# AUTO

***Release Latest***

**May 09, 2019**

# Contents

CHAPTER 1

# OPNFV Auto (ONAP-Automated OPNFV) Configuration Guide

## 1.1 Introduction

This document describes the software and hardware reference frameworks used by Auto, and provides guidelines on how to perform configurations and additional installations.

## 1.2 Goal

The goal of *Auto*

installation and configuration is to prepare an environment where the *Auto use cases*

can be assessed, i.e. where the corresponding test cases can be executed and their results can be collected for analysis. See the *Auto Release Notes <auto-releasenotes>*

for a discussion of the test results analysis loop.

An instance of ONAP needs to be present, as well as a number of deployed VNFs, in the scope of the use cases. Simulated traffic needs to be generated, and then test cases can be executed. There are multiple parameters to the Auto environment, and the same set of test cases will be executed on each environment, so as to be able to evaluate the influence of each environment parameter.

The initial Auto use cases cover:

- **Edge Cloud** (increased autonomy and automation for managing Edge VNFs)
- **Resilience Improvements through ONAP** (reduced recovery time for VNFs and end-to-end services in case of failure or suboptimal performance)
- **Enterprise vCPE** (automation, cost optimization, and performance assurance of enterprise connectivity to Data Centers and the Internet)

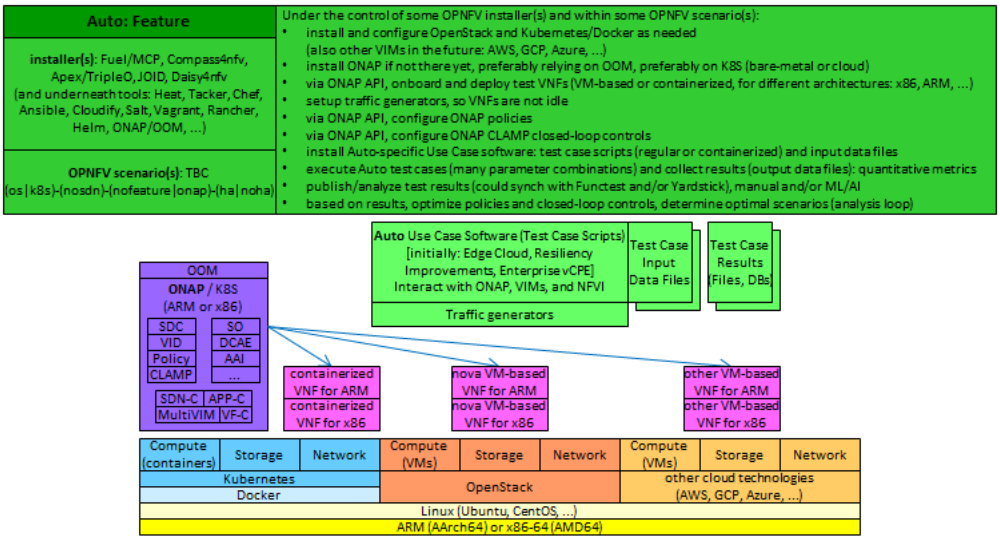
The general idea of the Auto feature configuration is to install an OPNFV environment (comprising at least one Cloud Manager), an ONAP instance, ONAP-deployed VNFs as required by use cases, possibly additional cloud managers

not already installed during the OPNFV environment setup, traffic generators, and the Auto-specific software for the use cases (which can include test frameworks such as Robot or Functest

The ONAP instance needs to be configured with policies and closed-loop controls (also as required by use cases), and the test framework controls the execution and result collection of all the test cases. Then, test case execution results can be analyzed, so as to fine-tune policies and closed-loop controls, and to compare environment parameters.

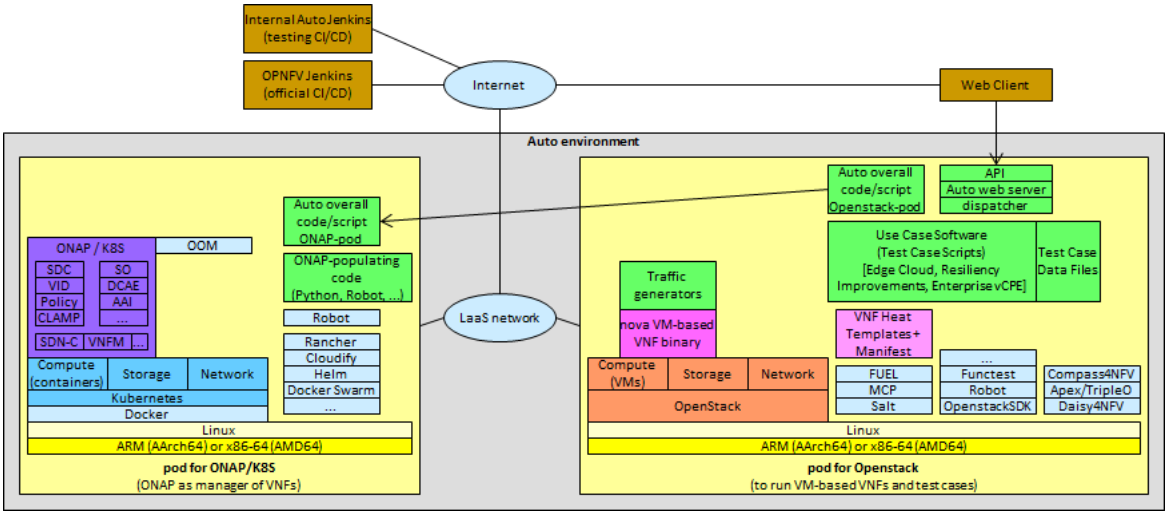
The following diagram illustrates execution environments, for x86 architectures and for Arm architectures, and other environment parameters (see the Release Notes for a more detailed discussion on the parameters). The installation process depends on the underlying architecture, since certain components may require a specific binary-compatible version for a given x86 or Arm architecture. The preferred variant of ONAP is one that runs on Kubernetes, while all VNF types are of interest to Auto: VM-based or containerized (on any cloud manager), for x86 or for Arm. In fact, even PNFs could be considered, to support the evaluation of hybrid PNF/VNF transition deployments (ONAP has the ability of also managing legacy PNFs).

The initial VM-based VNFs will cover OpenStack, and in future Auto releases, additional cloud managers will be considered. The configuration of ONAP and of test cases should not depend on the underlying architecture and infrastructure.

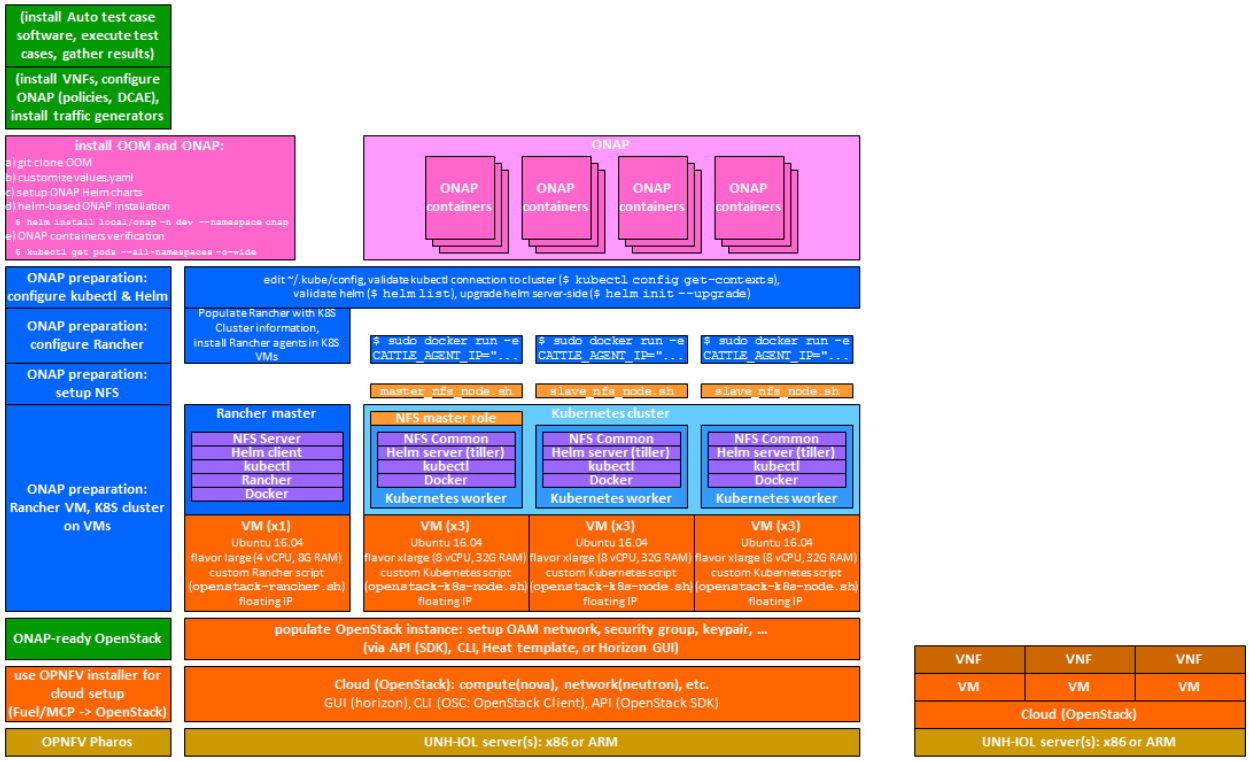



For each component, various installer tools will be considered (as environment parameters), so as to enable comparison, as well as ready-to-use setups for Auto end-users. For example, the most natural installer for ONAP would be OOM (ONAP Operations Manager). For the OPNFV infrastructure, supported installer projects will be used: Fuel/MCP, Compass4NFV, Apex/TripleO, Daisy4NFV. Note that JOID was last supported in OPNFV Fraser 6.2, and is not supported anymore as of Gambia 7.0.

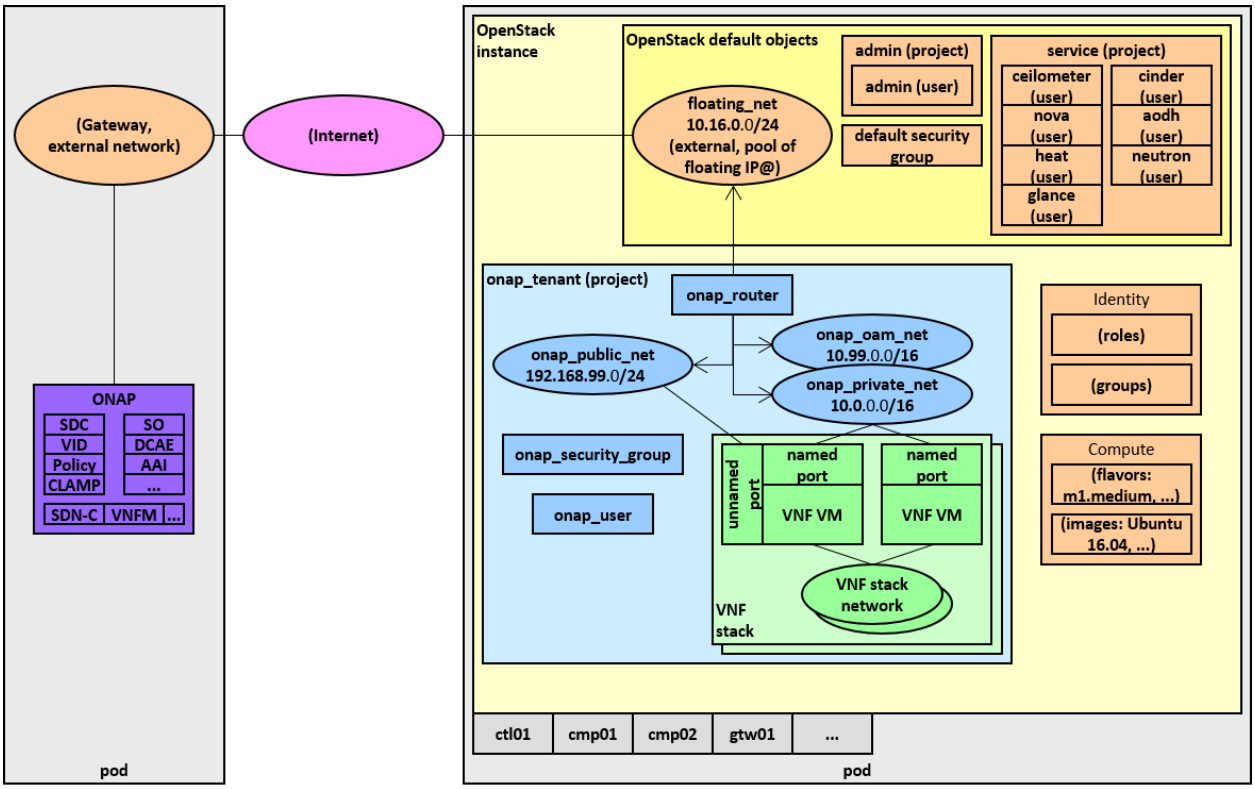
The initial version of Auto will focus on OpenStack VM-based VNFs, onboarded and deployed via ONAP API (not by ONAP GUI, for the purpose of automation). ONAP is installed on Kubernetes. Two or more servers from LaaS are used: one or more to support an OpenStack instance as provided by the OPNFV installation via Fuel/MCP or other OPNFV installers (Compass4NFV, Apex/TripleO, Daisy4NFV), and the other(s) to support ONAP with Kubernetes and Docker. Therefore, the VNF execution environment is composed of the server(s) with the OpenStack instance(s). Initial tests will also include ONAP instances installed on bare-metal servers (i.e. not directly on an OPNFV infrastructure; the ONAP/OPNFV integration can start at the VNF environment level; but ultimately, ONAP should be installed within an OPNFV infrastructure, for full integration).




ONAP/K8S has several variants. The initial variant considered by Auto is the basic one recommended by ONAP, which relies on the Rancher installer and on OpenStack VMs providing VMs for the Rancher master and for the Kubernetes cluster workers, as illustrated below for ONAP-Beijing release:




The OpenStack instance running VNFs may need to be configured as per ONAP expectations, for example creating instances of ONAP projects/tenants, users, security groups, networks (private, public), connected to the Internet by a Router, and making sure expected VM images and flavors are present. A script (using OpenStack SDK, or OpenStack CLI, or even OpenStack Heat templates) would populate the OpenStack instance, as illustrated below:

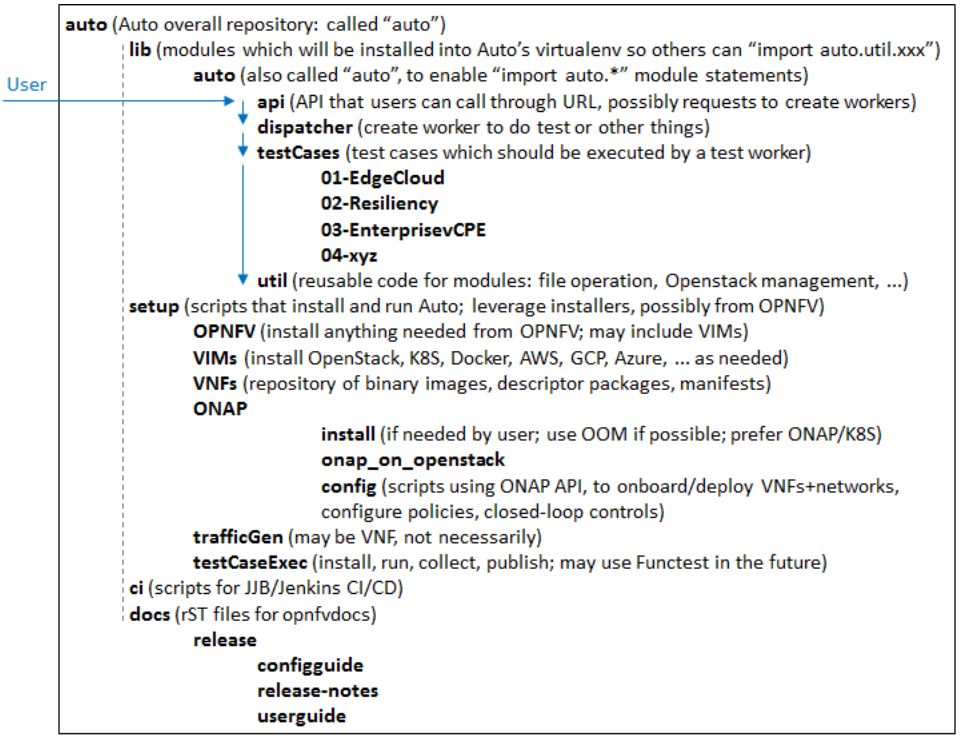



That script can also delete these created objects, so it can be used in tear-down procedures as well (use -del or –delete option). It is located in the Auto repository , under the setup/VIMs/OpenStack directory:

- auto_script_config_openstack_for_onap.py

Jenkins (or more precisely JJB: Jenkins Job Builder) will be used for Continuous Integration in OPNFV releases, to ensure that the latest master branch of Auto is always working. The first 3 tasks in the pipeline would be: install OpenStack instance via an OPNFV installer (Fuel/MCP, Compass4NFV, Apex/TripleO, Daisy4NFV), configure the OpenStack instance for ONAP, install ONAP (using the OpenStack instance network IDs in the ONAP YAML file).

Moreover, Auto will offer an API, which can be imported as a module, and can be accessed for example by a web application. The following diagram shows the planned structure for the Auto Git repository, supporting this module, as well as the installation scripts, test case software, utilities, and documentation.




## 1.3 Pre-configuration activities

The following resources will be required for the initial version of Auto:

- at least two LaaS (OPNFV Lab-as-a-Service) pods (or equivalent in another lab), with their associated network information. Later, other types of target pods will be supported, such as clusters (physical bare-metal or virtual). The pods can be either x86 or Arm CPU architectures. An effort is currently ongoing (ONAP Integration team, and Auto team), to ensure Arm binaries are available for all ONAP components in the official ONAP Docker registry.
- the Auto Git repository (clone from Gerrit Auto)

## 1.4 Hardware configuration

ONAP needs relatively large servers (at least 512G RAM, 1TB storage, 80-100 CPU threads). Initial deployment attempts on single servers did not complete. Current attempts use 3-server clusters, on bare-metal.

For initial VNF deployment environments, virtual deployments by OPNFV installers on a single server should suffice. Later, if many large VNFs are deployed for the Auto test cases, and if heavy traffic is generated, more servers might be necessary. Also, if many environment parameters are considered, full executions of all test cases on all environment configurations could take a long time, so parallel executions of independent test case batches on multiple sets of servers and clusters might be considered.

## 1.5 Feature configuration

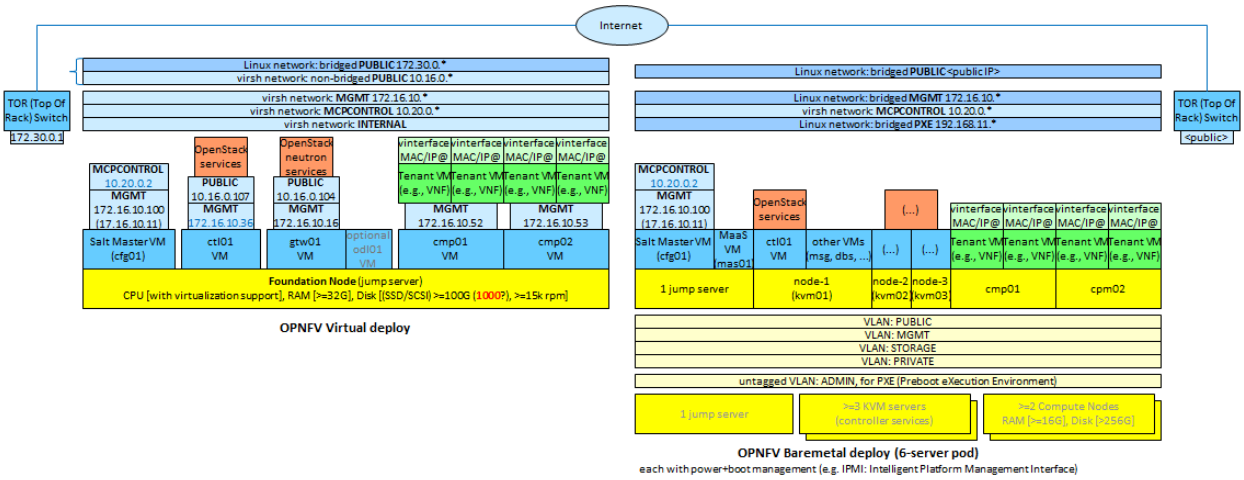
### 1.5.1 Environment installation

Current Auto work in progress is captured in the Auto Lab Deployment wiki page.

#### OPNFV with OpenStack

The first Auto installation used the Fuel/MCP installer for the OPNFV environment (see the OPNFV download page).

The following figure summarizes the two installation cases for Fuel: virtual or bare-metal. This OPNFV installer starts with installing a Salt Master, which then configures subnets and bridges, and install VMs (e.g., for controllers and compute nodes) and an OpenStack instance with predefined credentials.




The Auto version of OPNFV installation configures additional resources for the OpenStack virtual pod (more virtual CPUs and more RAM), as compared to the default installation. Examples of manual steps are as follows:

```
1. mkdir /opt/fuel
2. cd /opt/fuel
3. git clone https://git.opnfv.org/fuel
4. cd fuel
5. vi /opt/fuel/fuel/mcp/config/scenario/os-nosdn-nofeature-noha.yaml
```

These lines can be added to configure more resources:

```
   gtw01:
     ram: 2048
+  cmp01:
+    vcpus: 32
+    ram: 196608
+  cmp02:
+    vcpus: 32
+    ram: 196608
```

The final steps deploy OpenStack (duration: approximately between 30 and 45 minutes).

```
# The following change will provide more space to VMs. Default is 100G per cmp0x.␣
↪This gives 350 each and 700 total.
6. sed -i mcp/scripts/lib.sh -e 's/\(qemu-img create.*\) 100G/\1 350G/g'
```

(continues on next page)

(continued from previous page)

```
# Then deploy OpenStack. It should take between 30 and 45 minutes:
7. ci/deploy.sh -l UNH-LaaS -p virtual1 -s os-nosdn-nofeature-noha -D |& tee deploy.
↪log

# Lastly, to get access to the extra RAM and vCPUs, adjust the quotas (done on the␣
↪controller at 172.16.10.36):
8. openstack quota set --cores 64 admin
9. openstack quota set --ram 393216 admin
```

Note:

- with Linux Kernel 4.4, the installation of OPNFV is not working properly (seems to be a known bug of 4.4, as it works correctly with 4.13): neither qemu-nbd nor kpartx are able to correctly create a mapping to /dev/nbd0p1 partition in order to resize it to 3G (see Fuel repository, file mcp/scripts/lib.sh , function mount_image).
- it is not a big deal in case of x86, because it is still possible to update the image and complete the installation even with the original partition size.
- however, in the case of ARM, the OPNFV installation will fail, because there isn't enough space to install all required packages into the cloud image.

Using the above as starting point, Auto-specific scripts have been developed, for each of the 4 OPNFV installers Fuel/MCP, Compass4NFV, Apex/TripleO, Daisy4NFV. Instructions for virtual deployments from each of these installers have been used, and sometimes expanded and clarified (missing details or steps from the instructions). They can be found in the Auto repository , under the ci directory:

- deploy-opnfv-fuel-ubuntu.sh
- deploy-opnfv-compass-ubuntu.sh
- deploy-opnfv-apex-centos.sh
- deploy-opnfv-daisy-centos.sh

### ONAP on Kubernetes

An ONAP installation on OpenStack has also been investigated, but we focus here on the ONAP on Kubernetes version.

The initial focus is on x86 architectures. The ONAP DCAE component for a while was not operational on Kubernetes (with ONAP Amsterdam), and had to be installed separately on OpenStack. So the ONAP instance was a hybrid, with all components except DCAE running on Kubernetes, and DCAE running separately on OpenStack. Starting with ONAP Beijing, DCAE also runs on Kubernetes.

For Arm architectures, specialized Docker images are being developed to provide Arm architecture binary compatibility. See the *Auto Release Notes <auto-releasenotes>* for more details on the availability status of these Arm images in the ONAP Docker registry.

The ONAP reference for this installation is detailed here.

Examples of manual steps for the deploy procedure are as follows:

```
1  git clone https://gerrit.onap.org/r/oom
2  cd oom
3  git pull https://gerrit.onap.org/r/oom refs/changes/19/32019/6
4  cd install/rancher
5  ./oom_rancher_setup.sh -b master -s <your external ip> -e onap
6  cd oom/kubernetes/config
```

(continues on next page)

(continued from previous page)

```
7  (modify onap-parameters.yaml for VIM connection (manual))
8  ./createConfig.sh -n onap
9  cd ../oneclick
10 ./createAll.bash -n onap
```

Several automation efforts to integrate the ONAP installation in Auto CI are in progress. One effort involves using a 3-server cluster at OPNFV Pharos LaaS (Lab-as-a-Service). The script is available in the Auto repository , under the ci directory:

```
* deploy-onap.sh
```

### 1.5.2 ONAP configuration

This section describes the logical steps performed by the Auto scripts to prepare ONAP and VNFs.

#### VNF deployment

<TBC; pre-onboarding, onboarding, deployment>

#### Policy and closed-loop control configuration

<TBC>

### 1.5.3 Traffic Generator configuration

<TBC>

### 1.5.4 Test Case software installation and execution control

<TBC; mention the management of multiple environments (characterized by their parameters), execution of all test cases in each environment, only a subset in official OPNFV CI/CD Jenkins due to size and time limits; then posting and analysis of results; failures lead to bug-fixing, successes lead to analysis for comparisons and fine-tuning>

## 1.6 Installation health-check

<TBC; the Auto installation will self-check, but indicate here manual steps to double-check that the installation was successful>

## 1.7 References

Auto Wiki pages:

- Auto wiki main page
- Auto Lab Deployment wiki page

OPNFV documentation on Auto:

- *Auto Release Notes <release-notes>*
- *Auto use case user guides <auto-userguide>*

Git&Gerrit Auto repositories:

- Auto Git repository
- Gerrit for Auto project

## 1.8 Auto Post Installation Procedure

<TBC; normally, the installation is self-contained and there should be no need for post-installation manual steps; possibly input for CI toolchain and deployment pipeline in first section>

### 1.8.1 Automated post installation activities

<TBC if needed>

### 1.8.2 <Project> post configuration procedures

<TBC if needed>

### 1.8.3 Platform components validation

<TBC if needed>

CHAPTER 2

# OPNFV Auto (ONAP-Automated OPNFV) User Guide

## 2.1 Auto User Guide: Use Case 1 Edge Cloud

This document provides the user guide for Fraser release of Auto, specifically for Use Case 1: Edge Cloud.

### 2.1.1 Description

This use case aims at showcasing the benefits of using ONAP for autonomous Edge Cloud management.

A high level of automation of VNF lifecycle event handling after launch is enabled by ONAP policies and closed-loop controls, which take care of most lifecycle events (start, stop, scale up/down/in/out, recovery/migration for HA) as well as their monitoring and SLA management.

Multiple types of VNFs, for different execution environments, are first approved in the catalog thanks to the onboarding process, and then can be deployed and handled by multiple controllers in a systematic way.

This results in management efficiency (lower control/automation overhead) and high degree of autonomy.

Preconditions:

1. hardware environment in which Edge cloud may be deployed
2. an Edge cloud has been deployed and is ready for operation
3. ONAP has been deployed onto a Cloud, and is interfaced (i.e. provisioned for API access) to the Edge cloud

Main Success Scenarios:

- lifecycle management - start, stop, scale (dependent upon telemetry)
- recovering from faults (detect, determine appropriate response, act); i.e. exercise closed-loop policy engine in ONAP
  - verify mechanics of control plane interaction
- collection of telemetry for machine learning

Details on the test cases corresponding to this use case:

- Environment check
  - Basic environment check: Create test script to check basic VIM (OpenStack), ONAP, and VNF(s) are up and running
- VNF lifecycle management
  - VNF Instance Management: Validation of VNF Instance Management which includes VNF instantiation, VNF State Management and termination
  - Tacker Monitoring Driver (VNFMonitorPing):
    - Write Tacker Monitor driver to handle monitor_call and, based on return state value, create custom events
    - If Ping to VNF fails, trigger below events
      - Event 1 : Collect failure logs from VNF
      - Event 2 : Soft restart/respawn the VNF
  - Integrate with Telemetry
    - Create TOSCA template policies to implement ceilometer data collection service
    - Collect CPU utilization data, compare with threshold, and perform action accordingly (respawn, scale-in/scale-out)

### 2.1.2 Test execution high-level description

<TBC>

## 2.2 Auto User Guide: Use Case 2 Resiliency Improvements Through ONAP

This document provides the user guide for Fraser release of Auto, specifically for Use Case 2: Resiliency Improvements Through ONAP.

### 2.2.1 Description

This use case illustrates VNF failure recovery time reduction with ONAP, thanks to its automated monitoring and management. It:

- simulates an underlying problem (failure, stress, or any adverse condition in the network that can impact VNFs)
- tracks a VNF
- measures the amount of time it takes for ONAP to restore the VNF functionality.

The benefit for NFV edge service providers is to assess what degree of added VIM+NFVI platform resilience for VNFs is obtained by leveraging ONAP closed-loop control, vs. VIM+NFVI self-managed resilience (which may not be aware of the VNF or the corresponding end-to-end Service, but only of underlying resources such as VMs and servers).

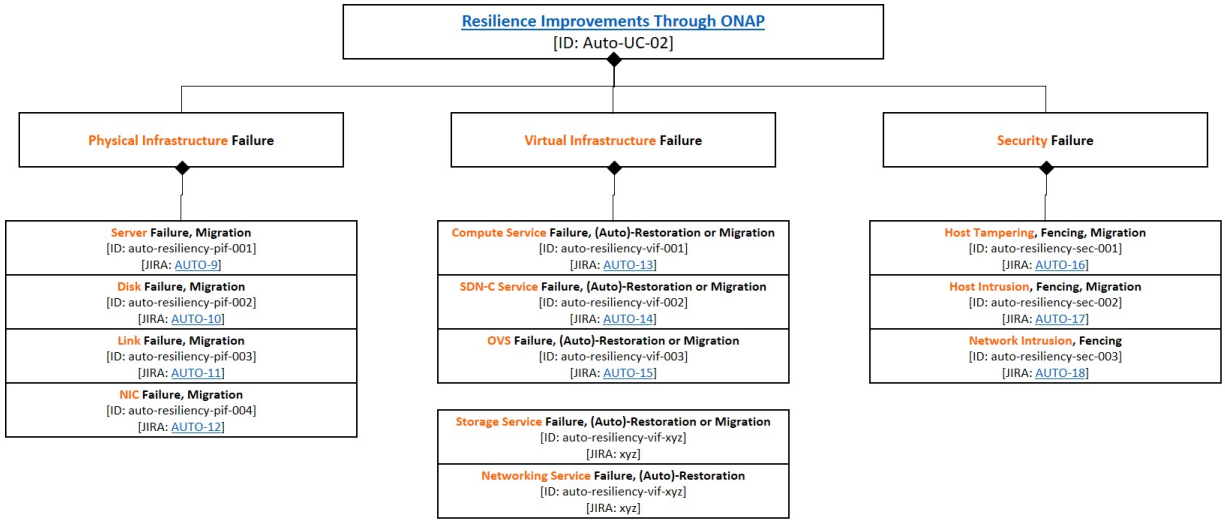
Also, a problem, or challenge, may not necessarily be a failure (which could also be recovered by other layers): it could be an issue leading to suboptimal performance, without failure. A VNF management layer as provided by ONAP may detect such non-failure problems, and provide a recovery solution which no other layer could provide in a given deployment.

Preconditions:

1. hardware environment in which Edge cloud may be deployed
2. Edge cloud has been deployed and is ready for operation
3. ONAP has been deployed onto a cloud and is interfaced (i.e. provisioned for API access) to the Edge cloud
4. Components of ONAP have been deployed on the Edge cloud as necessary for specific test objectives

In future releases, Auto Use cases will also include the deployment of ONAP (if not already installed), the deployment of test VNFs (pre-existing VNFs in pre-existing ONAP can be used in the test as well), the configuration of ONAP for monitoring these VNFs (policies, CLAMP, DCAE), in addition to the test scripts which simulate a problem and measures recovery time.

Different types of problems can be simulated, hence the identification of multiple test cases corresponding to this use case, as illustrated in this diagram:




Description of simulated problems/challenges, leading to various test cases:

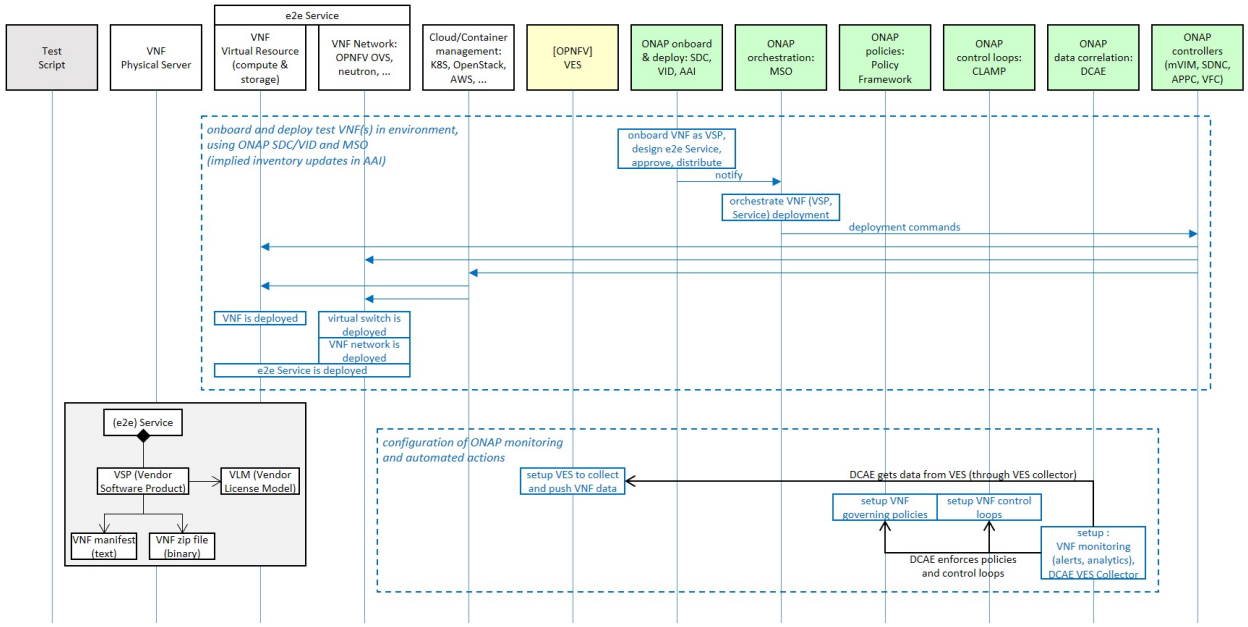
- Physical Infra Failure
  - Migration upon host failure: Compute host power is interrupted, and affected workloads are migrated to other available hosts.
  - Migration upon disk failure: Disk volumes are unmounted, and affected workloads are migrated to other available hosts.
  - Migration upon link failure: Traffic on links is interrupted/corrupted, and affected workloads are migrated to other available hosts.
  - Migration upon NIC failure: NIC ports are disabled by host commands, and affected workloads are migrated to other available hosts.
- Virtual Infra Failure
  - OpenStack compute host service fail: Core OpenStack service processes on compute hosts are terminated, and auto-restored, or affected workloads are migrated to other available hosts.
  - SDNC service fail: Core SDNC service processes are terminated, and auto-restored.
  - OVS fail: OVS bridges are disabled, and affected workloads are migrated to other available hosts.
  - etc.

- Security
  - Host tampering: Host tampering is detected, the host is fenced, and affected workloads are migrated to other available hosts.
  - Host intrusion: Host intrusion attempts are detected, an offending workload, device, or flow is identified and fenced, and as needed affected workloads are migrated to other available hosts.
  - Network intrusion: Network intrusion attempts are detected, and an offending flow is identified and fenced.

### 2.2.2 Test execution high-level description

The following two MSCs (Message Sequence Charts) show the actors and high-level interactions.

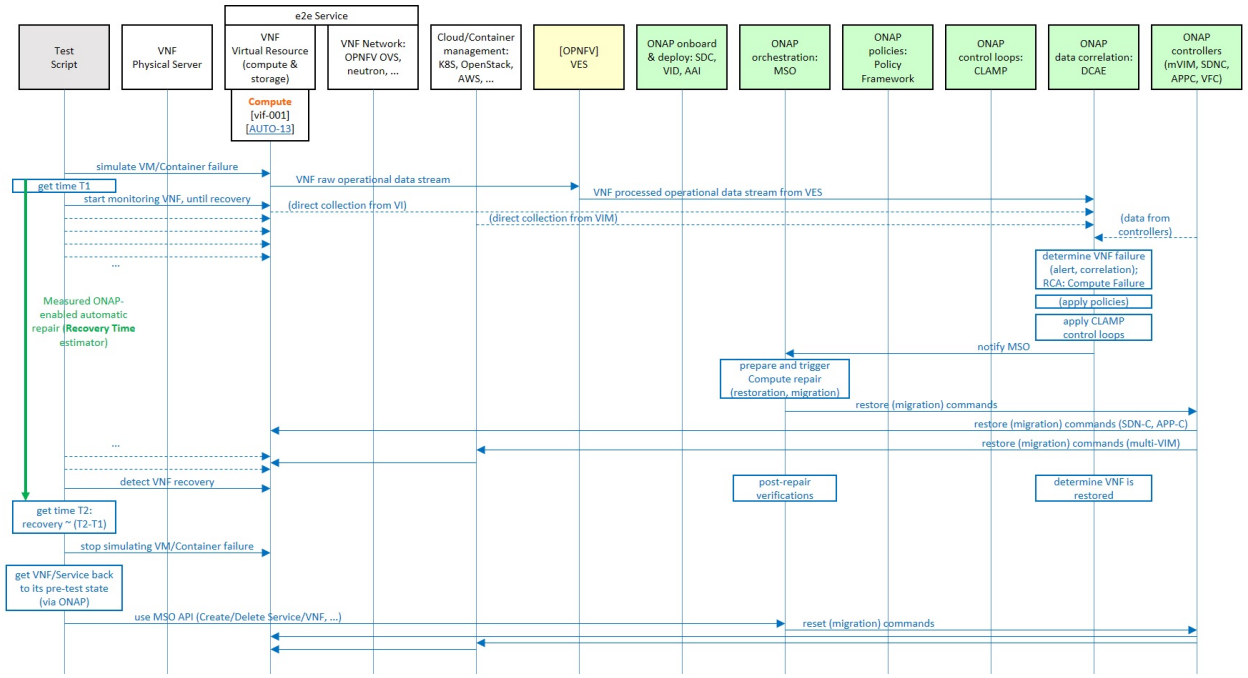
The first MSC shows the preparation activities (assuming the hardware, network, cloud, and ONAP have already been installed): onboarding and deployment of VNFs (via ONAP portal and modules in sequence: SDC, VID, SO), and ONAP configuration (policy framework, closed-loops in CLAMP, activation of DCAE).




The second MSC illustrates the pattern of all test cases for the Resiliency Improvements:

- simulate the chosen problem (a.k.a. a “Challenge”) for this test case, for example suspend a VM which may be used by a VNF
- start tracking the target VNF of this test case
- measure the ONAP-orchestrated VNF Recovery Time
- then the test stops simulating the problem (for example: resume the VM that was suspended)

In parallel, the MSC also shows the sequence of events happening in ONAP, thanks to its configuration to provide Service Assurance for the VNF.



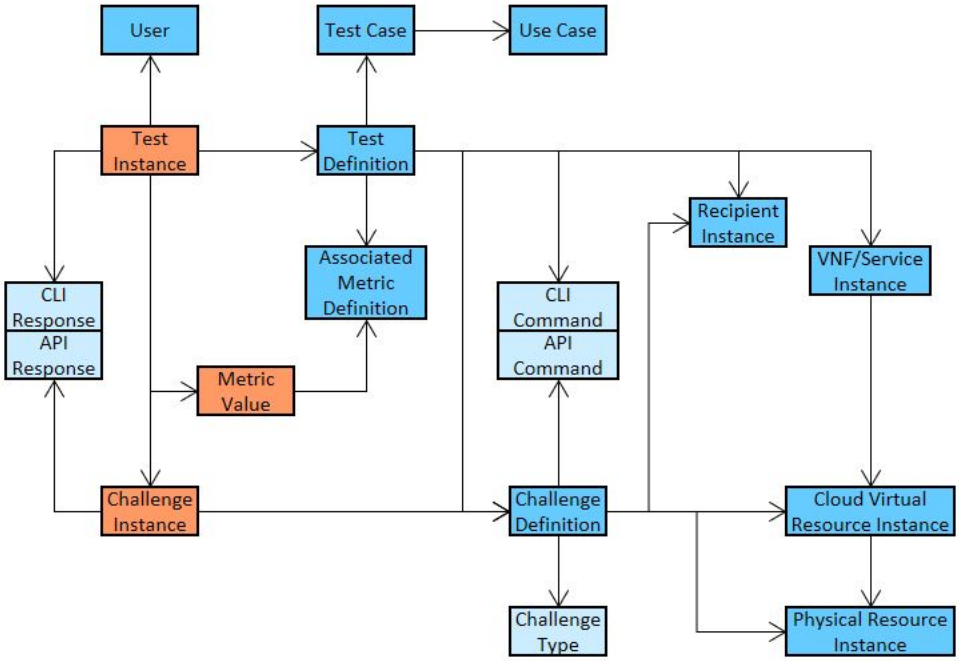

### 2.2.3 Test design: data model, implementation modules

The high-level design of classes identifies several entities, described as follows:

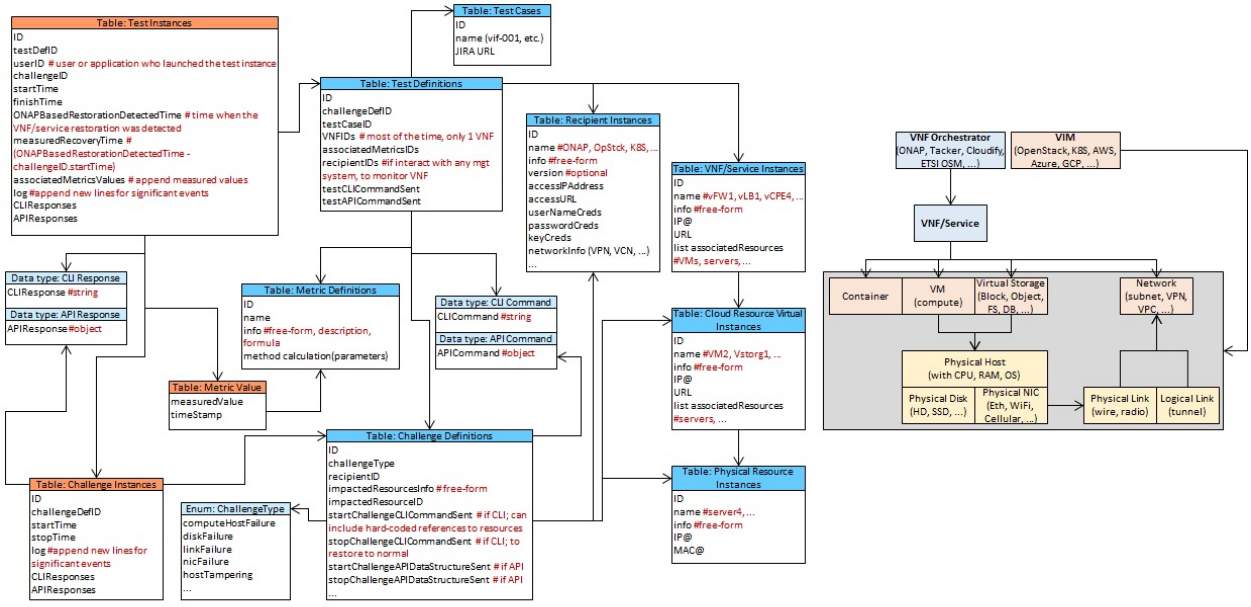
- `Test Case` : as identified above, each is a special case of the overall use case (e.g., categorized by challenge type)
- `Test Definition` : gathers all the information necessary to run a certain test case
- `Metric Definition` : describes a certain metric that may be measured for a Test Case, in addition to Recovery Time
- `Challenge Definition` : describe the challenge (problem, failure, stress, . . . ) simulated by the test case
- `Recipient` : entity that can receive commands and send responses, and that is queried by the Test Definition or Challenge Definition (a recipient would be typically a management service, with interfaces (CLI or API) for clients to query)
- `Resources` : with 3 types (VNF, cloud virtual resource such as a VM, physical resource such as a server)

Three of these entities have execution-time corresponding classes:

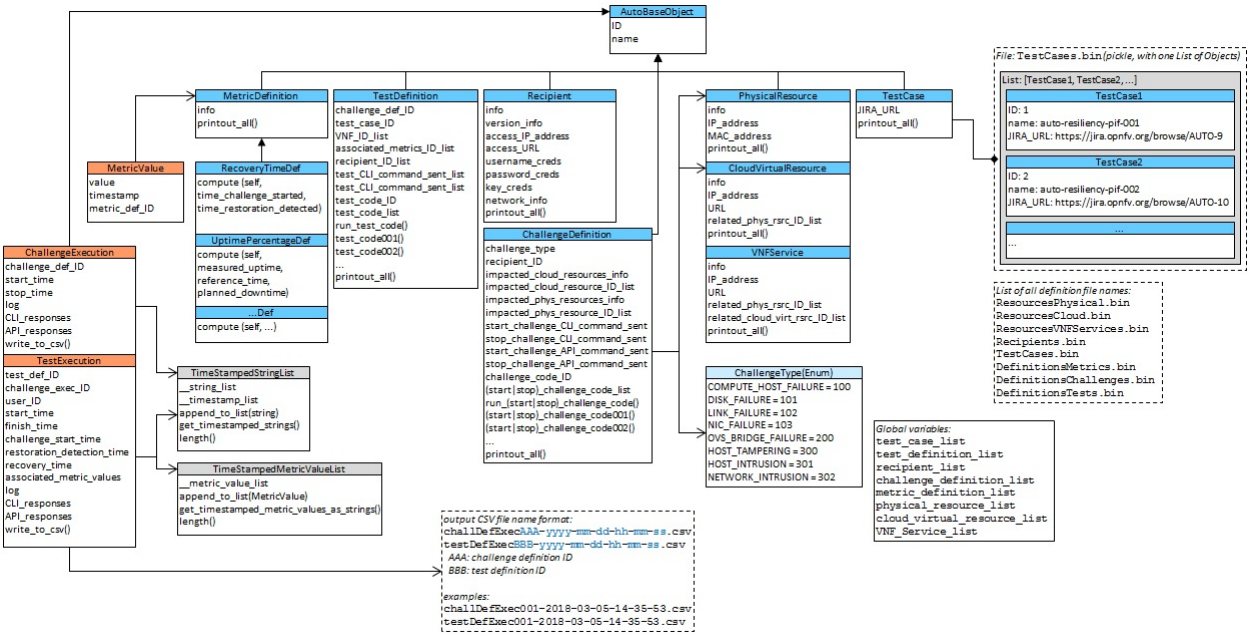
- `Test Execution` , which captures all the relevant data of the execution of a Test Definition
- `Challenge Execution` , which captures all the relevant data of the execution of a Challenge Definition
- `Metric Value` , which captures the quantitative measurement of a Metric Definition (with a timestamp)




The following diagram illustrates an implementation-independent design of the attributes of these entities:

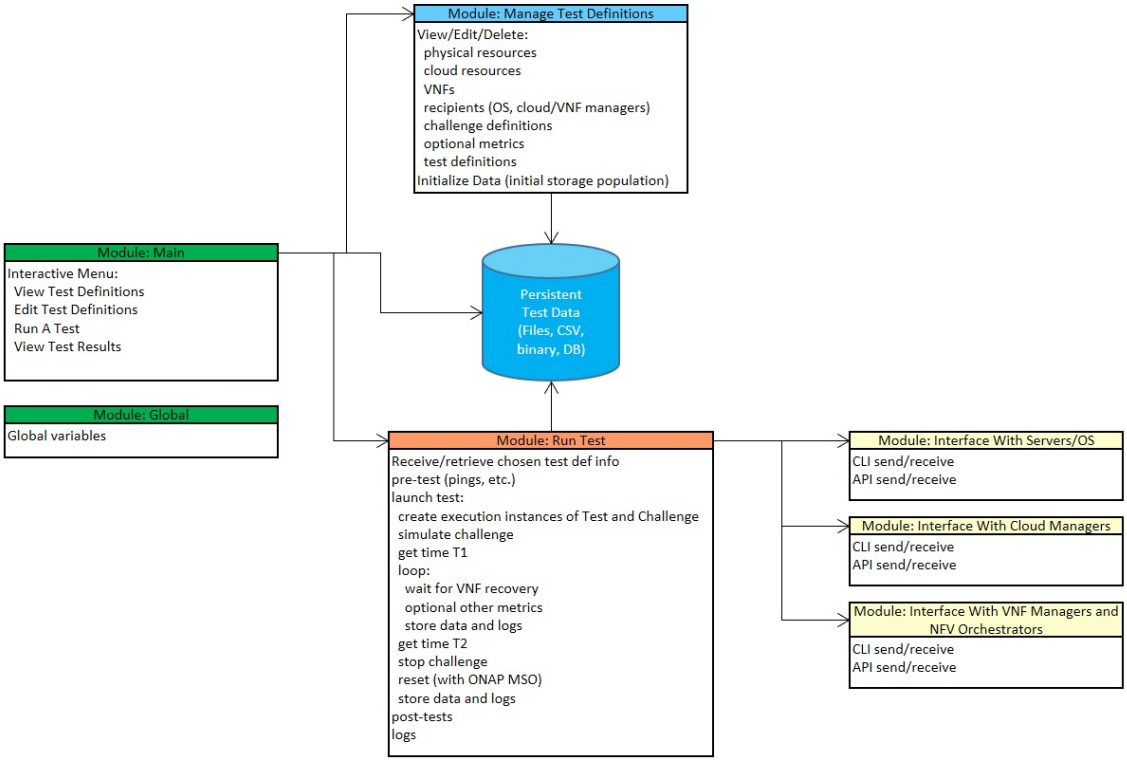



This next diagram shows the Python classes and attributes, as implemented by this Use Case (for all test cases):

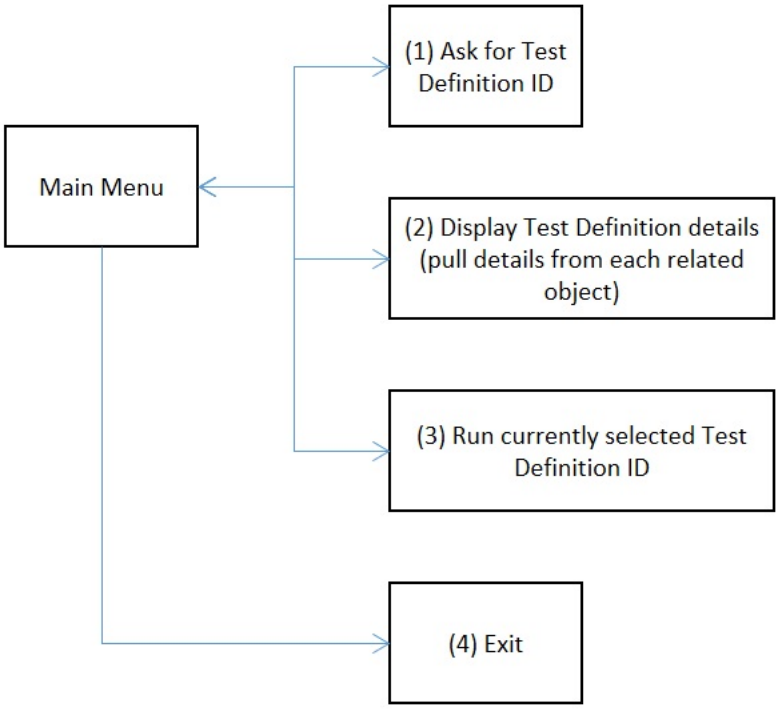



Test definition data is stored in serialization files (Python pickles), while test execution data is stored in CSV files, for easier post-analysis.

The module design is straightforward: functions and classes for managing data, for interfacing with recipients, for executing tests, and for interacting with the test user (choosing a Test Definition, showing the details of a Test Definition, starting the execution).



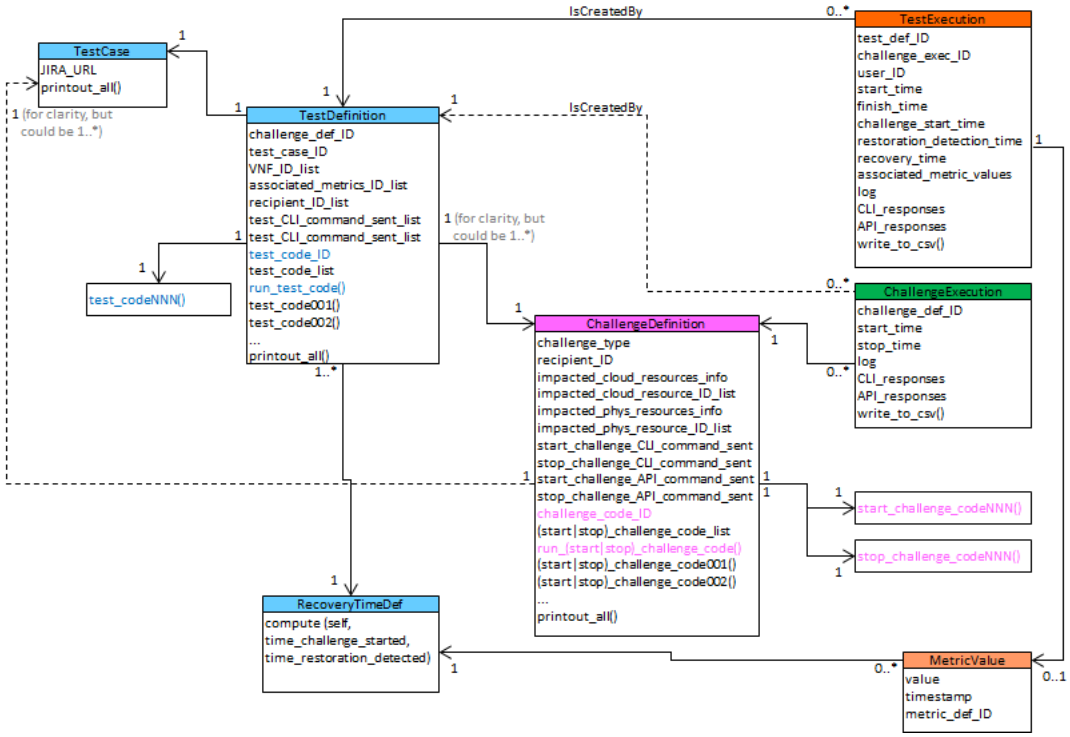

This last diagram shows the test user menu functions, when used interactively:



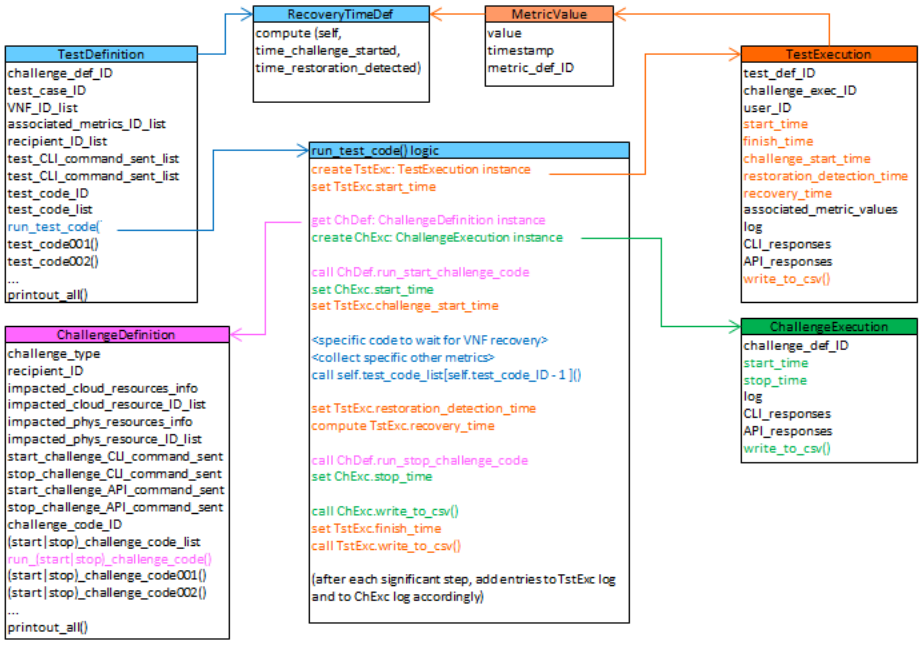

In future releases of Auto, testing environments such as Robot, FuncTest and Yardstick might be leveraged. Use Case code will then be invoked by API, not by a CLI interaction.

Also, anonymized test results could be collected from users willing to share them, and aggregates could be maintained as benchmarks.

As further illustration, the next figure shows cardinalities of class instances: one Test Definition per Test Case, multiple Test Executions per Test Definition, zero or one Recovery Time Metric Value per Test Execution (zero if the test failed for any reason, including if ONAP failed to recover the challenge), etc.



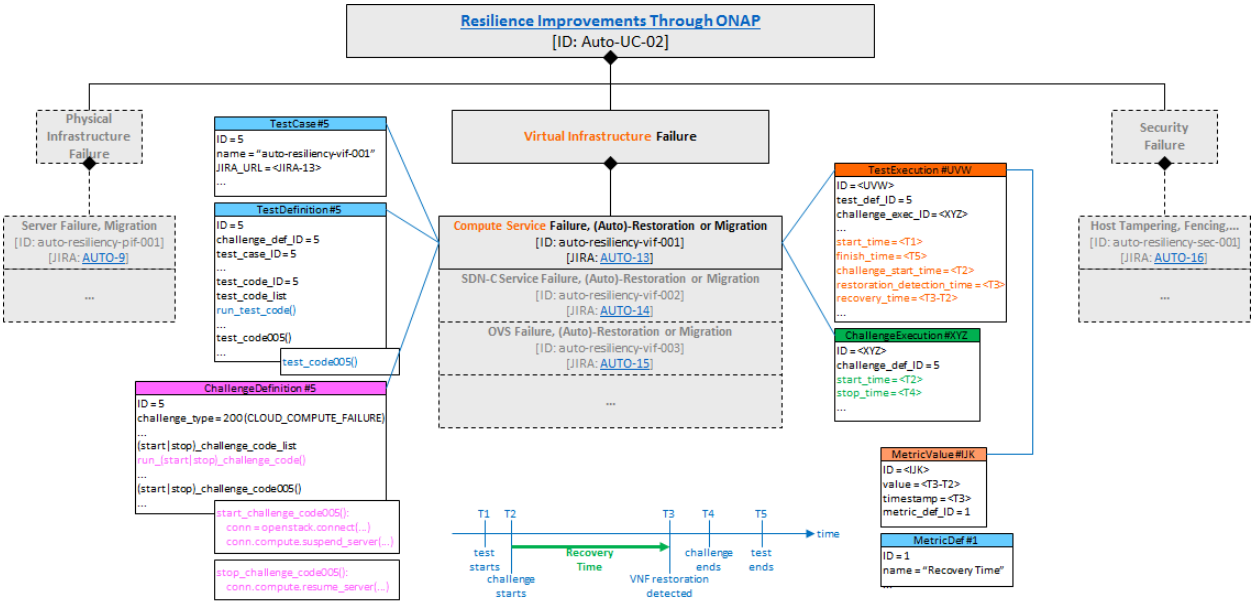

In this particular implementation, both Test Definition and Challenge Definition classes have a generic execution method (e.g., `run_test_code()` for Test Definition) which can invoke a particular script, by way of an ID (which can be configured, and serves as a script selector for each Test Definition instance). The overall test execution logic between classes is show in the next figure.




The execution of a test case starts with invoking the generic method from Test Definition, which then creates Execution instances, invokes Challenge Definition methods, performs the Recovery time calculation, performs script-specific actions, and writes results to the CSV files.

Finally, the following diagram show a mapping between these class instances and the initial test case design. It

corresponds to the test case which simulates a VM failure, and shows how the OpenStack SDK API is invoked (with a connection object) by the Challenge Definition methods, to suspend and resume a VM.




## 2.3 Auto User Guide: Use Case 3 Enterprise vCPE

This document provides the user guide for Fraser release of Auto, specifically for Use Case 3: Enterprise vCPE.

### 2.3.1 Description

This Use Case shows how ONAP can help ensure that virtual CPEs (including vFW: virtual firewalls) in Edge Cloud are enterprise-grade. Other vCPE examples: vAAA, vDHCP, vDNS, vGW, vBNG, vRouter, . . .

ONAP operations include a verification process for VNF onboarding (i.e., inclusion in the ONAP catalog), with multiple Roles (Designer, Tester, Governor, Operator), responsible for approving proposed VNFs (as VSPs (Vendor Software Products), and eventually as end-to-end Services).

This process guarantees a minimum level of quality of onboarded VNFs. If all deployed vCPEs are only chosen from such an approved ONAP catalog, the resulting deployed end-to-end vCPE services will meet enterprise-grade requirements. ONAP provides a NBI (currently HTTP-based) in addition to a standard GUI portal, thus enabling a programmatic deployment of VNFs, still conforming to ONAP processes.

Moreover, ONAP also comprises real-time monitoring (by the DCAE component), which can perform the following functions:

- monitor VNF performance for SLAs
- adjust allocated resources accordingly (elastic adjustment at VNF level: scaling out and in, possibly also scaling up and down)
- ensure High Availability (restoration of failed or underperforming services)

DCAE executes directives coming from policies described in the Policy Framework, and closed-loop controls described in the CLAMP component.

ONAP can perform the provisioning side of a BSS Order Management application handling vCPE orders.

Additional processing can be added to ONAP (internally as configured policies and closed-loop controls, or externally as separate systems): Path Computation Element and Load Balancing, and even telemetry-based Network Artificial Intelligence.

Finally, this automated approach also reduces costs, since repetitive actions are designed once and executed multiple times, as vCPEs are instantiated and decommissioned (frequent events, given the variability of business activity, and a Small Business market similar to the Residential market: many contract updates resulting in many vCPE changes).

NFV edge service providers need to provide site2site, site2dc (Data Center) and site2internet services to tenants both efficiently and safely, by deploying such qualified enterprise-grade vCPE.

Preconditions:

1. hardware environment in which Edge cloud may be deployed
2. an Edge cloud has been deployed and is ready for operation
3. enterprise edge devices, such as ThinCPE, have access to the Edge cloud with WAN interfaces
4. ONAP components (MSO, SDN-C, APP-C and VNFM) have been deployed onto a cloud and are interfaced (i.e. provisioned for API access) to the Edge cloud

Main Success Scenarios:

- VNF spin-up
  - vFW spin-up: MSO calls the VNFM to spin up a vFW instance from the catalog and then updates the active VNF list
  - other vCPEs spin-up: MSO calls the VNFM to spin up a vCPE instance from the catalog and then updates the active VNF list
- site2site
  - L3VPN service subscribing: MSO calls the SDNC to create VXLAN tunnels to carry L2 traffic between client's ThinCPE and SP's vCPE, and enables vCPE to route between different sites.
  - L3VPN service unsubscribing: MSO calls the SDNC to destroy tunnels and routes, thus disable traffic between different sites.
- site2dc (site to Data Center) by VPN
- site2internet
- scaling control (start with scaling out/in)

See ONAP description of vCPE use case for more details, including MSCs.

Details on the test cases corresponding to this use case:

- vCPE VNF deployment
  - Spin up a vFW instance by calling NBI of the orchestrator.
  - Following the vFW example and pattern, spin up other vCPE instances
- vCPE VNF networking
  - Subscribe/Unsubscribe to a VPN service: configure tenant/subscriber for vCPE, configure VPN service
  - Subscribe/Unsubscribe to an Internet Access service: configure tenant/subscriber for vCPE, configure Internet Access service
- vCPE VNF Scaling
  - ONAP-based VNF Scale-out and Scale-in (using measurements arriving in DCAE, policies/CLAMP or external system performing LB function)

- later, possibly also scale-up and scale-down

The following diagram shows these test cases:

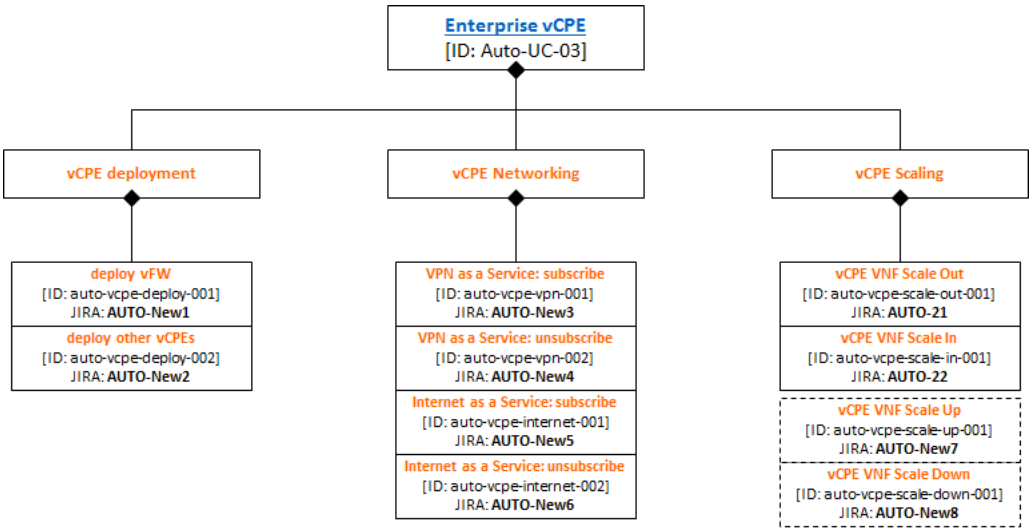
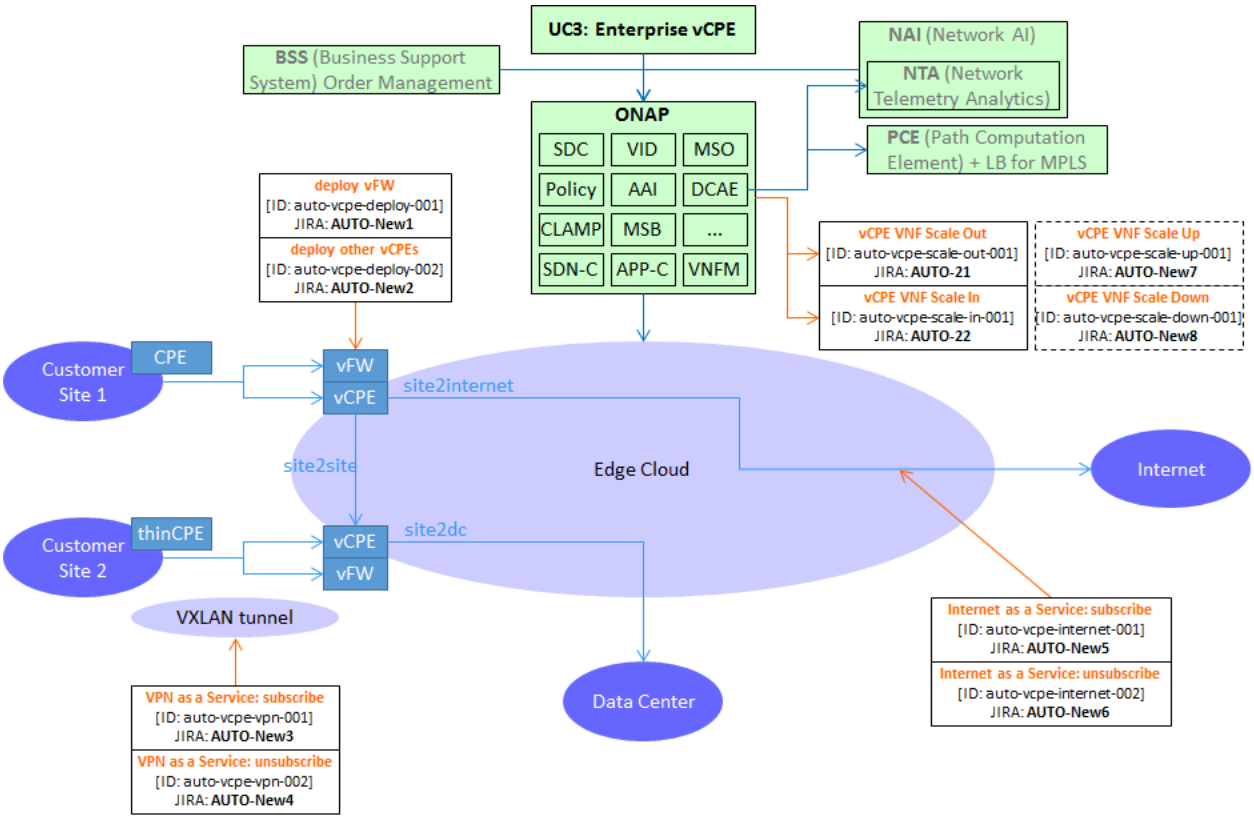



Illustration of test cases mapped to architecture, with possible external systems (BSS for Order Management, PCE+LB, Network AI:




### 2.3.2 Test execution high-level description

<TBC>

CHAPTER 3

# OPNFV Auto (ONAP-Automated OPNFV) Release Notes

## 3.1 Auto Release Notes

This document provides the release notes for the Gambia 7.0 release of Auto.

## 3.2 Important notes for this release

The initial release for Auto was in Fraser 6.0 (project inception: July 2017).

## 3.3 Summary

### 3.3.1 Overview

OPNFV is an SDNFV system integration project for open-source components, which so far have been mostly limited to the NFVI+VIM as generally described by ETSI.

In particular, OPNFV has yet to integrate higher-level automation features for VNFs and end-to-end Services.

As an OPNFV project, Auto (*ONAP-Automated OPNFV*) will focus on ONAP component integration and verification with OPNFV reference platforms/scenarios, through primarily a post-install process, in order to avoid impact to OPNFV installer projects (Fuel/MCP, Compass4NFV, Apex/TripleO, Daisy4NFV). As much as possible, this will use a generic installation/integration process (not specific to any OPNFV installer's technology).

- ONAP (a Linux Foundation Project) is an open source software platform that delivers robust capabilities for the design, creation, orchestration, monitoring, and life cycle management of Software-Defined Networks (SDNs). The current release of ONAP is B (Beijing).

Auto aims at validating the business value of ONAP in general, but especially within an OPNFV infrastructure (integration of ONAP and OPNFV). Business value is measured in terms of improved service quality (performance, reliability, ...) and OPEX reduction (VNF management simplification, power consumption reduction, ...), as demonstrated by use cases.

Auto also validates multi-architecture software (binary images and containers) availability of ONAP and OPNFV: CPUs (x86, ARM) and Clouds (MultiVIM)

In other words, Auto is a turnkey approach to automatically deploy an integrated open-source virtual network based on OPNFV (as infrastructure) and ONAP (as end-to-end service manager), that demonstrates business value to end-users (IT/Telco service providers, enterprises).

While all of ONAP is in scope, as it proceeds, the Auto project will focus on specific aspects of this integration and verification in each release. Some example topics and work items include:

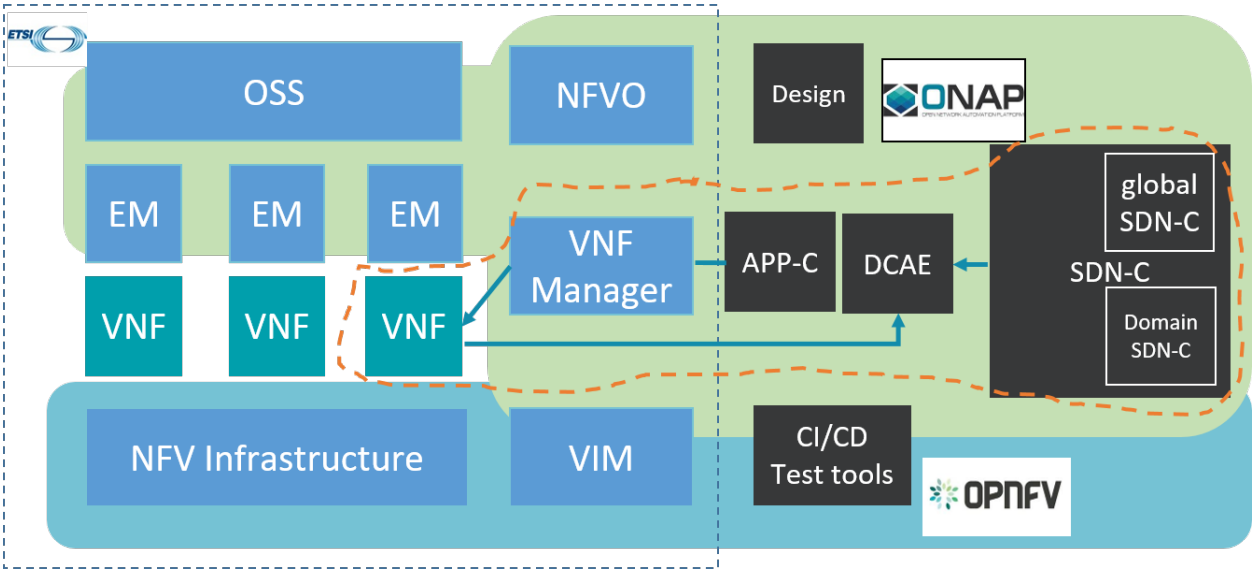
- How ONAP meets VNFM standards, and interacts with VNFs from different vendors
- How ONAP SDN-C uses OPNFV existing features, e.g. NetReady, in a two-layer controller architecture in which the upper layer (global controller) is replaceable, and the lower layer can use different vendor's local controller to interact with SDN-C. For interaction with multiple cloud infrastructures, the MultiVIM ONAP component will be used.
- How ONAP leverages OPNFV installers (Fuel/MCP, Compass4NFV, Apex/TripleO, Daisy4NFV) to provide a cloud instance (starting with OpenStack) on which to install the tool ONAP
- What data collection interface VNF and controllers provide to ONAP DCAE, and (through DCAE), to closed-loop control functions such as Policy Tests which verify interoperability of ONAP automation/lifecycle features with specific NFVI and VIM features, as prioritized by the project with OPNFV technical community and EUAG (End User Advisory Group) input.

  Examples:

  - Abstraction of networking tech/features e.g. through NetReady/Gluon
  - Blueprint-based VNF deployment (HOT, TOSCA, YANG)
  - Application level configuration and lifecycle through YANG (for any aspects depending upon OPNFV NFVI+VIM components)
  - Policy (through DCAE)
  - Telemetry (through VES/DCAE)

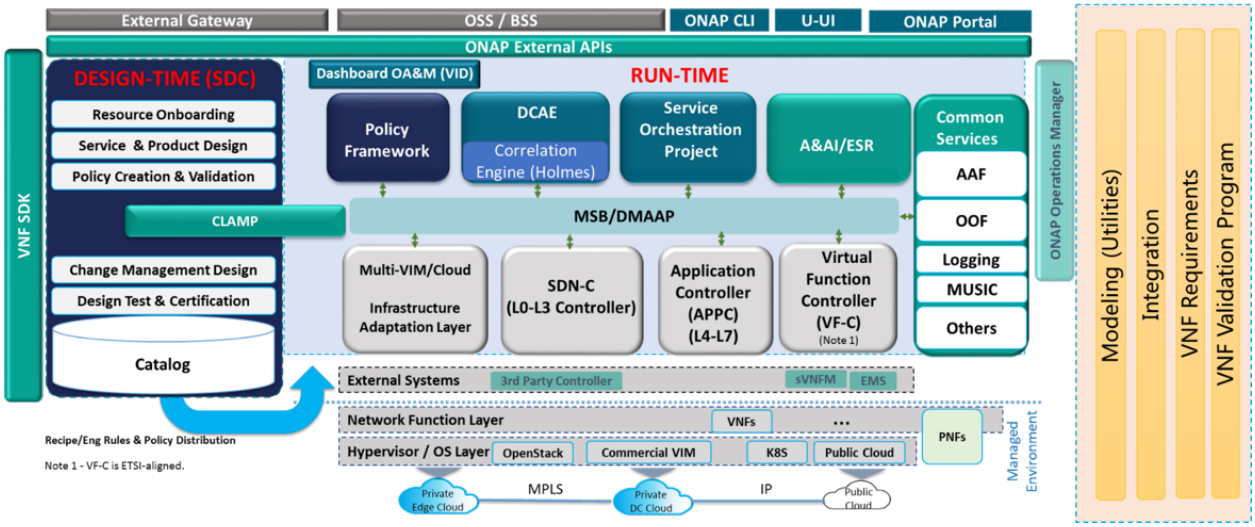
Initial areas of focus for Auto (in orange dotted lines; this scope can be expanded for future releases). It is understood that:

- ONAP scope extends beyond the lines drawn below
- ONAP architecture does not necessarily align with the ETSI NFV inspired diagrams this is based upon




The current ONAP architecture overview can be found here.

For reference, the ONAP-Beijing architecture diagram is replicated here:




Within OPNFV, Auto leverages tools and collaborates with other projects:

- use clouds/VIMs as installed in OPNFV infrastructure (e.g. OpenStack as installed by Fuel/MCP, Compass4NFV, etc.)
- include VNFs developed by OPNFV data plane groups (e.g., accelerated by VPP (Vector Packet Processing) with DPDK support, . . . )
- validate ONAP+VNFs+VIMs on two major CPU architectures: x86 (CISC), Arm (RISC); collaborate with OPNFV/Armband
- work with other related groups in OPNFV:
  - FuncTest for software verification (CI/CD, Pass/Fail)
  - Yardstick for metric management (quantitative measurements)
  - VES (VNF Event Stream) and Barometer for VNF monitoring (feed to ONAP/DCAE)

  - Edge Cloud as use case
- leverage OPNFV tools and infrastructure:
  - Pharos as LaaS: transient pods (3-week bookings) and permanent Arm pod (6 servers)
  - WorksOnArm (GitHub link)
  - possibly other labs from the community (Huawei pod-12, 6 servers, x86)
  - JJB/Jenkins for CI/CD (and follow OPNFV scenario convention)
  - Gerrit/Git for code and documents reviewing and archiving (similar to ONAP: Linux Foundation umbrella)
  - follow OPNFV releases (Releng group)

### 3.3.2 Testability

- Tests (test cases) will be developed for use cases within the project scope.
- In future releases, tests will be added to Functest runs for supporting scenarios.

Auto's goals include the standup and tests for integrated ONAP-Cloud platforms ("Cloud" here being OPNFV "scenarios" or other cloud environments). Thus, the artifacts would be tools to deploy ONAP (leveraging OOM whenever possible, starting with Beijing release of ONAP, and a preference for the containerized version of ONAP), to integrate it with clouds, to onboard and deploy test VNFs, to configure policies and closed-loop controls, and to run use-case defined tests against that integrated environment. OPNFV scenarios would be a possible component in the above.

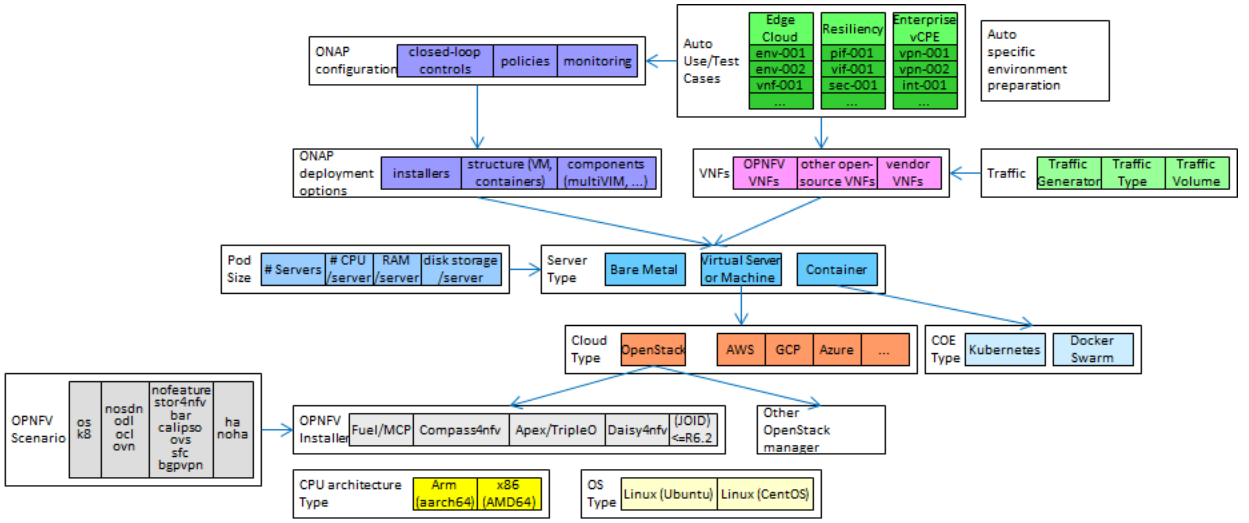
Installing Auto components and running a battery of tests will be automated, with some or all of the tests being integrated in OPNFV CI/CD (depending on the execution length and resource consumption).

Combining all potential parameters, a full set of Auto test case executions can result in thousands of individual results. The analysis of these results can be performed by humans, or even by ML/AI (Machine Learning, Artificial Intelligence). Test results will be used to fine-tune policies and closed-loop controls configured in ONAP, for increased ONAP business value (i.e., find/determine policies and controls which yield optimized ONAP business value metrics such as OPEX).

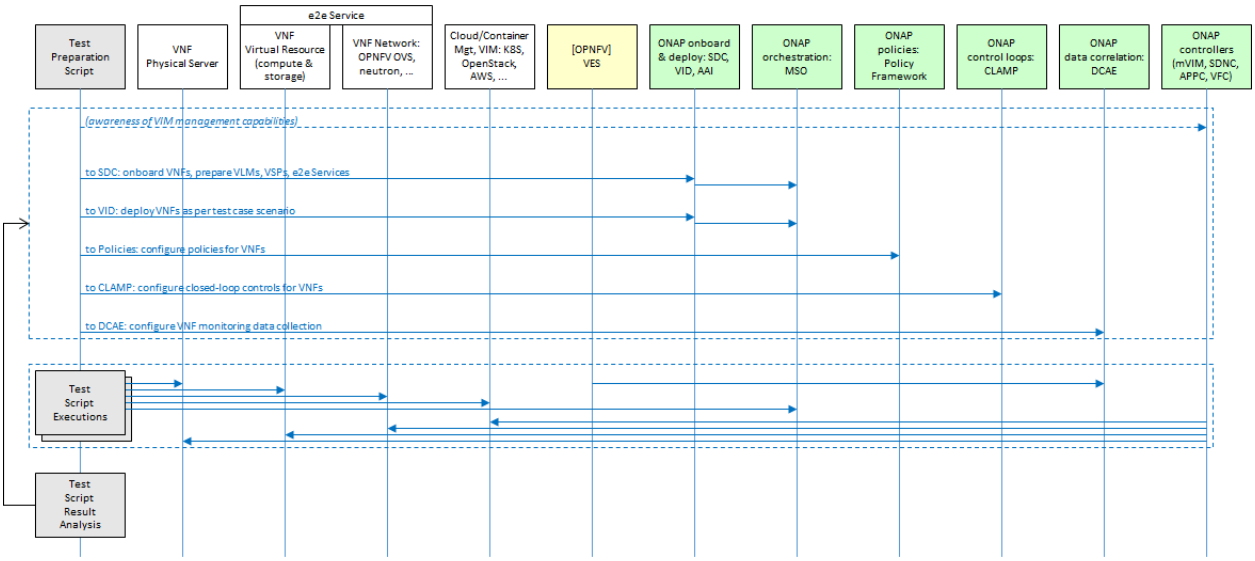
More precisely, the following list shows parameters that could be applied to an Auto full run of test cases:

- Auto test cases for given use cases
- OPNFV installer {Fuel/MCP, Compass4NFV, Apex/TripleO, Daisy4NFV}
- OPNFV availability scenario {HA, noHA}
- environment where ONAP runs {bare metal servers, VMs from clouds (OpenStack, AWS, GCP, Azure, ...), containers}
- ONAP installation type {bare metal, VM, or container, ... } and options {MultiVIM single|distributed, ... }
- VNF types {vFW, vCPE, vAAA, vDHCP, vDNS, vHSS, ... } and VNF-based services {vIMS, vEPC, ... }
- cloud where VNFs run {OpenStack, AWS, GCP, Azure, ... }
- VNF host type {VM, container}
- CPU architectures {x86/AMD64, ARM/aarch64} for ONAP software and for VNF software; not really important for Auto software;
- pod size and technology (RAM, storage, CPU cores/threads, NICs)
- traffic types and amounts/volumes; traffic generators (although that should not really matter);
- ONAP configuration {especially policies and closed-loop controls; monitoring types for DCAE: VES, ... }
- versions of every component {Linux OS (Ubuntu, CentOS), OPNFV release, clouds, ONAP, VNFs, ... }

The diagram below shows Auto parameters:




The next figure is an illustration of the Auto analysis loop (design, configuration, execution, result analysis) based on test cases covering as many parameters as possible :




Auto currently defines three use cases: Edge Cloud (UC1), Resiliency Improvements (UC2), and Enterprise vCPE (UC3). These use cases aim to show:

- increased autonomy of Edge Cloud management (automation, catalog-based deployment). This use case relates to the OPNFV Edge Cloud initiative.
- increased resilience (i.e. fast VNF recovery in case of failure or problem, thanks to closed-loop control), including end-to-end composite services of which a Cloud Manager may not be aware (VMs or containers could be recovered by a Cloud Manager, but not necessarily an end-to-end service built on top of VMs or containers).
- enterprise-grade performance of vCPEs (certification during onboarding, then real-time performance assurance with SLAs and HA, as well as scaling).

The use cases define test cases, which initially will be independent, but which might eventually be integrated to FuncTest.

Additional use cases can be added in the future, such as vIMS (example: project Clearwater) or residential vHGW

(virtual Home Gateways). The interest for vHGW is to reduce overall power consumption: even in idle mode, physical HGWs in residential premises consume a lot of energy. Virtualizing that service to the Service Provider edge data center would allow to minimize that consumption.

### 3.3.3 Lab environment

Target architectures for all Auto use cases and test cases include x86 and Arm. Power consumption analysis will be performed, leveraging Functest tools (based on RedFish/IPMI/ILO).

Initially, an ONAP-Amsterdam instance (without DCAE) had been installed over Kubernetes on bare metal on a single-server x86 pod at UNH IOL.

A transition is in progress, to leverage OPNFV LaaS (Lab-as-a-Service) pods (Pharos). These pods can be booked for 3 weeks only (with an extension for a maximum of 2 weeks), so they are not a permanent resource.

For ONAP-Beijing, a repeatable automated installation procedure is being developed, using 3 Pharos servers (x86 for now). Also, a more permanent ONAP installation is in progress at a Huawei lab (pod-12, consisting of 6 x86 servers, 1 as jump server, the other 5 with this example allocation: 3 for ONAP components, and 2 for an OPNFV infratructure: Openstack installed by Compass4NFV).

ONAP-based onboarding and deployment of VNFs is in progress (ONAP-Amsterdam pre-loading of VNFs must still done outside of ONAP: for VM-based VNFs, users need to prepare OpenStack stacks (using Heat templates), then make an instance snapshot which serves as the binary image of the VNF).

A script to prepare an OpenStack instance for ONAP (creation of a public and a private network, with a router, pre-loading of images and flavors, creation of a security group and an ONAP user) has been developed. It leverages OpenStack SDK. It has a delete option, so it can be invoked to delete these objects for example in a tear-down procedure.

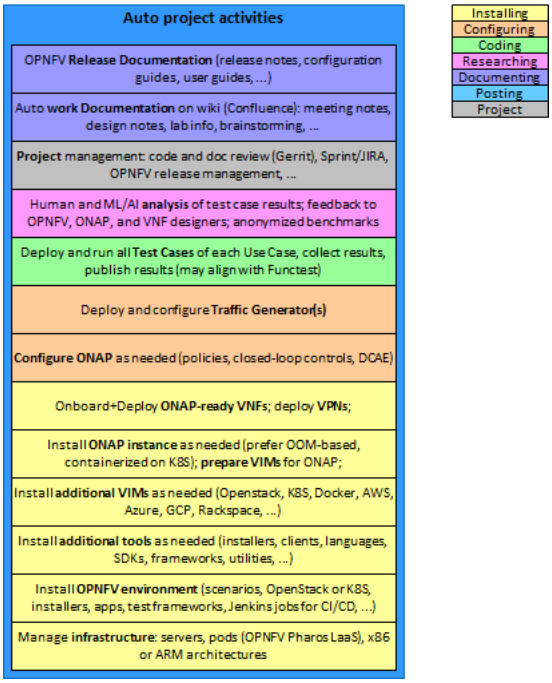
Integration with Arm servers has started (exploring binary compatibility):

- The Auto project has a specific 6-server pod of Arm servers, which is currently loaned to ONAP integration team, to build ONAP images
- A set of 14 additional Arm servers was deployed at UNH, for increased capacity
- ONAP Docker registry: ONAP-specific images for ARM are being built, with the purpose of populating ONAP nexus2 (Maven2 artifacts) and nexus3 (Docker containers) repositories at Linux Foundation. Docker images are multi-architecture, and the manifest of an image may contain 1 or more layers (for example 2 layers: x86/AMD64 and ARM/aarch64). One of ONAP-Casablanca architectural requirements is to be CPU-architecture independent. There are almost 150 Docker containers in a complete ONAP instance. Currently, more disk space is being added to the ARM nodes (configuration of Nova, and/or additional actual physical storage space).

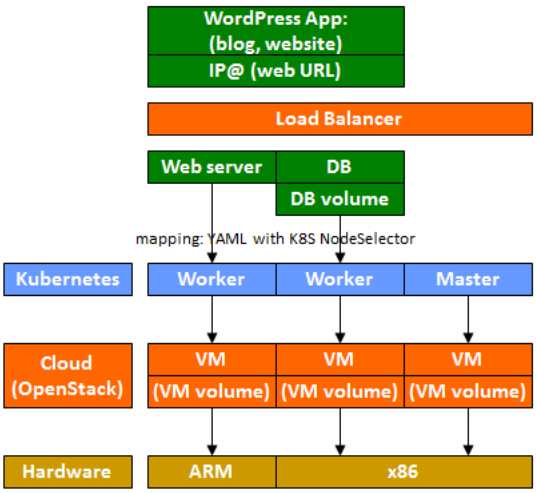
Test case design and implementation for the three use cases has started.

OPNFV CI/CD integration with JJD (Jenkins Job Description) has started: see the Auto plan description here. The permanent resource for that is the 6-server Arm pod, hosted at UNH. The CI directory from the Auto repository is here

Finally, the following figure illustrates Auto in terms of project activities:




Note: a demo was delivered at the OpenStack Summit in Vancouver on May 21st 2018, to illustrate the deployment of a WordPress application (WordPress is a platform for websites and blogs) deployed on a multi-architecture cloud (mix of x86 and Arm servers). This shows how service providers and enterprises can diversify their data centers with servers of different architectures, and select architectures best suited to each use case (mapping application components to architectures: DBs, interactive servers, number-crunching modules, ...). This prefigures how other examples such as ONAP, VIMs, and VNFs could also be deployed on heterogeneous multi-architecture environments (open infrastructure), orchestrated by Kubernetes. The Auto installation scripts covering all the parameters described above could expand on that approach.



## 3.4 Release Data

| | |
|---|---|
| **Project** | Auto |
| **Repo/commit-ID** | auto/opnfv-7.0.0 |
| **Release designation** | Gambia 7.0 |
| **Release date** | 2018-11-02 |
| **Purpose of the delivery** | Official OPNFV release |

### 3.4.1 Version change

#### Module version changes

- There have been no version changes.

#### Document version changes

- There have been no version changes.

### 3.4.2 Reason for version

#### Feature additions

Initial release 6.0:

- Fraser release plan
- use case descriptions
- test case descriptions
- in-progress test case development
- lab: OPNFV and ONAP (Amsterdam) installations

Point release 6.1:

- added Gambia release plan
- started integration with CI/CD (JJB) on permanent Arm pod
- Arm demo at OpenStack Summit
- initial script for configuring OpenStack instance for ONAP, using OpenStack SDK 0.13
- initial attempts to install ONAP Beijing
- alignment with OPNFV Edge Cloud
- initial contacts with Functest

Point release 6.2:

- initial scripts for OPNFV CI/CD, registration of Jenkins slave on Arm pod
- updated script for configuring OpenStack instance for ONAP, using OpenStack SDK 0.14

Point release 7.0:

- progress on Docker registry of ONAP's Arm images
- progress on ONAP installation script for 3-server cluster of UNH servers
- CI scripts for OPNFV installers: Fuel/MCP (x86), Compass, Apex/TripleO (must run twice)
- initial CI script for Daisy4NFV (work in progress)
- JOID script, but supported only until R6.2, not Gambia 7.0
- completed script for configuring OpenStack instance for ONAP, using OpenStack SDK 0.17
- use of an additional lab resource for Auto development: 6-server x86 pod (huawei-pod12)

**JIRA TICKETS for this release:**

JIRA Auto Gambia 7.0.0 Done

Manual selection of significant JIRA tickets for this version's highlights:

| JIRA REFERENCE | SLOGAN |
|---|---|
| AUTO-37 | Get DCAE running onto Pharos deployment |
| AUTO-42 | Use Compass4NFV to create an OpenStack instance on a UNH pod |
| AUTO-43 | String together scripts for Fuel, Tool installation, ONAP preparation |
| AUTO-44 | Build ONAP components for arm64 platform |
| AUTO-45 | CI: Jenkins definition of verify and merge jobs |
| AUTO-46 | Use Apex to create an OpenStack instance on a UNH pod |
| AUTO-47 | Install ONAP with Kubernetes on LaaS |
| AUTO-48 | Create documentation for ONAP deployment with Kubernetes on LaaS |
| AUTO-49 | Automate ONAP deployment with Kubernetes on LaaS |
| AUTO-51 | huawei-pod12: Prepare IDF and PDF files |
| AUTO-52 | Deploy a running ONAP instance on huawei-pod12 |
| AUTO-54 | Use Daisy4nfv to create an OpenStack instance on a UNH pod |
| | |

#### Bug corrections

**JIRA TICKETS:**

| JIRA REFERENCE | SLOGAN |
|---|---|
| | |
| | |

## 3.5 Deliverables

### 3.5.1 Software deliverables

7.0 release: in-progress Docker ARM images, install scripts, CI scripts, and test case implementations.

### 3.5.2 Documentation deliverables

Updated versions of:

- Release Notes (this document)

- User Guide
- Configuration Guide

(see links in References section)

## 3.6 Known Limitations, Issues and Workarounds

### 3.6.1 System Limitations

### 3.6.2 Known issues

None at this point.

**JIRA TICKETS:**

| JIRA REFERENCE | SLOGAN |
|---|---|
| | |
| | |

### 3.6.3 Workarounds

None at this point.

## 3.7 Test Result

None at this point.

| TEST-SUITE | Results: |
|---|---|
| | |
| | |

## 3.8 References

For more information on the OPNFV Gambia release, please see: http://opnfv.org/gambia

Auto Wiki pages:

- Auto wiki main page

OPNFV documentation on Auto:

- *Auto release notes <auto-releasenotes>*
- *Auto use case user guides <auto-userguide>*
- *Auto configuration guide <auto-configguide>*

Git&Gerrit Auto repositories:

- Auto Git repository

- Gerrit for Auto project

Demo at OpenStack summit May 2018 (Vancouver, BC, Canada):

- YouTube video (10min 52s): Integration testing on an OpenStack public cloud