively executed by the *SlipStream* deployment recipe. These files execute sequentially the different *deployment.py* files contained in each one of the folders corresponding to the CNSMO services to be enabled. If all the deployment functions called return a TRUE value, then the deployment is successful and all the services we wanted to enable are available.

Each of the CNSMO Services is explained as a component in the following sections. At the end of each deployment file (one for each set of services) a Python Flask Application is instantiated in a different port in order to act as internal API for holding requests from other components or from the external CNSMO-API. The definition of these internal APIs can be found in README.md files contained in each one of the subfolders of *cnsmo/src/main/python/net/i2cat/cnsmoservices/* inside its */app/ folder.*

2.2.2.SDN components

2.2.2.1. Deployment Architecture

Due to the SDN Overlay there is no fingerprint in the networking configuration. By virtue of the VPN Service, the regular traffic is conventionally routed whereas the SDN signalling is transported through a dedicated VPN. The SDN controller is the responsible for managing the networking of the cloud application and its instructions are applied to the *Open vSwitch* bridges in the required client machines. The public access to the Virtual Machines is maintained as a result of configuring the bridges as the default interface (see Figure 19). Thus, a true multi-cloud environment can be constructed regardless of the cloud provider technology. If a cloud owner does not provide public IP addresses, a proxy service should be used to forward all traffic to the application components residing within that specific cloud.

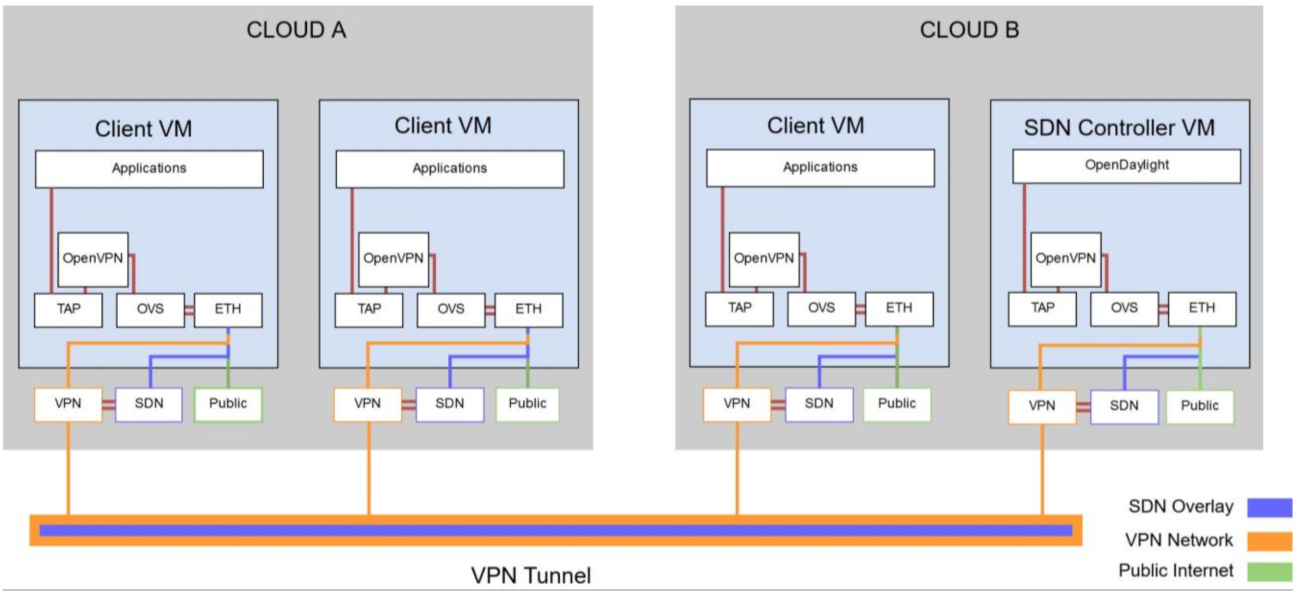


Figure 19: Internal interfacing of the SDN Overlay Network Service

2.2.2.2. SDN Client - Open vSwitch

Open vSwitch is a multilayer software conceived for switches, which objective is the implementation of a quality platform that supports standard management interfaces and exposes the functions of forwarding in a programmable way. We can say that Open vSwitch is one of the most popular implementations of a virtual OpenFlow switch. It is well adapted to work as a virtual switch in environments implemented with virtual machines. In addition to exposing standard control and visibility interfaces with the virtual network layer, it was designed to support distribution across multiple physical servers. Open vSwitch supports numerous Linux-based virtualization technologies.

The CNSMO-SDN client component installs Open vSwitch as a Linux package at post-deployment time in the client virtual machines. Afterwards, an Open vSwitch bridge is created in order to interconnect the VPN interface and external interface of the VM. Once the bridge is created, it is configured as the default route interface. Finally, the bridges subscribe to the SDN controller to delegate its control.

2.2.2.3. SDN Server -OpenDaylight

The CNSMO Server component uses OpenDaylight as SDN controller. OpenDaylight is an open-source project hosted by the Linux Foundation aimed at enhancing Software-defined networking. The OpenDaylight Controller is able to deploy in a variety of production network environments. It exposes open northbound APIs, which are used by applications. These applications, like the CNSMO-SDN component, use the Controller to collect information about the network, run algorithms to conduct analytics, and then use the OpenDaylight Controller to create new rules throughout the network.

OpenDaylight is downloaded by the CNSMO-SDN component at post deployment time in the server virtual machine. Subsequently, the OpenDaylight features are installed to allow the Open vSwitch bridges to be subscribed to the controller. Consequently, and thanks to its northbound APIs, CNSMO takes full control of the cloud application's networking.

At the end of deployment phase, a Python Flask App Server is deployed in order to handle requests defined conforming current internal API, from other components. The definition is specified in *cnsmo/src/main/python/net/i2cat/cnsmoservices/sdnoverlay/app/server.py.* The most important methods implemented for this API are:

- ***http://127.0.0.1:20199/sdn/server/flows/ , methods=[GET]***
  Gets all the SDN configured flows. If no parameter is passed (*GET -d '{}'*), it will return all configured flows for each of the nodes subscribed in the SDN controller. If we want the flows from a particular client, the user should specify the client ID using the parameter *ssinstanceid* from the body.

- ***http://127.0.0.1:20199/sdn/server/nodes/ , methods=[GET]***
  Returns a list of pairs with the *nodeid* and his corresponding VPN address. No parameters need to be configured.

- ***http://127.0.0.1:20199/sdn/server/filter/blockbyport/ , methods=[PUT]***
  Adds a new rule to filter flows by port. We need to specify the destination port, the address and the node name following the format: *{"tcp-destination-port":1919,"ip4-destination":"134.157.24.35/16","ssinstanceid":"Agent.1"}*

- ***http://127.0.0.1:20199/sdn/server/filter/blockbyport/instance/flow/ , methods=[DELETE]***
  Deletes the flow with *flowid = flowID* for given *ssinstanceid*.

2.2.3. DNS server

2.2.3.1. Dnsmasq

The CNSMO-DNS service uses dnsmasq to enable the DNS functionalities. Dnsmasq accepts DNS lookups and replies from a small local cache, or forwards them to a real recursive DNS server. It loads the contents of the */etc/hosts* file, in such a way that names of local hosts which do not appear in the global DNS can be solved.

When the DNS server receives a request, it first checks if the host is defined in the configuration file. If it is not, it checks the */etc/hosts* file of the machine. Finally, if the requested host is not in either file, it forwards the request to the upstream DNS (see Figure 20). The CNSMO-DNS service configures Dnsmasq to use the same upstream DNS servers as the default that the VMs get when spawned.

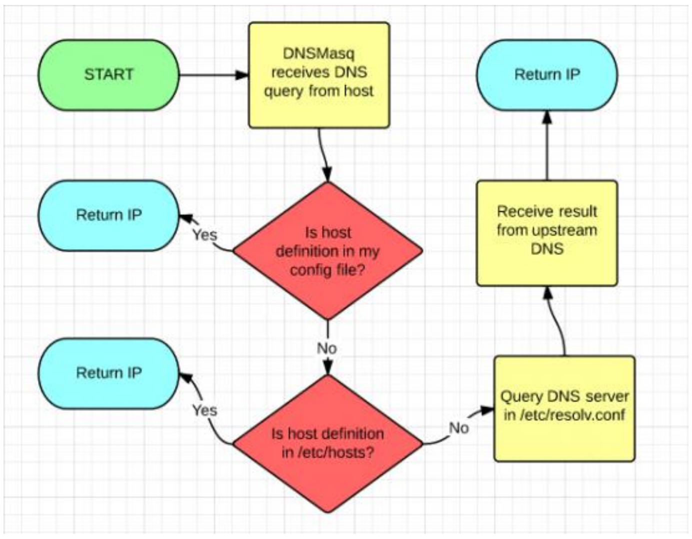

START
DNSMasq receives DNS query from host
Return IP
Is host definition in my config file?
Yes
Return IP
No
Receive result from upstream DNS
Return IP
Yes
Is host definition in /etc/hosts?
No
Query DNS server in /etc/resolv.conf

Figure 20: Workflow of a DNS request in Dnsmasq [Dnsmasq]

At the end of the deployment of CNSMO-DNS *a Python Flask Server* is enabled in order to offer two methods to be used from CNSMO-API: one method for listing the DNS records existing in DNS Server and another method for adding a new record (mapping IP address – machine name) to the DNS Server records file. The DNS server will be capable of resolving all the records it contains in the records file when some client instance will ask for the IP associated to a certain name.

2.2.4.VPN server

The logic of the CNSMO-VPN service has been enhanced to accept clients joining and leaving the VPN service dynamically. Previously, the VPN setup worked as a batch process considering only the clients available at that moment but now clients may join and leave whenever they need. This change opens the door to accept new VMs in the deployment, being a requirement for horizontal scaling of the CNSMO services as well as for the overall application.

2.2.4.1. Updated sequence diagram

In the previous implementation of CNSMO-VPN, the VPN service did not start until all the components were ready. Currently, once the VPN Server is ready, it starts the service without waiting for any client. This new behaviour implies a performance improvement and allows clients to join the VPN dynamically (Figure 21).

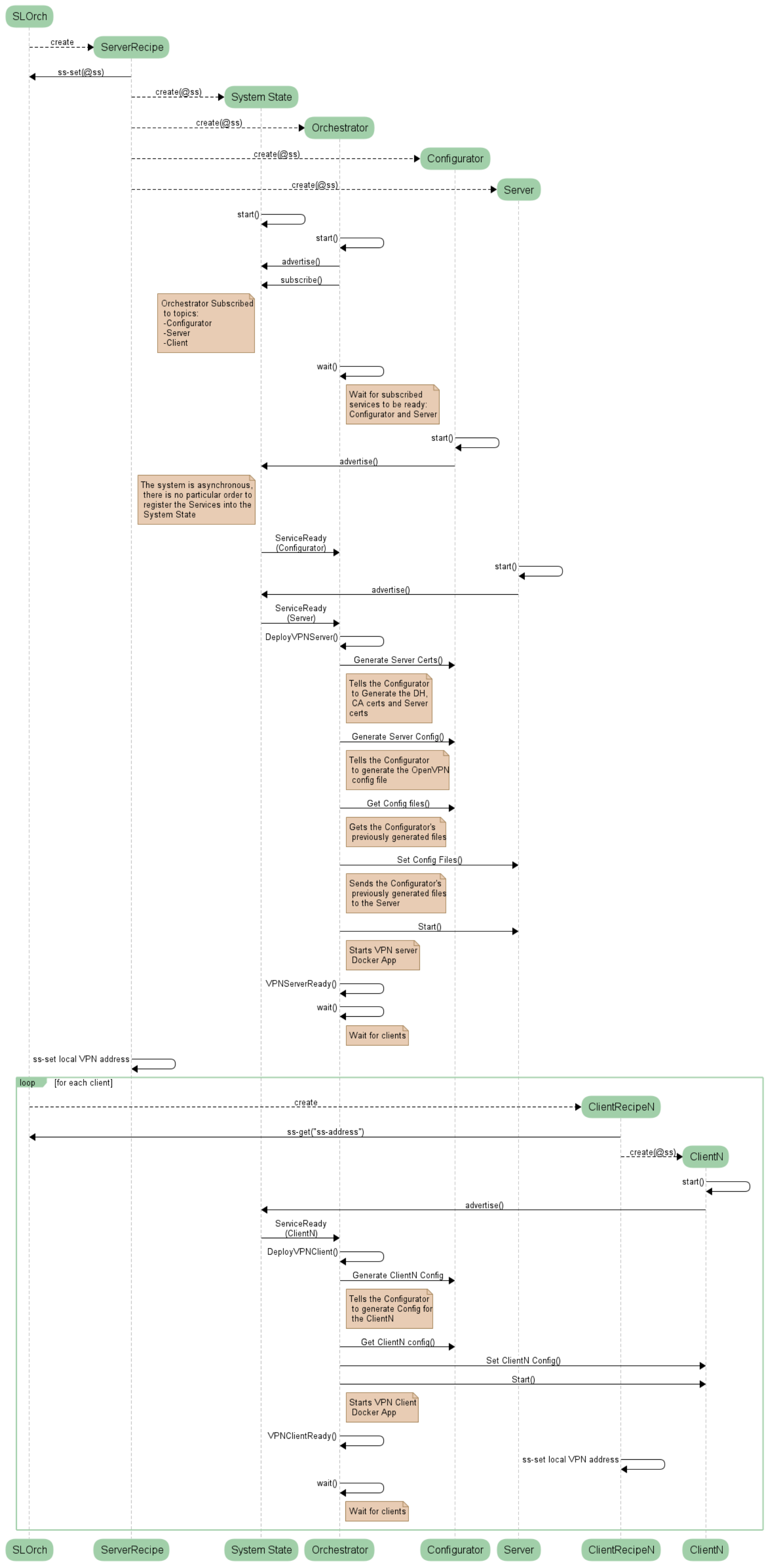


Figure 21: CNSMO-VPN server updated workflow

2.2.5. LB server

CNSMO-LB services (Load Balancing) have not been modified since the status they were when reported in previous deliverables of WP5. LB components can be deployed as well in a joint scenario with VPN, SDN and DNS services, thus, it is fully integrated with new version of CNSMO Services.

2.2.6. CNSMO WEB

2.2.6.1. General Architecture of CNSMO WEB

The CNSMO WEB component is composed by CNSMO-API and CNSMO-GUI. CNSMO-API provides http methods to the different APIs of the different CNSMO Services previously described. Its main objective is to be consumed by the CNSMO-GUI, but can be also used from an independent HTTP Client like Postman.

CNSMO-GUI provides a client-side application executable in a web browser that allows CYCLONE App Administrators to perform high level operations that are internally handled by CNSMO services.

Both are deployed automatically through a post-install script of the CNSMO Server component deployed from SlipStream. Figure 22 depicts how it works.

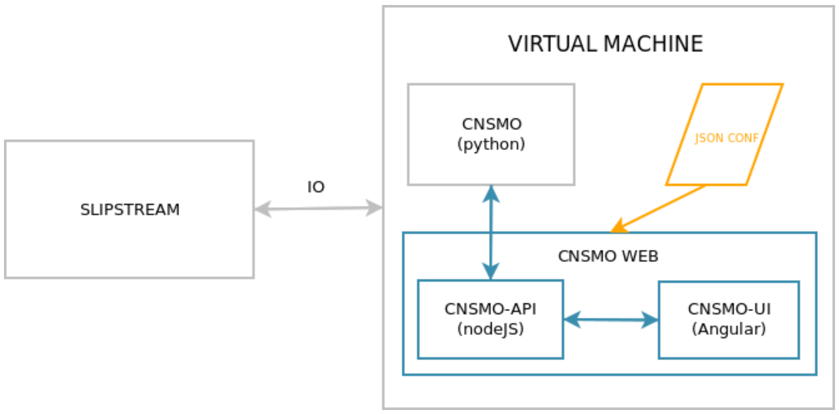


Figure 22: CNSMO WEB architecture CNSMO API

CNSMO API is implemented using *NodeJS* and is generated from the *ExpressJS* framework. The main elements of the developed structure are, Routes, Controllers and Middlewares. Its architecture is shown in Figure 23.

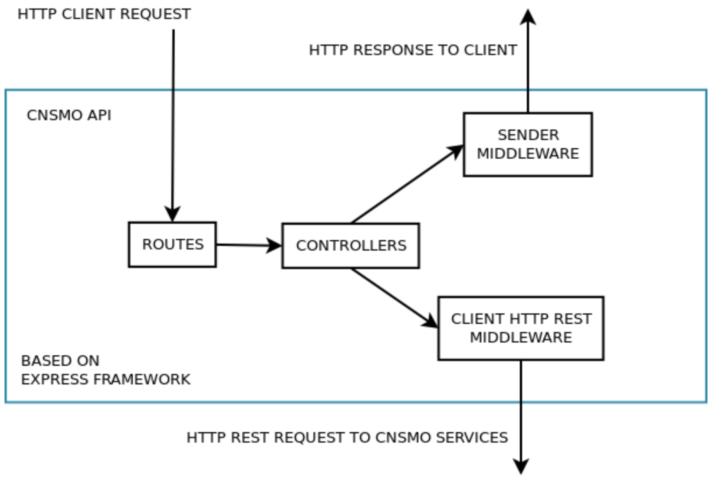

HTTP CLIENT REQUEST
HTTP RESPONSE TO CLIENT
CNSMO API
SENDER MIDDLEWARE
ROUTES
CONTROLLERS
CLIENT HTTP REST MIDDLEWARE
BASED ON EXPRESS FRAMEWORK
HTTP REST REQUEST TO CNSMO SERVICES

Figure 23: CNSMO API architecture

A route method is derived from one of the HTTP methods, and is attached to an instance of the express class. The following code (Figure 24) is an example of routes that are defined for the GET and the PUT methods to the TCP Ports of the app.

```
// TCP Ports (flows)
app.get(
  baseUrl + '/services/sdn/nodes/:instanceId/blockedTcpPorts',
  jsonParser,
  ensureAuthorized,
  services.sdn.getBlockedTcpPortsByNode
);
app.put(
  baseUrl + '/services/sdn/blockbyport',
  jsonParser,
  ensureAuthorized,
  validators.blockByPort,
  services.sdn.blockByPort
);
```

Figure 24: Route method examples

The list of implemented routes corresponds to the methods of the CNSMO-API listed in section 2.1.4.

Controllers are responsible for invoking the appropriate action, to manage CNSMO Services response/responses and to collect information in order to do a re-elaboration of data and prepare a readable and friendly response to client. The following code (Figure 25) is an example of controller *users.js* file.

```
'use strict';
var core = require('../core');
var send = core.helpers.send;
var credentials = core.initParams.credentials;
var auth = core.middlewares.auth;

function authenticate(req, res) {
const reqCredentials = req.body;
let result;
try {
  if (reqCredentials.username === credentials.username &&
    reqCredentials.password === credentials.password) {
    result = {
      code: 200,
      response: {
        token: auth.createToken(reqCredentials)
      }
    };
    send(res, result.code, result.response);
  } else {
    const error = {
      code: 401,
      message: 'Username or password not valid!'
    };
    throw error;
  }
} catch (e) {
  return send(res, e.code, e);
}
}

module.exports = {
authenticate: authenticate
};
```

Figure 25: *users.js* controller example

Different middlewares are used. Sender middleware and client middleware are the main ones. The first one is responsible of responding client's requests and the second one is responsible of requesting CNSMO Services. Both middlewares follow the HTTP REST philosophy.

The following code (Figure 26) is an example of how to use the send method that calls the sender middleware and the client middleware in a *getNodes* function of *services.js* controller. Middlewares are called by *cnsmoClient.get* (calls the client middleware) and sent (call the sender middleware).

```
function getNodes(req, res) {
cnsmoClient.get('http://127.0.0.1:20092/vpn/server/clients/')
 .then((result) => {
   var resp = Object.keys(result.data).map((key) => {
     var retObj = result.data[key];
     retObj.instanceId = key;
     return retObj;
   });
   return send(res, res.statusCode, resp);
 }).catch((err) => {
   console.log(err);
```

```
    const error = {
      code: 500,
      message: 'Error!'
    };
    return send(res, error.code, error);
  });
}
```

Figure 26: Example on how to use *send* method

In order to deploy CNSMO-API, the installation script automatically installs the version 6.x of *NodeJS*, then clones the repository and finally executes the *npm install* command. After these steps the application is ready to be executed. Before launching CNSMO-API, the configuration file is set like the following example (Figure 27) and moved to the following path: */config/env*.

```
{
BASE_URL: "www.base.url/",
MONGO_URL: "",
DOMAIN: "127.0.0.1",
PROTOCOL: "http",
port: 8080,
SWAGGER: true,
JWT_SECRET: "xxx",
TOKEN_EXPIRATION_DAYS: 10,
};
```

Figure 27: CNSMO API configuration file

Then, *NODE_ENV* variable is set to the name of the configuration file and finally *node app.js* is executed.

The wide application is tested using different tools and technologies:

- mocha
- chai
- istambul
- jslint
- check outdated
- security

The previous tools are launched before every commit by a script to repository in order to maintain a good status of the project and to have constantly a report of what is the percentage of coverage of tests in all the applications files.

2.2.6.2. CNSMO GUI

CNSMO GUI is implemented using *Angular* v4 and it is generated with *angular-cli* tool by launching the command *ng new cnsmo-gui*. Routes, classes, components, services and all the other elements that compose every single feature are implemented as independent modules in order to provide to the project all the scalability and the persistency that it needs.

The following example scheme describes the architecture structure of the GUI project.

```
> app
      > core
      > layout
            > firewall
                  > shared
                        > firewall.service.ts
                        > rule.ts
                  > firewall.routing.module.ts
                  > firewall.module.ts
                  > firewall.component.ts
                  > firewall.component.html
                  > firewall.component.scss
            > dns-records
> …

            > layout.component.ts
            > layout.component.html
            > layout.component.scss
            > layout.routing.module.ts
            > layout.module.ts
      > login
      > not-found
      > shared
> assets
> environments
> styles
```

Figure 28: Architecture structure of CNSMO-GUI

Regarding the *app path* where all the features of the application are developed, all the modules like *firewall, dns-records*, etc. are present in the layout path. *Layouts* represent graphically the structure of an application and they are also a module themselves.

The core path is another module that contains services, components and other helpful items that are used in the wide application and introduce required functionality like the *notification box* or the *http-client service*.

The Shared module contains modules that are useful for the others layout modules or also for the same layout itself. Clear examples are *the header* and *the sidebar* or the generic form input done in order to avoid redundancy of code.

Finally, the *login*, *not-found* and *sign-up* paths represent simply a few modules out of a general layout module and they are used for provide user access and a not found page in the case users try to access to a URL without content.

Figure 29 is an overview on how a generic component in a module works and interacts with a service.

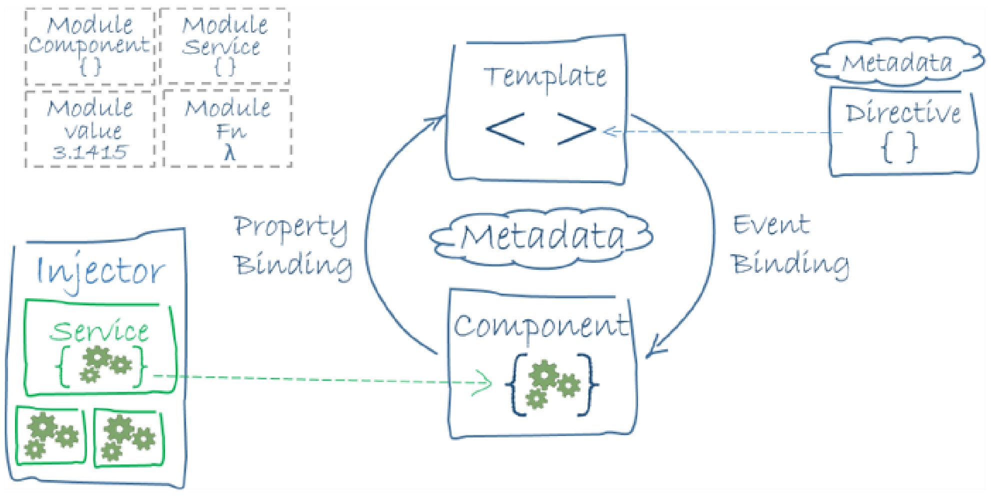


Figure 29: Overview of how generic component in a module works and interacts with a service

The *Angular Router* enables navigation from a layout view to the next as users perform application tasks. This module, as specified by the Angular standards, manages the authorization for navigating to concrete routes and user's permissions through the *auth.guard.ts* that is in the core path of CNSMO GUI application. The following code describe how the CNSMO *auth.guard* works (Figure 30):

```
@Injectable()
export class AuthGuard implements CanActivate {
constructor(
private router: Router,
private auth: AuthService) {
}
canActivate(
  next: ActivatedRouteSnapshot,
  state: RouterStateSnapshot): Observable<boolean> | Promise<boolean> | boolean {
   if (this.auth.currentUser && this.auth.currentUser.token) {
     return true;
   }
   this.router.navigate(['/login']);
   return false;
 }
}
```

Figure 30: CNSMO-GUI user navigation with auth.guard

When a user tries to access to some route from a layout view, this *guard* verifies if it has a token (permission). If not, it forces the navigation to /login route where *auth.guard* is not called.

The following code (Figure 31) showing *layout.routing.module.ts* describes how the guard is integrated into routes. Implemented *AuthGuard* is used as an element of array. This array is a value of both *canActivate*. Every child routes have other inner routes that can be activated if meet the condition of the *AuthGuard* (and eventually other guards can be implemented and added in the future versions).

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { LayoutComponent } from './layout.component';
import { AuthGuard } from 'app/shared';
const routes: Routes = [
  {
     path: '', component: LayoutComponent, canActivate: [AuthGuard],
     children: [
        { path: 'dashboard', loadChildren: './dashboard/dashboard.module#DashboardModule' },
        { path: 'firewall', loadChildren: './firewall/firewall.module#FirewallModule' },
        { path: 'nodes', loadChildren: './nodes/nodes.module#NodesModule' },
        { path: 'vpn', loadChildren: './vpn/vpn.module#VpnModule' }
     ]
  }
];
@NgModule({
imports: [RouterModule.forChild(routes)],
exports: [RouterModule]
})
export class LayoutRoutingModule { }
```

Figure 31: Guard integrated into routes

In order to build the project, the command *ng build --env=prod --port 8xxx* must be executed. This will create a compiled project divided in different minified bundles ready to be hosted in any static path of any server you want to use and it contains *on*ly pure HTML, CSS and *Javascript* files. This command also specifies to compile the project with the production variables defined in */environments/environment.prod.ts* and to use a specific port.

CNSMO-GUI provides only one user role. Ideally the owner is the same Nuv.la account owner. According to that, an output parameter is set in the *SlipStream* CNSMO Server component in order to indicate the desired username we want to access with. The password is randomly generated and showed in the SlipStream GUI only to the administrator which has deployed that particular Application Deployment. After the CNSMO WEB components have been deployed, the administrator can log in in the CNSMO-GUI using these credentials and accordingly authorization mechanisms would allow the access to all the features of the Web application that correspond with his CNSMO deployed services.

3. CNSMO integration with other CYCLONE components

3.1. Integration with SlipStream component

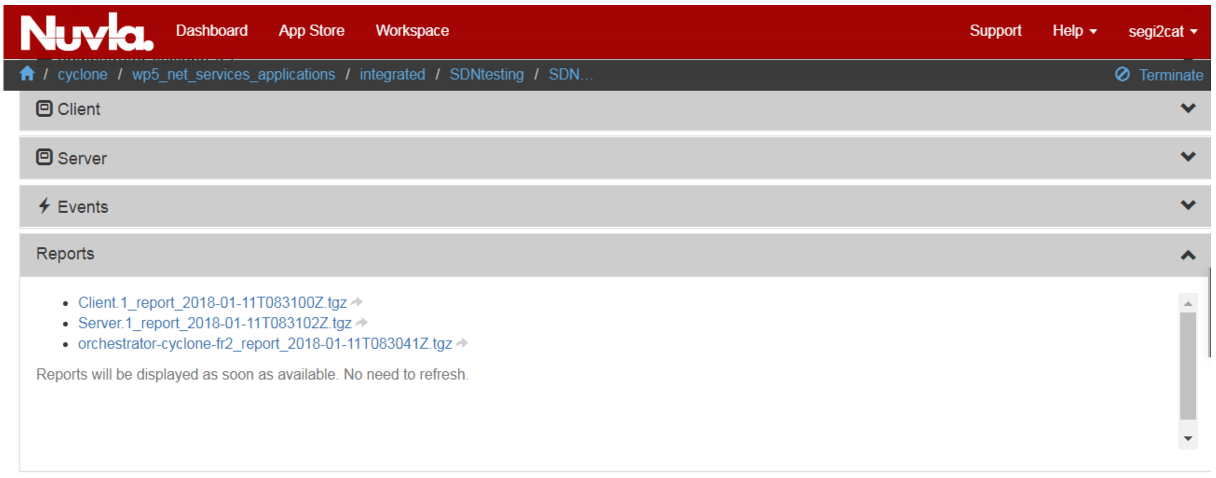
As commented in previous sections, CNSMO have been integrated with SlipStream to compound the CYCLONE software stack. The way how SlipStream users can include in their applications the CNSMO components is by means of making the components of the application inherit from the CNSMO Server and the CNSMO Agent SlipStream components. In that way, for each application workflow defined in the new component recipe (package installation, post-install, deployment, etc) the corresponding script of the CNSMO Server/Agent is executed right before the specific workflow of the new application component. According to that, at the end of the deployment of the application in SlipStream, the CNSMO Server components will be deployed inside the instance acting as server and the CNSMO Agent components will be deployed in all instances where client applications are running.

SlipStream's input and output parameters can be accessed from CNSMO deployment scripts, so that synchronisation between the different components of CNSMO is ahieved: VPN client with Server VPN, Server VPN with Server SDN, CNSMO GUI with SDN Server, etc. In addition, the creation of VPN certificates can be generated by SlipStream as this is one of the functionalities envisioned by *Nuvla*.

Finally, the generation of password for CNSMO API and GUI is done by a SlipStream tool as an additional example of integration between CNSMO and SlipStream.

3.2. Integration with the logging system

CNSMO creates a logging file, *cnsmo-integrated-deployment.log*, for each instance that is deployed. This log file uses different levels (ERROR, WARNING, INFO, DEBUG) of logging in order to annotate the relevant events that occur during the deployment of CNSMO. At the end of deployment, the Reporting workflow step, of the CNSMO components the executed scripts in SlipStream copies the CNSMO log file into the CNSMO_REPORT_DIR, creates a zipped (into .tgz file) file with the content of that folder and provides every instance's .tgz through SlipStream Interface. The SlipStream user who deployed the application can access these files under the *Reports* section of the individual deployment window.



- Client.1_report_2018-01-11T083100Z.tgz
- Server.1_report_2018-01-11T083102Z.tgz
- orchestrator-cyclone-fr2_report_2018-01-11T083041Z.tgz

Reports will be displayed as soon as available. No need to refresh.

Figure 32 Reports section of SlipStream current deployment

On the other hand, in WP4, a distributed logging system based on *Elasticsearch*, *Logstash* and *Kibana* (ELK) has been set up and adapted for use in multi-cloud environments. As it has been described in D4.5, the system enables central logging and evaluation of log data from other components. There are many different ways to let the ELK stack process the log data. The log data can be sent to the Logstash instance by using Beats data shippers or by configuring logging technologies that are used in self developed software, such as Log4j. According to that, CNSMO log files are also sent to the distributed logging system of CYCLONE, enabling the aggregation of all logs of all the deployments.

3.3. Integration with authentication and authorisation system

The authentication in CNSMO API and GUI is performed using username (specified by the user deploying the application in SlipStream) and password (generated randomly in SlipStream). The Authentication method of the CNSMO API returns a token that is used for authorize the different calls to services of CNSMO API (consumed also by CNSMO GUI). On the other hand, in WP4, CYCLONE has implemented the DACI system (Dynamic Access Control Infrastructure) described in deliverable D4.5 [CYCLONE-D4.5]. By deploying the DACI server as standalone VM in the CNSMO deployment, once login request is submitted through the CNSMO GUI the DACI Authorization Service is. For each of the CNSMO API methods different permissions can be configured for the users authorised for its consumption. This is stored in DACI and the authorisation service responds if a certain user has permission to make a particular invocation to the CNSMO API. When authorised, the call to the Method API would be allowed.

4. Review of other solutions and initiatives

A brief review of possible competitor projects and platforms is provided below. Other projects, solutions, platforms or start-ups might have started during the lifetime of the CYCLONE project, but the solutions presented below are, to the best of our knowledge, the most relevant ones

4.1.1. BEACON project

BEACON [BEACON] is an H2020 RIA project, which main goal is two-fold. On one hand, it researches and develops techniques to federated cloud network resources, and on the other hand it derives the integrated management cloud layer that enables an efficient and secure deployment of federated cloud applications. This project defines and implements a federated cloud network framework that enables the provision of federated cloud infrastructures, with special emphasis on inter-cloud networking and security issues, to support the automated deployment of applications and services across different clouds and datacentres.

BEACON focuses on a federated cloud network framework that has three main core management components, the BEACON Network Manager, the BEACON Network Agent and the BEACON broker, which is in charge of the federation of different cloud platforms and allow for the cross deployment of applications across those platforms.

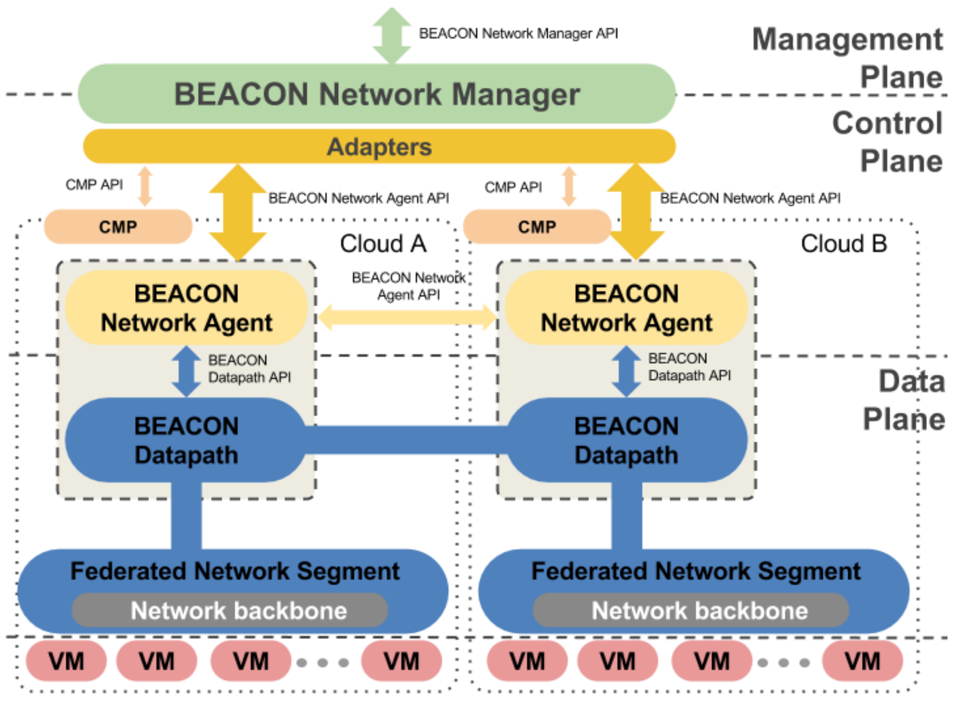


Figure 33: BEACON architecture for cloud network federation

Integration is achieved through the inter-operations of OpenNebula [OpenNebula] and OpenStack as cloud providers. The cloud manager of each platform is done using the BEACON Broker. The BEACON Network Agent and the BEACON Network Manager are integrated with the network manager of each platform.

The main difference between the BEACON Cloud network solution and CNSMO is that it is based purely on a federation of clouds, whether CNSMO can be used in any cloud, private or public, completely agnostic of any Cloud provisioning system and independently if the clouds are federated or not. Beside this, CNSMO does not need to be integrated with the network manager or the cloud manager of each platform as it is a pure overlay based solution.

4.1.2.MidNnet

MidoNet [MidoNet]is an open-source Network virtualization software for Infrastructure-as-a-Service (IaaS) clouds. It decouples IaaS clouds from the network hardware, creating an intelligent software abstraction layer between the end hosts and the physical network. It virtualizes the network functionality for IaaS products, such as OpenStack, providing functionally advanced, robust, scalable, and secure networks.

MidoNet allows building isolated virtual networks from L2 to L4 in a fully distributed system and allow per-tenant control of the network to create and change network topologies on the fly. Examples of the networking service it supports are:

- Distributed L4 Load Balancer
- NAT and floating IP addresses
- VxLAN

MidoNet runs software on standard x86 servers, and sits on top of a network underlay (for example, physical servers and switches), pushing the intelligent network functions to the edge of the network, in software. Therefore, it requires MidoNet Agents to run in software on the edge of the network in both the gateway nodes as well as the compute hosts, using tunnelling for communication between physical hosts in the underlay. In addition to the MidoNet Agents, there is also a state-management system, which coordinates with each of the distributed agents. For centralized management, MidoNet also provides a RESTful API server.

The MidoNet solution consists of these components:

- **MidoNet Network State Database nodes**. Passive data base component (Kasandra and Zookeeper): Zookeeper does the storage of the topology information and Kassandra to store everything else.
- **Agents** that need to be run in the user space on top of the Linux kernel on any hypervisor.
- **MidoNet Gateway Nodes**. It is a Linux white box that runs BGP to do outbound connections and use GRE or VxLANs for establishing internal tunnels.
- **Servers** running the MidoNet API, the MidoNet Agent (Midolman), and the Command Line Interface (CLI)
- **OpenStack controller nodes** hosting Nova and OpenStack Networking services
- **OpenStack Compute nodes**

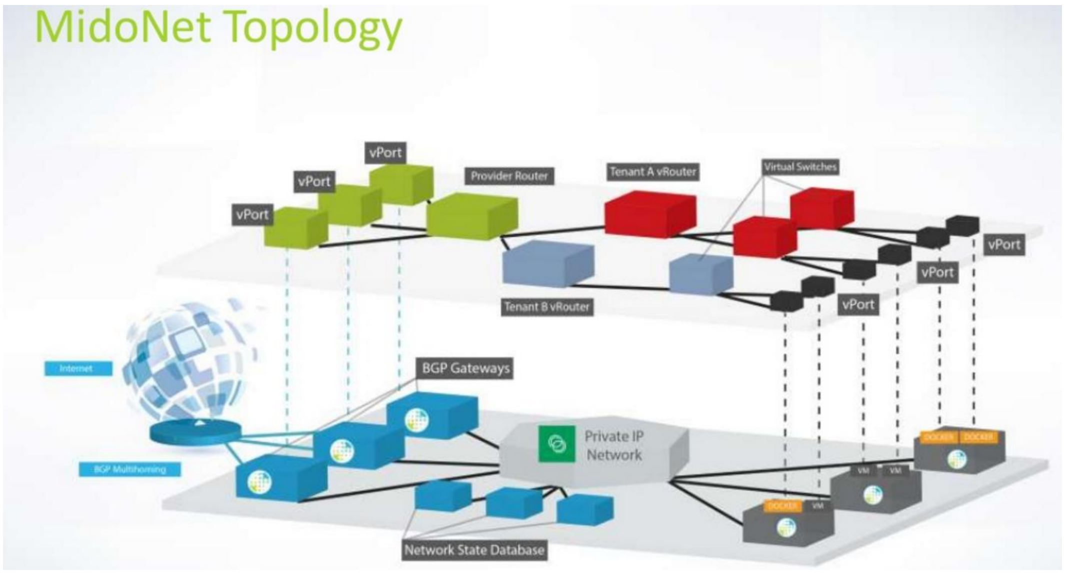


Figure 34: MidoNet topology

MidoNet does not work on top of OpenStack, but works as an OpenStack Neutron plugin, taking over nearly all of the networking functions currently found in OpenStack, including Layer 2 network isolation, Layer 3 routing, security groups, floating IPs, and more. It addresses the shortcomings in OpenStack Neutron by replacing the default Open vSwitch (OVS) plugin with the MidoNet plugin. Thus, there are two ways to deploy Midonet:

- Either you install the OpenStack with the Midonet.
- Either you install the OpenStack and then the Midonet components.

The main differences between Midonet and CNSMO is that (i) Midonet requires an agent to be installed in the gateway nodes to allow outbound connections and (ii) that Midonet works fully integrated with OpenStack as a Neutron plugin, while CNSMO can run as a standalone service directly on the application component VMs.

4.1.3.OpenContrail

OpenContrail is an Open-Source project that is built using standard-based protocols and provides all the necessary components for network virtualization (SDN controller, virtual router, analytics engine, and published northbound APIs). It brings together a scale-out framework and physical routers and switches to scale infrastructures beyond data centre of cloud boundaries in order to offer workload mobility in a hybrid environment. OpenContrail virtualizes the network but also adds networking capabilities to the hypervisors and integrates with orchestration engines (e.g. OpenStack, VMware, Microsoft, etc). Additionally, it includes an engine for the analysis of what is going on in the network.

Juniper acquired Contrail in December 2012 and began building on its SDN capabilities. OpenContrial is the open-source version of the Contrail controller offered by Juniper and available under an Apache 2.0 license.

The key aspects of the system are:

- **Network Virtualization**: Virtual networks are the basic building block of the OpenContrail approach. Access-control, services and connectivity are defined via high level policies. By implementing inter-network routing in the host, OpenContrail reduces latency for traffic crossing virtual-networks. Eliminating intermediate gateways also improves resiliency and minimizes complexity.
- **Network Programmability and Automation**: OpenContrail uses a well-defined data model to describe the desired state of the network. It then translates that information into configuration needed by each control node and virtual router. By defining the configuration of the network versus of a specific device, OpenContrail simplifies and automates network orchestration.
- **Big Data for Infrastructure**: The analytics engine is designed for very large scale ingestion and querying of structured and unstructured data. Real-time and historical data is available via a simple REST API, providing visibility over a wide variety of information.

OpenContrail supports features such as IP address management; policy-based access control; NAT and traffic monitoring. It interoperates directly with any network platform that supports the existing BGP/MPLS L3VPN standard for network virtualization. It can use most standard router platforms as gateways to external networks and can fit into legacy network environments. OpenContrail is modular and integrates into open cloud orchestration platforms such as OpenStack, CloudStack [CloudStack], and is currently supported across multiple Linux distributions and hypervisors.

OpenContrail is presented as a software stack integrated on several cloud orchestration systems which main target is to manage the network between the overlays and the underlay. It consists of two main components: the *OpenContrail Controller* and the *OpenContrail vRouter*.

- **The OpenContrail Controller** is a logically centralized but physically distributed Software Defined Networking (SDN) controller.
- **The OpenContrail vRouter** is a forwarding plane (of a distributed router) that runs in the hypervisor of a virtualized server. The vRouter forwarding plane sits in the Linux Kernel; and the vRouter Agent is the local control plane.

OpenContrail is integrated as a Neutron plugin for OpenStack. Neutron should be configured so that it asks Contrail to provision the VM interfaces. Thus, it requires control over OpenStack in order to make Neutron interwork with Contrail.

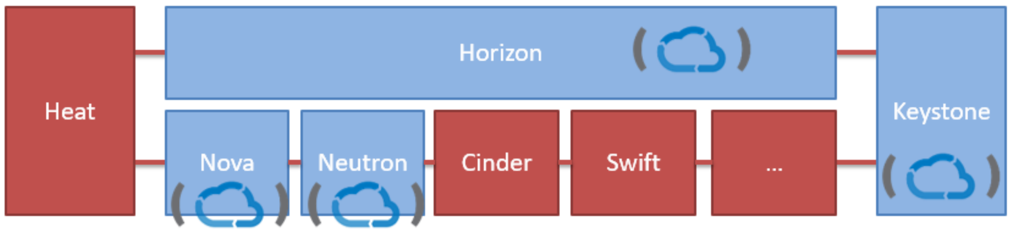


Figure 35: OpenContrail integration with OpenStack

The main difference between OpenContrail and CNSMO is that, this solution requires a certain integration with the Linux kernel to deploy the vRouter Forwarding Plane and also to install the vRouter agent in the user space. Also, all the VM traffic should be forwarded to the vRouter outside the Cloud, which may also entail a restriction depending on the topology of the cloud or the way OpenStack is configured. Moreover, OpenContrail requires deep knowledge of networking concepts and platform usage training, being burdensome for many entry level application providers.

4.1.4. Dragonflow

Dragonflow is a distributed SDN controller implementation for OpenStack Neutron supporting network services such as distributed switching, routing, DHCP and others. Dragonflow has the following objectives:

- Implement Neutron APIs using SDN principles, while keeping both Plug-in and Implementation fully under OpenStack project and governance.
- Open source.
- Lightweight and simple in terms of code size and complexity, so new users / contributors have a simple and fast ramp-up.
- Aim for performance-intensive environments, where latency is a big deal, while being small and intuitive enough to run on small ones as well.
- Completely pluggable design, easy to extend and enhance.

DragonFlow has a distributed database layer with database plugins for OVSDB, Cassandra, etc. New databases can easily be plugged into the framework. The controllers sync logical network topology databases and policy updates. The Dragonflow controllers in each compute node map this policy data and translate it into the OpenFlow pipeline into OVS. Dragonflow acts as a distributed virtual router for OpenStack clusters.

Functionality supported by agents in reference Neutron implementation is implemented as an App in the Dragonflow controller. The controller programs the flows in OVS to redirect the matching packets to the appropriate service.

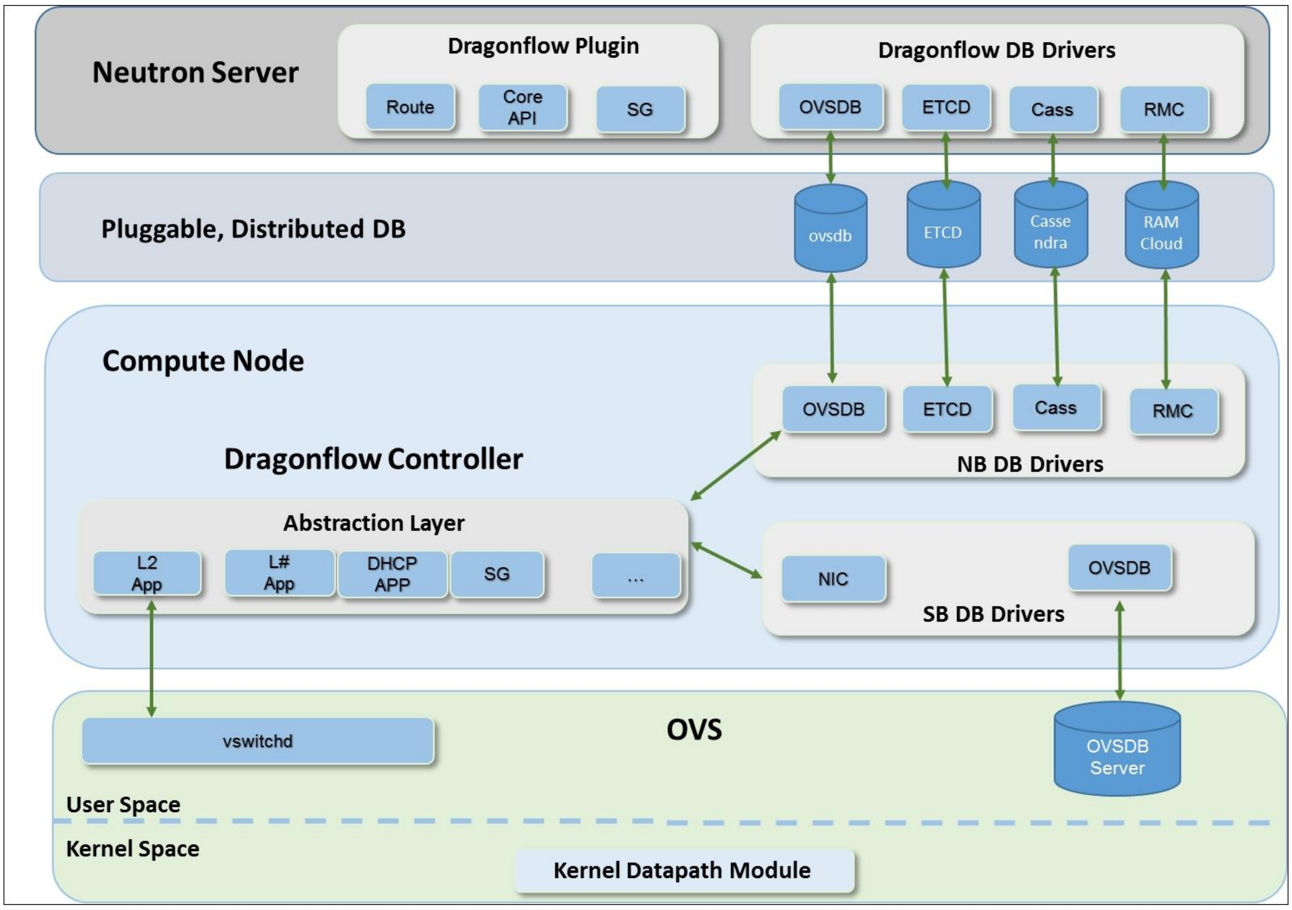


Figure 36: Neutron Dragonflow components

Dragonflow links to Kuryr, one of the OpenStack container subprojects, to bring SDN to the OpenStack containers environment. From a features perspective, Dragonflow supports L2/L3, Dynamic Host Configuration Protocol (DHCP), security groups and other advanced features.

The main difference between Dragonflow and CNSMO is that Dragonflow has been implemented as a Neutron plugin, therefore it depends completely on OpenStack forcing interoperability only between OpenStack based clouds.

4.1.5.Open Virtual Network (OVN)

Open Virtual Network (OVN) is an open source project originally launched by the Open vSwitch (OVS) team at Nicira (now part of VMware) that is supported by engineers from various commercial, private, and public entities. The objective is to develop a single, standard, vendor-neutral protocol for the virtualization of network switching functions, still based around the functionality first created for VMware virtual networks.

Using an open approach to virtual switching, any kind of workload virtualization platform – such as a hypervisor or container platform (e.g., Docker) – may reliably invoke networking functions using the same API. This not only improves scalability but enables easier live migration of virtual network components without hypervisor intervention.

OVN's main goal is to provide Layers 2 and 3 networking, which distinguishes it from general-purpose, software-defined networking (SDN) protocols and controllers.

Proponents say that OVN enables users to control cloud network resources. You can use OVN to connect VMs or containers into private L2 and L3 networks quickly and programmatically without provisioning VLANs or other physical network resources. OVN includes logical switches and routers, security groups, and L2/L3/L4 ACLs. It is implemented using a tunnel-based overlay network with protocols such as VXLAN, NVGRE, Geneve, STT, and IPsec.

OVN can also be used in OpenStack-based networks, where Open vSwitch is the most popular virtual-switch option. OVN developers are hoping their standard becomes accepted as the default network control plane for OpenStack, the open source hybrid cloud operations platform. (When the team's documents refer to a "CMS," or "cloud management system," it intends to be able to substitute this with "OpenStack.") The network facilitator in OpenStack is called Neutron, and its default control plane uses OVS.

While OVN seems a very promising networking solution to be used with OpenStack, it has focused on only OpenStack based clouds and seems that support with public service providers will not be implemented. Also, its main objective is to provide L2 and L3 tunnel-based connectivity and firewall between VMs and containers but forgets about other network services required for many application providers such as Load Balancing, DNS or QoS.

4.1.6. Other

Since some of the features of CNSMO are also provided by SD-WAN solutions, commercial and proprietary platforms can be found. However, they are mostly addressed to enterprise customers and they focus on merging different communication channels into virtual networks to connect several branches and locations of enterprise clouds.

While CNSMO could also address the SD-WAN market, the focus of CYCLONE is to address networking requirements of complex application providers. Therefore, SD-WAN platforms have been left out of this competitors' analysis.

5. OpenNaaS CNSMO next steps

The implementation of CNSMO within CYCLONE has focused on achieving the initial objectives envisioned by the project as well as addressing the network requirements imposed by the use case application service providers developed during the project lifetime. The final delivered prototype implements the identified features based on a modular design that makes the platform easy to extend with new functionalities and services. This section briefly summarizes some of the feature enhancements that could be implemented over the CYCLONE CNSMO component and gives a hint on the possible exploitation paths that are being considered to make CNSMO commercially exploitable.

5.1. Future enhancements

5.1.1. Traffic Shaping

The implementation of the CNSMO-SDN service using an SDN controller, allows a great degree of control over the network traffic. Being a purely overlay implementation, this means that within the VPN, full control over traffic flows can be achieved with the proper configuration. Currently, CNSMO uses SDN flows to allow all communication between application components within the VPN and to configure FW rules directly on each VM OVS. However, by using particular flow configurations and controlling OVS buffers, traffic shaping could be done, flow prioritization, including BW assignment per flow, creation of uni-directional flows, mirroring, multi-cast connections, grooming, etc. This can already be implemented in CNSMO by directly accessing the northbound API of the SDN controller through a 3rd party application. However, an interesting additional CNSMO feature would be to allow these functionalities from the CNSMO GUI itself or via a new network service.

5.1.2. Network Snapshot

Once a networked application is deployed including its enabled network services the application goes into an operation phase. During this phase, the deployment can change due to application scaling, VM migrations or changes in the network services performed by the administrator. CNSMO could periodically create a "snapshot" of the network including topology, IP addressing, network services configuration, etc. This would produce a historical of network "snapshots" that could be retrieved at any time and possibly be easily used in case of wanting to redeploy the application with the conditions and configurations it had at a certain point in time. This is useful in case of failures or in case the ASP wants to replicate its deployment.

5.1.3. Logging

In addition to the already implemented logging features supported by CNSMO, one of the new functionalities that can be provided by the CNSMO GUI, is enabling the access of specific deployment logs through a new Logging Tab. The idea would be to access the distributed logging system API specifying the particular deployment that corresponds to the authenticated user. Then, the user could request for Server as well as for Client logs.

5.1.4. Resilience and Fault Tolerance

CNSMO is currently based on a logically distributed architecture for network services deployment but currently implemented with a centralised CNSMO server. A possible enhancement would be to deploy the CNSMO server together with a backup instance in a different machine or even Cloud. This would switch seamlessly from the primary server to the backup one in case of failure or lack of reachability. Both instances should be synchronized at all times.

Similarly, every client instance of deployed network services could have a backup instance. But for the sake of saving resources, alternatively, and thanks to the Network Snapshot feature, a new client service instance could be deployed on the fly in case of failure, assuming the job of the failed instance.

5.1.5.Static IP addresses

Every application component needs an address to allow communication with other components. Typically, this address can be manually configured or dynamically assigned by a DHCP server. The current implementation of CNSMO uses OpenVPN to dynamically assign an IP address to each connecting client. However, sometimes you have to set a static IP Address for some VPN Clients, because they provide some server services which always must be reached at the same IP Address. A possible extension to CNSMO would be to allow the user to assign a static IP address to certain application components when access through a host name is not possible.

5.1.6.Additional Network Services

5.1.6.1. Intrusion Detection System (IDS)

An Intrusion Detection System is a system that monitors network traffic for suspicious activity and issues alerts when such activity is discovered. IDSs can be implemented via a specific hardware devide but also using a software application. An IDS that can be implemented in software can potentially be virtualised and be treated as a Virtual Network Function (VNF) in the same way that CNSMO treats its network services. There are open source implementation of IDS such as Snort [Snort] that CNSMO could use to implement such service for attack detection and together with CNSMO-SDN or CNSMO-FW, mechanisms for the mitigation of the attack could be developed such as honey pots or directly blocking the malicious traffic.

5.1.6.2. Network Proxy Server

A Proxy server is a server that works as intermediary for requests from clients seeking resources from other servers. Proxy servers can be used as content-control software, filtering of encrypted data, bypassing filters, loggin and eavesdropping, improving performance, access control or basically as security mechanisms. While most of public clouds allow the use of public IP addressed, in the case where a cloud provider uses only private addresses, a Proxy server is required to communicate internal application components with externally located components. This service could be easily impemented and configured in CNSMO using an open source Proxy server such as Squid [Squid] or Nginx [Nginx].

5.1.7.Service Chaining

Network service chaining is defined as a capability that uses SDN to create a chain of connected services. This is particularly useful when application components' traffic has to go through several network services before reaching its destination (e.g., Firewall, Proxy, IDS). Connections between service chain components may be contained within a single virtualized server or may cross network links between servers contained within the VPN. The primary advantage of network service chaining is to automate the way virtual network connections can be set up to handle traffic flows for connected services. For example, CNSMO-SDN could take a chain of services and apply it to different traffic flows depending on the source, destination or type of traffic.

5.2. **CNSMO Exploitation**

OpenNaaS CNSMO has been as a component part of the CYCLONE solution, and as such, its main target market is Application Service Providers. Nevertheless, as abovementioned CNSMO has been designed to be used also with different cloud orchestrators or as a standalone platform. This also opens the possibility to address other markets such as the SD-WAN market as further detailed in deliverable D3.4 [CYCLONE-D3.4].

The outcome implementation of CNSMO in CYCLONE has reached a Technology Readiness Level (TRL) of 7 (system prototype demonstration in operational environment), since it has been used in several use cases operationally and working over multiple clouds. In order to increase the TRL, an exhaustive quality and assurance testing should be done, to increase its robustness and performance.

i2CAT, as the main developer of CNSMO has endured an internal asset analysis where the technological assets with most potential to be commercialized are given priority for development. CNSMO has been selected in the first screening process as one of these assets. Therefore, the intention is to continue its development after the end of CYCLONE and realize a more comprehensive market and business analysis. Initial results of this analysis have already been reported in D3.4 [CYCLONE-D3.4].

In this direction, i2CAT has already started conversations with SixSq for seeking potential collaboration paths to exploit SlipStream and CNSMO integration, providing a Cloud networked solution.

6. Conclusions

This deliverable D5.3 completes the description of the CYCLONE Networking Services Manager and Orchestrator (CNSMO) platform. It has provided the rationale behind its design and implementation decisions towards fulfilling the initially defined objectives of the CYCLONE project as well as the support of network requirements coming from the identified use cases.

OpenNaaS CNSMO has evolved from its initial conception to a more lightweight, flexible and powerful tool, adapting to new network programmability trends and novel network virtualization techniques and concepts. This evolved design has been driven by the limitations detected during the first reporting period of the project and the requirements imposed by the selected use cased Application Service Providers (ASPs).

Overall, a fully-fledged networking solution has been delivered with a high degree of flexibility and control over virtualized infrastructures. The decision of providing an overlay only solution has been taken due to the fact that most application providers cannot have any access or control over physical resources, which would imply some level of interaction with the Cloud and Network provider management systems. Therefore, a fully virtual solution empowers ASPs to take control of a virtual infrastructure capable of deploying and operating a wide variety of network services such as the ones implemented during the project lifetime. The main assumption for CNSMO to be fully functional and offer the desired performance is that Internet connectivity with a certain level of QoS (bandwidth, latency) has to exist. Typically, this is already included as part of the SLAs between the Cloud Service Providers and the ASPs. If this information is provided to CNSMO, it could be used for enabling traffic shaping on the provisioned overlay network.

Moreover, the modular design of CNSMO also allows to add new network services and functions in a very easy way, making the platform robust to changes and environment requirements. The current final implementation of CNSMO done in CYCLONE has more than 8000 lines of codes, including all CNSMO components and services.

An additional important characteristic of CNSMO is that, being purely overlay, it can be deployed over any type of Cloud service provider, with the only requirement of having Internet accessibility. Tests over both private (OpenStack base) and public (Amazon EC2, Exoscale, MS Azure) clouds have been successfully performed.

Finally, in order to provide a full integrated solution, CNSMO has been implemented as a SlipStream application, allowing ASPs to enable the desired network services directly in a SlipStream recipe. This integration opens ways of collaboration between SixSq and i2CAT for potential joint exploitation.

CNSMO features have been proven extremely valuable for ASPs during the project thanks to its integration in various use cases and performed hackathons. The impact potential of CNSMO and CYCLONE in the cloud networking scenario is clear and the consortium will further explore commercialization possibilities after the end of the project

7. References

[BEACON]	BEACON - Enabling Federated Cloud Networking, http://www.beacon-project.eu/
[CloudStack]	Apache CloudStack – Open Source Cloud Computing, https://cloudstack.apache.org/
[CYCLONE-D3.4]	CYCLONE Deliverable D3.4 - Business plans and potential market
[CYCLONE-D4.5]	CYCLONE Deliverable D4.5 - Final CYCLONE secure action and resource models
[CYCLONE-D5.2]	CYCLONE Deliverable D5.2 - Specification of network management and service abstraction
[Dnsmasq]	DNSMasq, the Pint-Sized Super Dæmon!, http://www.linuxjournal.com/content/dnsmasq-pint-sized-super-d%C3%A6mon
[Dragonflow]	Dragonflow Neutron Implementation, https://wiki.openstack.org/wiki/Dragonflow
[MidoNet]	MidoNet - Open Source Network Virtualization for OpenStack, https://www.midonet.org
[Nginx]	Nginx – Flawless application delivery for the modern web, https://www.nginx.com/
[ODL]	OpenDaylight Project, https://www.opendaylight.org
[OpenContrail]	OpenContrail Architecture Document, http://www.opencontrail.org/opencontrail-architecture-documentation/
[OpenFlow]	OpenFlow Switch Specification, https://www.opennetworking.org/software-defined-standards/specifications/
[OpenNebula]	"IaaS Cloud Architecture: From Virtualized Datacenters to Federated Cloud Infrastructures", R. Moreno-Vozmediano, R. S. Montero, I. M. Llorente. IEEE Computer, vol. 45, pp. 65-72, Dec. 2012, http://www.opennebula.org
[OVS]	Open vSwitch, Production Quality, Multilayer Open Virtual Switch, http://openvswitch.org/
[Snort]	Snort - Network Intrusion Detection & Prevention System, https://www.snort.org/
[Squid]	Squid – Optimizing Web Delivery, http://www.squid-cache.org/

8. Acronyms

ACL	Access Control List
API	Application Programming Interface
ASP	Application Service Provider
B2B	Business to Business
BGP	Border Gateway Protocol
CLI	Command Line Interface
CNSMO	Cyclone Networking Services Manager and Orchestrator
CSP	Cloud Service Provider
CSS	Cascading Style Sheets
DC	Data Center
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name Server
E2E	End to End
EC2	Elasstic Cloud Computing
FW	Firewall
GRE	Generic Routing Encapsulation
GUI	Graphical User Interface
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IaaS	Infrastructure as a Service
IDS	Intrusion Detection System
IPR	Intellectual Property Rights
IPsec	IP Security
ISP	Internet Service Provider
IT	Informatino Technology
JSON	JavaScript Object Notation
LB	Load Balancer
MPLS	MultiProtocol Label Switching
NaaS	Network as a Service
NAT	Network Address Translation
NFV	Network Function Virtualization
ODL	OpenDaylight
OF	OpenFlow

OVS	Open virtual Switch
OVSDB	Open Virtual Switch Data Base
QoS	Quality of Service
REST	REpresentational State Transfer
SDN	Software Defined Networks
SD-WAN	Software Defined Wide Area Network
SLA	Servcie Level Agreement
STT	Secure Transaction Technology
TCP	Transmission Control protocol
TRL	Technology Readiness Level
URL	Uniform Resource Locator
VLAN	Virtual Local Area Network
VM	Virtual Machine
VNF	Virtual Network Function
VPN	Virtual Private Network
VXLAN	Virtual eXtensible Local Area Network
WP	Work Package

Annex 1: CNSMO API

This is an API which allows administrators to perform modifications and other operations over deployed CNSMO services.

Auth

- Authentication of user by credentials
  method: POST,
  url: **/authenticate**
  body params: { username, password }

Nodes

- Gets list of nodes (clients)
  method: GET,
  url: **/services/sdn/nodes**

- Gets list of flows
  method: GET,
  url: **/services/sdn/flows**

SDN Filters

- Gets list of blocked TCP ports of a node (client)
  method: GET,
  url: **/services/sdn/nodes/:instanceId/blockedTcpPorts**

- Block a port
  method: PUT,
  url: **/services/sdn/blockbyport**
  body params: {
  - tcp-destination-port: <destination port>,
  - ip4-destination: <$ipDestination/$CIDRMask>,
  - ssinstanceid: < client name >

  }

- Deletes a flow that contains a blocked port
  method: DELETE,
  url: **/services/sdn/nodes/:instanceId/flows/:flowId**

- Provides statistics of client's flows
  method: GET,
  url: **/services/sdn/nodes/:instanceId/flows/:flowId/monitoring**

Certs

- Generates all the certificates of the new client by its name
  method: POST,
  url: **/certs/clients/:name**

- Gets the key cert
  method: GET,
  url: **/certs/clients/:name/key**

- Gets the client cert
  method: GET,
  url: **/certs/clients/:name/cert**

- Gets the config cert
  method: GET,
  url: **/certs/clients/:name/config**

- Gets ca cert
  method: GET,
  url: **/certs/clients/:name/ca**

DNS

- Gets list of dns records
  method: GET,
  url: **/services/dns/records**

- Gets a dns record
  method: POST,
  url: **/services/dns/record**
  body params: {
    - dnsrecords: < list of string "$ip $clientName \n" >

  }

Firewall

- Adds new firewall rule to a list of rules
  method: POST,
  url: **/services/fw/rules**
  body params: {
    - direction: < "in"/"out" >,
    - protocol: < "tcp/"udp"/... >,
    - dst_port: < destination port >,
    - dst_src: < "dst"/"src" >,
    - ip_range: < "$ip/$CIDRMask" >,
    - action:< "acpt"/"rjct" >

  }

- Gets list of firewall rules
  method: GET,
  url: **/services/fw/rules**

<END OF DOCUMENT>