

**K-ONE Technical Document #36**  
**Integrated management system for High**  
**availability in multi-site**

**Document No. K-ONE #28**

**Version 0.2**

**Date 2018-12-26**

**Author(s) Hyunsik Yang, Minwook Kim**

## ■History

Version	Date	Author(s)	Contents
Draft - 0.1	2018. 12. 10	SSU Team	Write first draft
0.2	2018. 12. 26	SSU Team	Fix document compatibility issue

이 문서는 2018년도 정부(과학기술정보통신부)의 재원으로  
정보통신기술진흥센터의 지원을 받아 수행된 연구임 (No. 2015-0-  
00575, 글로벌 SDN/NFV 공개 소프트웨어 핵심 모듈/기능 개발)

This work was supported by Institute for Information & Communications  
Technology Promotion (IITP) grants funded by the Korea government  
(MSIT) (No. 2015-0-00575, Global SDN/NFV Open-Source Software Core  
Module/Function Development)

## **Summary**

Network function virtualization (NFV) provides flexible and scalable network services by leveraging software-based network appliances. In an NFV architecture, the virtual network function manager (VNFM) has a critical role in provisioning, configuring, and operating virtual network functions (VNFs). In addition, the VNF manager should provide other functions to guarantee high availability of network services, such as fault management, monitoring and auto scaling. Currently, many open source projects aim to implement these functions, but there is no standard method of monitoring and detecting failure of VNFs. Moreover, there is no integration model for High Availability in the multisite environment. Therefore, we review current research of High availability and describe what we done for HA in OpenStack environment.

---

## Contents

### **K-ONE #36. Integrated management for High availability in multi-site**

1. Review of High Availability .....	10
1.1. Introduction.....	10
1.2. ETSI GS NFV-Reliability .....	10
1.3. Monitoring Mechanism for HA.....	21
1.3.1. Fault Management Mechanism .....	21
1.3.2. Doctor.....	22
1.3.2.1. Doctor architecture .....	22
1.3.3. Monitoring tool.....	23
2. High availability in Tacker VNF Manager .....	24
2.1 Problem description .....	24
2.2 Architecture .....	26
2.3 Data Model.....	31
2.4 Demo results.....	32
3. Monitoring tool integration in Tacker VNF Manager.....	34
3.1. Design and implementation of monitoring tool interworking structure.....	34
3.2. Verification and analysis .....	37
3.3. Conclusion.....	41
4. Root Cause Analysis for High Availability .....	42
4.1. Action List for Root Cause.....	42

5. Multisite NFV Management System .....	44
5.1. Introduction.....	44
5.2. High availability for NFV multi-site environment.....	45
5.2.1. Detailed Architecture .....	46
5.2.2. Implementation.....	46
5.2.3. Use case 1- High Availability for IoT Cloud .....	50
5.2.3.1. Introduction.....	50

## List of Figures

Figure 1 An active-standby configuration of NFs .....	11
Figure 2 An example of the Active-Standby method in traditional environments ....	12
Figure 3 Failover flow for a traditional system with the Active-Standby method .....	12
Figure 4 A typical Active-Standby configuration of VNFs in the NFV architecture ...	13
Figure 5 Active-Standby method in the virtualised environment.....	14
Figure 6 Active-Standby failover in the virtualised environment.....	14
Figure 7 A failover procedure of VNFs in the Active-Standby configuration .....	14
Figure 8 Stateful VNFs with internal storage for states in the Active-Standby configuration .....	15
Figure 9 A failover procedure of VNFs synchronizing state information in the Active-Standby configuration .....	16
Figure 10 An Active-Active configuration of NFs.....	16
Figure 11 An example of remediation mechanism with Active-Active method in traditional physical environments.....	17
Figure 12 An example of Active-Active method in the NFV environment.....	18
Figure 13 A load balancing model in virtualised environments .....	19
Figure 14 Load balancing model with internal storage .....	20
Figure 15 A sample procedure that will occur when a VNF with the service application fails .....	21
Figure 16 Doctor Project Component.....	22
Figure 17 Fault Management Procedure for HA.....	23

Figure 18 Architecture for HA in Tacker.....	25
Figure 19 Cluster creation via CLI .....	32
Figure 20 Ping monitor driver detects VNF not working and execute recovery action .....	32
Figure 21 Recover completed .....	33
Figure 22 Interworking between and Open source monitoring tool in NFV environment .....	34
Figure 23 VNFM monitoring driver design structure .....	35
Figure 24 NFV environment configuration component .....	36
Figure 25 Zabbix VNFD Tosca Template .....	36
Figure 26 Open source monitoring tool in VNF Agnet Automatically install script..	37
Figure 27 API Request & Response .....	38
Figure 28 When the VNF generation is initialized, the user-data script log .....	38
Figure 29 Monitoring Information for each VDU .....	38
Figure 30 Trigger list based on condition value .....	39
Figure 31 Trigger Alarm Check for Fault Occurrence .....	39
Figure 32 Data collection changes due to outages .....	40
Figure 33 Custom actions based on triggers .....	40
Figure 34 Test Workflow .....	41
Figure 35. ETSI NFV Architecture with multiple-VIM .....	44
Figure 36. NFV multi-site environment management system architecture .....	45
Figure 37. Detailed Architecture of NFV multi-site environment management .....	46
Figure 38. Multi-Site topology GUI .....	47

Figure 39. The example of Management System UI.....	48
Figure 40. Testing Component Function result.....	48
Figure 41. Checking Component Function result.....	49
Figure 42. Automation Monitoring script.....	49



## **K-ONE #36. Integrated management system for High availability in multi-site**

## 1. Review of High Availability

### 1.1. Introduction

For a high availability system, failure detection, isolation and recovery should be automatic and reliable. NFV can provide support for a range of HA mechanisms, such as redundancy, heartbeats, data synchronization, clustering and periodic snapshots or lock-step shadow copies to provide stateful service recovery. The OSS/BSS may already have some level of HA designed into their provisioning of the VNFs and services. In initial stages as operators move NFs to NFV, they may leave these existing mechanisms in place, and use the NFV opaquely for abstraction of resource pool. However, as they design NFV-aware services, they can leverage HA mechanisms offered by NFV. To support HA Mechanism, ETSI also defined HA Definition and OPNFV also implement the part of the HA function in NFV architecture. To check the current status of HA research, we review the document that is related to HA, and review the implementation in OPNFV.

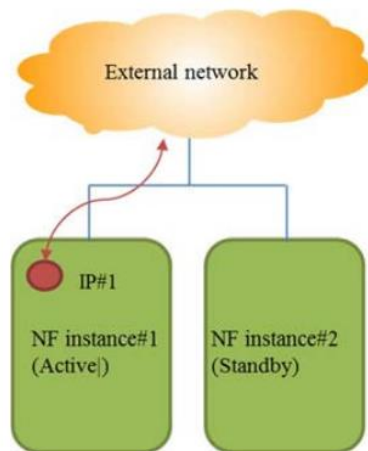
Finally, we explained our result of HA in this document.

### 1.2. ETSI GS NFV-Reliability

The document ETSI GS NFV REL 3 presents some VNF protection schemes, with which the reliability of NFs is realized. VNF protection schemes investigated include traditional Active-Standby method, Active-Active method and load balancing method with state transfer among VNFs[1][2].

#### **1. Active-Standby method**

The Active-Standby method is one of the popular redundancy methods adopted in many high availability systems. This configuration is depicted in Figure 1. Note that the Active-Standby method is referred to as one of the 2N redundancy model (N=1) in "Service Availability Forum Application Interface Specification (SA Forum AIS)".



**Figure 1 An active-standby configuration of NFs**

### State protection

Some NFs, such as stateful SIP proxies, use session state information for their operation. In this case, the active NF shares the state information with its standby NF to enable service continuity after a switch-over. The same principle applies for VNF Active-Standby protection schemes. There are two methods to store state information. The state information can be externalized or stored\_at/shared\_among peer mates. Restoration mechanisms with externalised state data have been described in ETSI GS NFV-REL 001. Hence, this clause focuses on the latter case where state information is stored at peer mates. Since shared state information needs to be kept consistent within the system, checkpointing or other methods will be used as described in ETSI GS NFV-REL 002[3].

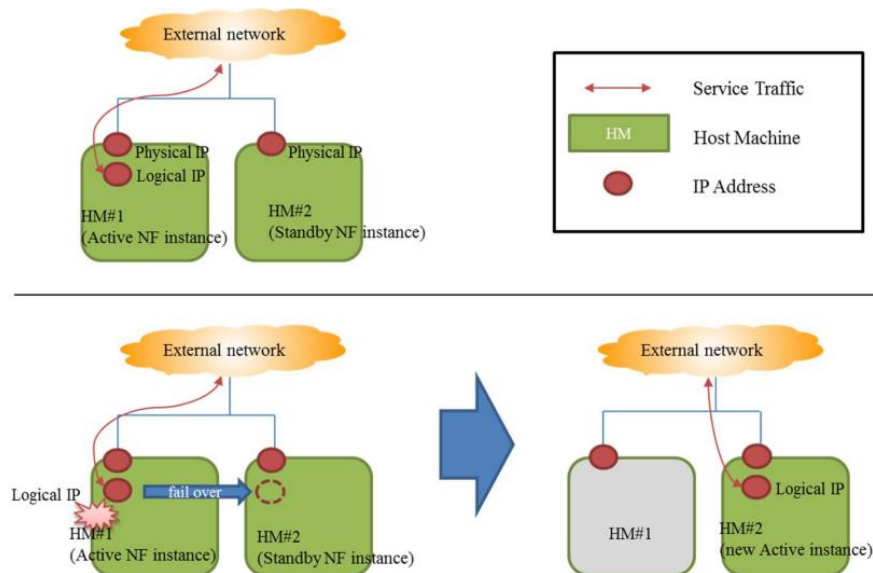
### Recovery and remediation phase

When an active NF instance fails, the standby NF instance takes over the active role so that the availability of the system is maintained. The possible main steps are as follows:

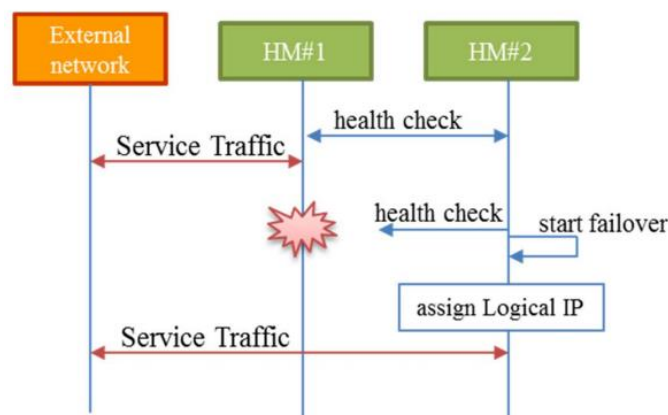
- 1) Failure detection of the active NF instance.
- 2) Failover from the former active NF instance to the former standby NF instance. The former standby instance becomes the new active instance.
- 3) Replacement of the failed NF instance with a new one which becomes the new standby NF instance.

4) Start of the state information replication to the new standby NF instance.

Transparent failover in traditional deployments is enabled by assigning one logical IP address that the clients use to communicate with the system. In addition, each instance has its own fixed IP address used for designating each NF instance distinctively. During normal operation, the active NF instance serves the logical IP address. If the active NF instance fails and the failure is detected, the former standby NF instance will become active, and starts serving packets destined to the logical IP address. Hence, the system can continue providing the service.

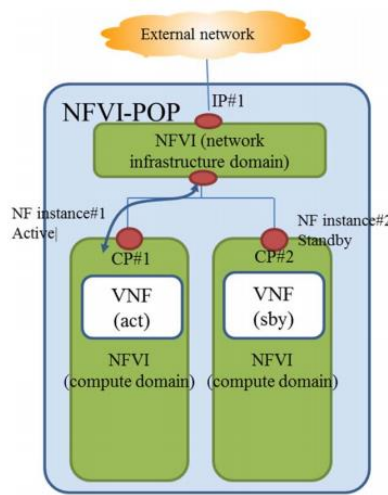


**Figure 2** An example of the Active-Standby method in traditional environments



**Figure 3** Failover flow for a traditional system with the Active-Standby method

In NFV deployments, the active VNF and clients on the external network are typically connected via NFVI (network infrastructure domain). This makes failover mechanism of virtualised systems different from that of the traditional ones. Since the interconnection between the active VNF and the next NF in the service chain is managed by NFV-MANO, NFV-MANO's availability influences the availability of the NS



**Figure 4 A typical Active-Standby configuration of VNFs in the NFV architecture**

In this configuration, the assumption is that the active VNF and the standby VNF are deployed in a location-disjoint mode. For this, NFV-MANO need to support anti-affinity rules for deploying VNFs so that each VNF instance is installed on a different compute domain. Each VNF instance has an internal connection point (CP), with an associated IP address. The two internal connection points are aggregated to a single connection point that is visible to systems on the external network. For example, a virtual router connects the IP address assigned to the external CP with the internal CP of the active VNF. Thereby, the active VNF can communicate with clients on the external network and can provide services. Like in traditional systems with Active-Standby protection, the standby VNF periodically checks the operational status of the active VNF, e.g. using a heartbeat mechanism, which enables failure detection of the active system within the required time for the service. Though NFV-MANO may also perform health check of the VNFs, it increases dependency on NFV-MANO to propagate information concerning the failure of an active VNF to the standby VNF via NFV-MANO. In case the active VNF fails, the standby VNF will initiate the fail-over procedure. It is realised by reassigning the external CP to the internal CP of the former standby VNF.

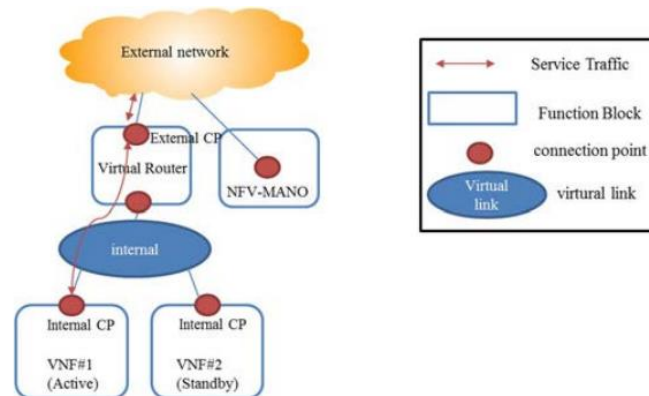


Figure 5 Active-Standby method in the virtualised environment

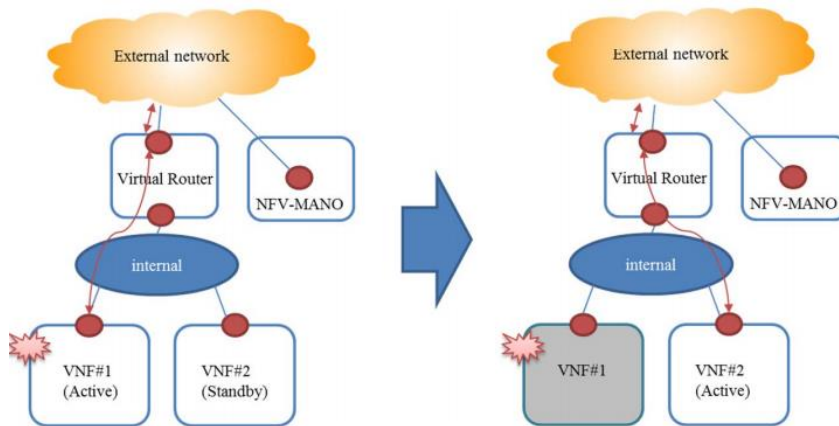


Figure 6 Active-Standby failover in the virtualised environment

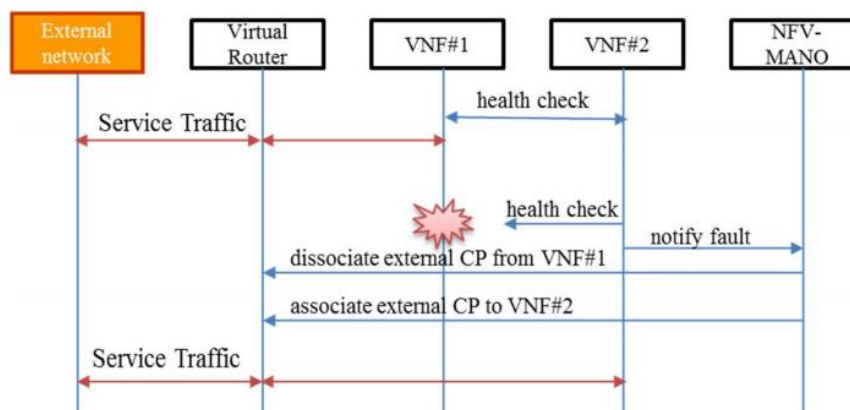
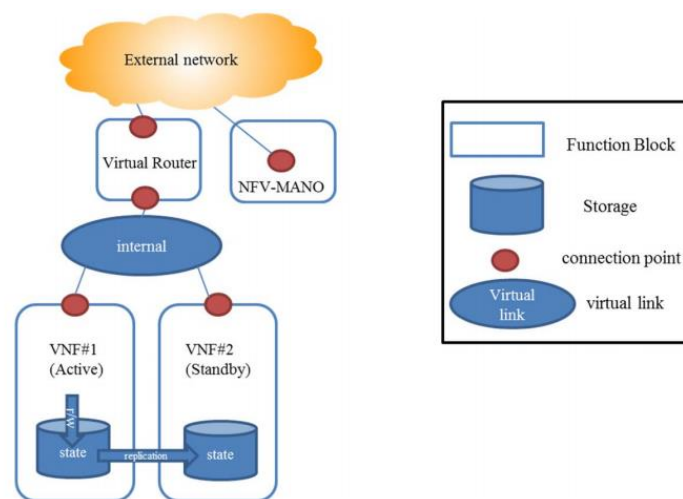
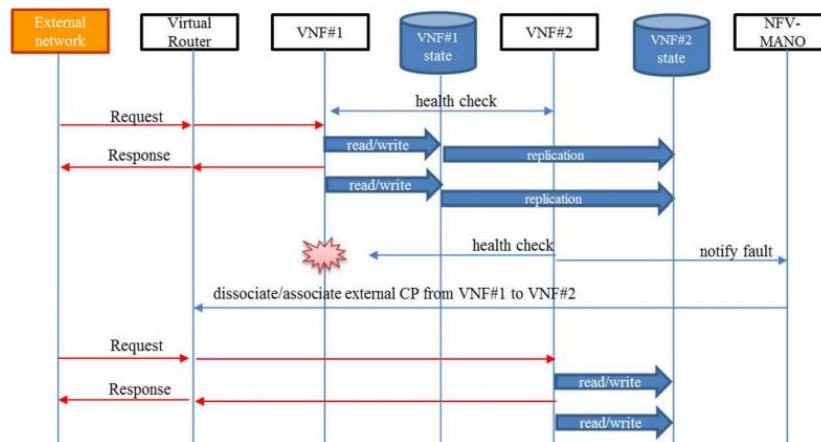


Figure 7 A failover procedure of VNFs in the Active-Standby configuration

In NFV environments, the mapping of the external CP to the internal CP is configured on the virtual router. This mapping cannot be reconfigured without NFV-MANO. Therefore, NFV-MANO should be informed about the failure of the active VNF, and associate the external CP with the internal CP of the former standby VNF instance. Moreover, it is also expected that NFV-MANO instantiates a new standby VNF. Note that the time required for changing the CP mapping affects the availability of the system, since the service is unavailable before reconfiguration, and that the time needed for instantiating a new standby VNF is related to the remediation period's duration. In addition to reconfiguring the CP mapping of the system, further consideration to keep the system's state information consistent is required. Figure 8 shows VNFs configured with the Active-Standby method with state replication between VNFs. The active VNF stores session state information in a storage within itself, and replicates the information to the storage within the standby VNF. Figure 9 shows the restoration procedure for this setup. Since the new active VNF has to check the consistency of the backup state information to judge whether the service can be continued before receiving the subsequent service traffic flow, the route change request sent to NFV-MANO should be triggered by the new active (i.e. the former standby) VNF. For the remainder of this clause, only Active-Standby configurations with state synchronization are described since they are a superset of those configurations without state synchronization.



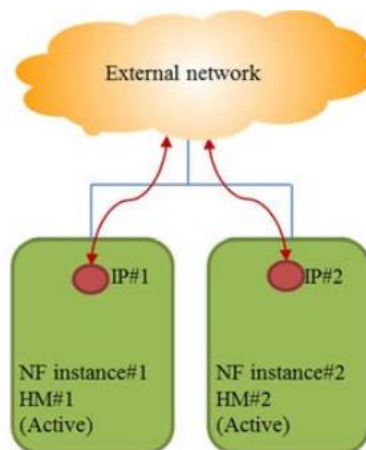
**Figure 8 Stateful VNFs with internal storage for states in the Active-Standby configuration**



**Figure 9 A failover procedure of VNFs synchronizing state information in the Active-Standby configuration**

## 2. Active-Active method

The Active-Active method is a configuration where two NF instances are simultaneously active for the same service (Figure 10). The external network can receive the service by accessing either IP addresses of these NF instances. This configuration is referred to as an Active-Active redundancy configuration (N-way active redundancy model N=2) in the SA Forum AIS document.



**Figure 10 An Active-Active configuration of NFs**

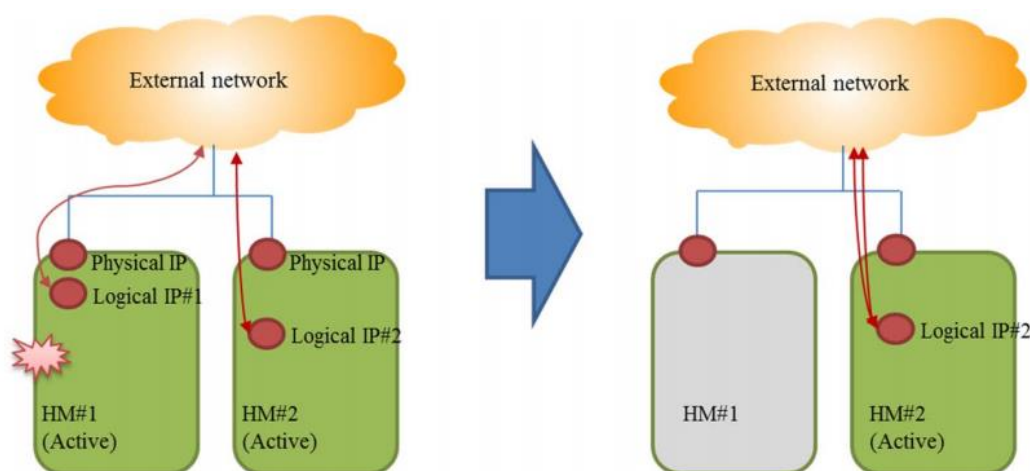


## State protection

This configuration does not contain a standby system. Therefore, externalisation should be used to maintain the state information for service continuity in case of system failure. The same principle applies for VNF Active-Active protection schemes, and external storages provided by NFVI may be used for the state protection.

## Recovery and remediation phase

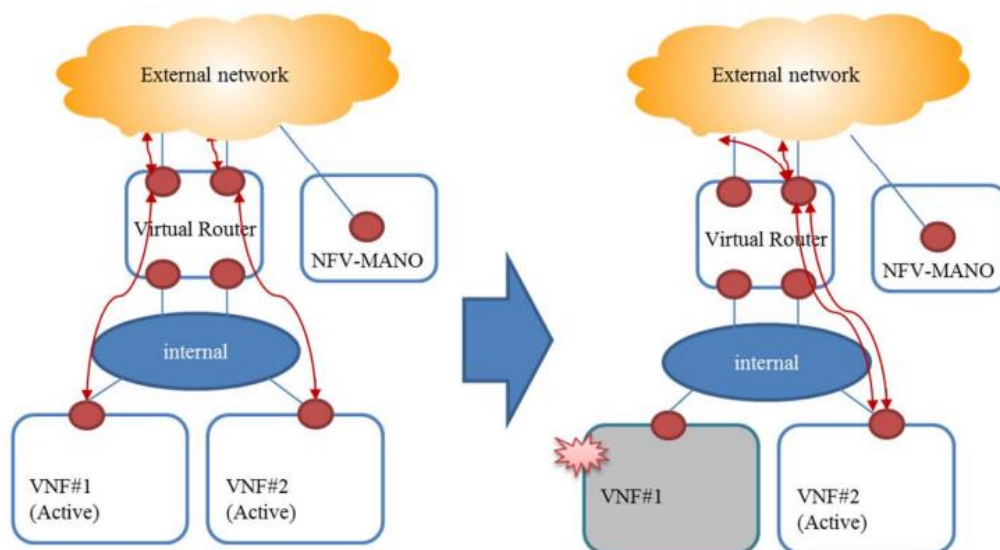
Figure 11 depicts a configuration change when NF instance#1 fails in a traditional system. External NFs can receive services from NF instance #2 when no answer is received from the NF instance #1, by changing the destination of requests for the NF from IP#1 to IP#2, which is usually done by the DNS in which the NFs are assigned to a primary NF and to a secondary NF. Note that when a load balancer is placed in front of the NF instances, external NFs can get services without knowing the failure of NF instance#1, since the load balancer can deliver all the requests to the NF instance #2 when the NF instance #1 fails



**Figure 11 An example of remediation mechanism with Active-Active method in traditional physical environments**

In NFV deployment, the active VNF and clients on the external network are connected via NFVI (network infrastructure domain) instead of using a logical IP address assigned to a virtual machine where the VNF is deployed. In this case, two VNFs are deployed on different compute domains according to anti-affinity rules. Two external CPs are assigned to the virtual router in a network infrastructure domain. Each external CP is connected respectively to an internal CP

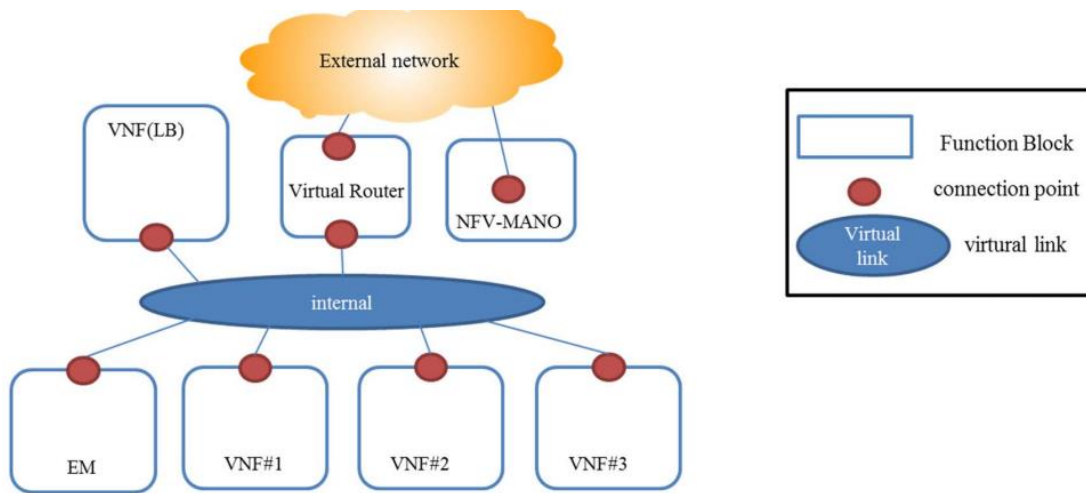
of each VNF. Figure 12 shows an example of Active-Active configuration in NFV environments. In this figure, NFV-MANO does not have to reconfigure the mapping of external CP and internal CP, since the clients in the external network change the destination of the request message so as to be able to get services from the VNF instance #2



**Figure 12 An example of Active-Active method in the NFV environment**

### **3. Load balancing Method**

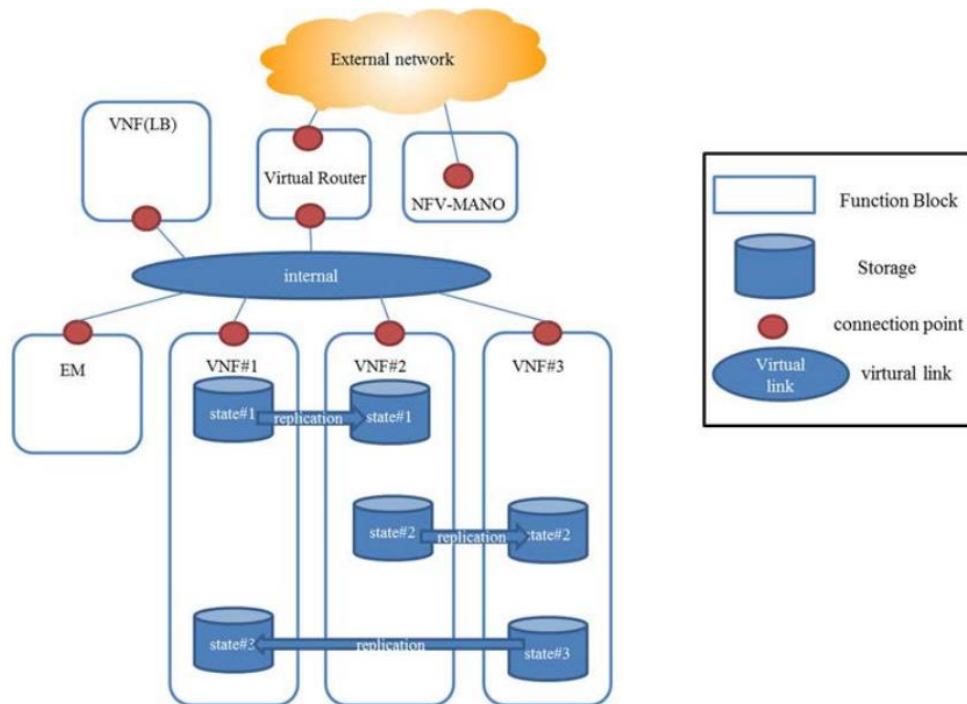
Load balancing is a popular method used for Web servers. In cloud environments, a load balancer is deployed in front of several VNFs. The service traffic goes through the load balancer and is distributed to several VNFs that handle the whole traffic (Figure 13). A load balancing method that corresponds to the N-way redundancy model of the SA Forum AIS document is investigated in this clause.



**Figure 13 A load balancing model in virtualised environments**

### State protection

When a VNF is stateful, e.g. SIP proxies, there are two methods to store state information. One method is to externalize the state information. The other is to share the state information among peer mates, which means that the state information of the service instances within VNFs have their standby information in other VNFs as shown in Figure 14. In this figure, the state information of the service instances running on VNF#1 is replicated in VNF#2 as a backup. Similarly, the one of service instances on VNF#2 (resp. #3) is replicated on VNF#3 (resp. #1)



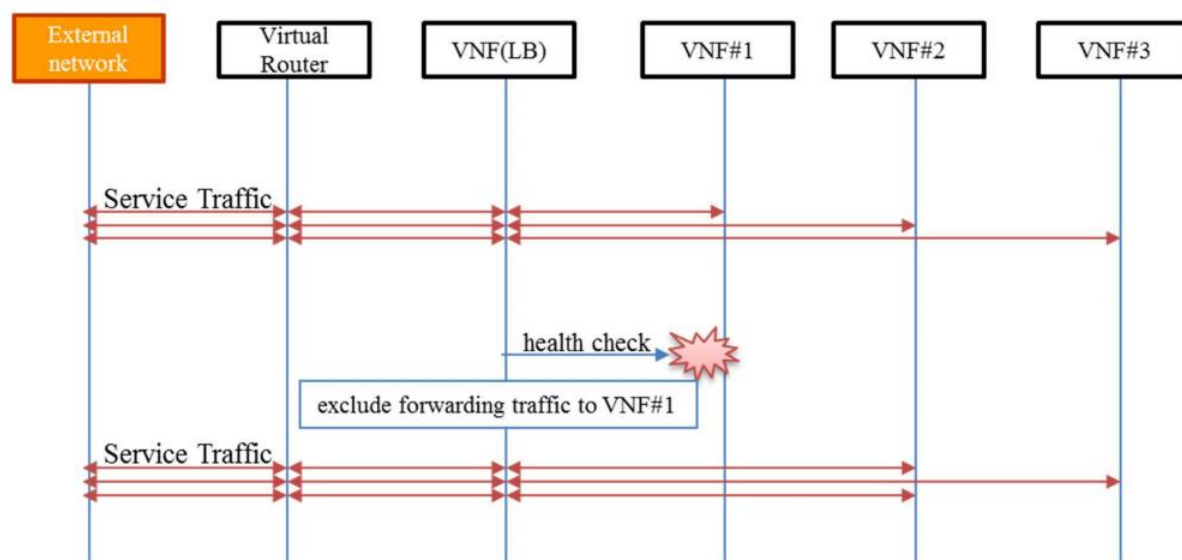
**Figure 14 Load balancing model with internal storage**

Restoration mechanisms with externalized state data have been described in ETSI GS NFV-REL 001. Hence, this clause mainly focuses on the latter case where state information is stored at peer mates. Since shared state information is to be kept consistent within the system, checkpointing or other methods will be used as described in ETSI GS NFV-REL 002.

### **Remediation and recovery phases**

As an assumption, a load balancer periodically checks the operational status of VNFs to which service traffic is delivered to avoid delivering traffic to the failed VNF. When the load balancer detects the failure of a VNF, it stops providing traffic to the failed VNF and delivers it to other VNFs so that service continuity is maintained (Figure 15). When VNFs are stateful, the traffic to the failed VNF has to be delivered to the VNF that stores the state information of the failed service instances as a backup. The standby service instances of the VNF continue the service with the backup information. In some cases, the standby service instances recognize the failure of the corresponding active service instances so as to take over the ongoing services. A possible way to do this is that VNFs check one another, so that the standby service instances within the VNF are aware of the failure of their corresponding active service instances. Since

the external CP that is visible to clients on the external network is only assigned to the virtual load balancer, NFV-MANO does not have to explicitly change the configuration between the load balancer and VNFs for remediation. For recovery, NFV-MANO should deploy a new VNF that compensates the performance and the availability lost due to the failure.



**Figure 15 A sample procedure that will occur when a VNF with the service application fails**

Note that a virtual load balancer should also be redundant, e.g. Active-Standby, to avoid single point of failure. For Active-Standby virtual load balancers, the same requirements for NFV-MANO as those derived from the investigation of the Active-Standby method.

## 1.3. Monitoring Mechanism for HA

### 1.3.1. Fault Management Mechanism

To support High Availability in the NFV environment is more complicated since VNF is composed of hardware level and application level. When fault occurs at NFVI level, VNF also affected by fault of NFVI. For example, when NFVI have some problem such as network, resource sufficient, VNF which belongs to that NFVI also affected by that fault. While, when VNF have some error, if maintenance system doesn't have monitoring algorithm, administrator

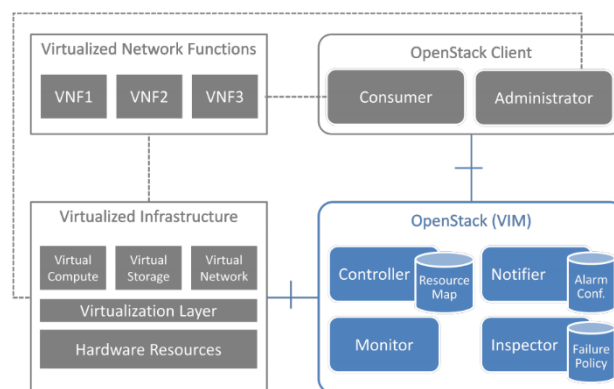
doesn't know the fault at the application level. Therefore, both monitoring algorithms are important to support high availability.

### 1.3.2. Doctor

The goal of Doctor project is to build an NFVI fault management and maintenance framework supporting high availability of the Network Services on top of the virtualized infrastructure. The key feature is immediate notification of unavailability of virtualized resources from VIM, to support failure recovery, or failure avoidance of VNFs running on them. [4]

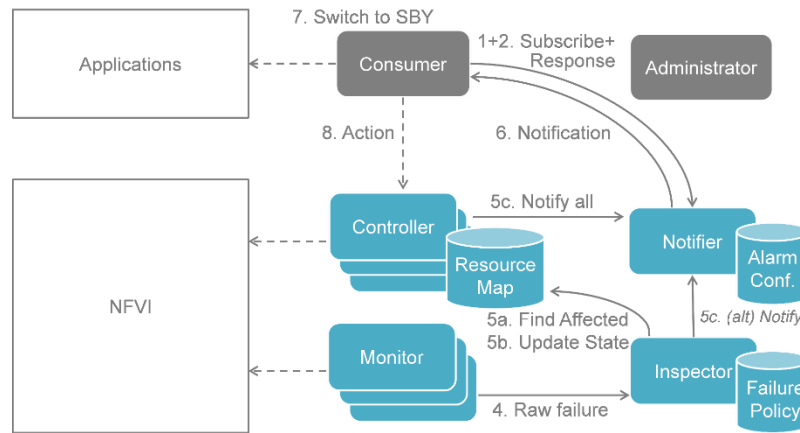
#### 1.3.2.1. Doctor architecture

As shown in Figure 16, the DOCTOR project defined a function block for status management in VIM such as an Inspector for determining defects with received monitoring information, a controller for managing mapping tables for physical resources and virtual resources, and a Notifier for generating events. Respectively. Those function could detect NFVI errors and recover from VNF affected by errors.



**Figure 16 Doctor Project Component**

Like a Figure 17, when a failure occurs, the following procedure will initiate a failover to provide high availability.



**Figure 17 Fault Management Procedure for HA**

First, the notification about the failure is registered in the Notifier. Then the Monitor continuously checks the status to see a failure has occurred. When failure occurs, the Monitor passes the fault information to the Inspector, which in turn determines what resource should be changed. According to result, the information is transmitted to the consumer through the Controller and the Notifier and a failure recovery process is performed to support high availability.

### 1.3.3. Monitoring tool

To support the monitoring framework in the software architecture for the NFV, several solutions have developed such as Zabbix, Nagios, Ceilometer. Nagios is a system and network monitoring project. Zabbix is the most well-known monitoring tool that monitors and tracks numerous kinds of network services, servers, etc. It also could notify the failure quickly since it used agent which is install inside the VNF [5]. In addition, Zabbix also provides APIs in the open source version, and Zabbix has a high performance and ease of development. Ceilometer is a telemetry service which provides resource usage and performance monitoring in the cloud infrastructures [6]. Moreover, it also can further enhance stability in cloud infrastructure environments where VNF is operating.

**Table 1 Comparison of monitoring tools**

<b>Monitoring Tool</b>	<b>Plugins</b>	<b>Triger/Alerts</b>	<b>WebApp</b>	<b>Service Monitoring</b>	<b>API</b>
Zabbix	Yes	Yes	Full Control	Yes	Yes
Nagios	Yes	Yes	Viewing	Yes	Yes Or Via Plugin
Ceilometer	No	Yes	Full Control	No	Yes

As a Table1, Zabbix has an advantage to integrate with other function since it supports plugin and API. Moreover, it also supports GUI to check the status. While, Ceilometer doesn't support plugin and it doesn't support application monitoring mechanism. However, ceilometer could provide monitoring mechanism without agent.

## 2. High availability in Tacker VNF Manager

### 2.1 Problem description

In Telco environment, the reliability of network service (NS) is highly required. Beside auto-healing and auto-scaling solutions, traditional redundancy configuration methods, such as Active-Active or Active-Standby (form a high availability cluster) could be applied to VNFs to ensure the reliability of the NS. These high availability cluster configurations enable fast recovery of VNFs operation in case of failure. Operators will need a way to deploy and manage such kind of high availability clusters via NFVO component. Refer to ETSI-REL [1] for VNF protection schemes in NFV system. it challenges of high availability cluster deployment involve several requirements that need to be handled by the NFV component as described as below:



- \* Resource usage information of each VIM and availability zone for optimal cluster member placement
- \* NFVO should find optimal locations for cluster members (based on predefined policies or dynamic network condition and resource usage status)
- \* Load balancing strategies and monitoring strategies (heartbeat between cluster members or from NFVO) Currently, in Tacker [7], there is no feature to create and manage such kind of high availability clusters. With this proposal, we intend to allow Tacker users to create and manage such kind of high availability clusters [8]. This function enables Tacker to deploy a cluster of Active and Standby VNFs based on predefined operator policies. These high availability policies includes VNF placement locations (i.e. availability zones, VIMs), load balancer configuration. On the other hand, we also define a new **\*\*recovery\*\*** actions which are going to be triggered when failure occurs from cluster members.

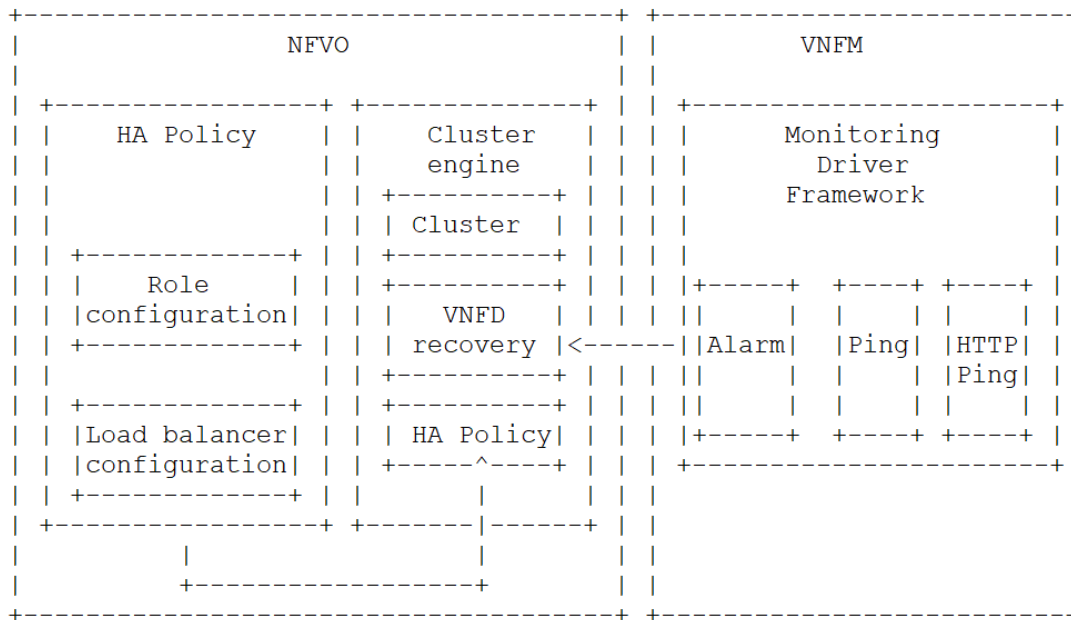


Figure 18 Architecture for HA in Tacker

## 2.2 Architecture

The high level changes needed to Tacker aim at accommodating a new feature in VNF cluster management will include changes to Tacker Client and Tacker Server. The changes include:

\* Adding a cluster REST APIs to Tacker Client which a client can use to CRUD cluster via command line interface. The client can use the REST APIs to deploy VNF cluster from VNFD (from existing VNF will be supported in future). The created cluster should include: (1) The created VNFs that are added to cluster as cluster members, these VNFs must be deployed from the same VNFD, (2) The Neutron load balancer that connects to cluster members aiming at load balancing traffic among cluster members, and (3) the attached deployment policy (Be described in detail as below).

The proposed operations are:

Example of CRUD cluster CLI:

```
.. code-block:: console

    tacker cluster-create

    tacker cluster-show

    tacker cluster-list

    tacker cluster-delete
```

Example of CRUD cluster-member CLI:

```
.. code-block:: console

    tacker cluster-member-add

    tacker cluster-member-show

    tacker cluster-member-list

    tacker cluster-member-delete
```

\* In order to make cluster creation become more flexible, **--policy-file** argument is added in cluster-related command, Tacker Client changes will read the policy-file and pass the deployment configuration to Tacker Server via REST API. The policy should include a role configuration, placement strategies attached members, and the load balancer configuration.

\* Tacker Server will also need to modify the NFVO extensions and NFVO plugin in order to incorporate VNF cluster related operations and resources.

\* The drivers for cluster will need to be written in order to handle both of cluster CRUD operations.

Note that this spec is supposed to only provide cluster management in single site, multi-site scenario will be considered in the future.

Because the policy needs to be defined and attached to a cluster, the policyfile needs to be created first, then it will be passed to cluster creation command as an argument via **--policy-file**. A typical policy template file is shown as below:

```
.. code-block:: yaml

    properties:
      role:
        active:
          VIM0: 1
        standby:
          VIM0: 1
      load_balancer:
        pool:
          protocol: HTTP
```

```
protocol_port: 80

subnet: subnet_mgmt

lb_method: ROUND_ROBIN

vip:

  subnet: subnet_mgmt

  protocol: HTTP

  protocol_port: 80

  target: CP1
```

The policy contains **role** and **load\_balancer** attributes. The **role** attribute indicates which VIM the VNF should deploy on and which member roles (Active or Standby) will be taken by VNFs. Note that deployment of cluster over multi-VIMs is outside of the scope in this spec, but it might be extended to support multi-VIMs cluster deployment later. The **load\_balancer** attribute contains configured parameters that will be used to invoke Neutron Client for deploying a Neutron load balancer. Because the VNF is treated as a cluster member - a basic unit in cluster, in order to deploy the cluster the VNFD template need to be created by the client first. There is no extension of the VNFD compared to the original one. However, in order to archive HA feature for deployed cluster, a new active whose name is **recovery** need to be defined and declared in VNFD as an triggered action in the case of failure. A sample of VNFD with **recovery** action can be seen in the following block:

```
.. code-block:: yaml

  toska_definitions_version: toska_simple_profile_for_nfv_1_0_0

  description: Demo example for VNF cluster

  metadata:

    template_name: sample-tosca-vnfd-cluster
```

```
topology_template:

  node_templates:

    VDU1:

      type: toska.nodes.nfv.VDU.Tacker

      capabilities:

        nfv_compute:

          properties:

            num_cpus: 1

            mem_size: 256 MB

            disk_size: 1 GB

          properties:

            image: cirros-0.3.5-x86_64-disk

            availability_zone: nova

            mgmt_driver: noop

            monitoring_policy:

              name: ping

              parameters:

                monitoring_delay: 45

                count: 3

                interval: 1

                timeout: 2
```

```
actions:
  failure: recovery
config: |
  param0: key1
  param1: key2
CP1:
  type: toska.nodes.nfv.CP.Tacker
  properties:
    management: true
    order: 0
    anti_spoofing_protection: true
  requirements:
    - virtualLink:
        node: VL1
    - virtualBinding:
        node: VDU1
VL1:
  type: toska.nodes.nfv.VL
  properties:
    network_name: net_mgmt
    vendor: Tacker
```

## 2.3 Data Model

Data model impact includes the creation of clusters, clustermembers tables

\* The clusters table will hold all the relevant attributes of cluster resource, while the attributes of the cluster members that belong to the created cluster are stored in the clustermembers table and has an individual cluster\_id attribute which is mapped to the cluster resources.

\* **lb\_info** and **role\_config** in clusters table will store all of the deployment data that are generated during cluster deployment. These attributes will be queried during recovery session include since the Neutron load balancer and cluster nodes information are included.

Table 2: Cluster table

id	string (UUID)	RO, All	generated	N/A	identity
tenant_id	String (UUID)	RW, All	None, (Required)	uuid	tenant ID to launch vnf-cluster
name	string	RW, All	None, (Required)	string	human-readable name
description	string	RW, All	None	string	description of cluster
status	string	RO, All	generated	string	Status of created cluster
vnfd_id	string (UUID)	RO, All	None (Required)	uuid	VNFD ID to use to deploy cluster members
lb_info	string	RO, All	None, (Required)	string	attributes of created load balancer
role_config	string	RO, All	None, (Required)	string	role configuration of cluster members

Table 3: Cluster member table

id	string (UUID)	RO, All	generated	N/A	identity
tenant_id	String (UUID)	RW, All	None, (Required)	uuid	tenant ID to launch cluster member
name	string	RW, All	None, (Required)	string	human-readable name
cluster_id	string (UUID)	RO, All	generated	uuid	ID of associated cluster
vnf_id	string (UUID)	RO, All	generated	uuid	ID of associated vnf
mgmt_url	string	RO, All	generated	string	Management URL of associated vnf
role	string	RW, All	generated	String	Role of member in cluster
lb_member_id	string (UUID)	RO, All	generated	uuid	ID of associated Neutron load balancer
placement_attr	string	RW, All	None (Required)	string	VIM name where cluster members are deployed



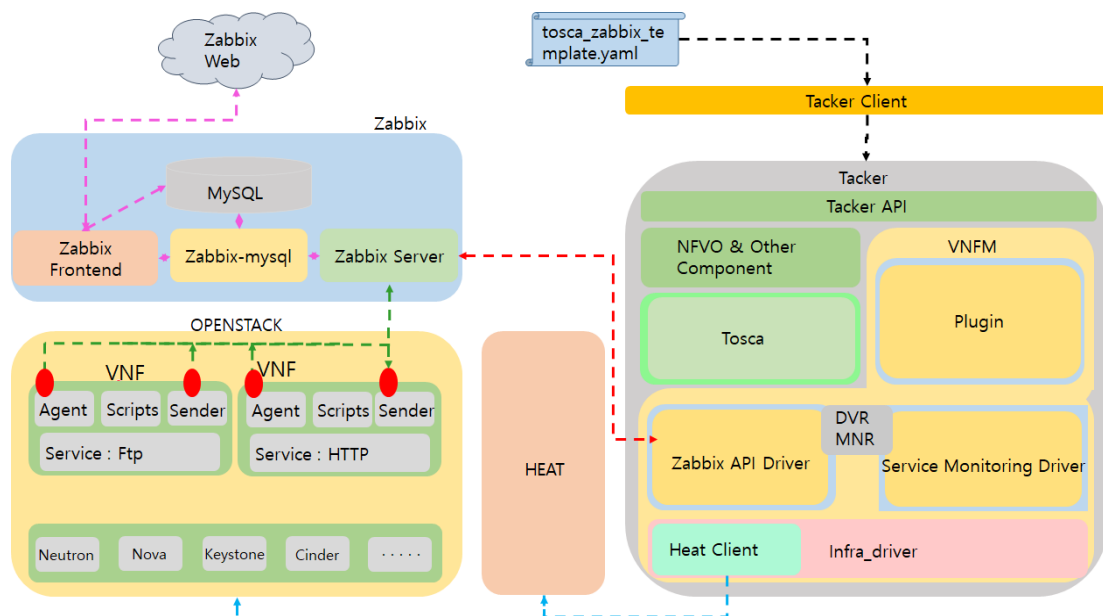


### Figure 21 Recover completed

### 3. Monitoring tool integration in Tacker VNF Manager

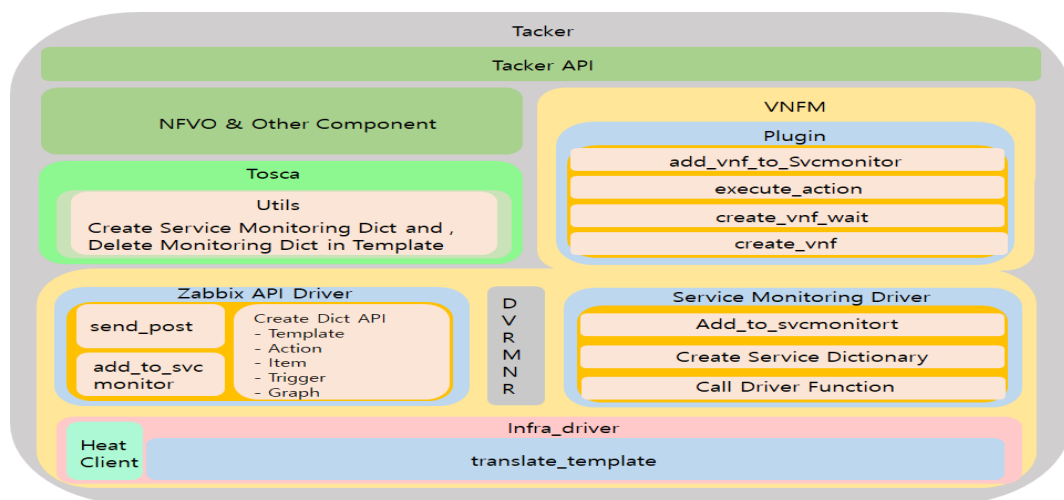
#### 3.1. Design and implementation of monitoring tool interworking structure

In order to provide high availability in NFV environment, it is necessary not only to monitor resource usage of VNF and VIM provided by existing MANO but also to monitor services inside VNF through interworking between VNFM constituting MANO and open source monitoring tool do. Open source monitoring tools need to work with VNFM for service management inside VNF. To do this, you must configure a driver for API interoperation between the open source monitoring tool and the VNFM, and make the Rest API request to open source monitoring. Each VNF created by VNFM must have an agent provided by an open source monitoring tool. This is because open source monitoring tools have difficulty to access VNF internally. Therefore, it collects status data about services through agents installed inside VNF and identifies the statuses of services. Open source monitoring tools should be integrated with VNFM by selecting an open source monitoring tool suitable for the infrastructure.



**Figure 22 Interworking between and Open source monitoring tool in NFV environment**

Figure 22 shows the overall design of open source monitoring tools and VNFM in a cloud NFV environment. Tacker provides VNFM, Zabbix is a monitoring server, and Openstack provides cloud infrastructure. Based on the information defined in the Tacker Template, the Zabbix Api Driver asks the Zabbix server for the API to generate templates, triggers, actions, and items. We also implemented a service monitoring driver to add plugins for additional open source monitoring tools.



**Figure 23 VNFM monitoring driver design structure**

Figure 23 shows the driver structure for interworking between VNFM and Monitoring Tool. Send\_post is a component for requesting an API to the Zabbix server. All driver functionality is requested from the server via Send\_post. add\_to\_svc\_monitor is a component for the service monitoring driver to call the Zabbix API driver. Dict API creation creates and requests dictionaries for each request in the Zabbix API format. Service monitoring driver is a driver that manages all open source monitoring tools and invokes the corresponding monitoring driver according to the monitoring name defined in the template. Service dictionary creation extracts only the service monitoring policy from the delivered template information, creates it in advance, and passes it through the Zabbix API. The plug-in determines if the template has a service monitoring policy and performs certain functions defined in the template.

This article uses the project shown in Figure 24 to monitor the services inside VNF. OpenStack is used by VIM to monitor services inside VNF. Openstack Tacker is used as VNFM and Zabbix is easy to develop, implemented and tested with monitoring tools. In a physical environment,

three mini-PCs were used to configure the controller, calculation, and monitor nodes, respectively.

VNF Manager : Openstack Tacker(Master ver.)  
 VIM : Openstack (Ocata ver.)  
 Physical Resource(NFVI) : Controll,Compute,Monitor Node  
 Monitoring Tool : Zabbix (3.0 ver.)

**Figure 24 NFV environment configuration component**

In this implementation, the policy for service monitoring is defined in the TOSCA Template and the user specifies the condition value of the desired trigger.

```
app_monitoring_policy:
  name: zabbix
  zabbix_username: Admin
  zabbix_password: zabbix
  zabbix_server_ip: 192.168.11.53
  zabbix_server_port: 80
  parameters:
    application:
      app_name: apache2
      app_port: 80
      ssh_username: ubuntu
      ssh_password: ubuntu
      app_status:
        condition: [down]
        actionname: cmd
        cmd-action: sudo service apache2 restart
      app_memory:
        condition: [greater,22]
        actionname: cmd
        cmd-action: sudo service apache2 stop
    OS:
      os_agent_info:
        condition: [down]
        actionname: cmd
        cmd-action: sudo service zabbix-agent restart
      os_proc_value:
        condition: [and less,22]
        actionname: cmd
        cmd-action: sudo reboot
      os_cpu_load:
        condition: [and greater,30]
        actionname: cmd
        cmd-action: sudo reboot
      os_cpu_usage:
        condition: [less,30]
        actionname: cmd
        cmd-action: sudo reboot
```

**Figure 25 Zabbix VNFD Tosca Template**

Figure 25 shows the Tosca template with the VNFD used by Tacker to create the VNF. Users

can configure various settings related to application monitoring in their VNFDs. Policies are largely divided into applications and operating systems. For an application, you can specify the type, state, memory, and recovery method of the application. Similarly, for the OS, you can set up multiple policies to monitor the inside of the VNF. When the Tacker generates the VNF, the monitoring driver sends the information to Zabbix.

In addition, scripts for automatic agent installation and configuration are created in the User\_data parameter after the VNF is created, and in the initialization step after the VNF is created. Figure 26 shows the script created with the User\_data parameter used in this document. Use the apt-get command to install the Zabbix agent and Apache for service monitoring. The sed statement finds the string in the file and replaces it with the desired value used to configure the zabbix -agentd.conf file and network.

```

user_data |
#!/bin/bash
apt-get -y install zabbix-agent
apt-get -y install apache2
sudo sed -i "2s/.*ifconfig ens3 | grep '.*W.*inet addr:W.*'| cut -d: -f2 | awk
'W'{ print $1 }W'"/g" "/etc/hosts"
sudo sed -i "s/Bcast/cast/etc/hostname/g" "/etc/hosts"
sudo sed -i "3s/.*192.168.11.53Wtmonitor/g" "/etc/hosts"
sudo /etc/initd/networking restart
sudo echo 'zabbix ALL=NOPASSWD: ALL' >> /etc/sudoers
sed -i "s/# EnableRemoteCommands=0/EnableRemoteCommands=1/"
"/etc/zabbix/zabbix_agentd.conf"
sed -i "s/Server=127.0.0.1/Server=192.168.11.53/" "/etc/zabbix/zabbix_agentd.conf"
sed -i "s/ServerActive=127.0.0.1/ServerActive=192.168.11.53:10051/"
"/etc/zabbix/zabbix_agentd.conf"
sed -i "s/Hostname=Zabbix server/Hostname=`cat /etc/hostname`/"
"/etc/zabbix/zabbix_agentd.conf"
  
```

**1. Install Zabbix-agent & apache2**

**2. Network Settings**

- Set hostname
- host ip configuration
- setting up monitoring node ip

**3. Zabbix user permission setting**

**# Setting up the zabbix-agent.conf file**

- Allow remote command
- Zabbix server ip address setting
- Zabbix active ip address setting
- Set hostname

**Figure 26 Open source monitoring tool in VNF Agent Automatically install script**

## 3.2. Verification and analysis

Tacker writes to the Zabbix driver in a pre-form for the API request, converts it to the Json format, and then sends it to the Zabbix server via the Post function. The Zabbix server receives the requested information and performs operations according to the API. Figure 27 shows the API request and response.

```
# API Request

{"params":{"templates":[{"templateid":"10641"}],"host":"vdu2","interfaces":[{"ip":
"192.168.11.185","useip":1,"dns":"","main":1,"type":1,"port":"10050"},"groups":
[{"groupid":"4"}]},"jsonrpc":"2.0","method":"host.create","auth":
"283be614672bda4d329c3891be3c2670","id":4}

# API Response

(' [ Response from Zabbix Server ] : ', {u'jsonrpc': u'2.0', u'result': {u'hostids': [u'10642']}, u'id': 4})
```

**Figure 27 API Request & Response**

Tacker asks the Zabbix server to register the VNF information through an API that contains information about the VNF. The Zabbix server starts monitoring and operates according to each procedure. The VNFD to which the service monitoring policy applies is loaded from the database when the VNF is created.

```
[ 56.010162] cloud-init[1252]: The following NEW packages will be installed:
[ 56.010406] cloud-init[1252]: zabbix-agent
[ 57.015316] cloud-init[1252]: 0 upgraded, 1 newly installed, 0 to remove and 12 not upgraded.
[ 57.016550] cloud-init[1252]: Need to get 184 kB of archives.
[ 57.017581] cloud-init[1252]: After this operation, 798kB of additional disk space will be used.
[ 57.018557] cloud-init[1252]: Get:1 http://nova.clouds.archive.ubuntu.com/ubuntu/xenial-updates/universe amd64 zabbix-agent amd64 1:2.4.7+dfsg-2ubuntu2.1 [184kB]
```

**Figure 28 When the VNF generation is initialized, the user-data script log**

Figure 28 shows a log of the execution of 'apt-get install zabbix-agent' in the user\_data script of the template for which the service monitoring policy is defined, and executes the sed script to configure the Zabbix agent's conf file as the next operation.

Name	Application	Item	Trigger	Graph	Interface	Template	Status	ZBXStatus
YDU1	Application	7	6	4	192.168.11.189:10050	Template Tacker YDU1	Active	Zbx On
YDU2	Application	7	6	4	192.168.11.189:10050	Template Tacker YDU2	Active	Zbx On

**Figure 29 Monitoring Information for each VDU**

Figure 29 shows two VNFs registered on the Zabbix server. After creating the VNF, Tacker invokes the Zabbix Api driver to request the creation of item, graph, and trigger actions that contain information about the data collection. Then, in order to register the monitoring VNF as a host, the VNF Host Create API is requested by transferring the template ID generated by the Zabbix Api driver in advance. When Zabbix server and Zabbix agent are connected

normally, Zbx lights up in green.

Having a separate template for each VNF is to be able to monitor with different conditions and items. Figure 30 shows that the user created a trigger based on the condition values specified in the template.

Severity	Name ▲	Expression	Status
Average	Template Tacker VDU1: Disk I/O is overloaded on {HOST.NAME}	{vdu1:system.cpu.util[,iowait].avg(1m)}>20	Enabled
Average	Template Tacker VDU1: Process Memory is lacking on {HOST.NAME}	{vdu1:proc.mem[apache2,root].avg(5)}>20	Enabled
Average	Template Tacker VDU1: Processor load is too high on {HOST.NAME}	{vdu1:system.cpu.load[percpu,avg1].avg(1m)}>5	Enabled
Average	Template Tacker VDU1: service is down on {HOST.NAME}	{vdu1:net.tcp.service[http].max(#3)}=0	Enabled
Average	Template Tacker VDU1: Too many processes running on {HOST.NAME}	{vdu1:proc.num[,run].avg(5s)}>2	Enabled
Average	Template Tacker VDU1: Zabbix agent on {HOST.NAME} is unreachable for 5 minutes	{vdu1:agent.ping.nodata(5m)}=1	Enabled

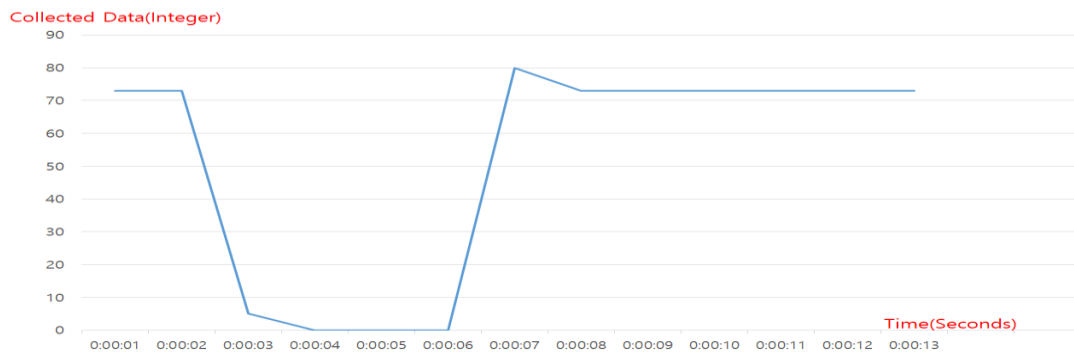
**Figure 30 Trigger list based on condition value**

Figure 31 shows the occurrence of a fault according to the data collected through the agent installed in the VNF according to the specified trigger condition. The time of the fault, the type of fault, and the location of the fault. If a failover occurs, the state is marked as resolved.

Severity	Status	Info	Time ▼	Age	Ack	Host	Name	Description
Average	PROBLEM		10/13/2017 11:28:42 AM	13s	No 4604	vdu2	Too many processes running on vdu2	Add
Average	PROBLEM		10/13/2017 11:28:37 AM	18s	No 9	vdu1	Process Memory is lacking on vdu1	Add
Average	OK		10/13/2017 11:28:37 AM	18s	No 8	vdu1	service is down on vdu1	Add
Average	PROBLEM		10/13/2017 11:28:30 AM	25s	No 4254	vdu1	Too many processes running on vdu1	Add
Average	PROBLEM		10/10/2017 10:29:12 PM	2d 12h 59m	No 1	vdu2	service is down on vdu2	Add

**Figure 31 Trigger Alarm Check for Fault Occurrence**

Figure 32 is a graphical representation of the changes in data collected for services running through agents in the VNF. The Y axis represents the number of data to be collected, and the X axis represents the time. The data collected from the service through the agent at 1-second intervals appears constant, but at the point where the service is stopped, the graph drops sharply and 0 is displayed. After an interval of about 4 seconds, the service is restored to failure and can be seen to show a rising graph.



**Figure 32 Data collection changes due to outages**

When the operation of the service is stopped, the Zabbix server does not have collected data, so a trigger occurs and an action command for the trigger is executed through the agent. The action generated according to the request is shown in Figure 33.

Steps	<input type="text" value="1"/> - <input type="text" value="2"/> (0 - infinitely)							
Step duration	<input type="text" value="0"/> (minimum 60 seconds, 0 - use action default)							
Operation type	<input type="button" value="Remote command"/>							
Target list	<table border="1"> <thead> <tr> <th>Target</th> <th>Action</th> </tr> </thead> <tbody> <tr> <td>Host: vdu1</td> <td><a href="#">Remove</a></td> </tr> <tr> <td><a href="#">New</a></td> <td></td> </tr> </tbody> </table>		Target	Action	Host: vdu1	<a href="#">Remove</a>	<a href="#">New</a>	
Target	Action							
Host: vdu1	<a href="#">Remove</a>							
<a href="#">New</a>								
Type	<input type="button" value="Custom script"/>							
Execute on	<input checked="" type="button" value="Zabbix agent"/> <input type="button" value="Zabbix server"/>							
Commands	<input type="text" value="service apache2 restart"/>							

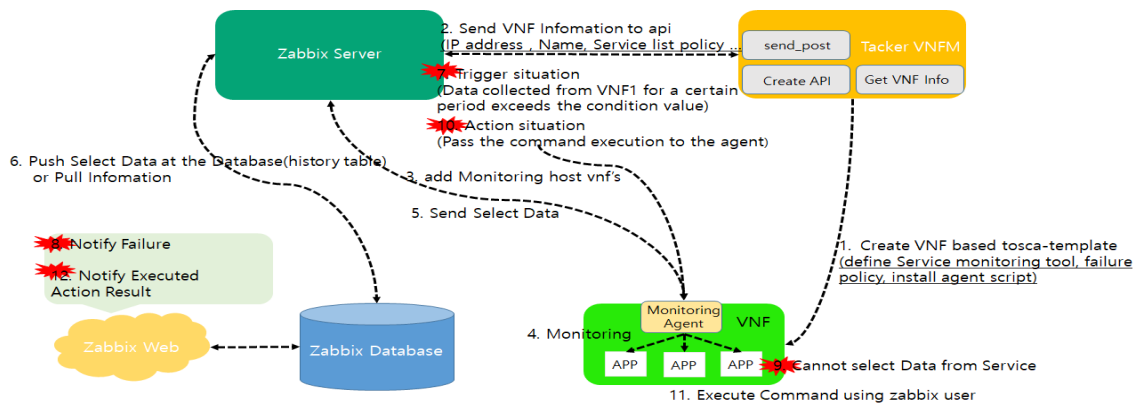
**Figure 33 Custom actions based on triggers**

The Zabbix agent receives the command to execute the command from the server and executes the corresponding command using the Zabbix user generated in advance. These actions are generated for each service and VNF.

In order to check whether the service is actually recovered according to the occurrence of the service failure, the service should be stopped in the VNF. After that, you can stop the service and check the fault detection on Zabbix web. In case of error, red message is displayed when solving red notification. Recovery time after failure occurrence is about 5 seconds, and according to action script execution speed, recovery according to action script execution is performed at the same time when a failure occurs. As a result, the notification of the failure occurs as if the recovery has already been completed. According to the development test, the tacker creates the VNF and registers it as the monitoring host of the Zabbix server. In addition, template, action, item, graph, and trigger were generated through API request and confirmed



the operation. In addition, we confirmed the API request and its response. Figure 34 shows the workflow of the above-mentioned series of processes, from generation of VNF to monitoring, trigger, and action generation.



**Figure 34 Test Workflow**

Through this series of processes, Tacker can operate stable VNF using Zabbix. Zabbix's performance is one of the best monitoring tools for monitoring applications. Common monitoring tools must support the API in order to be able to easily add additional monitoring tools to the plug-in format.

### 3.3. Conclusion

In this document, we have designed a structure for monitoring services in VNF in conjunction with VNFM and open source monitoring tools, and implemented and tested performance and multi-time measurement experiments. This is important because the stability of the VNFM and cloud infrastructures that provide VNF incrementally is an important technology, so it is important to manage the inside of the VNF to ensure stability. However, additional research is needed to provide additional services based on monitoring results. In future research, we plan to study the high availability structure that can cope with various obstacles based on the currently developed service monitoring.

## 4. Root Cause Analysis for High Availability

In addition to the above-mentioned monitoring and restoration techniques, techniques for analyzing the cause of failures are also important functions for providing high availability. The Vitrage project is a project that analyzes the root cause of a failure and can increase the availability by reducing the probability of additional disruptions. This chapter introduces the proposed structure to improve user accessibility and fault detection, and presents the integrated structure in the next chapter.

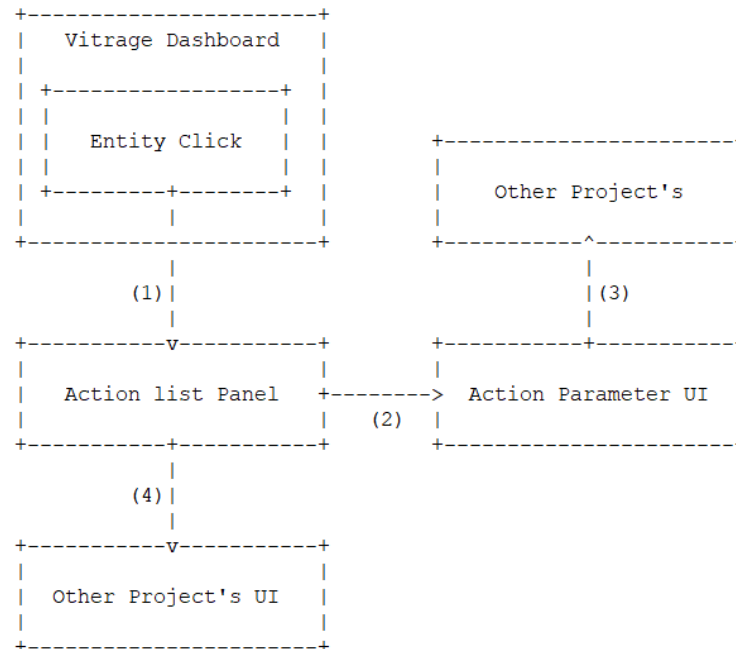
### 4.1. Action List for Root Cause

For each entity in the current entity graph, the user can see information about the entity through a click action. We will add an action list panel that provides multiple actions to the user through existing click actions. Users can click an entity to see a list of available actions from the drop-down menu. The drop-down menu is located at the bottom of the existing Info panel and configures the action list based on a setting file. The user can select one of these action lists and enter the specific parameters required to execute the action through the additional UI.

Examples of actions that can be provided:

- \* Performance Test : The user can perform tests by clicking entities such as Nova, Neutron, and others displayed in the Entity Graph. Like Mistral, the user can view a list of test scenarios in a new UI, enter various parameter values for testing, and then request performance testing through the Openstack Rally API. The range of performance support for this feature will be expanded in the future.
- \* Launch web page of monitoring tool : The web of the monitoring tool monitoring the VM or the physical node entity is displayed.

The overall workflow is as follows::



(1) Users can click an entity to see a list of actions that can be performed through the drop-down at the bottom of the Info panel. The action list is composed of the settings file configured by the user. This means that the list of actions depends on user's environment. Also, depending on the entity type, the action may be restricted.

(2) When the user selects an action from the action list, a detailed action list(e.g. Workflow list, Test scenario list) and a UI for inputting parameters are displayed. The user can then select detailed actions and enter parameters in the corresponding UI.

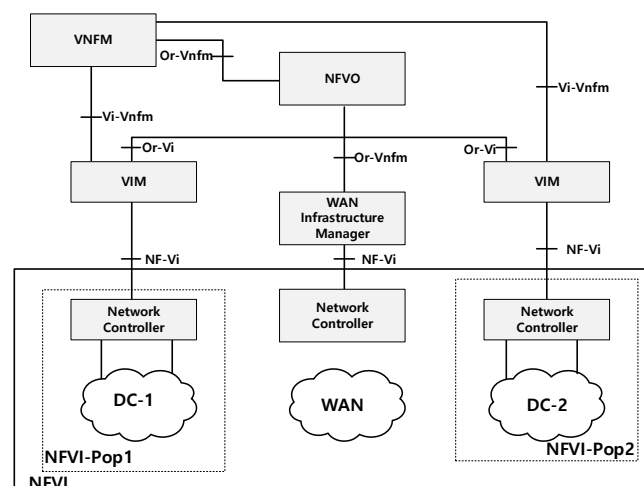
(3) Enter all the parameters in the parameter UI and press the OK button to request it from the other project.(e.g. Mistral, Rally)

(4) The result of the user's request can be viewed in a new tab with the related project's UI. An example configuration file is shown below:

## 5. Multisite NFV Management System

### 5.1. Introduction

The European Telecommunications Standards Institute (ETSI) has defined the basic NFV architectural framework with three entities: NFV infrastructure (NFVI), VNFs and Management and Orchestration (MANO) [7]. The MANO itself composed of NFV Orchestrator (NFVO), VNF Manager (VNFM) and Virtualization Infrastructure Manager (VIM) to manage virtual infrastructures, VNFs and provide the end-to-end network services. With multisite VIM support, users can choose to ~~deploying~~ VNFs on optimal placement in multiple VIMs [8].

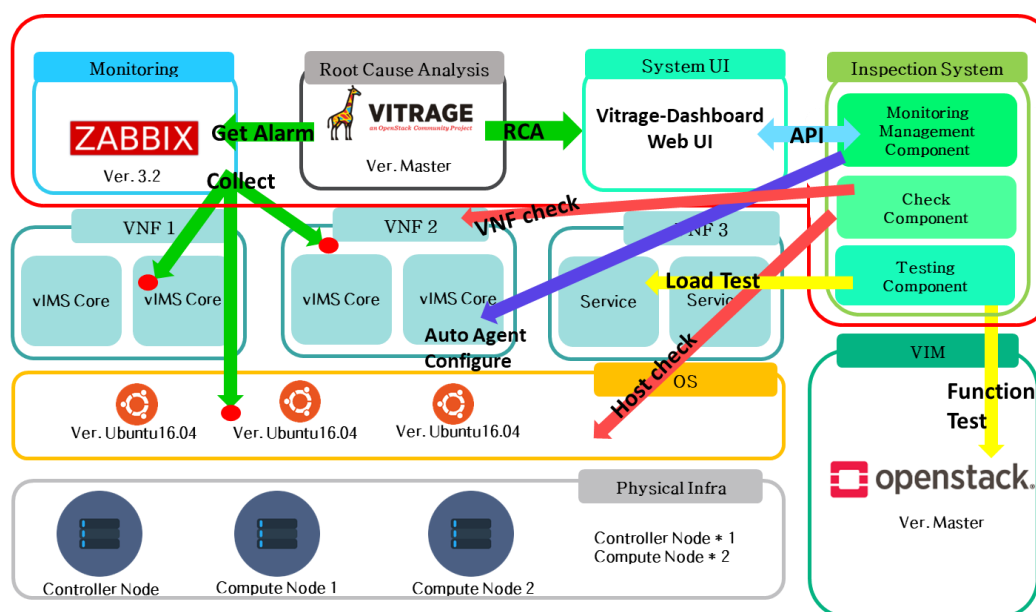


**Figure 35. ETSI NFV Architecture with multiple-VIM**

As shown in Fig. 35, the NFVO manages multiple VIMs, and network management by using network controller such as SDN controller that's required between sites. In multiple VIMs, each site may have the same virtualization technology, but in some scenarios, each site can use different virtualization technologies such as Container or VM. When VIMs have the same virtualization technology, multiple VIM sites can be managed by a single MANO. On the other hands, when multiple VIMs have different virtualization technologies, orchestrating them requires ~~need~~ multiple domain MANOs for each type of site or an extended functionality to MANO to support all sites. In other words, in a multi-site environment, an architecture that can manage a whole site from a single site is needed.

## 5.2. High availability for NFV multi-site environment

In order to provide high availability for site operation in an NFV multi-site environment, we will integrate the separately developed failure probe, system diagnostics, benchmarking and monitoring functions into one architecture, and provide high-level, high-availability functions through interaction between them.

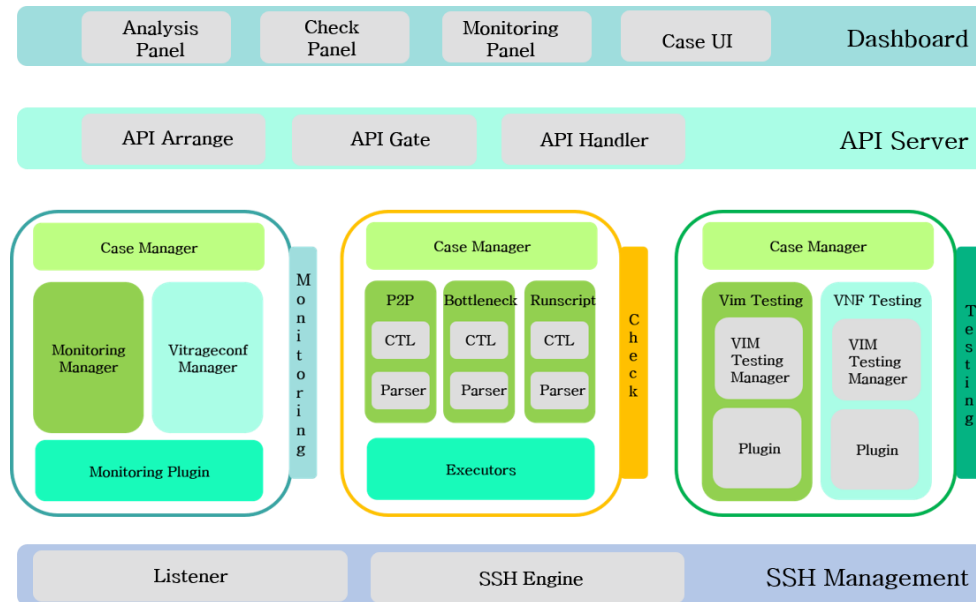


**Figure 36. NFV multi-site environment management system architecture**

In order to provide high availability in a multi-site environment, the following functions must be provided.

- Consolidate and deliver individually developed high availability features into one
- Full view of infrastructure, resources, and VNF in a multisite environment
- Fault Cause Analysis System based multi-angle management function
- Integration and automation of system inspection, benchmarking and monitoring functions
- Integrated system UI for efficient management of administrator's network

### 5.2.1. Detailed Architecture



**Figure 37. Detailed Architecture of NFV multi-site environment management**

- System based UI (System on the Vitrage-dashboard): The system UI provides the integrated UI to request the individual components to provide the function for component use and confirm the result. This allows administrators to view the entire infrastructure as a single view, and quickly identify disabilities to speed up disability and increase subscribers' satisfaction.
- Inspection System: Provides benchmarking, infrastructure view, monitoring, fault analysis and system check function in multi-site environment as a core component of this system. Providing real function through the component, it is possible to reliably perform the request of the administrator. These functions provide a multi-dimensional management technique, enabling low-cost, high-efficiency functions.

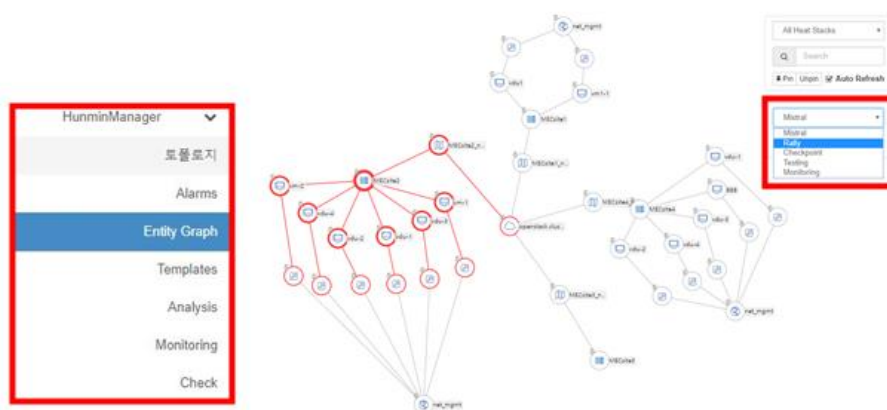
### 5.2.2. Implementation

This architecture is implemented in an OpenStack environment, and the overall configuration is as follows.

**Table 2. Specifications of Implementation**

<i>Entity</i>	<i>Condition</i>	<i>Version</i>
Physical Server(4)	<ul style="list-style-type: none"> <li>- Intel® Xeon® processor D-1520, Single-socket FCBGA 1667; 4-core, 8 threads, 45 W</li> <li>- RAM: 16 GB</li> <li>- Disk space: 256 GB</li> <li>- Controller Node (1) / Compute Node (3)</li> </ul>	
Cloud OS	OpenStack stable	Queens
VNF Manager	OpenStack Tacker	Master
RCA	OpenStack Vitrage	Master
Monitoring	Zabbix	

As shown below, User can check the multi-site environment through the graph topology and Information about each site can be found in the developed entity graph panel. Clicking on the Entity Graph panel allows user to check the entire physical infrastructure, virtualization, virtual machines and faults that make up the Multi-Site. When each entity is clicked, the action list containing the functions provided by the management system is operated on the right side, and the desired functions can be selectively used in the corresponding list.

**Figure 38. Multi-Site topology GUI**

**Figure 39. The example of Management System UI**

```
[00] Run test import case manager
[00] case_manager.CaseManager.start(case='vmf', tool='artillery', hosts='192.168.11.238', auth=['ubuntu', 'ubuntu'], w
f_testing_args=dict(count=3, "name": 10, "target_ip": 'http://192.168.11.238')
Plugin artillery : can't artillery quick --num 10 - count 3 http://192.168.11.238
[00] 192.168.11.238 Executing task 'run_ssh'
[00] 192.168.11.238 run_artillery quick --num 10 - count 3 http://192.168.11.238
[00] 192.168.11.238 Started phase 0, duration: 15 0:11:03:58(+0000) 2018-08-14
[00] 192.168.11.238 T T I D A L I F T T I D A L I Report @ 11:04:00(+0000) 2018-08-14
[00] 192.168.11.238 Scenarios launched: 3
[00] 192.168.11.238 Scenarios completed: 3
[00] 192.168.11.238 Requests completed: 30
[00] 192.168.11.238 RPS sent: 28.85
[00] 192.168.11.238 Request latency:
[00] 192.168.11.238 min: 0.7
[00] 192.168.11.238 max: 10.4
[00] 192.168.11.238 median: 0.9
[00] 192.168.11.238 p50: 2.3
[00] 192.168.11.238 p99: 10.4
[00] 192.168.11.238 Codes:
[00] 192.168.11.238 200: 30
[00] 192.168.11.238
[00] 192.168.11.238 All virtual users finished
[00] 192.168.11.238 Summary report @ 11:04:00(+0000) 2018-08-14
[00] 192.168.11.238 Scenarios launched: 3
[00] 192.168.11.238 Scenarios completed: 3
[00] 192.168.11.238 Requests completed: 30
[00] 192.168.11.238 RPS sent: 28.57
[00] 192.168.11.238 Request latency:
[00] 192.168.11.238 min: 0.7
[00] 192.168.11.238 max: 10.4
[00] 192.168.11.238 median: 0.9
[00] 192.168.11.238 p50: 2.3
[00] 192.168.11.238 p99: 10.4
[00] 192.168.11.238 Scenario counts:
[00] 192.168.11.238 0: 3 (100%)
[00] 192.168.11.238 Codes:
[00] 192.168.11.238 200: 30
[00] 192.168.11.238
```

48



```

stack@hwijinHong:~/Inspection-System/check$ python3 case_manager.py
Start Execute
=====
[192.168.11.61] Executing task 'run_ssh'
[192.168.11.61] run: ping -c 3 -q 192.168.11.61
[192.168.11.61] out: PING 192.168.11.61 (192.168.11.61) 56(84) bytes of data.
[192.168.11.61] out:
[192.168.11.61] out: --- 192.168.11.61 ping statistics ---
[192.168.11.61] out: 3 packets transmitted, 3 received, 0% packet loss, time 199
8ms
[192.168.11.61] out: rtt min/avg/max/mdev = 0.061/0.066/0.075/0.006 ms
[192.168.11.61] out:
Done.
=====

```

**Figure 41. Checking Component Function result**

Figure 40 and Figure 41 show the operation results of Check component and Testing component provided by the system. The results for the Check and Testing actions are available in the Results panel.

```

sudo apt-get -y update
sudo apt-get -y upgrade
sudo apt-get -y install rabbit-agent
sudo apt-get -y install apache

sudo sed -i "s#/#/#ifconfig /opt/stack/ | grep 'inet addr:' | cut -d: -f2 | awk '{ print $1 }'/" /opt/stack/Inspection-System/testhosts"
sudo sed -i "s#/#cat /etc/hostname /g" /opt/stack/Inspection-System/testhosts"
sudo sed -i "s#/#192.168.11.233/ monitor/g" /opt/stack/Inspection-System/testhosts"
sudo /etc/init.d/networking restart
sudo echo "rabbit rabbit@192.168.11.233" >> /etc/sudoers

sudo sed -i "s#/#EnableHostCommand=0/EnableHostCommand=1/" /etc/rabbit/rabbit_agentd.conf"
sudo sed -i "s#/#Server=127.0.0.1/Server=192.168.11.61/" /etc/rabbit/rabbit_agentd.conf"
sudo sed -i "s#/#ServerActive=127.0.0.1/ServerActive=192.168.11.61:80/" /etc/rabbit/rabbit_agentd.conf"
sudo sed -i "s#/#Hostname=rabbit server/Hostname= cat /etc/hostname/" /etc/rabbit/rabbit_agentd.conf"

sudo service apache restart
sudo service rabbit-agent restart
sudo echo "mount@ubuntu" | chpasswd
sudo echo "root@root" | chpasswd

```

**Figure 42. Automation Monitoring script**

Figure 43 show the automation script for monitoring. With this, this system can be provided monitoring function for physical environment and virtual environment automatically.

## 5.2.3. Use case 1- High Availability for IoT Cloud

### 5.2.3.1. Introduction

In IoT cloud, some availability mechanisms have been proposed, but they provided mechanism for specific data such as collected data, or it is not considered characteristic for IoT Cloud [13] [14]. IoT cloud environment should consider various environments compared to other cloud since they have many kinds of IoT device from sensor to mobile, and each application for IoT device should provide for each IoT devices. The services provided in the IoT cloud are smart home, smart city, healthcare, etc., and each service has different requirements and procedures for providing availability according to service characteristics. Therefore, the availability model should be provided considering the following characteristic. First, fault detection and recovery for services provided in the IoT cloud environment may be adopted according to the service characteristics and the priorities of the targets to be monitored. For example, one of the key features of smart metering services is that it collects and visualization data which is received from various IoT devices [15]. Smart metering service is not required real-time response and processing so much, so it is allowed to more time than service which required real-time response for detection and recovery time. In addition, there is not much data processing and database failures can be more critical than applications failures. Therefore, monitoring of the database is more important for availability, and failover models may be more suitable than redundancy models. On the other hand, v2v service which is used for communication between vehicles is required real-time response for real-time response and fast data processing. Therefore, it is need recovery mechanism and application monitoring for fast detection and recovery rather than smart metering service. In the case of v2v application, redundancy model is more suitable rather than other recovery mechanism since it is faster than other mechanisms. In addition, v2v applications are highly dependent on applications because they must process and respond quickly to data. Therefore, detection for an application can help to detect faults quickly [16]. The second, the recovery mechanism can be adopted according to the cloud environment. For example, in the case of a healthcare service or a CCTV service that requires real-time data processing, a redundancy model is suitable to recover it quickly. However, when cloud is resource-constrained environment such as edge cloud, resource may not be sufficient for configuring redundancy model. In this case,

failover mechanism is not faster than redundancy model, but failover mechanism should be used for availability.

## References

- [1] ETSI GS NFV-REL 003 v1.1.1, Report on models and features for end-to-end reliability
- [2] ETSI GS NFV-REL 001 v1.1.1, NFV Resiliency requirements
- [3] ETSI GS NFV-REL 002 v1.1.1, NFV, Report on Scalable for reliability management
- [4] OPNFV Doctor Project,  
<https://wiki.opnfv.org/display/doctor/Doctor+Home>(Accessed 02.03.18).
- [5] Zabbix, <http://www.zabbix.com/> (Accessed 02.03.18).
- [6] Openstack Ceilometer, <https://docs.openstack.org/ceilometer/latest/>(Accessed 02.03.18).
- [7] Openstack Tacker project, <https://wiki.openstack.org/wiki/Tacker>
- [8] Tacker HA deployment specification <https://review.openstack.org/#/c/502216/>
- [9] Networking-SFC, <https://docs.openstack.org/newton/networking-guide/config-sfc.html>
- [10] Tacker VNFFG, <https://github.com/openstack/tacker-specs/blob/master/specs/newton/tacker-vnffg.rst>

## ***K-ONE Technical Document***

- It is not prohibited to modify or distribute this documents without a permission of K-ONE Consortium.
- The technical contents of this document can be changed without a notification due to the progress of the project.
- Refer website below for getting more information of this document.

(Homepage: <http://opennetworking.kr/projects/k-one-collaboration-project/wiki>, E-mail: [k1@opennetworking.kr](mailto:k1@opennetworking.kr))

Written by K-ONE Consortium

Written in 20167/05