|  |
| --- |
|  |
| Title: | Doctor: Fault Management and Maintenance |
|  |  |
| from **Source**: | OPNFV Doctor project<https://wiki.opnfv.org/doctor>  |
|  |  |
| Editors:  | Ashiq Khan (NTT DOCOMO, khan@nttdocomo.com),Gerald Kunzmann (NTT DOCOMO, kunzmann@docomolab-euro.com) |
| Authors: | Ryota Mibu (NEC, r-mibu@cq.jp.nec.com) |
|  | Carlos Goncalves (NEC, Carlos.Goncalves@neclab.eu) |
|  | Tomi Juvonen (Nokia, tomi.juvonen@nsn.com) |
|  | … |
|  |  |
|  |  |
| Project creation date: | 2014-12-02 |
| Submission date: | 2015-02-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
* (ESTI NFV) NFVI: totality of all hardware and software components which build up the environment in which VNFs are deployed
* Virtual Resource: e.g. a Virtual Machine (VM), virtual network
* (User-/admin-side) Manager: VNFM or Orchestrator
* Controller: VIM

# Table of content

[Table of content 2](#_Toc411865886)

[1 Introduction [editor: Ashiq] 3](#_Toc411865887)

[2 Use cases and scenarios [editor: Ashiq] 3](#_Toc411865888)

[2.1 Faults 3](#_Toc411865889)

[2.1.1 Auto healing 3](#_Toc411865890)

[2.1.2 Safe Switching 4](#_Toc411865891)

[2.2 Maintenance 4](#_Toc411865892)

[3 High level architecture and general features [editor: Ashiq?] + Tommy (Ericsson) 5](#_Toc411865893)

[3.1 Architecture Overview 5](#_Toc411865894)

[3.2 General Features 6](#_Toc411865895)

[3.2.1 Detection 6](#_Toc411865896)

[3.2.2 Cognition 7](#_Toc411865897)

[3.2.3 Notification 7](#_Toc411865898)

[3.2.4 Recovery Action 7](#_Toc411865899)

[3.3 High level northbound interface specification [authors: Ashiq, Gerald] 8](#_Toc411865900)

[3.3.1 Fault management 8](#_Toc411865901)

[3.3.2 NFVI Maintenance 8](#_Toc411865902)

[3.4 High level northbound interface specification [authors: Ashiq + Ryota] 9](#_Toc411865903)

[3.5 Faults [author: Gerald] 9](#_Toc411865904)

[4 Gap analysis in upstream projects [editor: Carlos] [authors: Gerald, Carlos, Tomi, Ryota] 10](#_Toc411865905)

[4.1 OpenStack 11](#_Toc411865906)

[4.1.1 Ceilometer 11](#_Toc411865907)

[4.1.2 Nova 12](#_Toc411865908)

[4.1.3 Monasca (?) 13](#_Toc411865909)

[4.1.4 Ironic (?) 13](#_Toc411865910)

[4.2 Hardware monitoring tools 13](#_Toc411865911)

[4.2.1 Zabbix 13](#_Toc411865912)

[5 Detailed implementation plan [editor: Ryota] [authors: Gerald, Carlos, Tomi, Ryota] 13](#_Toc411865913)

[5.1 Information elements 13](#_Toc411865914)

[5.2 Detailed northbound interface specification 13](#_Toc411865915)

[5.3 Blueprints 13](#_Toc411865916)

[6 Summary and conclusion [editor: Ashiq] [authors: Gerald, …] 14](#_Toc411865917)

[6.1 Future plan 14](#_Toc411865918)

[7 References and bibliography 14](#_Toc411865919)

# Introduction [editor: Ashiq]

The goal of this project is to build 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 recovery of VNFs on them. Requirement survey and development of missing feature in NFVI and VIM are in scope of this project 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 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]

Problem description

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

* EPC VNFs are often in ACT-SBY configuration and need to switch to SBY as soon as relevant faults are detected in the ACT VNF.
* NFVI encompasses all elements in NFVI, e.g., Physical Machines, Hypervisors, Storage and Network elements.

In addition, VIM e.g. OpenStack needs to receive maintenance instruction from the operator/administrator

* Empty certain Physical Machines (PMs) so that maintenance works could be performed

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

# Use cases and scenarios [editor: Ashiq]

Telecom nodes often has very high requirements on in service performance. As a consequence they often utilize redundancy and high availability mechanisms. The HA support may be built-in or provided by the platform. In any case the HA support typically have 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 ACT node is unable to function properly due to fault or any other reason, the SBY node is instantly made ACT, and service could be provided without any 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 NFVI fault management and maintenance requirements aim at providing fast failure detection of physical and virtualised resources and remediation of the virtualise resources provided to consumers according to their predefined request to enable applications to recover to a fully redundant mode of operation.

There are three use cases to 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. Auto Healing (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

### Auto healing

.

In Fig. 1, a system-wide view of relevant functional blocks is presented. OpenStack is considered as the VIM implementation which has interfaces with the Resource Pool (NFVI in ETSI NFV terminology) and Users/Clients. VNF implementation is represented as VMs with different colours. User/Clients (VNFM or NFVO in ETSI NFV terminology) own/manage the respective VMs shown with the same colours.



Figure 1 - Fault management use case

The first requirement over here is that OpenStack needs to detect faults (1. Fault Notification in Fig. 1) in the Resource Pool which affects the proper functioning of the VMs on top of it. Relevant fault items should be configurable. OpenStack itself could be extended to detect such faults. A third party fault monitoring element can also be used which then informs OpenStack about such faults. However, the third party fault monitoring element would also be a component of VIM from an architectural point of view.

Once such fault is detected, OpenStack shall find out which VMs are affected by this fault. In the example in Fig. 1, VM-4 is affected by a fault in Hardware Server-3. Such mapping shall be maintained in OpenStack e.g. shown as the Server-VM info table in OpenStack in Fig. 1.

Once OpenStack detects which VMs are affected, it then finds out who is the owner/manager of the affected VMs (Step 2 in Fig. 1). In Fig.1, through an Ownership info table, OpenStack knows that for the red VM-4, the manager is the red User/Client. OpenStack then notifies (3. Fault Notification in Fig. 1) red User/Client about this fault, preferably with sufficient abstraction rather than detailed physical fault information.

The User/Client then switches to its SBY configuration and makes the SBY VNF to ACT state. It further initiates a process to instantiate/configure a new SBY. However, switching to SBY and creating a new SBY is a VNFM/NFVO level operation and therefore, outside the scope of this project. Doctor will not provide a dedicated interface for recovery actions.

Once the User/Client has switched to SBY configuration, it notifies (Step 4 “Instruction” in Figure 1) OpenStack. OpenStack 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” in Figure 1).

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

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.

This use case is very similar to Auto healing 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 OpenStack, or from a third party module which notifies OpenStack about an eminent fault. From Step 2~5, it is the same as “Auto healing” use case. In this case, the User/Client of a VM/VNF switches to SBY based on a predicted fault, rather than an occurred fault.

## 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, User/Client of VMs needs to know that such resources WILL be unavailable because of NFVI maintenance. This 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 VMs are necessary and a pause in vCPU is not allowed. The User/Client is in a position who can perform the switch between ACT and SBY, or switch to an alternative VNF forwarding graph so the hardware servers hosting the ACT VMs can be emptied for the upcoming maintenance operation. Once the target hardware servers are emptied (i.e. no VM running on top), the VIM/OpenStack can mark them with an appropriate flag so that these servers are not considered for VM hosting until these are flagged.

A high-level view of the maintenance procedure is presented in Fig. 2. VIM/OpenStack, through its northbound interface, receives a maintenance notification (1. Maintenance Instruction in Fig. 2) from the Administrator (e.g. a network operator) who mentions which hardware is subject to such maintenance. Maintenance operation includes replacement/upgrade of hardware, update/upgrade of the hypervisor/host OS etc.

The consequent steps to enable User/Client perform ACT-SBY switching are very similar to the fault management scenario. From VIM/OpenStacks internal database, it finds out which VMs are running on those particular Hardware Servers and who are the managers of those VMs (Step 2 in Fig. 2). The VIM/OpenStack then informs the respective Users/Clients (VNFMs or NFVO) (3. Maintenance Notification in Fig. 2). The Users/Clients then take necessary actions (e.g. switch to SBY, switch VNF forwarding graphs) and then notifies (4. Instruction in Fig. 2). Upon receiving such notification, the VIM/OpenStack takes necessary actions (5. Execute Instruction in Fig. 2) to empty the Hardware Servers so that consequent maintenance operation could be performed. Due to such similarity for Step 2~5, the maintenance procedure and the fault management procedure are kept 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: failures of virtualized resources and 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 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 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. ongoing session might be lost) while keeping availability (establishment or re-establishment of sessions are not affected), while 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 but also events like addition or removal of VMs due to scaling out/in, change of CM characteristics due to scaling up/down and SW upgrades. Some of these might have been requested by the application orits 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 application management organization and 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 functionality 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.

#### Correlation

The VIM shall correlate each fault to the impacted virtual resource and make the alarm available over the north bound interface for consumption by those impacted by 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.

### 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 a high level architecture of NFV focusing \*Virtualized Infrastructure\*.

\*Virtualized Infrastructure\* provides virtual resources, such as virtual machine (VM) and virtual network. Those virtual resources are used to run \*applications\* that could be component of a network service which is managed by user of NFVI. \*Virtualized Infrastructure Manager\* provides functionalities of controlling and viewing virtual resources on hardware (physical) resources to users and administrators. OpenStack is a prominent candidate for this \*Virtualized Infrastructure Manager\*. Administrator may control Virtualized Infrastructure without \*Virtualized Infrastructure Manager\*.



Figure 3 - High level architecture

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

## General Features

The following features are required to 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 error in virtual resources on them.

Unavailability of physical resource is detected by various monitoring and managing tools for H/W and S/W components.

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

The configuration shall enable Failure Selection and Aggregation.

Failure aggregation means VIM 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 failure on hardware monitoring tools.
* Future unavailability can be found by receiving maintenance instruction issued by administrator of the physical resource pool.

### Cognition

VIM should identify unavailability of virtualized resources that is or will be affected by failure on physical resource under them.

Unavailability of virtualized resource is found by referring the map of physical and virtualized resource. The cause of unavailability of virtualized resource could be different in technologies and policies of deployment, so the relation from physical resource to virtualized resource shall be configurable.

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

### Notification

There are two types of notification; event of virtualized resource and update of capacity of Resource Pool.

VIM has to notify unavailability of virtual resources to the user who owns it.

VIM also needs to notify unavailability of physical resources to administrator.

All notification should be transferred immediately to minimize network service stall and to avoid over assignment caused by delay of capability update.

There are multiple users who have receiver of notification on different location.

So VIM has to find out owner.

Moreover, there would be large number of virtual and physical resources in the real deployment, so polling state to VIM leads heavy traffic.

Thus, Pub/Sub Messaging Model is better for this notification, since it sends notification to owners who requested.

Note, VIM should accept individual notification URL for each resources only by its owner or administrator.

Event of virtualized resource is description of unavailability to inform the user.

Flexibility of notification is important; receiver function in user-side implementation could have different schema, location and policy (receive or not, aggregate events in the same cause, etc.).

### Recovery Action

VIM is required to execute actions to process fault recovery and maintenance.

All actions, done by VIM and NFVI after those notifications, should be instructed by the owner of resources or administrator.

Instructions are not always required after those notifications.

(Option) For prompt recovery, VIM can have feature to automate recovery actions.

A delegated action could automatically proceed if it already instructed: e.g. VIM can automatically evacuate VM which is labelled like ‘allow live-migration’ by the owner.

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

### Fault management



Figure 6 - High-level message flow for fault management

Step 1: Fault detection/notification

Step 2: Fault/event correlation and aggregation in VIM; find affected virtual resources

Step 3: Fault notification to user-side manager and decision on appropriate action to resolve the fault, e.g. switch to hot standby or live migration of the affected virtual resource

Step 4: Instructions to VIM requesting certain actions to be performed

Step 5: VIM is executing the requested action, e.g. it will migrate or terminate a virtual resource.

### NFVI Maintenance



Step 1: Maintenance trigger received from OAM

Step 2: Find affected virtual resources

Step 3: Fault notification to user-side manager and decision on appropriate action to resolve the fault, e.g. cold / live migration of the affected virtual resource

Step 4: Instructions to VIM requesting certain actions to be performed

Step 5: VIM is executing the requested action, e.g. it will migrate or terminate a virtual resource.

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

## 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.

## 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.

#### 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%22%20%5Cl%20%22alarms)
	+ 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**
* ...

#### Scalability of fault aggregation

* **Category**: Ceilometer
* **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

* **Category**: Ceilometer
* **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.

* **Category**: Nova
* **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](https://blueprints.launchpad.net/nova/%2Bspec/fencing)

#### Evacuate VMs on Maintenance mode

* **Category**: Nova
* **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 solution that integrates with OpenStack. Even though it’s still in its early days, it is of 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

* **Category:** Monasca
* **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

#### ItemName: Sensor monitoring

* **Category:** Monasca
* **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.

### Ironic (?)

## Hardware monitoring tools

### Zabbix

#### Delay in execution of actions

* **Category:** Zabbix
* **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

## Others

### VIM Southbound interface

#### Normalization of data collection models

* **Category**: VIM Southbound interface
* **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**:
* ...

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

## 5.1 Functional Blocks



Figure 4 - Functional blocks

### Monitor

Monitor has responsibility for monitoring Virtualized Infrastructure.

There are many existing tools and services to monitor H/W and S/W such as Zabbix.

### Inspector

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

Inspector has drivers for different types of events and resources to integrate any types of monitor and controller.

Inspector load failure policy which instruct failure selection and aggregation from raw events.

This failure policy is configured by administrator.

The reason for separation of Inspector and Controller is to make Controller focus on simple operation by avoiding tight integration of various health check mechanisms.

### Controller

Controller has responsibility for resource map, ability to accept update request for resource state (exposing as provider API) and notifies all events regarding virtual resources to Notifier.

Optionally, Controller has ability to poison state of virtual resources when received failure of mapped physical resource by Inspector.

Controller also calculates capability of Resource Pool when failure of physical resource received and notifies the update as event to Notifier.

VIM may have several controllers for each resource types such as Nova, Neutron and Cinder in OpenStack.

Each controller has database of virtual and physical resource which shall be master information in VIM.

### Notifier

Notifier has ability to register alarm regarding virtual resource with subscribe method such as API endpoint of User-side and Admin-side Manager, and to notify events by refering alarm configuration when it has received events from Controllers.

Notifier focus on selecting and aggregating failure events based on user configuration.

## Sequence



Figure 5 - Fault management scenario

## Information elements

## Detailed northbound interface specification

Interface specification: methods, inputs, outputs, … (should not be too related to OpenStack)

Describe entities , e.g. in JSON

## Blueprints

# 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. ETSI NFV GS…