|  |  |
| --- | --- |
| Title: | Doctor: Fault Management and Maintenance |
|  |  |
| **OPNFV** project: | Doctor <https://wiki.opnfv.org/doctor> |
|  |  |
| Editors: | Ashiq Khan (NTT DOCOMO, [khan@nttdocomo.com](mailto:khan@nttdocomo.com)),  Gerald Kunzmann (NTT DOCOMO, [kunzmann@docomolab-euro.com](mailto:kunzmann@docomolab-euro.com)) |
| Authors: | Ryota Mibu (NEC, [r-mibu@cq.jp.nec.com](mailto:r-mibu@cq.jp.nec.com)) |
|  | Carlos Goncalves (NEC, [Carlos.Goncalves@neclab.eu](mailto:Carlos.Goncalves@neclab.eu)) |
|  | Tomi Juvonen (Nokia, [tomi.juvonen@nsn.com](mailto:tomi.juvonen@nsn.com)) |
|  | … |
|  |  |
|  |  |
| Project creation: date: | 2014-12-02 |
| Submission date: | 2015-03-XX |
|  |  |
|  | |

**ABSTRACT:** “Doctor” is an OPNFV requirement project. Its scope is NFVI fault management and maintenance and it aims at developing and realizing the consequent implementation for the OPNFV reference platform.

This deliverable is …

**Definition of terms:**

Different SDOs and communities use different terminology related to NFV / Cloud / SDN. This list tries to define an OPNFV terminology, mapping/translating the OPNFV terms to terminology used in other contexts.

* **NFVI**: Virtualization Infrastructure such as HV; totality of all hardware and software components which build up the environment in which VNFs are deployed
* **VIM**: Virtualised Infrastructure Manager. functional block that is responsible for controlling and managing the NFVI compute, storage and network resources, usually within one operator's Infrastructure Domain (e.g. NFVI-PoP)
* **NFVO**: Network Functions Virtualisation Orchestrator. Functional block that manages the Network Service (NS) lifecycle and coordinates the management of NS lifecycle, VNF lifecycle (supported by the VNFM) and NFVI resources (supported by the VIM) to ensure an optimized allocation of the necessary resources and connectivity
* **VNFM**: Virtualised Network Function Manager. functional block that is responsible for the lifecycle management of VNF
* **Consumer**: User-side Manager; Consumer of the interfaces produced by the VIM; VNFM, NFVO, or Orchestrator in ETSI NFV terminology.
* **Administrator**: ~~e.g. OpenStack Cloud Administrator [6]~~ e.g. OAM in Telco context
* **Virtual Machine (VM)**: virtualized computation environment that behaves very much like a physical computer/server
* **Virtualised Storage**: virtualised non-volatile storage allocated to a VM
* **Virtual network**: virtual network routes information among the network interfaces of VM instances and physical network interfaces, providing the necessary connectivity
* **Virtual Resource**: a Virtual Machine (VM), a virtual network, or virtualised storage.
* **VNF**: Virtualised Network Function. Implementation of an Network Function that can be deployed on a Network Function Virtualisation Infrastructure (NFVI)

# Table of content

[Table of content 3](#_Toc413676180)

[1 Introduction [editor: Ashiq] 5](#_Toc413676181)

[2 Use cases and scenarios [editor: Ashiq] 5](#_Toc413676182)

[2.1 Faults 6](#_Toc413676183)

[2.1.1 Fault management using ACT-STB configuration 6](#_Toc413676184)

[2.1.2 Recovery based on fault prediction 7](#_Toc413676185)

[2.2 NFVI Maintenance 7](#_Toc413676186)

[2.2.1 VM Retirement 7](#_Toc413676187)

[3 High level architecture and general features [editor: Ashiq?] + Tommy (Ericsson) 8](#_Toc413676188)

[3.1 Functional overview [Tommy] 8](#_Toc413676189)

[3.1.1 Failures of virtualised resources 9](#_Toc413676190)

[3.1.2 Planned maintenance of virtualized resources 9](#_Toc413676191)

[3.2 Architecture Overview 9](#_Toc413676192)

[3.3 General Features 10](#_Toc413676193)

[3.3.1 Detection 10](#_Toc413676194)

[3.3.2 Cognition 10](#_Toc413676195)

[3.3.3 Notification 11](#_Toc413676196)

[3.3.4 Recovery Action 11](#_Toc413676197)

[3.4 High level northbound interface specification [authors: Ashiq, Gerald, Ryota] 11](#_Toc413676198)

[3.4.1 Fault management 11](#_Toc413676199)

[3.4.2 NFVI Maintenance 13](#_Toc413676200)

[3.5 Information elements and parameters 14](#_Toc413676201)

[3.6 Faults [author: Gerald] 14](#_Toc413676202)

[4 Gap analysis in upstream projects [editor: Carlos] [authors: Gerald, Carlos, Tomi, Ryota] 16](#_Toc413676203)

[4.1 VIM Northbound Interface 16](#_Toc413676204)

[4.1.2 VIM Southbound interface 17](#_Toc413676205)

[4.2 OpenStack 18](#_Toc413676206)

[4.2.1 Ceilometer 18](#_Toc413676207)

[4.2.2 Nova 18](#_Toc413676208)

[4.2.3 Monasca 19](#_Toc413676209)

[4.3 Hardware monitoring tools 20](#_Toc413676214)

[4.3.1 Zabbix 20](#_Toc413676215)

[5 Detailed implementation plan [editor: Ryota] [authors: Gerald, Carlos, Tomi, Ryota] 20](#_Toc413676216)

[5.1 Functional Blocks 21](#_Toc413676217)

[5.1.1 Monitor 21](#_Toc413676218)

[5.1.2 Inspector 21](#_Toc413676219)

[5.1.3 Controller 21](#_Toc413676220)

[5.1.4 Notifier 22](#_Toc413676221)

[5.2 Sequence 22](#_Toc413676222)

[5.2.1 Fault Management 22](#_Toc413676223)

[5.2.2 NFVI Maintenance 24](#_Toc413676224)

[5.3 Implementation plan for Release 1 26](#_Toc413676225)

[5.3.1 Fault management 26](#_Toc413676226)

[5.3.2 NFVI Maintenance 27](#_Toc413676227)

[5.4 Information elements 27](#_Toc413676228)

[5.5 Detailed northbound interface specification 28](#_Toc413676229)

[5.5.1 Fault management interface 28](#_Toc413676230)

[5.5.2 NFVI maintenance 30](#_Toc413676318)

[5.6 Blueprints 33](#_Toc413676319)

[5.6.1 Instance State Notification (Ceilometer) 33](#_Toc413676320)

[5.6.2 Event Publisher for Alarm (Ceilometer) 33](#_Toc413676321)

[5.6.3 Notification-driven alarm evaluator (Ceilometer) 33](#_Toc413676322)

[5.6.4 Report host fault to update server state immediately (Nova) 33](#_Toc413676323)

[5.6.5 Other related BPs 34](#_Toc413676324)

[6 Summary and conclusion [editor: Ashiq] [authors: Gerald, …] 34](#_Toc413676325)

[6.1 Future plan 34](#_Toc413676326)

[7 References and bibliography 34](#_Toc413676327)

# Introduction [editor: Ashiq]

The goal of this project is to build a fault management and maintenance framework supporting Network Services on top of virtualized infrastructure. The key feature is immediate notification of unavailability of virtualized resources from VIM, to support the recovery of VNFs running on them. Requirement survey and development of missing features in NFVI and VIM are in scope of this project in order to fulfil requirements for fault management and maintenance in NFV.

The purpose of this requirement project is to clarify the necessary features of NFVI fault management and maintenance, identify missing features in the current OpenSource implementation, provide implementation guideline in relevant upstream project to realize those missing features, and define the VIM northbound interfaces necessary to perform the task of NFVI fault management and maintenance in ETSI NFV context [1], the list of gaps in upstream projects, potential implementation architecture and plan, and the VIM northbound interface specification for resource reservation and capacity management.

**Problem description**

A Virtualised Infrastructure Manager (VIM), e.g. OpenStack, cannot detect certain Network Functions Virtualisation Infrastructure (NFVI) faults, i.e. Resource Pool faults. This feature is necessary to detect the faults and notify the Consumer in order to ensure the proper functioning of EPC VNFs like MME and S/P-GW.

* EPC VNFs are often in active standby (ACT-SBY) configuration and need to switch to SBY mode as soon as relevant faults are detected in the active (ACT) VNF.
* NFVI encompasses all elements building up the environment in which VNFs are deployed, e.g., Physical Machines, Hypervisors, Storage, and Network elements.

In addition, VIM, e.g. OpenStack, needs to receive maintenance instructions from the Consumer, i.e. the operator/administrator of the VNF.

* Change the state of certain Physical Machines (PMs), e.g. empty the PM, so that maintenance work can be performed at these machines.

Note: Although fault management and maintenance are different operations in NFV, both are considered as part of this project as –except for the trigger- they share a very similar work and message flow. Hence, from implementation perspective, these two are kept together in the Doctor project because of this high degree of similarity.

**Features**

* Detect unavailability of physical resources (receive failure/maintenance notification from various functions)
  + Unavailability of physical resource is detected by various functions monitoring and/or managing individual H/W and S/W components
  + The cause of unavailability of physical resource to detect shall be configurable
* Identify affected virtualized resources
* Execute actions to process fault recovery and maintenance

The features are explained in more detail in Section 3.3.

# Use cases and scenarios [editor: Ashiq]

Telecom nodes often have very high requirements on service performance. As a consequence they often utilize redundancy and high availability (HA) mechanisms. The HA support may be built-in or provided by the platform. In any case, the HA support typically has a very fast detection and reaction time to minimize service impact. HA support for OPNFV is discussed in the High Availability for OPNFV project.

As an example, these nodes can come in an Active-Standby (ACT-SBY) configuration which is a (1+1) redundancy scheme. ACT and SBY nodes (aka Physical Network Function (PNF) in ETSI NFV terminology) are in a hot standby configuration. If an ACT node is unable to function properly due to fault or any other reason, the SBY node is instantly made ACT, and affected services can be provided without any service interruption.

The ACT-SBY configuration needs to be maintained. This means, when a SBY node is made ACT, either the previously ACT node, after recovery, shall be made SBY, or, a new SBY node needs to be configured. The actual operations to instantiate/configure a new SBY are similar to instantiating a new VNF and therefore are outside the scope of this project.

The NFVI fault management and maintenance requirements aim at providing fast failure detection of physical and virtualised resources and remediation of the virtualised resources provided to Consumers according to their predefined request to enable applications to recover to a fully redundant mode of operation.

The following three use cases show typical requirements and solutions for automated fault management and maintenance in NFV. The use cases assume that the VNFs are in an ACT-SBY configuration.

1. Fault management using ACT-STB configuration (Triggered by critical error)
2. Recovery based on fault prediction (Preventing service stop by handling warnings)
3. VM Retirement (Managing service while H/W maintenance)

## Faults

### Fault management using ACT-STB configuration

In Figure 1, a system-wide view of relevant functional blocks is presented. OpenStack is considered as the VIM implementation (aka Controller) which has interfaces with the Resource Pool (NFVI in ETSI NFV terminology) and the Consumers. The VNF implementation is represented as different virtual resources marked by different colours. Consumers (VNFM or NFVO in ETSI NFV terminology) own/manage the respective virtual resources shown with the same colours.

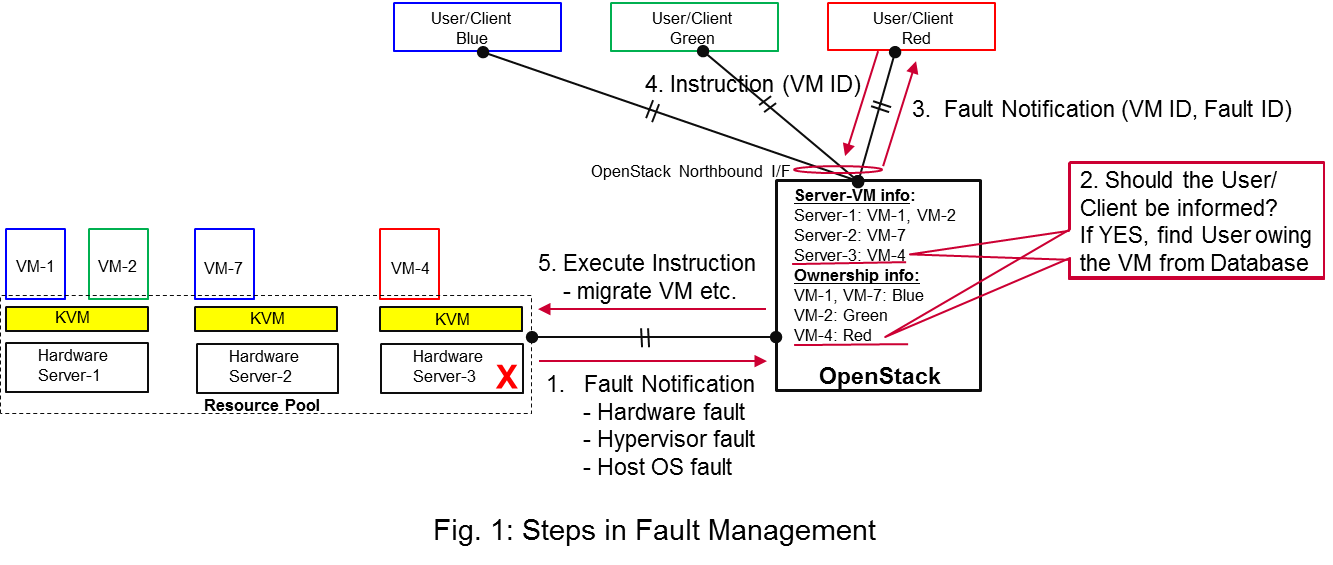


Figure 1 - Fault management use case

The first requirement in this use case is that the Controller needs to detect faults in the Resource Pool (“1. Fault Notification” in Figure 1) affecting the proper functioning of the virtual resources (labelled as VM-x) running on top of it. It should be possible to configure which relevant fault items should be detected. The VIM (e.g. OpenStack) itself could be extended to detect such faults. Alternatively, a third party fault monitoring tool could be used which then informs the VIM about such faults; this third party fault monitoring element can be considered as a component of VIM from an architectural point of view.

Once such fault is detected, the VIM shall find out which virtual resources are affected by this fault. In the example in Figure 1, VM-4 is affected by a fault in the Hardware Server-3. Such mapping shall be maintained in the VIM, depicted as the “Server-VM info” table inside the VIM.

Once the VIM has identified which virtual resources are affected by the fault, it needs to find out who is the Consumer (i.e. the owner/manager) of the affected virtual resources (Step 2). In the example shown in Figure 1, the VIM knows that for the red VM-4, the manager is the red Consumer through an Ownership info table. The VIM then notifies (Step 3 “Fault Notification”) the red Consumer about this fault, preferably with sufficient abstraction rather than detailed physical fault information.

The Consumer then switches to SBY configuration by switching the SBY VNF to ACT state. It further initiates a process to instantiate/configure a new SBY. However, switching to SBY mode and creating a new SBY machine is a VNFM/NFVO level operation and therefore outside the scope of this project. Doctor project does not create interfaces for such VNFM level configuration operations.

Once the Consumer has switched to SBY configuration, it notifies (Step 4 “Instruction” in Figure 1) the VIM. The VIM can then take necessary (e.g. pre-determined by the involved network operator) actions on how to clean up the fault affected VMs (Step 5 “Execute Instruction”).

The key issue in this use case is that a VIM (OpenStack in this context) shall not take a standalone fault recovery action (e.g. migration of the affected VMs) before the ACT-SBY switching is complete, as that might violate the ACT-SBY configuration and render the VNF out of service.

### Recovery based on fault prediction

The fault management scenario explained in Clause 2.1.1 can also be performed based on fault prediction. In such cases, in VIM, there is an intelligent fault prediction module which, based on its NFVI monitoring information, can predict an eminent fault in the elements of NFVI. A simple example is raising temperature of a Hardware Server which might trigger a pre-emptive recovery action. The requirements of such fault prediction in the VIM are investigated in the OPNFV “Fault prediction” project.

This use case is very similar to “Fault management using ACT-STB configuration” in Clause 2.1.1. Instead of a fault detection (Step 1 “Fault Notification in” Figure 1), the trigger comes from a fault prediction module in the VIM, or from a third party module which notifies the VIM about an eminent fault. From Step 2~5, the work flow is the same as in the “Fault management using ACT-STB configuration” use case, except in this case, the Consumer of a VM/VNF switches to SBY configuration based on a predicted fault, rather than an occurred fault.

## NFVI Maintenance

### VM Retirement

All network operators perform maintenance of their network infrastructure, both regularly and irregularly. Besides the hardware, virtualization is expected to increase the number of elements subject to such maintenance as NFVI holds new elements like the hypervisor and host OS. Maintenance of a particular resource element e.g. hardware, hypervisor etc. may render a particular server hardware unusable until the maintenance procedure is complete.

However, the Consumer of VMs needs to know that such resources **will** be unavailable because of NFVI maintenance. The following use case is again to ensure that the ACT-SBY configuration is not violated. A stand-alone action (e.g. live migration) from VIM/OpenStack to empty a physical machine so that consequent maintenance procedure could be performed may not only violate the ACT-SBY configuration, but also have impact on real-time processing scenarios where dedicated resources to virtual resources (e.g. VMs) are necessary and a pause in operation (e.g. vCPU) is not allowed. The Consumer is in a position to safely perform the switch between ACT and SBY nodes, or switch to an alternative VNF forwarding graph so the hardware servers hosting the ACT nodes can be emptied for the upcoming maintenance operation. Once the target hardware servers are emptied (i.e. no virtual resources are running on top), the VIM can mark them with an appropriate flag (i.e. “maintenance” state) such that these servers are not considered for hosting of virtual machines until these the maintenance flag is cleared (i.e. nodes are back in “normal” status).

A high-level view of the maintenance procedure is presented in Figure 2. VIM/OpenStack, through its northbound interface, receives a maintenance notification (Step 1 “Maintenance Instruction”) from the Administrator (e.g. a network operator) including information about which hardware is subject to maintenance. Maintenance operations include replacement/upgrade of hardware, update/upgrade of the hypervisor/host OS, etc.

The consequent steps to enable the Consumer to perform ACT-SBY switching are very similar to the fault management scenario. From VIM/OpenStacks internal database, it finds out which virtual resources (VM-x) are running on those particular Hardware Servers and who are the managers of those virtual resources (Step 2). The VIM then informs the respective Consumer (VNFMs or NFVO) in step 3 “Maintenance Notification”. Based on this, the Consumer takes necessary actions (e.g. switch to SBY configuration or switch VNF forwarding graphs) and then notifies (Step 4 “Instruction”) the VIM. Upon receiving such notification, the VIM takes necessary actions (Step 5 “Execute Instruction” to empty the Hardware Servers so that consequent maintenance operations could be performed. Due to the similarity for Steps 2~5, the maintenance procedure and the fault management procedure are investigated in the same project.

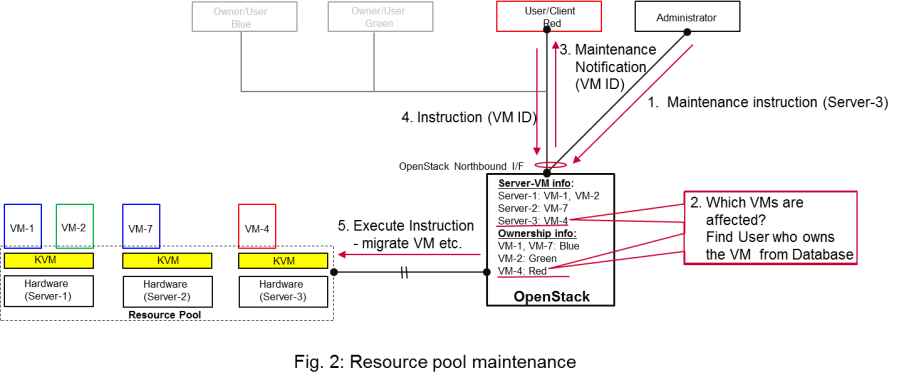


Figure 2 - Maintenance use case

# High level architecture and general features [editor: Ashiq?] + Tommy (Ericsson)

## Functional overview [Tommy]

The Doctor project circles around two distinct use cases: 1) management of failures of virtualized resources and 2) planned maintenance, e.g. migration, of virtualized resources. Both of them may affect a VNF/application and the network service it provides, but there is a difference in frequency and how they can be handled.

Failures are spontaneous events that may or may not have an impact on the virtual resources. The Consumer should as soon as possible react on the failure by switching to the STB node. The VIM should as soon as possible repair the lost services, i.e. restore the VM, VLAN or virtualized storage. How much the applications are affected varies. Applications with built-in HA support might experience a short decrease in retainability (e.g. an ongoing session might be lost) while keeping availability (establishment or re-establishment of sessions are not affected), whereas the impact on applications without built-in HA may be more serious. How much the network service is impacted depends on how the service is implemented. With sufficient network redundancy the service may be unaffected even when a specific resource fails.

Planned maintenance impacting virtualized resources on the other hand are events that are known in advance. This group includes e.g. migration due to SW upgrade of a compute host, change of CM characteristics due to scaling up/down and SW upgrades. Some of these might have been requested by the application or its management solution, but there is also a need for coordination on the actual operations on the virtual resources, There may be an impact on the applications and the service, but since they are not spontaneous events there is room for planning and coordination between the application management organization and the infrastructure management organization, including performing whatever actions that would be required to minimize the problems.

Failure prediction is the process of pro-actively identifying situations that may lead to a failure in the future unless acted on by means of maintenance activities. From application point of view, failure prediction may impact them in two ways: either the warning time is so short that the application or its management solution does not have time to react, in which case it is equal to the failure scenario, or there is sufficient time to avoid the consequences by means of maintenance activities, in which case it is similar to planned maintenance.

### Failures of virtualised resources

The functionalities related to failures of virtualised resources are:

#### Monitoring

The VIM shall monitor physical and virtual resources for unavailability and suspicious behaviour.

The physical resources are typically physical compute hosts, physical switches, physical storage equipment, but also additional equipment like fans, power supplies etc. The virtual resources are typically host OS, hypervisors, VLANs and virtual machines.

#### Detection

The VIM shall detect failures in physical and virtual resources in an unambiguous way. This may include also predicting upcoming faults. Note, fault prediction is out of scope of this project and is investigated in the OPNFV “Fault prediction” project.

#### Correlation

The VIM shall correlate each fault to the impacted virtual resource and make the alarm available over the northbound interface such that the Consumers impacted by the failure can take appropriate actions to recover from the failure.

#### Remediation

The VIM shall recover the failed virtual resources according to the default behaviour defined for that resource. In principle it means that an application can define which actions that can be taken. Examples are restart of the VM, migration of the VM or no action. However, this recovery operation in the VIM shall be coordinated by the Consumer.

### Planned maintenance of virtualized resources

The functionality is to be described.

## Architecture Overview

NFV and Cloud platform provide virtual resources and control functionality of them to users and administrators.

Figure 3 shows the high level architecture of NFV focusing on the *NFVI*, i.e. the virtualized infrastructure.

The NFVI provides virtual resources, such as virtual machines (VM) and virtual networks. Those virtual resources are used to run *applications* that could be component of a network service which is managed by the consumer(s) of the NFVI. The *Virtualized Infrastructure Manager* (VIM) provides functionalities of controlling and viewing virtual resources on hardware (physical) resources to the consumers, i.e. users and administrators. OpenStack is a prominent candidate for this VIM. The administrator may control the NFVI without using the VIM.



Figure 3 - High level architecture

Although OpenStack is the target upstream project where the new functional elements (Controller, Notifier, Monitory, and Inspector) are expected to be implemented, a particular implementation method is not assumed. Some of these elements may sit outside OpenStack and offer a northbound interface to OpenStack.

## General Features

The following features are required for the Virtualized Infrastructure Manager (VIM) to achieve high availability of applications (e.g. MME, S/P-GW) and the Network Services.

### Detection

VIM should detect unavailability of physical resources that might be cause errors/faults in virtual resources running on top of them. Unavailability of physical resource is detected by various monitoring and managing tools for hardware and software components.

The fault items/events to be detected shall be configurable.

The configuration shall enable Failure Selection and Aggregation. Failure aggregation means VIM can find out unavailability of physical resource from more than two non-critical failures related to the same resource.

There are two types of unavailability - immediate and future:

* Immediate unavailability can be detected by setting traps of raw failures on hardware monitoring tools.
* Future unavailability can be found by receiving maintenance instructions issued by the administrator of the physical resource pool or by failure prediction mechanisms.

### Cognition

VIM shall identify unavailability of virtualized resources that are or will be affected by failures on the physical resources under them. Unavailability of virtualized resource is found by referring to the mapping of physical and virtualized resources.

The relation from physical resources to virtualized resources shall be configurable, as the cause of unavailability of virtualized resources can be different in technologies and policies of deployment.

Failure aggregation is also required in this feature, e.g., a user may request more than two failures on standby VMs in an (N+M) deployment model.

### Notification

There are two types of notifications: a) notification about events of virtualized resource and b) notification on the update of the capacity of a resource pool.

The VIM shall notify the unavailability of virtual resources to the consumer owning it.

The VIM shall also notify the unavailability of physical resources to its administrator.

All notifications shall be transferred immediately in order to minimize the stalling time of the network service and to avoid over assignment caused by delay of capability updates.

There may be multiple consumers, so the VIM has to find out the owner of an faulty resource. Moreover, there may be a large number of virtual and physical resources in a real deployment, so polling the state of all resources to the VIM would lead to heavy signalling traffic. Thus, a publication/subscription messaging model is better suited for these notifications, as notifications are only sent to subscribed consumers .

Note: the VIM should only accept individual notification URLs for each resource by its owner or administrator.

Notifications reporting to the consumer about the unavailability of virtualized resources are including a description of the fault. Flexibility in the notifications is important, for example the receiver function in the consumer-side implementation could have different schema, location, and policies (e.g. receive or not, aggregate events with the same cause, etc).

### Recovery Action

The VIM is required to execute actions to process fault recovery and maintenance operations. All actions, done by the VIM and the NFVI after receiving those notifications, should be instructed/coordinated by the consumer (.i.e. the owner) of the resources or their administrator. Note, that instructions from the consumer to the VIM are not always required after such notifications.

(Option) For prompt recovery of faults, the VIM could have an additional feature to automate recovery actions for certain faults.

A delegated action could be automatically processed by the VIM if was already instructed earlier how to treat such type of faults, e.g, the VIM could automatically evacuate VM which are labelled with ‘allow live-migration’ by the owner of the VM.

## High level northbound interface specification [authors: Ashiq, Gerald, Ryota]

### Fault management

This interface allows the Consumer to subscribe to fault notification from the VIM. Using a filter, the Consumer can narrow down which faults should be notified. A fault notification will trigger the Consumer to switch from ACT to SBY configuration and initiate fault recovery actions. A fault query request/response message exchange allows the Consumer to find out about active alarms at the VIM. A filter can be used to narrow down the alarms returned in the response message.



Figure 4 - High-level message flow for fault management

The high level message flow for the fault management use case is shown in

Figure 4. It consists of the following steps:

Step 1: Request to subscribe to monitor specific virtual resources. A query filter can be used to narrow down the alarms the Consumer wants to be informed about.

Step 2: Request to update an existing subscription, either to remove/add virtual resources to be monitored or to update/remove the query filter.

Step 3: Each subscription (update) request is acknowledged with a subscribe response message. The response message contains information about the subscribed virtual resources, in particular if a subscribed virtual resource is in “alarm” state.

Step 4: The NFVI sends monitoring events for resources the VIM has been subscribed to. Note: this subscription message exchange between the VIM and NFVI is not shown in this message flow.

Step 5: Event correlation, fault detection and aggregation in VIM.

Step 6: Database lookup to find the virtual resources affected by the detected fault.

Step 7: Fault notification to Consumer.

Step 8: The Consumer switches to standby configuration (SBY)

Step 9: Instructions to VIM requesting certain actions to be performed on the affected resources, for example migrate/update/terminate specific resource(s). After reception of such instructions, the VIM is executing the requested action, e.g. it will migrate or terminate a virtual resource.

Step 10: Query request from Consumer to VIM to get information about the current status of a resource.

Step 11: Response to the query request with information about the current status of the queried resource. In case the resource is in “fault” state, information about the related fault(s) is returned.

Internal Note: Aligned with [IFA006\_Virtualized\_Resources\_Fault\_Management]

### NFVI Maintenance

The NFVI maintenance interface allows the Administrator to notify the VIM about a planned maintenance operation on the NFVI. A maintenance operation may for example be an update of the server firmware or the hypervisor. The MaintenanceRequest message contains instructions to change the state of the resource from “normal” to “maintenance”. After receiving the MaintenanceRequest, the VIM will notify the Consumer about the planned maintenance operation, whereupon the Consumer will switch to standby (SBY) configuration to allow the maintenance action to be executed. After the request was executed successfully (i.e., the physical resources have been emptied) or the operation resulted in an error state, the VIM sends a MaintenanceResponse message back to the Administrator.



Figure 5 - High-level message flow for NFVI maintenance

The high level message flow for the NFVI maintenance use case is shown in Figure 5. It consists of the following steps:

Step 1: Subscribe/response signalling to subscribe to fault/maintenance notifications.

Step 2: Maintenance trigger received from administrator.

Step 3: VIM switches NFVI resources to “maintenance” state. This, e.g., means they should not be used for further allocation/migration requests

Step 4: Database lookup to find the virtual resources affected by the detected maintenance operation.

Step 5: Maintenance notification to Consumer .

Step 6: The Consumer switches to standby configuration (SBY)

Step 7: Instructions from Consumer to VIM requesting certain recovery actions to be performed (step 7a). After reception of such instructions, the VIM is executing the requested action in order to empty the physical resources (step 7b).

Step 8: Maintenance response from VIM to inform the Administrator that the physical machines have been emptied (or the operation resulted in an error state).

Step 9: Administrator is coordinating and executing the maintenance operation/work on the NFVI.

Step 10: Query request from Consumer to VIM to get information about the current state of a resource.

Step 11: Response to the query request with information about the current state of the queried resource(s). In case the resource is in “maintenance” state, information about the related maintenance operation is returned.

## Information elements and parameters

Raw list of information elements and parameters (please refer to Section XXX for a more detailed specification of information elements).

* Fault Notification
  + VirtualResourceID: identifies the virtual resource that is affected by a fault on the underlying physical resource
  + FaultID: identifies the related fault in the underlying physical resource. This can be used to correlate different fault notifications caused by the same fault in the physical resource.
  + FaultDetails: provides more detailed information about the fault.
* Maintenance Request/Response
  + (Physical)ResourceID: identifies the physical resource on which the maintenance action is planned.
  + State = “maintenance”: requests the VIM to change the state of the resource to maintenance. State may also be set to “normal” after the maintenance action has been performed (or was canceled).
  + Result: informs the Administrator about the new state of the resource. It can also provide additional meta-data, e.g. on whether the request had been processed successfully or had resulted in an error.
* State Change Notification
  + VirtualResourceID: see above

## Faults [author: Gerald]

Faults in the listed elements need to be immediately notified to the VNFM in order to perform an immediate action like live migration or switch to a hot standby entity. In addition, a maintenance action should be triggered to, e.g., reboot the server or replace a defect hardware element.

Faults can be of different severity, i.e. critical, warning, maintenance, or info. Critical faults require immediate action as a severe degradation of the system has happened or is expected. Warnings indicate that the system performance is going down: related actions include closer (e.g. more frequent) monitoring of that part of the system or preparation for a cold migration to a backup VM. Type maintenance may trigger maintenance actions like a re-boot of the server or replacement of a faulty, but redundant HW. Info messages do not require any action.

Faults can be gathered by, e.g., enabling SNMP and installing some open source tools to catch and poll SNMP. When using for example Zabbix one can also put an agent running on the hosts to catch any other fault. Table 1 provides a list of high level faults that are considered within the scope of the Doctor project requiring immediate action by the VNFM.

Table 1 - High level list of faults

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Service | Fault | Severity | How to detect? | Comment | Action to recover |
| Compute Hardware | Processor/CPU failure, CPU condition not ok | Critical | Zabbix |  | Switch to hot standby |
| Memory failure / Memory condition not ok | Critical | Zabbix (IPMI) |  | Switch to hot standby |
| Network card failure, e.g. Network adapter connectivity lost | Critical | Zabbix / Ceilometer |  | Switch to hot standby |
| Disk crash | Info | RAID monitoring | Network storage is very redundant (e.g. RAID system) and can guarantee high availability. | Inform OAM |
| Disk aging | Info | S.M.A.R.T (IPMI or OS) |  | Inform OAM |
| Storage controller | Critical | Zabbix (IMPI) |  | Live migration if storage is still accessible; otherwise Hot Standby |
| PDU/power failure, power off, server reset | Critical | Zabbix / Ceilometer |  | Switch to hot standby |
| Power degradation, Power redundancy lost, Power threshold exceeded | Warning | SNMP |  | Live migration |
| Chassis problem (e.g. fan degraded/failed, chassis power degraded), CPU fan problem, Temperature/thermal condition not okay | Warning | SNMP |  | Live migration |
| Mainboard failure | Critical | Zabbix (IPMI) |  | Switch to hot standby |
| OS crash (e.g. kernel panic) | Critical | Zabbix |  | Switch to hot standby |
| Hyper- visor | System has restarted | Critical | Zabbix |  | Switch to hot standby |
| Hypervisor failure | Warning / Critical | Zabbix / Ceilometer |  | Migration / switch to hot standby |
| Zabbix / Ceilometer is unreachable | Warning | ? |  | Live migration |
| Network | SDN/OpenFlow Switch/Controller degraded/failed | Critical | ? |  | Switch to hot standby or reconfigure virtual network topology |
| HW failure of physical switch/router | Warning | SNMP | Redundancy of physical infrastructure is reduced or no longer available. | Inform OAM to replace defect HW and configure new HW |

# Gap analysis in upstream projects [editor: Carlos] [authors: Gerald, Carlos, Tomi, Ryota]

[<https://etherpad.opnfv.org/p/doctor_gap_analysis> ]

This section presents the findings of gaps on existing VIM platforms. The focus was to identify gaps Doctor depends on based on the requirements specified in Section XXX. The analysis work performed resulted in the identification of gaps of which are herein presented.

For a better and concise understanding, a standardized table format is used to identify the requirements, gaps and gap solutions when applicable. Table XXX describes the columns in the standardized table format.

## VIM Northbound Interface

#### Immediate Notification

* **Category**: VIM NB I/F
* **Type**: 'deficiency in performance'
* **Description**:
* To-be:
  + VIM has to notify unavailability of virtual resource (fault) to VIM user immediately.
  + Notification should be passed in '1 second' after fault detected/notified by VIM.
  + Also, the following conditions/requirement have to be met:
    - Only the user can receive notification of fault related to owned virtual resource(s).
* As-is:
  + OpenStack Metering 'Ceilometer' can notify unavailability of virtual resource (fault) to the owner of virtual resource based on alarm configuration by the user.
    - Ceilometer Alarm API: [http://docs.openstack.org/developer/ceilometer/webapi/v2.html#alarms](http://docs.openstack.org/developer/ceilometer/webapi/v2.html" \l "alarms)
  + Alarm notifications are triggered by alarm evaluator instead notification agents that might receive faults.
    - Ceilometer Architecture: <http://docs.openstack.org/developer/ceilometer/architecture.html#id1>
  + Evaluation interval should be equal to or larger than configured pipeline interval for collection of underlying metrics.
    - <https://github.com/openstack/ceilometer/blob/stable/juno/ceilometer/alarm/service.py#L38-42>
  + The interval for collection has to be set large enough which depends on the size of the deployment and the number of metrics to be collected.
  + The interval may not be less than one second in even small deployments. The default value is 60 seconds.
  + Alternative: OpenStack has a message bus to publish system events. Operator can allow user to connect this, but there are no functions to filter out other events that should not be passed to the user or does not requested by the user.
* Gap
  + Fault notifications cannot be received \***immediately**\* by Ceilometer.
* **Related blueprints**:
* ...

#### Maintenance Notification

* **Category**: VIM N/B I/F
* **Type:** 'missing'
* **Description**:
* To-be:
  + VIM has to notify unavailability of virtual resource triggered by NFVI maintenance to VIM user.
  + Also, the following conditions/requirements have to be met:
    - VIM should accept maintenance message from administrator and mark target physical resource "in maintenance".
    - Only the owner of virtual resource hosted by target physical resource can receive the notification that can trigger some process for applications which are running on the virtual resource (e.g. cut off VM).
* As-is:
  + OpenStack: None
  + AWS (just for study)
    - AWS provides API and CLI to view status of resource (VM) and to create instance status and system status alarms to notify you when an instance has a failed status check.  
      <http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/monitoring-instances-status-check_sched.html>
    - AWS provides API and CLI to view scheduled events, such as a reboot or retirement, for your instances. Also, those events will be notified via e-mail.  
      <http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/monitoring-system-instance-status-check.html>
* Gap
  + VIM user cannot receive maintenance notifications.
* **Related blueprints**
* ...

### VIM Southbound interface

#### Normalization of data collection models

* **Type**: 'missing'
* **Description**:
* To-be:
  + A normalized data format needs to be created to cope with the many data models from different monitoring solutions.
* As-is:
  + Data can be collected from many places (e.g. Zabbix, Nagios, Cacti, Zenoss). Although each solution establishes its own data models, no common data abstraction models exist in OpenStack.
* Gap:
  + Normalized data format does not exist.
* **Related blueprints**:
* ...

## OpenStack

### Ceilometer

OpenStack offers a telemetry service, Ceilometer, for collecting measurements of the utilization of physical and virtual resources [3]. Ceilometer can collect a number of metrics across multiple OpenStack components and watch for variations and trigger alarms based upon on the collected data.

#### Scalability of fault aggregation

* **Type:** 'scalability issue'
* **Description**:
* To-be:
  + Be able to scale to a large deployment, where thousands of monitoring events per second need to be analyzed
* As-is:
  + Performance issue when scaling to medium-sized deployments
* Gap:
  + Ceilometer seems not suitable for monitoring medium and large scale NFVI deployments
* **Related blueprints**:

1. Usage of Zabbix for fault aggregation. Zabbix can support a much higher number of fault events (up to 15.000 events per second, but obviously also has some upper bound  
   <http://blog.zabbix.com/scalable-zabbix-lessons-on-hitting-9400-nvps/2615/>
2. Decentralized/hierarchical deployment with multiple instances, where one instance is only responsible for a small NFVI

#### Monitoring of hardware and software

* **Type**: 'missing' (lack of functionality)
* **Description**:
* To-be:
  + Ceilometer was not designed to monitor hardware and software
* As-is:
  + Need to be able to detect the following faults: <https://wiki.opnfv.org/doctor/faults>
* Gap:
  + Ceilometer is not able to detect all faults listed in the link above
* **Related blueprints / workarounds**:
* Use other dedicated monitoring tools like Zabbix or Monasca

### Nova

OpenStack Nova [4] is a mature and widely known and used component in OpenStack cloud deployments. It is the main part of an infrastructure as a service system providing a cloud computing fabric controller, supporting a wide diversity of virtualization and container technologies.

Nova has proven throughout these past years to be highly available and fault-tolerant. Featuring its own API, it also provides a compatibility API with Amazon EC2 APIs.

#### Fencing instances of an unreachable host.

* **Type:** 'missing'
* **Description**:
* To-be:
  + Safe VM evacuation has to be preceded by fencing (isolate, shut down) the failed host. Failing to do so – when the perceived disconnection is due to some transient or partial failure – the evacuation might lead into two identical instances running together and having a dangerous conflict.
  + Fencing Instances of an unreachable host:   
    <https://wiki.openstack.org/wiki/Fencing_Instances_of_an_Unreachable_Host>
* As-is:
  + When a VM goes down due to a host HW, host OS or hypervisor failure, nothing happens in OpenStack. The VMs of a crashed host/hypervisor are reported to be live and OK through the OpenStack API.
* Gap:
  + Openstack does not fence instances of an unreachable host.
* **Related blueprints**:
  + <https://blueprints.launchpad.net/nova/+spec/fencing>

#### Evacuate VMs on Maintenance mode

* **Type:** 'missing'
* **Description**:
* To-be:
  + When maintenance mode for a compute host is set, trigger VM evacuation to available compute nodes before bringing the host down for maintenance
* As-is:
  + If setting a compute node to a maintenance mode, OpenStack only schedules evacuation of all VMs to available compute nodes if in-maintenance compute node runs the XenAPI and VMware ESX hypervisors. Other hypervisors (e.g. KVM) are not supported and, hence, guest VMs will likely stop running due to maintenance actions administrator may perform (e.g. hardware upgrades, OS updates).
* Gap:
  + Nova libvirt hypervisor driver does not implement automatic guest VMs evacuation when compute nodes are set to maintenance mode ($ nova host-update --maintenance enable <hostname>)
* **Related blueprints**:
* ...

### Monasca

Monasca is an open-source monitoring-as-a-service (MONaaS) solution that integrates with OpenStack. Even though it is still in its early days, it is the interest of the community that the platform be multi-tenant, highly scalable, performant and fault-tolerant. Companion with a streaming alarm engine and a notification engine, is a northbound REST API users can use to interact with Monasca. Hundreds of thousands of metrics per second can be processed [5].

#### Anomaly detection

* **Type:** 'missing' (lack of functionality)
* **Description:**
* To-be:
  + Detect the failure and perform a root cause analysis to filter out other alarms that may be triggered due to their cascading relation.
* As-is:
  + A mechanism to detect root causes of failures is not available.
* Gap:
  + Certain failures can trigger many alarms due to their dependency on the underlying root cause of failure. Knowing the root cause can help filter out unnecessary and overwhelming alarms.
* **Related blueprints / Workaround:**
* Monasca as of now lacks this feature, although the community is aware and working toward supporting it

#### Sensor monitoring

* **Type:** 'missing' (lack of functionality)
* **Description:**
* To-be:
  + It should support monitoring sensor data retrieving, for instance, from IPMI.
* As-is:
  + Monasca does not monitor sensor data.
* Gap:
  + Sensor monitoring is of the most importance. It provides operators status on the state of the physical infrastructure (e.g. temperature, fans)
* **Related blueprints / Workaround:**
* Monasca can be configured to use third-party monitoring solutions (e.g. Nagios, Cacti) for retrieving additional data.

## Hardware monitoring tools

### Zabbix

#### Delay in execution of actions

* **Type:** 'deficiency in performance'
* **Description:**
* To-be:
  + After detecting a fault, the monitoring tool should immediately execute the appropriate action, e.g. inform the manager through the NB I/F
* As-is:
  + A delay of around 10 seconds was measured in two independent testbed deployments
* Gap:
  + Cause of the delay needs to be identified and fixed
* **Related blueprints:**
* N/A

# Detailed implementation plan [editor: Ryota] [authors: Gerald, Carlos, Tomi, Ryota]

## Functional Blocks



Figure 5 - Functional blocks

### Monitor

The Monitor module has the responsibility for monitoring the virtualized infrastructure. There are already many existing tools and services (e.g. Zabbix) to monitor different aspects of hardware and software resources.

### Inspector

The Inspector module has the ability to receive various failure notifications regarding physical resource(s) from Monitor module(s), to find affected referring resource map from physical to virtual, and update the state of the virtual resource (and physical resource).

The Inspector has drivers for different types of events and resources to integrate any type of monitor and controller modules. It also uses a failure policy which instructs the Inspector on the failure selection and aggregation from raw events. This failure policy is configured by administrator.

The reason for separation of the Inspector and Controller modules is to make the Controller focus on simple operations by avoiding a tight integration of various health check mechanisms into the Controller.

### Controller

The Controller is responsible for maintaining the resource map, accepting update requests for the resource state(s) (exposing as provider API), notifying all events regarding virtual resources to the Notifier. Optionally, the Controller has the ability to poison the state of virtual resources mapping to physical resources for which it has received failure notifications from the Inspector. The Controller also re-calculates the capability of a Resource Pool when receiving a failure notification for a physical resource and it notifies the updated capacity as an event to the Notifier.

In a real-world deployment, the VIM may have several controllers for each resource types such as Nova, Neutron and Cinder in OpenStack. Each controller maintains a database of virtual and physical resources which shall be master information in VIM.

### Notifier

The focus of the Notifier is on selecting and aggregating failure events received from the controller based on policies mandated by the consumer. Therefore, it allows the Consumer (and Administrator) to subscribe for alarms regarding virtual resources (using a method such as API endpoint), and it will then notify events to the Consumer by referring to the alarm configuration after having received events from a Controller.

The northbound interface between the Notifier and the Consumer / Administrator is specified in Section 5.4.

## Sequence

### Fault Management



Figure 6 - Fault management work flow

Detailed work flow:

Step 1: Request to subscribe to monitor specific virtual resources. A query filter can be used to narrow down the alarms the Consumer wants to be informed about.

Step 2: Each subscription request is acknowledged with a subscribe response message. The response message contains information about the subscribed virtual resources, in particular if a subscribed virtual resource is in “alarm” state.

Step 3: The NFVI sends monitoring events for resources the VIM has been subscribed to. Note: this subscription message exchange between the VIM and NFVI is not shown in this message flow.

Step 4: Event correlation, fault detection and aggregation in VIM.

Step 5: Database lookup to find the virtual resources affected by the detected fault.

Step 6: Fault notification to Consumer.

Step 7: The Consumer switches to standby configuration (SBY)

Step 8: Instructions to VIM requesting certain actions to be performed on the affected resources, for example migrate/update/terminate specific resource(s). After reception of such instructions, the VIM is executing the requested action, e.g. it will migrate or terminate a virtual resource.

Step 9: Query request from Consumer to VIM to get information about the current status of a resource.

Step 10: Response to the query request with information about the current status of the queried resource. In case the resource is in “fault” state, information about the related fault(s) is returned.



Figure 7 - Fault management scenario

Figure 7 shows a more detailed message flow (Steps 4 to 6) between the 4 building blocks introduced in Section 5.1.

Step 4: The Monitor of the NFVI observed a fault and reports it to the Inspector.

Step 5b: The Inspector aggregates… using the failure policy (?).

Step 5a: The Inspector updates the state of the affected virtual/physical (?) resources in the Resource Map.

Step 5b: The Inspector queries the Resource Map to find the virtual resources affected by the raw fault.

Step 5c: The Controller observes a change of the resource state (???) and informs the Notifier.

Step 7: A fault notification is sent to northbound to the Consumer.

### NFVI Maintenance



Figure 8 – NFVI maintenance work flow

The detailed work flow for NFVI maintenance is shown in Figure 8 and has the following steps:

Step 1: Subscribe to fault/maintenance notifications.

Step 2: Response to subscribe request.

Step 3: Maintenance trigger received from administrator.

Step 4: VIM switches NFVI resources to “maintenance” state. This, e.g., means they should not be used for further allocation/migration requests

Step 5: Database lookup to find the virtual resources affected by the detected maintenance operation.

Step 6: Maintenance notification to Consumer.

Step 7: The Consumer switches to standby configuration (SBY)

Step 8: Instructions from Consumer to VIM requesting certain recovery actions to be performed (step 7a). After reception of such instructions, the VIM is executing the requested action in order to empty the physical resources (step 7b).

Step 9: Maintenance response from VIM to inform the Administrator that the physical machines have been emptied (or the operation resulted in an error state).

Step 10: Administrator is coordinating and executing the maintenance operation/work on the NFVI.

Step 11: Query request from Consumer to VIM to get information about the current state of a resource.

Step 12: Response to the query request with information about the current state of the queried resource(s). In case the resource is in “maintenance” state, information about the related maintenance operation is returned.

Figure 9 shows the implementation plan for the NFVI maintenance use case. Note that steps 1, 2, and 5 to 8a in the NFVI maintenance work flow are similar to the steps in the fault management work flow and share the same implementation plan in Release 1.



Figure 9 - NFVI Maintenance implementation plan

TODO: Only explain here the interface Admin to VIM. Which module is receiving the MaintenanceRequest from the Administrator? (Steps 3 and 9).

## Implementation plan for Release 1

### Fault management



Figure 10 - Implementation plan in OpenStack (Release 1)

Description for Fig. 10 TBD

Shall we describe the 3 different potential approaches and explain why we have selected this approach? (see slides Ryota from 5.3.2015)



Figure 11 – Implementation plan in Ceilometer architecture[[1]](#footnote-1)

Figure 11 … Approach is to a) Hook in notification agent (“publisher” in Ceilometer terminology) and b) Able to capture data update only for immediate notification

### NFVI Maintenance

Figure + Description TBD

For NFVI Maintenance, a quite similar implementation plan exists. Instead of a raw fault being observed by the Monitor … TBD

## Information elements

Simple information elements:

* ResourceID (Identifier): identifies the physical or virtual resource that is, e.g., affected by a fault or a maintenance action of the underlying physical resource.
* State (String): state of a resource. The state can be “normal”, “maintenance”, “down”, “error”:
* FaultID (Identifier): identifies the related fault in the underlying physical resource. This can be used to correlate different fault notifications caused by the same fault in the physical resource.
* FaultDetails (TBD): provides more detailed information about the fault.
* Metadata (Key-Value-Pairs): can contain additional information like details about resources in maintenance/error state.

Complex information elements:

* ResourceInfoType:
  + ResourceID
  + ResourceState
  + Metadata
* ResourceFaultType:
  + ResourceID
  + Faults (FaultType): For each resource, all faults including detailed information about the faults are provided.
    - FaultID
    - FaultDetails
* SubscribeFilterType
  + TBD
* FaultQueryFilterType:
  + TBD: narrows down the FaultQueryRequest, for example it limits the query to certain physical resources, a certain zone, a given fault type/severity/cause, or a specific FaultID.
* StateQueryFilterType:
  + TBD: narrows down the StateQueryRequest, for example it limits the query to certain physical resources, a certain zone, or a given state (e.g. only resources in “maintenance” state)
* ResourceState (ResourceStateType):
  + ResourceID (Identifier): identifies a physical or virtual resource(s)
  + State (String): returns the current state of the reported resource.

## Detailed northbound interface specification

INTERNAL NOTE: Interface specification: methods, inputs, outputs, … (should not be too related to OpenStack).  
Describe entities , e.g. in JSON

### Fault management interface

This interface allows the VIM to notify the Consumer about a virtual resource that is affected by a fault, either within the virtual resource itself or by the underlying virtualization infrastructure. The messages on this interface are shown in Figure 12 and explained in detail in the following subsections.



Figure 12 - Fault management NB I/F messages

#### SubscribeRequest (Consumer 🡪 VIM)

Subscription from Consumer to VIM to be notified about faults of specific resources. The faults to be notified about can be narrowed down using a subscribe filter.

Parameters:

* Filter (SubscribeFilterType): Information to narrow down the faults that shall be notified to the Consumer, for example limit to specific ResourceID(s), severity, or cause of the alarm.

#### SubscribeResponse (VIM 🡪 Consumer)

Response to a subscribe request message including information about the subscribed resources, in particular if they are in “fault/error” state.

Parameters:

* SubscribeID (Identifier): Unique identifier for the subscription. It can be used to delete or update the subscription.
* ResourceInfo (ResourceInfoType): Provides additional information about the subscribed resources, i.e., a list of the related resources, the current state of the resources, etc.

#### FaultNotification (VIM 🡪 Consumer)

Notification about a virtual resource that is affected by a fault, either within the virtual resource itself or by the underlying virtualization infrastructure. After reception of this request, the Consumer will decide about the optimal action to resolve the fault. This includes actions like switching to a hot standby virtual resource, migration of the fault virtual resource to another physical machine, termination of the faulty virtual resource and instantiation of a new virtual resource in order to provide a new hot standby resource. Existing resource management interfaces and messages between the Consumer and the VIM can be used for those actions, and there is no need to define additional actions on the Fault Management Interface.

Parameters:

* ResourceInfo (ResourceInfoType): List of faulty resources with detailed information about the faults (see Section XYZ).

#### FaultQueryRequest (Consumer 🡪 VIM)

Request to find out about active alarms at the VIM. A FaultQueryFilter can be used to narrow down the alarms returned in the response message.

Parameters:

* FaultQueryFilter (FaultQueryFilterType): narrows down the FaultQueryRequest, for example it limits the query to certain physical resources, a certain zone, a given fault type/severity/cause, or a specific FaultID.

#### FaultQueryResponse (VIM 🡪 Consumer)

List of active alarms at the VIM matching the FaultQueryFilter specified in the FaultQueryRequest.

Parameters:

* ResourceInfo (ResourceInfoType): List of faulty resources. For each resource all faults including detailed information about the faults are provided (see Section XYZ).











### NFVI maintenance

The NFVI maintenance interface allows …TBD

The messages defined in this northbound interface are shown in Figure XXX and described in detail in the following subsections.



Figure 13 – NFVI maintenance NB I/F messages

#### SubscribeRequest (Consumer 🡪 VIM)

Subscription from Consumer to VIM to be notified about maintenance operations for specific resources. The resources to be informed about can be narrowed down using a subscribe filter.

Parameters:

* Filter (SubscribeFilterType): Information to narrow down the faults that shall be notified to the Consumer, for example limit to specific ResourceID(s).

#### SubscribeResponse (VIM 🡪 Consumer)

Response to a subscribe request message, including information about the subscribed resources, in particular if they are in “maintenance” state.

Parameters:

* SubscribeID (Identifier): Unique identifier for the subscription. It can be used to delete or update the subscription.
* ResourceInfo (ResourceInfoType): Provides additional information about the subscribed resources, i.e., a list of the related resources, the current state of the resources, etc.

#### StateChangeNotification (VIM 🡪 Consumer)

Notification about a resource that was switched to “maintenance” state. After reception of this request, the Consumer will decide about the optimal action to address this request, e.g. to switch to the standby configuration.

Parameters:

* ResourceInfo (ResourceInfoType): List of resources where the state has been changed to maintenance (see Section XYZ).

#### MaintenanceRequest (Administrator 🡪 VIM)

Request to change the state of a list of resources, e.g., to “maintenance” state in order to prepare them for a planned maintenance operation.

Parameters:

* ResourceID
* State = “maintenance”:

#### MaintenanceResponse (VIM 🡪 Administrator)

Response message to inform the Administrator that the requested resources are now in maintenance state (or the operation resulted in an error) and the maintenance operation(s) can be executed.

Parameters:

* ResourceID
* Result (key-value pairs)

#### StateQueryRequest (Administrator 🡪 VIM)

Request to find out about resources in maintenance state. A filter can be used to narrow down the resources returned in the response message.

Parameters:

* StateQueryFilter (StateQueryFilterType): narrows down the StateQueryRequest, for example it limits the query to certain physical resources, a certain zone, or a given resource state.

#### StateQueryResponse (VIM 🡪 Administrator)

List of resources in maintenance state matching the filter specified in the StateQueryRequest.

Parameters:

* ResourceInfo (ResourceInfoType): List of resources. For each resource the current state is provided (see Section XYZ).



## Blueprints

Internal NOTE: For each BP, write a high level description, then paste the BP text as submitted to the community.

### Instance State Notification (Ceilometer)

This BP is adding … TBD

The blueprint submitted to OpenStack reads as follow:

*This BP proposes to add a new compute notification state entry to handle events from an instance (server) from nova. It also creates a new meter "instance.state".*

*The BP focuses on creating the metric "instance.state" by using a notification agent rather than a polling based approach, as it would be used with an “Event Publisher for Alarm” (see Section XYZ).*

### Event Publisher for Alarm (Ceilometer)

This BP is adding … TBD

The blueprint submitted to OpenStack reads as follow:

*The proposal is to create a new event publisher which can send messages to a new alarm evaluator [BP#3]. The publisher enables Ceilometer to provide event driven notifications to the user. Besides the existing Ceilometer usage for billing purposes, this BP enhances Ceilometer to provide additional notification capabilities to the user.*

### Notification-driven alarm evaluator (Ceilometer)

This BP is adding … TBD

The blueprint submitted to OpenStack reads as follow:

*This BP proposes a notification-driven alarm evaluator that is using event notifications received from a "event publisher for alarm" [BP#2]. The alarm evaluator does not execute any periodical task, but is triggered by alarm notifications. The alarm evaluator will aggregate and correlate different alarms, which will then be notified to the user in order to trigger recovery action(s) on the user-side (e.g. migrate, terminate, re-instantiate etc.).*

### Report host fault to update server state immediately (Nova)

This BP is adding … TBD

The blueprint submitted to OpenStack reads as follow*[[2]](#footnote-2)*:

*When a server goes down because of a host hardware, OS, or hypervisor error, the server state remains as operational in OpenStack API. A new API is needed to report that a host fault and to change the state of the server(s) immediately. The new API provides the possibility to externally detect any kind of host failure and to inform OpenStack about it.*

### Other related BPs

#### HA BP

TBD

#### pacemaker-servicegroup-driver

New API to report host fault might be very nice. Any external tool (like Doctor), could detect host error fast and use the API to mark host faulty (and fence the host by shutting down if needed). Anyhow this new API might not get so easily trough to openstack. For this there is already work to get pacemaker servicegroup driver to have host state change fast:

https://blueprints.launchpad.net/nova/+spec/pacemaker-servicegroup-driver.

For server state might have BP to see the host state when querying the server. Now if host state is changed faster, this might fulfill VNFM needs (with current way openstack works), before there is new NB IF in place (to report host faults) as that implementation might take longer as more complicated issue.

#### LibvirtWatchdog#Notifications

Related info about watchdog devices, quoting from BP [x]: "This is useful to cloud users to deal with problems in their guest OS, to kill off a mis-behaving instance to allow an external HA solution to move processing to another instance."

This BP is in status completed but the notification part that never got implemented. The intention I guess was that the nova-compute libvirt driver should also subscribe to WATCHDOG events from libvirt. nova-compute libvirt driver today is subscribed to LIFECYCLE events. I mention this because watchdog notifications are a topic in the ETSI REL spec.

So I guess some nova BP is also needed to add more libvirt notifications into nova compute libvirt driver?

https://wiki.openstack.org/wiki/LibvirtWatchdog#Notifications

Providing VNF faults through VIM could be helpful.

I think you can propose nova to make nova-compute subscribe to WATCHDOG events from libvirt and emit those events to Ceilometer.

It might be nice that user can express whether monitoring required by selecting flag in flavor or image property to reduce unnecessary notification in operator perspective.

# Summary and conclusion [editor: Ashiq] [authors: Gerald, …]

## Future plan

# References and bibliography

1. OPNFV, “Doctor Project,” [Online]. Available at <https://wiki.opnfv.org/doctor>
2. OpenStack, [Online]. Available at <https://www.openstack.org/>
3. OpenStack Telemetry (Ceilometer), [Online]. Available at <https://wiki.openstack.org/wiki/Ceilometer>
4. OpenStack Nova, [Online]. Available at <https://wiki.openstack.org/wiki/Nova>
5. OpenStack Monasca, [Online], Available at <https://wiki.openstack.org/wiki/Monasca>
6. OpenStack Cloud Administrator Guide, [Online]. Available at http://docs.openstack.org/admin-guide-cloud/content/
7. ETSI NFV GS…

1. http://docs.openstack.org/developer/ceilometer/architecture.html [↑](#footnote-ref-1)
2. https://blueprints.launchpad.net/nova/+spec/update-server-state-immediately [↑](#footnote-ref-2)