SUPERFLUIDITY

A SUPER-FLUID, CLOUD-NATIVE, CONVERGED EDGE SYSTEM

Research and Innovation Action GA 671566

DELIVERABLE I6.1:

INITIAL DESIGN FOR CONTROL FRAMEWORK

Deliverable Type:	Report
Dissemination Level:	CO
Contractual Date of Delivery to the EU:	31 May 2016
Actual Date of Delivery to the EU:	31 May 2016
Workpackage Contributing to the Deliverable:	WP6
Editor(s):	Haim Daniel (Red Hat) Livnat Peer (Red Hat)
Author(s):	Carlos Parada, Isabel Borges, Francisco Fontes (Altice Labs), George Tsolis (Citrix), Michael McGrath, Vicenzo Riccobene (Intel). Pedro Andres Aranda Gutierrez (Telefónica, I+D), John Thomson, Julian Chesterfield, Joel Atherley, Manos Ragiadakos (OnApp). Haim Daniel (Red Hat) Erez Biton (ALU-IL).

Internal Reviewer(s)	Michael McGrath (Intel) Pedro A. Aranda Gutiérrez (Telefónica, I+D) Gal Hammer (Red Hat)
Abstract:	This internal deliverable carries a report for gap analysis in supporting C-RAN, MEC and NFV requirements with OpenStack projects umbrella. Such properties and needs as dynamic scaling, traffic load balancing and provisioning have been put into research.
Keyword List:	Orchestration, Management

INDEX

List of Figures

List of Tables

Glossary

SUPERFLUIDITY DICTIONARY	
TERM	DEFINITION
UE	User Equipment

OSS	Operation Support System
VIM	Virtual Infrastructure Management
VM	Virtual Machine
MANO	Management And Orchestration
NFV	Network Function Virtualization
KPI	Key Platform Indicator

Table 1: SUPERFLUIDITY Dictionary.

1 Introduction

1.1 Deliverable description

The present document describes requirements towards the management and control framework. In addition to the definition of requirements, this internal deliverable introduces a draft for the architectural design of the framework.

All requirements are assigned a unique name, for future reference and own the following format: [ReqName-XX] where XX enumerates the same property requirements.

1.2 Quality review

Review Team member responsible of the deliverable: _____

VERSION CONTROL TABLE			
VERSION N.	PURPOSE/CHANGES	AUTHOR	DATE
1	I6.1 draft	Superfluidity project	May 2016

2 Requirements analysis

2.1 Our Approach

In order to tackle the challenge, our approach was split into several steps. As a first step we started by analyzing the use cases from WP2 as our input. The objective was the identification of shared attributes and the identification of common requirements that the use cases shared. After doing so, we had the next step ready – investigation of the aforementioned requirements' support in existing orchestration solutions. As a last step we need to identify the gaps between the requirements and each solution capabilities.

2.2 NFV Technical Requirements

2.2.1 Architecture

The following two figures depict the relevant ETSI NFV architectures: the main ETSI NFV and the MANO (Management ANd Orchestration). This MANO architecture highlights the management and orchestration components (dashed box), identifying in more detail the management and orchestration interfaces, and other sub-components, like Catalogues and Services/Resources Repositories.

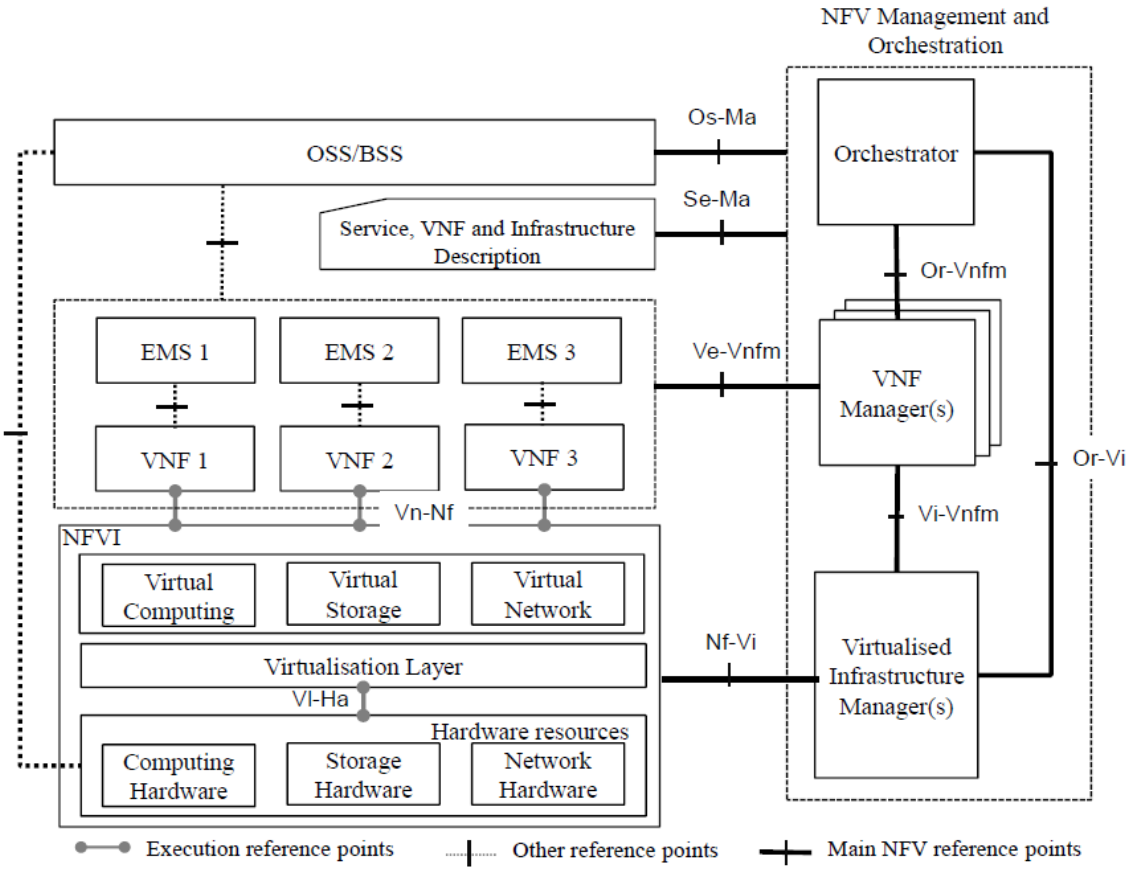


Figure 1: ETSI NFV reference architecture [ETSI-NFV].

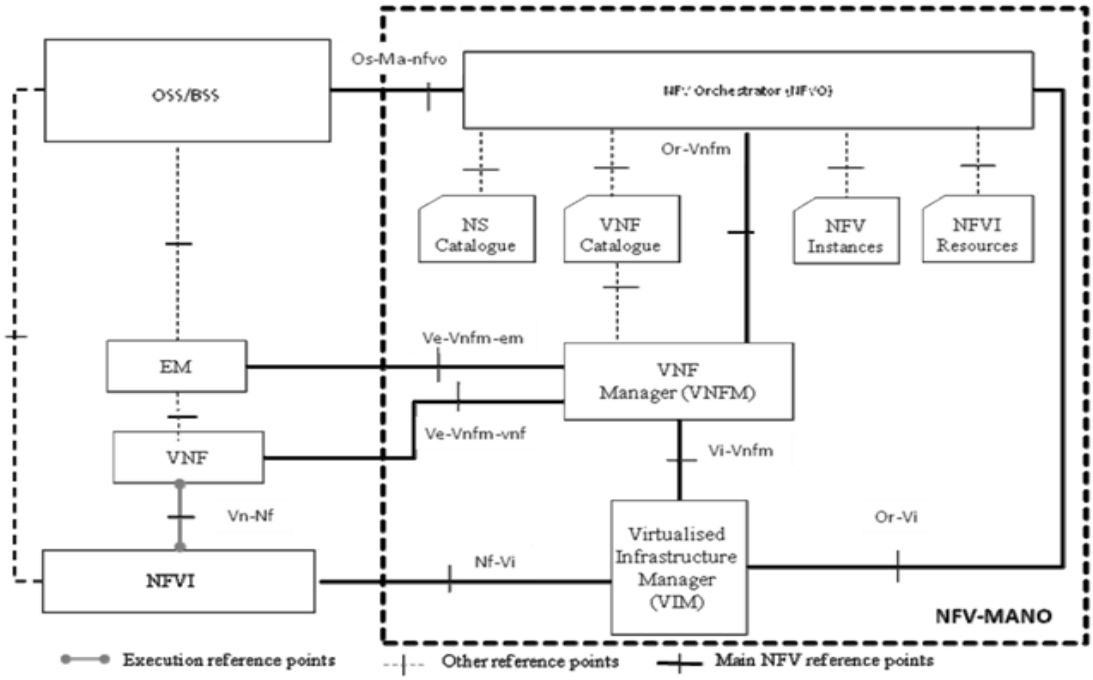


Figure 2 – ETSI NFV MANO reference architecture [ETSI-NFV-MANO].


2.2.2 Requirements

This section describes high-level technical requirements for a NFV management and orchestration system. More detailed requirements and flows can be found in Annexes 8.1.1 and 8.2.1, respectively.

[Onboarding-01] The MANO framework MUST support the on-boarding of VNFs and NSs, respectively into the NFV Catalogue and NS Catalogue, making them available for instantiation.

[Onboarding-02] The MANO framework SHOULD perform other actions than on-boarding regarding VNF and NS packages: Disable, Enable, Update, Query and Delete.

2.2.2.1 Application lifecycle

[Lifecycle-01] The MANO framework MUST support the following VNF and NS lifecycle management (LCM) operations:

- Instantiation
- Scaling
- Modification
- Termination

[Lifecycle-02] The MANO framework MUST be able to receive and process application LCM requests:

- From the OSS or a UE application
- Based on LCM rules.

[Lifecycle-03] The MANO framework MUST be able to identify the VNF/NS features they require to run. This will be the input for the decision on which location VNFs/NSs shall be provisioned.

[Lifecycle-04] The MANO framework MUST support the instantiation and termination of a running NFV or NS when required by the operator.

2.2.2.2 Application scheduling and instantiation


[Instantiation-01] The MANO framework MUST support the indication of the following virtualized resources:

- Compute
- Storage
- Network resources
- Specific hardware support

[Instantiation-02] The MANO framework MAY support the indication of the following requirements, such as:

- Latency
- Jitter
- Bandwidth

[Instantiation-03] The MANO framework MUST support the indication of physical location (PoP-DC).

[Instantiation-04] The MANO framework MUST consider cost requirements, which can be a translation of the operator's estimation for the deployment costs.

2.2.2.3 KPI's support

[Monitoring-01] The MANO framework MUST be able to collect infrastructure and service monitoring information, in order to feed KPI-based automated management and orchestration features.

2.2.2.4 Application Scaling

[Scaling-01] The MANO framework MUST be able to scale a VNF and/or NS, on OSS request or automatically based on KPIs, in order to increase/decrease the capacity.

[Scaling -02] The MANO framework MUST be able to terminate a VNF and/or NS whenever it is no longer required.

2.2.2.5 Load Balancing

As described in paragraph 4.3.1 of Deliverable D2.2, one of the VNF architecture options involves providing the load balancing function as part of NFVI. Moreover, VIMs (such as OpenStack) have the capability of managing common load balancing functions through an interface/API (OpenStack LBaaS, https://wiki.openstack.org/wiki/Neutron/LBaaS), which is also extensible to support different load balancing backends (in the case of OpenStack, through Neutron LBaaS plugins).

[LB-01] The MANO framework SHOULD be support load balancing function as part of the NFVI/VIM infrastructure. This requires integration with the application lifecycle and scaling functions.

[LB-02] The MANO framework SHOULD support standard load balancing features. OpenStack LBaaS captures these requirements at https://wiki.openstack.org/wiki/Neutron/LBaaS/requirements.

As also mentioned in paragraph 4.3.1 of Deliverable D2.2, in-network services occasionally require load balancers that operate in the so-called firewall mode: Unlike server load balancing, where the clustering can be realized using one load balancer, network service clustering requires two (logical) load balancers, one on each side of the cluster.

[LB-03] The MANO framework SHOULD ideally support firewall load balancing mode. However, this MAY require addressing gaps in NFVI/VIM (OpenStack LBaaS doesn't appear to support this case).

2.2.2.6 Service Function Chaining

The high-level architecture of Service Function Chaining (SFC), as specified by IETF (RFC 7665), was described in paragraph 4.3.3 of Deliverable D2.2. In this section we list the relevant requirements from the MANO side.

[SFC-01] The MANO framework MUST support the creation of Service Function Chains (SFCs), consisting of an ordered sequence of Service Functions (SFs).

SFs are virtual machines, or even physical devices, that perform a network function such as firewall, content filter, content cache, or any other function that requires processing of packets in a flow.

[SFC-02] The MANO framework MUST support SFCs with both simple (i.e. single SF) and complex (i.e. sequence of multiple SFs) Service Functions Paths (SFPs).

Materialisation of SFCs requires the cooperation of the NFV Orchestrator, VIM and SDN controller. The NFV-O provides the VNFFG definition (please refer to relevant requirements in this document), the VIM creates the SFC by attaching the SF VM instances to network ports and the SDN controller configures the network overlay fabric that interconnects these network attachment points.

According to the OPNFV SFC project (https://wiki.opnfv.org/display/sfc), SFC also depends on the VNF Manager:
http://artifacts.opnfv.org/sfc/brahmaputra/docs/design/architecture.html#vnf-manager

[SFC-03] The MANO VIM MUST support the attachment of SF VM instances to network ports to construct SFPs (for more details on how OpenStack aims to implement this capability, please refer to http://docs.openstack.org/developer/networking-sfc/system_design%20and_workflow.html and http://docs.openstack.org/developer/networking-sfc/api.html).

[SFC-04] A Service Function (SF) MAY actually consist of a cluster of VM instances. Each service instance cluster represents a group of like SF VM instances, which can be used for load balancing (please also see 2.2.2.5). The load balancing function MUST have the option to be sticky (i.e. sessions in progress must be sent through the same SF VM instance). The load balancing function MUST also have the option to ensure symmetric return traffic.

[SFC-05] The MANO VIM MUST be extensible to support the creation ("rendering") of SFPs in conjunction with different SDN controllers and renderers (e.g. OpenFlow, NETCONF, etc.).

The support of SFC-related requirements by the OpenDaylight SDN controller is described below:
https://wiki.opendaylight.org/view/Service_Function_Chaining:Main

[SFC-06] The MANO VIM MAY support a network overlay function that is part of the NFV infrastructure (OpenStack will provide a reference implementation using Open vSwitch).

For a complete implementation of SFC, the MANO framework would need to also support orchestration of the SFC Classifier, Service Function Forwarder (SFF) and SFC Proxy building blocks. For more information on how OpenStack aims to support these SFC functions, please refer to http://docs.openstack.org/developer/networking-sfc/ovs_driver_and_agent_workflow.html).


[SFC-07] The MANO VIM SHOULD support orchestration of SFC Classifiers. The MANO VIM MAY offer an implementation of an SFC Classifier that is part of the NFV infrastructure (OpenStack will provide a reference implementation using Open vSwitch).

[SFC-08] The MANO VIM SHOULD support the orchestration of Service Function Forwarder (SFF). The MANO VIM MAY also offer an implementation that is part of NFV infrastructure (OpenStack will provide a reference implementation using Open vSwitch).

[SFC-09] The MANO VIM SHOULD support orchestration of SFC Proxies. The MANO VIM MAY offer an implementation of an SFC Proxy that is part of the NFV infrastructure (OpenStack will provide a reference implementation using Open vSwitch).

[SFC-10] The reference implementation of the SFF, SFC Classifier and SFC Proxy (if available) SHOULD support the preferred SFC encapsulation scheme, NSH (please see IETF draft-ietf-sfc-nsh).

Please note that an initial implementation of a subset of the SFC requirements above was made available in OPNFV Brahmaputra, as a combination of OpenDaylight, OpenStack and Open vSwitch:
https://wiki.opnfv.org/display/PROJ/Project+Proposals+Service+Function+Chaining
An overview of how OPNFV Brahmaputra puts all the pieces together:
http://artifacts.opnfv.org/sfc/brahmaputra/docs/design/index.html
Further progress is apparently being made, targeting OPNFV Colorado:
https://wiki.opnfv.org/display/sfc/OPNFV+SFC+Colorado+Release+Plan

Finally, the requirements for supporting VNF Forwarding Graphs are outlined below:
https://wiki.opnfv.org/display/PROJ/Openstack+Based+VNF+Forwarding+Graph

2.3MEC Technical requirements

2.3.1 Architecture

The following Figure depicts the relevant ETSI MEC architecture. This architecture describes how a MEC environment should be organized, namely regarding the deployment of MEC App on top of a cloud environment, as well as the whole management and orchestration functions to support this operation.

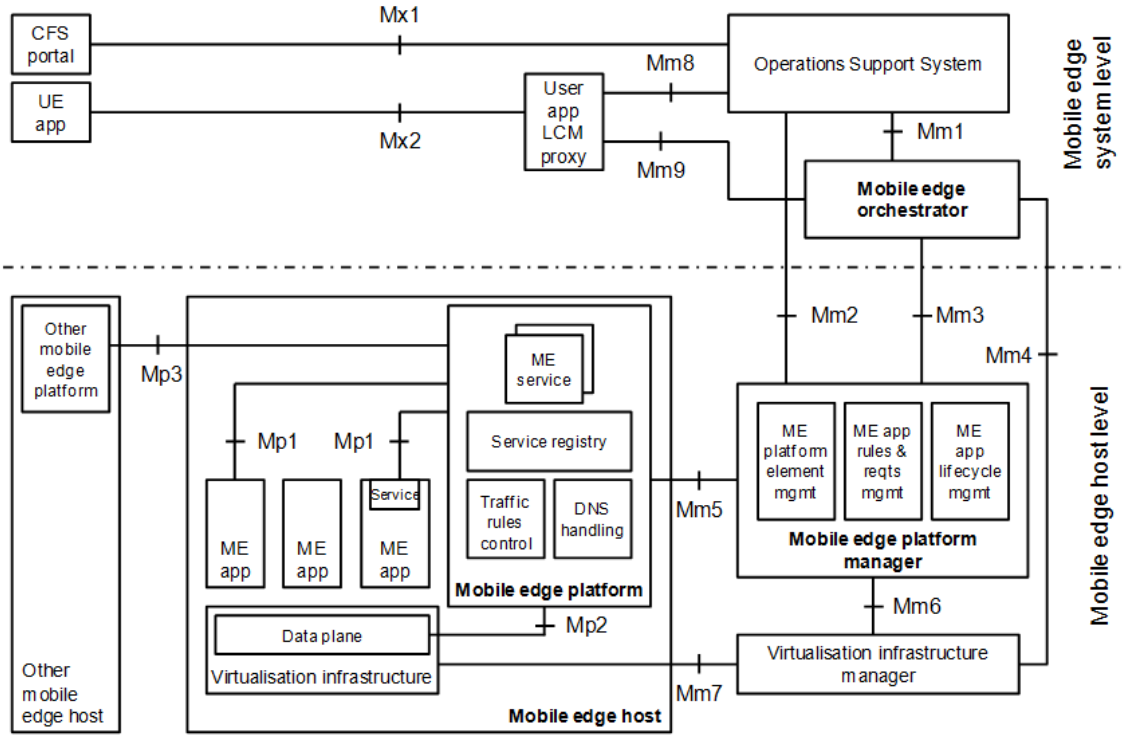


Figure 3: ETSI MEC reference architecture [ETSI-MEC]]

2.3.2 Requirements

This section describes high-level technical requirements for a MEC management and orchestration system. More detailed requirements and flows can be found in Annexes 8.1.1 and 8.2.1, respectively.

2.3.2.1 Application lifecycle

[Lifecycle-01] The management system MUST support the following application lifecycle management (LCM) operations:

- Instantiation
- Scaling
- Relocation
- Modification
- Termination

[Lifecycle-02] The management system MUST be able to receive and process application LCM requests:

- From the OSS, a third-party, or a UE application
- Based on LCM rules.


[Lifecycle-03] The management system MUST be able to identify the mobile edge features and services an application requires to run. This will be the input for the decision on which mobile edge host to provision the application.

[Lifecycle-04] The management system shall support the instantiation and termination of a running application when required by the operator.

2.3.2.2 Application scheduling and instantiation

[Instantiation-01] The management system MUST be able to deploy the application on mobile edge hosts in various locations, both in a central data center and at the edge of the Core Network.

[Instantiation-02] The management system MUST support the following deployment application models:

- One App instance per MEC Host, serving multiple users
- Multiple App instances per MEC Host, each serving a single user

[Instantiation-03] The management system MUST support the indication of the following virtualized resources:

- Compute
- Storage
- Network resources
- Specific hardware support

[Instantiation-04] The management system MUST support the indication of the following network connectivity resources:

- Connectivity to local networks
- External connectivity to the Internet
- Access to user traffic

[Instantiation-05] The management system MUST support the indication of the following latency requirements:

- Maximum
- Expected

[Instantiation-06] The management system MUST support the indication of physical location (edge).


[Instantiation-07] The management system MUST support the indication of service requirements:

- Mandatory - for MEC Apps to be able to operate.
- Optional - for MEC Apps can benefit from, if available.

[Instantiation-8] The management system MUST consider cost requirements, which can be a translation of the operator's estimation for the deployment costs.

2.3.2.3 Mobility support

[Mobility-01] The management system MUST support multiple MEC Hosts in different locations, including radio sites, aggregation points, or at the edge of the Core Network.

[Mobility-02] The MEC system MUST guarantee service continuity while the UE moves across the network (between different edges).

[Mobility-03] The MEC system MUST be able to maintain connectivity between a UE and a MEC App instance when the UE performs a handover to another cell.

[Mobility-04] The MEC system MUST be able to perform application instance relocation for MEC Apps dedicated to a single user.

[Mobility-05] The MEC system MUST be able to perform application state relocation for MEC Apps serving multiple users.

2.3.2.4 KPI's support

Virtualization of appliances increases the flexibility of service management and reduces deployment time and costs, but on the other hand it increases management complexity. This complexity can be addressed through intelligent orchestration of infrastructure resources and services. Current virtualization environments abstract the underlying infrastructure to simplify the deployment process as a consequence they also provide limited capabilities to support intelligent orchestration decisions e.g. resource aware deployments. Intelligent orchestration embraces different aspects of the service lifecycle including improved infrastructure management, intelligent deployment decisions and horizontal scaling management.

An intelligent deployment decision can be described as a deployment decision that takes into account at least two important considerations:

1. Allocation of the optimal quantity and type of resources to a workload on the most appropriate physical nodes.

2. Characterization and analysis of the target infrastructure platform to ensure both quantifiable performance and predictable behaviour of a deployed workload.

The following are the key requirements with regard to the fulfilment of service level KPIs:

[KpiTemplate-01] – The system MUST be able to dynamically define a workload deployment template to ensure that resource allocations can support required SLA's and SLO's.

[KpiScaling- 01] – The system MUST be able to determine the number and types of resources to support workload scaling in order to maintain KPI's and SLO's.

[Monitoring-01] – The MEC system MUST be able to collect infrastructure and service monitoring information, in order to feed KPI-based automated management and orchestration features.

2.3.2.5 Network Traffic control

[TControl-01] The management system must be able to provide provisioned MEC platforms with guaranteed network bandwidth.

[TControl-02] The management system must be able to rate limit the provisioned MEC platforms traffic flows.

[TControl-03] The management system must have the ability to selectively apply the traffic control on different types of traffic, and have the ability of traffic classification.

[TControl-04] Within the constraints set by the orchestration and management, an authorized mobile edge application shall be able to request the activation, update and deactivation of the mobile edge application traffic rules dynamically.

2.3.2.6 Scaling [WIP]

2.3.2.6.1 Event Handling Capacity

2.3.2.6.2 Application Scaling

[Scaling-01] – The MEC system MUST be able to scale a MEC App, on OSS request or automatically based on KPIs, in order to increase/decrease the capacity.

[Scaling-02] The MEC system MUST be able to terminate a MEC App whenever it is no longer required to serve users.

2.3.2.6.3 Containers Support [WIP]

2.3.2.6.4 Microkernels Support [WIP]

2.5Technical Requirements – C-RAN

Delivery D2.3 decomposes C-RAN into RFBs and further discuss the affinity of those RFBs. For completeness, the affinity graph between the different proposed RFBs is given in ： .

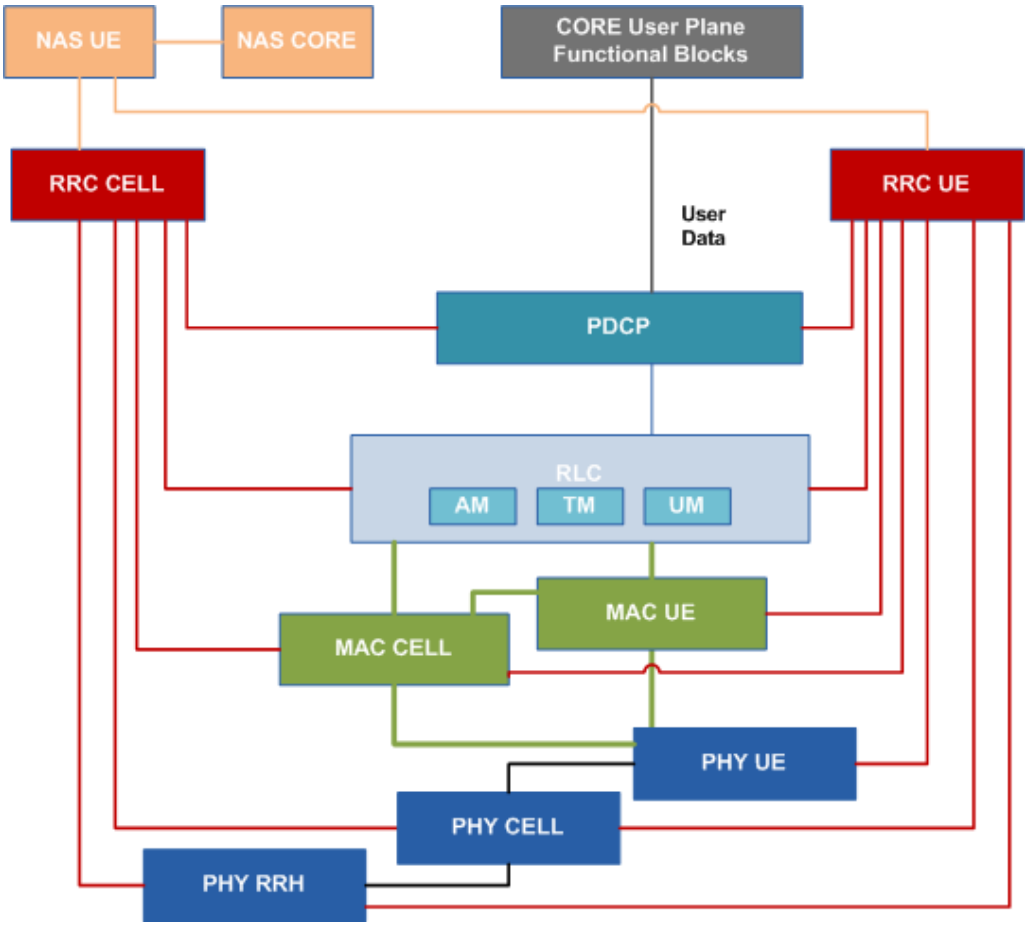


Figure 2: Affinity graph between different C-RAN functional blocks

Here, in Table 2,we further analyze the location, event handling capacity and scaling requirements from those functional blocks.

FUNCTIONAL BLOCK (FB)	EXAMPLES OF FB DECOMPOSITION	FB DEPLOYMENT LOCATION	EVENT HANDLING CAPACITY ()	APPLICATION SCALING REQUIREMENT
PHY RRH	Physical NF – not virtualized	Antenna site		Not Scalable as application
PHY Cell	all the processes executed for one cell, e.g. FFT/iFFT,	Antenna site or Front-End	every 10 ms (LTE radio frame length)	scaling decision may be reactive (based on computational latency of previous frame). Less than 10

	Modulation, Cyclic prefix	Cloud		ms requirement.
	Joint Multiuser Detection – jointly process the received signals from multiple UE from more than one RAP (MTPD, INS)		New UE could arrive or leave asynchronously. Scaling decision should be based on current computational latency and next state prediction	Scale in/out may be dependent on UE mobility. About 5-10 seconds worst case (bus, or train travelling between RAPs)
PHY User (UE)	HARQ must be sent 3 ms after receiving the frame	Front-End Cloud or EDGE cloud	new frame every 10 ms (LTE radio frame length) , but events that results capacity dependent on UE mobility.	Scale in/out may be dependent on UE mobility. About 5-10 seconds worst case (bus, or train travelling between RAPs)
	Convolution coding			
MAC Cell/Scheduling Real Time	ICIC (Intercell Interference Coordination)	Front-End Cloud or EDGE cloud	every 10 ms (LTE radio frame length), Works with a cluster of RAP's, scaling events not coming in peaks.	number of minutes in most cases, dependent on UE mobility. About 5-10 sec
	link adaptive part	antenna site	Dependent on current antenna measurements, need to be executed locally on antenna site, latency sensitive	10 ms If implemented in proactive fashion could be less time sensitive
MAC User (UE)	UE Power control	EDGE cloud	LTE case it can happen maximum 1000 times within a second per ue. capacity is Number of users in 1ms	not coming in peaks, 5-10 seconds worst case
RLC	It includes processes related to segmentation/concatenation of PDCP PDUs	EDGE cloud	*depends on the mobility and traffic intensity of UE. For the EDGE cloud*	number of minutes to scale

	based on information exchange with MAC and PDCP. Several modes are supported: Transparent, Acknowledged and Unacknowledged. Each case could be a separate FB		*slow change in number of ue associated with it.*	
PDCP Packet Data Convergence Protocol	transfer of user plane data, transfer of control plane data, header compression, ciphering, integrity protection.	EDGE cloud or Central cloud	Depends on ue activity levels, would change through the day in predictable manner (peak in the morning, less activity in the night, etc)	scaling not strict time constrained, and predictable. number of times in a day
RRC Cell		EDGE cloud or Central cloud		
RRC User (UE)	Handover UE measurements reporting, QoS management, paging	EDGE cloud or Central cloud	about 30% of UE are in the handover state, so with central deployment number of scaling events in a day	scaling not strict time constrained, number of times in a day
NAS User (UE)	It refers to the user procedures related to signaling between the UE and MME	EDGE cloud or Central cloud	Asynchronous, depends on user mobility. Because of deployment on central cloud slow change in number of the users in the whole network	scaling not strict time constrained, number of times in a day
NAS Core	MMEs load balancing, MME overload control, GTP-C signaling load control…	EDGE cloud or Central cloud		

Table 2: C-RAN RFB requirements

2.6 Generic Technical Requirements – NFV vs. MEC

NFV	MEC
NFVO only orchestrates Network Services (NS), not VNFs (for those are VNFMs)	MEO orchestrates MEC Apps (MEC has no combination of MEC Apps as NSs combine VNFs)
NFV has no services platform to provide services	MEC has a service platform to provide services to Apps, which must be managed (access, auth, etc.)
The deployment details of NSs (e.g. location) can be decided by the NFVO, but also by the VNFM	The deployment details of a MEC App is only determined by the MEO
Mobility issues are not very relevant (although in some cases may arise)	Mobility issues (state movement) are relevant
Location issues are not always relevant (although in some cases may happen)	Location issues are always relevant

Table 3: NFV vs MEC comparison


3 State of the art analysis

3.1 OpenStack

3.1.1 OpenStack Virtual Infrastructure Management (VIM)

This section provides a summary of the capabilities exposed by the virtual infrastructure which are relevant to the orchestration layer.

3.1.1.1 Network Traffic Control

Neutron has become OpenStack's 'networking as a service' de facto project, and provides multiple networking services, QoS is being one of the key features provided. The supported traffic control requirements in Mitaka release are rate limiting answering [TControl-02], and the dynamic activation/deactivation upon request [TControl-04]. However, on the downside the missing features are bandwidth guarantee [TControl-01] and having a more mature traffic classification capability [TControl-03] (e.g. layer 7), with the latter becoming an active discussion at the latest OpenStack summit.

3.1.1.2 Scheduling parameters

In order for the orchestration layer to be able of making a 'smart' scheduling decision, the VIM has to expose the required set of parameters for the orchestrator to take into an account. However, at this point in time, most of the aren't supported. On the upper side - requirements [AppSched-05] (description of the virtualized resources) can be satisfied by the usage of templates provided by such projects as Heat and Tacker as well as [AppSched-06] (Required network connectivity description). However, on the downside requirement [AppSched-08] (Physical location requirements) is hardly fulfilled. The possible solutions to accomplish that can be by made by the usage of OpenStack's Nova (compute project) regions and cells accompanied by custom Nova scheduler filters, a solution we're planning to research and experiment with in the following time frames.

3.1.1.3 Mobility support

While the OpenStack Nova (compute) project provides support for a subset of functionality for migrating VM instances from one physical host to another, it lacks some of the properties required for full mobility support: [Mobility-01], [Mobility-02]. The user of the migration feature in its current form cannot specify the physical host on which the VM will be migrated, as this decision is left out


to the scheduler. In addition to it, this process does not assume that the VM instance has sufficient storage available on the target host, and potentially can fail.

3.1.1.4 KPI Support

A KPI is a metric used to evaluate factors that are crucial to the performance of a workload or service. Operationally KPIs act as a simple set of indicators to measure data against -- a sort-of service success gauge. In order to appropriately monitor and measure KPIs requires quantitative and qualitative metrics. These metrics are typically captured through the use of telemetry providing both platform and service level data.

Current service orchestration approaches are based on the use of pre-defined configurations for the node(s) hosting the workloads. The Orchestrator then requests instantiation of the pre-defined configuration to bring the workload into service on specific hardware platform, for instance through usage of pre-compiled deployment templates (i.e. OpenStack Heat Orchestration Templates (HOT), TOSCA descriptors, etc.). These templates are managed by orchestration platforms through the use of catalogues, (for instance, OpenStack Murano project can be used to store and manage HOT templates for OpenStack Heat). However, this approach does not scale efficiently. As the number of different services to be supported by the platform increases as well as the granularity of service specific KPIs (Key Platform Indicators) and SLOs (Service Level Objectives) it results in a huge number of deployment templates to supported deployment of services. A more effect approach maybe based around the use of dynamic template definitions at deployment time to meet specified KPI's as described in section 4.1.1.

3.2 Cloudband

Cloudband management system is based on two main components, VNFM (VNF management) and NFVO (NFV orchestrator). In the following we would focus on the VNFM.

Cloudband VNF management system is mostly based on OpenStack and open source services. Specifically, on top of OpenStack main projects (NOVA, Neutron, Cinder and Glance) Heat is utilized for VNF deployment and resource allocation. To further allow VNF lifecycle management we utilized Mistral workflow engine that operates in conjunction with Heat. We note that the selection of a workflow engine for a generic VNF management has been identified as an efficient approach in terms of providing a quite broad generic management capabilities and with relatively low complexity (Odini, Marie-Paule. "Short Paper: Lightweight VNF manager solution for virtual functions." Intelligence in Next Generation Networks (ICIN), 2015 18th International Conference on. IEEE, 2015).

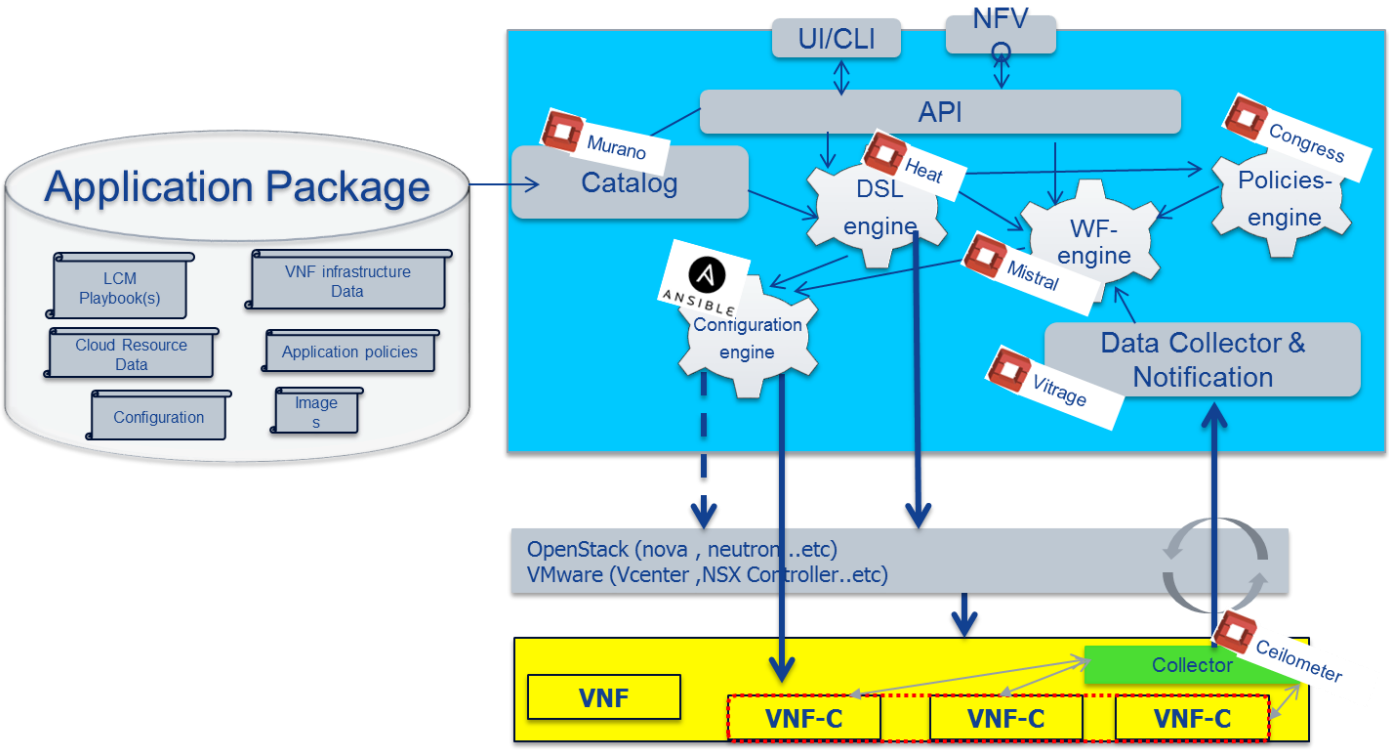


Figure 3: Openstack based generiv VNF management system

Figure 3 depicts the architecture for the VNF management system. As indicated, the architecture is based on OpenStack services, such as: Heat, Mistral, Murano, Ceilometer, Vitrage and possibly Congress. In addition, it utilizes Ansible as an open source configuration management. This architecture can support all of the operations that are required for a VNF lifecycle management, including deployment, monitoring, scaling healing and termination (as depicted in Figure 4).

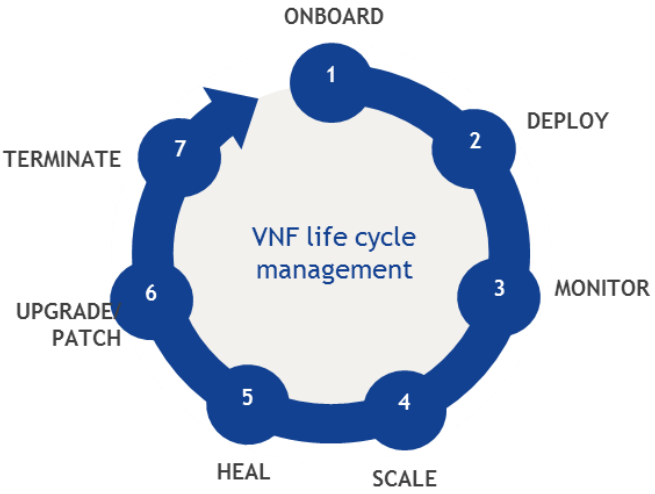


Figure 4: VNF lifecycle operation

For example, Deployment takes place once the onboarding process is complete. Deployment entails ensuring that the newly-introduced application is deployed with its name and the correct environment, on the correct VMs, with the right IPs, etc.

After the onboarding process is complete, the second LCM stage—Deployment takes place (Figure 5).

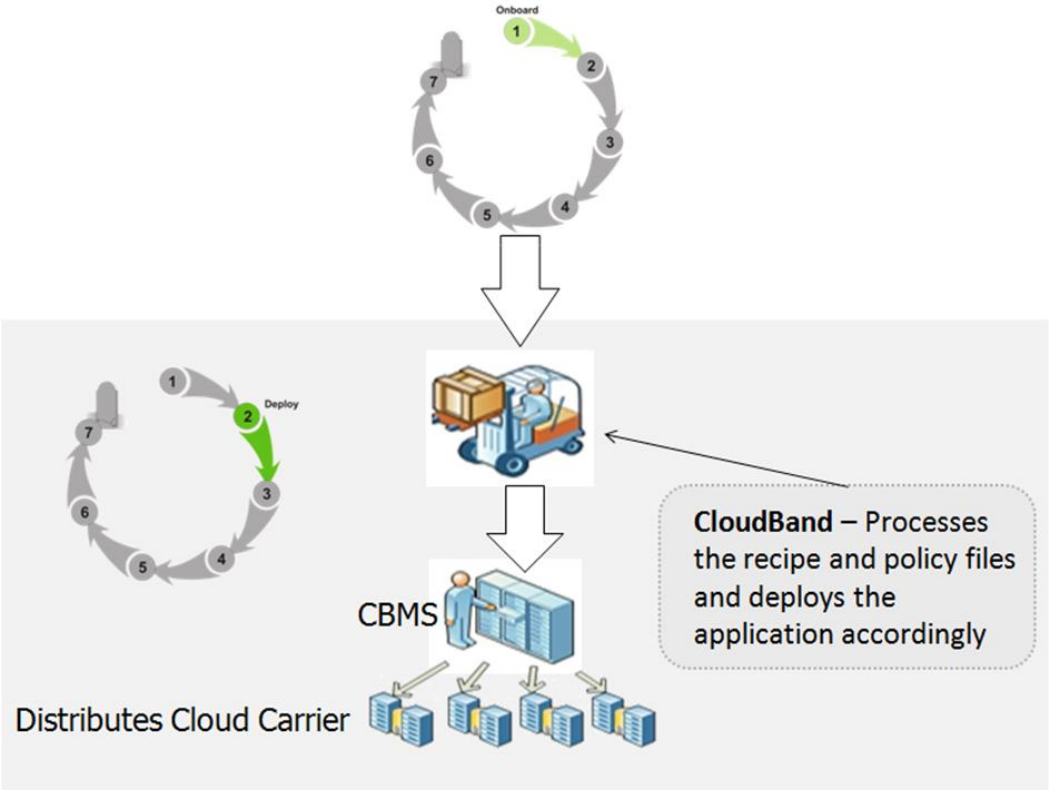


Figure 5: The deployment workflow

Only the customer user can deploy applications. There are two ways to deploy:

- From the Catalog (add application blueprint to the Catalog specified in onboarding)
- Direct deployment of Deploy Stack Directly on OpenStack Node

The HOT template is validated by OpenStack during deployment. No validation is performed when the HOT template is onboarded.

After an application is deployed, a service will be created in the MY CLOUD > DEPLOYMENTS. Under the service the customer user can see the stacks of the application.

For each deployment, a job will be created.

For the deployment to succeed, one should ensure that the Hot is valid and that all the required resources for the stack are on the node (for example, the image).

3.3OpenMano

OpenMANO implements components from the ETSI NFV MANO stack. Currently, the situation with regards to the requirements outlined in Section 2 is the following:

3.3.1 Network Traffic Control

OpenMANO supports the definition of link parameters in the VNFD descriptor as well as in the Network Scenario Descriptors (NSDs). They include the type of link (point-to-point, LAN-type, etc.) as well as quality of service parameters

3.3.2 Scheduling parameters

Currently, OpenMANO does not support scheduling internally. However, the OpenMANO component in the OpenMANO project controls a VIM where NFV services are offered including the creation and deletion of VNF templates, VNF instances, network service templates and network service instances using the openmano API. This can be used by other components to implement scheduling.

3.3.3 Mobility Support

Currently, OpenMANO concentrates on creating NFV-based scenarios. As such, the VNFDs are static and do not provide hooks to define mobility for the virtual machines (VMs) that are included in a VNFD.

3.3.4 KPI Support

OpenMANO offers a northbound interface, based on REST (openvim API), where enhanced cloud services are offered including the creation, deletion and management of images, flavours, instances and networks. The implementation follows the recommendations in NFV-PER001.


4 Management and Orchestration Design

This section intends to identify and describe the different available options regarding cloud infrastructure, cloud infrastructure management and orchestration. We also discuss the pros and cons and the best approaches to be followed by the project.

4.1 Cloud Infrastructure

The cloud infrastructure is the basis of the emerging cloud technology. It allows to create isolated virtual entities, with compute, storage and networking capabilities, appearing as if they were physical machines. The use of hypervisors (e.g. KVM, ESX) is still the most common virtualization technology. However, container-based technologies (e.g. Kubernetes, Dockers*) are getting momentum. ETSI NFV refers to this as NFV Infrastructure (NFVI); we will use this term from now on.

Independently of the virtualization technology in use, some architectural aspects need to be discussed and decided, in the context of the project, in order to find the best approach that fits with our requirements. Superfluidity shall support two different types of services: network functions (e.g. eNB, EPC) and applications (e.g. MEC).

4.1.1 Dynamic Definition of Service Deployment Templates to Support KPIs

In order to provide the intelligent of the orchestration process, automation is a key requirement to determine the best composition of quantity and types of resources to be allocated to a service according to its KPIs and SLOs and as a result to changing workload conditions due to user interactions. Providing automated and performant deployments and scaling decisions will enable both support for performance requirements and increased platform density in a scalable manner, which will result in increased efficiency in the management of features exposed by the platform and the infrastructure resources.

In the context of the Superfluidity project, the design and implementation of an automation framework is being developed in order to automate some aspects related to the generation of actionable insights for orchestration. The main goal, according to the premise above, is to automatically define a set of rules that can be interpreted by an orchestrator in order to make intelligent decisions with respect to the quantity and type of resources to be allocated to a service hosted by a VIM (Virtual Infrastructure Manager). To achieve this goal, there are three steps that must be automated and integrated in order to reduce the complexity of rules generation process:

- Execution of experiments which implement defined stress tests for VNF's and application workloads based on specific scenarios and deployment configurations of interest;
- Data collection using embedded telemetry systems and the automated discovery of infrastructure elements which can support execution of a workload;

- Data analysis to extract orchestration insights from the data and generate deployment rules to be used by orchestration platforms to make intelligent deployment decisions.

The general goal will be to generate an optimized version of a deployment template and the storage of the template into the main template catalogue used within the project. The optimal template could be given as the composition of the deployment configuration parameters and the related values to be used at deployment time. They can be determined through the adoption of a data analytics approach. For a given a service to be deployed, along with a list of KPIs/SLOs to be satisfied and a default deployment template, an experimental protocol can be defined and automated. Data analytics can be used to find potential mappings between the service specific KPIs/SLOs and the different deployment configurations explored by the experimental protocol.

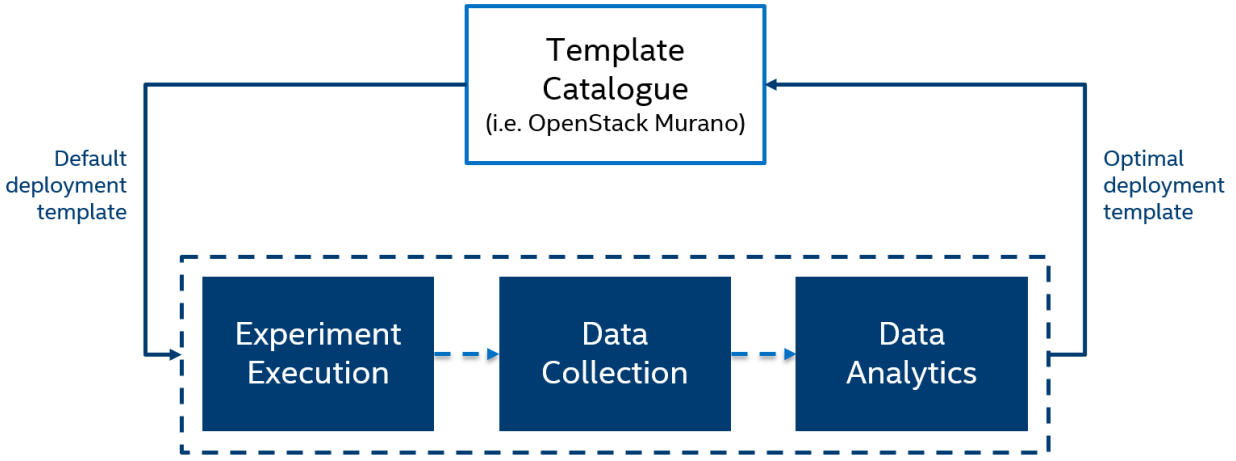


Figure 6: General workflow of the proposed solution

Horizontal scaling is also very important from an intelligent orchestration perspective: horizontally scaling a service involves either increasing or decreasing the number of resources to be used at runtime to ensure KPIs and SLOs compliance considering dynamic variations of the workload and its usage profile.

In order to explore the effects of horizontal scaling on a platform and on service performance, a similar workflow to the one discussed above can also be used. This would be supported by a specific experimental protocol and data analytics applied to the data collected during experiments.

The goal would be to find a mapping between the supported workload of the service and the number of active instances to be instantiated to support the workload. The expected output then would be the number of instance to activate with respect the current workload to be supported in order to satisfy the SLOs.

4.1.2 Option 1: One NFVI per Service

The easiest way to support different services is to use a separated cloud infrastructure (i.e. servers, storage, network) (see Figure 7Error! Reference source not found.). However, this leads to an nefficient use of resources, as there are no synergies between similar infrastructures. Furthermore, for an operator, the management effort is considerably larger, as isolated silos need to be built.

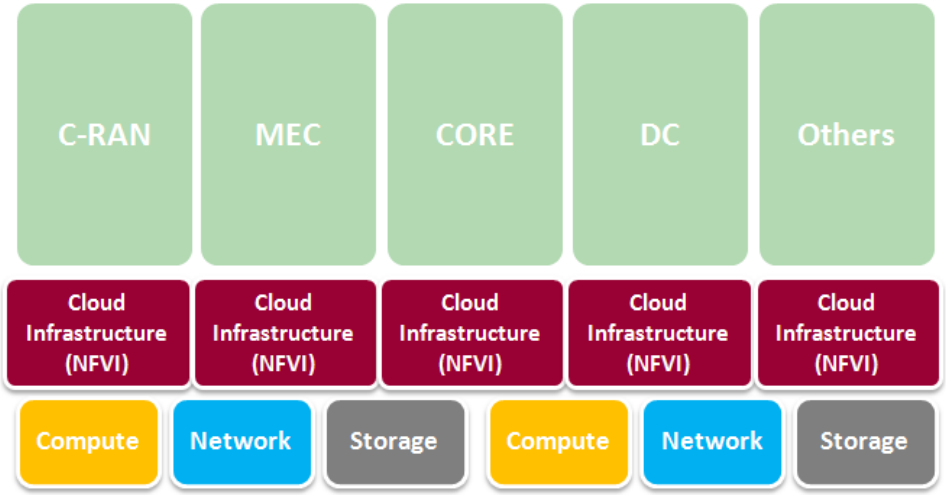


Figure 7: Option 1: One NFVI per Service.

Conclusion: *Inefficient and complex*

4.1.3 Option 2: Common NFVI for all Services eventually locations

To increase efficiency and reduce complexity, it is preferable to have a common infrastructure, which can be used to hold all kinds of services, eventually even in multiple locations (see sections below). For this to be possible, it is required to ensure that all services can rely on similar infrastructure standards. After some discussions among service specialists, we were not able to identify any service specificities that prevent this approach. For this reason, it seems that the best strategy is to have a common cloud infrastructure (NFVI) for all services. This model increases efficiency and simplifies management. The Error! Reference source not found. depicts this view.

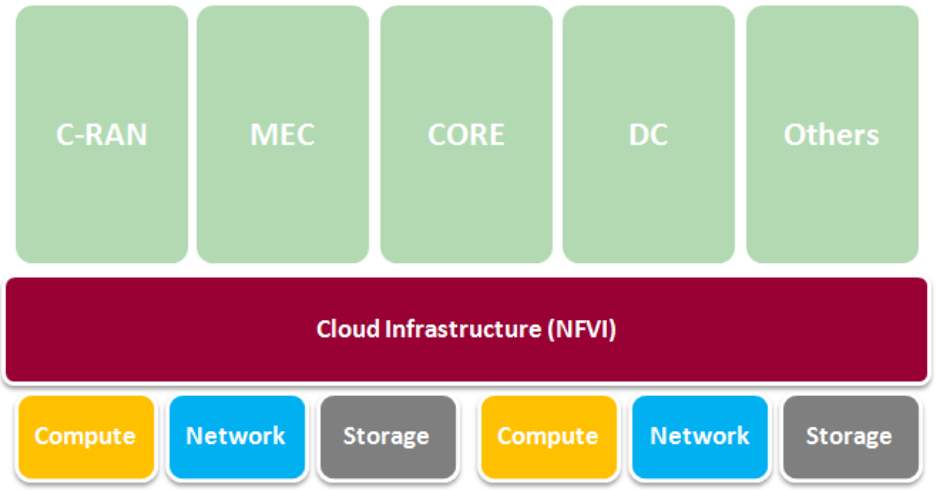


Figure 8: Option 2: Common NFVI for all Services and eventually locations.

Conclusion: *Preferred*

4.2 Cloud Infrastructure Management

To manage a cloud infrastructure (NFVI) a controller/manager is required. This manager is responsible to interact with the hypervisors and provide users with the capacity to manage (create, remove, update and delete virtual resources (compute, storage, network). ETSI NFV refers to this as *Virtual Infrastructure Management* (VIM); we will use this term from now on. Today, the reference for this component is the open source OpenStack solution. Although there are others like OpenVIM, OpenStack is clearly a *de facto* standard.

Assuming that a common NFVI can support all services (see section above), it is important to define the strategy to efficiently support the management of resources spread across a large number of datacenters (core and, especially, edges). As described below, there are several options, each with pros and cons.

4.2.1 Option 1: One local VIM per NFVI

The simplest and most common approach is to use one VIM per NFVI, i.e. one manager/controller per cloud infrastructure (datacenter). Following this approach, the VIM function is deployed locally on the datacenter (e.g. edge) and manages all NFVI resources located there (see Figure 9Error! eference source not found.). This has the advantage of being a well-known and resilient approach, as inter-datacenter connectivity is not required. However, it has two main disadvantages. Firstly, this may lead to a large number of VIMs, making the life of the upper Orchestration layer more complex, as it needs to interact with multiple VIMs endpoints. Secondly, the use of multiple VIMs may prevent the use of some capabilities like “VM live migration” among different locations, which may be an important feature. Up to now, it is not clear whether this feature is required and has

advantages when compared to other models (e.g. service migration at Orchestration level). Some work still needs to be done to evaluate this.

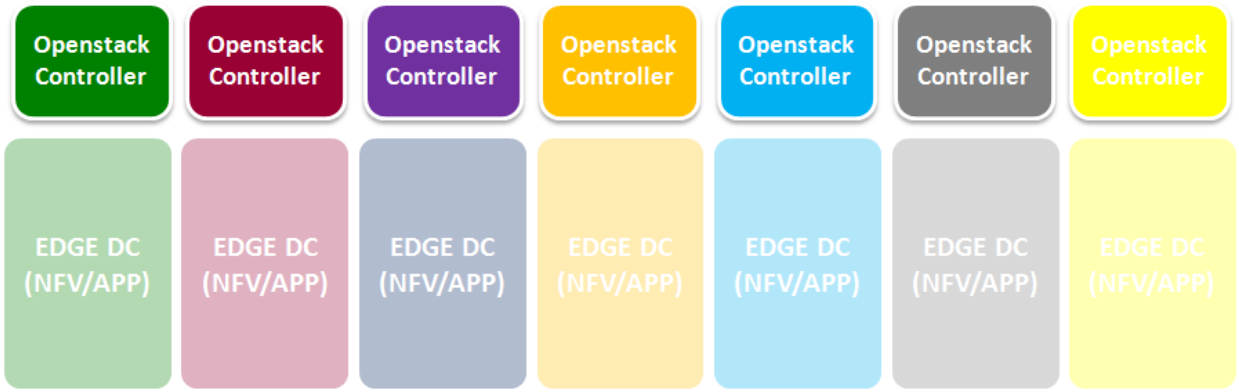


Figure 9: Option 1: One local VIM per NFVI.

Conclusion: *Acceptable*

4.2.2 Option 2: Single centralized VIM for all NFVIs

The use of a single VIM for all NFVIs located in multiple datacenters (core and edges) is another option to consider. In this case, a single centralized VIM is able to manage all resources located in different locations, providing an external view of a single and federated large datacenter (see Figure 10Error! Reference source not found.). The different locations can be identified, when needed, based n regions. This approach has the advantage of simplifying the life for the Orchestration layer, as it has a single VIM as endpoint, where all resources can be requested. On the other hand, it allows the use of features like “live migration”, only possible within the same VIM domain, as referred above. However, it has also some disadvantages. From one side, it makes the VIM operation more complex, as it needs to manage a large amount of resources and locations. Furthermore, there may exists some limitations on the number of managed resources. Finally, the manager/controller is no longer local to the NFVI, resulting in traffic increase and delay for the actions to be taken, making also appropriate connectivity a requirement. Anyway, today this seems to not be a hard limitation, as services today already are highly connectivity dependent.

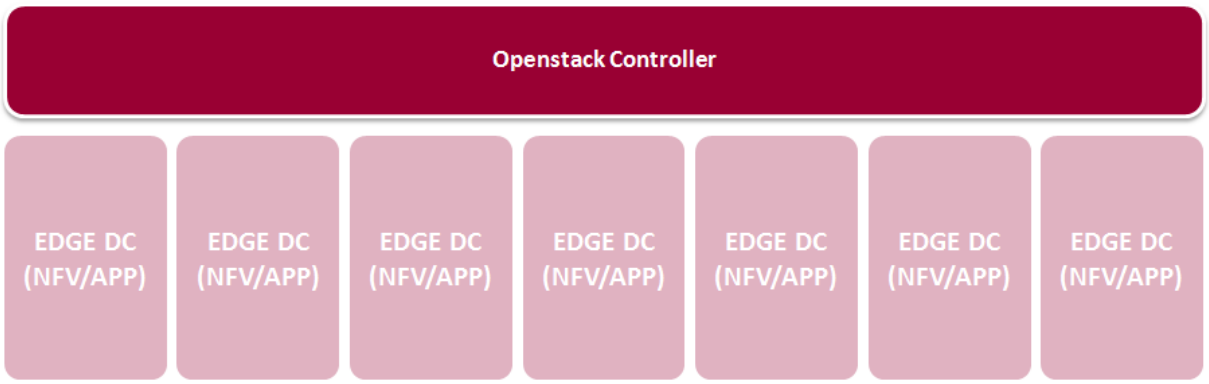


Figure 10: Option 2: Single centralized VIM for all NFVIs.

Conclusion: *Acceptable*

4.2.3 Option 3: Hybrid Option 1 and Option 2

There is still a compromise approach between the two options referred above. In this hybrid approach, multiple datacenters (NFVIs) are grouped into zones and managed by a single VIM (see Figure 11ERROR! REFERENCE SOURCE NOT FOUND.). This option intends to take advantage of the best of oth worlds, overtaking some limitations. The group sizing needs still to be defined, but it may depend on a case by case. Compared to option 1, it reduces the number VIM endpoints to a more reasonable number, making the Orchestrator's task easier. On the other hand, it allows users to take advantage of features like "live migration" within the same zone; if groups are properly defined, it can lead to a good tradeoff. Compared to option 2, it can reduce overall complexity and overtake any resource management limitations. In this option, the manager is also no longer local to the NFVI; however, this seems not today a hard limitation as stated above.

Note that in the two extreme cases, this solution is similar to the previous options. If groups are very small, we may lead to groups of a single NFVI, meaning Option 1. On the other extreme, large groups may lead to a single group, meaning Option 2. With this flexibility, it is reasonable to consider this the best option.

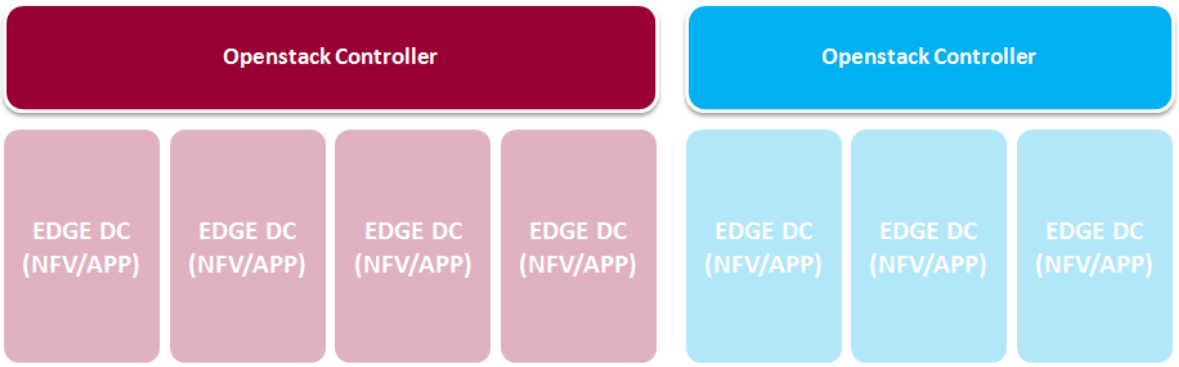


Figure 11: Option 3: Hybrid Option 1 and Option 2.

Conclusion: *Preferred*

4.3 Cloud Management and Orchestration

Running on top of Cloud Management, the Orchestration layer is responsible to build complex services by combining and interconnecting the required pieces, on the right locations. Among other things, the Orchestration is able to select the appropriate resources in the right place, based on predetermined constraints. For this, it requires interaction with VIMs. However, as Orchestration can be a very complex task, it will not be simply a single piece, but a set of them, dealing partially with the Orchestration tasks. This section discusses some Orchestration strategies and how do they map to the Infrastructure Management (VIMs).

4.3.1 Option 1: One Orchestrator for all Services and locations

A simple approach to orchestrate all Services in all locations is to use a single Orchestrator. One multi-purpose Orchestrator can deal with all resources and has the advantage of having an overall view of all services, taking eventually advantage of some synergies from that. This model is depicted in Figure 12Error! Reference source not found.. However, the Orchestrator needs to deal with ervice specificities and it may be hard to have a common Orchestrator to handle all that. On the other hand, in real world, different vendors provide different Services, and it is very likely each one brings its own Orchestration for his particular Service. In that case, this solution can be hard to achieve, both technically and commercially.

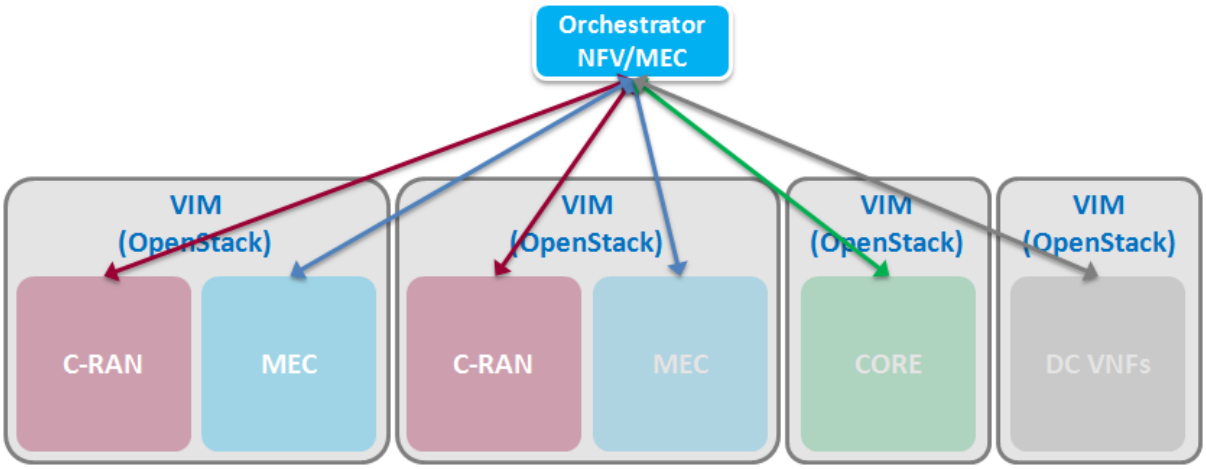


Figure 12: Option 1: One Orchestrator for all Services and locations.

Conclusion: Non-realistic

4.3.2 Option 2: One Orchestrator per Service

Another approach is to use different Orchestrators to comprise the overall Orchestration layer. In this case, each Orchestrator is in charge of part of the overall Orchestration tasks (see Figure

13ERROR! REFERENCE SOURCE NOT FOUND.). As state above, a dedicated Orchestrator per Service seems a ealistic approach; however, other options may also be reasonable. For example, if a vendor provides the C-RAN and the Core, maybe a single Orchestrator can take care of both. Similarly, if an operator has multiple C-RAN vendors, which is common, different Orchestrators may be needed for the same service, one for each particular vendor.

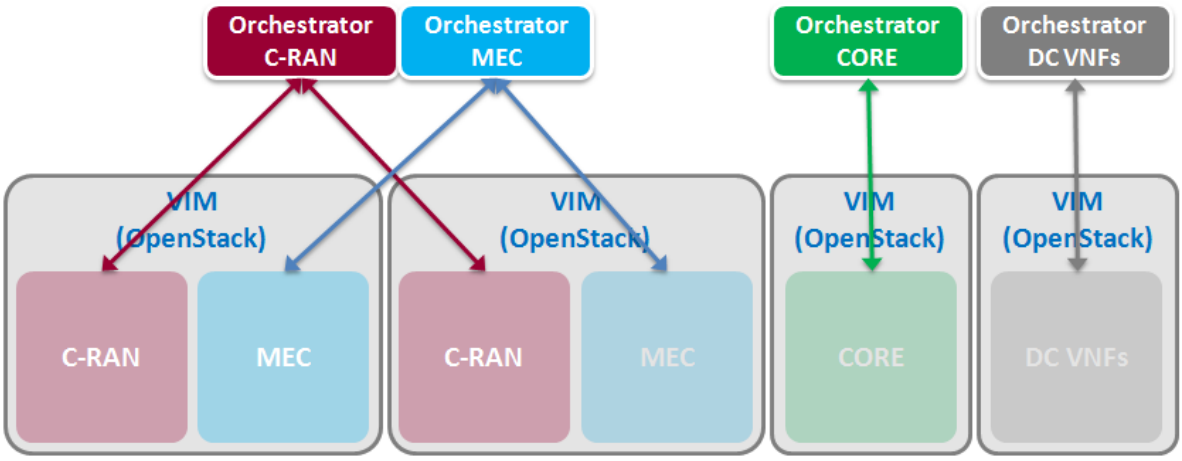


Figure 13: Option 2: One Orchestrator per Service.

Conclusion: Preferred

4.4Orchestration Layer

As described in the section above, the Orchestration layer may be composed by multiple Orchestrators, each of them devoted to a particular part of the Orchestration tasks/domains, namely to a particular Service (and from a particular vendor). In this situation, it is relevant to discuss how these Orchestrators can talk to each other and how an operator can have a global view and control about the Services. This section discusses the available options and interfacing models that can be used for this purpose.

4.4.1 Option 1: Northbound and Southbound Interfaces

One possible option leads to the creation of a Top Orchestrator, which integrates all the Service Orchestrators. In this case, Service Orchestrators interact with the Top Orchestration using a Northbound interface (Southbound interface from the Top Orchestrator perspective). For this option, the interaction between Service Orchestrators is not required, as everything is coordinated via the Top Orchestrator. Here, the Operator will own the Top Orchestrator and must integrate it with all Service Orchestrators. The **Error! Reference source not found.** depicts this hierarchical odel.

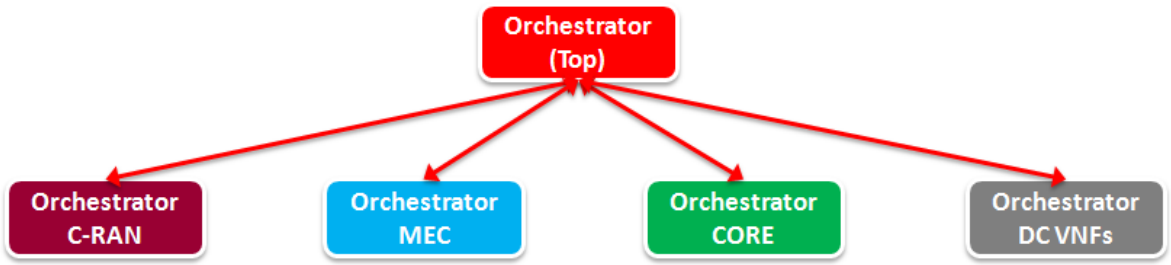


Figure 14: Option 1: Northbound and Southbound Interfaces.

Conclusion: *Acceptable*

4.4.2 Option 2: Eastbound and Westbound Interfaces

Another option is to make Service Orchestrators to integrate with each other's, using East and Westbound interfaces, in order to build an overall service. In this case, Service Orchestrators need to potentially integrate with all (or at least some) of the other Service Orchestrators, making things apparently more difficult and complex (more integrations required – partial/full mesh). On the other hand, the operator does not have any central Orchestration point where he can control the whole system, but instead multiple Orchestrations, one per Service, which in some cases, may difficult obtaining a global orchestration view. **Error! Reference source not found.** Depicts this peer-o-peer model.

Figure 15: Option 2: Eastbound and Westbound Interfaces.

Conclusion: *Difficult*

4.4.3 Option 3: Hybrid Option 1 and Option 2

There is still a compromise approach between the two options referred above. In this approach, a Top Orchestrator integrates all Service Orchestrators (interfaces Northbound and Southbound) in a central Orchestration point. This approach reduces the number of integrations required and provides to the operator an overall Orchestration view. Additionally, Eastbound and Westbound interfaces can be used in order to improve the efficiency of the system, in cases where the integration is preferable between Service Orchestrators. The number of interactions among Service Orchestrators and between Service Orchestrators and the Top Orchestrator will depend on the particular cases. The **ERROR! REFERENCE SOURCE NOT FOUND.** depicts this hybrid model.

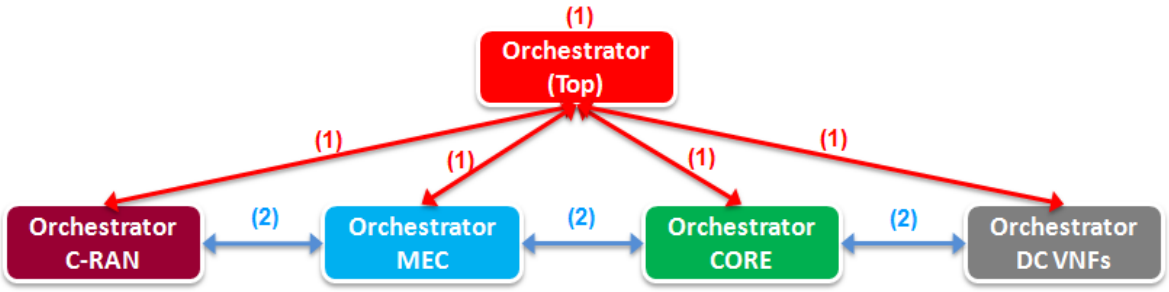


Figure 16: Option 3: Hybrid Option 1 and Option 2.

Conclusion: *Preferred*

5 Management Tooling

5.1 MicroVisor Orchestration

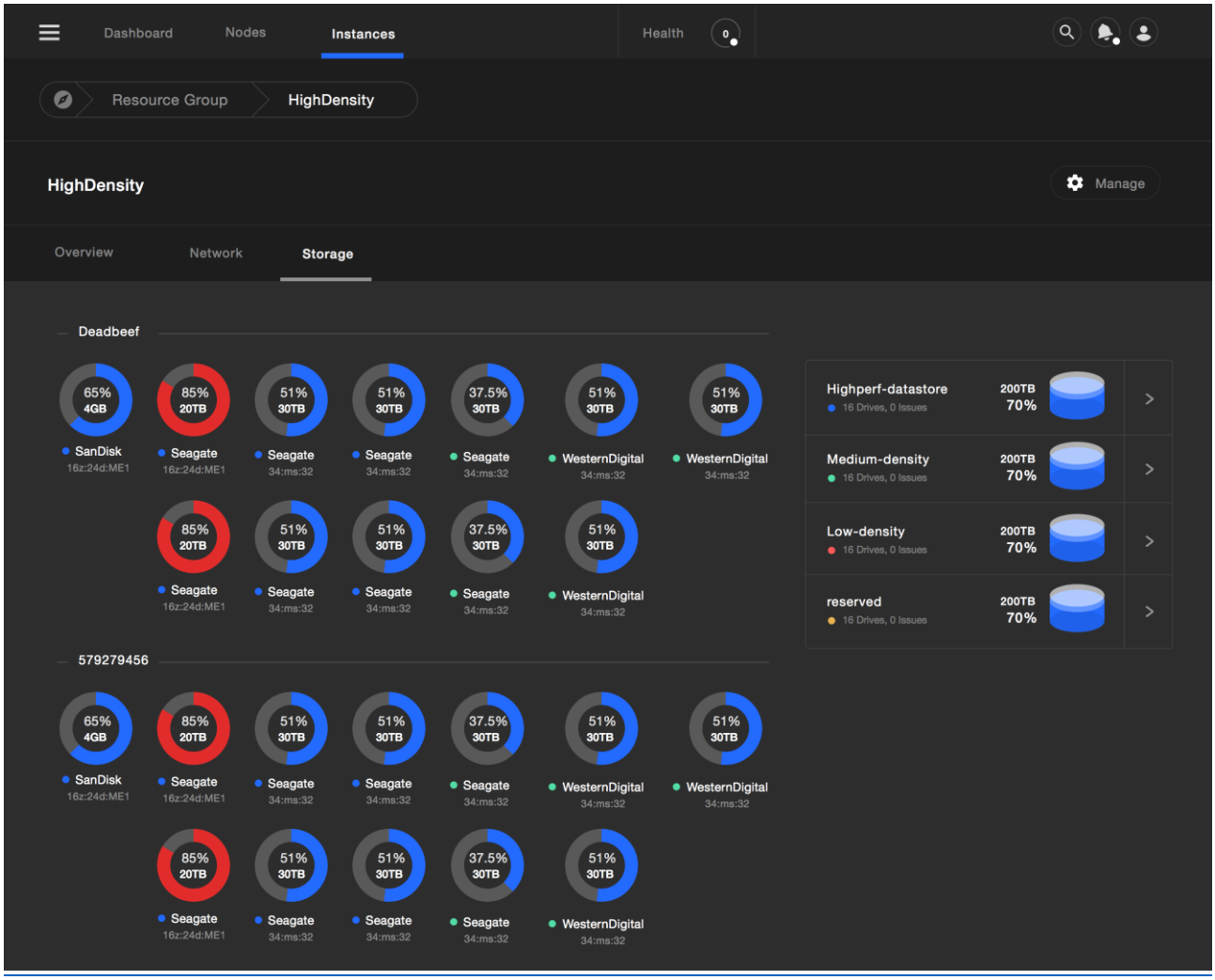
One of the most challenging requirements captured in Section 2 is [AppSched-03], that states that an application must be provisioned in <10ms. For a VIM such as OpenStack, this poses a number of challenges as this Python-based framework was designed for 'traditional' VMs that usually comprise a Linux or Windows-based guest OS. These VMs are heavy-weight and need miniaturization before they could start in the order of seconds. Containers and other light-weight virtualization techniques as those currently investigated in Superfluidity can start up much faster. When looking to approach fast provisioning and orchestration tools it is important to profile all aspects of the virtualization workflow. This will be reported by T5.2, where an analysis of different virtualization techniques is being carried out.

In order to support <10ms provisioning times it is important to consider the design of the orchestration platform and to remove overheads. The MicroVisor, Hypervisor platform that OnApp are bringing to Superfluidity is purpose-built, light-weight, distributed and focused on maximising the performance of virtual workloads running on distributed resources. As such, there have been improvements carried out to the MicroVisor orchestration framework that can be used to help decide on decisions for the rest of the Superfluidity orchestration tools. The MicroVisor UI is based on OpenStack and has had various improvements to be able to manage the expected workloads of Superfluidity.

5.1.1 UI design for managing a large collection of resources

Virtual workloads that are going to load in <10ms are potentially going to be far more numerous than standard visualization approaches currently account for. Horizon, which is the OpenStack Dashboard can handle the scale of Virtual Machines that currently are used by large enterprises, but will likely have some scalability issues when faced with orders of magnitudes more VMs than are currently used. A rethink of the UI is therefore needed for it to display the information available to administrators and end-users in a useful manner.

Computer assisted workload placement will therefore move from being just an optimization effort, to being a tool to help manage the workloads at the scales that are expected. A mockup diagram showing a possible visualization of the physical to virtual workloads is shown in Figure 17Error! eference source not found.. This Figure captures the physical, network overlay and virtual resources and how they relate to each other. The work is ongoing to determine which visualization mechanisms users and administrators will find useful.


Dashboard
Nodes
Instances
Health
0
Resource Group
HighDensity
HighDensity
Manage
Overview
Network
Storage
Deadbeef
65% 4GB
SanDisk
16z:24d:ME1
85% 20TB
Seagate
16z:24d:ME1
51% 30TB
Seagate
34:ms:32
51% 30TB
Seagate
34:ms:32
37.5% 30TB
Seagate
34:ms:32
51% 30TB
WesternDigital
34:ms:32
51% 30TB
WesternDigital
34:ms:32
85% 20TB
Seagate
16z:24d:ME1
51% 30TB
Seagate
34:ms:32
51% 30TB
Seagate
34:ms:32
37.5% 30TB
Seagate
34:ms:32
51% 30TB
WesternDigital
34:ms:32
579279456
65% 4GB
SanDisk
16z:24d:ME1
85% 20TB
Seagate
16z:24d:ME1
51% 30TB
Seagate
34:ms:32
51% 30TB
Seagate
34:ms:32
37.5% 30TB
Seagate
34:ms:32
51% 30TB
WesternDigital
34:ms:32
51% 30TB
WesternDigital
34:ms:32
85% 20TB
Seagate
16z:24d:ME1
51% 30TB
Seagate
34:ms:32
51% 30TB
Seagate
34:ms:32
37.5% 30TB
Seagate
34:ms:32
51% 30TB
WesternDigital
34:ms:32
Highperf-datastore
16 Drives, 0 Issues
200TB
70%
Medium-density
16 Drives, 0 Issues
200TB
70%
Low-density
16 Drives, 0 Issues
200TB
70%
reserved
16 Drives, 0 Issues
200TB
70%

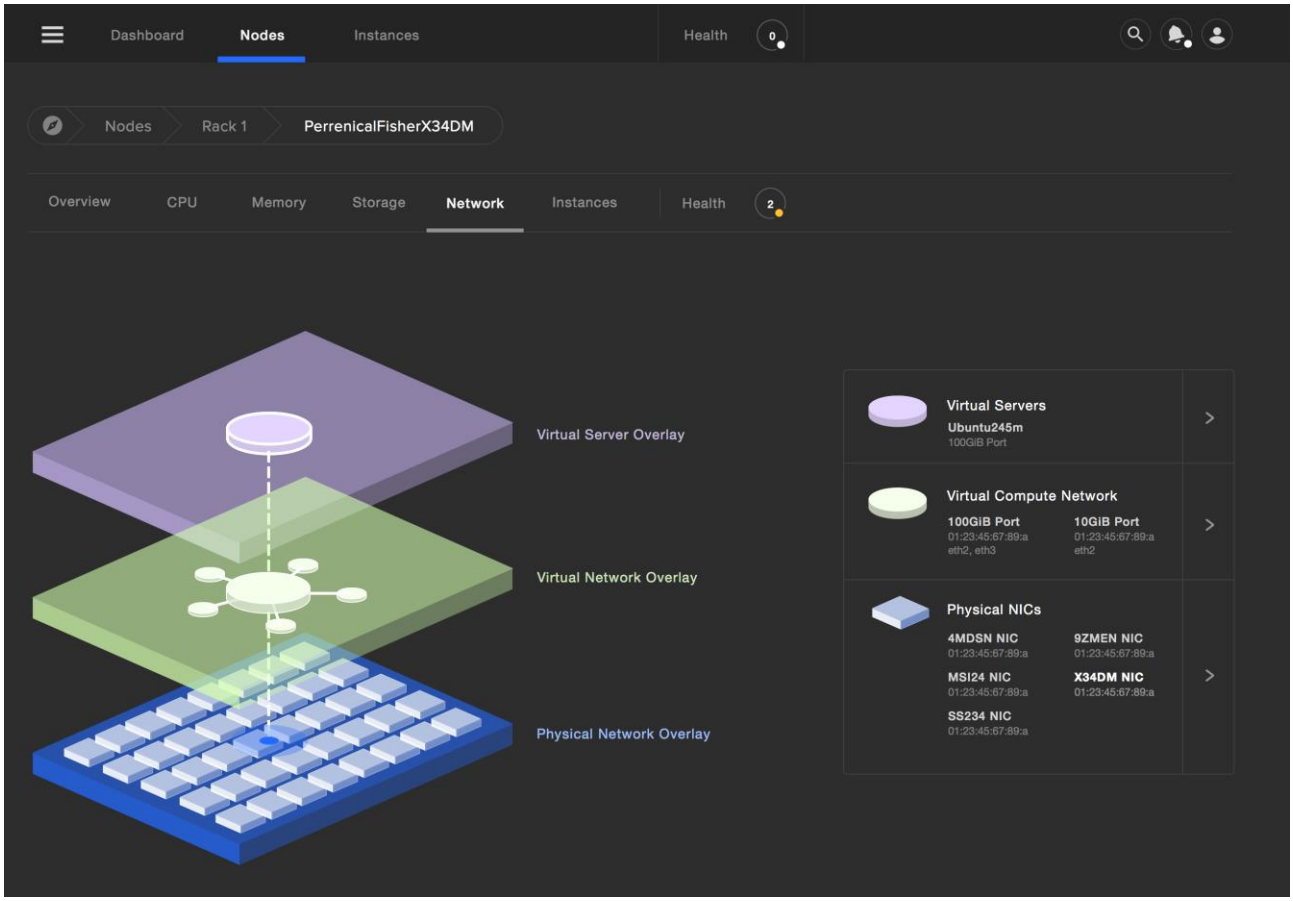


Figure 17: Mock-up diagram showing a UI that relates virtual to physical resources

In Error! Reference source not found. a visualization mock-up of the administration panel is shown. n this visualisation, the physical racks have a number of rack servers that are numbered and can be probed for more information. Each rack then has a number of compute nodes that can be contained within a single physical server. The CPU load of each compute unit is then visualised, with standard traffic light colouring used to indicate low-utilisation (green), through to heavily loaded compute units (red). This gives an administrator a powerful tool to quickly identify if there are any servers that are struggling and to indicate issues that potentially need to be resolved either through computer assisted orchestration, or manual intervention.

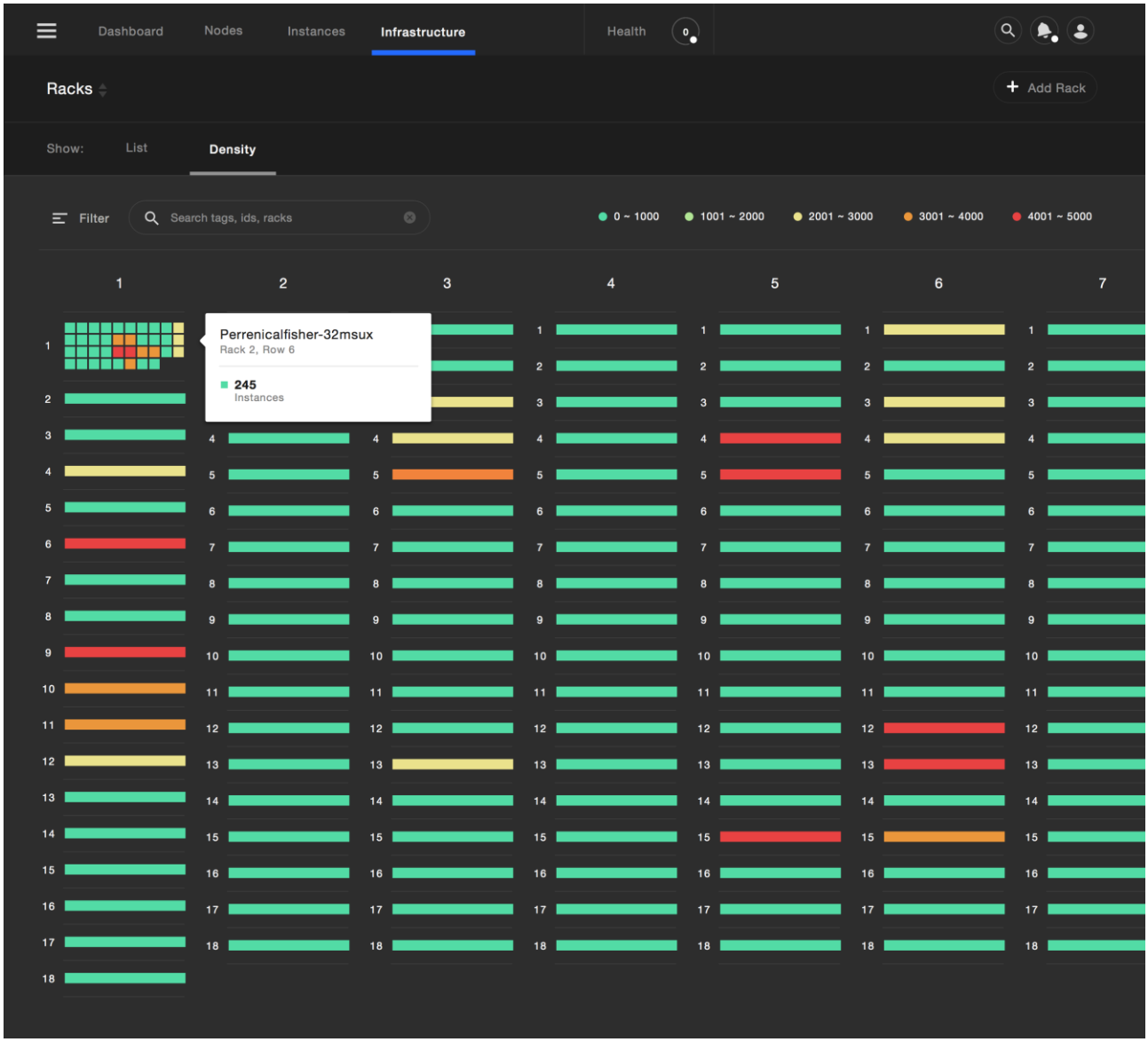


Figure 18: Mock-up diagram showing the rack utilization

Aside from the physical to virtual mapping and CPU load that have been shown in the previous Figures, it is also important to show the utilization of the storage resources. A mock-up Figure showing the utilization of the storage can be seen in Figure 19**Error! Reference source not found.**. isks that are close to being full are shown in red with the less utilized disks being coloured in blue. All of the storage resources are associated with particular racks and are separated accordingly. Also shown in the diagram is the notion of tiered storage performance levels. Given that certain virtualized workloads may have different I/O requirements it is important for the system to indicate different performance levels. This information can be captured in the data models in T4.1 and then analysed by the algorithms and heuristics in T5.1 to decide on where to place the workloads.

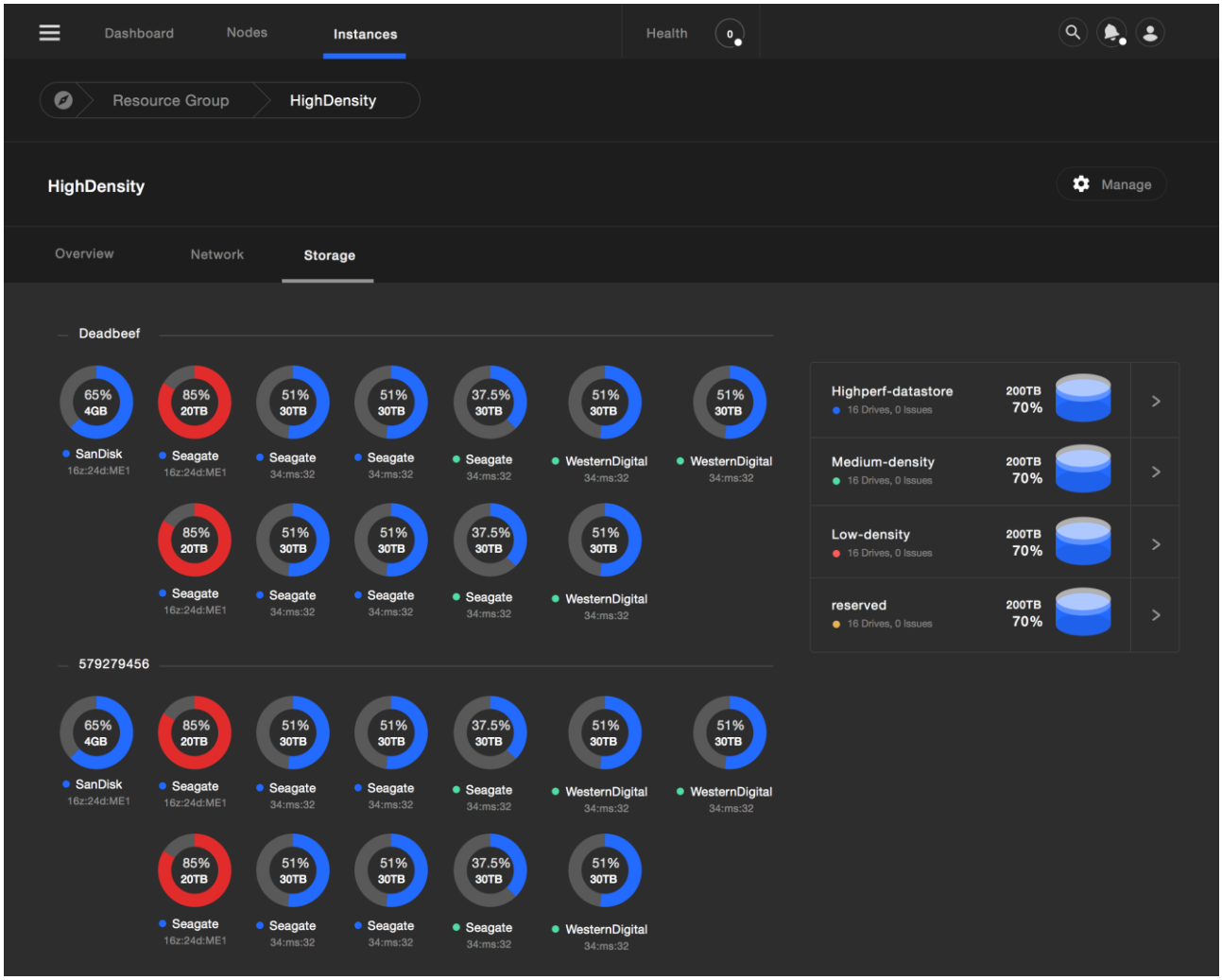


Figure 19: Mock-up diagram showing the storage utilization in the management UI

Given that SDN networking will also allow reconfiguration of a network, it is important for both the management platform and the orchestration system to be able to capture and possibly modify the network topology. This will allow maximization of the performance for a given set of workloads and configurations decided by the administrator. In **Error! Reference source not found.** a mock-up of the etwork mapping UI is shown. This can be used to visualize the current network topology and also could be used to capture modifications required of the network that could be then mapped to the network routers and hypervisors through tools such as OpenDaylight or others.

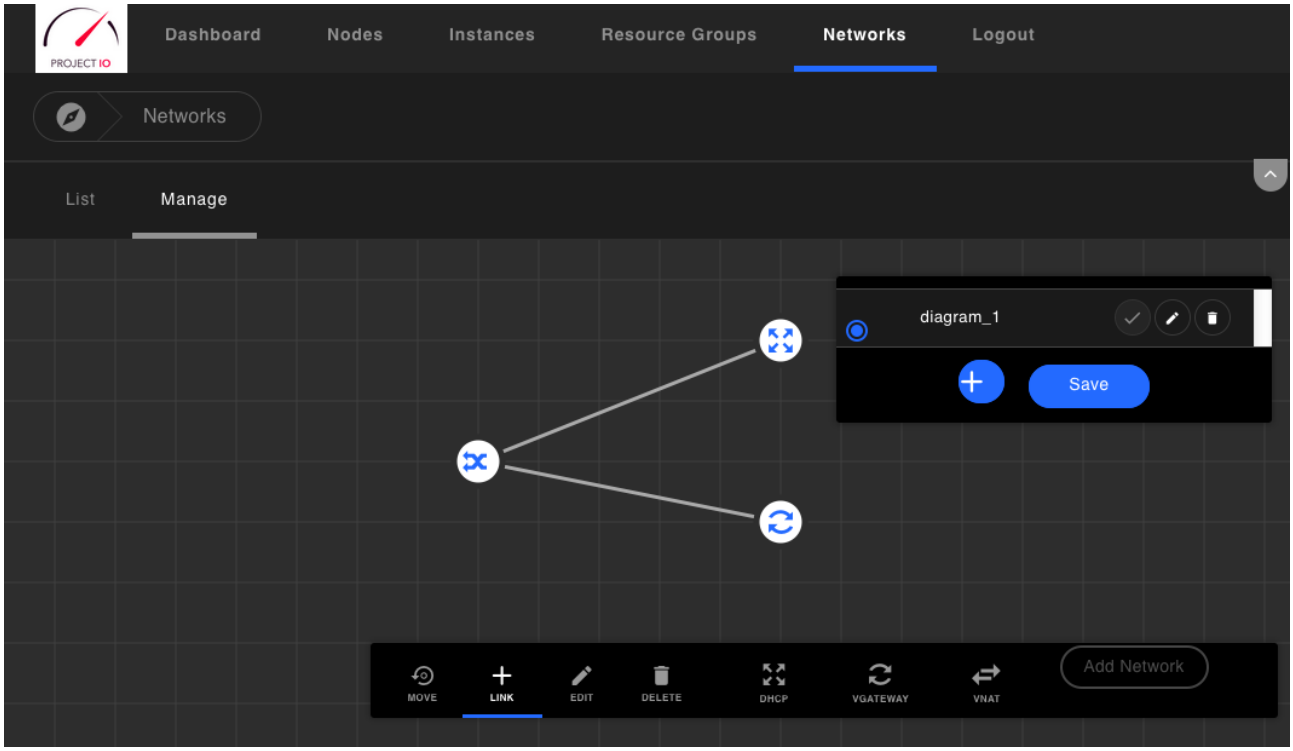


Figure 20: Mock-up showing the network planner UI

6 Conclusion

This document provides a report on the ongoing progress of Task 6.1 in Superfluidity project. The progress has been made on several challenges: recognition of the requirements from the control framework, analysis of the existing work of MEC workgroup, looking into subset of the currently existing solutions on the market and highlighting the gaps between the requirements and the solutions. A very important progress has been made on the management and orchestration design side, as introduced in Section 4 and on the follow up side the work in that direction will split between the selection from the described models, estimation of the virtual infrastructure work effort to provide support for the missing features and an implementation for a subset of those features.

7 References

[1] Mobile Edge Computing (MEC); Technical Requirements
http://www.etsi.org/deliver/etsi_gs/MEC/001_099/002/01.01.01_60/gs_MEC002v010101p.pdf

[2] Mobile Edge Computing (MEC); Framework and Reference Architecture

8 Annexes

8.1 Detailed Orchestration Requirements

This annex intends to collect the most relevant NFV and MEC Management and Orchestration requirements, in order to assist on the evaluation of available tools (open source or others in the market) to select which can be used by the project. Ideally, we expect to be able to use the same tools for both, as they share many commonalities, even though there are also some specificities. At the time or writing this document, there is an ongoing work (Work Item), within the ETSI MEC group, in order to perform the gap analysis between NFV and MEC management and orchestration, evaluating whether existing NFV tools can be reused in co-located MEC deployments.

Note: The requirement definitions are compliant with [RFC2119].

8.1.1 NFV

8.1.1.1 Generic

[NFV-Generic-01] The NFVO **MUST** perform three main functions:

- VNFs and NSs on-boarding
- Resources orchestration
- NS orchestration

[NFV-Generic-02] The VNF/NS on-boarding function **MUST** load into the NFV ecosystem the NFV/NS metadata and images to make them ready to be deployed.

[NFV-Generic-03] The Resources Orchestration function **MUST** interact with the NFVI (via VIM), managing the resources associated to NSs (e.g. create, query, terminate).

[NFV-Generic-04] The NS orchestration function **MUST** perform NS lifecycle management (LCM).

8.1.1.2 Repositories

[NFV-Repositories-01] The NFVO **MUST** be associated to the following repositories:

- VNF Catalogue, which stores the catalog of deploy-able VNFs
- NS Catalogue, which stores the catalogue of deploy-able NSs (VNFs, VLDs, VNFFGDs, PNFDs)
- NFV Instances, which stores the instance records of VNFs and NSs already deployed
- NFVI Resources, which stores the available, reserved and used resources used by VNFs/NS

[NFV-Repositories-02] The VNF Catalogue repository **MUST** store the list of deploy-able VNFs, storing a record per VNF with the following information (among many other):

- VNFD (VNF Descriptor)
    - Software version

- Virtual links and connection points
- VNF Monitoring metrics/KPIs
- VNF LCM policies and scripts
- List of VNFCs/VDUs
  - Version
  - CPU
  - Memory
  - Storage
  - Monitoring
  - LCM policies
- List of software images (1 per VDU)

[NFV-Repositories-03] The VNF Catalogue MUST be accessible by:

- The VNFO (*rw, master*). Example functions: on-boarding, obtain required resources, etc.
- The VNFMs (*ro*). Example functions: obtain VNF details like, VDU specs, LCM policies, etc.

[NFV-Repositories-04] The NS Catalogue repository MUST store the list of deploy-able NSs, storing a record per NS with the following information (among many other)

- NSD (NS Descriptor)
  - Version
  - List of VNFDs
    - (see above)
  - List of PNFDs (PNFDs)
    - Software version
    - Connection point
  - List of VNFFGDs (VNF Forwarding Graphs Descriptors)
    - List of VNFs
    - Forwarding path
  - List of VLDs (VL Descriptors)
    - VLAN ID
    - Connectivity Type
  - NS Monitoring metrics/KPIs
  - NS LCM policies and scripts

[NFV-Repositories-05] The NS Catalogue MUST be accessible by

- The VNFO *(rw, master)*. Example functions: on-boarding, get infrastructure resources, get list of NFVDs, PNFDs, VLDs or VNFFGDs.

[NFV-Repositories-06] The NFV Instances repository MUST store the list of deployed VNFs and NSs, storing a record per VNF (VNFR) and NS (NSR) with the following information (among many other)*

- VNFR

    - ID
    - Software version
    - Referenced VNFD
    - Managed VNFM
    - Parent NS
    - VIM/Localization
    - List of VLs
    - IP address
    - Active monitoring
    - Status
- NSR
    - ID
    - List of VNFRs
        - (see above)
    - List of PNFRs
        - Parent NS
        - VNFFG
        - IP Address
    - List of VLRs
        - Parent NS
        - VIM/Location
        - QoS allocated
    - List of VNFFGRs
        - Parent NS
        - Forwarding path
    - Active monitoring
    - Status

[NFV-Repositories-07] The NFV Instances repository MUST be accessible by:

- The NFVO *(rw, master)*. Example functions: store VNF/NS instance records, update VNF/NS instance records, get VNF/NS instance records.

  Note: The VNFM could make sense here to access VNF instances, but in the MANO specs there is no interface for that.

[NFV-Repositories-08] The NFVI Resources repository MUST keep track of available, reserved and allocated resources, in particular regarding:

- Available links
- Available CPU
- Available Memory

- Available Storage
- Reserved VLs
- Reserved VMs
- Reserved virtual storage
- Allocated VLs
- Allocated VMs
- Allocated virtual storage

[NFV-Repositories-09] The NFVI Resources MUST be accessible by

- The VNFO *(rw, master)*. Example functions: store resources, update resources, get reserve resources.

[NFV-Repositories-10] The NFVO MUST be able to correlate the NS/VNF instances records in the NVF Instances repository with the corresponding virtual resources in the NFVI Resources repository.

8.1.1.3 On-boarding

[NFV-Onboarding-01] The NFVO MUST on-board VNFs and NSs packages before they become available to be used/deployed in the NFV ecosystem.

[NFV-Onboarding-02] The NFVO SHOULD perform other actions than on-boarding regarding VNFs and NSs packages

- Disable VNF/NS packages, to deactivate temporarily the availability of VNFs/NSs
- Enable VNF/NS packages, to activate the availability of VNFs/NSs
- Update VNF/NS packages, to upgrade VNFs/NSs (descriptors and/or software images)
- Query VNF/NS packages, to inquire information about descriptors and images
- Delete VNF/NS packages, to remove VNFs/NSs (descriptors and software images)

[NFV-Onboarding-02] The NFVO MUST be able to deal with multiple infrastructure domains (VIMs/NFVIs).

[NFV-Onboarding-03] The VNFs on-boarding process MUST include the following actions

- Validating the correctness of the VNFD descriptor;
- Validating the correctness of the VDU constructs;
- Validating the correctness of the images;
- Storing the VNFD into the VNF Catalogue;
- Storing the VDU into the VNF Catalogue;
- Copy all VNF images to (potential) target VIMs (not mandatory at on-board time, but highly RECOMMENDED)
- Validating the correctness of the VNF functionality*;

  ** This step MAY be done at different stages.*


[NFV-Onboarding-04] The NSs on-boarding process **MUST** include the following actions

- Validating the correctness of all descriptors (NSDs, VNFFGs and VLDs**);
- Validating the existence of the VNFs**;
- Validating the correctness of the NS functionality*;
- Storing all associated descriptors (NSDs, VNFFGs and VLDs*) in the NS Catalogue;

** This step **MAY** be done at different stages.*

*** Assuming VNFs have being already on-boarded (it **MAY** also be done altogether).*

8.1.1.4 Instantiation

[NFV-Instantiation-01] The NFVM **MUST** support the on-demand instantiation of new VNFs under NFVO request (under the scope of an NS instantiation).

[NFV-Instantiation-02] The NFVM **MUST** support the VNF instantiation, using the descriptor (already on-boarded).

[NFV-Instantiation-03] The NFVM **MUST** validate whether the target VNF is available in the VNF Catalogue.

[NFV-Instantiation-04] The NFVM **MUST** create the infrastructural resources for the VNF

- Option 1 – Reservation on MEO, allocation on VIM, completion notification on MEO
- Option 2 – Via NFVO

[NFV-Instantiation-05] The VIM **MUST** validate and authorize the VNFM requests of resources (in Option 1).

[NFV-Instantiation-06] The VIM **MUST** validate and authorize the NFVO requests of resources (in Option 2).

[NFV-Instantiation-07] The NFVO **MUST** validate and authorize the VNFM requests of resources (in Option 2).

[NFV-Instantiation-08] The NFVM **MUST** add to the NFV Instances repository the new VNF instance record once the deployment is completed.

- Option 1 – Directly on the NFV Instances repository and notifying the NFVO
- Option 2 – Via NFVO

[NFV-Instantiation-09] The NFVM **MUST** access the VNF instance (or EM) to perform setup configurations once VNFCs are up and running, if required.

[NFV-Instantiation-10] The NFVO **MUST** instantiate on-demand new NSs, on OSSs requests.

[NFV-Instantiation-11] The NFVO **MUST** instantiate NSs, using the NS related descriptors (already on-boarded).

[NFV-Instantiation-12] The NFVO **MUST** validate whether the NS is available in the NS Catalogue.

[NFV-Instantiation-13] The NFVO MUST request to the correspondent VNF Managers the creation of the required VNFs.

[NFV-Instantiation-14] The NFVO MUST create resources on behalf the VNF Managers (Option 2) or get notified/make reservations about the creation of resources by the VNFMs (Option 1).

[NFV-Instantiation-15] The NFVO MUST request to the correspondent VIMs (descriptors MUST identify location, directly or indirectly) the creation of the required infrastructure, namely VLs and VNFFGs.

[NFV-Instantiation-16] The VIM MUST validate and authorize the NFVO's or VNFM's requests or reservation of resources.

[NFV-Instantiation-17] The NFVO MUST add to the NFV Instances repository the new NS instance once the NS deployment is completed.

[NFV-Instantiation-18] The NFVO MUST update the NFV Instances repository with the new NS instance – on the VNFM behalf (Option 2) or be notified of that (Option 1) – once the NS deployment is completed.

8.1.1.5 Monitoring

[NFV-Monitoring-01] The VNFM SHOULD get infrastructural metrics, alerts and KPIs from VIMs.

[NFV-Monitoring-02] The NFVO MUST get infrastructural metrics, alerts and KPIs from VIMs.

[NFV-Monitoring-03] The NFVM SHOULD get service alerts, metrics and KPIs from VNF instances (or EMs).

[NFV-Monitoring-04] The NFVO MUST get service alerts, metrics and KPIs from VNFMs regarding VNF instances.

[NFV-Monitoring-05] The OSSs MUST get alerts, metrics and KPIs from VNF instances (or EMs) or VNFOs.

[NFV-Monitoring-06] The VNFM and VNFOs SHOULD use metrics, alerts and KPIs for multiple purposes

- Fault detection, correlation and recovery
- Performance Management
- Scaling in/out/down/up
- Logging and statistics

[NFV-Monitoring-07] The File descriptors (VNFD, NSD, etc.) MUST be able to describe metrics, alerts and KPIs and associated respective thresholds (SLAs).

8.1.1.6 Modification

[NFV-Modification-01] The VNF instances MAY be modified by VNFMs during runtime operation in multiple ways


- Scaling in/out/down/up
- Fault detection, correlation and recovery
- Moving (all or some composing VNFCs) to another location
- Live Upgrade (to a new software version)

[NFV-Modification-02] The VNFM SHOULD perform scaling in/out/down/up on either NFVO or Back-office request.

[NFV-Modification-03] The VNFM MUST perform scaling in/out/down/up automatically, triggered by the analysis on monitoring data or other information, and according to the policies defined on the VNFD.

[NFV-Modification-04] To perform VNF scaling in/out/down/up, VNFMs MUST allocate or dispose infrastructural resources – directly on VIMs or via NFVO – and access to the VNFCs for service reconfiguration.

[NFV-Modification-05] The VNFM SHOULD perform fault detection and self-healing based on monitoring data.

[NFV-Modification-06] To perform full or partial VNF relocation, VNFMs MAY

- Option 1 – Perform VNFCs live migration and perform any required VNFCs reconfiguration.
- Option 2 – Create new VNFCs and VLs in the new location and dispose VNFCs and VLs in the old location, and access to the new VNFCs for service reconfiguration.

[NFV-Modification-07] The VNFM SHOULD perform upgrades on NFVO demand.

[NFV-Modification-08] NS instances SHOULD be modified by the NFVO during runtime operation in multiple ways

- Scaling in/out/down/up composing VNFs
- Fault detection, correlation and recovery
- Moving (all or some composing VNFs) to other locations
- Change VLs and VNFFGs

[NFV-Modification-09] The NFVO MUST perform scaling in/out/down/up of composing VNFs, VLs or VNFFGs on OSSs request.

[NFV-Modification-10] The NFVO MUST perform scaling in/out/down/up of composing VNFs, VLs or VNFFGs automatically, triggered by the analysis on monitoring data or other information, and according to the policies defined on NSD.

[NFV-Modification-11] The NFVO SHOULD perform fault detection and self-healing based on monitoring data.

[NFV-Modification-12] The NFVO MUST move some VNFs to other locations and relocate VLs/VNFFG, either automatically – according to NSD policies - or on OSS request.


[NFV-Modification-13] The NFVO MUST modify the VLs or VNFFG, either automatically – according to NSD policies – or on OSS request. Examples functions: link bandwidth, VNFFG connection points.

8.1.1.7 Termination

[NFV-Termination-01] The VNFM MUST terminate on-demand existing VNF instances (on VNFO request).

[NFV-Termination-02] The VNFM MUST validate whether the VNF is available on the NFV Instances repository.

[NFV-Termination-03] The VNFM MUST dispose the infrastructural resources of the VNF

- Option 1 – Directly to the VIM, notifying the NFVO
- Option 2 – Via NFVO

[NFV-Termination-04] The VIM MUST validate and authorize the NFVM requests to dispose resources (Option 1).

[NFV-Termination-05] The NFVO MUST validate and authorize the NFVM requests to dispose resources (Option 2).

[NFV-Termination-06] The VNFM MUST remove from the NFV Instances repository the VNF instance record once the disposal is completed.

- Option 1 – Directly on the NFV Instances repository and notifying the NFVO
- Option 2 – Via NFVO

[NFV-Termination-07] The NFVM SHOULD access the VNF instance (or EM) to perform termination procedures.

[NFV-Termination-08] The NFVO MUST dispose existing NSs on OSS request.

[NFV-Termination-09] The NFVO MUST validate whether the NS is available on the NFV Instances repository.

[NFV-Termination-10] The NFVO MUST request to the correspondent VNFM the disposal of the VNFs.

[NFV-Termination-11] The NFVO MUST request the disposal of resources on behalf of VNFMs (Option 1) or be notified about the disposal of resources by VNFMs (Option 2).

[NFV-Termination-12] The NFVO MUST request to the correspondent VIMs the disposal of the resources, namely VLs and VNFFGs.

[NFV-Termination-13] The VIM MUST validate and authorize the NFVO and VNFM requests for resources disposal.

[NFV-Termination-14] The NFVO MUST remove from the NFV Instances repository the NS instance once the NS disposal is completed.

[NFV-Termination-15] The NFVO MUST remove from the NFV Instances repository the NS instance record once the disposal is completed.

8.1.2 MEC

8.1.2.1 Generic

[MEC-Generic-01] The MEC System **MUST** perform application lifecycle management actions: on-boarding, instantiation, modification and termination.

[MEC-Generic-02] The MEC System **MUST** perform application security actions: authentication, authorization.

[MEC-Generic-03] The MEC System SHOULD support mobility of UEs, i.e. the ability to support the same MEC application on different MEC hosts, managing the seamless handover (from one cell to another associated or not with the same MEC host).

[MEC-Generic-04] The MEC System **MUST** perform with MEC App orchestration.

[MEC-Generic-05] The MEC System **MUST** perform with resource orchestration.

[MEC-Generic-06] The MEC system **MUST** have information about the mobile edge system (e.g. list of edges, available services, mapping to mobile network access points).

[MEC-Generic-07] The MEO **MUST** interact with the VIM for resources management (e.g. CRUD).

[MEC-Generic-08] The MEO **MUST** interact with the MEPM for MEC App Lifecycle management (LCM) and Platform Management.

8.1.2.2 Repositories

[MEC-Repositories-01] The MEO **MUST** be associated to the following repositories

- MEC App Catalogue, which stores the catalog of the deployable MEC apps (app, rules, requirements such as required resources, maximum latency, required or useful services, etc.)
- MEC Hosts Inventory, which stores the list of MEC hosts (mapping between Cell IDs' and MEC Servers) and the list of available services per host (RNIS, LOC, DNS, etc.)
- MEC App Instances, which stores the instance records of MEC apps running at MEC hosts
- MECI (MEC Infrastructure) Resources, which stores the available, reserved and used resources used by MEC apps in all MEC Hosts

[MEC-Repositories-02] The MEC App Catalogue repository **MUST** store the list of deployable MEC Apps, storing a record per MEC App with the following information (among many others)

- MEC App Descriptor
  - MEC App ID
  - Software version
  - List of dependencies (required MEC services that are needed for the mobile edge application to be able to run)
  - Virtual links and connection points

- List of AppCs/VDUs
  - Version
  - CPU
  - Memory
  - Storage
  - Monitoring
  - LCM policies
- List of software images (1 per VDU)
- MEC App monitoring metrics/KPIs
- MEC App LCM policies and scripts
- List of requirements on connectivity (connectivity to applications/services within the MEC system, to local networks, or to Internet)
- List of requirements on mobility (e.g. application state relocation, application instance relocation)
- List of MEC App SLA requirements (latency, throughput, …)
- DNS mapping
- TOF rules

[MEC-Repositories-03] The MEC Hosts Inventory repository MUST store the list of MEC Hosts, storing a record containing available Hosts and mapping to Cell-IDs per Host (among others)

- MEC Host record
  - Services List
  - Serving Cell-IDs

[MEC-Repositories-04] The MEC App Instances repository MUST store the list of MEC Apps, storing a record per MEC App with the following information (among many other):

- MEC App instance record
  - ID
  - Software version
  - Referenced MEC App Descriptor
  - Managing MEPM
  - VIM/Location
  - List of VLs, VDUs and Connectors
  - IP address
  - Active monitoring metrics/KPIs
  - Status

[MEC-Repositories-05] The MEC Resources repository MUST keep track of available, reserved and allocated resources, in particular

- Available links


- Available CPU
- Available Memory
- Available storage
- Reserved VLs
- Reserved VMs
- Reserved virtual storage
- Allocated VLs
- Allocated VMs
- Allocated virtual storage

[MEC-Repositories-06] The MEO MUST be able to correlate the MEC App instances records in the MEC Instances repository with the corresponding virtual resources in the MEC VIM Resources repository.

8.1.2.3 On-boarding

[MEC-Onboarding-01] The MEO MUST on-board MEC App packages before they become available to be used/deployed in the MEC system. The on-boarding includes loading application image and application descriptor. The MEC App on-boarding process includes the following actions:

- MEC App Description validation
- Validation of application images
- Store the MEC App Descriptor in the MEC App Catalogue
- Store MEC App images in the MEC App Catalogue repository
- Copy MEC App images to (potential) target VIMs*

** This step MAY be done at different stages.*

[MEC-Onboarding-02] The MEO SHOULD perform other actions regarding MEC App packages:

- Enable, to activate temporarily the availability of MEC App instantiations
- Disable, to deactivate temporarily the availability of MEC App instantiations
- Update, to update MEC App (descriptors and/or software images versions)
- Query, to get information about descriptors and images
- Delete, to remove MEC App (descriptors and images) from Catalogue

8.1.2.4 Instantiation

[MEC-Instantiation-01] The MEPM MUST support MEC App instantiation, using the MEC App descriptor and images (already on-boarded).

[MEC-Instantiation-02] The MEPM MUST support MEO on-demand instantiation of a new MEC App.

[MEC-Instantiation-03] The MEPM MUST create the infrastructural resources for the MEC App, according to MEO instructions


- Option 1 – Reservation on MEO, allocation on VIM, completion notification on MEO
- Option 2 – Via MEO

[MEC-Instantiation-04] The VIM **MUST** validate and authorize MEPM requests of resources (in Option 1).

[MEC-Instantiation-05] The VIM **MUST** validate and authorize the MEPM requests of resources (in Option 1).

[MEC-Instantiation-06] The VIM **MUST** validate and authorize the MEO requests of resources (in Option 2).

[MEC-Instantiation-07] The MEO **MUST** validate and authorize the MEPM requests of resources (in Option 2).

[MEC-Instantiation-08] The MEO **MUST** check resources availability in its MECI Resources repository.

[MEC-Instantiation-09] The MEO **SHOULD** check resources availability with the VIM.

[MEC-Instantiation-10] The MEPM **MUST** add MEC App instances to the MEC App Instances repository, once the deployment is completed:

- Option 1 – Directly on the MEC App Instances repository, notifying MEO
- Option 2 – Via MEO

[MEC-Instantiation-11] The MEPM **MUST** access the MEC App instance to perform setup configurations once it is up and running, if required.

8.1.2.5 Monitoring

[MEC-Monitoring-01] The MEPM **SHOULD** get infrastructural metrics, alerts and KPIs from VIMs.

[MEC-Monitoring-02] The MEPM **SHOULD** get service alerts, metrics and KPIs from MEC App instances and/or MEC services.

[MEC-Monitoring-03] The MEO **MUST** get infrastructural metrics, alerts and KPIs from VIMs.

[MEC-Monitoring-04] The MEO **MUST** get service alerts, metrics and KPIs from MEPM regarding MEC App instances and/or MEC services.

[MEC-Monitoring-05] The OSSs **MUST** get alerts, metrics and KPIs from the MEPM and/or MEO.

[MEC-Monitoring-06] The MEPM and MEO **SHOULD** use metrics, alerts and KPIs for multiple purposes

- Fault detection, correlation and recovery
- Performance Management
- Scaling in/out/down/up
- Logging and statistics

[MEC-Monitoring-07] The MEC App descriptor MUST describe metrics, alerts and KPIs and associated respective thresholds (SLAs).

8.1.2.6 Modification

[MEC-Modification-01] The MEC App instances SHOULD be modified by MEPM during runtime operation in multiple ways

- Scaling in/out/down/up
- Recovery from failure or degradation
- Live Upgrade

[MEC-Modification-02] The MEPM SHOULD perform scaling in/out/down/up on either MEO or Back-office (MEPM GUI interface) request.

[MEC-Modification-03] The MEPM MUST perform scaling in/out/down/up automatically, triggered by the analysis on monitoring data or other information, and according to the policies defined on the MEC App Descriptor.

[MEC-Modification-04] To perform MEC App scaling in/out/down/up, MEPM MUST allocate or dispose infrastructural resources (directly on VIMs or via MEO) and perform MEC App reconfigurations.

[MEC-Modification-05] The MEPM SHOULD perform fault detection and self-healing based on monitoring data.

[MEC-Modification-06] The MEPM SHOULD perform live upgrade to another MEC App software version on MEO or back-office demand.

8.1.2.7 Mobility

[MEC-Mobility-01] The MEC System MUST be able to maintain service continuity for moving UEs, ensuring the MEC Apps “follow” them.

[MEC-Mobility-02] The MEO SHOULD determine the best location for MEC Apps relocation based on App requirements

[MEC-Mobility-03] The MEO SHOULD have to move MEC App instances between mobile edge hosts in order to continue to satisfy the requirements of the MEC App. *NOTE: Requirements of the MEC App can include latency, compute resources, storage resources, etc.*

[MEC-Mobility-04] To perform full or partial MEC App relocation, the MEPM MAY

- Option 1 – Perform a resources live migration of MEC App and perform any required reconfiguration.
- Option 2 – Create and configure a new MEC App in the new location and terminate the MEC App in the old location.


[MEC-Mobility-05] The MEO SHOULD relocate a MEC App based on relocation requests coming from UE/MEP triggers or OSS requests, to another mobile edge host, fulfilling the requirements for that MEC App.

[MEC-Mobility-06] The MEO MUST interact with old and new VIMs to perform MEC App relocation.

[MEC-Mobility-07] The MEO MUST perform the MEC App instantiation before starting the MEC App state relocation.

[MEC-Mobility-08] The MEO MUST allow the interaction and state synchronization between the two MEC App instances: on the source and target mobile edge hosts.

[MEC-Mobility-09] The MEO MUST support and supervise MEC App instance relocation between a mobile edge host and an external cloud environment.

8.1.2.8 Termination

[MEC-Termination-01] The MEPM MUST terminate on-demand existing MEC App instances on MEO request, according to the orchestration rules.

[MEC-Modification-02] The MEPM MUST dispose the infrastructural resources of the MEC App instance

- Option 1 – Directly to the VIM, notifying the MEO
- Option 2 – Via MEO

[MEC-Termination-03] The VIM MUST validate and authorize the MEPM requests to dispose resources (Option 1).

[MEC-Termination-04] The MEO MUST validate and authorize the MEPM requests to dispose resources (Option 2).

[MEC-Termination-05] The VIM MUST validate and authorize the MEO requests to dispose resources (Option 2).

[MEC-Termination-06] The MEPM MUST remove from the MEC App Instances repository the MEC App instance record once the disposal is completed.

- Option 1 – Directly on the MEC App Instances repository and notifying the MEO
- Option 2 – Via MEO

[MEC-Termination-07] The MEPM SHOULD access the MEC App instance to perform termination procedures.

[MEC-Termination-08] The MEO MUST dispose on-demand existing MEC App instances on OSS request.

8.2 Detailed Orchestration Flows

8.2.1 NFV

This section intends to identify the Management and Orchestration flows on NFV environments. These flows, pictures and text, are originally retrieved from [ETSI-NFV-MANO].

8.2.1.1 VNF On-boarding

The following Figure depicts the VNF on-boarding flow.

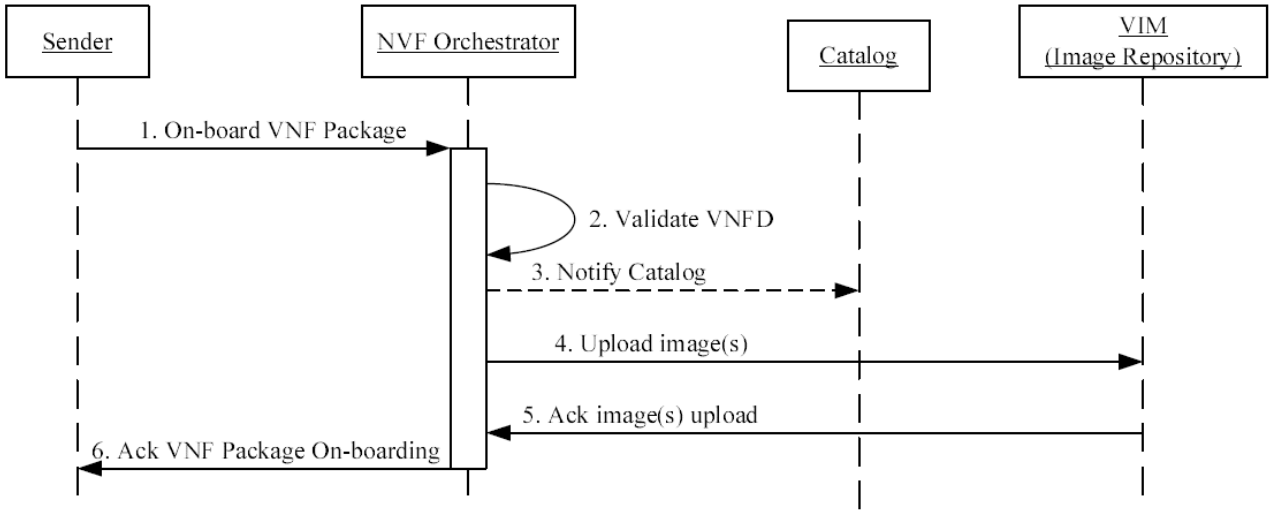


Figure 21: Orchestration Flows: VNF On-boarding [ETSI-NFV-MANO].

See [ETSI-NFV-MANO] *for the main steps for VNF on-boarding,*

8.2.1.2 VNF Instantiation

The following Figure depicts the VNF instantiation.

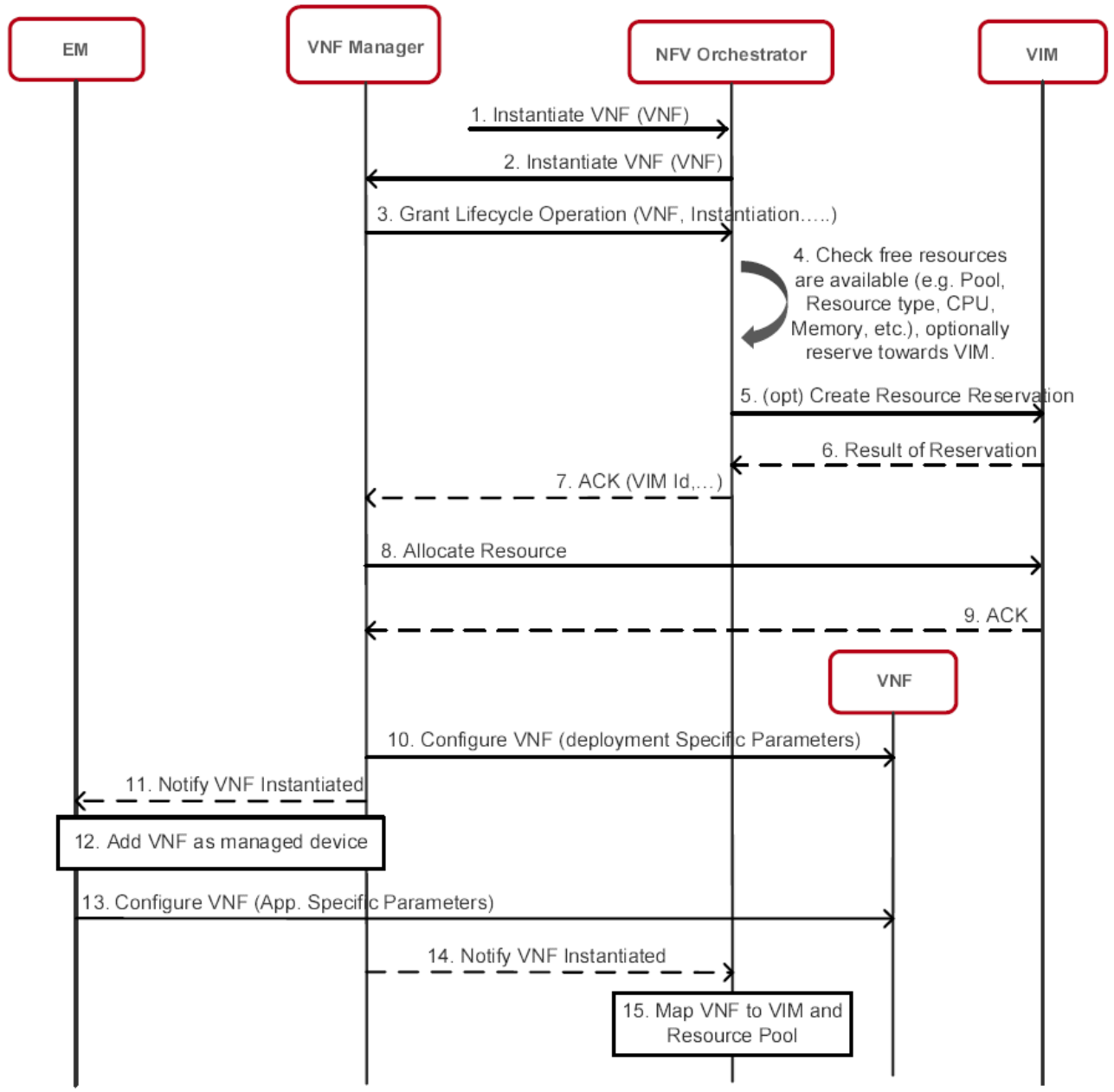


Figure 22: Orchestration Flows: VNF Instantiation [ETSI-NFV-MANO].

See [ETSI-NFV-MANO] *for the main steps for VNF Instantiation.*

8.2.1.3 VNF Scaling Out

The following Figure depicts the VNF scale-out flow.

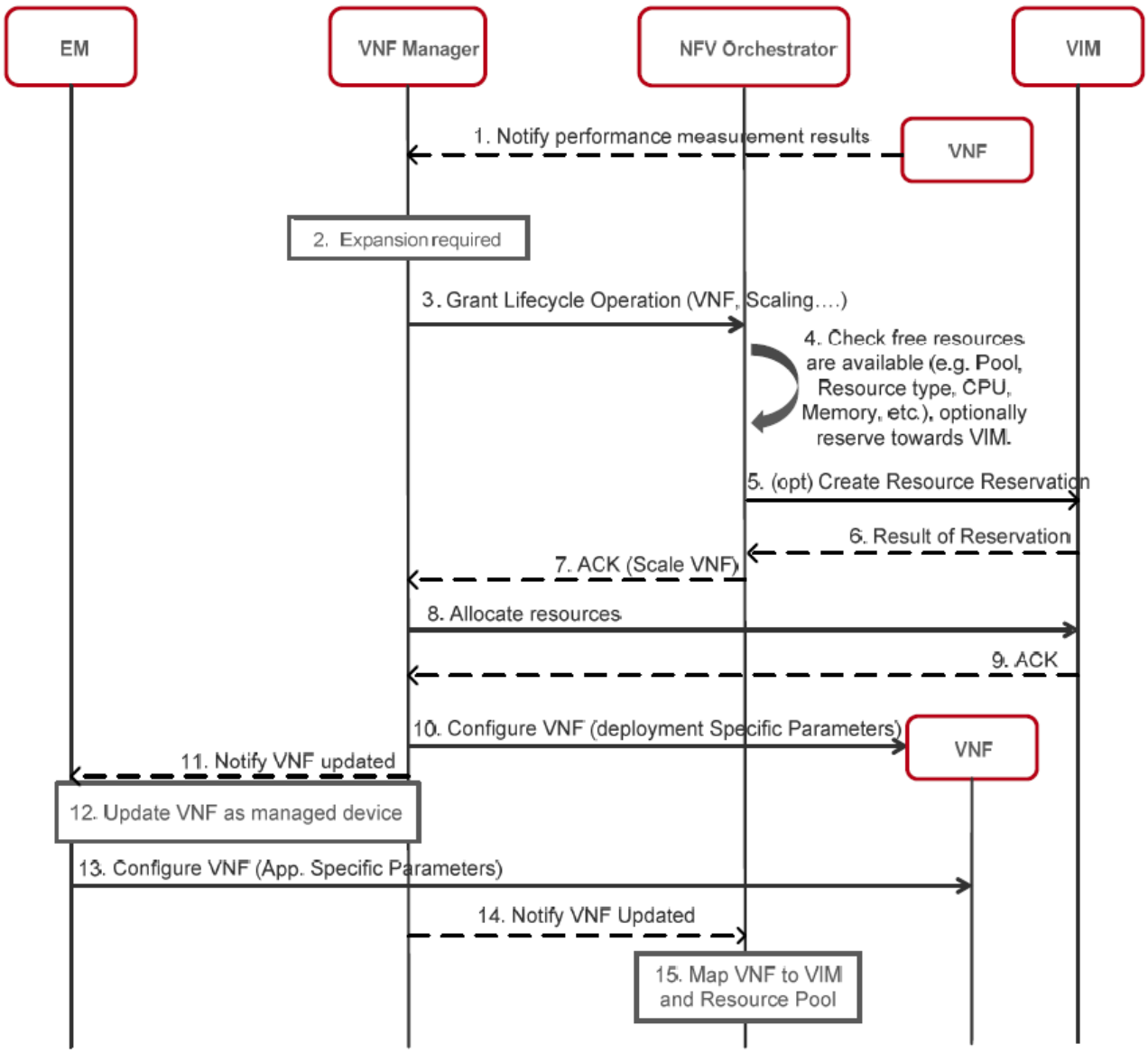


Figure 23: Orchestration Flows: VNF Scale-out [ETSI-NFV-MANO].

See [ETSI-NFV-MANO] *for the main steps for VNF scale-out.*

8.2.1.4 VNF Scaling In

The following Figure depicts the VNF scale-in flow.

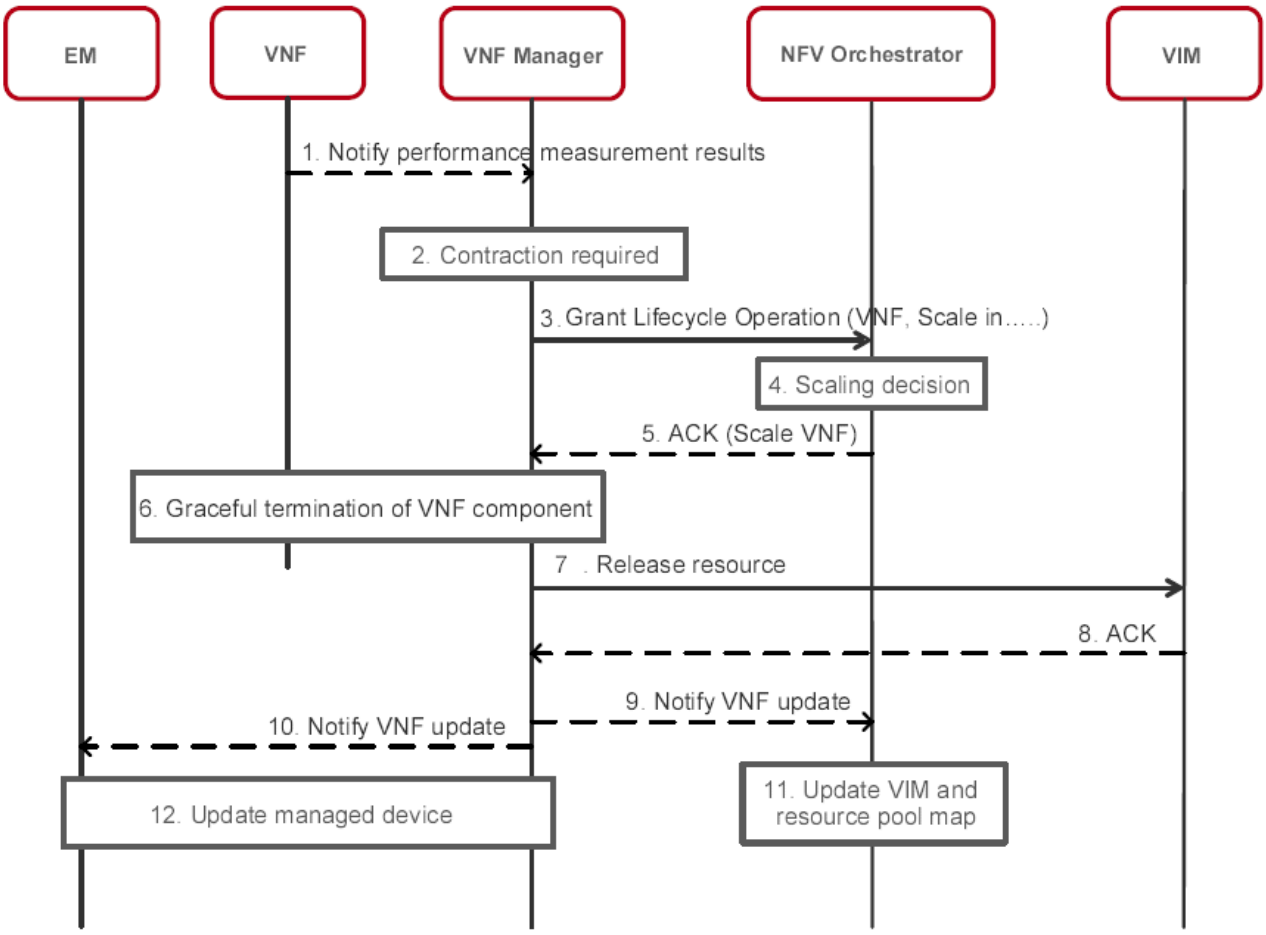


Figure 24: Orchestration Flows: VNF Scale-in [ETSI-NFV-MANO].

See [ETSI-NFV-MANO] *for the main steps for VNF scale-in.*

8.2.1.5 VNF Termination

The following Figure depicts the VNF termination flow.

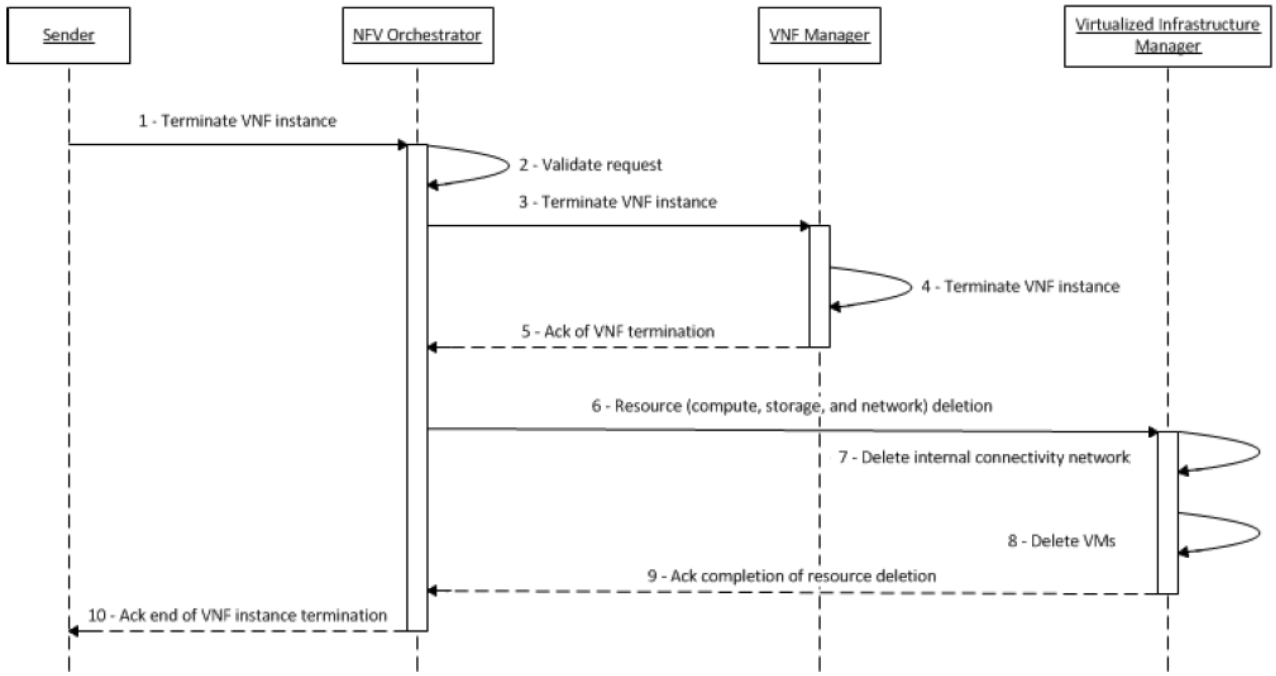


Figure 25: Orchestration Flows: VNF Termination [ETSI-NFV-MANO].

See [ETSI-NFV-MANO] *for the main steps for VNF termination.*

8.2.1.6 NS On-boarding

The following Figure depicts the VNF on-boarding flow.

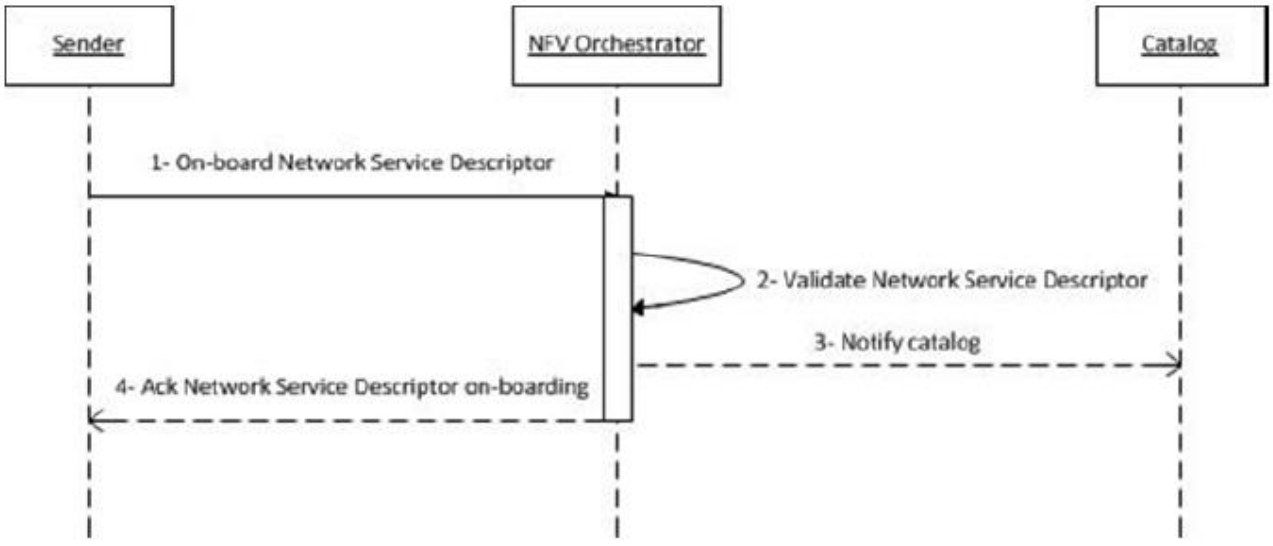


Figure 26: Orchestration Flows: NS On-boarding [ETSI-NFV-MANO].

See [ETSI-NFV-MANO] *for the main steps for NS on-boarding.*

8.2.1.7 NS Instantiation

The following Figure depicts the NS instantiation flow.

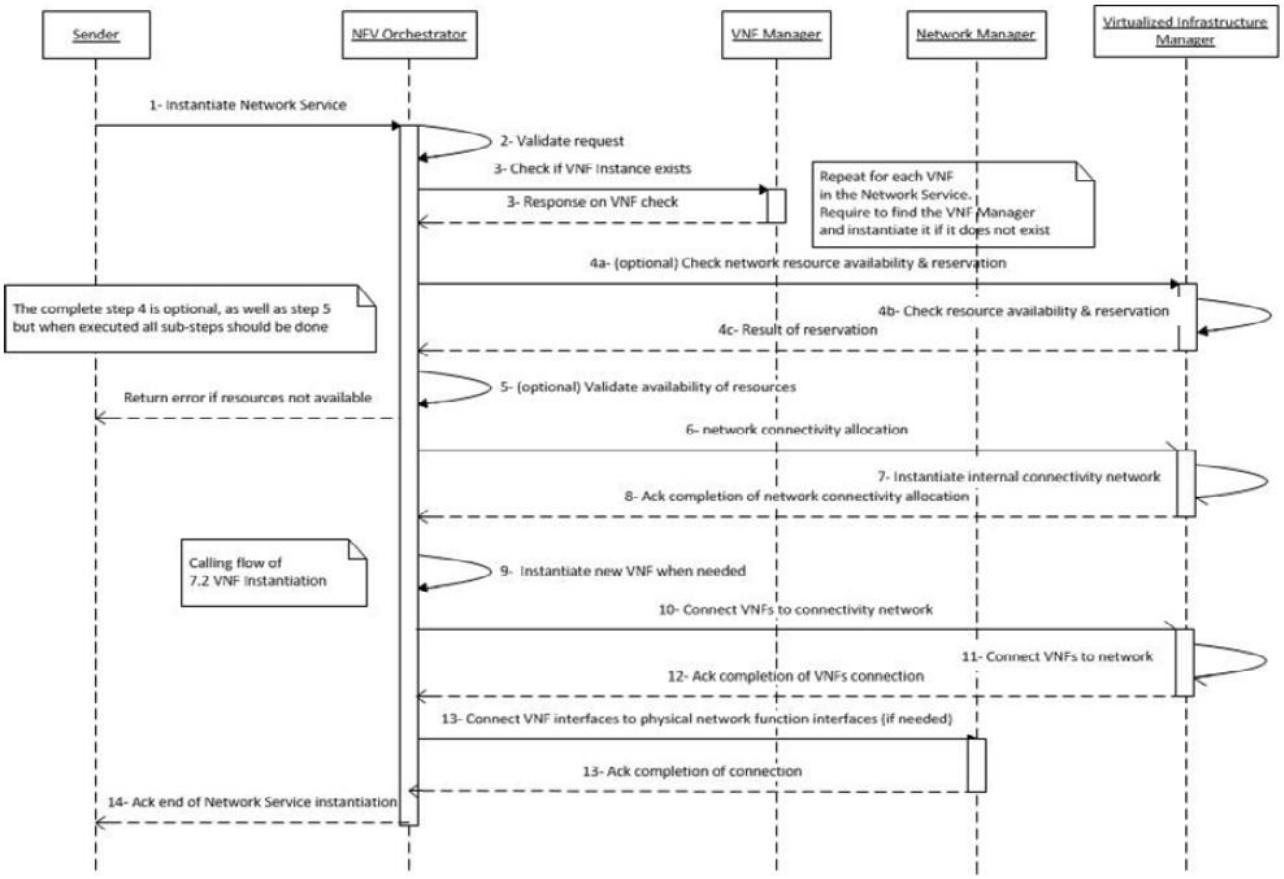


Figure 27: Orchestration Flows: NS Instantiation [ETSI-NFV-MANO].

See [ETSI-NFV-MANO] *for the main steps for NS instantiation.*

8.2.1.8 NS Scaling Out

The following Figure depicts the NS scale-out flow.

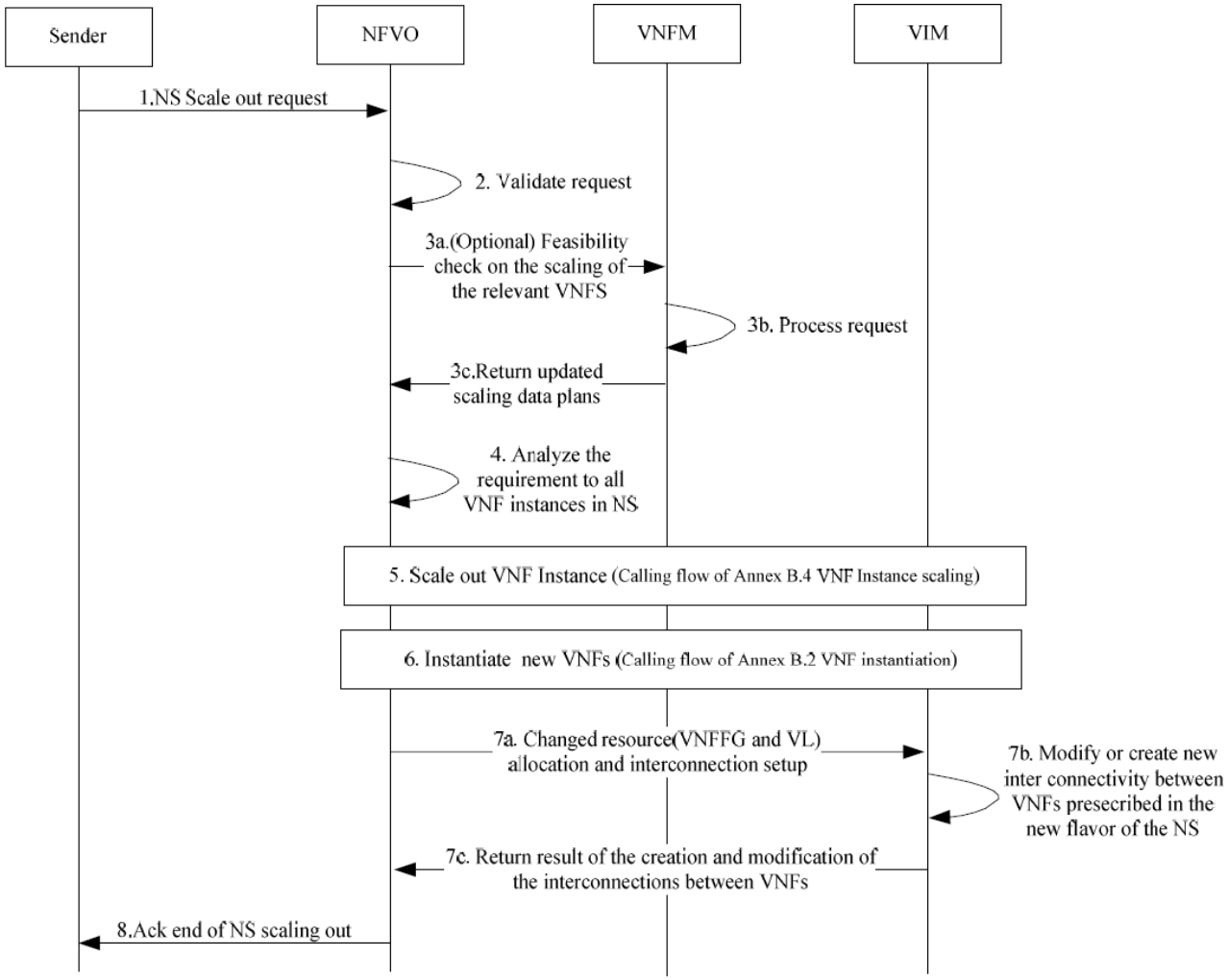


Figure 28: Orchestration Flows: NS Scale-out [ETSI-NFV-MANO].

See [ETSI-NFV-MANO] *for the main steps for NS scale-out.*

8.2.1.9 NS Scaling In

The following Figure depicts the NS scale-in flow.

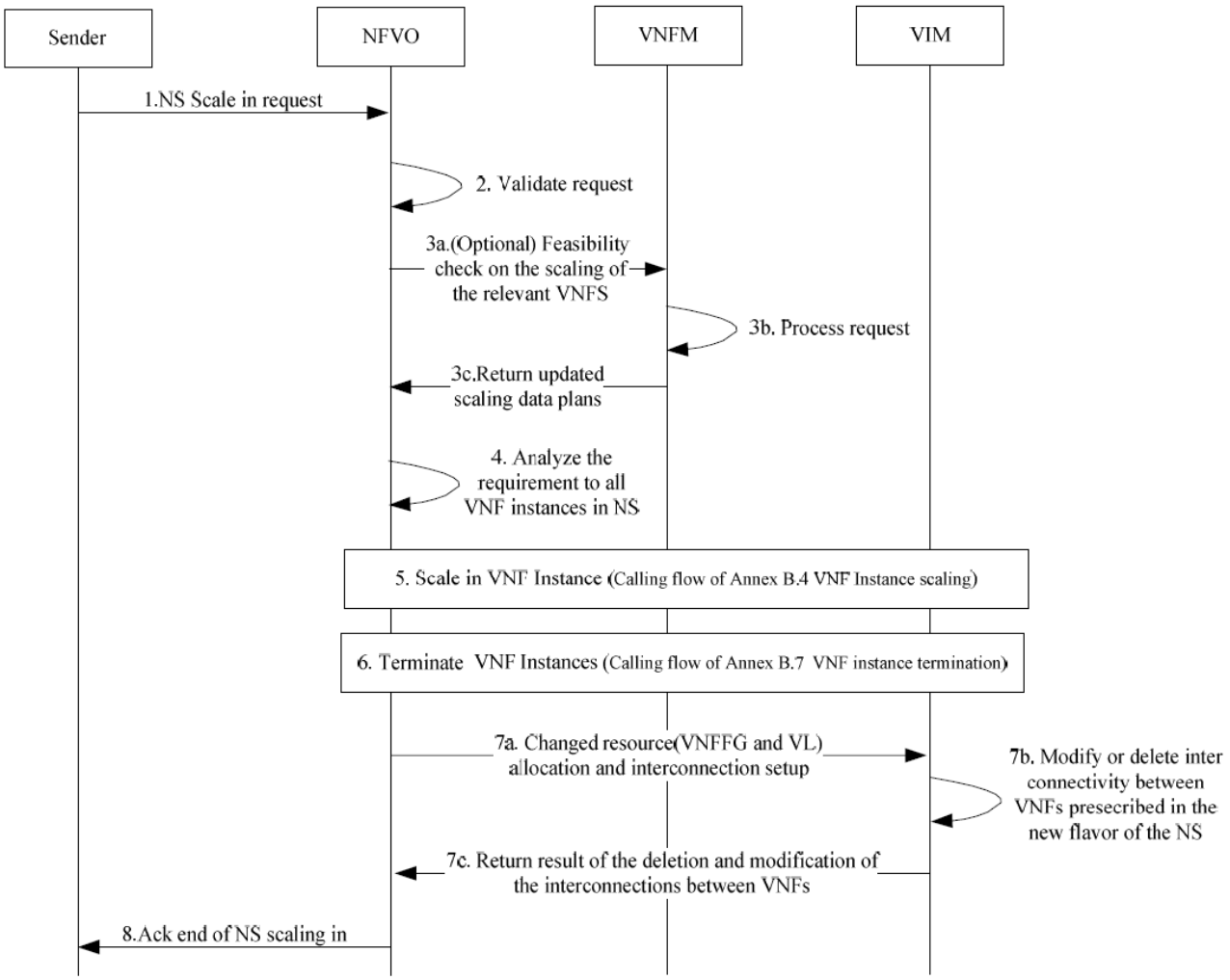


Figure 29: Orchestration Flows: NS Scale-in [ETSI-NFV-MANO].

See [ETSI-NFV-MANO] *for the main steps for NS scale-in.*

8.2.1.10 NS Termination

The following Figure depicts the NS termination flow.

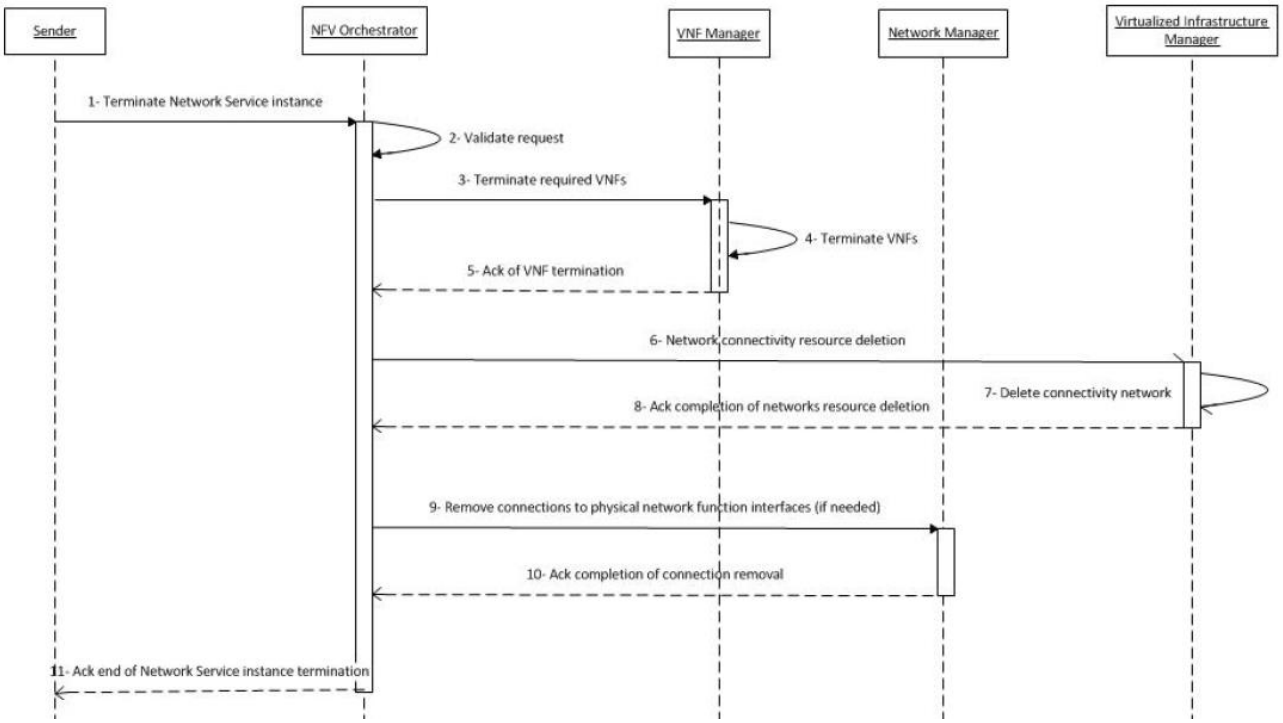


Figure 30: Orchestration Flows: NS Termination [ETSI-NFV-MANO].

See [ETSI-NFV-MANO] *for the main steps for NS termination.*

8.2.2 MEC

This section intends to identify the Management and Orchestration flows on MEC environments.

8.2.2.1 MEC App On-boarding

The following Figure depicts the MEC App on-boarding flow.

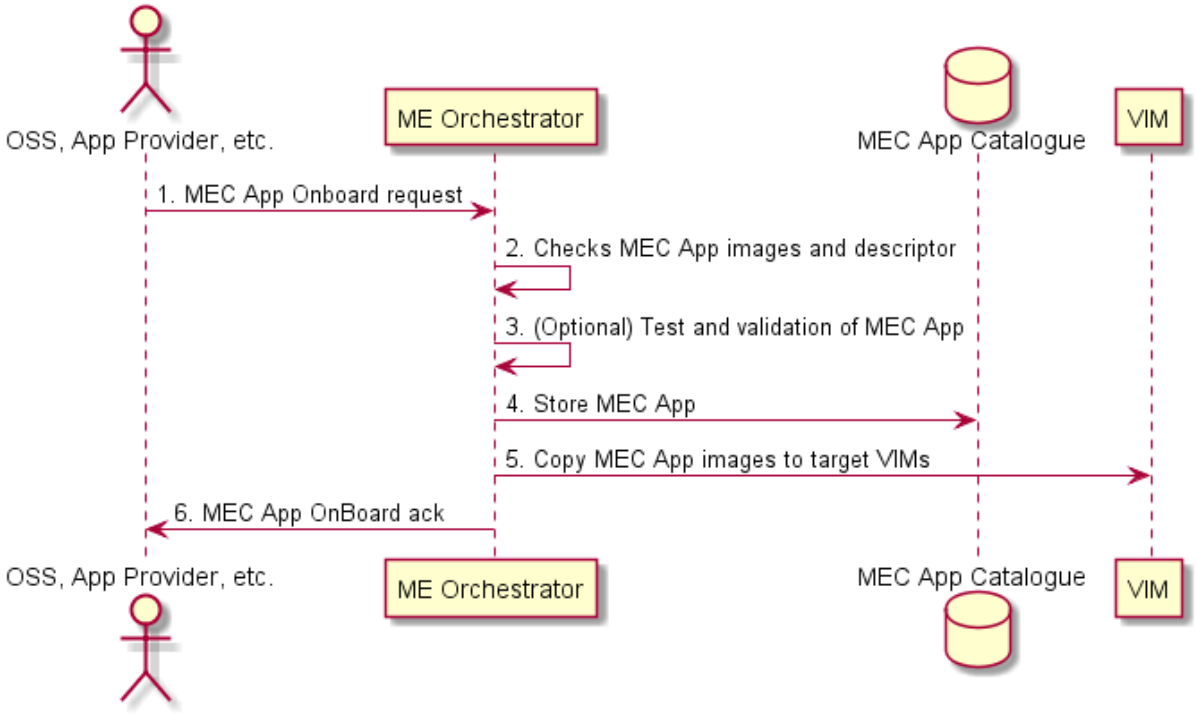


Figure 31: Orchestration Flows: MEC App On-boarding.

The main steps for MEC App on-boarding are:

1 MEO is requested to on-board a MEC App Package, which includes the MEC App Descriptor, as well as a set of images. The sources of this request can be OSSs or MEC App providers.

2 MEO checks the MEC App Descriptor and images, namely:
   a) Validating the correctness of the MEC App Descriptor (format, mandatory items, etc.)
   b) Validating the integrity and authenticity of the images (format, checksum).

3 Optionally, MEO may test the MEC App, e.g. by deploying (in a test VIM) the images using the Descriptor's definitions.

4 MEO stores the images and the Descriptor in the MEC App Catalogue.

5 Images are copied to all VIMs (edges) where this MEC App can potentially be deployed.

6 MEO acknowledges the on-boarding request to the entity who issued the request (links to step 1).

8.2.2.2 MEC App Instantiation

The following Figure depicts the MEC App instantiation flow.

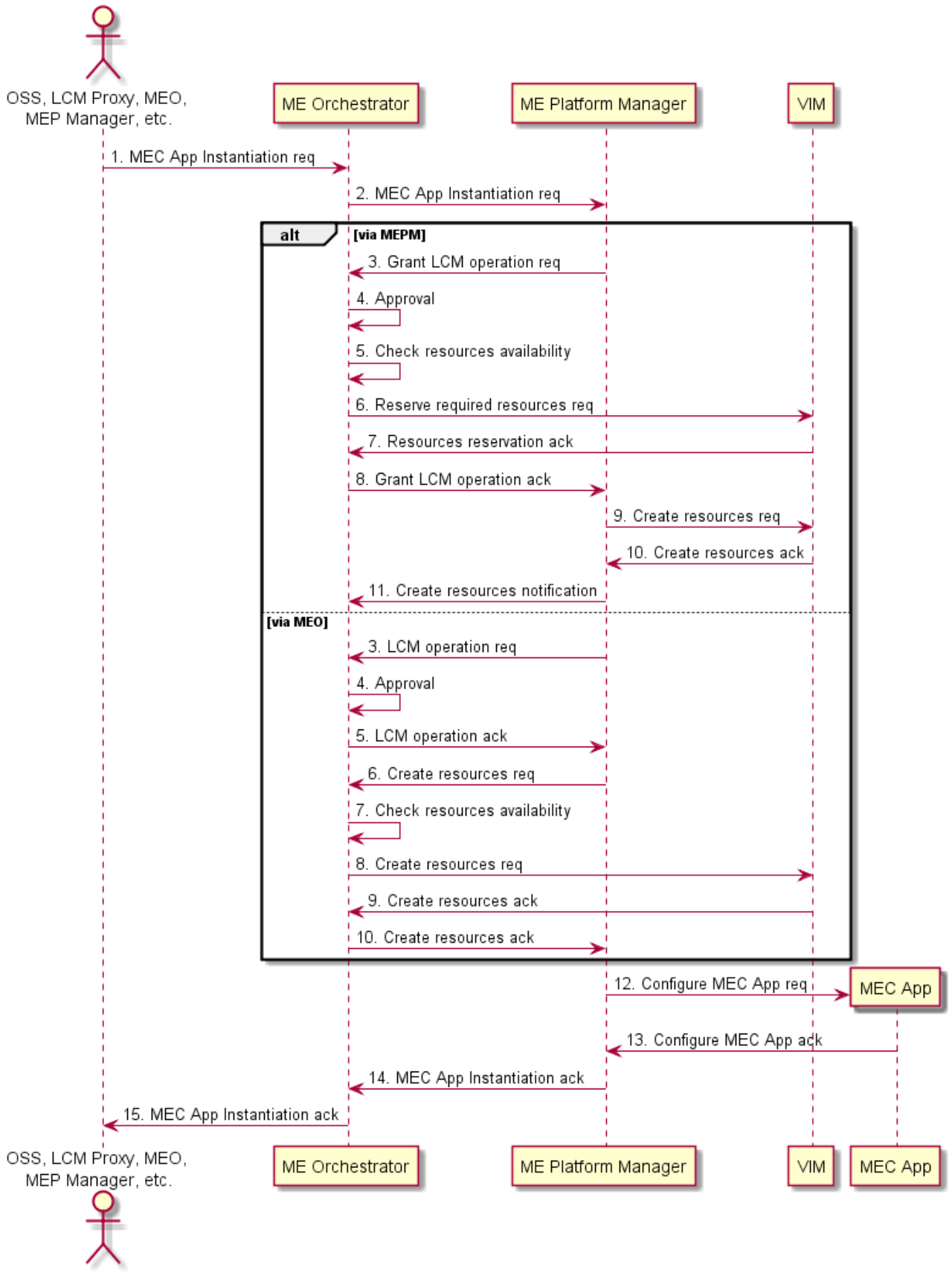


Figure 32: Orchestration Flows: MEC App Instantiation.


The main steps for MEC App instantiation are:

1 MEO is requested to instantiate a MEC App (previously on-boarded). Sources of this request can be OSSs, LCM Proxy, MEO itself (internal decision) or MEP Manager (MEPM).

2 MEO finds the MEPM in change for this MEC App on the target edge and requests to it the MEC App instantiation.

Option 1 (Resource management performed by MEPM directly to the VIM)

3 MEPM requests granting to MEO for the instantiation of a MEC App according to the MEC App Descriptor (CPU, Memory, IP, etc.). This will both authorize the instantiation operation and reserve resources.

4 MEO approves MEPM requested instantiation operation.

5 MEO checks resources availability, considering the grant request and the available resources on its database.

6 MEO performs the resources reservation for MEC App instantiation in the appropriated VIM.

7 VIM acknowledges the resources reservation (links to step 6).

8 MEO acknowledges to MEPM the request for granting both authorization and resources (links to step 3).

9 MEPM requests to VIM the resources, previously granted, for MEC App instantiation.

10 VIM acknowledges MEPM for the creation of resources (links to step 9).

11 MEPM notifies MEO, informing that the reserved resources were created.

Option 2 (Resource management performed via MEO on behalf of VIM)

3 MEPM requests to MEO an authorization to instantiate MEC App according to the MEC App Descriptor (CPU, Memory, IP, etc.).

4 MEO approves to MEPM the requested instantiation operation.

5 MEO acknowledges to MEPM for MEC App instantiation (links to step 3).

6 MEPM requests to MEO the creation of the required resources. MEO acts as a proxy to the appropriate VIM, asking for resources on MEPM behalf.

7 MEO checks resources availability on its resources database.

8 MEO requests to VIM the creation of the required resources (on MEPM's behalf).

9 VIM acknowledges MEO for the creation of resources (links to step 8).

10 MEO acknowledges MEPM for the creation of resources (links to step 6).

End Options

12 Once MEC App resources are created and VDUs are up and running, MEC App is requested to configure deployment specific parameters.

13 MEC App acknowledges for the configuration of deployment specific parameters (links to step 12).

14 MEPM acknowledges the MEC App instantiation to MEO (links to step 2).

15 MEO acknowledges the MEC App instantiation to the initial source, which can be OSSs, LCM Proxy, MEO itself (internal decision) or MEPM (links to step 1).

8.2.2.3 MEC App Scaling Out

The following Figure depicts the MEC App scale-out flow.

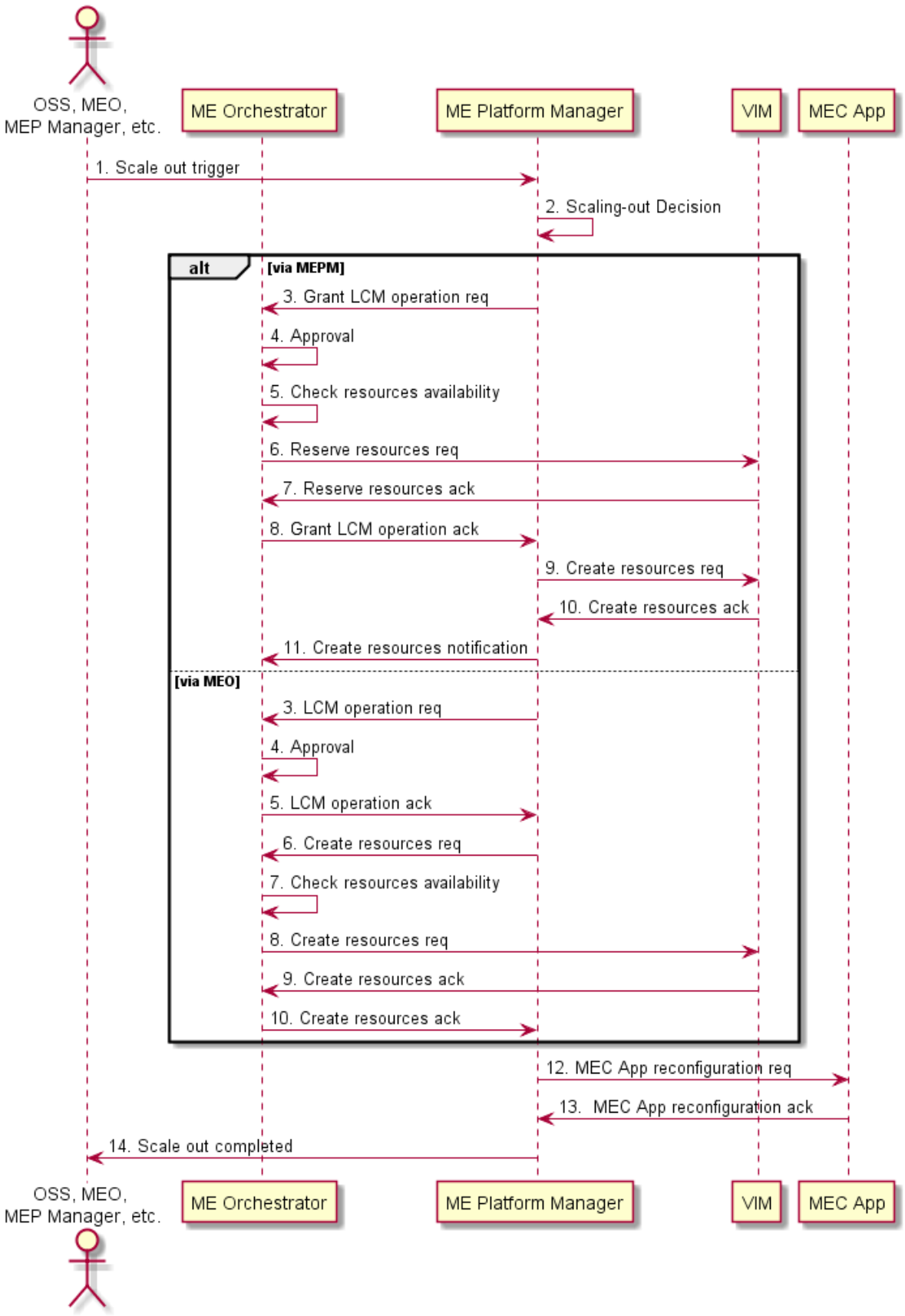


Figure 33: Orchestration Flows: MEC App Scale-out.


The main steps for MEC App scale-out are:

1 MEPM is triggered (e.g. monitory) by an external source (OSSs, MEO or MEPM itself - internal decision), indicating a service degradation (e.g. increasing latency), or the need for more resources (e.g. high CPU utilization).

2 MEPM decides to perform a scale-out based on the trigger received, expanding the MEC App.

Option 1 (Resource management performed by MEPM directly to the VIM)

3 MEPM requests granting to MEO for the scale-out of a MEC App according to MEC App Descriptor (CPU, Memory, IP, etc.). This will both authorize the scale-out operation and reserve additional resources.

4 MEO approves the MEPM request for the scale-out operation.

5 MEO checks resources availability, considering the request and the available resources on its resources database.

6 MEO performs the reservation of additional resources to scale-out the MEC App in the appropriate VIM.

7 VIM acknowledges the reservation of additional resources (links to step 6).

8 MEO acknowledges to MEPM the request for authorization and additional resources (links to step 3).

9 MEPM requests to the VIM additional resources, previously granted, for MEC App scale-out.

10 VIM acknowledges MEPM for the creation of additional resources (links to step 9).

11 MEPM notifies MEO, informing that the reserved additional resources were created.

Option 2 (Resource management performed via MEO on behalf of VIM)

3 MEPM requests to the MEO authorization to scale-out the MEC App according to the MEC App Descriptor (CPU, Memory, IP, etc.).

4 MEO approves the MEPM request to perform the scale-out operation.

5 MEO acknowledges the MEPM request to perform the MEC App scale-out (links to step 3).

6 MEPM requests to MEO the creation of additional resources. MEO acts as a proxy to the appropriate VIM, asking for additional resources on MEPM's behalf.

7 MEO checks resources availability on its resources database.

8 MEO requests to VIM the creation of additional resources (on MEPM's behalf).

9 VIM acknowledges MEO for the creation of additional resources (links to step 8).

10 MEO acknowledges MEPM for the creation of additional resources (links to step 6).

End Options

12 Once MEC App additional resources are created and VDUs are up and running, MEC App is requested by MEPM to reconfigure deployment specific parameters, in order to expand capacity.

13 MEC App acknowledges MEPM for the reconfiguration of deployment specific parameters (links to step 12).
14 MEPM acknowledges the MEC App scale-out to the MEO (links to step 2).
15 MEO acknowledges the MEC App scale-out to the initial source, which can be OSSs, MEO or MEPM itself (links to step 1).

8.2.2.4 MEC App Scaling In

The following Figure depicts the MEC App scale-in flow.

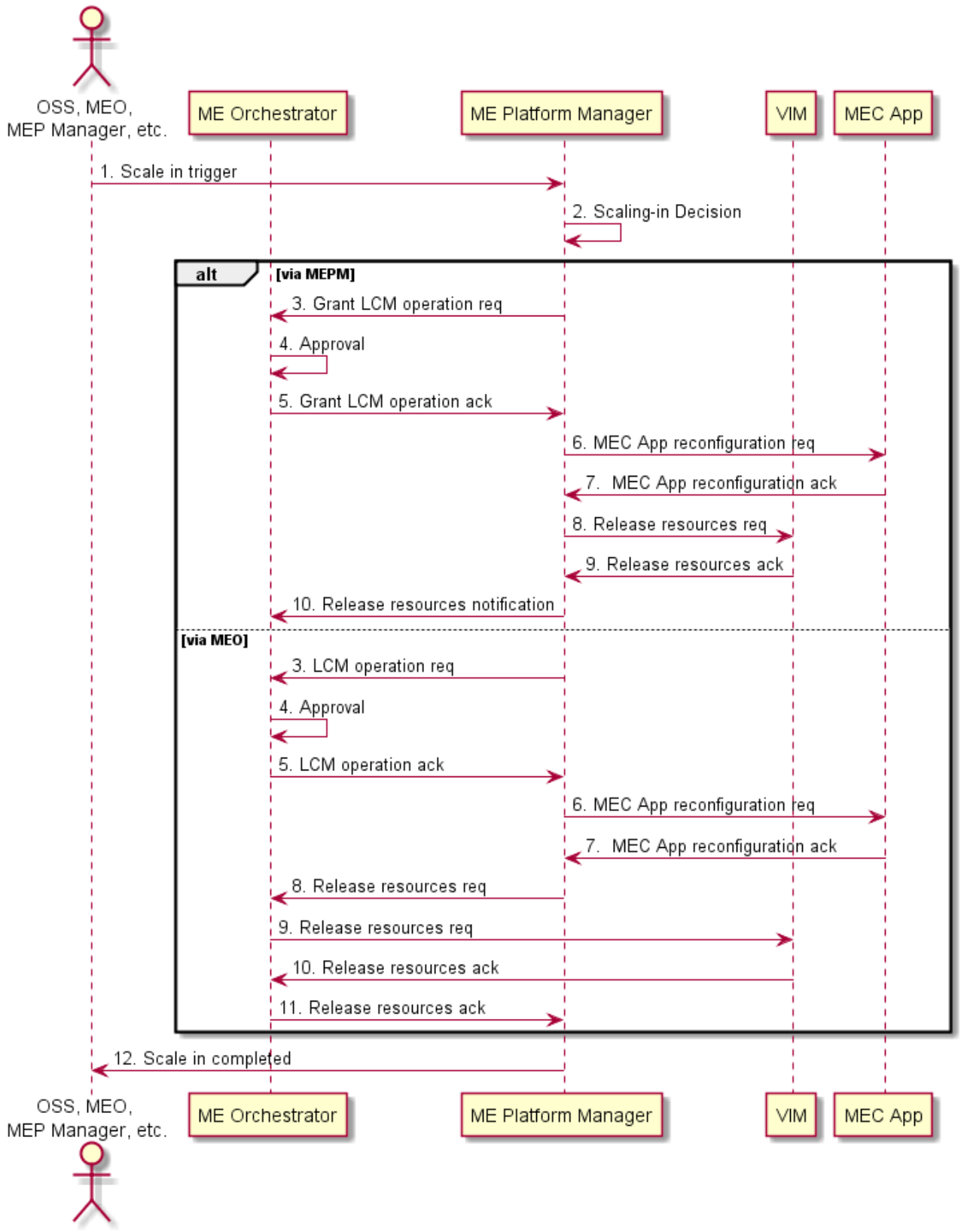


Figure 34: Orchestration Flows: MEC App Scale-in.

The main steps for MEC App scale-in are:


1 MEPM is triggered (e.g. monitory) by an external source (OSSs, MEO or MEPM itself - internal decision), indicating an over-provisioning of the MEC App (e.g. very low latency), or low resources utilization (e.g. very low CPU utilization).
2 MEPM decides to perform a scale-in based on the trigger received, contracting the MEC App.

Option 1 (Resource management performed by MEPM directly to the VIM)

3 MEPM requests grant to MEO to scale-in a MEC App according to MEC App Descriptor (CPU, Memory, IP, etc.). This will authorize the scale-in operation.
4 MEO approves the MEPM scale-in operation.
5 MEO acknowledges to MEPM the request for authorization and additional resources (links to step 3).
6 Before some MEC App resources can be removed, the MEPM requests to the MEC App to reconfigure deployment specific parameters, in order to contract the MEC App capacity.
7 MEC App acknowledges MEPM for the reconfiguration of deployment specific parameters (links to step 6).
8 MEPM requests VIM to dispose unused resources from the MEC App.
9 VIM acknowledges MEPM for the disposal of unused resources (links to step 8).
10 MEPM notifies MEO, informing that the disposal of unused resources was done.

Option 2 (Resource management performed via MEO on behalf of VIM)

3 MEPM requests to MEO authorization to scale-in the MEC App according to the MEC App Descriptor (CPU, Memory, IP, etc.).
4 MEO approves to MEPM the scale-in operation.
5 MEO acknowledges to MEPM the MEC App scale-in (links to step 3).
6 Before some MEC App resources can be removed, MEC App is requested by MEPM to reconfigure deployment specific parameters without those resources, in order to contract the MEC App capacity.
7 MEC App acknowledges MEPM for the reconfiguration of deployment specific parameters (links to step 6).
8 MEPM requests to MEO the removal of unused resources. MEO acts as a proxy to the appropriate VIM, asking for resources disposal on MEPM's behalf.
9 MEO requests to VIM the removal of unused resources (on MEPM's behalf).
10 VIM acknowledges MEO for the removal of unused resources (links to step 9).
11 MEO acknowledges MEPM for the removal of unused resources (links to step 8).

End Options

12 MEO acknowledges the MEC App scale-out operation to the initial source, which can be OSSs, MEO or MEPM itself (links to step 1).

8.2.2.5 MEC App Relocation

The following Figure depicts the MEC App relocation flow.

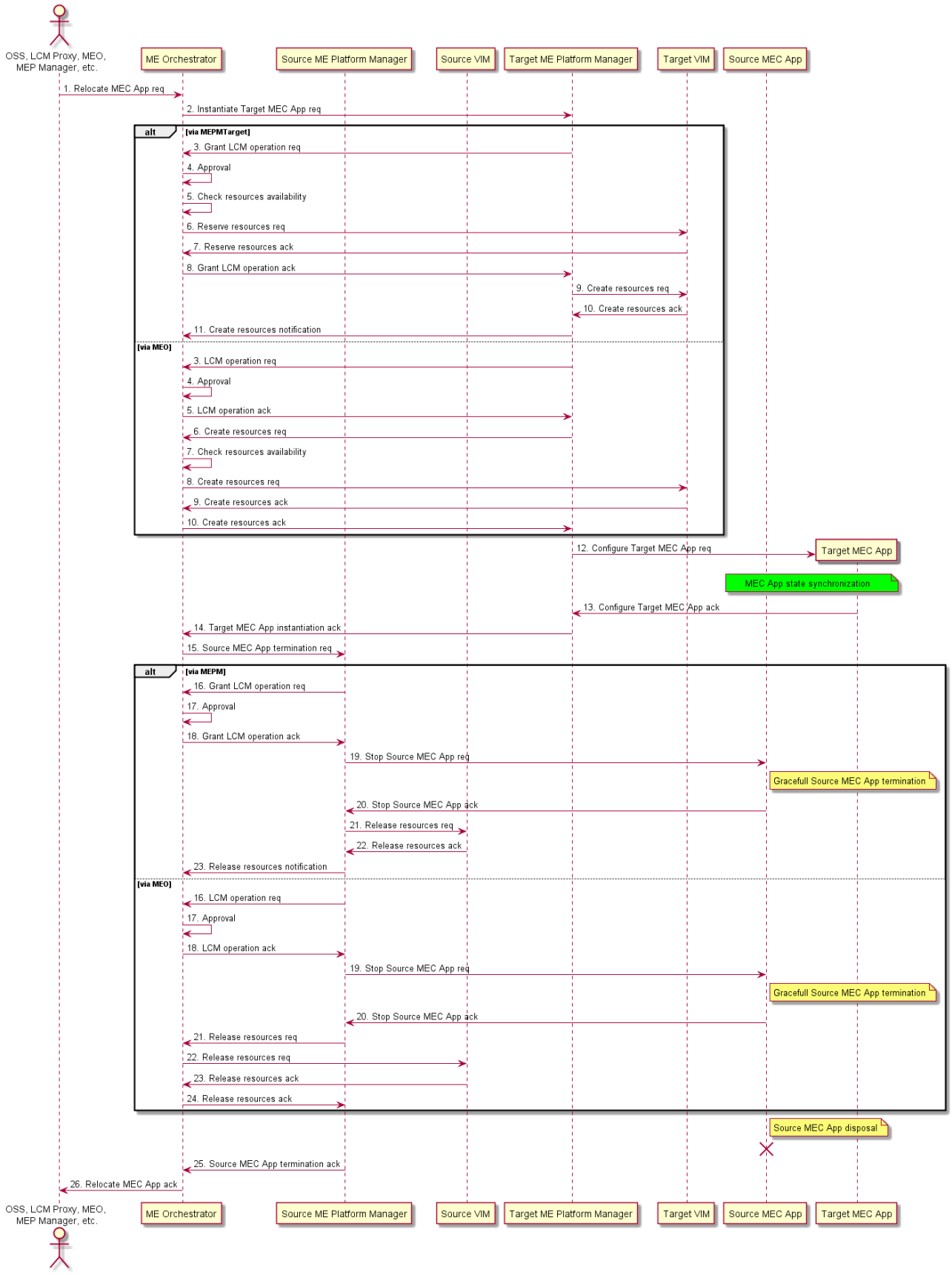


Figure 35: Orchestration Flows: MEC App Relocation.


The main steps for MEC App relocation are:

Note 1: It is assumed a strategy of instantiation of the target MEC App instance followed by the termination of the source MEC App. Other options (e.g. live migration) are also possible.

Note 2: It is assumed a relocation between two regions managed by different VIMs.

1 MEO is requested to relocate a MEC App (previously instantiated). Initial sources of this request can be OSSs, LCM Proxy, MEO itself (internal decision) or MEP Manager (MEPM).

Target Instantiation Phase

2 MEO finds the MEPM in change for this MEC App on the target edge and requests the target MEC App instantiation.

Option 1 (Resource management performed by MEPM directly to the VIM)

3 MEPM requests to the MEO the instantiation of the target MEC App according to MEC App Descriptor (CPU, Memory, IP, etc.). This will both authorize the instantiation operation and reserve resources.
4 MEO approves the MEPM instantiation operation.
5 MEO checks resources availability, considering the request and the available resources on its resources database.
6 MEO performs the resources reservation for target MEC App instantiation in the appropriate target VIM.
7 Target VIM acknowledges the resources reservation (links to step 6).
8 MEO acknowledges to MEPM the request for authorization and resources (links to step 3).
9 MEPM requests to target VIM the creation of resources, previously granted, for target MEC App instantiation.
10 Target VIM acknowledges MEPM for the creation of resources (links to step 9).
11 MEPM notifies MEO, informing that the reserved resources were created.

Option 2 (Resource management performed via MEO on behalf of VIM)

3 MEPM requests to MEO authorization to instantiate the target MEC App according to the MEC App Descriptor (CPU, Memory, IP, etc.).
4 MEO approves to MEPM the instantiation operation.
5 MEO acknowledges to MEPM for target MEC App instantiation (links to step 3).
6 MEPM requests to MEO the creation of the resources. MEO acts as a proxy to the appropriate target VIM, asking for resources on MEPM's behalf.
7 MEO checks resources availability on its resource database.
8 MEO requests to the target VIM the creation of resources (on MEPM's behalf).
9 Target VIM acknowledges MEO for the creation of resources (links to step 8).
10 MEO acknowledges MEPM for the creation of resources (links to step 6).

End Options

12 Once target MEC App resources are created and VDUs are up and running, target MEC App is requested to configure deployment specific parameters.
<Source and target MEC Apps should sync their states if needed (MEC App dependent). This is a MEC App specific procedure>
13 Target MEC App acknowledges for the configuration of deployment specific parameters (links to step 12).
14 MEPM acknowledges the target MEC App instantiation to MEO (links to step 2).

Source Termination Phase

15 MEO finds the MEPM in charge for the source MEC App on the edge it is running, and requests the source MEC App termination.

Option 1 (Resource management performed by MEPM directly to the VIM)

16 MEPM requests to MEO for the termination of the source MEC App. This will authorize the termination operation.
17 MEO approves the MEPM termination operation.
18 MEO acknowledges the MEPM to perform the termination operation (links to step 16).
19 MEPM stops gracefully the source MEC App service before dispose resources.
20 MEC App acknowledges the service stop to MEPM (see step 19).
21 MEPM requests the source VIM to dispose resources.
22 Source VIM acknowledges MEPM for the disposal of resources (see step 21).
23 MEPM notifies MEO, informing that the resources were disposed.

Option 2 (Resource management performed via MEO on behalf of VIM)

15 MEPM requests to MEO authorization to terminate source MEC App. This will authorize the termination operation.
16 MEO approves MEPM the requested termination operation.
17 MEO acknowledges to MEPM for source MEC App instantiation (links to step 16).
18 MEPM stops gracefully the source MEC App service before dispose resources.
19 Source MEC App acknowledges the service stop to MEPM (see step 19).
20 MEPM requests to MEO the disposal of source MEC App resources.
21 MEO requests to source VIM the disposal of source MEC App resources (on MEPM's behalf).
22 Source VIM acknowledges MEO for the disposal of resources (links to step 21).
23 MEO acknowledges MEPM for the disposal of resources (links to step 20).

End Options

24 Once the source MEC App resources are released, MEPM acknowledges the source MEC App termination to MEO (links to step 15).
25 MEO acknowledges the MEC App relocation to the initial source, which can be OSSs, LCM Proxy, MEO itself (internal decision) or MEP Manager (MEPM) (links to step 1).

8.2.2.6 MEC App Termination

The following Figure depicts the MEC App termination flow.

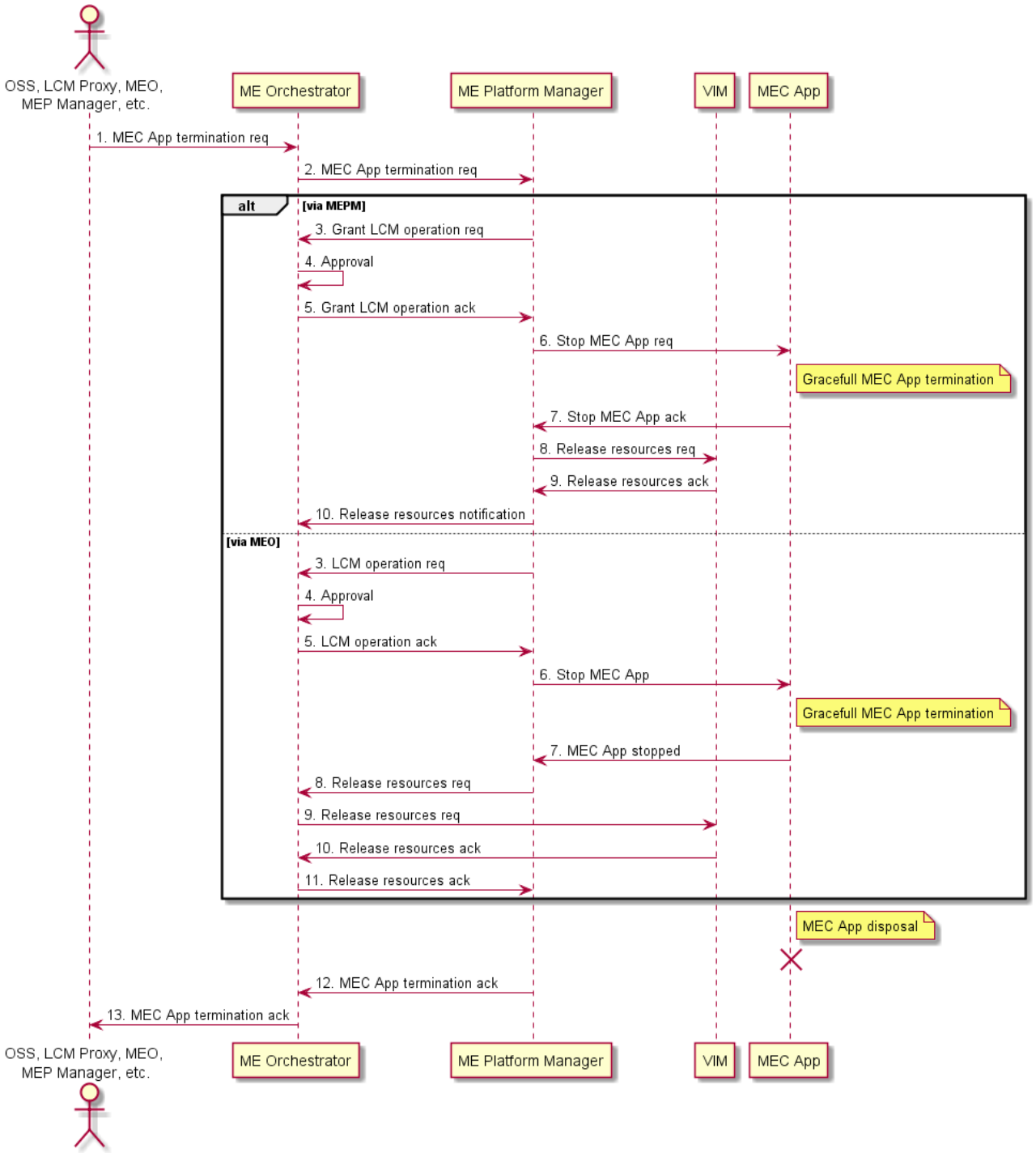


Figure 36: Orchestration Flows: MEC App Termination.

The main steps for MEC App termination are:

1 MEO is requested to terminate a MEC App (previously instantiated). Initial sources of this request can be OSSs, LCM Proxy, MEO itself (internal decision) or MEP Manager (MEPM).

2 MEO finds the MEPM in charge for this MEC App on the edge it is running and requests the MEC App termination.

Option 1 (Resource management performed by MEPM directly to the VIM)

3 MEPM requests grants to the MEO for the termination of a MEC App. This will authorize the termination operation.
4 MEO approves the MEPM termination operation.
5 MEO acknowledges the MEPM to perform the termination operation (links to step 3).
6 MEPM stops gracefully the MEC App service before dispose resources.
7 MEC App acknowledges the service stop to MEPM (see step 6).
8 MEPM requests the VIM to dispose resources.
9 VIM acknowledges MEPM for the disposal of resources (see step 8).
10 MEPM notifies MEO, informing that the resources were disposed.

Option 2 (Resource management performed via MEO on behalf of VIM)

3 MEPM requests to the MEO authorization to terminate the MEC App. This will authorize the termination operation.
4 MEO approves to MEPM the termination operation.
5 MEO acknowledges to MEPM for MEC App termination (links to step 3).
6 MEPM stops gracefully the MEC App service before dispose resources.
7 MEC App acknowledges the service stop to the MEPM (see step 6).
8 MEPM requests to MEO the disposal of MEC App resources.
9 MEO requests to VIM the disposal of MEC App resources (on MEPM's behalf).
10 VIM acknowledges MEO for the disposal of resources (links to step 9).
11 MEO acknowledges MEPM for the disposal of resources (links to step 8).

End Options

12 Once the MEC App resources are released, MEPM acknowledges the MEC App termination to MEO (links to step 2).
13 MEO acknowledges the MEC App termination to the initial source, which can be OSSs, LCM Proxy, MEO itself (internal decision) or MEP Manager (MEPM) (links to step 1).