



Openet Weaver® Enterprise

Contents

Welcome to Openet Weaver® Enterprise Edition Online Help.....	7
Preface.....	7
Command Syntax Conventions.....	12
Legal Notice.....	13
Quick Links.....	14
Release Notes.....	15
Openet Weaver Overview.....	17
Openet Weaver Benefits.....	18
Openet Weaver Features.....	19
Openet Weaver Architecture.....	22
Openet Weaver Call Flows.....	27
Configuration Manager Directory Structure.....	28
Openet Weaver Use Cases.....	30
Openet Weaver Quick Start Guide.....	32
Download and install the Virtual image.....	32
Installing Openet Weaver on the Virtual image.....	33
Running the Health Check.....	36
Example VNF.....	38
Deploy Example VNF without VIM.....	40
Deploy Example VNF with VIM.....	42
Accessing the VNF.....	46
Installing Openet Weaver.....	47
Hardware and Software Requirements.....	48
Third-Party Components Delivered with Openet Weaver.....	50
Extracting the TAR File.....	51
Performing Pre-Installation Tasks.....	52
Users and Privileges.....	52
Configuring Rsync to Run in SSH Mode.....	55
Configuring Custom Rsync.....	59
Configuring Openet Weaver for HTTPS.....	61
Installing and Configuring Keycloak.....	66
Configuring VIM (OpenStack) for Openet Weaver.....	70

Configuring Master and Hot-Standby Nodes for Local High Availability.....	83
Creating an Installation File.....	92
Performing the Initial Installation.....	130
Performing Post-Installation Tasks.....	133
Setting Environment Variables.....	133
Installing the CLI on a Client Node.....	134
Verifying Resource Agent Connectivity.....	135
Keycloak Security Post Installation Setup.....	136
Accessing Resources Through the VNF-M GUI.....	137
Checking Health Status.....	138
Modifying Deployments.....	139
Performing Configuration Updates.....	142
Modifying Microservices.....	144
Reinstalling Content Packs.....	151
Updating Out of the Box Procedures	151
Granting and Revoking OVLM User Sudo Privileges.....	151
Scaling Deployments.....	153
Restarting Microservices.....	154
Openet Weaver VNF-M GUI.....	154
Logging into the VNF-M GUI.....	155
VNF-M GUI Pages.....	156
Selecting a Tenant.....	163
Using the Search Function.....	165
Identifying Mandatory Fields.....	165
Checking system Health Status.....	166
Logging Out of the VNF-M GUI.....	167
Using Openet Weaver in a Secure Environment.....	170
Creating VNFs	173
Creating VNF Packages.....	173
Creating a Tenant.....	174
Creating a VNF Workspace and VNFC Containers.....	177
Creating and Uploading Metadata for the VNFC.....	177
Uploading Packages and Configuration Files.....	181
Creating and Uploading a VNF Descriptor (VNFD) File.....	182
Creating a VNF Instance.....	196
Creating the VNF Package.....	198
Creating and Uploading the VNF Topology YML File.....	198
Instantiating a VNF from a VNF package.....	204
Creating Custom Procedures with OVLM Python API.....	204
Listing VNF Resources.....	209

Understanding Dynamic Configuration.....	213
Validating VNFC Configuration Files.....	218
Adding an Application User to Openet Weaver.....	228
Setting Up a Reference VNF.....	229
Creating a VNF Topology File.....	235
Instantiating a Reference VNF.....	236
Running the instantiate_reference_vnf Workflow.....	238
Installing the Weaver Integration Module and Configuring the VNF Reference	242
Managing VNFs.....	243
Managing VNF packages.....	243
Importing a VNF package.....	243
Exporting a VNF package.....	250
Deleting a VNF package.....	251
Running VNF Workflows.....	252
Instantiating VNFs.....	256
Upgrading VNFs.....	271
Updating VNFs.....	283
Rolling Back VNFs.....	294
Starting VNFs.....	300
Stopping VNFs.....	303
Checking That a VNF is Up.....	307
Terminating a VNF.....	315
Running VNFC Workflows.....	323
Reinstantiating VNFCs without VIM.....	327
Upgrading VNFCs.....	331
Updating VNFCs.....	345
Rolling Back VNFCs	357
Healing a VNFC	368
Starting VNFCs.....	372
Stopping VNFCs.....	380
Running a custom_action_vnfc Workflow	388
Cancelling a Workflow.....	397
Managing Additional Parameters for the VNF-M GUI.....	398
Creating a VNF Metadata File.....	398
Downloading and Uploading VNF Metadata.....	401
JSON Schema for VNF Metadata.....	403
Configuring Workflows to Retry Automatically.....	405
Synchronizing Performance Manager and Fault Manager Manually.....	409
Synchronizing Performance Manager Manually.....	409
Synchronizing Fault Manager Manually.....	411
Listing Workflows.....	412

Checking Workflow Status.....	413
Listing Instantiated VNFs.....	414
Deleting a VNF.....	415
Managing Performance.....	416
Managing Performance Policies	417
Uploading a Performance Policy.....	418
Downloading a Performance Policy.....	420
Disabling a Policy on a VNF Instance.....	420
Deleting a Performance Policy.....	421
Checking VNF Health.....	422
Setting Up <code>is_vnf_healthy</code> to Run Periodically.....	422
Managing Metric Logs.....	424
Managing Faults.....	426
Fault Policy Structure.....	426
Policy Based Alarm Filtering.....	429
Managing Fault Policies	430
Uploading a Fault Policy.....	430
Downloading a Fault Policy.....	430
Disabling a Policy on a VNF Instance.....	430
Deleting a Fault Policy.....	431
Managing MIB Files.....	432
Managing Alarms.....	433
Synchronizing A VNF Instance Configuration.....	434
Retrieving Alarms.....	435
Deleting a VNF Instance Configuration.....	436
SNMP Traps.....	436
Configuring Auto Healing Support for a VNF.....	437
Associating Alarms with a VNFC.....	439
Updating a VNFC with SNMP Support.....	439
Configuring Fault Manager with a Health Check Policy.....	442
Creating the package and Instantiating.....	444
Synchronize the VNF Instance Policy with Fault Manager.....	445
Generate and Examine an SNMP Trap.....	445
Openet Weaver Operational Commands	450
Troubleshooting Openet Weaver.....	452
Troubleshooting Installation Issues.....	453
Troubleshooting Workflow Runtime Errors.....	457
Troubleshooting OpenStack-Related Issues.....	468

Troubleshooting Deployment Manager Issues.....	468
Openet Weaver VNF-M API Reference.....	468
Openet Weaver VNF-M CLI Reference.....	469
Openet Weaver Python API Reference.....	469
Glossary.....	469

Welcome to Openet Weaver® Enterprise Edition Online Help

Openet Weaver is Openet's VNF Lifecycle Manager (OVLM). It is a generic Virtual Network Function manager (VNF-M) for managing VNFs at the software level instead of the virtual machine level. This allows for the fine-grained control of VNFs when managing VNF-M lifecycle events, for example application installation and configuration.

This online help provides information about how to use Openet Weaver (v1.11) Enterprise Edition (EE) and includes topics about:

- Installing Openet Weaver
- Creating Openet Weaver packages to define VNF configurations
- Using Openet Weaver workflows to automate VNF management tasks
- Monitoring and retrieving VNF performance metrics in Openet Weaver
- Openet Weaver system operations
- Openet Weaver Administration and maintenance
- Securing the system using Keycloak

To get started quickly on using Openet Weaver, see the [Quick Start Guide](#).

Preface

The Preface discusses objectives, audience, conventions, and organization of this document and explains how to find additional information on related products and services.

Objectives

This online help describes how to deploy and configure Openet Weaver.

Audience

This online help is intended for System integrators and partners who are deploying and configuring Openet applications and third-party VNFs using Openet Weaver.

Revision History

The revision history records technical changes to this online help. The table shows the software release number and revision number for the change, the date of the change, and a brief summary of the change.

Table 1: Revision History

Release No.	Date	Change Summary
1.11	April 2018	A new workflow was added— <code>instantiate_reference_vnf</code> . This workflow is used to recreate a reference installation from a VNF template in a new node or environment.

Release No.	Date	Change Summary
1.10	November 2017	<ul style="list-style-type: none"> • Resource Agents no longer need to be predefined in the installation file and installed part of the overall Openet Weaver installation process. Resource Agents can now be installed automatically during VNF instantiation based on the VNF topology file using Deployment Manager, a new microservice. • A new workflow, instantiate_vnfc_without_wim, restores a VNFC instance for an instantiated VNF instance. • An SNMP trap is sent out when there is an outage of any Openet Weaver microservice. • The health status of each VNFC instance is now reported for VNFC rollback. • Custom SSH and Rsync is now supported as an alternative to the built-in SSH and Rsync. • A new workflow, vnf_event_unlock, unlocks VNF instances that are locked due to an error condition. The workflow updates the state of the VNF instance in Instance Inventory Manager. • Resource Agents can be upgraded and rolled back using the VNF-M GUI, the Deployment Manager CLI, or REST API.
1.9	August 2017	<ul style="list-style-type: none"> • Keycloak (An identity and access management system) is now integrated with Openet Weaver. • The following changes were made to Openet Weaver installation: <ul style="list-style-type: none"> • Microservices can be configured to deploy to any location, including user space. • Python modules can be installed in user space and run under the Python virtual environment. This provides isolation of python library dependencies. • Staging operations can be configured to run using a custom Rsync wrapper with built-in SSH. • VNF Instantiation workflows now stop quickly and gracefully when one of the VNFC instances fails. • Changes have been made to selected workflows to enhance health status reporting: <ul style="list-style-type: none"> • Health status of each VNFC instance is reported in the front end REST/CLI response. • The workflows report UP, DOWN, and UNKNOWN health status. UNKNOWN health status is reported when the Workflow Engine is not able to communicate with a Resource Agent or there are script error. • The enhancement applies to the following workflows: <ul style="list-style-type: none"> • VNF instantiation • VNF upgrade • VNF rollback • is VNF up • VNFC upgrade • Non-ascii encoding is supported by Jinja rendering. The default encoding is UTF-8. • The Workflow Engine microservice is upgraded to version 10.70.

Release No.	Date	Change Summary
1.8	June 2017	<ul style="list-style-type: none"> • The following improvements were made to Resource Agent Configuration: <ul style="list-style-type: none"> • You can customize the VNF template repository root and log directories to reside in the user space. • You can configure log rotation based on time period and file size or a combination of both. • Procedural logging messages are enriched with VNF information. • The VNF lifecycle procedural level log file name can be configured to contain the VNFC label. • The following improvements were made to Openet Weaver VNF and VNFC workflows. <ul style="list-style-type: none"> • The Update workflow can be executed at both VNF and VNFC level. • The Update workflow supports sequential and parallel execution order. • The <code>is_vnf_up</code> workflow response now contains the health status of all VNFC instances in a VNF instance. • The <code>custom_action_vnfc</code> response structure was Improved so that REST API users can parse the response easily. • A few repeatable rollback edge cases were fixed at VNF and VNFC level. • End users can predefine additional parameters for workflows at the VNF and VNFC level. The Openet Weaver GUI can display these parameters to enhance usability. • The number of sudo privileges required to install Openet Weaver successfully was reduced to a minimum. End users can assign and remove sudo privileges for Openet Weaver users running lifecycle management procedures on the fly.

Release No.	Date	Change Summary
1.7	April 2017	<ul style="list-style-type: none"> • Additional parameter passing <ul style="list-style-type: none"> • Openet Weaver's additional parameter passing provides a means to pass static and dynamic parameters from the Front End to the Resource Agent, so that the parameters can be used to manage the VNF. VNF lifecycle management procedures must be flexible enough to execute different logic, for a given workflow type, using additional parameters supplied by the end users at runtime. • VNFC instance lifecycle management <ul style="list-style-type: none"> • Lifecycle management was extended to support the orchestration of start, stop, upgrade and rollback VNFC. This allows you more granular control of VNF instance lifecycles, down to individual VNFC-instance level. • Instantiation -start flag <ul style="list-style-type: none"> • A start flag was added to the instantiation flow that gives you the option of starting the VNF on instantiation or performing the start as a separate operation. When you do not start a VNF on instantiation, you can complete instantiation in a shorter time window than you might with time-consuming start processes. • Custom VNFC action <ul style="list-style-type: none"> • The custom_action_vnfc workflow was enhanced to support multiple VNFC instances on a single VM through the CLI or REST API. <p style="margin-left: 20px;">Note: the VNF-M GUI will be supported in a later release.</p> • Python Module libraries to assist in the development of VNF lifecycle management procedures <ul style="list-style-type: none"> • A library of Python modules is included that can be downloaded and installed independently from Openet Weaver. Developers can use the library to start and test VNF lifecycle management procedures. • The ability to specify a VNFC user, under which VNF lifecycle procedures can be run <ul style="list-style-type: none"> • Developers can configure a run_as user as metadata per VNFC during template development. Specifying a VNFC user can help create a clean separation between internal Openet Weaver activities and VNFC management activities such as installing, starting, and stopping for example. • VNF lifecycle procedures can be reusable and independent of the run_as user.

Release No.	Date	Change Summary
1.6	February 2017	<ul style="list-style-type: none"> • The following workflows now support parallel operation: <ul style="list-style-type: none"> • Rollback • Upgrade • Multiple VNFCs running in the same VM is now supported. • The workflow_submit -force option can be used to run an applicable workflow against specific VNFCs. • The keepalived configuration is updated to use snmp support provided by keepalived with agent-x enabled for fail-over, as well as the NIS user and NIS group support. • A standalone Openet Weaver CLI installer can install individual CLIs onto nodes other than the from management node. <p>The Workflow type is sent from the Front End to the Resource Agent so that the same set of VNF level lifecycle procedures can be used in multiple lifecycle operations.</p> <ul style="list-style-type: none"> • You can now use NIS Domain and NIS User with Openet Weaver. • The Openet Weaver Installer can now stop and restart Microservices after a configuration change.
1.5	December, 2016	<ul style="list-style-type: none"> • The VNF-M GUI was added. • The following lifecycle workflows were added: <ul style="list-style-type: none"> • Rollback • Custom workflow • The following security features were added: <ul style="list-style-type: none"> • HTTPS • Rsync using SSH tunnelling • Non-root user privileges • Password encryption • There is now high availability with support for active and hot standby nodes. • Enhanced VNFD to support multiple network interfaces for VNFs to be instantiated onto OpenStack-managed VMs. • Updated OpenStack release support with integration with OpenStack Releases Kilo and Liberty.
1.4	September, 2016	<ul style="list-style-type: none"> • The Fault Manager microservice was added. • The following Lifecycle workflows were added <ul style="list-style-type: none"> • Upgrade • Update (Patches and Configuration) • Terminate • Healing
1.3	August, 2016	<ul style="list-style-type: none"> • VNF Descriptors (VNFDs) are now supported • OpenStack is Integrated and packaged with Openet Weaver • TOSCA templates are translated to HEAT templates for the OpenStack environment • Openet Weaver now supports Instantiation Lifecycles with and without Virtualized Infrastructure Manager.

Release No.	Date	Change Summary
1.2	July 8th, 2016	<ul style="list-style-type: none"> The Performance Manager microservice was added. Openet Weaver now has the ability to perform VNF Health checks and collect metrics Ubuntu support is now provided for Resource Agents
1.1 CE	May 19, 2016	<ul style="list-style-type: none"> Multi-tenancy is now supported There is now a VNF import and export feature You can retrieve additional information about VNFs including: <ul style="list-style-type: none"> a list of packages per VNF A list of all instantiated packages Detailed information about instantiated packages You can retrieve a list of all HPOO workflows Service health checks can now be performed using a front-end CLI call packages are now versioned. Synchronous and asynchronous API calls
1.0 CE	April 1, 2016	First Release

Command Syntax Conventions

A guide to understanding the command syntax conventions used in this document.

Table 2: Command Syntax Guide

Convention	Description	Example
Bold	This denotes literal information that must be entered on the command line exactly as shown. This applies to command names, keywords and non-variable options.	<code>AssociateWorkspaceAndCorrelator workspaceName cteName1 [cteNameK ... cteNameN]</code>
<i>Italic</i>	Italic text is variable and must be provided by the user. In the example to the right, you would replace workspaceName and cteName1 with the name of the specific workspace and CTE.	<code>AssociateWorkspaceAndCorrelator workspaceName <i>cteName1</i></code>
Brackets []	The information enclosed in brackets [] is optional. Anything not enclosed in brackets must be specified.	<code>workspaceName <i>cteName1</i> [<i>cteNameK</i> ... <i>cteNameN</i>]</code>
Braces { x x x }	A choice of keywords (represented by x) appears in braces separated by vertical bars. You must select one.	<code>{-i <i>IP_address</i> -n <i>host_name</i> }</code>

Convention	Description	Example
Vertical bar	Mutually exclusive options are separated by a vertical bar (). A vertical bar can be used to separate optional or required options.	{ -i IP_address -n host_name }
Ellipsis ...	An ellipsis (...) indicates that the previous option can be repeated multiple times with different values. It can be used inside or outside of brackets.	AssociateWorkspaceAndCorrelator workspaceName cteName1 [cteNameK ... cteNameN]
system output	Examples of output displayed on the screen.	HTTP/1.1 500 Server error Content-Type: application/json; charset=UTF-8 Transfer-Encoding: chunked Server: Jetty(9.1.2.v20140210) {"WARNINGS": ["Resource 'resourcename' is already disabled for runtime node 'serverId'"]}
Bold screen font	GUI elements (tabs, menu choices, menu names, controls).	"Click File > Open "
Angle Brackets <>	Used for parameters, for example in URIs.	http://<host>:<port>/v1/action/<snapshot_id>

Legal Notice

The use of this online help ("document") is subject to a user agreement. The product it describes is covered by several patents.

Legal

No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording, or by any information storage or retrieval system, without the written permission of Openet. This document and the software described within are the sole intellectual property of Openet. The information contained in these pages is to the best of our knowledge true and accurate at the time of publication and is provided solely for information purposes. None of the information may be reproduced without the prior written approval of Openet. Openet accepts no liability for any loss or damage howsoever arising as a result of use of or reliance on this information, whether authorized or not.

Openet Documentation User Agreement

This document and the related materials contains confidential material supplied by Openet under strict terms and conditions governed by its licence agreements that are binding upon recipients. By accepting this document and reading or otherwise using the contents, you agree to be bound by the terms set out below:

1. You agree that:

- a. You will treat these materials as confidential materials (as defined in the Openet agreements).
- b. You will not duplicate, copy, transcribe or share in any form these materials.

No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording, or by any information storage or retrieval system, without the written permission of Openet. This document and the software described within are the sole intellectual property of

Openet. The information contained in these pages is to the best of our knowledge true and accurate at the time of publication and is provided solely for information purposes. None of the information may be reproduced without the prior written approval of Openet. Openet accepts no liability for any loss or damage howsoever arising as a result of use of or reliance on this information, whether authorized or not.

- c. You have been provided with these materials on a restricted basis to allow you only to use the software within the scope of the application package and to the internal business operations of the licenced and authorized user.
 - d. Openet retains all ownership and intellectual property rights to the materials.
 - e. You will prohibit (a) the removal or modification of any program markings or any notice of Openet or its proprietary rights; (b) the making by anyone of the materials available in any manner to any third party for use in the third party's business operations.
 - f. You will prohibit the reverse engineering, disassembly or decompilation of the software (the foregoing prohibition includes but is not limited to review of data structures or similar materials, methods, processes, designs or other know-how).
2. You disclaim, to the extent permitted by applicable law, Openet's liability for (a) any damages, whether direct, indirect, incidental, special, punitive or consequential, and (b) any loss of profits, revenue, data or data use, arising from the use of the software.
 3. You will prohibit, and not aid in any way, the publication of any of the materials.
 4. You acknowledge that you are not permitted to modify the materials, or create derivative works therefrom.
 5. You will inform Openet immediately on becoming aware of any breach of these conditions.

Trademark Protection

DSD, Smarter Engagement, Openet Accelerate, Enigma, Model Ink, Assure, Connect, Quant, Stokado, and Compass are trademarks of Openet Telecom Ltd. ConnectedHuman, Decision Core, FusionWorks, Openet, the Openet logo, Openet Express Solutions, and Weaver are registered trademarks of Openet Telecom Ltd.

Patent Protection

This product may be protected by patents, including patent numbers EP 2175588, EP 2518941, EP 2518942, EP 2518943, EP 2605501, US 8655308, US 8675659, US 8699332, US 8726376, US 8725896, US 8725820, US 8824370, US 8880726, US 8929859, US 8943221, US 9125076, US 9130760, US 9173081, US 9204279, US 9300531, US 9306891, US 9313339, US 9313640, US 9338126, US 9363224, US 9363177, US 9380018, US 9439129, US 9444692, US 9450766, US 9497611, EP 2518945, US 9516449, US 9544751, US 9565063, US 9565074, US 9602676, US 9609492, US 9641958, US 9641403, US 9686084, EP 3059935, US 9755891, EP 2816792, and EP 2466828. It may also be protected by additional pending patent applications.



6 Beckett Way
Park West Business Park
Dublin 12
www.openet.com
Tel: +353 1 620 4600
Fax: +353 1 620 4990

Quick Links

Use these links to get started quickly on tasks.

What Task Do You Want to Perform

I want to:

[*Get Started Quickly Using the predefined VM environment*](#)

[Install Openet Weaver](#)

[Create an Install Properties File for Installation](#)

[Configure Openet Weaver](#)

[Create a VNF package](#)

[Instantiate a VNF instance](#)

[Manage VNF Performance](#)

[Get an Openet Weaver service health check](#)

[Add Openet Weaver agent nodes](#)

Release Notes

These release notes were first published on April 20th, 2018 and describe new features, known issues, bug fixes, and usage tips for this release of Openet Weaver 1.11.

New and Changed Information

System Requirements

For the predefined VM environment used in the Quick Start Guide, see [Download and Install the Virtual Image](#) in the Openet Weaver Enterprise webhelp.

For deploying Openet Weaver in a lab environment, see [Hardware and Software Requirements](#) in the Openet Weaver enterprise webhelp.

New Features

The following table lists the features that are available in this release:

Table 3: New Features

Feature	Description
New workflow—instantiate_reference_vnf	A new workflow was added—instantiate_reference_vnf. This workflow is used to recreate a reference installation from a VNF template in a new node or environment.

Deprecated Features

The ovlm-deploy.sh script no longer manages the installation of Resource Agents. Therefore Resource Agent host properties are no longer required in the Installation file. Instead Resource Agents are installed automatically based on the VNF instance topology file during installation. Deployment Manager provides the capability to manage the installation of Resource Agents.

Installation Notes

The installation process for Openet Weaver is described in the Webhelp, which contains full instructions for completing an installation, including details of the minimum system requirements.

Limitations

The following table describes the limitations of this Openet Weaver release:

Feature	Description
Integration with Security Service Module	This feature is limited to non-VIM workflows.
Multiple VNFCs from same VNF sharing a VM	The following limitations apply to this feature <ul style="list-style-type: none"> It is not supported for the VIM use cases. It works only in non-VIM use cases. It is not support by the heal_vnfc workflow, which works on a single VNFC per VM. It is not supported by the optional micro-services Fault Manager and Performance Manager.
VNF Instantiation with -start flag	This is not supported for VIM use cases. it is supported for the non-VIM use cases only.
HTTPS requires domain name in topology yml	HTTPS support requires a domain name to be used in the hostname that is defined in the VNF Topology YML file.
Scale workflows	The scale_in_vnf and scale_out_vnf workflows are not supported although they are visible in the CLI and the Workflow Engine GUI.
VNF instance health status enhancement	This feature is limited to the following workflows: <ul style="list-style-type: none"> custom vnfc action is vnf up vnf instantiation vnf upgrade vnf rollback
Managing Resource Agent microservice using Deployment Manager	Support is limited to install, re-install, upgrade and rolling back upgrade of Resource Agent microservice. Refer to Webhelp for more information.

Open Caveats

The following table lists the open caveats for this release of Openet Weaver.

Table 4: Open Caveats

Summary	Issue No.
Terminate VNF with force 1 does not perform cleanup of RA repository.	OVM-4833
ovlm-deploy fails on performance manager installation/config-update.	OVM-4749
weaver [utilities] folder is not updated if exist.	OVM-4371
Weaver fails to instantiate VNF if there are multiple networks with the same name.	OVM-4326
Encrypt-password.sh removes comments and whitespace from yaml properties file.	OVM-3853
FE response message format is inconsistent.	OVM-3698
When there is no Openstack Image with a valid required attribute in Openstack , VAL throws a 500 unhandled exception.	OVM-3489
RA procedure log with additional_parameters is not standardized	OVM-3485
VNFCs listed in topology are instantiated, even if there is no Openet Weaver template definition for VNFC.	OVM-3475

Summary	Issue No.
For vnf_instance_id in number, ovlm-fe instantiated_vnf list command produce output in incorrect format (number in quote).	OVM-3386
ovlm-fe workflow status doesn't validate or use required parameters .	OVM-3248
VAL does not retrieve all the image that belongs to the current user.	OVM-3242
is_vnf_healthy flow fails when resource agent is not sending metrics.	OVM-3233
There is missing VNFD validation for duplicate names.	OVM-3214
FM contains unexpected set of WARN/ERRORS in logs.	OVM-3039
PM contains unexpected set of WARN/ERRORS in logs.	OVM-3038
Incorrect error message displayed when wrong vnf template id is used to instantiate vnf.	OVM-2656
PM - WF not up or incorrect triggerWorkflowID results in a 500 and no metrics added to Filesystem.	OVM-2361
Invalid network Id is not returned in error response.	OVM-1842
Body field being requested when executing custom flow in Weaver Workflow Dashboard.	OVM-1401

Third Party Software

This Openet product may include software components developed and owned by third parties. Details of such third party software components, their respective licences, and instructions on how to obtain their source code (where applicable) are available within the "Openet Third Party Software Acknowledgement" document, which is contained within the product.

Service and Support

Openet provides a comprehensive technical support program. To contact Openet Technical Support, use one of these methods:

- Email: support@openet.com
- Telephone: +353-1-620-4620 (EMEA) +1-866-673-6382 (Americas) +60-3-228-98515 (APAC)
- Visit the Openet Customer Support Portal: <http://support.openet.com>

Openet Weaver Overview

Openet Weaver, aligned with the ETSI Network Function Virtualization (NFV) architecture, is a flexible and generic virtual network function manager. It can manage Virtual Network Functions (VNFs) of different types that originate from multiple vendors. Openet Weaver can manage single instance VNFs or complex multi-instance VNFs that require advanced lifecycle management procedures.

Openet Weaver is an extensible framework that allows for custom lifecycle procedures and custom workflows. Its VNF management provides extensive control over lifecycle events. In contrast, other solutions manage VNFs at the higher level of Virtual Machine (VM) image.

VNFs are responsible for handling specific network functions that run on one or more virtual machines (VMs), on top of hardware and the networking infrastructure. Individual VNFs can be connected or combined as building blocks to offer a full-scale networking communication service.

Background

The NFV standards were established, in part, because of a widespread business requirement to remove proprietary hardware solutions that inhibited a company's ability to roll out new services to customers. Modifying or upgrading a hardware-based system was time consuming and prohibitively expensive. Replacing the system could be even more expensive and attempting to integrate components from other vendors often resulted in interoperability issues, all of which resulted in vendor lock-in.

Software-based VNFs were created to provide the same network functionality as their hardware counterparts while reducing the associated costs. New services can be deployed more quickly, for example. However, under the NFV standard a number of vendors deliver devices that require the vendor's own specific Virtual Network Function Manager (VNF-M) or have other restrictions that keep them locked in to that vendor. As a result, operators have been moving from a world of dedicated hardware to a world of dedicated software.

Openet Weaver is the first VNF-M to provide a viable alternative to using multiple VNF-Ms to manage VNFs from multiple vendors. Openet's generic VNF-M has customizable software lifecycle management that allows integration and interoperability with multiple vendors to simplify the management of the network functions for operators. VNF-M, with its customizable software lifecycle management, enables integration and interoperability with multiple vendors to simplify the management of an operator's network functions.

Capabilities

Openet Weaver can be used to perform VNF management tasks including:

- Configuration Management
- Monitoring
- Recovery
- VNF orchestration
- Lifecycle management

Openet Weaver Benefits

Openet Weaver is an ETSI-aligned, generic VNF Manager that is agnostic to the Virtualized Infrastructure Manager (VIM) layer. It provides the following benefits:

- Simplified VNF orchestration management
- Consistent management capabilities for complex NFV ecosystems
- Provides an alternative to vendor lock-in solutions
- The unified VNF management interface helps avoid VNF-M and EMS silos

Typical Virtual Network Function Managers (VNF-Ms) provide integration with the VIM only, which means lifecycle management is driven by virtual machines (VMs). As a result, those VNF managers have a limited set of VIM and VM actions.

Openet Weaver is driven by the VNF software lifecycle. This provides a broader scope of actions. For example, rather than spawning only a new VM instance for in-service upgrade, the integrated VNF-M and Element Management System (EMS) can upgrade software and configurations in the VM. This fine-grain control and visibility over VNFs reduces operation time and complexity when performing such tasks as:

- Installing
- Starting
- Stopping
- Checking Status

Each change to the deployment implemented using Openet Weaver is tracked and version controlled.

Openet Weaver lets you create your own flows without having to upgrade or apply software patches. Openet Weaver pre-installs common lifecycle management flows outlined in NFV including VNF instantiation and health checks. Using a workflow mapping file, an open framework is provided that can be used to integrate new flows into Openet Weaver.

In addition, custom procedures can be created in Openet Weaver so that new operations can be added.

Openet Weaver Features

Openet Weaver has the following features:

- Generic Multi-tenant VNF-M Architecture
- Metadata driven model
- Fine-grained control of VNF package creation
- Flow-driven VNF lifecycle management
- Multiple VNFC deployment on the Same VM
- Converged VNF-M and Element Management System (EMS)

Generic Multi-tenant VNF-M Architecture

Openet Weaver is based on a generic VNF-M architecture that provides the operator's global orchestrator with a single unified interface with which to manage the lifecycle of all the VNFs on the network.

This diagram illustrates a common example of NFV architecture. Although it complies with the ETSI standard, it shows that orchestrators must integrate with separate VNF-M and Element Management System (EMS) silos, which are stacks of VNFs and their proprietary VNF managers.

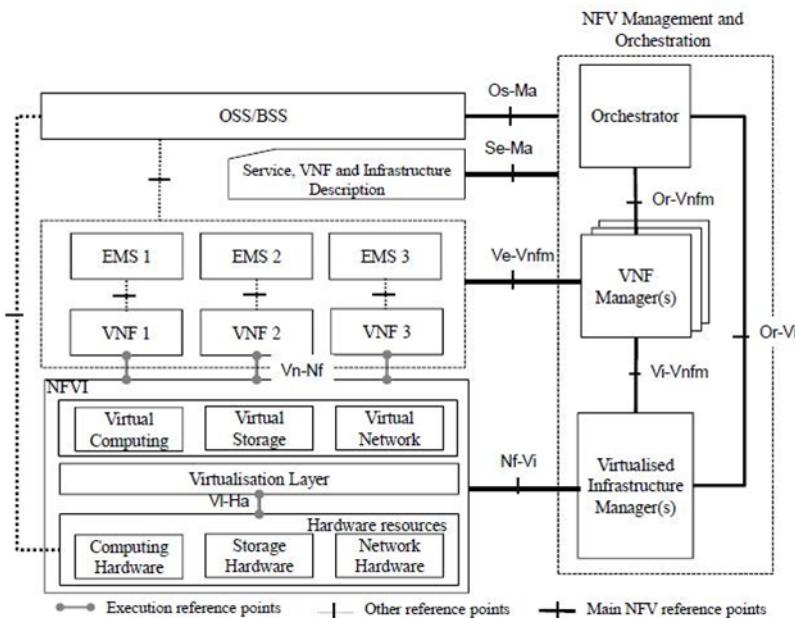


Figure 1: NFV Management and Orchestration Architecture

This diagram shows the more efficient generic VNF-M design provided by Openet Weaver that supports VNFs from multiple vendors and does not require management silos for each vendor.

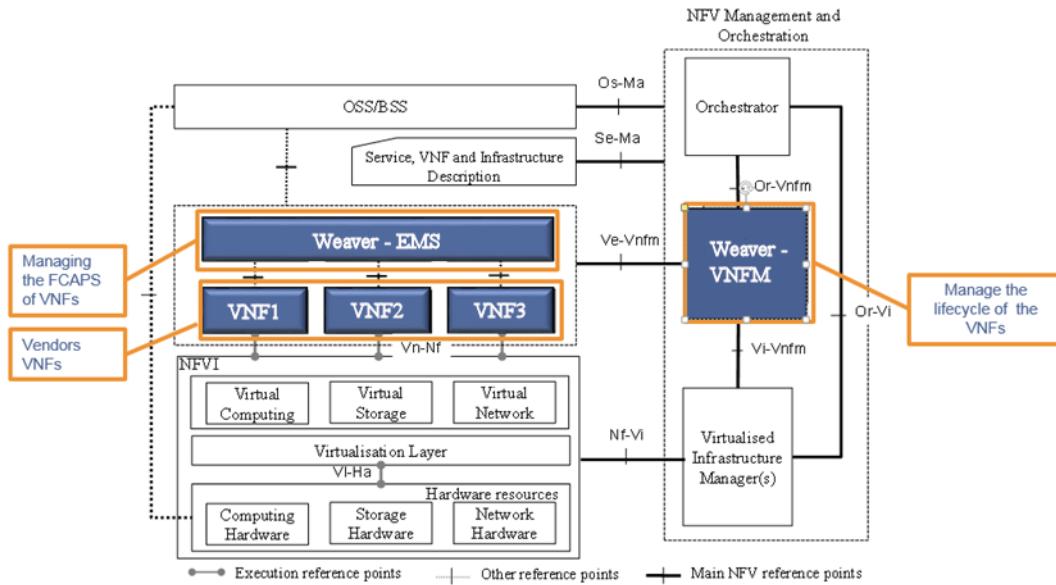


Figure 2: Openet Weaver VNF-M

Openet Weaver is designed for multi-tenancy, which is one of its key attributes for delivering a single, consolidated VNF-M platform to manage VNFs from different vendors. Multi-tenancy allows for a single instance of a software application to serve multiple customers. In the context of Openet Weaver, the VNF-M and the EMS components are the shared services. The tenant is a logical group of VNFs to which a customer or organization requires common, isolated access so that they can manage their VNFs in a production or non-production system. For example, the VNFs might belong to one customer or be managed by one data center.

The following are examples of tenants:

- An MVNO
- A system integrator
- An operator
- A vendor

A multi-tenant VNF-M and EMS provides cost savings over the single tenant (siloeed) set of VNF-M and EMS management systems. In a single tenant architecture, each customer has their own VNF-M and EMS instance that must be operated and maintained. With multi-tenancy architecture, the operator only has to learn, operate, patch and maintain a single system instead of maintaining a number of completely isolated and inherently different management systems. In addition, with multi-tenancy the orchestrator only required to integrate with a single set of APIs, which reduces the time to deliver new services to market.

Openet Weaver is designed to be a reliable, scalable, upgradeable without downtime, secure, and fast multi-tenant system. Tenants can customize each microservice in the Openet Weaver platform through the use of metadata. Microservices can be scaled in or out depending on the number of tenants and lifecycle operations they support.

The APIs work to abstract away the complexity of managing each of the underlying distributed VNFs. This allows the orchestrator to operate on the VNFs as if they were a single entity as opposed to a set of distributed components, which provides a higher level of abstraction. This simplifies VNF orchestration management and creates service chains.

Metadata-driven Model

Openet Weaver uses metadata to provide a flexible VNF-M and EMS layer for multi-tenant VNF lifecycle management. Each VNF tenant provides a configurable metadata set that determines the behaviour of the Openet Weaver microservices for the VNFs they manage. This allows a single flexible VNF-M-EMS to service a wide range of VNF variations.

Each VNF tenant provides metadata for their VNF components, along with providing EMS procedures as part of the VNF package.

Tenant metadata can be modified quickly when there are new requirements. Existing metadata can be used to rebuild a customer's Openet Weaver-VNF-M and Openet Weaver-EMS environment in another system or datacenter.

Metadata can be tracked and version controlled. It is human readable and file based in YML and JSON format.

VNF packages

A VNF package is a logical container for VNF procedures and workflows.

Openet Weaver comes with a number of management procedures that work out of the box. Those procedures can be customized and new ones created to meet tenant VNF requirements. You can create new packages by customizing the package supplied with Openet Weaver to perform different tasks. For example, you can specify which software to install and the order in which the packages install.

Packages allow you to deploy, manage, start, stop, and troubleshoot at VNF level. Configuration is maintained in a single archived file. They are used in conjunction with the workflows described below to perform automated VNF-M tasks.

Packages are version controlled, making it possible to roll back to a previous version if required.

Flow-driven VNF Lifecycle Management

Each distributed VNF lifecycle operation is executed in Openet Weaver using a workflow. A workflow is a logically linked sequence of steps where each step is an Openet Weaver API call. Each API call represents an action with an input and an output that indicates success or failure. If a step is successful the flow proceeds to the next step. If the execution of the step results in failure an action can be made to pause the flow, rollback the flow or retry the step.

EMS Designed for NFV

The key functionality of the Openet Weaver Element Management System (EMS) is divided into five key areas:

- Fault
- Configuration
- Accounting
- Performance
- Security

Collectively these areas are referred to as FCAPS.

Note: Openet Weaver v1.11 supports the fault, configuration, performance, and security aspects of FCAPS.

The EMS architecture is driven completely by APIs that are designed specifically for network function virtualization in order to facilitate the automation of VNF lifecycle operations. The primary purpose of the Openet Weaver EMS is to present northbound APIs to the VNF-M so that they can be used within a VNF lifecycle workflow. The EMS is also responsible for continuously monitoring the performance and faults of each VNF component after deployment.

Multiple VNFC Deployment on the Same VM

Openet Weaver supports one-to-one mapping between a VNFC instance and a VM, which is the standard deployment model in a cloud environment. Openet Weaver also supports running multiple VNFC instances on the same VM, which can reduce the performance overhead because there are fewer VMs running in a private, virtualized environment.

Note: Multiple VNFCs can be run on a single VM only for VNFs without VIM configuration.

Converged VNF-M and EMS

Openet Weaver provides a tightly coupled and converged VNF-M and EMS platform. This convergence allows for more seamless management of VNFs. The VNF-M Workflow Engine is pre-integrated with all of Openet Weaver EMS APIs. This facilitates VNF lifecycle management beyond the capabilities of a traditional, loosely coupled VNF-M and EMS system.

The converged system makes a wider range of EMS FCAPS capabilities available to the Openet Weaver VNF-M, and so providing tenants with more granular control and visibility into the health of their VNFs when creating VNF lifecycle flows.

The Workflow Engine has out of the box flow operations for the following:

- Instantiating a VNF, including updating and starting
- Starting a VNF
- Checking whether a VNF is up
- Upgrading a VNF
- Rolling back an upgraded VNF
- Updating a VNF
- Checking the performance health of a VNF
- Healing a VNF
- Stopping a VNF
- Terminating a VNF
- Starting a VNFC
- Upgrading a VNFC
- Updating a VNFC
- Rolling back a VNFC
- Stopping a VNFC

Openet Weaver Architecture

The Openet Weaver VNF-M product architecture is illustrated below.

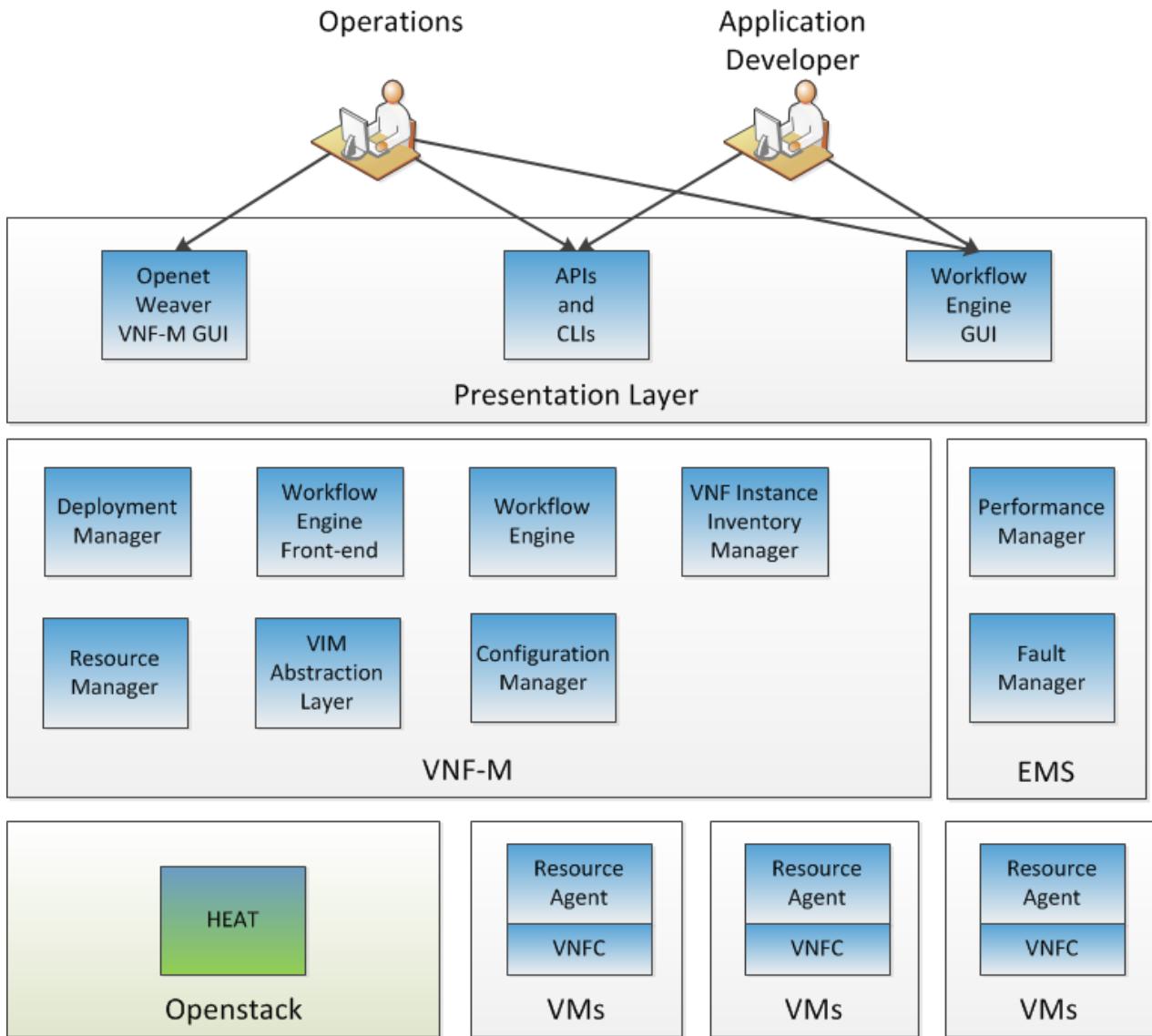


Figure 3: Openet Weaver VNF-M Architecture

The architecture is mainly composed of interconnected microservices, which are components that perform specific functions within Openet Weaver and communicate with each other through API calls. Microservices can be deployed and scaled out independently without impacting other services. Openet Weaver is integrated with a Workflow Engine to provide user interfaces for package development and workflow management.

The Openet Weaver architecture is described in the following sections:

- Presentation Layer
- Virtual Network Function Manager (VNF-M)
- Element Management System (EMS)
- OpenStack
- Virtual Machines (VMs)

Presentation Layer

The presentation layer provides the following interfaces with which to interact with Openet Weaver:

- An Openet Weaver VNF-M GUI for managing VNFs
- CLI and API commands for working with:

- The Workflow Engine
- Configuration Manager
- Performance Manager
- Fault Manager

For example, the API commands can be used to connect a third-party orchestrator.

- A GUI that provides an interface to the Workflow Engine

Virtual Network Function Manager (VNF-M)

The Openet Weaver VNF-M consists of the following microservices:

- Deployment Manager
- Workflow Engine Front-end
- Workflow Engine
- Resource Manager
- Configuration Manager
- VNF Instance Inventory Manager
- VIM Abstraction Layer

Deployment Manager

The Deployment Manager is used to upload artifacts that perform the Openet Weaver Installation. Therefore it is the first microservice installed when you deploy Openet Weaver. Deployment Manager is a mandatory microservice.

Note: Deployment Manager is currently used to install Resource Agents only.

Workflow Engine Front-End

The Workflow Engine Front-End is a facade layer that provides a generic interface between the Workflow Engine and NFV orchestrators or clients. It provides REST and CLI APIs.

The Workflow Engine Front-End is responsible for creating workflow instances. Workflows are structured sequences of actions to be performed on VNF nodes. Metadata stored in YML files is used to map a given workflow to a specific tenant, VNF and package.

Although the Workflow engine front end is used to create a workflow instance, the Workflow Engine is responsible for running the workflow.

Openet Weaver provides the following predefined workflows:

- Instantiate, which includes updating and starting a VNF
- Updating a VNF
- Upgrading a VNF
- Rolling back a VNF
- Healing a VNF
- Starting a VNF
- Stopping a VNF
- Checking whether a VNF is up
- Terminating a VNF

Workflow Engine

The Workflow Engine enables the automation of VNF management tasks by running workflows when a request to do so is received from the Workflow Front End. The Workflow Engine coordinates tasks through REST and HTTP calls to the Resource Manager and the Configuration Manager. A workflow is run for each node in the system being managed by Openet Weaver.

The Workflow Engine stores the predefined workflows that are provided with Openet Weaver. The Workflow Engine associates the appropriate package with the workflow it is running, based on the package ID associated with the instance of the workflow created by the front end.

The Workflow Engine GUI provides an interface for running workflows.

Resource Manager

The Resource Manager is an open and flexible task execution framework. It takes actions that originate in the Workflow and passes them to the Resource Agents for execution. It also sends the response from the Resource Agents back to the Workflow Engine after the action is complete.

The following are examples of requests that can be sent to the Resource Manager through the API interface:

- Installing software
- Updating configuration
- Upgrading software
- Start VNF components
- Stop VNF components
- Deleting VNF components

Note: The examples include out of the box actions as well as custom actions.

Configuration Manager

The Configuration Manager provides tenant and VNF management from a VNF repository perspective. It is used for storing and managing packages and the related information, which is stored in an internal VNF repository at file-category level as shown in Configuration Manager Directory Structure. The repository includes the following types of information:

- VNF topology—the VNF Topology file is a YML format file that defines which VNFs and VNF instances are deployed to which node in the configuration.
- Procedures—Openet Weaver uses out of the box procedures. Procedures are written in Python. The out of the box procedures include `dynamic_config_vnfc_v2.py`, `is_vnfc_down_v2.py`. Customisation of procedures is done by overriding the supplied procedures.
- Configuration Files—configuration files are application specific files. For example `httpd.conf` can be supplied in `jinja2` format to Openet Weaver as part of the dynamic configuration procedures. They are then localised on the runtime node.
- Metadata—metadata is supplied in YML format. This file defines procedure definitions, VNFC package names, service names, configuration files and parameters of the VNFC.
- VNFC Packages—software packages can be loaded as RPM's to the VNFC configuration to be installed on the runtime nodes.

Configuration Manager allows multiple VNF package developers to collaborate in development. packages are versioned—developers can see who created or modified a package and when it happened.

VNF Instance Inventory Manager

VNF Instance Inventory Manager is used to persist VNF instance information over the VNF instance lifecycle. It stores data about the state of the VNF instance, for example whether the VNF is instantiating, termination active or inactive. It also stores health check information.

VIM Abstraction Layer (VAL)

The Openet Weaver VIM abstraction Layer (VAL) provides a bridge between the orchestrator and the VIM. The VAL is designed to perform the following functions:

- Read, validate, and parse Tosca VNF templates, also known as VNF descriptors (VNFDs)
- Convert a Tosca template to a VIM's implementator template, such as HOT (Heat Orchestration Template) that is specific to OpenStack.
- Manage VIM through API or CLI commands. For example, you can:
 - Create and delete stacks
 - Set up virtualization deployment unit (VDU) specifications related to scaling, for example the dimension of the CPUs, storage space, and the amount of RAM
 - Create network endpoints
 - Set up connectivity between VNFs or VDUs

- Set up security levels
- Instantiate VNFs

The VAL provides the workflow Engine with a generic API to manage virtual infrastructure resources (compute, network and storage). This allows for the backend VIM to change without requiring an update to the workflow Engine flows.

Element Management System (EMS)

The Openet Weaver EMS consists of the following microservices:

- Performance Manager
- Fault Manager

Performance Manager

Performance Manager is responsible for collecting Openet Weaver performance metrics and querying performance health. Performance Manager can also be used to trigger scalability workflows under certain circumstances, such as when CPU usage or the TPS rate remains consistently high for a specified time period. Performance checks can be scheduled to run at regular intervals.

Note: Scalability flows will be available in a future release of Openet Weaver when the VIM Abstraction Layer is integrated.

A performance policy can be specified on a per-VNF basis and stored in Configuration Manager along with the other VNF data. Captured performance metrics are stored in Performance Manager.

Fault Manager

Fault Manager is the Openet Weaver SNMP manager. SNMP enabled VNF components send SNMP alerts (trap messages) to Fault Manager. In some cases this can trigger the Openet Weaver out of the box auto-healing workflow, which attempts to correct the fault.

Openet Weaver supports granular traps, each of which have a unique object identifier (OID) that allows the SNMP manager to distinguish the SNMP traps. Human-readable OID information is stored in a Management Information Base (MIB) file. Fault Manager accesses this file when handling traps sent by the SNMP agent.

Virtual Machines (VMs)

Virtual Network Function Components (VNFCs) instantiated in Openet Weaver are run on VMs that are nodes within the system.

Openet Weaver requires the deployment of a Resource Agent on each VNF node in the system. Resource Agents execute work flow steps on business processing nodes based on the jobs they receive from the Resource Manager. For example, pulling software and configuration files from the vnf repository on the Configuration Manager.

Resource Agent

Each Resource Agent exposes asynchronous API interfaces to the Resource Manager so that they can communicate about the lifecycle operations to perform on the VNFs on a VNF runtime node based on the pre-defined procedures from the VNF package associated with the current workflow. The agent keeps track of job status, such as whether it is running, completed, or whether it failed before completing. The resulting output is returned to the Resource Manager, passed back through the Openet Weaver microservices, and returned to you as feedback.

OpenStack

OpenStack is the Virtualized Infrastructure Manager (VIM) supported by Openet Weaver. The VIM is responsible for controlling and managing network resources within an operator's infrastructure domain. HEAT is an OpenStack service that orchestrates multiple package-based VNFs.

Note: OpenStack is a third-party VIM. Openet Weaver is integrated with Openstack releases Kilo, Liberty and Mitaka.

Openet Weaver Call Flows

Openet Weaver is capable of running workflows with and without a Virtualized Infrastructure Manager (VIM).

Typical Instantiation Workflow without VIM

The following us a simplified instantiation workflow without VIM.

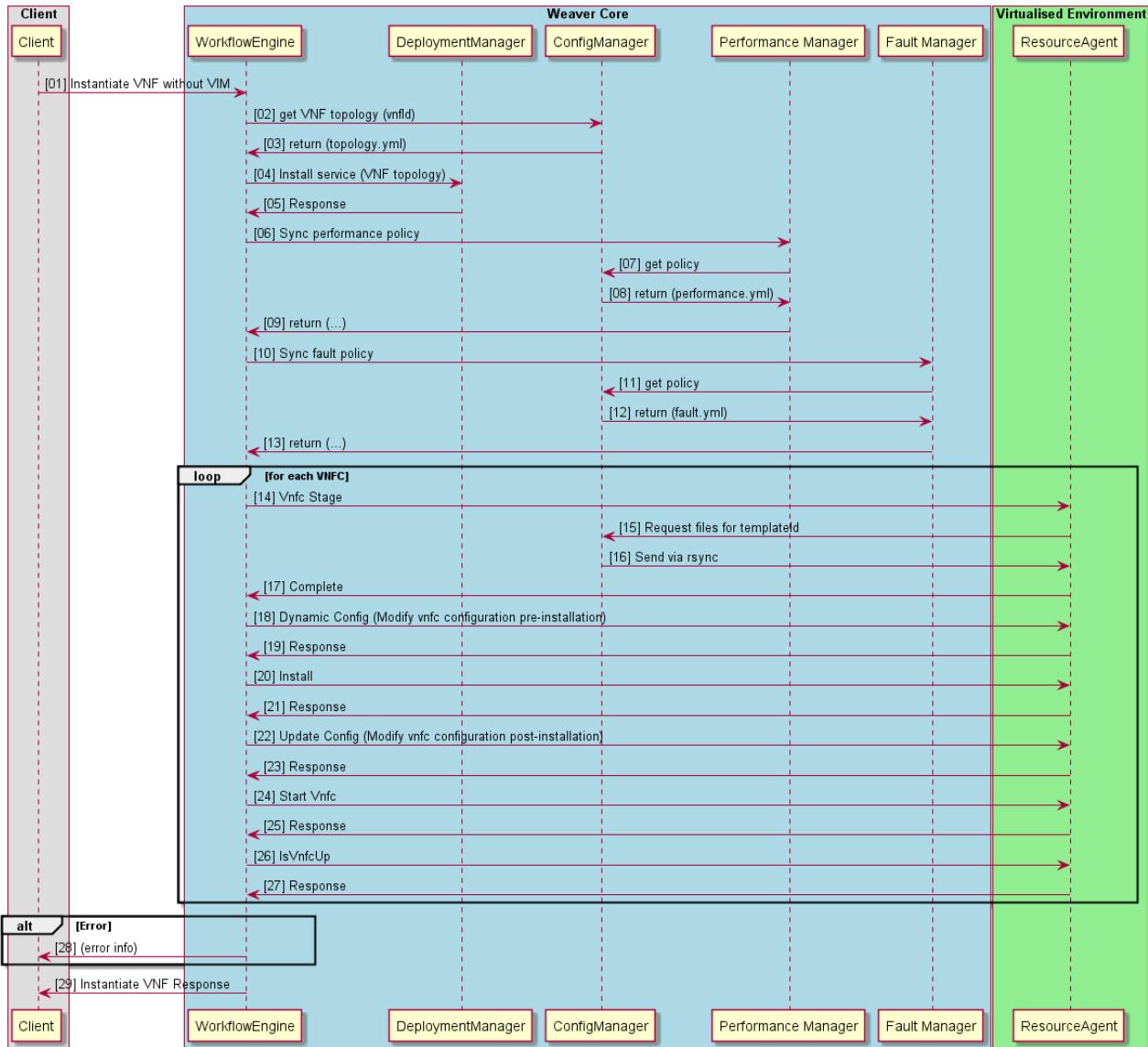


Figure 4: VNF-Level Instantiation

Typical Instantiation Workflow with VIM

The following is a simplified instantiation workflow with VIM.

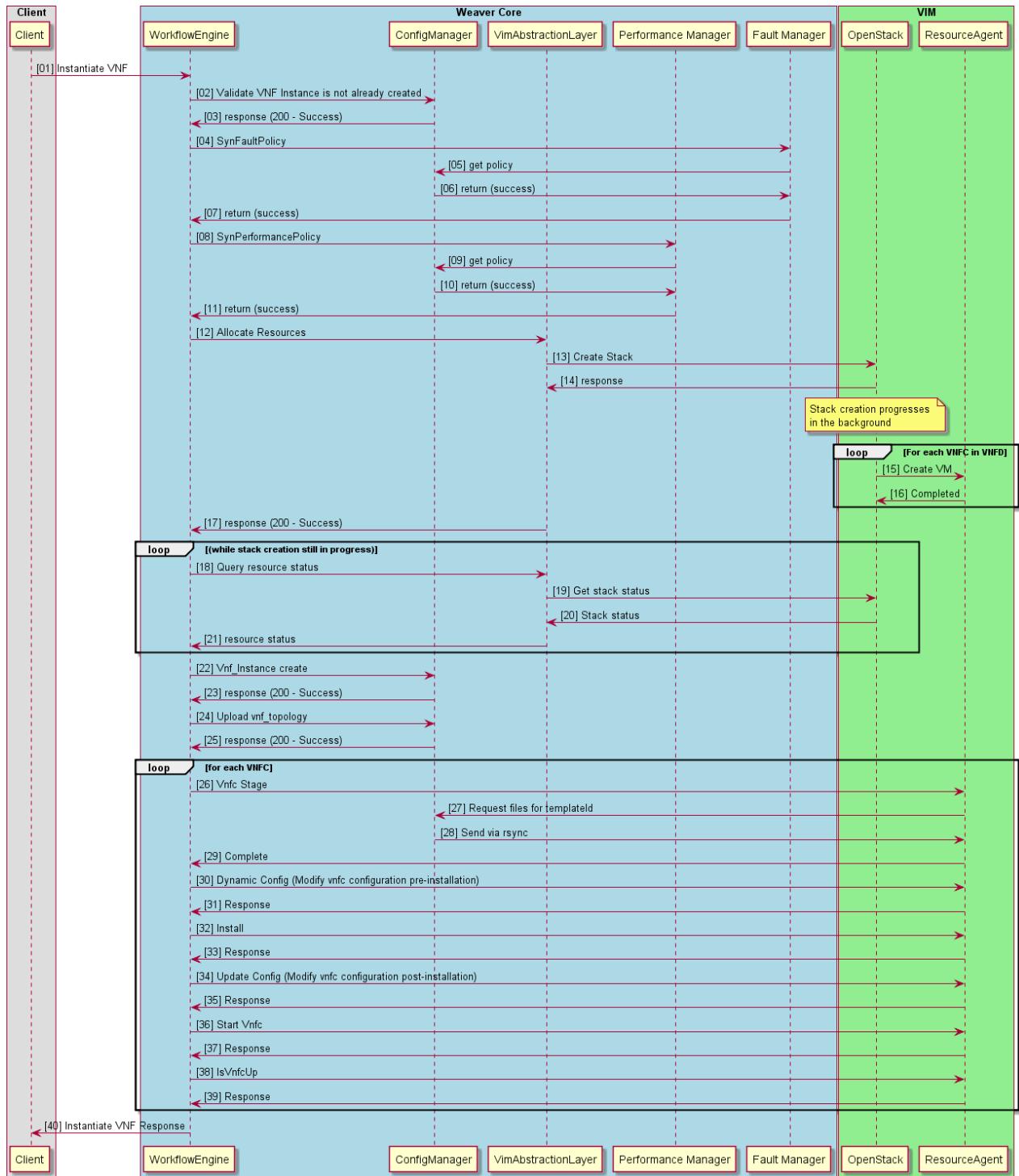
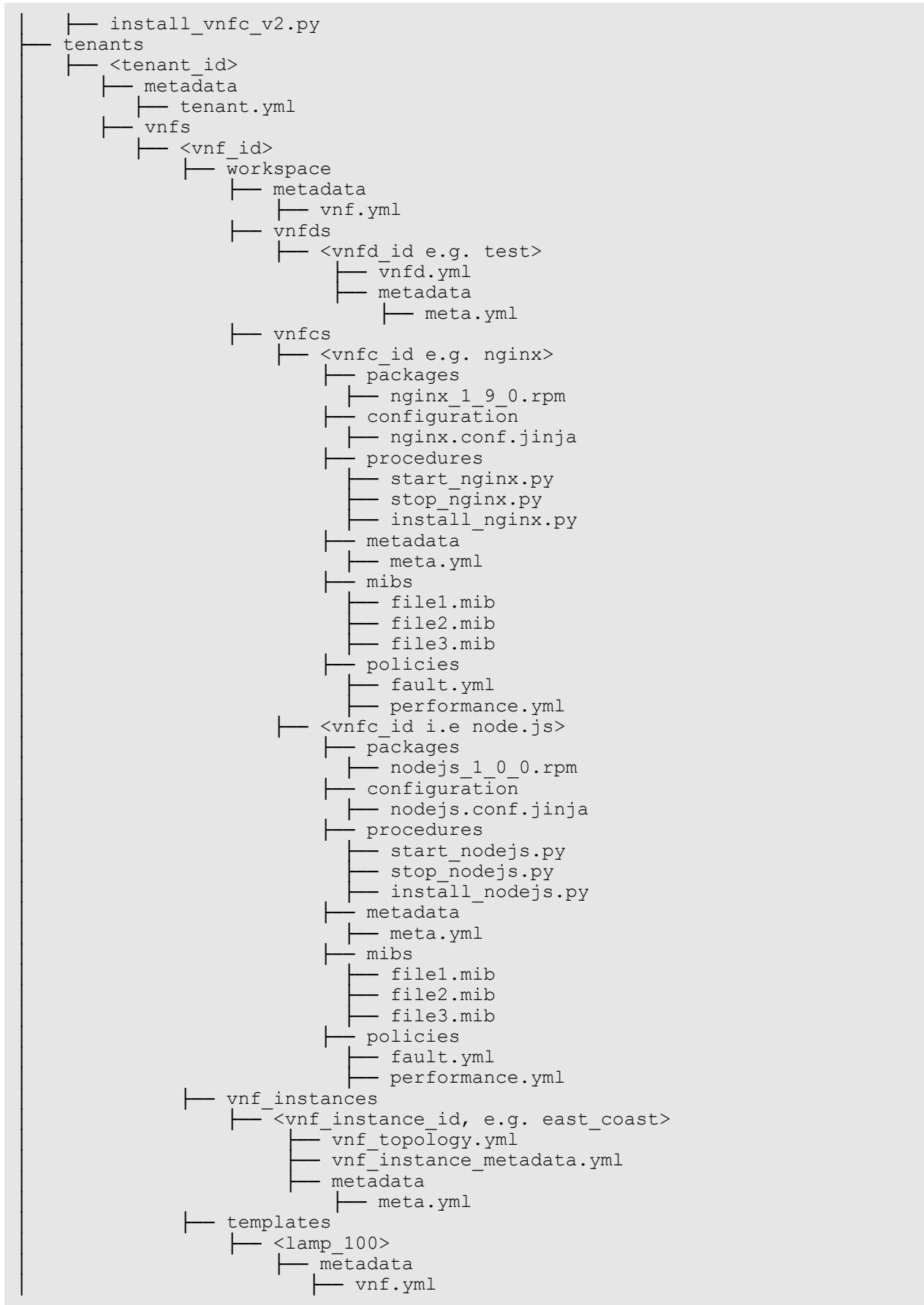


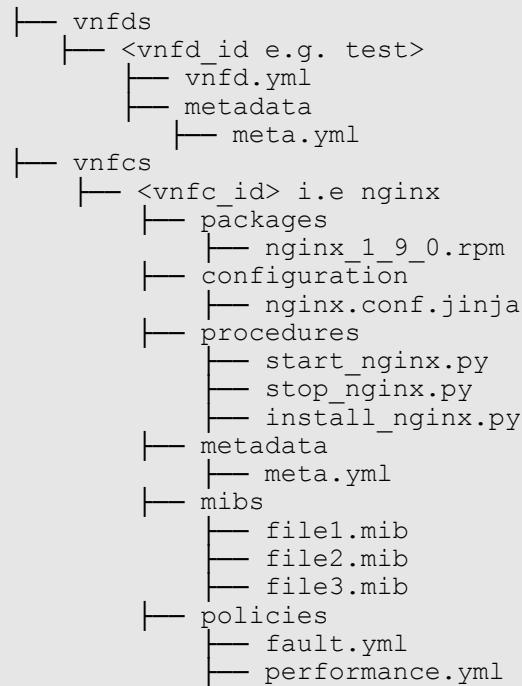
Figure 5: VM-Level Instantiation

Configuration Manager Directory Structure

Configuration Manager stores VNF packages and package-related information in the directory structure shown below:

```
$RepositoryRoot
  ootb_procedures
    start_vnfc_v2.py
    stop_vnfc_v2.py
```





Openet Weaver Use Cases

This topic shows the main Openet Weaver use cases and associates them with the role that is most likely to perform them.

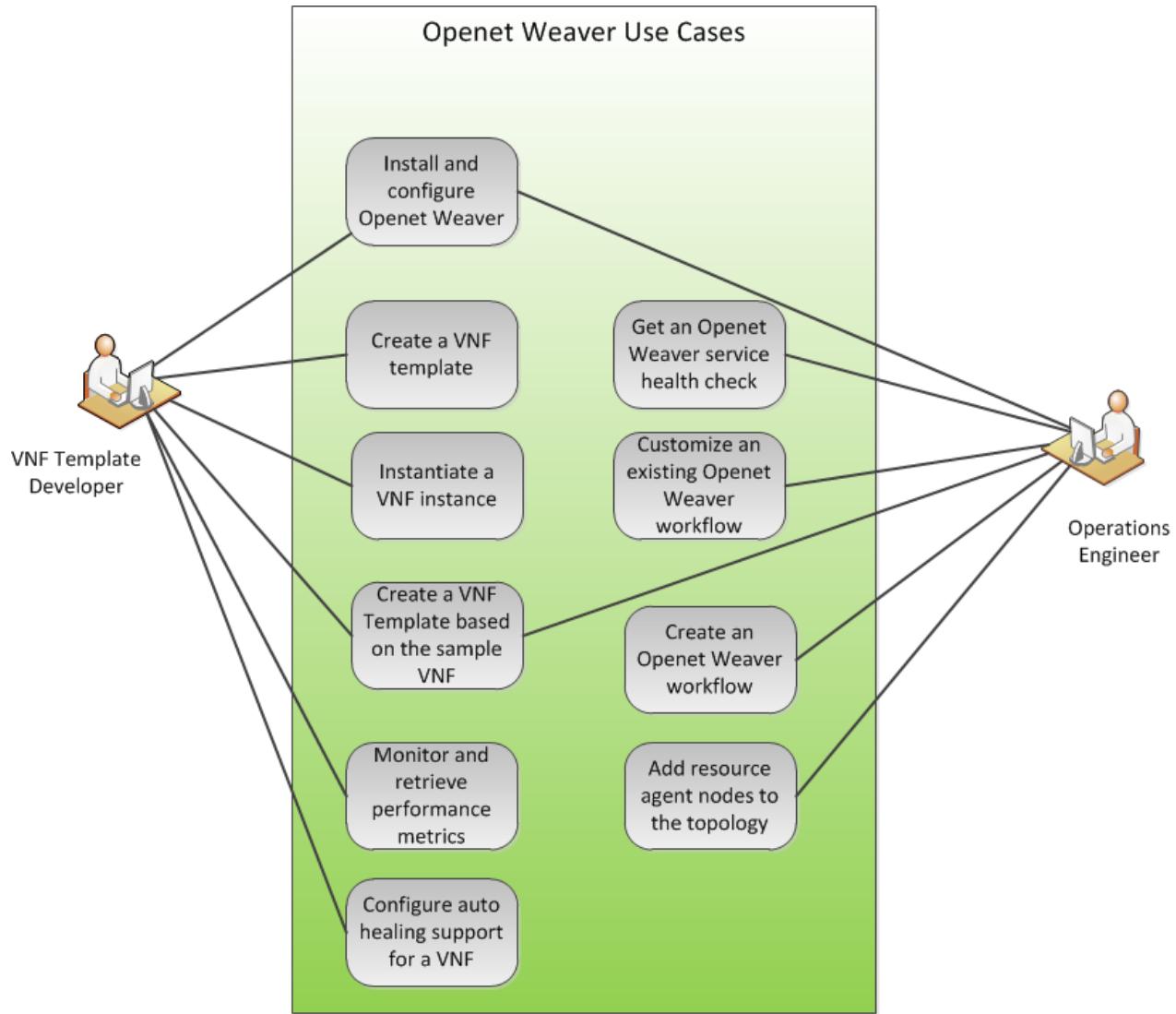


Figure 6: Openet Weaver Use Cases

The use cases are detailed in the following sections of this webhelp:

- Install and configure Openet Weaver—see [Installing Openet Weaver in a Lab Environment](#)
Note: See the [Quick Start Guide](#) for installing Openet Weaver in the predefined VM environment
- Create a VNF package—see [Creating VNFs](#)
- Create a VNF package based on the sample VNF—see [Example VNF](#)
- Instantiate a VNF instance—see [Running a Workflow](#)
- Monitor and retrieve performance metrics—see [Managing Performance](#)
- Get an Openet Weaver service health check—see [Running the Health Check](#)
- Customize an existing Openet Weaver workflow—see [Modifying Existing Workflows](#)
- Create an Openet Weaver Workflow—see [Creating New Workflows](#)
- Add new resource agent nodes to the topology—see [Creating an Install Topology File](#)
- Configure auto healing support for a VNF—See [Configuring Auto Healing Support for a VNF](#)

Openet Weaver Quick Start Guide

The Quick Start Guide shows you how to get started quickly with the pre-configured virtual machine and a sample VNF configuration.

This guide provides you with all the details required to get started quickly with Openet Weaver. You can download a pre-configured virtual machine image, install Openet Weaver, create a VNF package from the supplied workspace and instantiate the VNF on the virtual machine.

The process has the following steps:

- Downloading the virtual machine—an OVA image runs in the Oracle virtual box and is used to allow you to evaluate Openet Weaver
- Installing Openet Weaver—download and deploy the Openet Weaver software package
- Running the Health Check—verify the deployment by running the provided system health check
- Deploying the sample VNF—create a VNF package and instantiate it on the virtual machine

Download and install the Virtual image

Installing the sample virtual machine image to test Openet Weaver on a laptop.

Before downloading, ensure your computer meets the system requirements. The following table outlines the system requirements hardware and software required to run the virtual machine image.

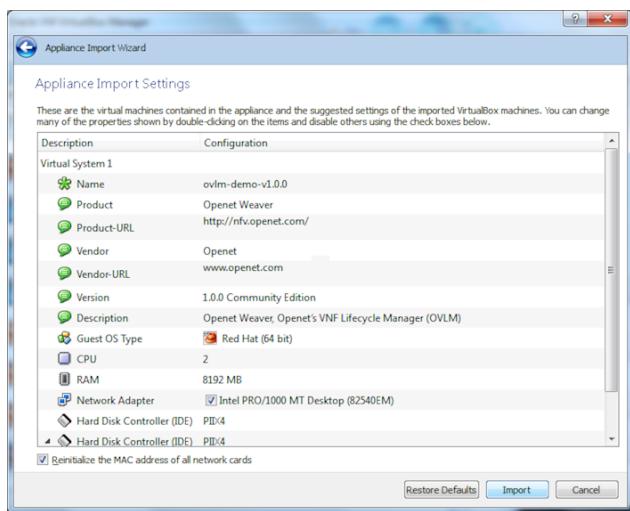
Table 5: System Requirements

Requirements	Description
System RAM	Minimum 12GB, 8GB required for VM image
CPU	Intel x64 quad core 2.1Ghz or higher
Disk space	40 Gigabytes of free disk space
Host Operating System	Windows or Linux 64 bit Operating System
Oracle Virtual Box	Oracle virtual box 4.1.18
Supported Browsers	Microsoft Internet Explorer 11.x Mozilla Firefox (latest) Google Chrome (latest)
OpenStack	The Virtualized Infrastructure Manager (VIM) supported by Openet Weaver. This is required only if you intend to use VIM to manage your VNFs. Openet Weaver v1.11 is tested against OpenStack Liberty.

To download and install the virtual image, perform these steps:

1. Download the Quick Install VM image ovlm-demo-<release_version>-demo.ovf from <http://accelerate.openet.com/weaver>.
2. In Oracle Virtual box select **File > Import Appliance**.
3. In the import wizard, select **choose / open appliance** and browse to the VM image file.
4. Click **Open**.
5. Click **Next**.

- Review the import settings (be sure to select the check box reinitialize the MAC address of all network cards), and click **Import**.



- Select the imported Virtual Machine in the virtual box windows and click **Start**.

The virtual machine window is displayed and you can log in.

Logging on to the Virtual Machine

- To access the virtual machine, enter the user name and password:

```
ovlm-demo login: root
password: weaver
```

Note: The virtual machine enables port forwarding to access the virtual machine from your host operating system. You can ssh to the local host on port 2222 and log in as above.

```
ssh root@localhost -p 2222
```

Note: The VM keyboard is US mapped. To change to UK key mapping run the following command: `localectl set-keymap uk`

Install OVLM

Installing Openet Weaver on the Virtual image

Installing Openet Weaver on the sample virtual machine

Before proceeding with the installation, ensure your computer has access to the internet as the installer downloads the latest version of Openet Weaver.

Note: Access to the base yum repository is required for dependencies i.e rsync.

- Download the Openet Weaver package to the virtual machine by running wget on the Openet Weaver-demo node:

```
wget http://accelerate.openet.com/ovlm/1.6/ovlm-core-install-community-<release_version>.tar
```

- A third party package is already on the virtual machine, untar both packages to the same folder on the file system using the following commands. The path used for this is /root/ovlm-thirdparty-dependencies.

```
tar -xvf ovlm-core-install-community-<release_version>.tar
tar -xvf ovlm-thirdparty-dependencies-<version_number>.tar
```

3. Change in to the install directory created by extracting the tar files.

```
cd ovlm-install
```

4. Update the openstack: credentials section of the ovlm-install-sample.yml file with the properties described in the table below. OpenStack must be installed and setup correctly before you can perform this step. Refer to the Openstack documentation for more information.

Note: This step is optional if Weaver is not being used to communicate with OpenStack to manage virtual machines.

Properties	Description
tenant	The OpenStack project/tenant that is used by Openet Weaver.
username	The OpenStack project username for authentication.
password	The OpenStack project user password for authentication.
auth-url	<p>The OpenStack Identity Service's endpoint. You can find the value for this property in the OpenStack Dashboard (Horizon) in the Identity field on the API tab of the Access & Security page as shown below this table.</p> <p>Note: This property is required only if you have installed OpenStack to use with Openet Weaver.</p>

Service	Service Endpoint
Compute	http://10.0.124.61:8774/v2/25179b6bf9534e8a9b58d8de3b0dad5c
Network	http://10.0.124.61:9696
Volumev2	http://10.0.124.61:8776/v2/25179b6bf9534e8a9b58d8de3b0dad5c
Computev3	http://10.0.124.61:8774/v3
S3	http://10.0.124.61:8080
Image	http://10.0.124.61:9292
Cloudformation	http://10.0.124.61:8000/v1
Volume	http://10.0.124.61:8776/v1/25179b6bf9534e8a9b58d8de3b0dad5c
EC2	http://10.0.124.61:8773/services/Cloud
Orchestration	http://10.0.124.61:8004/v1/25179b6bf9534e8a9b58d8de3b0dad5c
Object Store	http://10.0.124.61:8080/v1/AUTH_25179b6bf9534e8a9b58d8de3b0dad5c
Identity	http://10.0.124.61:5000/v2.0

Figure 7: OpenStack Dashboard Access & Security

5. Encrypt the passwords defined in the in the ovlm-install-sample.yml file using the following command

```
./utilities/cipher/encrypt_scripts/encrypt-password.sh -f  
ovlm-install-sample.yml
```

6. Execute the installer using the following command

```
./ovlm-deploy.sh -u root -i ovlm-install-sample.yml --create-sample-vnf
```

The following table describes the command parameters.

Table 6: Installation parameters

Parameter	Description
-i [Install properties file]	A file that defines the installation properties and topology. In other words, it defines which services are installed on each node. A sample file is provided in the extracted ovlm-core-install.tar file. It is called ovlm-install/ovlm-install-sample.yml
-u [Installation user]	The user the installation runs as. The user must have sudo access or be the root user. The user cannot be called ovlm because the installer creates an ovlm user during installation.
--create-sample-vnf	Creates the sample package (microblog_v1) that is used to instantiate the sample VNF used in this section of the webhelp.

Note: There is also a --skip-local-cli-install option. However, this option cannot be used together with --create-sample-vnf parameter because --create-sample-vnf might not work if the Configuration manager CLI is not installed on local machine.

The Openet Weaver installer connects to each node and installs the services define in the Openet Weaver install properties file, this might take a number of minutes, the user sees a successful disconnect from each node when complete as shown below:

```
Disconnecting from node-3... done.
Disconnecting from node-1... done.
Disconnecting from node-2... done.
Disconnecting from node-6... done.
Disconnecting from node-4... done.
Disconnecting from node-5... done.
```

Next [Running the Health Check](#) on page 36

7. Prepare a VNF topology file.

The following is an example of a VNF topology file:

```
vnfc_ids:
blogserver:
- hostname: 192.168.122.249
server_name: ra1.openet-telecom.lan
vnfc_instance_id: 1
- hostname: 192.168.122.248
server_name: ra2.openet-telecom.lan
vnfc_instance_id: 2
nginx:
- hostname: 192.168.122.34
server_name: ra3.openet-telecom.lan
vnfc_instance_id: 1
postgres:
- hostname: 192.168.122.66
server_name: ra4.openet-telecom.lan
vnfc_instance_id: 1
```

8. if you have enabled security authentication, log in to the authentication service by entering the following command:

```
eval $(on-auth-client -u <username> -p <password>)
```

9. Upload the VNF topology file you created in step 7 to Configuration Manager by entering the following command:

```
ovlm-cm topology upload -t default -v microblog -i dc_1 -f vnf_topology.yml  
-p microblog_v1
```

Running the Health Check

A health check flow is provided that can be executed post installation to verify the Openet Weaver services are running.

Now that you have a successful installation, you can verify that everything is running as expected. You can do this by running the health status command from the Front End CLI or by performing a health check using the Workflow Engine GUI.

Note: You can also perform a system health check in the VNF-M GUI. See [Checking System Health Status](#) for more information.

Performing a System Health Check from the CLI

You can perform a system health check by running the health status command from the CLI.

Before running the health status command you need to set the Front End baseurl environment variable as shown below:

```
[root@ovlm-demo ~]# export OVLME_BASEURL=http://node-1:28050
```

Then you can check the health status:

```
[root@ovlm-demo ~]# ovlm-fe health status
```

The following is an example of the resulting output:

```
data:  
  flow_result:  
    Output:  
      services:  
        - instances:  
          - status: running  
            url: http://node-1:28020/health  
            service_id: resource_manager  
        - instances:  
          - status: running  
            url: http://node-1:28100/health  
            service_id: fault_manager  
        - instances:  
          - status: running  
            url: http://node-1:28050/health  
            service_id: frontend  
        - instances:  
          - status: running  
            url: http://node-1:28010/health  
            service_id: configuration_manager  
        - instances:  
          - status: running  
            url: http://node-2:28030/health  
          - status: running  
            url: http://node-3:28030/health  
          - status: running
```

```

    url: http://node-4:28030/health
    - status: running
      url: http://node-5:28030/health
      service_id: resource_agent
    - instances:
      - status: running
        url: http://node-1:28090/health
        service_id: performance_manager
      - instances:
        - status: running
          url: http://node-1:28120/health
          service_id: instance_inventory_manager
method: GET
request: http://node-1:28050/api/v2/health/status
workflow_instance_status: COMPLETED
status: 200

```

Performing a System Health Check from the Workflow Engine GUI

To perform a system health check from the Workflow Engine GUI, follow these steps:

1. To run the health check from the Workflow Engine GUI, open your supported browser and enter the following URL: **http://Workflow_Engine_host:8089/**.
 2. Click **Run Management**.
 3. Expand the Library tree by clicking **Flow Launcher > Library > Openet > OVLM > HealthCheck > CheckAllHealthServices**.
 4. Check the box **Open Run After Launch** and click the **Run** button on the bottom right.
- The result of the health check shows all running services.

Drill Down

CheckAllHealthServices Completed – Resolved ✓

Step Name	Transition Message
Get URLs from se...	success
Check Health Ser...	success
Intermediate Rep...	success
Service Iterator	has more
Get URLs from se...	success
Check Health Ser...	success
Intermediate Rep...	success
Service Iterator	has more
Get URLs from se...	success
Check Health Ser...	success
Intermediate Rep...	success
Service Iterator	no more
Report services	success
Resolved : success	

Flow Graph **Step Details**

Resolved : success

Run ID: 113402494
Step ID: 6126487a-cdec-459b-9869-9c299571ae73
Start Time: 11:28:24 AM
End Time: 11:28:24 AM
Response: N/A
Duration: 0.001 seconds
Primary Result: N/A
Step Results: Output {"services":[]}
Worker Group: N/A
Worker ID:
Transition Mess...

[Show Full Tracking...](#)

5. Connect to one of the runtime nodes and stop runtime agent services.

```

ssh node-5
systemctl stop ovlm-ra

```

6. Execute step 4 again with the services stopped. You see a failure on the stopped node displayed. This screen shows the result of the health check showing a service that is down.

Step Name	Transition Message
Get URLs from se...	success
Check Health Ser...	success
Intermediate Rep...	success
Service Iterator	has more
Get URLs from se...	success
Check Health Ser...	success
Intermediate Rep...	success
Service Iterator	has more
Get URLs from se...	success
Check Health Ser...	success
Intermediate Rep...	success
Service Iterator	no more
Report services	failure
Error : failure	

7. Restart the runtime agent service on node-5 by executing the following command:

```
systemctl start ovlm-ra
```

Next [Example VNF](#) on page 38

Example VNF

Out of the box, the installation provides a microblog server as a VNF. This VNF consists of one load balancer, two application servers and one postgres database node. The VNF can be deployed with or without a Virtualized Infrastructure Manager (VIM) configuration depending on whether you decide to allow VM management capabilities such as the allocation of specific hardware resources during instantiation.

The microblog VNF workspace is pre-loaded under `/usr/lib64/ovlm/cm/repository-root/tenants/default/vnfs/microblog/workspace`. This exists on node-1 where configuration manager is installed. It has three VNFCs defined:

- blogserver
- nginx
- postgres

Important: The PostgreSQL database is installed using YUM installation during instantiation, upgrade, and rollback workflows. This requires Openet Weaver to establish a connection to the YUM repository. Ensure that you have Internet connectivity before running those workflows.

The structure of the VNF configuration is shown below:

Note: This microblog server is an example only and is **not** suitable to deploy in production environments.



The VNFC consists of configuration information, metadata, VNFC packages, and procedures required for each VNFC.

Table 7: VNFC Configuration details

Item	Description
packages	This location holds the install VNFC packages (RPMs and DEBs) for the VNFCs.
configuration	This location holds the application configuration files in jinja2 format that are localized when deployed on the run time nodes.
procedures	This location holds the custom Python procedures that can be used to override the Openet Weaver out of the box procedures.

Item	Description
metadata	This location holds VNFC meta data that defines customization and parameters in YML format.
mibs	This location holds the Management Information Base (MIB) files.
policies	This location holds the fault management and performance management files.

The VNF configuration also consists of an instance that defines the topology of the deployment. The instance and topology file are required when deploying a vnf without VIM configuration. When deploying with VIM integration, a VNF descriptor (VNFD) is required. With VIM configuration, instances and their topology files are generated automatically during instantiation.

Table 8: Description of Nodes

Node	Description
ovlm-demo	Base operating system of the VM that holds the containers.
node-1	LXC container that holds the front-end, configuration manager and resource manager microservices you must be connected to for this node to run CLI commands.
node-2	LXC container that holds a runtime node where nginx is deployed.
node-3	LXC container that holds a runtime node where the micro blog app server is deployed.
node-4	LXC container that holds a runtime node where the second micro blog app server is deployed.
node-5	LXC container that holds a runtime node where the postgres database is deployed.
node-6	LXC container that holds the Workflow Engine component.

Deploy Example VNF without VIM

Deploying the configured VNF from the CLI

To deploy the VNF, create a package from the workspace and instantiate the package. The below steps show examples of packages using the out of the box package to deploy the VNF.

- Set the Front end baseurl environment variable to be OVLMFE_BASEURL=http://node-1:28050

```
[root@ovlm-demo ~]# export OVLMFE_BASEURL=http://node-1:28050
```

- Set the Configuration Manager baseurl environment variable to be OVLMCM_BASEURL=http://node-1:28010

```
[root@ovlm-demo ~]# export OVLMCM_BASEURL=http://node-1:28010
```

- Create a package of the out of the box workspace using the following command ovlm-cm template create -t <tenant_id> -v <vnf_id> -p <vnf_template_id>

```
[root@ovlm-demo ~]# ovlm-cm template create -t default -v microblog -p microblog_v1
```

4. Instantiate the VNF by calling the instantiate_vnf_without_vim flow with the command `ovlm-fe workflow submit -t <tenant_id> -v <vnf_id> -l <vnf_instance_id> -y <workflow_type> -p <vnf_template_id> -sync <1 | 0>`

```
[root@ovlm-demo ~]# ovlm-fe workflow submit -t default -v microblog -i sample
-y instantiate_vnf_without_vim -p microblog_v1 -sync 1
```

Note: The optional sync parameter specifies whether the API call is synchronous. The call is synchronous If set to 1 as in the above command, and the API call waits to execute until the end of the current workflow run or until the timeout value specified by the synchronous-request-delay option in the front-end configuration field. If sync is set to 0 the call is asynchronous, which is the default setting..

Command result:

```
workflow_instance_id: '101100243'
[root@node-1 ~]#
```

Note: The workflow submit command returns a `workflow_instance_id` which can be used to check the status of the flow.

5. Check the status of the workflow by using the following command `ovlm-fe workflow status -t <tenant_id> -v <vnf_id> -I <vnf_instance_id> -w <workflow_instance_id>`

```
[root@ovlm-demo ~]# ovlm-fe workflow status -t default -v microblog -i sample
-w 101100243
```

Note: Workflow instance id 101100243 is the one returned from the previous step

Running the command results in the following output:

```
data:
  method: GET
  request: http://node-1:28050/api/v1/tenants/default/vnfs/microblog/vnf_instances/sample/workflow_instances/101100243
  workflow_instance_status: RUNNING
status: 200
```

6. Once the instantiate flow has completed you can run the `is_vnf_up` command to check VNF status execute the workflow with the following example.

```
[root@ovlm-demo ~]# ovlm-fe workflow submit -t default -v microblog -i sample
-y is_vnf_up -p microblog_v1
```

Command result:

```
workflow_instance_id: '101100528'
[root@node-1 ~]#
```

7. Check the status of the `is_vnf_up` flow with the workflow status command

```
[root@ovlm-demo ~]# ovlm-fe workflow status -t default -v microblog -i sample
-w 101100528
```

Successful result shows a status code 200

```
data:
  method: GET
  request: http://node-1:28050/api/v1/tenants/default/vnfs/microblog/vnf_instances
```

```
/sample/workflow_instances/101100528
  workflow_instance_status: COMPLETED
status: 200
```

Failure result shows a status code of 500

```
Error Status: 500.
Error Code: EXECUTE_WORKFLOW_FAILED.
Detail: {Result=Lane( 3): Server 'node-4' ERROR: 'Is Vnfc Up' failed.
}
```

Next [Accessing the VNF](#) on page 46

Deploy Example VNF with VIM

Deploying the configured VNF from the CLI

To deploy the VNF, modify the VNF descriptor (VNFD) in the workspace, create a package from the workspace, and instantiate the package. The steps below show examples of packages using the out of the box package to deploy the VNF.

- Set the Front end baseurl environment variable to be OVLMFE_BASEURL=http://node-1:28050

```
[root@ovlm-demo ~]# export OVLMFE_BASEURL=http://node-1:28050
```

- Set the Configuration Manager baseurl environment variable to be OVLMCM_BASEURL=http://node-1:28010

```
[root@ovlm-demo ~]# export OVLMCM_BASEURL=http://node-1:28010
```

- Download the VNFD from the workspace and save it as vnfd.yml using the command ovlm-cm vnfd download -t <tenant_id> -v <vnfd_id> -i <vnfd_id> -f <path/filename>

```
[root@ovlm-demo ~]# ovlm-cm vnfd download -t default -v microblog -i vnfd1
-f vnfd1.yml
```

- Open vnfd.yml in a text editor and the VL1:properties for the available network as shown in the example below.

```
VL1:
  type: openet.nodes.nfv.VL
  properties:
    network_id: 79c2bffa-c8c7-4425-a974-190eb5a5f9ca
    subnet_id: 2d3b4597-7a48-49a3-bece-047586ec7daf
    vendor: openet
```

- Upload the modified vnfd.yml file using the command ovlm-cm vnfd upload -t <tenant_id> -v <vnfd_id> -i <vnfd_id> -f <path/filename>

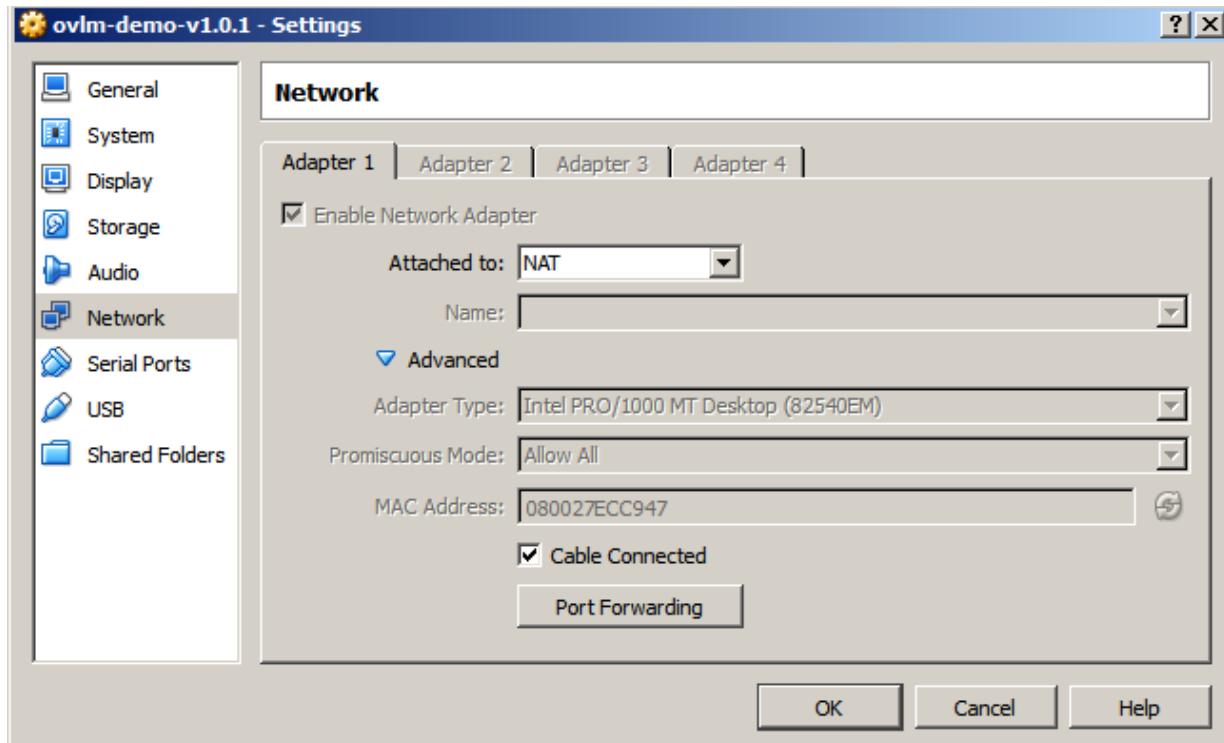
```
[root@ovlm-demo ~]# ovlm-cm vnfd upload -t default -v microblog -i vnfd1 -f
vnfd1.yml
```

- Create a package of the out of the box workspace using the following command ovlm-cm template create -t <tenant_id> -v <vnfd_id> -p <vnfd_template_id>

```
[root@ovlm-demo ~]# ovlm-cm template create -t default -v microblog -p
microblog_v1
```

Note: If you performed this step previously, delete that package before performing this step again.

7. Using the ipconfig command, retrieve your IP address (Windows Desktop). This is different from the VM image provided IP address.
8. Use the retrieved IP address for port forwarding rules configuration in your imported VM. Click **Machine > Settings > Network > Advanced > Port Forwarding**.



Note: In the following examples, the local host IP address 10.3.10.100 is used, and the Guest IP is 10.0.2.15.

Add the port forwarding rules by using add icon on the right hand side of the **Port Forwarding Rules** configuration window.

Name	Protocol	Host IP	Host Port	Guest IP	Guest Port
Nginx	TCP	127.0.0.1	8080	10.0.2.15	8080
OVLM-CM	TCP	10.3.10.100	28010	10.0.2.15	28010
OVLM-WF	TCP	127.0.0.1	8089	10.0.2.15	8089
Rsync	TCP	10.3.10.100	28000	10.0.2.15	28000
SSH	TCP	127.0.0.1	2222	10.0.2.15	22

9. Add the following port forwarding rules as described in the table below and click **OK**.

Name	Protocol	Host IP	Host Port	Guest IP	Guest Port
OVLM-CM	TCP	10.3.10.100	28010	10.0.2.15	28010
Rsync	TCP	10.3.10.100	28000	10.0.2.15	28000

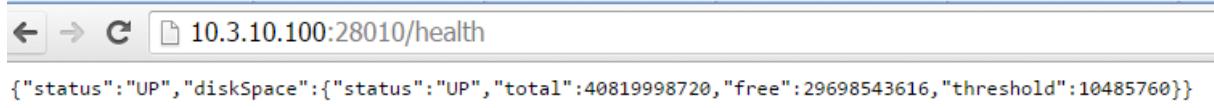
Any requests to your desktop machine are now forwarded to your VM image.

Note: Ensure you add the correct host IP address.

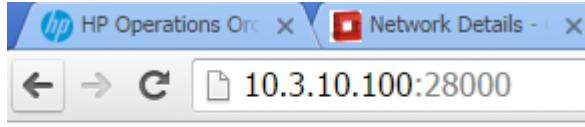
10. Ensure the request to the VM is forwarded to the respective node (node-1):

```
[root@ovlm-demo ovlm-install]# iptables -t nat -A PREROUTING -i enp0s3 -p tcp
--dport 28000 -j DNAT --to 10.3.10.100:28000
[root@ovlm-demo ovlm-install]# iptables -t nat -A PREROUTING -i enp0s3 -p tcp
--dport 28010 -j DNAT --to 10.3.10.100:28010
```

11. Using the configured OVLM-CM host IP address, perform the following local host test to ensure port forwarding is configured correctly:



12. Using the configured Rsync host IP address, verify the configuration is correct:



13. In the event of an error, ssh to node-1, and change the Rsync host value in the ovlm-val.yml file located in /etc/ovlm/val.

```
[root@ovlm-demo ~]# ssh node-1
Last login: Mon Aug  8 07:12:29 2016 from 10.0.3.1
[root@node-1 ~]#
```

```
[root@node-1 ~]# vi /etc/ovlm/val/ovlm-val.yml
```

Amend the original value (node-1):

```
configuration_manager:
  rsync_host: node-1
  rsync_port: 28000
  url: http://node-1:28010
```

Update to the correct rsync_host value:

```
configuration_manager:
  rsync_host: 10.3.10.100
  rsync_port: 28000
  url: http://10.3.10.100:28010
```

- 14.** Restart the VIM Abstraction Layer (VAL) microservice so that the changes to ovlm-val.yml can take effect using the command `systemctl restart serviceName`

```
[root@ovlm-demo ~]# systemctl restart ovlm-val
```

- 15.** Instantiate the VNF by calling the `instantiate_vnf` flow with the command `ovlm-fe workflow submit -t <tenant_id> -v <vnf_id> -l <vnf_instance_id> -y <workflow_type> -p <vnf_template_id> vnf_id <vnfd_id> -sync 1`

```
[root@ovlm-demo ~]# ovlm-fe workflow submit -t default -v microblog -i sample
-y instantiate_vnf \
-p microblog_v1 vnf_id vnf1 -sync 1
```

Note: The optional sync parameter specifies whether the API call is synchronous. The call is synchronous If set to 1 as in the above command, and the API call waits to execute until the end of the current workflow run or until the timeout value specified by the synchronous-request-delay option in the front-end configuration field. If sync is set to 0 the call is asynchronous, which is the default setting..

Command result:

```
workflow_instance_id: '101101930'
[root@node-1 ~]#
```

Note: The `workflow submit` command returns a `workflow_instance_id` which can be used to check the status of the flow.

- 16.** Check the status of the workflow by using the command `ovlm-fe workflow status -t <tenant_id> -v <vnf_id> -I <vnf_instance_id> -w <workflow_instance_id>`

```
[root@ovlm-demo ~]# ovlm-fe workflow status -t default -v microblog -i sample
-w 101101930
```

Note: Workflow instance id 101101930 is the one returned from the previous step

Running the command results in the following output:

```
data:
  method: GET
  request: http://node-1:28050/api/v1/tenants/default/vnfs/microblog/vnf_instances/sample/workflow_instances/101100243
  workflow_instance_status: RUNNING
status: 200
```

- 17.** Check periodically until complete:

```
[root@ovlm-demo ovlm-install]# ovlm-fe workflow status -t default -v microblog
-i dc_sample -w 101101930
data:
  method: GET
  request: http://node-1:28050/api/v1/tenants/default/vnfs/microblog/vnf_instances/sample/workflow_instances/101101930
  workflow_instance_status: COMPLETED
status: 200
```

18. To view the deployed blogserver you must ssh to node-1 to retrieve the IP address of the nginx from the CM node:

```
[root@ovlm-demo ~]# ssh node-1
Last login: Mon Aug  8 09:23:59 2016 from 10.0.3.1
```

19. Retrieve the value of the nginx IP address:

```
root@node-1 vnf_instances]# cat /usr/lib64/ovlm/cm/repository-root/tenants/default/vnfs/microblog/vnf_instances/dc_sample/vnf_topology.yml
vnfc_ids:
  nginx:
    - hostname: 10.0.3.81
      server_name: node-2
      vnfc_instance_id: 1
      network:
        virtual_link1:
          private_ip: 10.0.3.81
  blogserver:
    - hostname: 10.0.3.82
      server_name: node-3
      vnfc_instance_id: 2
      network:
        virtual_link1:
          private_ip: 10.0.3.82
    - hostname: 10.0.3.83
      server_name: node-4
      vnfc_instance_id: 3
      network:
        virtual_link1:
          private_ip: 10.0.3.83
  postgres:
    - hostname: 10.0.3.84
      server_name: node-5
      vnfc_instance_id: 1
      network:
        virtual_link1:
          private_ip: 10.0.3.84
```

nginx ip = 10.0.124.69

20. In your browser, enter the following address to verify that the blogserver is up:

<http://10.0.124.69:8080>

Next [Accessing the VNF](#) on page 46

Accessing the VNF

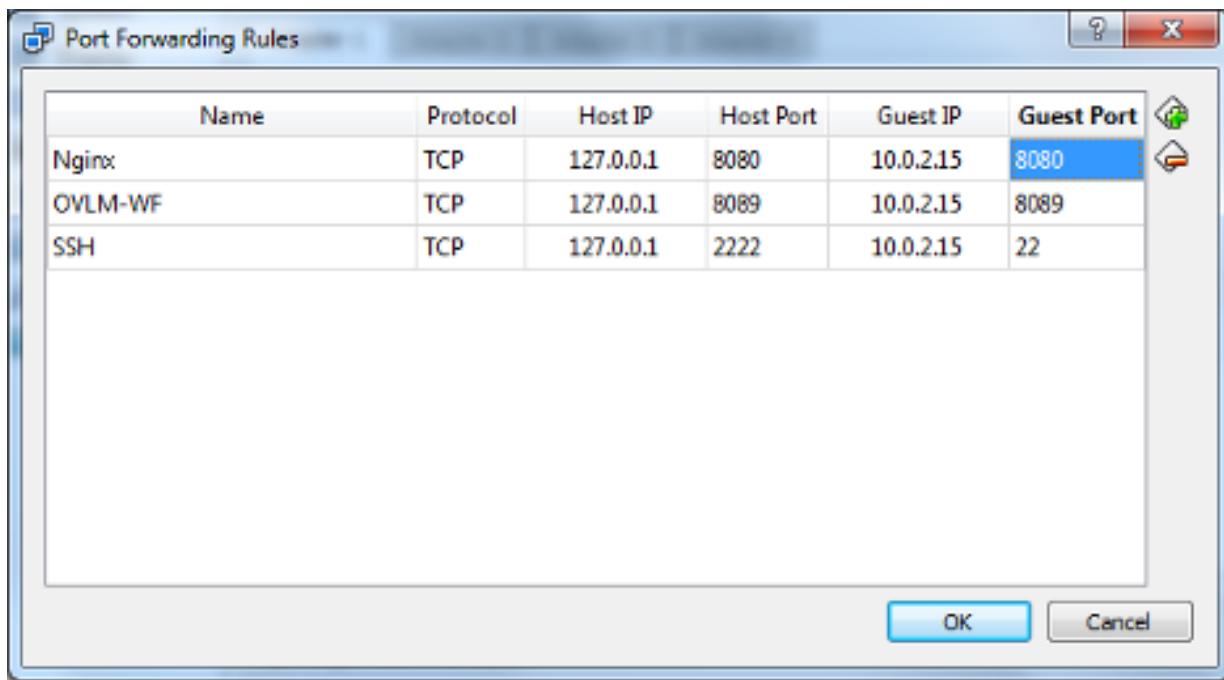
Accessing the micro blog created by the out of the box VNF package.

The micro blog application running on the VNFCs can be accessed from the host operating system through the web browser.

- To register an account on the blog server, open the web browser and enter <http://localhost:8080>.

The screenshot shows the MiniTwit application interface. At the top, there is a navigation bar with links for "my timeline", "public timeline", and "sign out [john]". Below the navigation bar, the title "My Timeline" is displayed. A text input field with the placeholder "What's on your mind john?" is present, along with a "Share" button. A single tweet from user "john" is shown, stating "Openet Weaver launches 31st March – 2016-03-31 @ 12:40". At the bottom of the page, a footer bar displays the text "MiniTwit – A Flask Application".

2. The virtual machine enables port forwarding so you can access the load balancer for the micro blog using a local port. The default is 8080. If you want to change this, select to the virtual box, and click **Setting > Network > Port Forwarding**.



3. To change the default port, double-click **Host Port 8082** and type a new port you can now access with the updated port **<http://localhost:<newport>>**.

Installing Openet Weaver

There are several distribution packages for Openet Weaver. This installation process refers to installing the tar file package in a test or production environment. For information on using the VM container set-up, see the Openet Weaver Quick Start Guide.

You must be the root user or have root user privileges to install Openet Weaver.

The installer can be run on one of the target nodes or from a separate node. When run from a separate node, the installation process connects to the target nodes, installing and configuring the software services on each node. The example deployment described in this section uses the Openet Weaver management node (node-1) as the installation node.

A number of target nodes can be used to host the different micro-services (Configuration Manager, Resource Manager, etc.) as well as runtime nodes. This example uses three nodes. The management node, node-1, acts as the installer node and contains the Configuration Manager, the Resource Manager and the Front-End components. The runtime nodes (node-2 and node-3) are used for VNF's.

Important: Use the following topics sequentially to install Openet Weaver in a test or production environment.

Hardware and Software Requirements

Openet Weaver has been certified and tested on CentOS 7.3. and RHEL 7.3 For the example deployment three VM's are required, prepared with the hardware resources specified in the following table.

Table 9: Openet Weaver Hardware Requirements

Item	node-1	node-2	node-3
CPU	2	2	2
RAM	8GB	2GB	2GB
Disk-Space	15GB	10GB	10GB

The following table lists the additional hardware requirements for the host node If you are installing Keycloak.

Table 10: Keycloak Hardware Requirements

Item	Requirement
RAM	1GB
DISK-Space	2GB

Table 11: Prerequisites for Installation

Item	Description
Unix account	To install RPMs and perform configuration you need a Unix account with sudo access.
Internet Access	To download third party dependencies from standard CentOS repositories.
Passwordless Key Based Access	Use passwordless key based authentication and have passwordless sudo access configured for all nodes. Although this is not mandatory, it reduces the overall deployment time because password entries are not required.

To install and run Openet Weaver you must download and install the third-party software in the following table.

Other required third-party software is supplied with Openet Weaver. See [Third Party Components delivered with Openet Weaver](#) for more information.

Table 12: Openet Weaver Software Requirements

Item	Description
Operating System	<p>CentOS 7.3, RHEL 7.3</p> <p>Note: A minimum install is sufficient provided base repositories are configured so that Openet Weaver can download the required dependencies.</p>
Resource Agent Operating system	<p>For Resource Agents, Openet Weaver provides support for the following Operating systems:</p> <ul style="list-style-type: none"> • CentOS 7.3 • RHEL 7.3 • Ubuntu 14.04 • Ubuntu 15 or Higher
Rsync	<p>Rsync version 3.0.9 or higher.</p> <p>This is file synchronization and transfer software.</p>
SSH	<p>This can be any SSH software that supports SSH2 specifications, for example openssh_6.6.x.</p> <p>Software that enables Openet Weaver to connect to local or remote systems to perform actions including</p> <ul style="list-style-type: none"> • Login • Configuration checking • File operations <p>Any SSH software that supports SSH2 specifications can be used, for example, openssh_6.6.x.</p>
OpenStack	<p>The Virtualized Infrastructure Manager (VIM) supported by Openet Weaver.</p> <p>Openet Weaver v1.11 is tested against OpenStack Liberty.</p>
keepalived	<p>keepalived release 1.2.13-8.el7.x86_64</p> <p>This is an implementation of VRRP. keepalived is required only if you are deploying Openet Weaver with high availability (HA)</p>
Java	<p>JDK release 1.8.</p> <p>Required when using Keycloak to implement Openet Weaver security. The version of Keycloak (3.2.1)supplied with Openet Weaver was tested against JDK 1.8.</p>
Cloud-init	<p>Cloud-init release 17.1</p> <p>cloud- init allows you to launch an instance which will integrate with a configuration management solution of your choice.</p>
Python 2.7.x	<p>Python release 2.7.x</p> <p>Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language</p>

When the hardware and software requirements are in place, the next step is to [Extract the installation tar file](#).

Third-Party Components Delivered with Openet Weaver

The table below lists required third-party software components that are delivered with Openet Weaver

Table 13: Required Third-Party software Delivered with Openet Weaver

Item	Description
HP Operations Orchestration (HPOO)	HPOO release 10.70. This is the workflow engine microservice that manages interaction between multiple microservices.
Influxdb	influxdb release 1.0.0. This is a time-series database that is used to store time-event data.
grafana	grafana release 3.1.0. This is a Web User Interface for visualizing time-series data.
python-psutil	python-psutil release 2.2.1. This is a cross-platform library for retrieving information in Python about running processes and system utilization, for example: <ul style="list-style-type: none"> • CPU utilization • System memory • Disk usage • Sensors
pip	pip release 9.0.1. The PyPA-recommended Python package installer, required for runtime node for Resource Agent and reference installation node.
PyYAML	PyYAML release 3.12. This is a YAML parser and emitter for Python.
Jinja2	Jinja2 release 2.10. Jinja is a template engine for the Python programming language. Jinja2 is a library for Python that is designed to be flexible, fast and secure.
Markupsafe	Markupsafe release 1.0. This software implements a unicode subclass that supports HTML strings.
Babel	Babel release 0.9.6. A collection of tools for internationalizing applications written in Python.
facter	facter release 2.4.6 Profiling library for the system to elicit node facts.
virtualenv	virtualenv release 15.1.0. This is a tool to create isolated Python environments.

Item	Description
keycloak	keycloak release 3.2.1. This is open-source identity and access management software that is used to implement security for openet Weaver.
net-snmp-libs	net-snmp-libs release 5.7.2. The net-snmp-libs package contains the runtime client libraries for shared binaries and applications.
net-snmp-utils	net-snmp-utils release 5.7.2. The net-snmp package contains various utilities for use with the NET-SNMP network management project. Install this package if you need utilities for managing your network using the SNMP protocol

Extracting the TAR File

You receive Openet Weaver as a tar file. To extract the file follow these steps:

1. Transfer the file to node-1 that is specified in the hardware and software requirements.
Node-1 is used as the installer node for the example lab deployment.

Note: You can use a separate machine to install the software provided it has the same software and hardware requirements as the node-1 node.

2. Extract the tar files on to the disk by entering the following command:

```
tar -xvf ovlm-core-install-enterprise-<release_version>.tar
tar -xvf ovlm-thirdparty-dependencies-<version_number>.tar
```

Note: Ensure that both files are in the same directory before extracting since they both extract to the same directory.

Extracting the files creates a directory named ovlm-install.

3. Change directory to where the tar was extracted by entering the following command: cd ovlm-install
4. View the directory contents by entering the following command: ls

The directory contains a number of files and sub directories, the libraries and rpms for the different services, a sample Openet Weaver install properties file, a manifest file, a sample VNF configuration and the ovlm-deploy script used to install the product. The following is an example of what could be in the directory:

```
ovlm-install
|--- config
|--- debs
|--- hpoo
|--- installer
|--- jce
|--- jre
|--- licence
|--- manifest.yml
|--- ovlm-deploy.sh
|--- ovlm-dm-util.sh
|--- ovlm-install-https-sample.yml
|--- ovlm-install-sample.yml
|--- python-pkg
|--- rpms
|--- samples
|--- ssh-keys
```

```

|-- utilities
`-- vnf_topology.yml

```

 **CAUTION:** The manifest.yml file captures installation information such as dependencies for Openet Weaver. Do not modify this file because the Openet Weaver installer depends on it to install successfully.

After extracting the tar file, you are ready to [Perform pre-installation tasks](#).

Performing Pre-Installation Tasks

This section describes Openet Weaver pre-installation tasks to be performed before running the installer. The users and privileges section describes the privileges you need to install Openet Weaver, and how to create ovlm users to operate the system. All tasks are mandatory except under the following conditions.

- Configuring Rsync to Run in SSH mode is applicable if you are using the Openet Weaver supported Rsync implementation. If you decide to use a custom implementation you are responsible for the custom Rsync configuration.
- Configuring Openet Weaver for HTTPS is mandatory only if you are planning to use the HTTPS protocol.
- Installing and configuring Keycloak is necessary only if you are enabling Keycloak security
- Configuring VIM is necessary only if you are planning to create VNFs with a Virtualized Infrastructure Manager (VIM).
- Configuring master and hot-standby nodes is mandatory only if you creating an HA deployment.

Users and Privileges

In order to install Openet Weaver you need to be a super user, which in this context is a root user or a user with root user privileges. This is because there are a number of commands that, by default, require sudo privileges to run. The commands in the following table require sudo privileges.

Table 14: Sudo Privilege Commands

Sudo Privilege Command	Description
/sbin/useradd	Creates a new user or updates the default new user information.
/sbin/userdel	Modifies the system account files, deleting all entries that refer to the user name.
/sbin/usermod	Modifies or changes attributes for an existing user account.
/bin/mkdir	Creates directories on a file system.
/bin/rm	Deletes files or directories.
/bin/rsync	Rsync is a fast and extraordinarily versatile file copying tool. It copy locally, to/from another host over any remote shell, or to/from a remote rsync daemon.
/bin/chown	Changes the user and/or group ownership of a specified file.
/bin/chmod	Changes permissions for files or directories.
/bin/rpm	Builds, installs, queries, verifies, updates, and erases individual software packages.
/usr/bin/yum	Performs package installation, removes old packages, and queries the installed and/or available packages among many other commands and services.

Sudo Privilege Command	Description
/usr/bin/apt-get	Works with Ubuntu's Advanced Packaging Tool (APT) library to perform software installation, remove existing software packages and upgrade existing software packages
/usr/bin/dpkg	Installs, builds, removes and manages Debian packages.
/usr/sbin/service	Runs a System V init script or upstart job in as predictable an environment as possible.
/usr/bin/systemctl status	Shows the status of a systemd service.
/usr/bin/systemctl start	Activates a systemd service.
/usr/bin/systemctl stop	Deactivates a systemd service.
/usr/bin/systemctl restart	Restarts a systemd service.
/usr/bin/systemctl is-active	Checks whether a systemd service is active.
/usr/bin/systemctl daemon-reload	Reloads the systemd manager configuration. This reruns all generators, reload all unit files, and recreate the entire dependency tree.
/bin/bash	Runs the GNU Bourne-Again SHell.
/bin/sh	
/bin/ls	Lists files and related information.
/bin/echo	Echos text STRING(s) to standard output.
/usr/bin/echo	Echos text STRING(s) to standard output.
/usr/sbin/groupadd	The groupadd command creates a new group account using the values specified on the command line plus the default values from the system.
/usr/bin/mv	Moves (rename) files.
/usr/bin/rm	Deletes files or directories.
/usr/bin/cp	Copies files.
/usr/bin/tar	GNU tar saves multiple files in a single tape or disk archive, and can restore individual files from the archive.
/usr/bin/sed	Runs the Stream editor for filtering and transforming text.
/usr/bin/python	Python is an interpreted, interactive, object-oriented programming language.
/bin/update-ca-trust	You use update-ca-trust(8) to manage a consolidated and dynamic configuration feature of Certificate Authority (CA) certificates and associated trust.
/usr/bin/touch	Updates the access and modification times of each FILE to the current time.
/usr/bin/ssh	Runs the OpenSSH SSH client (remote login program).

The super user is responsible for creating the ovlm user for running the system. The ovlm user also needs the sudo privileges listed in the above table.

You can enable the commands in the table above for the installation user by appending the following lines to the sudoers file on all servers for which OVLM microservices are being installed, replacing <install user> with the username of the installation user as specified at the -u parameter when you run ovlm-deploy.

```
Cmnd_Alias OVLM_DEPLOYMENT_CMDS = /bin/chmod, /bin/chown, /bin/mkdir, /bin/bash,
/bin/ls, /bin/echo, /usr/bin/echo, /usr/sbin/groupadd, /usr/sbin/useradd,
/usr/bin/mv, /usr/bin/rm, /usr/bin/cp, /usr/bin/tar, /usr/bin/sed,
/usr/bin/systemctl, /usr/bin/yum, /bin/rpm, /usr/bin/python,
/bin/update-ca-trust, /usr/bin/touch, /bin/sh, /usr/bin/ssh, /sbin/userdel,
/sbin/usermod, /bin/rm,. /bin/rsync, /usr/bin/apt-get, /usr/bin/dpkg,
/usr/sbin/service
<install user> ALL=(ALL) NOPASSWD: SETENV: OVLM_DEPLOYMENT_CMDS
```

As part of your deployment planning, you need to determine whether you are going to install Openet Weaver with master and hot-standby nodes for high availability (HA). If you are not creating an HA deployment, you can simply let the installer create the ovlm user on the local node automatically during deployment, which it will if one is not created before installation.

If you are deploying Openet Weaver with HA, you need to create an ovlm user on the NIS domain server before installation. This is necessary so that the ovlm user can access the shared repository-root from the master or hot-standby nodes, which the ovlm user created locally by the deployment script is not capable of, which can lead to NIS-related issues such as not being able to install packages, not being able to start services, or long delays in response times to clients.

For security reasons, the ovlm user created on the NIS domain server must be created with no login shell and no home directory to prevent the user from accessing the management node.

To create an ovlm user on the NIS domain server, use the following command.

```
useradd -g ovlm ovlm --shell=/sbin/nologin --no-create-home
```

You can add to the sudo privileges listed in the table above by defining new sudo privileges in the installation file. An example is shown below.

```
resource_agent:
hosts:
- server01
- server02
- server03
- server04
properties:
logging:
level:
com.openet.modules.ovlm: DEBUG
server:
port: 62086
protocol: http
sudo_privileges_required: true
sudo_privileges:
- /bin/bash
- /bin/passwd
```

When you run the Openet Weaver installer it updates the sudo privileges in VNFCs.

You can also update sudo privileges after installation by updating the installation file and running the command `./ovlm-deploy.sh -i install_properties_file -u installation_user`.

See [Granting and Revoking OVLM User Sudo Privileges](#) for more information about changing the ovlm user's sudo privileges after installation.

After performing this step you are ready to configure Rsync.

If you are using the Openet Weaver support Rsync implementation refer to [Configuring Rsync to run in SSH mode](#).

If you are using a custom implementation of Rsync then refer to [Configuring Custom Rsync](#).

Configuring Rsync to Run in SSH Mode

Openet Weaver transfers VNF component files from their main repository to VNF runtime nodes. The file transfer mechanism is built on Rsync. The Rsync service is started on the management node. The agent Rsync instances connect to it in client mode. Agents must have read-only access to the main repository.

Important: This topic and its sub pages apply only when the Openet Weaver rsync server configuration is used, as specified by a value of true in the `rsync_daemon_mode` parameter in the `configuration_manager` block of the installation file. If `rsync_daemon_mode` is set to false a custom rsync configuration is used and the user is responsible for configuring rsync to run in SSH mode.

For more information see [Creating an Installation File](#).

The Openet Weaver installer can be used to enable secure ssh at installation time. This topic describes how to generate keys, check that tunneling works, and how to configure management nodes and agent nodes manually post-installation so you can troubleshoot tunneling or change the security mode. Apply all manual configuration changes to the management node and every runtime node.

SSH tunneling is illustrated below.

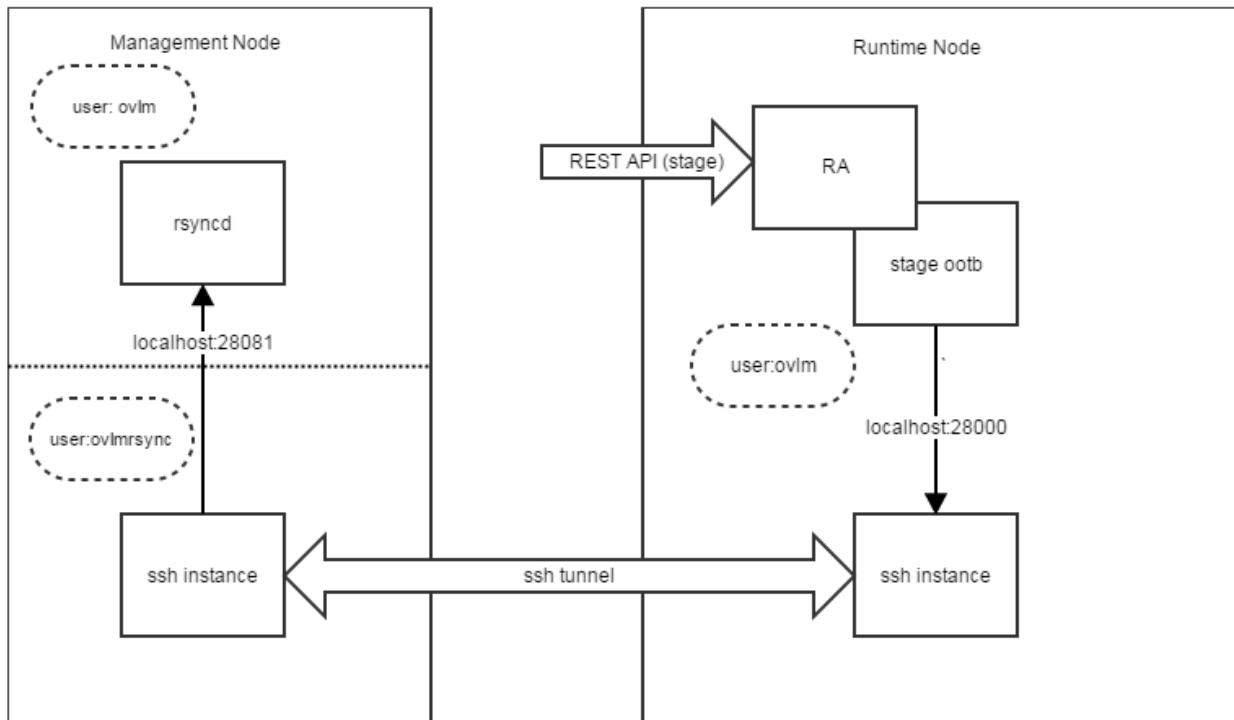


Figure 8: SSH Tunneling Between a Management Node and a Runtime Node

Generating Keys

Rsync traffic encryption is disabled by default. The Openet Weaver product installer provides a set of properties to setup security automatically during installation, which is covered in the topic on running the installer. Although default keys are available. You can generate your own using the `ssh-keygen` command as in the following example.

```
# ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa): /tmp/ovlmrsync_rsa
Enter passphrase (empty for no passphrase):
```

```

Enter same passphrase again:
Your identification has been saved in ovlmrsync_rsa.
Your public key has been saved in ovlmrsync_rsa.pub.
The key fingerprint is:
1d:06:9b:64:7d:56:79:3c:fe:4e:7d:45:45:83:0d:f9 root@node-2
The key's randomart image is:
+-- [ RSA 2048] ----+
|       +. o*=+|
|       o +. oo.o=|
|       o oo  oo.| |
|       o .   E.| |
|       S .     +| |
|           =| |
|           o.| |
|           .| |
|           | |
+-----+| |

# ls /tmp
ovlmrsync_rsa  ovlmrsync_rsa.pub

```

Configuring the Management Node Manually

To configure the management node so that it can be switched to secured mode, perform the following steps:

Note: All commands in this procedure must be run with sudo permissions.

1. Create a secured ovlmrsync user for ssh access to the management node using the command #useradd -s /bin/bash ovlmrsync.

Note: You must create a secured user with locked password access because there is no login for a standard Openet Weaver user.

By default, running the command creates a user with locked password access. In the event that the user already exists you can disable the user password using the following command.

```
# passwd -l ovlmrsync
```

The result should be similar to the following.

```

Locking password for user ovlmrsync.
passwd: Success

```

2. Create a home directory for the user.
3. Generate a pair of ssh keys using the command # ssh-keygen

The result of running the command should be similar to the following.

```
Generating public/private rsa key pair.
```

4. When prompted, enter a name for the file to save the key as in the example blow.

```
Enter file in which to save the key (/root/.ssh/id_rsa): ovlmrsync_rsa
```

5. Press **Enter** twice when prompted for a passphrase as in the example below. This field should be left empty.

```

Enter passphrase (empty for no passphrase):
Enter same passphrase again:

```

You should see output similar to the following.

```
Your identification has been saved in ovlmrsync_rsa.
Your public key has been saved in ovlmrsync_rsa.pub.
The key fingerprint is:
1d:06:9b:64:7d:56:79:3c:fe:4e:7d:45:45:83:0d:f9 root@node-2
The key's randomart image is:
+-- [ RSA 2048] ----+
|      +. o*=+=|
|      o +. oo.o=|
|      o oo  oo.| |
|      o .    E.| |
|      S .     +| |
|          =| |
|          o.| |
|          .| |
|          | |
+-----+
```

```
# ls
ovlmrsync_rsa  ovlmrsync_rsa.pub
```

6. Authorize the generated public key for the ovlmrsync user by adding it to the \${ovlmrsync_home}/.ssh/authorized_keys file. Prefix the key with extra rules to increase security as in the following example.

```
no-pty,permitopen="localhost:28000",command="/bin/echo do-not-send-commands"sh-rsa
AAAABdgz1yc2EAAAQABAAQDUDDNLba8wRsY/T5xkXZ6sj8FN91qd
q2v8Im5IoZ5XtW9vZx8VT4QDD1gSld6mWigxOXzWz0z2za5h7nEE1vshSxOtRsraYo
qMLI+2C6wDvCb949zOytaDZvXl8ZGofQdOdCYJq3aDjEq6mQuTzd54Zgw2U1CuhzDa
3hNw8xx3S/v4iSQ CfKETD6h7vDUuj+MC4mZ/nckZPRyQS Ylm5GuptTz8Dd6MsConoP
9SQKHWoHPmp8euyOPZmbv1i4wSpTCnSnwxbTPWISy3BQEjoREB+Lb2Mzn/Wjfs2XNi
3UVdMotAVVY9nJexCtWJfbKp9vlhb4SMQhCPW+PIGeXN010D
root@ovlm-demo
```

In the above example the following restrictions are added:

- Login is not allowed
- Commands cannot be run remotely
- Access to only local 28000 rsync server port is granted

It is assumed that the same key pair is used for all runtime nodes. So the generated public key (ovlmrsync_rsa.pub) is installed on management node and the private key (ovlmrsync_rsa) is installed to each runtime node.

7. Configure the Rsync service by adding the following to the /etc/rsyncd-ovlm.conf file.

```
address = 127.0.0.1
hosts allow = 127.0.0.1
```

Adding the above forces Rsync to accept connections only from the loopback interface. After adding the new configuration the rsyncd-ovlm.conf file should look similar to the following.

```
uid = root
gid = root
# you are setting the Rsync port to 7778 because SSH tunnelling via port 7777
# is used.
port = 28000
log file = /var/log/ovlm/rsync/rsyncd.log
pid file = /var/run/ovlm/rsync/rsyncd.pid
# The main module. This could be broken into multiple modules per tenant /
vnf
```

```
[ovlm]
max connections = 32
# As rsync is not run as root (and ovlm-user doesn't have sudo access) chroot
# is not allowed.
use chroot = no
path = /usr/lib64/ovlm/cm/repository-root
comment = The location of tenants
auth users = ovlm
secrets file = /etc/ovlm/rsyncd.secrets
read only = true
# To enforce the max connections limit
lock file = /usr/lib64/ovlm/cm/../rsync/rsyncd.lock
list = yes
```

8. To apply the changes to the rsyncd-ovlm.conf file, restart Configuration Manager using the command #systemctl restart ovlm-cm.
9. Verify that the rsync port attribute in the /etc/rsyncd-ovlm.conf file and the permitopen attribute in \${ovlmrsync_home}/.ssh/authorized_keys use the same port number.
10. Verify that Configuration Manager restarted.

Configuring the Runtime Node Manually

To configure runtime nodes so that they can be switched to secured mode, perform the following steps:

Note: All commands in this procedure must be run with sudo permissions.

1. Add a configuration_manager section to the /etc/ovlm/ra/ovlm-ra.yml file as in the following example.

```
configuration_manager:
  secured_stage: true
```

2. Place the private key that was generated during the management node configuration process in the /etc/ovlm/ra/ovlmrsync_id_rsa file of each runtime node.

The following example assumes that the Resource Agent is running under the ovlm user.

```
# su - ovlm
$ cp <path_to_key>/ovlmrsync_rsa /etc/ovlm/ra/ovlmrsync_id_rsa
$ chmod 600 /etc/ovlm/ra/ovlmrsync_id_rsa
```

3. Add the management node to the list of known hosts using the command \$ ssh-keyscan management_node_host 2>&1 | sort -u - /home/ovlm/.ssh/known_hosts > /home/ovlm/.ssh/known_hosts.tmp && mv /home/ovlm/.ssh/known_hosts.tmp /home/ovlm/.ssh/known_hosts
4. To apply the changes made to the runtime node, restart each Resource Agent using the command # systemctl restart ovlm-ra
5. Verify that ssh is enabled by running the following commands on each Resource Agent to create a secure tunnel, run Rsync, and remove the tunnel.

```
# su - ovlm
// start tunnel, where 28080:localhost:28000 is <local-port>:localhost:<remote-port>
$ ssh -i /etc/ovlm/ra/ovlmrsync_id_rsa -N -f -M -S /tmp/test_socket -L 28080:localhost:28000 ovlmrsync@<management-node>

// run rsync with port set to <local-port> from the step above
$ rsync --dry-run -vhaHX --del --timeout=100 --port=28080
--password-file=/etc/ovlm/ra/cm.rsync.secret ovlm@localhost::ovlm/tenants/default/vnfs/microblog/templates/microblog_v1/metadata /tmp
```

```
// remove the tunnel
$ ssh -S /tmp/test_socket -O exit <management-node>
```

After configuring Rsync, the next step you take depends on what Openet Weaver Features you are deploying:

- If you are going to use the HTTPS protocol, refer next to [Configuring Openet Weaver for HTTPS](#).
- Otherwise, if you are enabling Keycloak security, refer next to [Installing and Configuring Keycloak](#).
- Otherwise, if you intend to create VNFs with VIM, refer next to [Configuring VIM \(OpenStack\) for Openet Weaver](#).
- Otherwise, if you are creating a highly available (HA) deployment, refer next to [Configuring Master and Hot-Standby Nodes for High Availability](#).
- Otherwise, if you are implementing none of the above, refer next to [Creating an Installation File](#).

Configuring Custom Rsync

By default, Openet Weaver installs and starts Rsync on the management node where Configuration Manager is installed and at each Resource Agent node. Rsync is used to:

- Stage templates from Configuration Manager to Resource Agent nodes
- Manage internal logic at Resource Agent nodes during operation.

As an alternative to using the Openet Weaver Rsync server configuration, you can use a custom implementation of Rsync that has built-in SSH for stage operations. When you use a custom implementation, you are responsible for the configuration. This section provides information about how to make that implementation compatible with Openet Weaver. This includes enabling custom Rsync in the installation file and making sure your custom Rsync implementation supports and matches all flags supported by the Openet Weaver supported implementation of Rsync.

Note: do not set the following when using custom rsync:

```
configuration_manager:
  secured_stage: true
```

Enabling Custom Rsync

Enabling custom Rsync in the installation file requires you to give the rsync_daemon_mode parameter a value of false in the configuration_manager block as in the following example:

```
configuration_manager:
  hosts:
    - node-1
  properties:
    server:
      port: 28010
  rsync_daemon_mode: false  #optional, the default value is true.
  rsync_port: 28000
```

Ensure that the resource_agent: metadata: stage: parameters: rsync: bin property contains the fully qualified path to Rsync as in the following example:

```
resource_agent:
  properties
    resource_agent:
      metadata:
        stage:
          parameters:
            rsync:
              timeout: 100
              bin: rsync
              share_name: ":ovlm"
```

```

secret_user: ovlm
ssh_user: ovlmrsync
ssh_port: 28000
secret_file: /etc/ovlm/ra/cm.rsync.secret

```

When configuring the resource _agent block for a custom implementation if Rsync, you must also be aware of the following:

- The following parameters have no effect when rsync_daemon_mode is set to false:
 - share_name
 - secret_files.
- The Configuration Manager repository-root is used for staging operations
- you must set up SSH key-exchange between the Openet Weaver management node that hosts Configuration Manager and all Resource Agent nodes. You must also start the SSH key-exchange at the Openet Weaver management node where Configuration Manager is hosted. In a typical setup the service_owner (default ovlm) saves the private key in its home directory of the secret_user authorized_keys, for example ~/.ssh/id_rsa
- Make sure the port numbers match between the Rsync server, the rsync_port parameter in the configuration_block, and the resource_agent block in the installation file
- Use the secret_user parameter to change the Rsync user

Note: See [Creating an Installation File](#) for more detailed information for all the parameters in the examples above.

Rsync Flag Reference

The following table describes the flags that are used in the Openet Weaver supported Rsync implementation (v3.0.9). Your custom Rsync implementation must support and match all the flags in the table:

flag	Description
-r	Recursive
-o	Preserve owner (super-user only)
-g	Preserve group
--delete	Rsync to delete extraneous files from the receiving side
-v	Verbose
--exclude	Exclude files/folder
-h	Output numbers in a human-readable format
-a	Archive mode
-c	Skip based on checksum
-i	Output a change-summary
-H	Hard links

After configuring Rsync, the next step you take depends on what Openet Weaver Features you are deploying:

- If you are going to use the HTTPS protocol, refer next to [Configuring Openet Weaver for HTTPS](#).
- Otherwise, if you are enabling Keycloak security, refer next to [Installing and Configuring Keycloak](#).
- Otherwise, if you intend to create VNFs with VIM, refer next to [Configuring VIM \(OpenStack\) for Openet Weaver](#).
- Otherwise, if you are creating a highly available (HA) deployment, refer next to [Configuring Master and Hot-Standby Nodes for High Availability](#).
- Otherwise, if you are implementing none of the above, refer next to [Creating an Installation File](#).

Configuring Openet Weaver for HTTPS

This topic describes how to configure Openet Weaver for HTTPS support.

The main topic takes you through an overview of the configuration process up through running the installer. For more detailed information about some of the steps refer to the topics linked to the bottom of this page.

To configure and verify HTTPS setup on Openet Weaver, follow these steps:

1. Create a certs folder in which to put SSL certificates using the command `mkdir -p ./ovlm-install/certs`
2. Copy the server certificate to the certs folder with the name `server.crt` using the command `cp server.cert ./ovlm-install/certs/server.crt`.
3. Copy the server private key to the certs folder with the name `server.key` using the command `cp private.key ./ovlm-install/certs/server.key`
4. Generate a bundle from a list of certificates using the command `sudo su -c "cat intermediate1.crt intermediate2.crt root.crt > ./ovlm-install/certs/ca_bundle"`
5. If you are using multiple SSL certificates, update the `ca_bundle` with the intermediate SSL certificates you are adding to the system using the command `sudo su -c "cat intermediate_service.crt root_service.crt > ./ovlm-install/certs/ca_bundle"`.

For example, if you are adding certificates for Resource Agents, you might use the following command:

```
cat intermediate_ra.crt root_ra.crt > ./ovlm-install/certs/ca_bundle
```

If you are not performing step 5, skip to step 9.

6. If you performed step 5, Verify that the certificates were added to the CA bundle using the command `keytool -printcert -v -file ca_bundle`
7. If you performed step 5, create a `keystore.jks` file for that microservice using the following commands:

```
openssl pkcs12 -export -name demo -in <service>.cert -inkey private_key.key -out service_keystore.p12

keytool -importkeystore -destkeystore <service>_keystore.jks -srckeystore service_keystore.p12 -srcstoretype pkcs12
```

In the above example, replace `service` with the string that represents the microservice at which the SSL certificates are targeted. `<service>.cert` must be the same filename used in step 5.

Note: The JKS for the main SSL certificate set is generated automatically during installation so you only need to generate JKS files for additional SSL certificates.

8. Copy the generated JKS file to the certs folder using the command `cp service_keystore.jks ./ovlm-install/certs/server.crt`.

Note: For more information about steps 1 through 8, see the subtopic [SSL Certificates and Generating a CA Bundle](#).

9. Provide values for the HTTPS installation properties in the installation file.

The installation file has a Protocol section with the following properties:

```
protocol: https
https_configuration:
  server:
    keystore_path: key_store.jks
    keystore_password: password
  client:
    verify_server_certificate: Yes
    truststore_path: trust_store.jks
```

If you are using multiple certificates you must create a block for each certificate set. In the following example there are SSL certificates specific to Resource Agents in addition to the main certificate set.

```
server:
  port: 28086
  protocol: https
  keystore_path: keystore.jks
  keystore_password: password
client:
  verify_server_certificate: Yes
  truststore_path: trust_store.jks
server:
  port: 28086
  protocol: https
  keystore_path: ra_keystore.jks
  keystore_password: password
client:
  verify_server_certificate: Yes
  truststore_path: trust_store.jks
```

10. Encrypt the passwords in the installation file using the command `utilities/cipher/encrypt_password.sh -f installation_file`.

Note: Configure the entire file before running the encryption script. For more information see the *Encrypting Passwords* topic.

11. Run the installer as described in the Running the Installer topic.

During deployment java key stores are created: key_store.jks and trust_store.jks. Key stores are copied into `etc/ovlm/<services>/`.

Configuration services YML files are updated during installation and intermediate certificates (ca_bundle) are installed on host machines.

Note: Management and Resource Agent deployment hosts must have same domain as the certificates common name, for example CN=*.domain.com

Note: Steps 9 through 11 are part of the installation process described in subsequent topics of the *Installing Openet Weaver* section of the webhelp and are performed as part of overall installation.

Once installation is complete you can enable SSL for CLI commands either by configuring environment variables or passing parameters in the CLI command. For more information see the subtopic Enabling SSL for the CLI.

Note: Before instantiating VNFs using SSL, update the VNF topology file. Replace any IP addresses specified for hosts with hostnames. The following is an example.

```
vnfc_ids:
  blogserver:
    - hostname: rt1-dub-all-lane.anycom.com
      server_name: rt1-dub-all-lane.anycom.com
      vnfc_instance_id: 1
    - hostname: rt1-dub-all-lane.anycom.com
      server_name: rt1-dub-all-lane.anycom.com
      vnfc_instance_id: 2
    - hostname: rt2-dub-all-lane.anycom.com
      server_name: rt2-dub-all-lane.anycom.com
      vnfc_instance_id: 3
  nginx:
    - hostname: rt3-dub-all-lane.anycom.com
      server_name: rt3-dub-all-lane.anycom.com
      vnfc_instance_id: 1
  postgres:
    - hostname: rt4-dub-all-lane.anycom.com
```

```
server_name: rt4-dub-all-lane.anycom.com
vnfc_instance_id: 1
```

SSL Certificates and Generating CA Bundles

SSL certificates are required for Openet Weaver to work with HTTPS. Each certificate includes a cryptographic key and information about the organization to which the certificate belongs. Certificates are required for the following:

- Identification to the server
- Encrypting the data that's being transmitted

Certificates and their keys (server.crt, ca_bundle, server.key) must be in PEM format.

Place certificates under in the `./ovlm-install/certs` directory. Each filename is related strictly to data inside. Certificates are described in the following table.

filename	Description	Comments
server.key	Contains the private key.	Rename the file from your own key file to server.key.
server.crt	contains the public certificate.	Rename the file from your own domain or server crt file to server.crt.
ca_bundle	A bundle of certificates (trust certificates) concatenated in a specific order. It includes the following: <ul style="list-style-type: none"> • All intermediate certificates • The root certificate 	This must be generated manually.



Attention: Create the certs folder manually before enabling HTTPS. Create the folder in the same path as the `ovlm-deploy.sh` script. Then put your own certificate files in the certs folder. You must also rename your own private key, cert and `ca_bundle` according to file names `server.key`, `server.crt` and `ca_bundle` as in the table above when you place them in the certs folder.

You can generate a `ca_bundle` from a list of certificates using the command `sudo su -c "cat intermediate1.crt intermediate2.crt root.crt .ovlm-install/certs/ca_bundle"`.

```
sudo su -c "cat ./certs/intermediate1.crt ./certs/intermediate2.crt ../ \
certs/root.crt > ../certs/ca_bundle"
```

Generating a CA bundle generates intermediate certs with the public root certificate. The application doesn't know anything about intermediate certificates so it simply passes these certs through a trust store like `ca_bundle`, or Java Trust Storage for Java applications, then points the application to this bundle. Place trust certificates inside the `ca_bundle`.

If have your own certificates, place them in the certs folder. Newer certificates overwrite old ones.

Setting Up the Installation File to Deploy HTTPS Automatically

Before installation, configure the installation file so that the HTTPS protocol and associated properties are specified. encrypt all passwords with the utility `encrypt-password.sh`. See the section on Creating an installation file for configuration specifics.

During installation:

- The Installer creates `keystore_path` and `truststore_path` files automatically in the certs folder based on the certificates and private key you provide
- All microservices have protocol set to https
- Items from the server and clients section of the protocol settings are appended to each microservice

- Keystore and truststore files are uploaded to each microservice installed folder and configured to use them
- The ca_bundle is uploaded to each microservice and registered in system automatically to support CLI server verification

After installation you can verify the HTTPS protocol is set up by running the health status flow on You should see a response with information about the URIs through which services interact with each other. The schema should be https for each service status returned.

Note: During installation the ca-bundle is copied to CentOS: /etc/pki/ca-trust/source/anchors Ubuntu: /usr/local/share/ca-certificates and updated. Therefore intermediate certificates (ca_bundle) are installed on host machines .

Enabling SSL for the CLI

You can enable SSL for CLI commands. There are two ways to do this:

- By passing an optional parameter through the command line parameter
- By configuring environment variables

The following table describes the configuration parameters used in both options.

Table 15: SSL CLI Command Parameters

CLI Command Parameter	Environment Variable	Description
--ca_bundle_path	OVLM<microservice abbreviation>_CA_BUNDLE_PATH	<p>ca_bundle_path provides path to your own certificate authority bundle. This is useful for cases where self-signed certificates are used. When you specify our own certificate bundle only the requests that can be verified with that bundle succeed.</p> <p>By default, the installer automatically uploads ca_bundle into the microservice host and registers it.</p> <p>Note: You are not required to provide ca_bundle option if valid (not self-signed) certificates are used.</p>
--verify_server_certificate	OVLM<microservice abbreviation>_VERIFY_SERVER_CERTIFICATE	<p>By default, server certificate verification is disabled. The following is an example of enabling it by passing a parameter in a CLI command:</p> <p>Any value other than "yes" renders server certificate verification disabled .</p> <p>Note: The value "yes" is not case sensitive.</p>

The following is an example of passing parameters in a Front-end CLI command:

```
ovlm-fe health status --baseurl=https://server:28085 --verify_server_certificate=yes --ca_bundle_path=/path/to/certs/ca_bundle
```

alternatively, you can set environment variables for each of the Openet Weaver microservices. The following is an example of setting an environment variable for server certificate verification for all Front-end CLI commands:

```
export OVLMFE_VERIFY_SERVER_CERTIFICATE=YES
export OVLMFE_CA_BUNDLE_PATH=/path/to/certs/ca_bundle
export OVLMFE_BASEURL=https://server:28085
ovlm-fe health status
```

Similarly, the following is an example of setting an environment variable for server certificate verification for all Configuration Manager CLI commands:

```
export OVLMCM_VERIFY_SERVER_CERTIFICATE=YES
export OVLMCM_CA_BUNDLE_PATH=/path/to/certs/ca_bundle
export OVLMCM_BASEURL=https://server:28010
ovlm-cm vnf list -t default
```

Possible CLI Warning Messages

The following table describes the possible non-critical warning messages you could receive when running a CLI command with SSL.

Table 16: CLI Warning Messages

Warning	Full Text	Description	Resolution
InsecureRequestWarning	usr/lib/python2.7/site-packages/urllib3/connectionpool.py:769: InsecureRequestWarning: Unverified HTTPS request is being made. Adding certificate verification is strongly advised. See: https://urllib3.readthedocs.org/en/latest/security.html InsecureRequestWarning)	This can indicate that you are using the HTTPS protocol but --verify_server_certificate is not set to yes in the CLI command.	Run the CLI command with --verify_server_certificate=yes or set the corresponding environment variable for the microservice.
SecurityWarning	/usr/lib/python2.7/site-packages/urllib3/connection.py:251: SecurityWarning: Certificate has no 'subjectAltName', falling back to check for a 'commonName' for now. This feature is being removed by major browsers and deprecated by RFC 2818. (See https://github.com/shazow/urllib3/issues/497 for details.) SecurityWarning	You might receive this warning when connecting to a host with a certificate that lacks SAN(Subject Alternative Names). It is possible the certificate is self-signed.	Use certificates with SAN.

Possible CLI Error Messages

The following table describes the possible SSL-related error messages that can stop a CLI command from completing its run.

Error Message	Possible Reason	How to Resolve
Certificate verify failed	Certificate validity can't be verified.	If the error message relates to a self-signed certificate, add the path to the certificate with the flag --ca_bundle_path. If the error relates to a CA signed certificate, install the certificate on the machine as described in SSL Certificates and Generating CA Bundles .
Hostname doesn't match the URL	The hostname in the certificate doesn't match the URL name.	There are two ways to resolve this error: <ul style="list-style-type: none">• Use a certificate with the same hostname that is included in the URL.• Use a baseurl with the same hostname as the certificate, for example ovlm-fe health status - -baseurl=https://url:port --verify_server_certificate=yes
Couldn't find or load certificate	The path to the certificate is incorrect.	Check whether the path and certificate passed through option - -ca_bundle_path are valid.
Server doesn't support 'https' protocol	The server is running without supporting HTTPS.	Restart the server with HTTPS support

For warning or error messages that do not appear in the above tables you can reference <http://urllib3.readthedocs.io/en/latest/reference/index.html?highlight=sslerror>

Including Additional Certificates after Deployment

This topic describes how to add SSL certificates to a deployment that already has a main set of certificates, so that Openet Weaver operates using multiple certificates. The main certificate set targets all microservices, and the additional certificates target specific microservices.

Note: HTTPS must be configured and deployed in Openet Weaver before following this procedure.

To add SSL certificates to Openet Weaver, follow these steps:

1. Update ca_bundle with intermediate certificates of service which use additional set of certs if it needed.
2. Create service keystore .jks file
3. Update topology.yml to use additional set of certs for specified service
4. Encrypt the topology file
5. Run the installer

After configuring Openet Weaver for HTTPS, the next step you take depends on what Openet Weaver Features you are deploying:

- If you are enabling Keycloak security, refer next to [Installing and Configuring Keycloak](#).
- Otherwise, if you intend to create VNFs with VIM, refer next to [Configuring VIM \(OpenStack\) for Openet Weaver](#).
- Otherwise, if you are creating a highly available (HA) deployment, refer next to [Configuring Master and Hot-Standby Nodes for High Availability](#).
- Otherwise, if you are implementing none of the above, refer next to [Creating an Installation File](#).

Installing and Configuring Keycloak

If you decide to deploy Openet Weaver with the security server you need to perform the following tasks before installation:

- Prepare certificates for Keycloak
- Install Keycloak
- Configure Keycloak
- Start and test Keycloak

This section provides you with the information you need to install Keycloak and deploy it with Openet Weaver. For further information see the Keycloak product documentation.

Preparing Certificates for Keycloak

You must create a Java keystore for Keycloak and copy in the certificates you created for the other microservices in [Configuring Openet Weaver for HTTPS](#).

For Keycloak, you must also create a keystore for these certificates. For geo-redundant deployments you must do this on the active node and the passive node. You use your key and certificate to create a PKCS12 file, which is compatible to be imported into a Java Keystore (JKS).

To create a Java keystore for keycloak, follows these steps:

1. Create a temporary directory for extracting files used to create the keystore by entering the following command:

```
mkdir temp-jdk
```

2. Change to the directory you created in step 1 using the following command:

```
cd temp-jdk
```

3. Extract JDK into the directory using the following commands:

```
tar xzf ../../ovlm-install/jre/server-jre-8u60-linux-x64.tar.gz
tar xzf ../../ovlm-install/jce/jce-policy-8.tar.gz
mv UnlimitedJCEPolicyJDK8/* jdk1.8.0_60/jre/lib/security/
rm -rf UnlimitedJCEPolicyJDK8
```

4. Create the keycloak.jks file using the following commands:

```
openssl pkcs12 -export -name weaver -in server.crt -inkey server.key -out
server.p12 -passout pass:<password>

keytool -importkeystore -destkeystore keycloak.jks -srckeystore server.p12
-srcstoretype pkcs12 -alias weaver -srcstorepass password -noprompt
-deststorepass password -deststoretype JKS -destalias keycloak
```

Installing Keycloak

If you are using Keycloak in your Openet Weaver deployment you must install it on each management node, so for geo-redundant deployments you must install it on the active node and the passive node.

Before you can install Keycloak you must copy the following files from the Openet Weaver installer to a management node:

- ovlm-install/rpms/keycloak-pkg-1.2.3-x86_64.rpm
- ovlm-install/jre/server-jre-8u60-linux-x64.tar.gz
- ovlm-install/jce/jce-policy-8.tar.gz
- keycloak_keystore.jks, which you created in [Preparing Certificates for Keycloak](#).

Next you select a location and install Keycloak from the RPM using either of the following commands:

```
#RPM installation
sudo rpm -ivh keycloak-pkg.rpm

#YUM installation
sudo yum -y install keycloak-pkg.rpm
```

Note: See [Hardware and Software Requirements](#) for more information about Keycloak versions and requirements.

The Keycloak RPM creates a dedicated Keycloak user and group by default.

By default, Keycloak is installed with /opt as the installation directory with the directories shown in the following table. These directories can be relocated as part of installing Keycloak:

Directory	Description
/opt/keycloak	The main application directory.
/etc/keycloak	The location of the configuration files.
/var/log/keycloak	The location of the log files.
/var/run/keycloak	The location of the PID file
/var/opt/keycloak	The location of runtime data, such as database files.
/usr/lib/systemd/system	

To relocate any of the above directories use the --relocate parameter when installing from the RPM, for example:

```
sudo rpm --ivh --relocate /opt/keycloak=/usr/local/keycloak keycloak-pkg.rpm
```

All directories are owned by a keycloak user except for /usr/lib/systemd/system, and are not visible to users who are not in the Keycloak group.

Note: If you relocate /usr/lib/systemd/system, systemd integration becomes unavailable.

Installing Keycloak as the Weaver Service Owner

You have the option of installing Keycloak as the Weaver Service Owner user. That is, Openet Weaver installs and executes as the ovlm user by default, but Keycloak can be installed to run as the same user as Openet Weaver.

To install Keycloak as a custom user, export the KEYCLOAK_USER and KEYCLOAK_USER_GROUP environment variables prior to Keycloak installation, as follows:

```
export KEYCLOAK_USER=ovlm
export KEYCLOAK_USER_GROUP=ovlm
```

Removing Keycloak

In the event that you want to remove Keycloak after installation, run the following command from the Keycloak main application directory:

```
sudo rpm -e keycloak
```

RPM removes only the files that it knows about, which are the files that it installed. After removal, some runtime files might be present in the following directories:

- /etc/keycloak
- /opt/keycloak
- /var/log/keycloak
- /var/opt/keycloak
- /var/run/keycloak

You must delete any files remaining in those directories to purge the Keycloak installation completely. The keycloak user and group must also be deleted manually if you want to remove them.

Note: Files in /etc/keycloak are marked as configuration files and are backed up with postfix .rpmsave when you remove the Keycloak package. Therefore you have a backup of configuration in the event you reinstall or upgrade Keycloak.

Configuring Keycloak

This section describes the post installation tasks you need to perform after installing Keycloak.

Configuring the Keystore

To begin configuring Keycloak, copy your Keystore to `$JBOSS_HOME/standalone/configuration` as shown in the following example:

```
sudo cp /path/to/keycloak.jks /etc/keycloak/standalone
```

Configuring the Standalone Configuration

You must modify the `etc/keycloak/standalone/standalone.xml` to refer to the keycloak keystore by adding two blocks to the file and adding an administrator. You might also want to change the port settings.

You add the following block under `<security-realms>`:

```
<security-realm name="UndertowRealm">
    <server-identities>
        <ssl>
            <keystore path="keycloak.jks" relative-to="jboss.server.config.dir" keystore-password="password" />
        </ssl>
    </server-identities>
</security-realm>
```

You add the following block above the existing `<http-listener>`

```
<https-listener name="https" socket-binding="https" security-realm="UndertowRealm"/>
```

Configuring keycloak.conf

You might need to open and edit `/etc/keycloak/keycloak.conf` to change values for one or more of the following items:

- The Java path
- The bind address

If Java 8 is not available on the path to all users, then you will must point to the specific Java 8 installation by uncommenting `JAVA_HOME` then adding the location of your Java installation as shown in the following example:

```
JAVA_HOME=/usr/java/jdk1.8.0_144
```

The `KEYCLOAK_BIND` setting is used to bind IP addresses. By default, the Keycloak RPM installas with IPv4 settings. The following example binds all IPv4 addresses, which is the default:

```
KEYCLOAK_BIND=0.0.0.0
```

You can change the value to bind only specific interfaces as in the following example:

```
KEYCLOAK_BIND=192.58.201.202
```

Configuring keycloak.properties

The keycloak.properties file contains a list of port mappings. The Wildfly application server used by Keycloak requires several active ports. You can change the ports in the keycloak.properties file. The HTTP and HTTPS ports are the ports you use to access Keycloak.

To update the HTTPS port in the keycloak.properties file, use the following command:

```
sudo bash -c "sed -i \"s|\(jboss[.]https[.]port\)=.*|\1=<port_number>|g\" /opt/ovlm/keycloak/etc/keycloak.properties"
```

Creating a Keycloak Admin User

You can create a Keycloak admin user and password that can be used to create and manage user accounts by entering the following command:

```
sudo -u <username> bash -c "export JAVA_HOME=/usr/java/jdk1.8.0_144; /opt/keycloak/bin/add-user-keycloak.sh -u <admin_user> -p <admin_password>"
```

Starting, Testing, and Stopping Keycloak

This topic describes how to start, test, and stop Keycloak.

Starting Keycloak

To start Keycloak by use the following command:

```
systemctl start keycloak
```

Testing Keycloak

Use the following command to perform a sanity test to verify that the SSL certs are working, replacing the URL below with the one that corresponds to your deployment.

```
curl --cacert /path/tocerts/ca_bundle https://<keycloak-server>:<port_number>/
```

Assuming setup was successful, you can now use your browser to log into the system, using the administrator user you created. Keycloak is now ready for Openet Weaver installation.

Stopping Keycloak

For normal operations where you want security authorization enabled, Keycloak must be kept running. However, In the event that you want to To stop Keycloak use the following command:

```
systemctl stop keycloak
```

After Installing, configuring, and testing Keycloak, the next step you take depends on what Openet Weaver Features you are deploying:

- If you intend to create VNFs with VIM, refer next to [Configuring VIM \(OpenStack\) for Openet Weaver](#).
- Otherwise, if you are ceating a highly available (HA) deployment, refer next to [Configuring Master and Hot-Standby Nodes for High Availability](#).
- Otherwise, if you are implementing none of the above, refer next to [Creating an Installation File](#).

Configuring VIM (OpenStack) for Openet Weaver

This topic describes how to configure VIM for OpenStack by mapping properties and changing default security settings.

This topic describes VIM configuration for Openet Weaver in the following sections.

Mapping and Configuring Glance Image Properties

The VIM is used in the allocation of resources to VNFs when instantiated. Each VNF configured for VIM has a VNF descriptor (VNFD) that defines the virtual deployment units (VDUs). VDUs are composed of properties that specify the hardware and operating system resources the VNF requires. The following is an example of VNF requirements specified in a VNFD:

```
capabilities:
host:
properties:
disk_size: 4 GB
mem_size: 4096 MB
num_cpus: 2
os:
properties:
architecture: x86_64
distribution: centos
type: linux
version: 7.1
```

In the above VNFD example, the host properties specify hardware requirements and the os properties specify operating system requirements.

In order for the values configured in a VNFD to have any effect, the VM infrastructure has to be matched with the configuration of the VIM used with Openet Weaver, which is OpenStack. OpenStack uses virtual hardware templates called flavors for defining hardware requirements. In addition, OpenStack uses Glance as a catalog and repository for virtual disk images.

Mapping Host Properties

In the above VNFD example, the host properties are mapped to the OpenStack glance flavor and the os properties are mapped to the OpenStack glance image.

Openet Weaver works only with flavors that are defined in the OpenStack environment. The following matching priority is used to find the flavor which is best match to the host properties:

1. RAM / Memory
2. Number of CPUs
3. Disk Size

The priority order means that Openet Weaver first filters for flavors that match the mem_size property from the VNFD. Then it filters for flavors that match the num_cpus property. Finally, it filters for the num_cpus properties. If there are multiple flavors with same configuration (i.e. same RAM, same VCPUs, and same Root Disk), then the selection of flavor is undefined.

Mapping OS Properties for the Resource Agent Image

With VNFD os properties, Openet Weaver looks for the OpenStack image that has the same properties as those defined in the VNFD, with the mapping shown in the following table.

Table 17: VNFD to OpenStack Image Mapping

VNFD properties	OpenStack Glance Image Properties
architecture	architecture
distribution	os_distro
type	os_type
version	os_version

If there are multiple images with same properties (a.k.a. metadata), then the selection of image is undefined. An error is returned when there is no match for os properties.

Configuring OpenStack Glance RAgent Image Properties

OpenStack glance image properties are configured for the Resource Agent image only. To configure the properties manually follow these steps:

1. Login to the OpenStack dashboard (Horizon) and select **Images** in the navigation pane.

The Images page displays.

Image Name	Type	Status	Public	Protected	Format	Size	Actions
WeaveCentOS-7	Image	Active	Yes	Yes	QCOW2	370.8 MB	Launch Instance
CentOS_Img	Image	Active	No	No	QCOW2	8.7 MB	Create Volume
CentOS	Image	Active	No	No	QCOW2	427 bytes	Update Metadata
CentOS_Img	Image	Active	No	No	QCOW2	372.0 MB	Launch Instance
CentOS	Image	Active	No	No	QCOW2	8.7 MB	Launch Instance
CentOS_Img	Image	Active	Yes	No	QCOW2	12.7 MB	Launch Instance
CentOS	Snapshot	Active	No	Yes	QCOW2	21.1 MB	Launch Instance
CentOS_Img	Snapshot	Active	Yes	Yes	QCOW2	1.2 GB	Launch Instance
CentOS	Image	Active	Yes	Yes	QCOW2	370.8 MB	Launch Instance

2. Find the Openet Weaver Resource Agent image in the Image name column, then select **Update Metadata** in the Actions column.

The Update Image Metadata dialog box displays.

3. Add custom metadata to the image as per the following list:

Custom Metadata:

- architecture
- os_distro
- os_type
- os_version

Update Image Metadata

You can specify resource metadata by moving items from the left column to the right column. In the left columns there are metadata definitions from the Glance Metadata Catalog. Use the "Other" option to add metadata with the key of your choice.

Available Metadata	
Custom	os_distro
No available metadata	

Existing Metadata	
architecture	x86_64
description	CentOS with pre-i

You can specify resource metadata by moving items from the left column to the right column. In the left columns there are metadata definitions from the Glance Metadata Catalog. Use the "Other" option to add metadata with the key of your choice.

4. Supply a value for each of the custom metadata fields you created in [step 3](#).

The metadata values must map to the corresponding VNFD properties.

Note: See [Table 1](#) for how the custom metadata maps to the VNFD file.

The screenshot shows the 'Update Image Metadata' dialog. On the left, under 'Available Metadata', there is a 'Custom' entry with a '+' button. Below it, a message says 'No available metadata'. On the right, under 'Existing Metadata', there are four entries: 'architecture' (x86_64), 'description' (CentOS with pre-), 'os_distro' (centos), and 'os_version' (7.1). The 'os_version' entry is highlighted with a blue background. At the bottom, there is a note about 'os_version (os_version)' followed by 'Cancel' and 'Save' buttons.

Available Metadata

Custom +

No available metadata

Existing Metadata

architecture	x86_64	-
description	CentOS with pre-	-
os_distro	centos	-
os_type	linux	-
os_version	7.1	-

os_version (os_version)

Cancel Save

5. Click **Save** to save the updated metadata.
Confirm that your metadata saved correctly before exiting Horizon.
6. On the Images page, click the Image Name of the Openet Weaver image.
The image details page for the image displays.

The screenshot shows the 'Image Details' page for 'Weaver.CentOS-7'. The 'Information' section includes fields like Name (WeaverCentOS-7), Description (CentOS with pre-installed Resource Agent), ID (623b0af5-wbb3-4a10-a154-6bdab447c1f1), Owner (admin), Status (Active), Protected (No), Checksum (3bf1867b9272befffbfb43e607914), and Last Updated (July 26, 2016, 3:25 p.m.). The 'Specs' section shows Size (8 bytes), Container Format (BARE), and Disk Format (QCOW2). The 'Custom Properties' section contains entries for architecture (x86_64), os_distro (centos), os_type (LINUX), and os_version (7.1).

At the bottom of the page you can see the Custom properties created in [step 3](#) and [step 4](#).

Related concepts

[Creating and Uploading a VNF Descriptor \(VNFD\) File](#) on page 182

Changing OpenStack Default Security Settings

By default, the security rules for an OpenStack project/tenant are as follows:

- Allow all outgoing connection from OpenStack's VM instance.
- Block all incoming connection to OpenStack's VM instance.

This means that OpenStack cannot receive communication from Openet Weaver until the incoming connection setting is changed.

To change the default settings so that OpenStack can communicate with Openet Weaver, follow these steps:

1. Login to the OpenStack dashboard (Horizon) and select **Project** in the navigation pane to expand that section.

The screenshot shows the 'Usage Overview' page. The 'Usage Summary' section allows selecting a period of time to query usage. The 'Select a period of time to query its usage:' field has 'From: 2016-07-28' and 'To: 2016-07-29' selected. A 'Submit' button is present. Below this, a message states 'Active Instances: 0 Active RAM: 0Bytes This Period's VCPU-Hours: 396.23 This Period's GB-Hours: 7800.80 This Period's RAM-Hours: 801628.59'. The 'Usage' section displays a table with columns: Project Name, VCPUs, Disk, RAM, VCPU Hours, Disk GB Hours, and Memory MB Hours. One item is listed: admin (0 VCPUs, 0 Bytes, 396.23 VCPU Hours, 7800.80 Disk GB Hours, 801628.59 Memory MB Hours). A 'Download CSV Summary' link is also visible.

2. Select **Compute > Access & Security** under Project in the Navigation pane.
The Access & Security page displays.

The screenshot shows the 'Access & Security' page in the OpenStack Horizon interface. The left sidebar is collapsed, and the main area displays the 'Access & Security' section. A table lists one item: 'default' with a description 'Default security group'. A 'Manage Rules' button is visible in the 'Actions' column for this row.

- Locate the default security group and click the **Manage Rules** button in that row.
The Manage Security Group Rules page displays.

The screenshot shows the 'Manage Security Group Rules' page for the 'default' security group. The table lists six rules:

Direction	Ether Type	IP Protocol	Port Range	Remote IP Prefix	Remote Security Group	Add Rule	Actions
Egress	IPv6	Any	Any	/0	-	+ Add Rule	Delete Rule
Egress	IPv4	Any	Any	0.0.0.0/0	-	+ Add Rule	Delete Rule
Ingress	IPv6	Any	Any	-	default	+ Add Rule	Delete Rule
Ingress	IPv4	Any	Any	-	default	+ Add Rule	Delete Rule
Ingress	IPv4	ICMP	Any	0.0.0.0/0	-	+ Add Rule	Delete Rule
Ingress	IPv4	TCP	22 (SSH)	0.0.0.0/0	-	+ Add Rule	Delete Rule

- Click the **Add Rule** button.
The Add Rule page displays.

The screenshot shows the 'Add Rule' page. The form fields are as follows:

- Rule ***: Custom TCP Rule
- Direction**: Ingress
- Open Port ***: Port 28030
- Remote ***: CIDR 0.0.0.0/0

Description:

Rules define which traffic is allowed to instances assigned to the security group. A security group rule consists of three main parts:

Rule: You can specify the desired rule template or use custom rules, the options are Custom TCP Rule, Custom UDP Rule, or Custom ICMP Rule.

Open Port/Port Range: For TCP and UDP rules you may choose to open either a single port or a range of ports. Selecting the "Port Range" option will provide you with space to provide both the starting and ending ports for the range. For ICMP rules you instead specify an ICMP type and code in the spaces provided.

Remote: You must specify the source of the traffic to be allowed via this rule. You may do so either in the form of an IP address block (CIDR) or via a source group (Security Group). Selecting a security group as the source will allow any other instance in that security group access to any other instance via this rule.

- Create a new rule by changing the following OpenStack security default:

- Replace the value of the Rule field by selecting **Custom TCP Rule**
- Replace the value of the Direction field by selecting **Ingress**
- Replace the value of the Open Port field by selecting **Port**
- In the Port field type in the number of the Resource Agent port

Note: The Resource Agent port number can be found in the Openet Weaver installation file.

6. Click the **Add** button to add the new rule.

Related tasks

[Creating an Installation File](#) on page 92

Creating an Image for Resource Agents

In order to instantiate VNFs in an OpenStack environment, Openet Weaver requires Resource Agents to have additional libraries on all runtime nodes assigned to VNF. To help you achieve this, Openet Weaver provides a `prepare-ra-image.py` script as a part of the product installer. The purpose of the script is to automate creation of the Openet-Weaver-ready cloud image and dependencies used during instantiation in an OpenStack environment with on-the-fly VM creation. The script is written on Python and uses the `guestfish` (<http://libguestfs.org/>) tools for editing files inside guests, scripting changes to VMs.

After creating the image, perform the following tasks to configure OpenStack for using the image and verifying that it works:

- Upload the image
- Import a key pair
- Launch an instance
- Connect to the instance using SSH

these tasks are linked as sub-procedures at the bottom of this topic.

Software Requirements

You can download Cloud image from the official CentOS repository <http://cloud.centos.org/centos/7/images/>.

The script requires you to download and install the following third-party tools:

- `guestfish ("libguestfs", "libguestfs-tools-c")`libraries)
- `xz`, which is a compression tool

If these tools are missing from the system, the script tries to install them by itself and prompt you to approve them. In this event ensure the script is running under permissions which allow you to install software packages (root or sudo user). If they are already installed root permissions are not required.

Hardware Requirements

The hardware requirements for running the script are as follows:

- 2.5 GB disk space
- 1 GB RAM

Script Usage

You can find the `prepare-ra-image.py` script in `/ovlm-install/utilities/cloud-runtime-image/prepare-ra-image.py`.

The command to run the script is `prepare-ra-image.py [-h] -i image_path [-c] [-s] [-p root_password]`

The script parameters are described in the following table.

Table 18: prepare-ra-image.py Script Parameters

Parameter	Meaning	Mandatory	Description	Comments
-h	help	No	Show this help message and exit.	
-i	image-path	Yes	The path to the qcow image (supported qcow/qcow2/qcow.xz/qcow2.xz extensions).	The script only supports qcow/qcow2 images. Using a different format results in an error.
-c	compression-mode	No	Enables xz compression for the output image.	You can compress the target image (also with xz archiver) by passing the -c option.
-s	silent-mode	No	Skip the third-party tools installation prompt.	To skip the confirmation message and install all required dependencies automatically, run the script in a silent mode with the -s option.
-p	root-password	No	Set a password for the root user of the image.	

If the image is compressed by the xz tool and has an .xz extension it is decompressed. If the image has a different compression format or extension the script raises the error. The script modifies the decompressed image. The target image location is same as source image location.

The script provides short status messages printed to the stdout. If you require more detail info check the log file located in the same folder.

The following is an example run of the prepare-ra-image.py script

```
[test-user@ovlm-demo ovlm-install]$ ./utilities/cloud-runtime-image/prepare-ra-image.py -i
..../images/centos-7-1503-x64_86-GenericCloud.qcow.xz
The next thirdparty utilities are installed: libguestfs, libguestfs-tools-c
[y/n]: y
Decompressing image "/home/test-user/images/centos-7-1503-x64_86
-GenericCloud.qcow.xz":
/home/test-user/images/centos-7-1503-x64_86-GenericCloud.qcow.xz (1/1)
 100 %      273.4 MiB /  958.4 MiB = 0.285      43 MiB/s      0:22

Start image configuration:
Cleanup image yum repository cache
Upload and install jre on image
Upload and install package "/home/test-user/ovlm-install/rpms/python-ps
util-2.2.1-1-el7.x86_64.rpm" on image
The package "python-psutil-2.2.1-1-el7.x86_64.rpm" is installed successfully
Upload and install package "/home/test-user/ovlm-install/rpms/facter-2.
4.6-1.el7.x86_64.rpm" on image
The package "facter-2.4.6-1.el7.x86_64.rpm" is installed successfully
Upload and install package "/home/test-user/ovlm-install/rpms/net-snmp-
libs-5.7.2-24.el7_2.1.x86_64.rpm" on image
The package "net-snmp-libs-5.7.2-24.el7_2.1.x86_64.rpm" is installed
successfully
Upload and install package "/home/test-user/ovlm-install/rpms/net-snmp-
utils-5.7.2-24.el7_2.1.x86_64.rpm" on image
The package "net-snmp-utils-5.7.2-24.el7_2.1.x86_64.rpm" is installed
successfully
Upload and install package "/home/test-user/ovlm-install/rpms/ovlm-reso
urce-agent-1.5.1-1325.x86_64.rpm" on image
The package "ovlm-resource-agent-1.5.1-1325.x86_64.rpm" is installed
```

```
successfully
Configuration is finished successfully. The RA target image
"/home/test-user/images/centos-7-1503-x64_86-GenericCloud.qcow"
is ready to deploy
```

Installing Software Packages

The script installs the software packages required for the runtime nodes. The list of software packages is defined in the script. Internet Access on the configuration (host) node is required to resolve the dependencies on the image (guest node) during YUM installation. If any VNFC packages are pre-installed in the same or newer version, installation is skipped for those software packages. Newer versions are installed where older ones exist.

Software Packages Installed to the CentOS 7 Cloud Image

The following factor packages are installed as part of running the script:

- hwdata base-repo
- pciutils base-repo
- ruby base-repo
 - ruby-irb base-repo
 - ruby-libs base-repo
 - rubygem-bigdecimal base-repo
 - rubygem-io-console base-repo
 - rubygem-json base-repo
 - rubygem-psych base-repo
 - rubygem-rdoc base-repo
 - rubygems base-repo

The following software packages are also installed:

- net-snmp
 - net-snmp-libs
 - net-snmp-utils
- jre1.8.0_60
- python-psutil
- ovlm-resource-agent
 - python-jinja2 - optional-repo

Uploading an Image in OpenStack

To upload the image created for OpenStack follow these steps.

1. Log in to the OpenStack dashboard.
2. Select the associated project from the drop-down menu at the top left.
3. Open **Projects > Compute** and click **Images**.
4. Click **Create Image**.

The Create an Image dialog box is displayed.

Create An Image

Name *

Description

Description:
Currently only images available via an HTTP/HTTPS URL are supported. The image location must be accessible to the Image Service.

Please note: The Image Location field MUST be a valid and direct URL to the image binary. URLs that redirect or serve error pages will result in unusable images.

Image Source

Image File

Format *

QCOW2 - QEMU Emulator

Architecture

Minimum Disk (GB) ?

Minimum RAM (MB) ?

Public

Protected

Create Image

- Enter field values as described in the following table.

Field	Value
Image Name	Enter a name for the image.
Image Description	Enter a brief description of the image. You can leave this field empty.
Image Source	Choose the image source from the drop-down list. Your choices are Image Location and Image File.

Field	Value
Image File or Image Location	Based on your selection for Image Source, either enter the location URL of the image in the Image Location field or browse for the image file on your file system and add it.
Format	Select the image format QCOW2 for the image.
Architecture	Specify the architecture. For example, i386 for a 32-bit architecture or x86_64 for a 64-bit architecture. You can leave this field empty.
Minimum Disk (GB)	Leave this field empty.
Minimum RAM (MB)	Leave this field empty.
Public	The access permission for the image.
Protected	Select this check box to ensure that only users with permissions can delete the image. Possible values are: <ul style="list-style-type: none"> • Yes • No

6. Click **Create Image**.

When you click the button tThe image is queued to be uploaded. It might take some time before the status changes from Queued to Active.

Creating a Key Pair

You must generate a key pair for secure SSH tunneling.

To generate a key pair, follow these steps:

1. At the Openet Weaver CLI, generate SSH key pairs with the ssh-keygen command `ssh-keygen -t rsa -f cloud.key`.
2. Log in to the OpenStack dashboard.
3. Select the associated project from the drop-down menu at the top left.
4. Open **Projects > Compute** and click the **Access & Security** category.
5. Click the **Key Pairs** tab, which shows the key pairs that are available for this project.
6. Click **Import Key Pair**.

the Import Key Pair dialog box opens.

7. Enter a name for your key pair.
8. In the **Public Key** field, copy the contents of the public key file you generated in step 1.
9. Click **Import Key Pair**.
10. Save the *.key file locally.
11. Change the permissions so that only you can read and write to the file by running the command `$ chmod 0600 <yourPrivateKey>.key`.
12. Make the key pair known to SSH by running the command `$ ssh-add <yourPrivateKey>.key`.

The Compute database registers the public key of the key pair.

The Dashboard lists the key pair on the Access & Security tab.

Launching a VM Instance in OpenStack

To launch a VM instance in OpenStack, follow these steps:

1. Log in to the OpenStack dashboard.

2. Select the associated project from the drop-down menu at the top left.

3. Open **Projects > Compute** and click the **Instances** category.

the dashboard shows each instance with details including the following:

- Name
- Private IP address
- Floating IP address
- size
- Status
- Task
- Power state

4. Click **Launch Instance**.

The Launch Instance dialog box displays.

5. On the **Details** tab of the Launch iInstance dialog box, enter a name for the VM in the **Instance Name** field.

6. On the **Image** field of the **Source** tab select **Boot Source** and select the image you created using the script from the list.

7. On the **Flavor** tab specify the size for the instance you are launching.

Note: The flavor is selected based on the size of the image selected for launching an instance. For example, while creating an image, if you have entered the value in the Minimum RAM (MB) field as 2048, then on selecting the image, the default flavor is m1.small.

8. On the **Key Pair** tab select the Import Key pair.

9. Click **Launch Instance**.

Connecting to the VM Instance Through SSH

You use the downloaded key pair to connect to your instance using SSH. To set this up follow these steps:

1. Copy the IP address for your VM instance.

2. At the CLI, make a secure connection to the instance using the command `ssh -i key centos@instance_ip_addr`.

3. At the prompt, type yes.

4. Start Resource Agent microservice service using the command `sudo systemctl start ovlm-ra`.

5. Check microservice status using the command `sudo systemctl status ovlm-ra`

You should see a response similar to the following.

```
ovlm-ra.service - OVLM Resource Manager Agent
   Loaded: loaded (/usr/lib/systemd/system/ovlm-ra.service; disabled)
   <<< CHECK >>>   Active: active (running) since Thu 2016-12-01 16:46:51 UTC;
     1min 20s ago
     Process: 975 ExecStart=/etc/init.d/ovlm-ra start (code=exited,
       status=0/SUCCESS)
     Process: 971 ExecStartPre=/bin/chown -R ovlm:ovlm /var/run/ovlm (code=exited,
       status=0/SUCCESS)
     Process: 970 ExecStartPre=/bin/mkdir -p /var/run/ovlm (code=exited,
       status=0/SUCCESS)
     Main PID: 979 (java)
        CGroup: /system.slice/ovlm-ra.service
                  └─979 /usr/lib64/ovlm/java/jdk1.8.0_60/bin/java -Xms256M
                     -Xmx256M
                     /usr/lib64/ovlm/ra/resource-agent-rest-impl-1.5.1-SNAPSHOT-microse
                     rvice.jar
```

6. Display the microservice log using the command `tail -n5 /var/log/ovlm/ra/ovlm-ra.log`

You should see a response similar to the following.

```
$ tail -n 5 /var/log/ovlm/ra/ovlm-ra.log
2016-12-01 16:46:50.415  INFO 979 --- [           main] o.s.web.servlet.DispatcherServlet
: FrameworkServlet 'dispatcherServlet': initialization completed in 67
ms
2016-12-01 16:46:50.673  INFO 979 --- [           main] o.eclipse.jetty.server.ServerConnector
: Started ServerConnector@3b0b183c{HTTP/1.1}{0.0.0.0:28030}
2016-12-01 16:46:50.676  INFO 979 --- [           main] .s.b.c.e.j.JettyEmbeddedServletContainer
: Jetty started on port(s) 28030 (http/1.1)
2016-12-01 16:46:50.691  INFO 979 --- [           main] com.openet.modules.ovlm.ra.Application
: Started Application in 20.649 seconds (JVM running for 23.384)
<<< CHECK >>> 2016-12-01 16:46:50.691  INFO 979 --- [           main]
com.openet.modules.ovlm.ra.Application
: OVLM Resource Agent is ready
```

After configuring VIM (OpenStack) for Openet Weaver, the next step you take depends on what Openet Weaver Features you are deploying:

- If you are creating a highly available (HA) deployment, refer next to [Configuring Master and Hot-Standby Nodes for High Availability](#).
- Otherwise, refer next to [Creating an Installation File](#).

Configuring Master and Hot-Standby Nodes for Local High Availability

This topic describes how to setup keepalived on Openet Weaver VNF-M nodes for local Master and hot-standby support to ensure local high availability of the system.

Implementing keepalived-based high availability requires that master and hot-standby nodes be local, that is, in the same datacenter.

Keepalived uses the Virtual Router Redundancy Protocol (VRRP) protocol to monitor the state of the master and hot-standby nodes and perform virtual IP switching to the hot-standby node when it detects a fault on master node. An example of a network using keepalived is shown below.

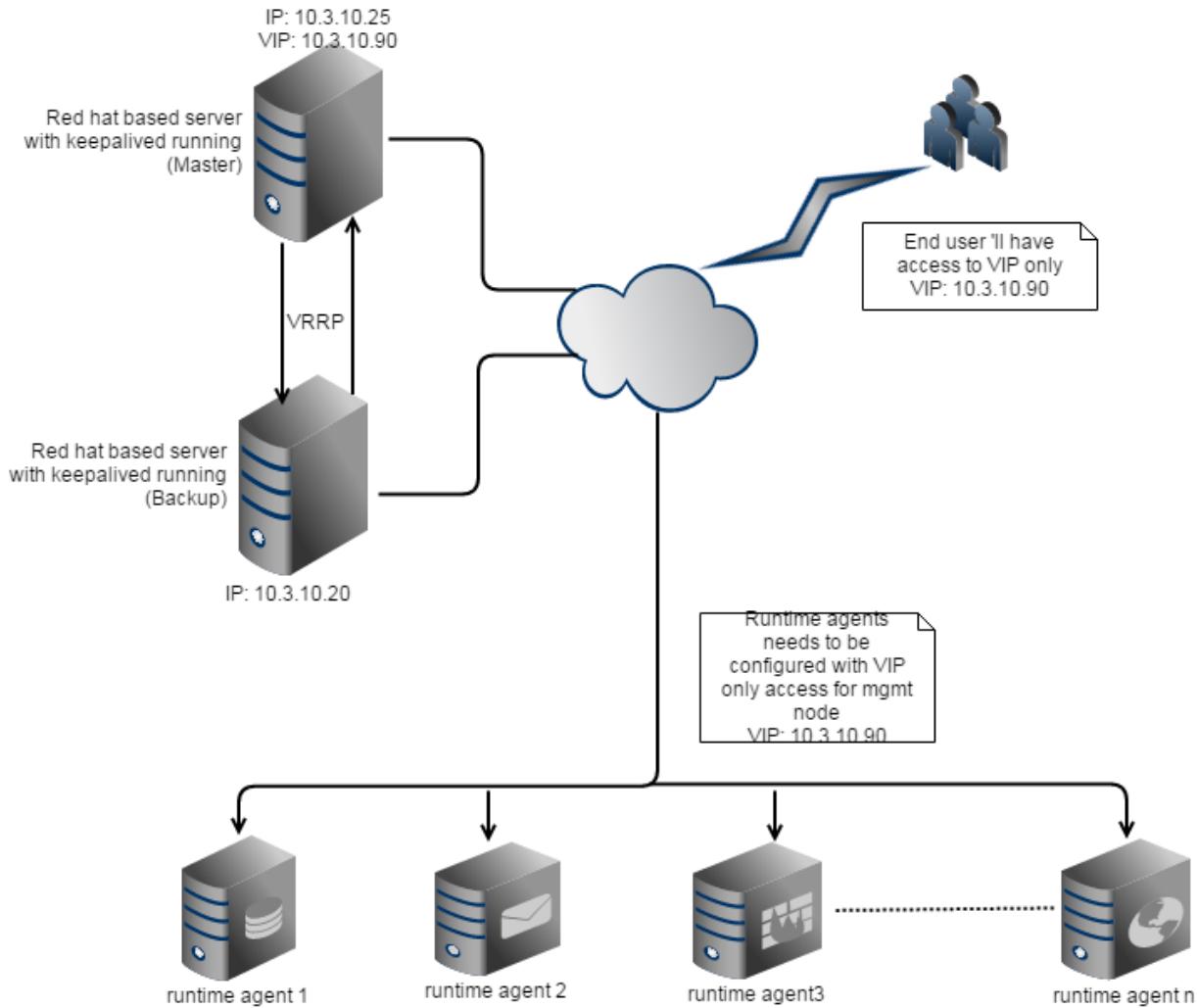


Figure 9: Network with keepalived deployed

As a prerequisite to setting up keepalived, perform the following tasks:

- Allocate a valid Virtual IP (VIP) This is an IP address that doesn't correspond to an actual physical network interface but is in the valid IP range for that Network.
- Modify iptable rules on the master and hot-standby nodes to allow VRRP traffic (keepalived communicates on the VRRP protocol). VIP failover does not work unless these rules are modified.

In the following example firewall rules are added to allow VRRP communication using the multicast IP address 224.0.0.18 and the VRRP protocol (112) on each network interface that Keepalived controls. The commands below are run on the master and hot-standby nodes.

```
> iptables -I INPUT -i <network_interface> -d 224.0.0.0/8 -p vrrp -j ACCEPT
> iptables -I OUTPUT -o <network_interface> -d 224.0.0.0/8 -p vrrp -j ACCEPT
> service iptables save
```

Note: The configuration described for master and hot-standby support is valid only when all Openet Weaver management services are installed on one VM.

Before you can deploy keepalived on Openet Weaver you must install the prerequisite software, configure SNMP traps on the target hosts, and configure snmpd with Agentx support. To do so, follow these steps:

1. Install the software packages listed below on the master and hot-standby nodes using the command `sudo yum install package`

The following software packages are required to for keepalived. They are listed with the minimum release requirement:

- keepalived release 1.2.13-8.el7.x86_64
- net-snmp release 5.7.2-24.el7_2.1.x86_64
- net-snmp-utils release 5.7.2-24.el7_2.1.x86_64
- python-jinja2 release 2.7.2-2.el7.noarch

Note: Notify script triggered by keepalived in-case of VIP failover uses the snmptrap command to send the traps.

2. On each target host, open the `/etc/snmp/snmptrapd.conf` file for editing and add the following line:

```
authCommunity log,execute,net public
```

Note: Adding this line allows traps to be captured.

3. On each target host, start snmptrapd so that it can receive traps by running the command `sudo service snmptrapd start`
4. On each target host, set snmptrapd to start automatically at system startup by running the command `chkconfig snmptrapd on`
5. Verify that KEEPALIVED-MIB is copied into `/usr/share/snmp/mibs` on all target hosts.

The file is required by the hosts to interpret received traps. By default, keepalived installation extracts KEEPALIVED-MIB.txt at the following location: `/usr/share/snmp/mibs/KEEPALIVED-MIB.txt`

Next you configure snmpd to support AgentX so that keepalived can automatically send status reports through traps to the target nodes.

6. On each master and backup node open `/etc/snmp/snmpd.conf` for editing.
7. In each snmpd.conf file instance, add the following line to enable snmp systemview to decode KEEPALIVED-MIB:

```
view systemview included .1.3.6.1.4.1.9586.100.5
```

8. In each snmpd.conf file instance, add the following line to grant write access for KEEPALIVED-MIB systemview so that values can be manipulated to control keepalived behavior.

```
access notConfigGroup ""any noauth exact systemview systemview none
```

9. In each snmpd.conf file instance, add the following lines to enable agentx support:

```
rocommunity public
master agentx
```

10. In each snmpd.conf file instance, add the target host on which snmptrapd is running to the trapsink attribute as shown below, replacing <target_host_ip> with the IP of the specific host:

```
trapsink <target_host_ip> public
```

At this point the snmp.conf file should look similar to the following:

```
# Make at least snmpwalk -v 1 localhost -c public system fast again.
#      name          incl/excl    subtree          mask(optional)
view   systemview   included   .1.3.6.1.2.1.1
view   systemview   included   .1.3.6.1.2.1.25.1.1
view   systemview   included   .1.3.6.1.4.1.9586.100.5
===== line added for
KEEPALIVED-MIB (OID=.1.3.6.1.4.1.9586.100.5)
```

```

# Finally, grant the group read-only access to the systemview view.
#         group          context sec.model sec.level prefix read   write  notif
#access  notConfigGroup ""      any       noauth    exact   systemview none
none
access  notConfigGroup ""      any       noauth    exact   systemview systemview
none ======> updated line to grant write access for the view

# =====
# Settings to enable agentx support
# =====
rocommunity public
master agentx

# =====
# Target node on which snmptrapd is running needs to be configured here
# agentx will use this to send any traps generated by agentx
# =====
trapsink <targer_host_ip> public

```

11. On each master and backup node, enable snmpd by running the command `sudo systemctl enable snmpd`
 12. On each target host, set snmpd to start automatically at system startup by running the command `sudo systemctl start snmpd`
- After performing these steps you can deploy keepalived to Openet Weaver. The steps required to do that depend on whether you are using the HTTP or HTTPS protocol.

After deployment you must configure keepalived. The procedure is the same for HTTP and HTTPS.

Deploying keepalived for HTTP

This topic describes how to deploy keepalived for HTTP. You must configure Openet Weaver installer parameters for keepalived support and for installing Weaver on hot-standby and master nodes.

In the installation file, set override-host parameter for each microservice. The parameter stores the VIP value that is used to access master and hot-standby nodes. The installation file also requires a block for keepalived to specify the hostname of the target master and hot-standby Nodes so that the dependent packages can be installed on those nodes.

Note: See *Creating an Installation File* for more information about setting installation file parameters.

This step can be performed before Openet Weaver installation, and therefore can be completed when you first run the installer. If you choose to configure the parameters after installation you must run the installer again with the updated install file as in the following example.

```
./ovlm-deploy.sh -i ../ovlm-install-sample.yml -u ovlminstall
```

The installer copies the keepalived configuration scripts to `/usr/lib64/ovlm/utilities/keepalived/` in the subdirectories shown below on master and hot-standby nodes.

```

> cd /usr/lib64/ovlm/utilities/keepalived/
> tree
.
├── samples
│   └── configuration
│       ├── nopreempt
│       │   ├── keepalived.conf.BACKUP
│       │   └── keepalived.conf.MASTER
│       └── preempt
│           └── keepalived.conf.BACKUP

```

```

    └── keepalived.conf.MASTER
scripts
├── generate-vnfm-monitor.sh
├── keepalived-setup.sh
└── python
    ├── setup
    │   ├── keepalived-conf.py
    │   └── templates
    │       └── keepalived.conf.jinja
    └── vnfm-monitor
        ├── templates
        │   ├── monitor.jinja
        │   └── notify.jinja
        └── vnfm-monitor.py
systemd
└── keepalived.service
11 directories, 12 files

```

⚠ Warning: When using the override-host parameter, the ovlm-deploy command cannot include the --create-sample-vnf option as keepalived is not yet running so there is no virtual IP assigned to master or hot-standby node. after keepalived is up and running you can execute the install script with the --create-sample-vnf option.

Deploying keepalived for HTTPS

To deploy keepalived on master and hot-standby nodes running the HTTPS protocol, follow these steps:

1. Update /etc/hosts to resolve VIP allocations using the hostname on the master node, hot-standby node, and all Resource Agents.

This step is required because HTTPS certificates are associated with the hostname or domain name of the host. You cannot generate certificates for https using IP addresses only.

The following is an example of /etc/hosts with VIP allocations.

```

> sudo cat /etc/hosts
127.0.0.1    localhost localhost.localdomain localhost4 localhost4.localdomain
4
::1          localhost localhost.localdomain localhost6 localhost6.localdomain
6
10.3.10.XX   master-ip.node.lan master-ip
===== > hostname for VIP
10.3.18.XX   node-master.node.lan node-master
10.3.18.XX   node-backup.node.lan node-backup
192.168.XX.XX rt-node-1.node.lan rt-node-1
192.168.XX.XX rt-node-2.node.lan rt-node-2
192.168.XX.XX rt-node-3.node.lan rt-node-3
192.168.XX.XX rt-node-4.node.lan rt-node-4

```

2. Configure Weaver installer parameters for keepalived support and for installing Weaver on hot-standby and master node.

In the installation file, set override-host parameter for each microservice except for Resource Agents. The parameter stores the VIP value that is used to access master and hot-standby nodes. The installation file also requires a block for keepalived to specify the hostname of the target master and hot-standby nodes so that the dependent packages can be installed on those nodes.

Note: See *Creating an Installation File* for more information about setting installation file parameters.

This step can be performed before Openet Weaver installation, and therefore can be completed when you first run the installer. If you choose to configure the parameters after installation you must run the installer again with the updated install file and the --secure-stage-traffic option as in the following example.

```
./ovlm-deploy.sh -i /path/to/ovlm-install-sample.yml -u ovlminstall
--secure-stage-traffic
```

The installer copies the keepalived configuration scripts to `/usr/lib64/ovlm/utilities/keepalived/` in the subdirectories shown below on master and hot-standby nodes.

```
> cd /usr/lib64/ovlm/utilities/keepalived/
> tree
.
├── samples
│   └── configuration
│       ├── nopreempt
│       │   ├── keepalived.conf.BACKUP
│       │   └── keepalived.conf.MASTER
│       └── preempt
│           ├── keepalived.conf.BACKUP
│           └── keepalived.conf.MASTER
└── scripts
    ├── generate-vnfm-monitor.sh
    ├── keepalived-setup.sh
    └── python
        ├── setup
        │   ├── keepalived-conf.py
        │   └── templates
        │       └── keepalived.conf.jinja
        └── vnfm-monitor
            ├── templates
            │   ├── monitor.jinja
            │   └── notify.jinja
            └── vnfm-monitor.py
└── systemd
    └── keepalived.service
11 directories, 12 files
```



Warning: When using the override-host parameter, the ovlm-deploy command cannot include the --create-sample-vnf option as keepalived is not yet running so there is no virtual IP assigned to master or hot-standby node. After keepalived is up and running you can execute the install script with the --create-sample-vnf option.

Configuring keepalived

To configure keepalived for HTTP or HTTPS after deployment, follow these steps.

1. On the master node, change directory to `/usr/lib64/ovlm/utilities/keepalived/scripts`.
2. On the master node generate a VNF-M monitoring and notification script using the command `generate-vnfm-monitor.sh -i installation_file`

The command parameters are described in the following table.

Table 19: vnfm-monitor.sh Parameters

Parameter	Mandatory	Description	Value
-i	Yes	The monitoring script is generated based on the microservices configured in installation properties file. There is a block for each microservice generated dynamically.	path of ovlm-install.yml properties file

The generated check_vnfm_status.sh script performs the following tasks:

- It checks the status of each microservice
- If the service is down then the script returns with exit code 1 to indicate a fault state on keepalived level.
- if the service is up the script returns with exit code 0 indicating that all microservices are up and running.

The generated notify.sh script restarts Instance Inventory Manager when it detects a master state, so that the latest information is read into cache.

The following is an example of the command to run the script.

```
>./generate-vnfm-monitor.sh -i /home/workspace/code/OVLM_trunk/acceptance/ \
configuration/userprops/ovlm-install-sample.yml
```

3. On each hot-standby node, change directory to /usr/lib64/ovlm/utilities/keepalived/scripts.
4. On each hot-standby node generate a VNF-M monitoring and notification script using the command generate-vnfm-monitor.sh -i *installation_file*

The following is an example of the command to run the script on the hot-standby node.

```
>./generate-vnfm-monitor.sh -i /home/workspace/code/OVLM_trunk/ \
acceptance/configuration/userprops/ovlm-install-sample.yml
```

5. In the /usr/lib64/ovlm/utilities/keepalived/scripts/ directory on the master node, set up keepalived by running the script keepalived-setup.sh -s *state* -i *interface* -o *own-ip* -p *peer-ip* -v *virtual-ip*[-e *preemptt*] [-r *router-id*]

The keepalived setup script perform following tasks:

- Generates keepalived.conf for master and hot-standby nodes based on input
- Copies the VNF-M monitoring script and notify script generated previously to the target location, which is set to /usr/lib64/ovlm/utilities/keepalived/bin/
- Copies generated keepalived.conf to the default keepalived conf location, which is /etc/keepalived
- Copies keepalived.service to the default service location, whic is /usr/lib/systemd/system/
- Reloads systemd-deamon to reflect the update in keepalived.service, which overrides the default keepalived.service with support of restart set as "always".
- Restarts keepalived.service

The command parameters are described in the following table.

Table 20: keepalived-setup.sh Parameters

Parameter	Mandatory	Description	Value
-s	Yes	Whether the node is master or hot-standby.	Possible values: <ul style="list-style-type: none"> • MASTER • BACKUP

Parameter	Mandatory	Description	Value
-i	Yes	The name of the network interface that keepalived monitors.	valid interface name
-o	Yes	The IP of the node currently acting as master or hot-standby.	valid IPv4 IP
-p	Yes	The IP of the peernode currently acting as master or hot-standby.	valid IPv4 IP
-v	Yes	The VIP for supporting failover in the master to hot-standby node.	valid IPv4 IP
-e	No	Whether keepalived is set to be preempt or non-preempt. The default is preempt (true). This parameter has no significance for the hot-standby node because BACKUP conf must always be set in preempt mode.	Possible values: <ul style="list-style-type: none">• true• false
-r	No	The router ID for master and hot-standby nodes. The default is 1.	Integer from 1 to 255

The following is an example of the command to run the script.

```
>./keepalived-setup.sh -s MASTER -i enp0s3 -o 10.3.18.37 -p 10.3.18.61 \
-v 10.3.10.90 -e false
```

- In the `/usr/lib64/ovlm/utilities/keepalived/scripts/` directory on the hot-standby node, set up keepalived by running the script `keepalived-setup.sh -s state -i interface -o own-ip -p peer-ip -v virtual-ip[-e preempt] [-r router-id]`

The following is an example of the command to run the script on the hot-standby node.

```
./keepalived-setup.sh -s BACKUP -i enp0s3 -o 10.3.18.61 \
-p 10.3.18.37 -v 10.3.10.90
```

- Verify the keepalive status on the master and hot-standby nodes by running the following command on each node:

```
sudo systemctl status keepalived
```

The following is an example of running the command on the master node and the resulting output.

```
[ovlm@10.0.0.27]$ sudo systemctl status keepalived
● keepalived.service - LVS and VRRP High Availability Monitor
  Loaded: loaded (/usr/lib/systemd/system/keepalived.service; enabled; vendor
  preset: disabled)
    Active: active (running) since Mon 2016-11-07 09:56:16 GMT; 12s ago
      Process: 18164 ExecStart=/usr/sbin/keepalived $KEEPALIVED_OPTIONS
                 (code=exited, status=0/SUCCESS)
    Main PID: 18165 (keepalived)
       CGroup: /system.slice/keepalived.service
               └─18165 /usr/sbin/keepalived -D
                  ├─18166 /usr/sbin/keepalived -D
                  ├─18167 /usr/sbin/keepalived -D
Nov 07 09:56:17 kl-000914-vm01 Keepalived_vrrp[18167]: VRRP_Instance(VI_1)
Transition to MASTER STATE
Nov 07 09:56:18 kl-000914-vm01 Keepalived_vrrp[18167]: VRRP_Instance(VI_1)
Entering MASTER STATE
Nov 07 09:56:18 kl-000914-vm01 Keepalived_vrrp[18167]: VRRP_Instance(VI_1)
setting protocol VIPs.
Nov 07 09:56:18 kl-000914-vm01 Keepalived_vrrp[18167]: VRRP_Instance(VI_1)
```

```

Sending gratuitous ARPs on enp0s3 for 10.3.10.90
Nov 07 09:56:18 kl-000914-vm01 Keepalived_healthcheckers[18166]: Netlink
reflector reports IP 10.3.10.90 added
Nov 07 09:56:21 kl-000914-vm01 Keepalived_vrrp[18167]: VRRP_Instance(VI_1)
Received lower prio advert, forcing new election
Nov 07 09:56:21 kl-000914-vm01 Keepalived_vrrp[18167]: VRRP_Instance(VI_1)
Sending gratuitous ARPs on enp0s3 for 10.3.10.90
Nov 07 09:56:21 kl-000914-vm01 Keepalived_vrrp[18167]: VRRP_Instance(VI_1)
Received lower prio advert, forcing new election
Nov 07 09:56:21 kl-000914-vm01 Keepalived_vrrp[18167]: VRRP_Instance(VI_1)
Sending gratuitous ARPs on enp0s3 for 10.3.10.90
Nov 07 09:56:23 kl-000914-vm01 Keepalived_vrrp[18167]: VRRP_Instance(VI_1)
Sending gratuitous ARPs on enp0s3 for 10.3.10.90
[ovlminstall@kl-000914-vm01 scripts]$

```

The following is an example of running the command on the master node and the resulting output.

```

[ovlm@10.0.0.27]$ sudo systemctl status keepalived
● keepalived.service - LVS and VRRP High Availability Monitor
  Loaded: loaded (/usr/lib/systemd/system/keepalived.service; enabled; vendor
    preset: disabled)
    Active: active (running) since Mon 2016-11-07 09:56:20 GMT; 15s ago
      Process: 12194 ExecStart=/usr/sbin/keepalived $KEEPALIVED_OPTIONS
                 (code=exited, status=0/SUCCESS)
        Main PID: 12195 (keepalived)
          CGroup: /system.slice/keepalived.service
                  └─12195 /usr/sbin/keepalived -D
                      ├─12196 /usr/sbin/keepalived -D
                      ├─12197 /usr/sbin/keepalived -D
                      ├─12198 /usr/sbin/keepalived -D
                      ├─12199 /usr/sbin/keepalived -D
                      ├─12200 /usr/sbin/keepalived -D
                      ├─12201 /usr/sbin/keepalived -D
                      ├─12202 /usr/sbin/keepalived -D
                      ├─12203 /usr/sbin/keepalived -D
                      ├─12204 /usr/sbin/keepalived -D
                      ├─12205 /usr/sbin/keepalived -D
                      ├─12206 /usr/sbin/keepalived -D
                      ├─12207 /usr/sbin/keepalived -D
                      ├─12208 /usr/sbin/keepalived -D
                      ├─12209 /usr/sbin/keepalived -D
                      ├─12210 /usr/sbin/keepalived -D
                      ├─12211 /usr/sbin/keepalived -D
                      ├─12212 /usr/sbin/keepalived -D
                      ├─12213 /usr/sbin/keepalived -D
                      ├─12214 /usr/sbin/keepalived -D
                      ├─12215 /usr/sbin/keepalived -D
                      ├─12216 /usr/sbin/keepalived -D
                      ├─12217 /usr/sbin/keepalived -D
                      ├─12218 /usr/sbin/keepalived -D
                      ├─12219 /usr/sbin/keepalived -D
                      ├─12220 /usr/sbin/keepalived -D
                      ├─12221 /usr/sbin/keepalived -D
                      ├─12222 /usr/sbin/keepalived -D
                      ├─12223 /usr/sbin/keepalived -D
                      ├─12224 /usr/sbin/keepalived -D
                      ├─12225 /usr/sbin/keepalived -D
                      ├─12226 /usr/sbin/keepalived -D
                      ├─12227 /usr/sbin/keepalived -D
                      ├─12228 /usr/sbin/keepalived -D
                      ├─12229 /usr/sbin/keepalived -D
                      ├─12230 /usr/sbin/keepalived -D
                      ├─12231 /usr/sbin/keepalived -D
                      ├─12232 /usr/sbin/keepalived -D
                      ├─12233 /usr/sbin/keepalived -D
                      ├─12234 /usr/sbin/keepalived -D
                      ├─12235 /usr/sbin/keepalived -D
                      ├─12236 /usr/sbin/keepalived -D
                      ├─12237 /usr/sbin/keepalived -D
                      ├─12238 /usr/sbin/keepalived -D
                      ├─12239 /usr/sbin/keepalived -D
                      ├─12240 /usr/sbin/keepalived -D
                      ├─12241 /usr/sbin/keepalived -D
                      ├─12242 /usr/sbin/keepalived -D
                      ├─12243 /usr/sbin/keepalived -D
                      ├─12244 /usr/sbin/keepalived -D
                      ├─12245 /usr/sbin/keepalived -D
                      ├─12246 /usr/sbin/keepalived -D
                      ├─12247 /usr/sbin/keepalived -D
                      ├─12248 /usr/sbin/keepalived -D
                      ├─12249 /usr/sbin/keepalived -D
                      ├─12250 /usr/sbin/keepalived -D
                      ├─12251 /usr/sbin/keepalived -D
                      ├─12252 /usr/sbin/keepalived -D
                      ├─12253 /usr/sbin/keepalived -D
                      ├─12254 /usr/sbin/keepalived -D
                      ├─12255 /usr/sbin/keepalived -D
                      ├─12256 /usr/sbin/keepalived -D
                      ├─12257 /usr/sbin/keepalived -D
                      ├─12258 /usr/sbin/keepalived -D
                      ├─12259 /usr/sbin/keepalived -D
                      ├─12260 /usr/sbin/keepalived -D
                      ├─12261 /usr/sbin/keepalived -D
                      ├─12262 /usr/sbin/keepalived -D
                      ├─12263 /usr/sbin/keepalived -D
                      ├─12264 /usr/sbin/keepalived -D
                      ├─12265 /usr/sbin/keepalived -D
                      ├─12266 /usr/sbin/keepalived -D
                      ├─12267 /usr/sbin/keepalived -D
                      ├─12268 /usr/sbin/keepalived -D
                      ├─12269 /usr/sbin/keepalived -D
                      ├─12270 /usr/sbin/keepalived -D
                      ├─12271 /usr/sbin/keepalived -D
                      ├─12272 /usr/sbin/keepalived -D
                      ├─12273 /usr/sbin/keepalived -D
                      ├─12274 /usr/sbin/keepalived -D
                      ├─12275 /usr/sbin/keepalived -D
                      ├─12276 /usr/sbin/keepalived -D
                      ├─12277 /usr/sbin/keepalived -D
                      ├─12278 /usr/sbin/keepalived -D
                      ├─12279 /usr/sbin/keepalived -D
                      ├─12280 /usr/sbin/keepalived -D
                      ├─12281 /usr/sbin/keepalived -D
                      ├─12282 /usr/sbin/keepalived -D
                      ├─12283 /usr/sbin/keepalived -D
                      ├─12284 /usr/sbin/keepalived -D
                      ├─12285 /usr/sbin/keepalived -D
                      ├─12286 /usr/sbin/keepalived -D
                      ├─12287 /usr/sbin/keepalived -D
                      ├─12288 /usr/sbin/keepalived -D
                      ├─12289 /usr/sbin/keepalived -D
                      ├─12290 /usr/sbin/keepalived -D
                      ├─12291 /usr/sbin/keepalived -D
                      ├─12292 /usr/sbin/keepalived -D
                      ├─12293 /usr/sbin/keepalived -D
                      ├─12294 /usr/sbin/keepalived -D
                      ├─12295 /usr/sbin/keepalived -D
                      ├─12296 /usr/sbin/keepalived -D
                      ├─12297 /usr/sbin/keepalived -D
                      ├─12298 /usr/sbin/keepalived -D
                      ├─12299 /usr/sbin/keepalived -D
                      ├─12300 /usr/sbin/keepalived -D
                      ├─12301 /usr/sbin/keepalived -D
                      ├─12302 /usr/sbin/keepalived -D
                      ├─12303 /usr/sbin/keepalived -D
                      ├─12304 /usr/sbin/keepalived -D
                      ├─12305 /usr/sbin/keepalived -D
                      ├─12306 /usr/sbin/keepalived -D
                      ├─12307 /usr/sbin/keepalived -D
                      ├─12308 /usr/sbin/keepalived -D
                      ├─12309 /usr/sbin/keepalived -D
                      ├─12310 /usr/sbin/keepalived -D
                      ├─12311 /usr/sbin/keepalived -D
                      ├─12312 /usr/sbin/keepalived -D
                      ├─12313 /usr/sbin/keepalived -D
                      ├─12314 /usr/sbin/keepalived -D
                      ├─12315 /usr/sbin/keepalived -D
                      ├─12316 /usr/sbin/keepalived -D
                      ├─12317 /usr/sbin/keepalived -D
                      ├─12318 /usr/sbin/keepalived -D
                      ├─12319 /usr/sbin/keepalived -D
                      ├─12320 /usr/sbin/keepalived -D
                      └─12321 /usr/sbin/keepalived -D

Nov 07 09:56:20 kl-000914-vm02 Keepalived_vrrp[12197]: Registering Kernel
netlink command channel
Nov 07 09:56:20 kl-000914-vm02 Keepalived_vrrp[12197]: Registering gratuitous
ARP shared channel
Nov 07 09:56:20 kl-000914-vm02 Keepalived_vrrp[12197]: Opening file
'/etc/keepalived/keepalived.conf'.
Nov 07 09:56:20 kl-000914-vm02 Keepalived_vrrp[12197]: Configuration is using
: 64531 Bytes
Nov 07 09:56:20 kl-000914-vm02 Keepalived_vrrp[12197]: Using LinkWatch kernel
netlink reflector...
Nov 07 09:56:20 kl-000914-vm02 Keepalived_vrrp[12197]: VRRP sockpool:
[ifindex(2), proto(112), unicast(1), fd(10,11)]
Nov 07 09:56:20 kl-000914-vm02 Keepalived_vrrp[12197]: VRRP_Script(check_vnmf_
_status) succeeded
Nov 07 09:56:21 kl-000914-vm02 Keepalived_vrrp[12197]: VRRP_Instance(VI_1)
Transition to MASTER STATE
Nov 07 09:56:21 kl-000914-vm02 Keepalived_vrrp[12197]: VRRP_Instance(VI_1)
Received higher prio advert
Nov 07 09:56:21 kl-000914-vm02 Keepalived_vrrp[12197]: VRRP_Instance(VI_1)
Entering BACKUP STATE

```

Managing keepalived Using SNMP

This topic describes commands that can be used to manage a keepalived master or backup node using SNMP:

- Check the complete configuration of a master or backup node
- Check the current priority value of a keepalived master or backup node
- Trigger failover manually

Note: The commands in this topic can be run locally or remotely.

You can check the configuration of a master or backup node using the command `snmpwalk -v2c -cpublic Master or Backup Host IP KEEPALIVED-MIB::vrrpInstanceTable`

The following is an example of the command in use:

```
snmpwalk -v2c -cpublic 10.3.10.90 KEEPALIVED-MIB::vrrpInstanceTable
```

The following is an example of the output of running the command:

```
KEEPALIVED-MIB::vrrpInstanceName.1 = STRING: VI_1
KEEPALIVED-MIB::vrrpInstanceVirtualRouterId.1 = Gauge32: 1
KEEPALIVED-MIB::vrrpInstanceState.1 = INTEGER: master(2)
KEEPALIVED-MIB::vrrpInstanceInitialState.1 = INTEGER: master(2)
KEEPALIVED-MIB::vrrpInstanceWantedState.1 = INTEGER: master(2)
KEEPALIVED-MIB::vrrpInstanceBasePriority.1 = INTEGER: 102
KEEPALIVED-MIB::vrrpInstanceEffectivePriority.1 = INTEGER: 107
KEEPALIVED-MIB::vrrpInstanceVipsStatus.1 = INTEGER: allSet(1)
KEEPALIVED-MIB::vrrpInstancePrimaryInterface.1 = STRING: enp0s3
KEEPALIVED-MIB::vrrpInstanceTrackPrimaryIf.1 = INTEGER: notTracked(2)
KEEPALIVED-MIB::vrrpInstanceAdvertisementsInt.1 = Gauge32: 1 seconds
KEEPALIVED-MIB::vrrpInstancePreempt.1 = INTEGER: noPreempt(2)
KEEPALIVED-MIB::vrrpInstancePreemptDelay.1 = Gauge32: 0 seconds
KEEPALIVED-MIB::vrrpInstanceAuthType.1 = INTEGER: password(1)
KEEPALIVED-MIB::vrrpInstanceLvsSyncDaemon.1 = INTEGER: disabled(2)
KEEPALIVED-MIB::vrrpInstanceGarpDelay.1 = Gauge32: 0 seconds
KEEPALIVED-MIB::vrrpInstanceSsmtpAlert.1 = INTEGER: disabled(2)
KEEPALIVED-MIB::vrrpInstanceNotifyExec.1 = INTEGER: disabled(2)
```

You can check the current priority of a keepalived master or backup node using the command `snmpwalk -v2c -cpublic Master or Backup Host IP KEEPALIVED-MIB::vrrpInstanceBasePriority.1`

The following is an example of the command in use:

```
snmpwalk -v2c -cpublic 10.3.10.90 KEEPALIVED-MIB::vrrpInstanceBasePriority.1
```

The following is an example of the output of running the command:

```
KEEPALIVED-MIB::vrrpInstanceBasePriority.1 = INTEGER: 102
```

You can trigger failover manually by updating the priority using the command `snmpset -v2c -cpublic Master or Backup Host IP KEEPALIVED-MIB::vrrpInstanceBasePriority.1 = 80`

After configuring VIM (OpenStack) for Openet Weaver you are ready to [Create an Installation File](#).

Creating an Installation File

The Openet Weaver deployment is defined in an install properties file. The file specifies where to install the Openet Weaver microservices and on what nodes to install them.

A typical stand-alone installation has one Openet Weaver Management node and one or more runtime nodes. A sample installation file, ovlm-install-sample.yml, is supplied in the downloaded Openet Weaver tarball. You must edit the file to specify the hosts for each of the microservices.

There is a configuration block for protocol settings, log rotation, installation directories, service owners, and for each microservice in the Openet Weaver deployment. The blocks are described in the following table.

Table 21: Microservices to Install

Block Heading (Service Name)	Mand.	Description
protocol	yes	<p>This block is used for configuring protocol settings globally for all microservices. You can configure Openet Weaver to use HTTP or HTTPS.</p> <p>When using HTTPS, there are other protocol parameters that are mandatory in the Protocol section (see #unique_8/unique_8_Connect_42_table_11AB854E60214036AC6A2ABF563FBF37). You are not required to configure server settings and client settings because each microservice can be either. When they initiate https communication, they are the server. When they respond to communication initiated by another microservice, they are the client.</p> <p>See Configuring Openet Weaver for HTTPS before configuring the protocol settings.</p>
log_rotation	No	This block is used for configuring log rotation properties.
installer	No	This block is used for configuring Openet Weaver to use a customer's existing SSH implementation. When it is not present, Openet Weaver defaults to the internal SSH support.
installation_directories	No	<p>This block is used for configuring installation the location of the following Openet Weaver directories:</p> <ul style="list-style-type: none"> • bin • cfg • run <p>Note: Installation directories can also be configured for individual microservices within microservice blocks.</p>
service_owner	No	This block is used for configuring service owner properties that allow Openet Weaver to be used with a Unix user and group other than the default ovlm and ovlmrsync.
keepalived	Yes for HA	This block is used for configuring keepalived properties related to deploying Openet Weaver with local high availability.
snmp	Yes	This block is used to configure the SNMP properties.
deployment_manager	Yes	This block is used for configuring Deployment Manager microservice properties.
frontend	Yes	This block is used for configuring Front End microservice properties.
workflow_engine	Yes	This block is used for configuring Workflow Engine microservice properties.
configuration_manager	Yes	This block is used for configuring Configuration Manager microservice properties.
resource_manager	Yes	This block is used for configuring Resource Manager microservice properties.

Block Heading (Service Name)	Mand.	Description
resource_agent	Yes	<p>This block is used for configuring Resource Agent microservice properties. Although this is a mandatory microservice, Resource Agents do not need to be installed when installing Openet Weaver if you are using Virtualized Infrastructure Manager (VIM). That is because Resource Agents can be embedded in the OpenStack glance image.</p> <p>Resource Agents must be installed if you intend to deploy VNFs that are not configured for VIM.</p> <p>Note: Openet Weaver supports Resource Agents running specific versions of CentOS, Redhat or Ubuntu operating systems. During installation, the installation script detects which OS is being run and installs the appropriate software package on the Resource Agent.</p>
vnmf_gui	Yes	This block is used for configuring VNF-M GUI microservice properties.
performance_manager	No	This block is used for configuring Performance Manager microservice properties.
fault_manager	No	This block is used for configuring Fault Manager microservice properties.
vim_abstraction_layer	No	This block is used for configuring VIM Abstraction Layer (VAL) microservice properties.
instance_inventory_manager	Yes	This block is used for configuring Instance Inventory Manager microservice properties, mainly related to the housekeeping of persistent events.
auth_server	No	<p>This block is used for configuring Authorization Server microservice properties.</p> <p>Note: If Authorization Server configuration is not provided, then no security is set up.</p>
reference_vnf	No	This block is used for configuring the Reference VNF node, which is used as a reference installation for deploying other VNF applications.

Each microservice configuration block contains two sub blocks:

- Host block
- Properties block

Host Block

At least one host must be specified for each microservice you intent to deploy. All microservices can be installed to a single host with the exception of the Resource Agents, which must be installed to separate hosts. However, for production environments, it is recommended that you install each microservice to a separate host.

The microservices and components are installed to the hosts configured in the install properties file when you run the Openet Weaver installer. In the event that you are installing a microservice to multiple hosts, you have the option of specifying properties for those microservices in the host block, so that each host can be configured differently.

Properties Block

The properties block for each microservice contains application-specific property configuration that must be configured on the target system. The ovml-install-sample.yml file provided with Openet Weaver has preconfigured default ports for each microservice. Other properties are defined only as required. For most Openet Weaver microservices these are Spring Boot Application YML properties. The exception is the Workflow Engine, which has an http_port property instead of a server.port.

See the Install Properties Prioritization topic linked at the bottom of this page for more information about the priority order of properties defined in the installation file.

To create an installation file for the sample lab deployment, follow these steps:

1. Open ovlm-install-sample.yml in a text editor.
2. For each microservice or component, define host and port settings specific to the environment. The configurable installation file properties are described in the pages linked below along with their possible values and an example configuration block for each section. Specify properties under the properties heading, except where noted. For the examples provided node-1 is used as the host for all microservices and components except the Resource Agents, which run on the runtime nodes.
3. Save your changes as a new file.

Sample installation File

You can see a sample configuration file for Openet Weaver at the following location:

```
ovlm-packaging-1.5.x-SNAPSHOT-core-install.tar ./ovlm-install/ovlm-in
stall-https-sample.yml
```

When the installation file is configured you are ready to [run the Openet Weaver installer](#).

Protocol Settings

The protocol block is where you configure protocol settings globally for all microservices. You can configure Openet Weaver to use HTTP or HTTPS.

When using HTTPS, there are other protocol parameters that are mandatory in the Protocol section (see [#unique_78/unique_78_Connect_42_table_11AB854E60214036AC6A2ABF563FBF37](#)). You are not required to configure server settings and client settings because each microservice can be either. When they initiate https communication, they are the server. When they respond to communication initiated by another microservice, they are the client.

See [Configuring Openet Weaver for HTTPS](#) before configuring the protocol settings.

The following is an example of a protocol block:

```
# Example Protocol Configuration
protocol: https
https_configuration:
  server:
    keystore_path: demo_key.jks
    keystore_password: password
    key_password: password
  client:
    verify_server_certificate: No
    truststore_path: demo_trust.jks
```

Note: Create one section for each set of SSL certificates.

The following table lists configurable protocol properties you can define in the installation file.

Property	Mandatory	Description	Default Value
protocol	No	The protocol that is used during communication with the server. Possible values: <ul style="list-style-type: none"> • http • https 	http

Property	Mandatory	Description	Default Value
https_configuration: server: keystore_path	Yes (for https)	The path to the Java keystore JKS file where the certificate and private key are saved. These can be authorization certificates or public key certificates, plus corresponding private keys used in SSL encryption, for example. The path must be different from the client: truststore_path. Note: JKS files are generated automatically for the main certificate set during installation based on installer file configuration settings. JKS certificates for additional certificates must be generated manually.	demo_key.jks
https_configuration: server: keystore_password	No	The password for the keystore in plain text. Note: The password must be encrypted before deployment. For more information see Encrypting Passwords .	N/A
https_configuration: server: key_password	No	The password for the private key, if used. Note: The password must be encrypted before deployment.	N/A
https_configuration: client: verify_server_certificate	No	Informs the client whether server certificate validation is required. Possible values are: <ul style="list-style-type: none">• Yes• No	No

log_rotation Settings

The log_rotation block is where you configure log rotation properties.

The following is an example of a log_rotation block:

```
# Example log rotation configuration
log_rotation:
  interval: monthly
  max_file_size: 5000000
```

When configured correctly, log rotation results in multiple archived log files that are suffixed with the date on which they were archived followed by an integer ID representing the count of files rotated on that day. The following is an example.

```
-rw-r--r-- 1 ovlm ovlm    863 Jun  1 16:57 ovlm-ra.log
-rw-r--r-- 1 ovlm ovlm 30966 Jun  1 16:56 ovlm-ra.log.2017-06-01.0.log
```

When multiple size-based rotations are triggered within the defined interval than the suffix is incremented.

Note: Log rotation currently applies only to Resource Agents.

The following table lists configurable log_rotation properties you can define in the installation file.

Property	Mandatory	Description	Default Value
interval	Yes	The amount of time that elapses before logs are rotated. Possible values are: <ul style="list-style-type: none">• daily• weekly• monthly	daily
max_file_size	Yes	An integer representing the maximum file size in bytes.	5000000
https_configuration: client: truststore_path	No when verify_server set to false	The path to the Java truststore JKS file where trusted or CA certificates are saved. If this parameter is absent then the default java truststore is used: \$JAVA_HOME/jre/lib/security/cacerts. The path must be different from the client: truststore_path. Note: Java truststore files are generated automatically during installation based on installer file configuration settings.	truststore_path: demo_trust.jks

installer Settings

The installer block is where you configure Openet Weaver to use a customer's existing SSH implementation. When it is not present, Openet Weaver defaults to the internal SSH support.

The following is an example of a installer block:

```
# Example installer block
installer:
  ssh_bin: /usr/bin/ssh
  rsync_bin: /usr/bin/rsync
  ssh_options: ''
```

The following table lists configurable installer properties you can define in the installation file.

Property	Mandatory	Description	Default Value
ssh_bin	No	The path to the SSH bin directory.	
rsync_bin	No	The Path to the rsync bin directory.	
ssh_options	No	A set of options that are appended to the SSH command when it is invoked. Values can be any valid flag supported by SSH.	
sshcmd_wrapper_enabled	No	Whether the sshcmd_wrapper is enabled. Possible values are: <ul style="list-style-type: none">• True• False	

installation_directories Settings

The installation_directories block is where you configure the location of the following Openet Weaver directories:

- bin
- cfg
- run

The following is an example of a installation_directories block:

```
# Example installation_directories block
installation_directories:
  path_bin: /opt/ovlm/bin/
  path_cfg: /opt/ovlm/etc/
  path_run: /opt/ovlm/run/
```

Note: Include this section in the installation file only when you changing the directories above from their default locations. The default paths are used when the parameters are not specified.

Note: Use these parameters to make global changes to paths that affect all microservices. Paths can also be configured for specific microservices by including an installation_paths block within the microservice block, in which case that path overrides the global path. PATH_RUN and PATH_CFG cannot be reconfigured for Workflow Engine.

The following table lists configurable installation_directories properties you can define in the installation file.

Property	Mutiny	Description	Default Value
path_bin	No	The path to the PATH_BIN directory.	/usr/lib64/ovlm
path_cfg	No	The path to the PATH_CFG directory.	/etc/ovlm
path_run	No	The path to the PATH_RUN directory.	/var/run/ovlm

service_owner Settings

The service_owner block is where you configure service owner properties that allow Openet Weaver to be used with a Unix user and group other than the default ovlm and ovlmrsync .

The following is an example of a service_owner block:

```
# Example Service Owner Configuration
service_owner:
  user: user1
  user_group: group2
  ssh_user: ssh_user3
```

Note: The service_owner section is optional, but when it is used all parameters are mandatory. When it is not used the default values specified below are used.

The following table lists configurable service_owner properties you can define in the installation file.

Property	Mutiny	Description	Default Value
user	Yes	The Unix user that "owns" Openet Weaver. All process, directory and file ownership belongs to this configured Unix user.	ovlm
user_group	Yes	The Unix group to which the user configured in this block belongs.	ovlm
ssh_user	Yes	A Unix user that is created in the Configuration Manager node. This user is used when triggering Rsync in SSH mode.	ovlmrsync

keepalived Settings

The keepalived block is where you configure keepalived settings related to deploying Openet Weaver with local high availability.

The following is an example of a keepalived block:

```
# Example Keepalived Configuration
keepalived:
  hosts:
    - master-node
    - backup-node
```

The following table lists configurable keepalived properties you can define in the installation file.

Property	Mandatory	Description	Default Value
hosts:	Yes when deploying with master and backup node support, otherwise no	The host names used for keepalived. Both default values must be used, for example: Note: If you deploy VIM and the DNS is not configured on the VIM, the hosts <i>must</i> be specified by IP address instead of by hostname.	- master-node - backup-node

snmp Settings

The snmp block is where you configure snmp settings for Openet Weaver.

The following is an example of a snmp block:

```
# Example snmp block
snmp:
  manager: snmp-manager
  community: public
```

The following table lists configurable snmp properties you can define in the installation file.

Property	Mandatory	Description	Default Value
manager:	Yes	The URI connection string for the SNMP manager. This can be the hostname or the hostname and port combined in the following format: <host>:<ip> For example: snmpmgr:162.	
community:	Yes	The SNMP community string.	public

deployment_manager Settings

The deployment_manager block is where you configure the Deployment Manager microservice properties.

The following is an example of a deployment_manager block:

```
# Example Deployment Manager Configuration
deployment_manager:
  hosts:
    - node-1
  properties:
    server:
      port: 28130
    spring:
      datasource:
        username: dm
        password: dm
    service-installation:
      max-pool-size: 10
```

```

    timeout: 600
repository-root: /usr/lib64/ovlm/dm/repository-root
storage:
    minimum-allocation: 1048576

```

The following table lists configurable deployment_manager properties you can define in the installation file. You must define host and port settings specific to the environment. Specify properties under the properties heading, except where noted.

Property	Multi	Description	Default Value
hosts:	Yes	<p>The name of the microservice host machines.</p> <p>Note: If you deploy VIM and the DNS is not configured on the VIM, the hosts <i>must</i> be specified by IP address instead of by hostname.</p> <p>There must be an entry for the master host and the hot-standby (backup) host when keepalived is used, for example:</p> <pre># Configuration Manager Configuration configuration_manager: hosts: - <master_host> - <hot-standby_host></pre>	None
server: port:	Yes	<p>The port on which a microservice runs the REST API listener. Any port number greater than 1024. The port number must be less than the ephemeral port range defined on the host.</p> <p>The sample installation YML file provided with Openet Weaver uses a port setting of 28130 for deployment_manager.</p>	None
override_host:	No	For keepalived setup, this is the Virtual IP (VIP) that is used for accessing the master and hot-standby nodes. The VIP must be an address that doesn't correspond to an actual physical network interface but must be in a valid IP range for that Network.	None
path_bin	No	The path to the bin directory PATH_BIN directory.	/usr/lib64/ovlm
logging: level: com.openet. modules.ovlm	No	<p>The value can be one of the following:</p> <ul style="list-style-type: none"> • TRACE • DEBUG • INFO • WARN • ERROR 	WARN
path_cfg	No	The path to the PATH_CFG directory.	/etc/ovlm
path_run	No	The path to the PATH_RUN directory.	/var/run/ovlm
logging:file	No	The location of the logging files. When not specified the default location is used.	/var/log/ovlm/ <microservice>/
repository-root	No	The directory on the host where Deployment Manager stores all service installation-related package and configuration.	/usr/lib64/ovlm /dm/ repository-root

Property	Mandatory	Description	Default Value
spring: datasourceusername	Yes	The user name for the database.	dm
spring: datasourcepassword	Yes	The password for the database. Note: The password must be encrypted before deployment. For more information see Encrypting Passwords .	dm
service-installation: max-pool-size	No	The maximum thread pool size to be used for performing service installation.	10
service-installation: timeout	No	How long service installation waits for the operation to complete, in seconds.	600
storage: minimum-allocation	No	The minimum limit for allocating storage, in bytes. A value of 0 (zero) means there is no minimum limit.	1048576

frontend Settings

The frontend block is where you configure the Front End microservice properties.

The following is an example of a frontend block:

```
# Example Frontend Configuration
frontend:
  hosts:
    - node-1
  properties:
    server:
      port: 28050
```

The following table lists configurable frontend properties you can define in the installation file. You must define host and port settings specific to the environment. Specify properties under the properties heading, except where noted.

Property	Mandatory	Description	Default Value
hosts:	Yes	<p>The name of the microservice host machines.</p> <p>Note: If you deploy VIM and the DNS is not configured on the VIM, the hosts <i>must</i> be specified by IP address instead of by hostname.</p> <p>There must be an entry for the master host and the hot-standby (backup) host when keepalived is used, for example:</p> <pre># Configuration Manager Configuration configuration_manager: hosts: - <master_host> - <hot-standby_host></pre>	None
server: port:	Yes	<p>The port on which a microservice runs the REST API listener. Any port number greater than 1024. The port number must be less than the ephemeral port range defined on the host.</p> <p>The sample installation YML file provided with Openet Weaver uses a port setting of 28050 for frontend .</p>	None

Property	Mandatory	Description	Default Value
override_host:	No	For keepalived setup, this is the Virtual IP (VIP) that is used for accessing the master and hot-standby nodes. The VIP must be an address that doesn't correspond to an actual physical network interface but must be in a valid IP range for that Network.	None
path_bin	No	The path to the bin directory PATH_BIN directory.	/usr/lib64/ovlm
logging: level: com.openet. modules.ovlm	No	The value can be one of the following: <ul style="list-style-type: none">• TRACE• DEBUG• INFO• WARN• ERROR	WARN
path_cfg	No	The path to the PATH_CFG directory.	/etc/ovlm
path_run	No	The path to the PATH_RUN directory.	/var/run/ovlm
logging:file	No	The location of the logging files. When not specified the default location is used.	/var/log/ovlm/ <microservice>/
timeout	No	The optional timeout parameter to specify how long the synchronous call waits for the operation to complete. The default value is 60 seconds.	60

workflow_engine Settings

The workflow_enginer block is where you configure the Workflow Engine microservice properties.

The following is an example of a workflow_engine block:

```
workflow_engine:
  ...
  properties:
    logging:
      path: /opt/ovlm/log/wf/
  ...
```

The following table lists configurable workflow_engine properties you can define in the installation file. You must define host and port settings specific to the environment. Specify properties under the properties heading, except where noted.

Property	Mandatory	Description	Default Value
hosts:	Yes	<p>The name of the microservice host machines.</p> <p>Note: If you deploy VIM and the DNS is not configured on the VIM, the hosts <i>must</i> be specified by IP address instead of by hostname.</p> <p>There must be an entry for the master host and the hot-standby (backup) host when keepalived is used, for example:</p> <pre># Configuration Manager Configuration configuration_manager: hosts: - <master_host> - <hot-standby_host></pre>	None

Property	Mandatory	Description	Default Value
server: port:	Yes	The port on which a microservice runs the REST API listener. Any port number greater than 1024. The port number must be less than the ephemeral port range defined on the host. The sample installation YML file provided with Openet Weaver uses a port setting of 8089 for workflow_engine.	None
override_host:	No	For keepalived setup, this is the Virtual IP (VIP) that is used for accessing the master and hot-standby nodes. The VIP must be an address that doesn't correspond to an actual physical network interface but must be in a valid IP range for that Network.	None
path_bin	No	The path to the bin directory PATH_BIN directory.	/usr/lib64/ovlm
logging:path	No	The location of the Workflow Engine logging files. When not specified the default location is used. Note: The path you enter for this parameter must end in a trailing forward slash (/). Any string after the final forward slash is ignored by the installer and not added to the path.	

configuration_manager Settings

The configuration_manager block is where you configure the Configuration Manager microservice properties.

The following is an example of a configuration_manager block:

```
# Example Configuration Manager Configuration
configuration_manager:
  hosts:
    - node-1
  properties:
    server:
      port: 28010
      rsync_port: 28000
```

The following table lists configurable configuration_manager properties you can define in the installation file. You must define host and port settings specific to the environment. Specify properties under the properties heading, except where noted.

Property	Mandatory	Description	Default Value
hosts:	Yes	The name of the microservice host machines. Note: If you deploy VIM and the DNS is not configured on the VIM, the hosts <i>must</i> be specified by IP address instead of by hostname. There must be an entry for the master host and the hot-standby (backup) host when keepalived is used, for example: <pre># Configuration Manager Configuration configuration_manager: hosts: - <master_host> - <hot-standby_host></pre>	None

Property	Mandatory	Description	Default Value
server: port:	Yes	The port on which a microservice runs the REST API listener. Any port number greater than 1024. The port number must be less than the ephemeral port range defined on the host. The sample installation YML file provided with Openet Weaver uses a port setting of 28010 for configuration_manager.	None
override_host:	No	For keepalived setup, this is the Virtual IP (VIP) that is used for accessing the master and hot-standby nodes. The VIP must be an address that doesn't correspond to an actual physical network interface but must be in a valid IP range for that Network.	None
path_bin	No	The path to the bin directory PATH_BIN directory.	/usr/lib64/ovlm
logging: level: com.openet. modules.ovlm	No	The value can be one of the following: <ul style="list-style-type: none">• TRACE• DEBUG• INFO• WARN• ERROR	WARN
path_cfg	No	The path to the PATH_CFG directory.	/etc/ovlm
path_run	No	The path to the PATH_RUN directory.	/var/run/ovlm
logging:file	No	The location of the logging files. When not specified the default location is used.	/var/log/ovlm/ <microservice>/
rsync_damm_mod	No	Whether the Openet Weaver rsync server configuration is used or whether the customer provides a custom configuration for the rsync server is used in the deployment. Possible values are: <ul style="list-style-type: none">• true—Openet Weaver rsync server configuration• false—customer rsync server configuration	true
rsync_port	Yes	The port used to access the rsync server. Note: This port number should match the rsync server port number configured on the Openet Weaver management node and the configuration_manager: rsync_port set in the resource_agent block in the installation file.	28000
multipart: maxFileSize	No	The maximum size permitted for uploaded files.	4096MB
multipart: maxRequestSize	No	The maximum size allowed for multipart/form-data requests.	4096MB
process-timeout:	No	An integer value in microseconds.	10000
repository-root:	No	The directory on the host where Configuration Manager stores all VNF-related configuration.	/usr/lib64/ovlm/config ository-root
storage: minimum-allocation	No	The minimum limit for allocating storage, in bytes. A value of 0 (zero) means there is no minimum limit.	1048576

Property	Mandatory	Description	Default Value
template-validation-script-execution-timeout	Yes	<p>The maximum time in seconds allowed to execute the internal Jinja validation script per VNFC instance. The value must be a positive integer.</p> <p>For example, if the value is set to 60 seconds and the VNF instance to be validated has five VNFC instances, then each VNFC instance will have a maximum of 60 seconds to execute validation on all VNFC instances before the internal validation script is terminated.</p>	60

resource_manager Settings

The resource_manager block is where you configure the Resource Manager microservice properties.

The following is an example of a resource_manager block:

```
# Example Resource Manager Configuration
resource_manager:
  hosts:
    - node-1
  properties:
    server:
      port: 28020
```

The following table lists configurable resource_manager properties you can define in the installation file. You must define host and port settings specific to the environment. Specify properties under the properties heading, except where noted.

Property	Mandatory	Description	Default Value
hosts:	Yes	<p>The name of the microservice host machines.</p> <p>Note: If you deploy VIM and the DNS is not configured on the VIM, the hosts <i>must</i> be specified by IP address instead of by hostname.</p> <p>There must be an entry for the master host and the hot-standby (backup) host when keepalived is used, for example:</p> <pre># Configuration Manager Configuration configuration_manager: hosts: - <master_host> - <hot-standby_host></pre>	None
server: port:	Yes	<p>The port on which a microservice runs the REST API listener. Any port number greater than 1024. The port number must be less than the ephemeral port range defined on the host.</p> <p>The sample installation YML file provided with Openet Weaver uses a port setting of 28020 for resource_manager.</p>	None
override_host:	No	For keepalived setup, this is the Virtual IP (VIP) that is used for accessing the master and hot-standby nodes. The VIP must be an address that doesn't correspond to an actual physical network interface but must be in a valid IP range for that Network.	None
path_bin	No	The path to the bin directory PATH_BIN directory.	/usr/lib64/ovlm

Property	Mandatory	Description	Default Value
logging:level: com.openet.modules.ovlm	No	The value can be one of the following: <ul style="list-style-type: none"> • TRACE • DEBUG • INFO • WARN • ERROR 	WARN
path_cfg	No	The path to the PATH_CFG directory.	/etc/ovlm
path_run	No	The path to the PATH_RUN directory.	/var/run/ovlm
logging:file	No	The location of the logging files. When not specified the default location is used.	/var/log/ovlm/<microservice>/
agent-port:	No	Any port number greater than 1024. The port number must be less than the ephemeral port range defined on the host. The port number on Resource Manager must be the same as the server.port key value under Resource Agent.	28030
polling-retries-number:	No	Integer value that represents the maximum number of status requests Resource Manager sends to Resource Agent when querying action execution status. The value is used for calculating the time interval between query requests, for example: <pre>time_interval = \ <action_timeout_from_component_metadata> / \ <polling-retries-number></pre>	4
max-pool-size:	No	Integer value that represents the maximum number of actions that can be executed simultaneously.	100

resource_agent Settings

The resource_agent block is where you configure Resource Agent microservice properties.

It is mandatory to include this block in the installation file. However, Resource Agents do not need to be installed when installing Openet Weaver if you are using Virtualized Infrastructure Manager (VIM). That is because Resource Agents can be embedded in the OpenStack glance image.

Resource Agents must be installed if you intend to deploy VNFs that are not configured for VIM.

Note: Openet Weaver supports Resource Agents running specific versions of CentOS, Redhat or Ubuntu operating systems. During installation, the installation script detects which OS is being run and installs the appropriate software package on the Resource Agent.

The following is an example of a resource_agent block:

```
# Example Resource Agent Configuration
resource_agent:
  properties:
    server:
      port: 28030
    sudo_privileges:
      - /bin/postgresql-setup
  resource_agent:
    metadata:
```

```

stage:
  timeout: 100
parameters:
  rsync_timeout: 100

```

Note: See [Configuring Resource Agent Persistence for Logging and Repository Root](#) for configuration information about setting up centralized storage for log and repository root.

The following table lists configurable resource_agent properties you can define in the installation file. You must define host and port settings specific to the environment. Specify properties under the properties heading, except where noted.

Property	Mandatory	Description	Default Value
hosts:	Yes	<p>The name of the microservice host machines.</p> <p>Note: If you deploy VIM and the DNS is not configured on the VIM, the hosts <i>must</i> be specified by IP address instead of by hostname.</p> <p>There must be an entry for the master host and the hot-standby (backup) host when keepalived is used, for example:</p> <pre> # Configuration Manager Configuration configuration_manager: hosts: - <master_host> - <hot-standby_host> </pre> <p>Note: This property is not required for Resource Agent if VNFs are to be deployed with VIM configuration.</p>	None
server: port:	Yes	<p>The port on which a microservice runs the REST API listener. Any port number greater than 1024. The port number must be less than the ephemeral port range defined on the host.</p> <p>The sample installation YML file provided with Openet Weaver uses a port setting of 28030 for resource_agent.</p>	None
override_host:	No	<p>For keepalived setup, this is the Virtual IP (VIP) that is used for accessing the master and hot-standby nodes. The VIP must be an address that doesn't correspond to an actual physical network interface but must be in a valid IP range for that Network.</p>	None
path_bin	No	<p>The path to the bin directory PATH_BIN directory.</p> <p>Important: The following information applies to the Resource Agents only.</p> <p>Installing a Resource Agent creates a Python virtual environment under this path:</p> <pre><path_bin>/ra/python_virtualenv</pre> <p>If path_bin is not defined, the path_bin in the installation_directory block is used. If both are not defined, the default value is used.</p> <p>By default, the location of Resource Agent python virtual environment is /usr/lib64/ovlm/ra/python_virtualenv.</p>	/usr/lib64/ovlm

Property	Mandatory	Description	Default Value
logging:level: com.openet.modules.ovlm	No	The value can be one of the following: <ul style="list-style-type: none"> • TRACE • DEBUG • INFO • WARN • ERROR 	WARN
path_cfg	No	The path to the PATH_CFG directory.	/etc/ovlm
path_run	No	The path to the PATH_RUN directory.	/var/run/ovlm
logging:path	No	The location of the logging files. When not specified the default location is used.	/var/log/ovlm/ra
logging: level	No	The lowest logging level to log. All higher levels will also be logged. For example if the level is specified as DEBUG, then ERROR and WARNING messages will also be logged.	
append_hostname_to_path	No	<p>Whether the following have a child directory named after the hostname:</p> <ul style="list-style-type: none"> • application-log-path • repository-root-path • procedures-log-path <p>The purpose of the child directory is to prevent the log from being appended to a single file. This supports the centralized logging use case.</p> <p>Possible values for this parameter are</p> <ul style="list-style-type: none"> • true • false • When true the application-log-path and repository-root-path has an additional child directory named after the hostname. • if procedures-log-path is not using the default value, it also has a child directory named after hostname. • when false, the child directories are not created. 	false
resource_agent:max-pool-size	No	integer value that represents the max number of parallel actions to be executed.	/usr/lib64/ovlm/job s
resource_agent:job-storage	No	directory location on host where job execution statuses are stored.	/usr/lib64/ovlm/ra
resource_agent:procedures-home	No	directory location on host for scripts representing build in features like stage or metrics collectors.	/var/ovlm/pro cedures
resource_agent:procedures-log-path	No	The absolute path to the VNF procedure execution logs directory.	/var/ovlm/pro cedure or \$bgpnh/pro cedures
resource_agent:proceduretmpPath	No	directory location on host for creating temporary files required by RA .	/tmp
resource_agent:pm-enable:	No	Possible values are true and false. Performance Manager is enabled when the value is true.	true

Property	Mandatory	Description	Default Value
resource_agent: pm-send-period:	No	An integer that specifies the how often Resource Agent sends performance metrics, in seconds.	10
resource_agent: fm-enable:	No	A boolean that specifies whether Fault Manager is enabled. Possible values are true and false. Fault manager is enabled when the value is true.	true
resource_agent: health-check-period	No	An integer that specifies the frequency of vnfc health checking.	
resource_agent: repository-root	No	The absolute path to the Resource Agent repository root directory.	/opt/ovl640/rw/agent/repository
resource_agent: metadata: stage: procedure_name:	No	The procedure name for staging.	stage_v2.py
resource_agent: metadata: stage: timeout:	No	An integer representing the staging timeout value in seconds.	60
resource_agent: metadata: stage: parameters: user:	No	Required user permission to run rsync. root is the recommended value.	None
resource_agent: metadata: stage: parameters: group:	No	Required group permission to run rsync. root is the recommended value.	None
resource_agent: metadata: stage: parameters: rsync: timeout	No	An integer representing the rsync timeout value in seconds.	60
resource_agent: metadata: stage: parameters: rsync: bin	No	The path where the rsync command is invoked.	rsync
resource_agent: metadata: stage: parameters: rsync: share_name	No	The name of the rsync module. This setting is applicable only if the rsync_daemon_mode property is set to true in the configuration_manager block in the installation file. When it is set to false a customer-configured rsync is used. In such a case the Configuration Manager repository-root is used during staging.	":ovlm"

Property	Mandatory	Description	Default Value
resource_agent: metadata: stage: parameters: rsync: secret_user	No	The name of the rsync user.	ovlm
resource_agent: metadata: stage: parameters: rsync: ssh_user	No	Used for setting ssh user tunneling. This setting is applicable only if the '--secure-stage-traffic' parameter is used when you run the ovlm-deploy script at installation time, and the 'rsync_daemon_mode' property is set to true in the configuration_manager block in the installation file.	ovlmrsync
resource_agent: metadata: stage: parameters: rsync: ssh_port	No	The port on which the Resource Agent listens for SSH traffic.	28000
resource_agent: metadata: stage: parameters: rsync: secret_file	No	The location of the rsyncd.secrets file. This setting is applicable only if the 'rsync_daemon_mode' property is set to true in the configuration_manager block in the installation file. When it is set to false a customer-configured rsync is used. In such you must set up an SSH key-exchange between the Openet Weaver management node where Configuration Manager is hosted and all Resource Agent nodes. You must also start the SSH key-exchange at the Openet Weaver management node where Configuration Manager is hosted.	/tmp/rsync.secret
configuration_manager: rsync_host:	Yes	The name of the machine hosting rsync.	None
configuration_manager: rsync_port:	Yes	The port used to access the rsync server. Note: This port number should match the rsync server port number configured on the Openet Weaver management node and the 'rsync_port' set in the configuration_manager block in the installation file.	28000
configuration_manager: url:	Yes	The Configuration Manager REST API endpoint in the form http://<host>:<port>.	port: 28090
performance_manager: url:	No	The Performance Manager REST API endpoint for reporting performance metrics in the form http://<host>:28090.	port: 28090

Property	Mandatory	Description	Default Value
sudo_privileges	No	<p>Used to define sudo privileges in addition to the mandatory ones. When you use this parameter you must also create a sudo user who has these specific privileges assigned.</p> <p>This parameter takes a list of one or more command names, directories, and other aliases. A command name is a fully qualified filename that can include shell-style wildcards. The following is an example of sudo_privileges configuration:</p> <pre>sudo_privileges: - /bin/bash - /bin/passwd - /bin/kill - /bin/ls [:alpha:]*</pre> <p>Note: The installer only supports overwrite option for user-defined sudo privileges.</p>	

vnmf_gui Settings

The vnmf_gui block is where you configure the VNF-M GUI microservice properties.

The following is an example of a vnmf_gui block:

```
# Example VNFM GUI Configuration
vnmf_gui:
  hosts:
    - node-1
  properties:
    server:
      port: 28200
```

The following table lists configurable vnmf_gui properties you can define in the installation file. You must define host and port settings specific to the environment. Specify properties under the properties heading, except where noted.

Property	Mandatory	Description	Default Value
hosts:	Yes	<p>The name of the microservice host machines.</p> <p>Note: If you deploy VIM and the DNS is not configured on the VIM, the hosts <i>must</i> be specified by IP address instead of by hostname.</p> <p>There must be an entry for the master host and the hot-standby (backup) host when keepalived is used, for example:</p> <pre># Configuration Manager Configuration configuration_manager: hosts: - <master_host> - <hot-standby_host></pre>	None
server: port:	Yes	<p>The port on which a microservice runs the REST API listener. Any port number greater than 1024. The port number must be less than the ephemeral port range defined on the host.</p> <p>The sample installation YML file provided with Openet Weaver uses a port setting of 28200 for vnmf_gui.</p>	None

Property	Mandatory	Description	Default Value
override_host:	No	For keepalived setup, this is the Virtual IP (VIP) that is used for accessing the master and hot-standby nodes. The VIP must be an address that doesn't correspond to an actual physical network interface but must be in a valid IP range for that Network.	None
path_bin	No	The path to the bin directory PATH_BIN directory.	/usr/lib64/ovlm
logging:level: com.openet.modules.ovlm	No	The value can be one of the following: <ul style="list-style-type: none">• TRACE• DEBUG• INFO• WARN• ERROR	WARN
path_cfg	No	The path to the PATH_CFG directory.	/etc/ovlm
path_run	No	The path to the PATH_RUN directory.	/var/run/ovlm
logging:file	No	The location of the logging files. When not specified the default location is used.	/var/log/ovlm/<microservice>/
idle_timeout_mins:	No	The amount of time before a user is logged out of the VNF-M GUI after their last interaction with the interface.	30

performance_manager Settings

The performance_manager block is where you configure the Performance Manager microservice properties.

The following is an example of a performance_manager block:

```
# Example Performance Manager Configuration
performance_manager:
  hosts:
    - node-1
  properties:
    server:
      port: 28090
```

The following table lists configurable performance_manager properties you can define in the installation file. You must define host and port settings specific to the environment. Specify properties under the properties heading, except where noted.

Property	Mandatory	Description	Default Value
hosts:	Yes	<p>The name of the microservice host machines.</p> <p>Note: If you deploy VIM and the DNS is not configured on the VIM, the hosts <i>must</i> be specified by IP address instead of by hostname.</p> <p>There must be an entry for the master host and the hot-standby (backup) host when keepalived is used, for example:</p> <pre># Configuration Manager Configuration configuration_manager: hosts: - <master_host> - <hot-standby_host></pre>	None
server: port:	Yes	<p>The port on which a microservice runs the REST API listener. Any port number greater than 1024. The port number must be less than the ephemeral port range defined on the host.</p> <p>The sample installation YML file provided with Openet Weaver uses a port setting of 28090 for performance_manager.</p>	None
override_host:	No	For keepalived setup, this is the Virtual IP (VIP) that is used for accessing the master and hot-standby nodes. The VIP must be an address that doesn't correspond to an actual physical network interface but must be in a valid IP range for that Network.	None
path_bin	No	The path to the bin directory PATH_BIN directory.	/usr/lib64/ovlm
logging: level: com.openet. modules.ovlm	No	<p>The value can be one of the following:</p> <ul style="list-style-type: none"> • TRACE • DEBUG • INFO • WARN • ERROR 	WARN
path_cfg	No	The path to the PATH_CFG directory.	/etc/ovlm
path_run	No	The path to the PATH_RUN directory.	/var/run/ovlm
logging:file	No	The location of the logging files. When not specified the default location is used.	/var/log/ovlm/ <microservice>/
metric-files-size:	No		2MB
metric-files-history:	No		7
configuration_manager url:	Yes	The Configuration Manager REST API endpoint in the form http://<host>:<port>.	None
frontend: url:	Yes	The Workflow Front-End REST API endpoint in the form http://<host>:<port>.	None

fault_manager Settings

The fault_manager block is where you configure the Fault Manager microservice properties.

The following is an example of a fault_manager block:

```
# Example Fault Manager
fault_manager:
  hosts:
    - node-1
  properties:
    server:
      port: 28100
    snmp:
      trap_port: 16200
```

The following table lists configurable fault_manager properties you can define in the installation file. You must define host and port settings specific to the environment. Specify properties under the properties heading, except where noted.

Property	Mandatory	Description	Default Value
hosts:	Yes	<p>The name of the microservice host machines.</p> <p>Note: If you deploy VIM and the DNS is not configured on the VIM, the hosts <i>must</i> be specified by IP address instead of by hostname.</p> <p>There must be an entry for the master host and the hot-standby (backup) host when keepalived is used, for example:</p> <pre># Configuration Manager Configuration configuration_manager: hosts: - <master_host> - <hot-standby_host></pre>	None
server: port:	Yes	<p>The port on which a microservice runs the REST API listener. Any port number greater than 1024. The port number must be less than the ephemeral port range defined on the host.</p> <p>The sample installation YML file provided with Openet Weaver uses a port setting of 28100 for fault_manager.</p>	None
override_host:	No	For keepalived setup, this is the Virtual IP (VIP) that is used for accessing the master and hot-standby nodes. The VIP must be an address that doesn't correspond to an actual physical network interface but must be in a valid IP range for that Network.	None
path_bin	No	The path to the bin directory PATH_BIN directory.	/usr/lib64/ovlm
logging: level: com.openet. modules.ovlm	No	<p>The value can be one of the following:</p> <ul style="list-style-type: none"> • TRACE • DEBUG • INFO • WARN • ERROR 	WARN
path_cfg	No	The path to the PATH_CFG directory.	/etc/ovlm
path_run	No	The path to the PATH_RUN directory.	/var/run/ovlm
logging:file	No	The location of the logging files. When not specified the default location is used.	/var/log/ovlm/<microservice>/

Property	Mandatory	Description	Default Value
server: snmp: trap_port:	No	The port on which Fault Manager listens for SNMP requests.	16200
configuration_manager_url	Yes	The Configuration Manager REST API endpoint in the form http://<host>:<port>.	None
frontend: url:	Yes	The Workflow Front-End REST API endpoint in the form http://<host>:<port>.	None

vim_abstraction_layer Settings

The vim_abstraction_layer block is where you configure the VIM Abstraction Layer (VAL) microservice properties.

The following is an example of a vim_abstraction_layer block:

```
# Example VIM Abstraction Layer
vim_abstraction_layer:
  hosts:
    - node-1
  properties:
    server:
      port: 28040
    openstack:
      credentials:
        tenant: demo
        username: admin
        password: secret
        auth-url: http://openstack-host:5000/v2.0
```

The following table lists configurable vim_abstraction_layer properties you can define in the installation file. You must define host and port settings specific to the environment. Specify properties under the properties heading, except where noted.

Property	Mandatory	Description	Default Value
hosts:	Yes	<p>The name of the microservice host machines.</p> <p>Note: If you deploy VIM and the DNS is not configured on the VIM, the hosts <i>must</i> be specified by IP address instead of by hostname.</p> <p>There must be an entry for the master host and the hot-standby (backup) host when keepalived is used, for example:</p> <pre># Configuration Manager Configuration configuration_manager: hosts: - <master_host> - <hot-standby_host></pre>	None
server: port:	Yes	<p>The port on which a microservice runs the REST API listener. Any port number greater than 1024. The port number must be less than the ephemeral port range defined on the host.</p> <p>The sample installation YML file provided with Openet Weaver uses a port setting of 28040 for vim_abstraction_layer.</p>	None

Property	Mandatory	Description	Default Value
override_host:	No	For keepalived setup, this is the Virtual IP (VIP) that is used for accessing the master and hot-standby nodes. The VIP must be an address that doesn't correspond to an actual physical network interface but must be in a valid IP range for that Network.	None
path_bin	No	The path to the bin directory PATH_BIN directory.	/usr/lib64/ovlm
logging:level: com.openet.modules.ovlm	No	The value can be one of the following: <ul style="list-style-type: none"> • TRACE • DEBUG • INFO • WARN • ERROR 	WARN
path_cfg	No	The path to the PATH_CFG directory.	/etc/ovlm
path_run	No	The path to the PATH_RUN directory.	/var/run/ovlm
logging:file	No	The location of the logging files. When not specified the default location is used.	/var/log/ovlm/<microservice>/
repository-root:	No	The directory where the VIM abstraction layer stores all VIM-workflow-related configuration and metadata.	/usr/lib64/ovlm/vale/repository-root
multipart:maxFileSize:	No	The maximum size limit for uploaded files.	4096MB
multipart:maxRequestSize:	No	the maximum size limit for multipart or form-data requests.	4096MB
process-timeout:	Yes	The timeout period for translation process execution, in millisecond.	10000
stack-removal-t	No	Specifies whether the stack is removed from OpenStack automatically when the process times out. Possible values are: <ul style="list-style-type: none"> • true (automatic removal) • false (the stack must be removed manually) Use the default value of true under normal circumstances If you are troubleshooting the system it can be useful to set this attribute to false so that the stack remains while you review the issue.	true
status-polling:max-retries	No	The number of times OpenStack attempts to perform either of the following tasks before failing the operation: <ul style="list-style-type: none"> • Stack creation • Stack deletion 	10
status-polling:retry-level	No	The interval, in seconds, between attempts at performing the operations specified in status-polling: max-retries.	5
storage:minimallocation	No	An integer representing the minimum amount of storage space that is allocated in bytes. A value of zero means that there is no minimum limit.	1048576
python-path:	No	The path to the python interpreter.	/usr/bin/python

Property	Mandatory	Description	Default Value
openstack:domain	No	The domain part of the server hostname that is appended to the VDU name when resources are allocated to a VNF, for example anycom.com. The VDU name is specified in the VNFD file. when this field is not populated the hostname consists of only the VDU name.	-
openstack:credentials:tenant	Yes	The OpenStack project/tenant that is used by Openet Weaver. Note: If OpenStack is using Keystone identity service v3 the tenant must be specified as a UUID value, for example ce348ac4-60a4-4d53-be04-6a92651751b1.	admin
openstack:credentials:username	Yes	The OpenStack project username for authentication. The user must have admin privileges for the project and/or the tenant.	admin
openstack:credentials:password	Yes	The OpenStack project user password for authentication. Note: The password must be encrypted before deployment. For more information see Encrypting Passwords .	admin
openstack:credentials:region	No	Sets the availability zone. See Determining Available Regions for information about determining what value to use.	
openstack:credentials:auth-domain	Yes, if keystone identity service is used.	The OpenStack identity domain, which includes the high level containers for projects, users and groups. Note: The password must be encrypted before deployment.	empty
openstack:credentials:auth-url	Yes	The OpenStack Identity Service's endpoint. You can find the value for this property in the OpenStack Dashboard (Horizon) in the Identity field on the API tab of the Access & Security page.	None
openstack:credentials:release	Yes	The OpenStack release name.	liberty
mappingrefreshinterval:	No	An integer representing the refresh interval for updating flavor and image mapping files, in seconds. A value of zero means that updating occurs always. A value of -1 means never update.	0
delete-resource:querymaxretry:	No	An integer representing the maximum number of retries allowed when querying a resources status after resource deletion is triggered, in seconds.	10
delete-resource:queryretryinterval:	No	An integer representing the interval between queries when querying a resources status after resource deletion is triggered, in seconds.	5

instance_inventory_manager Settings

The instance_inventory_manager block is where you configure the Instance Inventory Manager microservice properties, mainly related to the housekeeping of persistent events.

The following is an example of an instance_inventory_manager block:

```
# Example Instance Inventory Manager
instance_inventory_manager:
```

```

hosts:
  - node-1
properties:
  server:
    port: 28120
  logging:
    level:
      com.openet.modules.ovlm: DEBUG

```

The following table lists configurable instance_inventory_manager properties you can define in the installation file. You must define host and port settings specific to the environment. Specify properties under the properties heading, except where noted.

Property	Multi	Description	Default Value
hosts:	Yes	<p>The name of the microservice host machines.</p> <p>Note: If you deploy VIM and the DNS is not configured on the VIM, the hosts <i>must</i> be specified by IP address instead of by hostname.</p> <p>There must be an entry for the master host and the hot-standby (backup) host when keepalived is used, for example:</p> <pre># Configuration Manager Configuration configuration_manager: hosts: - <master_host> - <hot-standby_host></pre> <p>Note: This property is not required for Resource Agent if VNFs are to be deployed with VIM configuration.</p> <p>Note: This value is ignored for auth_server but must be present to prevent errors when running the deploy script. However, the correct host value (the same one as on the certificate) must be present in the url parameter in the auth_server block.</p>	None
server: port:	Yes	<p>The port on which a microservice runs the REST API listener. Any port number greater than 1024. The port number must be less than the ephemeral port range defined on the host.</p> <p>The sample installation YML file provided with Openet Weaver uses a port setting of 28120 for instance_inventory_manager.</p>	None
override_host:	No	For keepalived setup, this is the Virtual IP (VIP) that is used for accessing the master and hot-standby nodes. The VIP must be an address that doesn't correspond to an actual physical network interface but must be in a valid IP range for that Network.	None
path_bin	No	The path to the bin directory PATH_BIN directory.	/usr/lib64/ovlm
logging: level: com.openet. modules.ovlm	No	<p>The value can be one of the following:</p> <ul style="list-style-type: none"> • TRACE • DEBUG • INFO • WARN • ERROR 	WARN
path_cfg	No	The path to the PATH_CFG directory.	/etc/ovlm

Property	Mandatory	Description	Default Value
path_run	No	The path to the PATH_RUN directory.	/var/run/ovlm
logging:file	No	The location of the logging files. When not specified the default location is used.	/var/log/ovlm/<microservice>/
persistence_storage	No	Type of storage to be applied. Note: Instance Inventory Manager can currently support only filesystem storage.	filesystem
repository-root	No	The directory on the host where Instance Inventory Manager stores all event records.	/var/lib/ovlm/info/repository-root
storage:minimumallocation	No	The minimum disk storage space, in bytes, required to be available for storing event records by Instance Inventory Manager .	1048576 (1 MB)
housekeeping:enable	No	Determines whether Inventory Instance Manager housekeeping is performed on event records, so that the growth of files being persisted is kept to a specified number of files. Possible values are: <ul style="list-style-type: none">• true• false The default value is true, which indicates that housekeeping activities are to be performed.	true
housekeeping:schedule	No	The time that housekeeping is scheduled to occur, in cron notation (second, minute, hour, date, month, day of week). Important: If incorrect notation is used, housekeeping is not performed even when enabled.	0 0 3 * * MON (Every Monday at 3 am)
housekeeping:active-record-count	No	The number of file records to be retained. The most recent event records are retained for each unique VNF instance. A VNF instance record is considered active from the time the VNF instance is instantiated until the time it is terminated. Note: The setting also applies to each VNFC instance record under each VNF. A unique VNF instance record is a combination of tenant_id, vnf_id and vnf_instance_id. Note: You must not set the value below 20. If the value is set below 20, it is defaulted to 20 internally.	20 (records)

Property	Mandatory	Description	Default Value
housekeeping: retain: inactive-recordduration	No	<p>The duration, in minutes, that inactive VNF instance event records are retained. A VNF instance record is considered inactive when an instantiated VNF instance is terminated.</p> <p>A unique VNF instance record is a combination of tenant_id, vnf_id and vnf_instance id.</p> <p>The housekeeping task removes inactive records that exceed the inactive-record-duration value since the record creation time:</p> <ul style="list-style-type: none"> • Inactive and within inactive-record duration—the most recent event record for the unique VNF instance is retained. • Inactive and beyond inactive-record-duration—all event records for the unique VNF instance are deleted <p>Note: In both cases, the setting also applies to each VNFC instance record under each VNF.</p> <p>A value of 0 (expires immediately) is used if the attribute is set to less than zero.</p>	10080 (7 days in minutes)

auth_server Settings

The auth_server block is where you configure the Authorization Server microservice properties.

Note: If Authorization Server configuration is not provided, then no security is set up.

The following is an example of an auth_server block:

```
#auth_server:
  hosts:
    - node-1
  admin:
    username: admin
    password: password
  url: http://node-1:8686/auth
  properties:
    server:
      port: 8686
```

The following table lists configurable auth_server properties you can define in the installation file. You must define host and port settings specific to the environment. Specify properties under the properties heading, except where noted.

Property	Mandatory	Description	Default Value
hosts:	Yes	<p>The name of the microservice host machines.</p> <p>Note: If you deploy VIM and the DNS is not configured on the VIM, the hosts <i>must</i> be specified by IP address instead of by hostname.</p> <p>There must be an entry for the master host and the hot-standby (backup) host when keepalived is used, for example:</p> <pre># Configuration Manager Configuration configuration_manager: hosts: - <master_host> - <hot-standby_host></pre> <p>Note: This value is ignored for auth_server but must be present to prevent errors when running the deploy script. However, the correct host value (the same one as on the certificate) must be present in the url parameter in the auth_server block.</p>	None
server: port:	Yes	<p>The port on which a microservice runs the REST API listener. Any port number greater than 1024. The port number must be less than the ephemeral port range defined on the host.</p> <p>Note: This value is ignored for auth_server but must be present to prevent errors when running the deploy script. However, the correct port value (the same one as on the certificate) must be present in the url parameter in the auth_server block.</p>	None
override_host:	No	For keepalived setup, this is the Virtual IP (VIP) that is used for accessing the master and hot-standby nodes. The VIP must be an address that doesn't correspond to an actual physical network interface but must be in a valid IP range for that Network.	None
path_bin	No	The path to the bin directory PATH_BIN directory.	/usr/lib64/ovlm
logging: level: com.openet. modules.ovlm	No	<p>The value can be one of the following:</p> <ul style="list-style-type: none"> • TRACE • DEBUG • INFO • WARN • ERROR 	WARN
path_cfg	No	The path to the PATH_CFG directory.	/etc/ovlm
path_run	No	The path to the PATH_RUN directory.	/var/run/ovlm
logging:file	No	The location of the logging files. When not specified the default location is used.	/var/log/ovlm/<microservice>/
admin: username	Yes	The authorization server administrator username	
admin: password	Yes	<p>The authorization server administrator password.</p> <p>Note: The password must be encrypted before deployment. For more information see Encrypting Passwords.</p>	

Property	Mandatory	Description	Default Value
admin: url	Yes	The authorization server base URL. If you are using HTTPS the hostname and port number must be compatible with the certificate.	

reference_vnf Settings

The reference_vnf block is where you configure the settings for managing a reference VNF. The following is an example:

```
reference_vnf:
  properties:
    repository-root: /opt/ovlm/reference_vnf/repo-root
    logging:
      level: WARNING
    metadata:
      remote_execute:
        timeout: 120
      preinstall:
        timeout: 120
      stage:
        timeout: 120
```

The following table lists configurable reference_vnf properties you can define in the installation file. Specify properties under the properties heading, except where noted.

Property	Mandatory	Description	Default Value
repository-root	No	The absolute path to the reference VNF's root directory. This has the same functionality as resource_agent repository root.	/usr/lib64/ovlm/reference_vnf/repository-roo
logging:level	No	Logging level used for python procedure. Possible values are: <ul style="list-style-type: none"> • DEBUG • INFO • WARNING • ERROR • CRITICAL 	INFO
logging:rotate_interval_days	No	The amount of days that elapses before logs are rotated.	1
logging:rotate_max_size_bytes	No	An integer representing the maximum file size in bytes.	5000000
metadata: remote_execute: timeout	No	The amount of time allowed to run procedure scripts and transfer reference VNF data from the Resource Manager to the Reference VNF node. This time out specifically relates to running the following procedures: <ul style="list-style-type: none"> • stage • dynamic_config_reference_vnfc • install_reference_vnfc • update_config_reference_vnfc • post_install_reference_vnfc 	600

Property	Mandatory	Description	Default Value
metadata: preinstall: timeout	No	The amount of time allowed for the Resource Manager to run the preinstall script before execution of the script times out.	60
metadata: stage: timeout	No	The amount of time allowed for the stage procedure to run before execution of the procedure times out.	60
metadata:stage : parameters: rsync: timeout	No	The amount of time, in seconds before execution of the Rsync process times.	60
metadata:stage : parameters: rsync: bin	No	The path to where the Rsync command is invoked.	rsync
metadata:stage : parameters: rsync: share_name	No	The name of the Rsync module.	"":ovlm"
metadata:stage : parameters: rsync: secret_user	No	The name of the Rsync user.	ovlm
metadata:stage : parameters: rsync: ssh_user	No	The parameter used for setting SSH user tunneling.	ovlmrsync
metadata:stage : parameters: rsync: ssh_port	No	The port on which the reference_vnf Rsync listens for SSH traffic.	28000
metadata:stage : parameters: rsync: secret_file	No	The location of the Rsyncd secret file.	/etc/ovlm/reference_vnf/cm.rsync.secret

Install Properties Prioritization

When you create an installation file you specify service properties for that host in the service properties block. Default properties might also be defined by the microservices in their respective RPM packages. Any of those properties that are redefined in the installation file take precedence over values set in the RPM.

Therefore the priority order is as follows:

1. Host Level Properties
2. Defaults in RPM

This topic provides examples of property prioritization.

Multiple Hosts with Identical Service Properties

Example 1 shows a Resource Agent configuration in an Installation file. In this example the service is installed on two hosts. The port numbers, defined as service properties, are the same for both hosts.

```
# Resource Agent Configuration
resource_agent:
  hosts:
    - server01
    - server02
  properties:
    server:
      port: 29999
      protocol: http
```

Application Configuration from RPM and Installation File

Example 2 shows a resource agent configuration taken from an RPM package and an installation file with service-level configuration, and host-level configuration for an application.

The RPM has the following Configuration:

```
server:
    port: 9090

repository-root: "/usr/lib64/ovlm/ra/repository-root"

logging:
    level:
        com.openet.modules.ovlm.resourceagent: DEBUG

multipart:
    maxFileSize: 4096MB
    maxRequestSize: 4096MB

default:
    tenant_id: default
```

In the installation file, Resource Agent has the following configuration:

```
# Resource Agent Configuration
resource_agent:
    properties:
        server:
            port: 28080
            protocol: http
```

In this example, the RPM port default value of 9090 is overridden in the installation file. In the file, the service properties has the port defined as 28080, and that default value is configured for the resource agent.

Determining the Availability Region

One of the OpenStack installation properties is openstack: credentials: region. You can run a curl command in your OpenStack environment in order to determine the value to assign to the attribute.

The following command and response shows the curl command and example output. In this case RegionOne is the availability area for the system, and the value to which OpenStack: credentials: region should be set.

```
1 curl \
2 -H "Content-Type: application/json" \
3 -d '{"auth":{"tenantName": "admin", "passwordCredentials":{"username": "admin", "password": "admin"}} }' \
4 http://10.0.124.61:5000/v2.0/tokens -k | python -mjson.tool | pygmentize
-1 json
5
6 {
7     "access": {
8         "metadata": {
9             "is_admin": 0,
10            "roles": []
11        },
12        "serviceCatalog": [
13            {
```

```

14         "endpoints": [
15             {
16                 "adminURL": "http://192.168.0.2:87
74/v2/25179b6bf9534e8a9b58d8de3b0dad5c",
17                     "id": "004861d8aeed4cccd8800d3c0f18caeae",
18                     "internalURL": "http://192.168.0.2:87
74/v2/25179b6bf9534e8a9b58d8de3b0dad5c",
19                     "publicURL": "http://10.0.124.61:87
74/v2/25179b6bf9534e8a9b58d8de3b0dad5c",
20                     "region": "RegionOne"
21             }
22         ],
23         "endpoints_links": [],
24         "name": "nova",
25         "type": "compute"
26     },
27     {
28         "endpoints": [
29             {
30                 "adminURL": "http://192.168.0.2:9696",
31                 "id": "55cc5dccc1485f893ce2802107aef1",
32                 "internalURL": "http://192.168.0.2:9696",
33                 "publicURL": "http://10.0.124.61:9696",
34                 "region": "RegionOne"
35             }
36         ],
37         "endpoints_links": [],
38         "name": "neutron",
39         "type": "network"
40     },
41     {
42         "endpoints": [
43             {
44                 "adminURL": "http://192.168.0.2:87
76/v2/25179b6bf9534e8a9b58d8de3b0dad5c",
45                     "id": "572071d572d3453b845f6a35343b8d5e",
46                     "internalURL": "http://192.168.0.2:87
76/v2/25179b6bf9534e8a9b58d8de3b0dad5c",
47                     "publicURL": "http://10.0.124.61:87
76/v2/25179b6bf9534e8a9b58d8de3b0dad5c",
48                     "region": "RegionOne"
49             }
50         ],
51         "endpoints_links": [],
52         "name": "cinderv2",
53         "type": "volumev2"
54     },
55     {
56         "endpoints": [
57             {
58                 "adminURL": "http://192.168.0.2:8774/v3",
59                 "id": "15c033bb5a704ae89164cf493e61c3da",
60                 "internalURL": "http://192.168.0.2:8774/v3",
61                 "publicURL": "http://10.0.124.61:8774/v3",
62                 "region": "RegionOne"
63             }
64         ],
65         "endpoints_links": [],
66         "name": "novav3",
67         "type": "computev3"
68     },
69     {

```

```

70         "endpoints": [
71             {
72                 "adminURL": "http://192.168.0.2:8080",
73                 "id": "4be552c0fe1b414dbeef82699925a91b",
74                 "internalURL": "http://192.168.0.2:8080",
75                 "publicURL": "http://10.0.124.61:8080",
76                 "region": "RegionOne"
77             }
78         ],
79         "endpoints_links": [],
80         "name": "swift_s3",
81         "type": "s3"
82     },
83     {
84         "endpoints": [
85             {
86                 "adminURL": "http://192.168.0.2:9292",
87                 "id": "35f89d38cd0d42b2aef138709835dc71",
88                 "internalURL": "http://192.168.0.2:9292",
89                 "publicURL": "http://10.0.124.61:9292",
90                 "region": "RegionOne"
91             }
92         ],
93         "endpoints_links": [],
94         "name": "glance",
95         "type": "image"
96     },
97     {
98         "endpoints": [
99             {
100                 "adminURL": "http://192.168.0.2:8000/v1",
101                 "id": "116966e115cb4420a1380a42435f7950",
102                 "internalURL": "http://192.168.0.2:8000/v1",
103                 "publicURL": "http://10.0.124.61:8000/v1",
104                 "region": "RegionOne"
105             }
106         ],
107         "endpoints_links": [],
108         "name": "heat-cfn",
109         "type": "cloudformation"
110     },
111     {
112         "endpoints": [
113             {
114                 "adminURL": "http://192.168.0.2:87
76/v1/25179b6bf9534e8a9b58d8de3b0dad5c",
115                 "id": "1ea2174edeed4161a62146e63a62e628",
116                 "internalURL": "http://192.168.0.2:87
76/v1/25179b6bf9534e8a9b58d8de3b0dad5c",
117                 "publicURL": "http://10.0.124.61:87
76/v1/25179b6bf9534e8a9b58d8de3b0dad5c",
118                 "region": "RegionOne"
119             }
120         ],
121         "endpoints_links": [],
122         "name": "cinder",
123         "type": "volume"
124     },
125     {
126         "endpoints": [
127             {
128                 "adminURL": "http://192.168.0.2:8773/services/Admin",

```

```

129                     "id": "8a6e413763ed4b9ca29de2e50e0930e0",
130                     "internalURL": "http://192.168.0.2:87
73/services/Cloud",
131                     "publicURL": "http://10.0.124.61:87
73/services/Cloud",
132                     "region": "RegionOne"
133                 }
134             ],
135             "endpoints_links": [],
136             "name": "nova_ec2",
137             "type": "ec2"
138         },
139     {
140         "endpoints": [
141             {
142                 "adminURL": "http://192.168.0.2:80
04/v1/25179b6bf9534e8a9b58d8de3b0dad5c",
143                 "id": "1ee0e4844ee64c588b540cfb0a0b1b13",
144                 "internalURL": "http://192.168.0.2:80
04/v1/25179b6bf9534e8a9b58d8de3b0dad5c",
145                 "publicURL": "http://10.0.124.61:80
04/v1/25179b6bf9534e8a9b58d8de3b0dad5c",
146                 "region": "RegionOne"
147             }
148         ],
149         "endpoints_links": [],
150         "name": "heat",
151         "type": "orchestration"
152     },
153     {
154         "endpoints": [
155             {
156                 "adminURL": "http://192.168.0.2:80
80/v1/AUTH_25179b6bf9534e8a9b58d8de3b0dad5c",
157                 "id": "aa944bcc50be4302a281d5cc43de19b0",
158                 "internalURL": "http://192.168.0.2:80
80/v1/AUTH_25179b6bf9534e8a9b58d8de3b0dad5c",
159                 "publicURL": "http://10.0.124.61:80
80/v1/AUTH_25179b6bf9534e8a9b58d8de3b0dad5c",
160                 "region": "RegionOne"
161             }
162         ],
163         "endpoints_links": [],
164         "name": "swift",
165         "type": "object-store"
166     },
167     {
168         "endpoints": [
169             {
170                 "adminURL": "http://192.168.0.2:35357/v2.0",
171                 "id": "1d5b833b35f54e819505dead3cd858a4",
172                 "internalURL": "http://192.168.0.2:5000/v2.0",
173                 "publicURL": "http://10.0.124.61:5000/v2.0",
174                 "region": "RegionOne"
175             }
176         ],
177         "endpoints_links": [],
178         "name": "keystone",
179         "type": "identity"
180     }
181 ],
182 "token": {

```

```

183         "audit_ids": [
184             "c7MTCa3qQdShL6QfyTIilQ"
185         ],
186         "expires": "2016-12-09T11:55:39.037414Z",
187         "id": "gAAAAABYSo2rFAQWshiiYoNT6Yy9bGsr1vw0AvCztNs
Qrg1c0EJj5PnXcq4O4_NFF4qiAKSu8oPnrf6h3SikMBB7hbF7ZoHiA1xhn4rLqz0ui
5fhy-bSB5GDxr4jQEcry8w4XM3ltHthsVaddcrV7q8xlcZOICn0UcNTtYArIvz0wJZfCFBJQ7E",
188         "issued_at": "2016-12-09T10:55:39.000000Z",
189         "tenant": {
190             "description": "admin tenant",
191             "enabled": true,
192             "id": "25179b6bf9534e8a9b58d8de3b0dad5c",
193             "is_domain": false,
194             "name": "admin",
195             "parent_id": null
196         }
197     },
198     "user": {
199         "email": "admin@localhost",
200         "enabled": true,
201         "id": "82c5b03b84ff4b5da8dd69d5333444b7",
202         "name": "admin",
203         "roles": [
204             {
205                 "id": "87c58773275a4fa6b393f97a77a46d15",
206                 "name": "admin"
207             }
208         ],
209         "roles_links": [],
210         "username": "admin"
211     }
212 }
213 }
```

Configuring Resource Agent Persistence for Logging and Repository Root

This topic describes how to configure Resource Agent persistence in the installation file so that Resource Agent logging and the root repository use a common path to centralized storage. To enable the centralized storage option, follow the example below resource_agent configuration below. In this example, /opt/ovlm is the centralized storage location.

```

resource_agent:
  properties:
    append_hostname_to_path: true                                # new support
(default:false)
  server:
    port: 28030
  logging:
    path: /opt/ovlm/logs/ra                                    # optional
    level:
      com.openet.modules.ovlm: DEBUG
  sudo_privileges_required: true
  sudo_privileges:
    - /bin/postgresql-setup
  resource_agent:
    repository-root: /opt/ovlm/ra/repository-root/          # optional
    procedures-log-path: /opt/ovlm/logs/ra/procedures/      # optional, if not
specified will be default to ${logging:path}/procedures
```

In the example above the append_hostname_to_path parameter is set to true so that the following have a child directory named after the hostname:

- application-log-path
- repository-root-path
- procedures-log-path

The logging path, resource_agent: repository root, and resource_agent: procedures-log-path parameters are configured so that they all use a location under /opt/ovlm for storage. All three parameters have the default paths listed in the installation file when not declared specifically.

The following is an example of the log directory tree based on the resource_agent configuration above.

```

opt
`--ovlm
  '--logs
    '--ra
      |--node1
      |   '--ovlm-ra.log
      |--node2
      |   '--ovlm-ra.log
      '--procedures
        |--node1
        |   |--stage_v5.log
        |   |--dynamic_config_vnfc_v2.log
        |   |--install_vnfc_v2.log
        |   |--update_v2.log
        |   |--update_config_vnfc_v3.log
        |   |--start_vnfc_v2.log
        |   '--is_vnfc_up_v2.log
        '--node2
          |--stage_v5.log
          |--dynamic_config_vnfc_v2.log
          |--install_vnfc_v2.log
          |--update_v2.log
          |--update_config_vnfc_v3.log
          |--start_vnfc_v2.log
          '--is_vnfc_up_v2.log

```

The following is an example of the repository root directory tree based on the resource_agent configuration.

```

opt
`--ovlm
  '--ra
    '--repository-root
      |--node1
      |   |--stage
      |   |--preactive
      |   |--active
      |   '--rollback
      '--node2
        |--stage
        |--preactive
        |--active
        '--rollback

```

Encrypting Passwords

Encrypting the plain-text passwords enhances security. You are required to encrypt passwords in the installation file before deployment. If you do not encrypt an error will be generated.

You can also encrypt passwords after deployment, for example when you change a password. Doing so requires a few additional steps.

The following table lists the passwords to be encrypted and the associated microservices.

Password	Associated Microservice
auth_server: admin: password	Authorization Server
protocol: https_configuration: server: keystore_password	All microservices
protocol: https_configuration: server: key_password	All microservices
vim_abstraction_layer: openstack: credentials:password	VAL Abstraction Layer
deployment_manager: database: credentials: password	Deployment Manager

Encrypting Passwords Before Deployment

Assuming that all your microservice configuration is done through the installation file, after you complete updating that file with plain text passwords you can encrypt them by running the command `<weaver_install_directory>/ovlm/utilities/cipher/encrypt_scripts/encrypt-password.sh -f path to installer.properties file`

Note: `<weaver_install_directory>` is the directory to which the TAR file was extracted.

 **CAUTION:** Enter only plain text passwords in the installation file before you run the encryption script. Do not copy encrypted passwords into these files. Otherwise you run the danger of encrypting the passwords twice, in which case they do not work.

Running the command encrypts all the passwords in the file, making them ready for deployment. An encryption file (.installer.png) is generated when the encryption script is run successfully. This encryption file is used to decrypt the password in the associated configuration file at microservice runtime or during other operations. The .installer.png encryption file is stored in the Openet Weaver installation directory.

 **Attention:** The encryption file must be protected, kept safe and, if necessary, reused if Openet Weaver needs to be reinstalled. It is crucial to successful password decryption in Openet Weaver microservices and other services. Without decryption, password-protected services are unavailable.

Note: Although it is possible to run the encrypt-password script against microservice configuration files, it is recommended that you perform all configuration through the install file, rerunning the deployment script if reconfiguration is required.

Encrypting individual Passwords after Deployment

To make a post-deployment change to an individual password, or a subset of passwords you can run the encrypt-password.sh script using the plain text password as input using the command `./ovlm-install/utilities/cipher/encrypt_scripts/encrypt-password.sh password`.

After running the command, copy the output to the corresponding attribute in the installation file. When you are done, run the installer again to update Openet Weaver.

Performing the Initial Installation

This topic describes how to install Openet Weaver after the pre-installation tasks are completed.

Pre-Installation Summary

At this point in the installation process you have, at a minimum performed the following tasks:

- Transferred the Openet Weaver TAR file you received to the ovlm-install directory or the installer node.
- Extracted the TAR files on to the disk.
- Change directory to ovlm-install
- Configured users and privileges.

- Configured Rsync using either the Openet Weaver supported Rsync implementation or a custom Rsync implementation, a task that, for the Openet Weaver supported Rsync implementation, includes generating an SSH key pair and placing the generated files inside the ssh-keys folder.
- Note:** You are responsible for the Rsync configuration for custom Rsync deployments.
- Performed some or all of the optional pre-installtion tasks as described in [Performing Pre-Installation tasks](#) which, depending on your specific deployment might include:
 - Configuring Openet Weaver for HTTPS, a task that incldes generating certificates and CA bundles, and setting up server and client truststores.
 - Installing and configuring Keycloak
 - Configuring VIM (OpenStack) for Openet Weaver
 - Configuring master and hot-standby nodes for HA.
 - Created an installation file and encrypted all passwords in the file.

To review the requirements and steps for these tasks see [Extracting the Tar File](#) and [Performing Pre-Installation tasks](#).

When you are satisfied that you have completed the prerequisite tasks above that are applicable to you're deployment, you are ready to install Openet Weaver.

The Installation Machine

Note: You can run the installation process from one of the machines you are using as an Openet Weaver node, or you can run it from a separate machine provided it has the same specifications (or higher) as described in [Hardware and Software Requirements](#).

Note: You require sudo access or you must be the root user to run the installation script. Additionally, since CLI packages are installed over SSH (simulating remote management) it is recommended that you configure passwordless SSH from the node executing the deployment to itself where local packages are installed. This is more a convenience than a requirement, because if passwordless SSH is not configured you are simply prompted for a password as in the following example.

```
2016-03-28 00:05:15,030 - INFO - ovlm-deploy - <module> - Locally installing
CLI packages for Configuration Manager and Front-end
[localhost] Login password for 'root':
2016-03-28 00:05:29,038 - INFO - ovlm-deploy - install_rpm - Installing
ovlm-configuration-manager-cli on localhost
2016-03-28 00:05:30,838 - INFO - ovlm-deploy - install_rpm - Installing
ovlm-workflow-frontend-cli on localhost
```

Note: As part of installation the deployment utility installs several RPM packages locally on the node that is executing the deployment process. The first software package installed is the ovlm-dependencies package, upon which the deployment utility depends. The CLI packages for Configuration Manager and the Front-end are also installed locally, which allows for convenient execution of CLI commands post deployment.

Installing Openet Weaver

- To install Openet Weaver, including all microservices *except* for Resource Agents, run the Openet Weaver installer by entering the following command:

A Resource Agents (RA) is installed automatically on a runtime node as part of the VNF Instantiation process when Openet Weaver detects a runtime node does not have any Resource Agent installed based on the VNF instance topology file.

```
./ovlm-deploy.sh [-h]-i install_properties_file -u installation_user
[--create-sample-vnf] [--secure-stage-traffic] [--skip-local-cli-install]
```

Note: Not all arguments are shown, only those that pertain to initial installation.

The command line arguments to the installer for initial installation are outlined in the following table. For post-installation arguments see [Modifying Deployments](#).

Table 22: ovlm-deploy.sh initial installation parameters

Parameter	Description
-h	Displays the help for this command.
-i	The path to the install properties file that defines where to install each microservice and component. For example, ovlm-install.yml.
-u	<p>The installation user. The user must have sudo access or be root user on all the nodes that are installed. The user argument cannot have a value of "ovlm" since the deployment process creates remote users by that name, and those user have limited sudo access.</p> <p>See the Sudo Privilege Commands table in Users and Privileges for a list of commands that require sudo privileges.</p> <p>Note: It is recommended that password-less SSH is enabled between each node.</p>
--create-sample-vnf	<p>Dynamically creates the VNF topology based on the Openet Weaver topology for the out of the box VNF and creates ready-to-deploy packages from the VNF.</p> <p> Warning: When using the override-host parameter to in the installation file to configure keepalived, you cannot create the sample vnf in the initial installation because keepalived is not yet running so there is no virtual IP assigned to Master or Backup node. After keepalived is up and running you can run the install script with the --create-sample-vnf option.</p>
--secure-stage-traffic	Used to enable stage traffic encryption for ssh tunelling. User ovlmrsync is created locally and on the NIS domain server, unless it already exists and is enabled in both places.
--skip-local-cli-install	<p>Skips local installation of Configuration Manager and Front-end CLIs.</p> <p>Note: This option cannot be used together with --create-sample-vnf parameter because --create-sample-vnf might not work if the Configuration manager CLI is not installed on local machine.</p>

The following is an example of running the install script:

```
./ovlm-deploy.sh -i ovlm-install.yml -u
root
```

Running the script creates a log file for the install in the directory from which you executed the installer. If you receive any installation errors, review the log file for troubleshooting information.

The log filename is ovlm-deploy-log-<date>-<time>.log

- Run the following command to prepare the files that must be uploaded to Deployment Manager.

```
./ovlm-dm-util.sh prepare upload_files
```

The following is an example of a response to the command:

```
2017-10-06 15:08:33,910 - INFO - ovlm-dm-util - <module> - Log file for this
DM util: ovlm-dm-util-log-20171006-150833.log
2017-10-06 15:08:33,924 - INFO - ovlm-dm-util - pack_ovlm_installer - Packing
OVLM installer into single tarball...
2017-10-06 15:08:41,581 - INFO - ovlm-dm-util - pack_ovlm_installer - Completed
packing OVLM installer.
2017-10-06 15:08:41,581 - INFO - ovlm-dm-util - pack_ovlm_configuration -
```

```
Packing OVLM nstallation configuration...
2017-10-06 15:08:42,615 - INFO - ovlm-dm-util - pack_ovlm_configuration -
Completed packing OVLM installation configuration.
2017-10-06 15:08:42,615 - INFO - ovlm-dm-util - list_prepared_files - List
of prepared files for uploading to Deployment Manager
INSTALL_FILE location : /home/ovlminstall/testHome/working/ovlm-inst
all/artifact/ovlm-install.tar.gz
CERTIFICATE_FILE location : /home/ovlminstall/testHome/working/ovlm-inst
all/artifact/certs.tar.gz
RSYNC_SSH_KEY_FILE location : /home/ovlminstall/testHome/working/ovlm-inst
all/artifact/ssh-keys.tar.gz
ENCRYPTION_KEY_FILE location : /home/ovlminstall/testHome/working/ovlm-inst
all/artifact/.installer.png
```

3. If security authentication(Keycloak) is enabled, log in to the authentication service by entering the following command:

```
eval $(on-auth-client -u <username> -p <password>)
```

Note: Skip this step if security authorization is not enabled.

4. Upload Resource-Agent-related binary files to Deployment Manager by entering the following command:

```
ovlm-dm artifact upload -f <installation_file>
```

The installation_file parameter can refer to the INSTALL_FILE location as shown in the response example for step 2.

5. Upload configuration files to Deployment Manager by entering the following command:

```
ovlm-dm configuration create [-i<installation_file>] [-u <installation_user>]
[-secure_stage_traffic <true/false>] [-s <rsync_ssh_key_file>] [-e
<encryption_key_file>] [-c <certificate_file>] [-k <installer_ssh_key_file>]
```

Be aware of the following when you run the configuration create command:

- The installer_ssh_key_file must be located at \$HOME/.ssh/id_rsa
- The rsync_ssh_key_file parameter can refer to the RSYNC_SSH_KEY_FILE location as in the example in step 2.
- The encryption_key_file parameter can refer to the ENCRYPTION_KEY_FILE location as in the example in step 2.
- The certificate_file parameter can refer to the CERTIFICATE_FILE location as in the example in step 2, and is required only when running Openet Weaver in HTTPS mode.

Performing Post-Installation Tasks

This section describes Openet Weaver configuration tasks to be performed after running the installer. Some are mandatory, such as creating a sudo user and configuring Rsync to run in SSH mode. Others are not mandatory but are good practice, such as setting environment variables or checking the health status of the system. Configuring OpenStack is technically unnecessary but can be useful if you are creating VNFs for use in Virtual Infrastructure Management. Configuring Keycloak is required only if you installed Openet Weaver with Keycloak security.

Setting Environment Variables

All Openet Weaver commands require a baseurl that specifies the host and port of the microservice that corresponds to the command.

Note: If you are deploying Keycloak with Openet Weaver, there is a properties file that contains the baseurl environment variables defined in this topic. If you are not deploying Keycloak you must define them manually as below.

The baseurl can be passed as a command argument in the following form:

baseurl=http://localhost:28050

However, it is much more convenient (and less prone to error) if you declare base URLs as environment variables when you deploy Openet Weaver.

Any parameter can be defined as an environment variable as follows:

OVLM<microservice>_<parameter>

For example, the base URL argument can be set at the CLI for the front end as an environment variable as follows:

```
export OVLMFE_BASEURL=http://host:port
```



Attention: When you are deploying master and hot-standby nodes for local high availability (keepalived), use the host specified as the override_host in the installation file so that the environment variable continues to work in the event of failover.

The following example sets environment variables so that you can run Front End, Configuration Manager, Performance Manager, and Fault Manager CLI commands without passing baseurl arguments at runtime.

```
export OVLMFE_BASEURL=http://node-1:28050
export OVLMCM_BASEURL=http://node-1:28010
export OVLMPM_BASEURL=http://node-1:28090
export OVLMFM_BASEURL=http://node-1:28100
export OVLMMDM_BASEURL=http://localhost:28130
export OVLMFA_BASEURL=http://localhost:28800
export OVLM_BASEURL=http://10.3.18.26:28888
```

Note: See your installation file for actual host names and port numbers.

Installing the CLI on a Client Node

You can install the Openet Weaver Command Line Interface (CLI) on a non-management node by copying the self-extracting ovlm-client installer to that node and running it.

The stand-alone CLI allows you to run commands on the following microservices:

- Front-End
- Configuration Manager
- Performance Manager
- Fault Manager

As a prerequisite for Installing the CLI you must have a RHEL-compatible environment with YUM and RPM managers installed. You must also have direct access to the YUM RPM repositories from which software package dependencies can be installed.

To install the CLI use the command `sudo bash ovlm-client-<release_version>.sh`

For example

```
sudo bash ovlm-client-1.6.0.sh
```

Running the command installs the CLI RPMs and produces output similar to the following.

```
OVLM CLI Bundle Installer
Package python-2.7.5-48.el7.x86_64 already installed and latest version
Package python-dateutil successfully installed
Package python-setuptools successfully installed
Package python-six successfully installed
Package python-urllib3 successfully installed
Package python-yaml successfully installed
Package config-manager-cli-1.6.0-SNAPSHOT.rpm successfully installed
```

```
Package fault-manager-cli-1.2.0-SNAPSHOT.rpm successfully installed
Package front-end-cli-1.6.0-SNAPSHOT.rpm successfully installed
Package performance-manager-cli-1.4.0-SNAPSHOT.rpm successfully installed
```

Verifying Resource Agent Connectivity

You might have configured Rsync to run in SSH mode as one of the pre-installation tasks. If you did, after installation you can check whether ssh tunneling was enabled successfully by using the following API command at the Resource Agent nodes.

```
curl http://localhost:28030/connectivity/<service>
```

In this case, the service is rsync.

Note: The connectivity API can also accept performance_manager or fault_manager as the service parameter. If you do not specify a service, all three microservices are checked for connectivity status.

The form of the response is as follows.

```
{
  "<service>" : {
    "url" : "<connection string>",
    "status" : "SUCCESS | FAILED | DISABLED",
    "secured" : "true | false"
  }
}
```

The following table describes the response to the connectivity API.

Table 23: Connectivity API Response Description

Response	Description
<service>	The microservice being tested for connectivity.
url	A string with connection parameters that are used to check connectivity.
status	The connection status, which can be one of the following: <ul style="list-style-type: none"> SUCCESS—the connection is working. FAILED—there are connection or configuration issues. DISABLED—the current service is disabled in the Resource Agent configuration file.
secured	When the service is rsync this is the SSH endpoint for rsync to establish an encrypted channel Note: When the service is Performance Manager or Fault Manager, the response is a true or false value that indicates whether the connection is secured.

The following is an example of an API call and response to check for rsync connectivity.

```
curl http://localhost:28030/connectivity/rsync
{
  "rsync" : {
    "url" : "rsync://localhost:28081/:ovlm",
    "status" : "FAILED",
    "secured" : "ssh -L 28081:localhost:28000 ovlmrsync@localhost"
  }
}
```

Keycloak Security Post Installation Setup

If you installed Keycloak as part of your deployment, you must log in to the Keycloak Web User Interface and configure the application users who will be using Openet Weaver.

Note: Each time a user begins a secure session using Openet Weaver they must run an authorization script to obtain an access token that will be used by the CLI and API commands in this document. for more information see [Logging in to a Secure Environment](#).

You should also be aware of the Weaver.profile file that is created during deployment.

Configuring System Users

For users to be able to log into weaver, either using the VNF-M GUI or from the microservice CLIs you first need to create those users in Keycloak.

Using Keycloak you can create new users or import a user base from another source - such as LDAP or Active Directory. This topic provides an example of creating one user.

to create a user, follow these steps:

1. Navigate to the Keycloak administration console. This can be found at <protocol>://keycloak-host:8443/auth/admin/
2. Log in using your Administrator credentials.
3. In the Keycloak main interface, click **Users** on the left menu bar.

This User List page displays. Here you can view all users within the Weaver Realm, which is the only realm in which you should create users. Before you create your first user there should be no users in that realm.

Note: You can add users to the master realm to allow extra administrators for the system.

4. Click **Add user**, which is on the right.

The Add User page displays.

5. Complete all relevant fields.
6. Click **Save** when you are finished.

The Users Detail page displays, where you can add additional attributes..

7. To add login credentials, click on the Credentials pane, then type in a password for the user in both fields. Make sure the **Temporary Password** field is set to off.
8. Click **Reset Password** when finished.

Alternatively, as a Keycloak admin user you can create keycloak operational users from the CLI using the following command:

```
sudo -u <keycloak_admin_user> bash -c "export JAVA_HOME=/opt/ovlm/keycloak/jdk1.8.0_60; /opt/ovlm/keycloak/opt/bin/add-user-keycloak.sh -u <keycloak_operational_username> -p <keycloak_operational_user_password> -r weaver"
```

For more information about configuring users see the Keycloak documentation.

Profile File

During deployment a Profile file is created named Weaver.profile. The file is generated to the same directory where the ovlm-deploy.sh script is executed. This file contains node locations to the following, which it also exports:

```
OVLMCM_BASEURL
OVLMFM_BASEURL
OVLMPM_BASEURL
OVLMFE_BASEURL
```

```
ON_OIDC_CLIENT_ID
ON_OIDC_TOKEN_URL
```

The following is an example of the generated weaver.profile file when keycloak is installed:

```
OVLMCM_BASEURL=node-1
export OVLMCM_BASEURL
OVLMFM_BASEURL=node-1
export OVLMFM_BASEURL
OVLMPM_BASEURL=node-1
export OVLMPM_BASEURL
OVLMFE_BASEURL=node-2
export OVLMFE_BASEURL
ON_OIDC_CLIENT_ID=weaver-cli
export ON_OIDC_CLIENT_ID
ON_OIDC_TOKEN_URL=http://10.0.27.96:8686/auth/realms/weaver/protocol/o
penid-connect/token
export ON_OIDC_TOKEN_URL
```

Accessing Resources Through the VNF-M GUI

The Openet Weaver VNF-M GUI uses a cross-origin resource sharing (CORS) mechanism. Therefore, the browser you use with the VNF-M GUI must be able to access resources from the Front-end, (FE) Configuration Manager (CM), and Instance Inventory Manager (IIM) microservices.

The following is an example Openet Weaver installation file showing the CM configuration properties:

```
.....
configuration_manager:
  hosts:
    - node-1
  properties:
    server:
      port: 28010
      rsync_port: 28000
instance_inventory_manager:
  hosts:
    - node-1
  properties:
    server:
      port: 28120
frontend:
  hosts:
    - node-1
  properties:
    server:
      port: 28050
.....
```

The above example shows that the CM micro-service is installed on the host node-1 and port 28010. Therefore , your browser must be able to access http or https://node-1:28010 (depending on which protocol you deployed with) to interact with CM resources. Your browser also requires the access to the IIM and FE microservices.

The following is an example of how the three microservices are configured during installation in the /etc/ovlm/vnfm-gui/ovlm-vnfm-gui.yml file on the node where VNFM-GUI micro-service is installed.
The following is an example of the file before editing:

```
configuration_manager:
  url: http://node-1:28010
instance_inventory_manager:
  url: http://node-1:28020
```

```
frontend:
  url: http://node-1:28050
```

If a DNS server is not configured for the network, then you must ensure your browser has access to the microservices by replacing the hostname with a physical IP address in the file as shown below:

```
configuration_manager:
  url: http://10.0.2.10:28010
instance_inventory_manager:
  url: http://10.0.2.10:28020
frontend:
  url: http://10.0.2.10:28050
```

After making the configuration changes, you must restart the VNF-M GUI micro-service by running the following CLI command:

```
sudo systemctl restart ovlm-vnfm-gui
```

If Weaver is installed on nodes in a cloud environment (such as Amazon EC2 or OpenStack) using private IP, you might need to edit the file `/etc/ovlm/vnfm-gui/ovlm-vnfm-gui.yml` on the node where VNF-M GUI is installed to use public IP for the CM, FE, and IIM URLs.

Alternatively, you can map the IP address(es) to the hostnames in the host file on the computer where client browser is running. For example, assuming the IP address of node-1 is 10.0.2.10., you must modify `/etc/hosts` in the Linux environment, as shown in the example below:

```
10.0.2.10      node-1
```

Checking Health Status

A health status flow is provided that can be run post installation to verify the Openet Weaver services are running. The following command performs one health check per service:

Note: Be sure to set the `baseurl` environment variables before using CLI commands. See [setting environment Variables](#) for more information.

```
ovlm-fe health status
```

Note: See the [API reference](#) for the equivalent API command.

the following is an example result of running the health status command.

```
data:
  workflow_instance_status: COMPLETED
  flow_result:
    Output:
      services:
        - service_id: resource_manager
          instances:
            - url: http://ovlm-vm:28080/health
              status: running
        - service_id: fault_manager
          instances:
            - url: http://ovlm-vm:28100/health
              status: running
        - service_id: frontend
          instances:
            - url: http://ovlm-vm:28085/health
              status: running
        - service_id: vnfm_gui
```

```

instances:
- url: http://ovlm-vm:28200/health
  status: running
- service_id: configuration_manager
  instances:
  - url: http://ovlm-vm:28082/health
    status: running
- service_id: vim_abstraction_layer
  instances:
  - url: http://ovlm-vm:28060/health
    status: running
- service_id: performance_manager
  instances:
  - url: http://ovlm-vm:28090/health
    status: running
- service_id: instance_inventory_manager
  instances:
  - url: http://ovlm-vm:28120/health
    status: running
- service_id: deployment_manager
  instances:
  - url: http://ovlm-vm:28130/health
    status: running
- service_id: resource_agent
  instances:
  - url: http://rt1-ovlm-vm:28086/health
    status: running
  - url: http://rt2-ovlm-vm:28086/health
    status: running
  - url: http://rt3-ovlm-vm:28086/health
    status: running
  - url: http://rt4-ovlm-vm:28086/health
    status: running
request: http://ovlm-vm:28085/api/v2/health/status
method: GET
status: 200

```

Modifying Deployments

The Openet Weaver install script `./ovlm-deploy.sh` supports post-installation modifications, for example:

- Updating configuration
- Upgrading microservices
- Rolling back microservices
- Reinstalling microservice
- Scaling Deployments
- Reinstalling content packs
- Updating out-of-the-box procedures
- Skipping the Installation of local CLI
- Granting and Revoking OVLM User Sudo Privileges
- Restart microservices

For post-installation modification, you run the Openet Weaver installer using the following command:

Note: Not all arguments are shown, only those that pertain to post-installation modifications.

```

./ovlm-deploy.sh [-h]-i install_properties_file -u installation_user
[--force-install <service1, service2, ...>] [--install-cp] [all | pathname,...]

```

```
[--ootb-procedures-addon] [--skip-local-cli-install] [--restart <service>
[host-1, host-2,...]]
```

The command line arguments to the installer for post-installation are outlined in the following table. For initial installation arguments see [Performing the Initial Installation](#).

Table 24: ovlm-deploy.sh post-installation parameters

Parameter	Description
-i	The path to the install properties file that defines where to install each microservice and component. For example, ovlm-install.yml.
-u	The installation user. The user must have sudo access or be root user on all the nodes that are installed. The user argument cannot have a value of "ovlm" since the deployment process creates remote users by that name, and those user have limited sudo access. Note: It is recommended that passwordless SSH is enabled between each node.
--force-install [service1, service2, ...]	Forces software package installation for a list of Openet Weaver microservices or other services, for example keepalived. The following options are available: <ul style="list-style-type: none"> • configuration_manager • resource_manager • workflow_engine • frontend • instance_inventory_manager • fault_manager • vim_abstraction_layer • performance_manager • vnfm_gui • keepalived • auth_server • deployment_manager  CAUTION: Using --force-install with authorization server removes the existing Realm and creates a new one. To avoid the risk of the new realm being misconfigured, delete the existing realm before running --force unstall with auth_server. Note: The default behavior is that only new versions of software packages are installed. Same or older versions are skipped.
--install-cp	Install content packs only. The default option 'all' installs or reinstall all content packs into the Workflow Engine. Otherwise all files are installed by pathname.

Parameter	Description
--ootb-procedures-addon	<p>Updates the out of the box procedures in the ootb_procedures directory of the Configuration Manager repository. The source file that contains the procedures must be a TAR or GZ file. Any existing procedures are overwritten when this option is used.</p> <p>Note: Configuration manager's microservice has to be installed successfully prior to execution with this option. Therefore you cannot use this option on a clean install.</p> <p>After initial installation, you can enable security mode for an existing Configuration Manager by running the install script again using the --force-install configuration_manager parameter and passing the key to the product installer.</p>
--skip-local-cli-install	<p>Do not install CLIs for Configuration Manager and FrontEnd microservices.</p> <p>Note: This option cannot be used together with --create-sample-vnf parameter because --create-sample-vnf might not work if the Configuration manager CLI is not installed on local machine.</p>
--restart <service> [host-1, host2, ...]	<p>Restart all, or selected instances of an Openet Weaver microservice or other service type, for example keepalived.</p> <p>When --restart is used, it is mandatory to provide a service as an argument. Only one service can be restarted per script run. To restart multiple services, you must run the installer once for each service. for example:</p> <p>Services are specified using the same options as those for --force-install.</p> <p>Specifying instances is optional. When this option is not used, all instances are restarted for specified service types. When the option is used, instances are specified as nodes.</p> <p>Restart fails and an error is returned under the following circumstances:</p> <ul style="list-style-type: none"> • An incorrect or non-existent service is specified • An incorrect or non-existent service instance is specified • The specified service is not found in the topology file • A connection cannot be established to an instance node • In the event that a connection cannot be established to an instance node, check SSH connectivity and ensure the node has sudo access.

Limitations

The following limitations apply to modifying Openet Weaver deployments:

- When changes are made to the installation file, when the ovlm-deploy.sh script is run the changes are pushed to all microservices except for Resource Agents.
- Configuration changes take effect immediately on all active microservices. Active microservice the ones that were deployed successfully, as you can see from the deployment_status SUCCESS. Microservices that have a

deployment_status of SKIPPED have the configuration pushed to them, but those changes do not take affect until the microservice is restarted.

- In a scenario where configuration updates are made for Resource Agents only, the configurations are not pushed to other microservices. You can resolve this issue by using the same installation file you used to update the Resource Agents and following the steps in the [Updating Configuration for all Microservices except Resource Agents](#)
- As part of the ovlm-deploy.sh script run, it systemctl start is run on all microservices except for Resource Agents. Therefore, microservices that are down are started. Microservices that are already started are not affected by systemctl start.

The following table documents further limitations that apply to Resource Agents only:

Limitation	Remarks
Restarting Resource Agents	<p>Not supported.</p> <p>You must log in and restart manually, for example:</p> <pre>sudo systemctl restart ovlm-ra</pre>
Starting and stopping Resource Agents	<p>Not supported.</p> <p>You must log in and start or stop manually.</p> <p>For example, to stop a Resource Agent use the following command:</p> <pre>sudo systemctl restart ovlm-ra</pre> <p>To start a Resource Agent use this command:</p> <pre>sudo systemctl start ovlm-ra</pre>
Uninstalling Resource Agents	<p>Not supported.</p> <p>You must log in and uninstall manually, for example:</p> <pre>rpm -e --nodeps ovlm-resource-agent</pre> <p>Note: You must log in to Deployment Manager manually and run a query to update the uninstallation date.</p>
Updating Resource Agent configurations	<p>Very limited support for sudo privileges.</p> <p>You can log in to the Resource Agent node manually, change the configuration, then restart the Resource Agent.</p>

Performing Configuration Updates

This topic describes how to update Openet Weaver microservice configuration after initial installation.

Note: These steps assume you are retaining your ovlm-install workspace. If not you need to perform the tasks described in [Extracting the TAR File](#) and [Performing Pre-Installation Tasks](#) before performing configuration updates.

The following configuration changes are described in this topic:

- Updating SSL Certificates
- Updating SSH Keys
- Updating Configuration for all Microservices except Resource Agents
- Updating Configuration for Resource Agents using the Deployment Manager CLI

Most configuration changes are made by making changes to the Installation file. However, you can also update SSL certificates, SSH keys, or both in the same configuration update. You must then apply your changes by either updating all microservices except Resource Agents, or by updating Resource Agents only. In many cases you must do both.

Updating SSL Certificates

If you are running Openet Weaver in HTTPS mode you must perform this task. To do so, perform these steps:

1. Generate the required files for the certificates. Refer to [SSL Certificates and Generating CA Bundles](#).
2. Set up server's keystore and client's truststore properties in the installation file. Refer to [Configuring Openet Weaver for HTTPS](#).
3. Place new certificates in the certs folder. Newer certificates overwrite old ones. The following is an example of the certs folder structure in ovlm-install:

```
ovlm-install
`--certs
  |-- key_store.jks
  |-- trust_store.jks
  '-- ca_bundle
```

4. Follow up this task by performing one of the following tasks:

- [Updating Configuration for all Microservices except Resource Agents](#)
- [Updating Configuration for Resource Agents using the Deployment Manager CLI](#)

Updating SSH Keys

To update your SSH keys, regenerate an SSH key pair and place the pair inside the ssh-keys folder. Follow the steps in [Configuring Rsync to Run in SSH Mode](#) to perform this task.

The following is an example of the ssh-keys directory structure inside ovlm-install. The private key and the public key must be named exactly as shown, in that order, in the example:

```
ovlm-install
`--ssh-keys
  |-- ovlmrsync_id_rsa
  '-- ovlmrsync_id_rsa.pub
```

- Follow up this task by performing one of the following tasks:

- [Updating Configuration for all Microservices except Resource Agents](#)
- [Updating Configuration for Resource Agents using the Deployment Manager CLI](#)

Updating Configuration for all Microservices except Resource Agents

When you are finished making configuration changes, which can include changes to the installation file, SSL certificates, and SSH keys, you can push the changes to all microservices except for Resource Agents by running the following command:

```
./ovlm-deploy.sh -i <install_properties_file> -u <installation_user>
[--skip-local-cli-install]
```

Note: If you have made changes on repository-root, you must move existing repository-root directory to new repository-root directory manually.

Updating Configuration for Resource Agents using the Deployment Manager CLI

When you are finished making configuration changes, which can include changes to the installation file, SSL certificates, and SSH keys, you can push the changes to all Resource Agents only by following these steps:

1. Log in to the authentication service using the following command:

```
eval $(on-auth-client -u <username> -p <password>)
```

2. If you made any configuration changes, upload the related files, for example, the installation YML file. by entering the following command:

```
ovlm-dm configuration create [-i<installation_file>] [-u <installation_user>]
[-secure_stage_traffic <true/false>] [-s <rsync_ssh_key_file>] [-e
<encryption_key_file>] [-c <certificate_file>] [-k <installer_ssh_key_file>]
```

Note: All the parameter in the above command are optional, you only use those that relate to your configuration changes.

3. Push the configuration to all Resource Agents using the following Deployment Manager CLI command:

```
ovlm-dm service install -a '[ "<node_1>", "<node_2>", "<node_3>",
"<node_4>" ]'
```

In the above command, the -a parameter is the list of Resource Agent nodes, which must be in JSON format.

Note: If you have made changes on repository-root, you must move existing repository-root directory to new repository-root directory manually.

Modifying Microservices

This section describes how to update, roll back, and reinstall microservices.

Upgrading all Microservices Except Resource Agents

This topic describes how to upgrade Openet Weaver. The upgrade affects all microservices except Resource Agents.

To upgrade REsource Agents see [Upgrading Resource Agents](#).

The upgrade steps assume that you are upgrading only, and not making any changes to your deployment. This prevents the need to perform many of the pre-installation tasks and post-installation tasks that need to be performed for an initial installation.

1. Transfer the Openet Weaver TAR file you received to the ovlm-install directory or the installer node

Note: You can use a separate machine to install the software provided it meets the software and hardware requirements.

2. Extract the TAR files on to the disk by entering the following command:

```
tar -xvf ovlm-core-install-enterprise-<release_version>.tar
tar -xvf ovlm-thirdparty-dependencies-<version_number>.tar
```

Note: Ensure that both files are in the same directory before extracting since they both extract to the same directory.

See the topic on extracting the TAR file for more information about the first two steps in this task.

3. Update the existing installation file so that there are no encrypted passwords in the file.

Passwords are entered in plain text in the installation file, then encrypted by running a script against that file. When the encryption script is run, a key file named installer.png is created and copied to every server in the topology file.

When you upgrade Openet Weaver with a new version, there is no installer.png file in the new ovlm-install directory. For this reason, you must generate a new key file. Before you generate the file, you need to revert the encrypted passwords in the existing installation file to plain text. Otherwise the encrypted passwords are re-encrypted and cause access issues when an attempt is made to use their text-based passwords.

For more information about the installation file see [Creating an Installation File](#).

4. Change directory to ovlm-install

```
cd ovlm-install
```

5. Encrypt the passwords in the installation file using the following command.

```
./utilities/cipher/encrypt_scripts/encrypt-password.sh -f <installation_file>
```

For more information about password encryption see [Encrypting Passwords](#).

6. Install the Openet Weaver upgrade using the ovlm-deploy.sh script.

Note: You must use the --force-flag when running the ovlm-deploy script to upgrade the existing installation of weaver.

When you run the installer against an existing deployment, the default behaviour is that Openet Weaver RPMs are installed only where the installer is attempting to install is a new version of an existing software package or where there is no existing RPM installed, for example when you are adding a node. If the RPMs being installed are the same version as what is already installed on a node, RPM installation for that node is skipped.

In other words, if you have an environment with an Openet Weaver deployment already installed and you execute a deployment of a newer version the old RPM packages are removed and the newer RPM packages are installed.

There might be times when you must reinstall the same version of an RPM on a node, for example in the event that a microservice must be reinstalled due to a corrupt configuration. In such a case you can use the --force-rpm-install command line argument to the ./ovlm-deploy.sh script to override the default behaviour. The argument instructs the deployment utility to remove any existing software packages and install the version in the current package.

Note: If you are installing Openet Weaver with anything other than the default directory for PATH_BIN that can be configured in the installation_paths section of the installation file, you need to copy the repository root from the old repository root to the new repository root in the location you specified. If the default location is used (/usr/lib64/ovlm) the repository root is copied automatically during upgrade. .

See [Creating an Installation File](#) for more information about configuring paths.

If you are planning on integrating keycloak security when you upgrade, refer to the following sections:

- [Installing and Configuring Keycloak](#)
- [Keycloak Security Post Installation Setup](#)

Related tasks

[Performing the Initial Installation](#) on page 130

Upgrading Resource Agents

This topic describes how to upgrade Resource Agents for Openet Weaver from the CLI or by using the VNF-M GUI.

The upgrade steps assume that you are upgrading only, and not making any changes to your deployment. This prevents the need to perform many of the pre-installation tasks and post-installation tasks that need to be performed for an initial installation.

Related concepts

[VNF-M GUI Pages](#) on page 156

Upgrading Resource Agents from the CLI

To upgrade Resource Agents using the CLI, follow these steps

1. Transfer the Openet Weaver TAR file you received to the ovlm-install directory or the installer node

Note: You can use a separate machine to install the software provided it meets the software and hardware requirements.

- Extract the TAR files on to the disk by entering the following command:

```
tar -xvf ovlm-core-install-enterprise-<release_version>.tar
tar -xvf ovlm-thirdparty-dependencies-<version_number>.tar
```

Note: Ensure that both files are in the same directory before extracting since they both extract to the same directory.

See the topic on extracting the TAR file for more information about the first two steps in this task.

- Change directory to ovlm-install

```
cd ovlm-install
```

- Prepare the files to be uploaded to Deployment Manager by entering the following command:

```
./ovlm-dm-util.sh prepare upload_files
```

- If security authentication(Keycloak) is enabled, log in to the authentication service by entering the following command:

```
eval $(on-auth-client -u <username> -p <password>)
```

Note: Skip this step if security authorization is not enabled.

- Upload Resource-Agent-related binary files to Deployment Manager by entering the following command:

```
ovlm-dm artifact upload -f <installation_file>
```

The <install_file> is the TAR file prepared in step 4. The following is an example of a response to uploading the file:

```
2017-10-06 15:08:33,910 - INFO - ovlm-dm-util - <module> - Log file for this
DM util: ovlm-dm-util-log-20171006-150833.log
2017-10-06 15:08:33,924 - INFO - ovlm-dm-util - pack_ovlm_installer - Packing
OVLM installer into single tarball...
2017-10-06 15:08:41,581 - INFO - ovlm-dm-util - pack_ovlm_installer - Completed
packing OVLM installer.
2017-10-06 15:08:41,581 - INFO - ovlm-dm-util - pack_ovlm_configuration -
Packing OVLM nstallation configuration...
2017-10-06 15:08:42,615 - INFO - ovlm-dm-util - pack_ovlm_configuration -
Completed packing OVLM installation configuration.
2017-10-06 15:08:42,615 - INFO - ovlm-dm-util - list_prepared_files - List
of prepared files for uploading to Deployment Manager
INSTALL_FILE location : /home/ovlminstall/testHome/working/ovlm-inst
all/artifact/ovlm-install.tar.gz
CERTIFICATE_FILE location : /home/ovlminstall/testHome/working/ovlm-inst
all/artifact/certs.tar.gz
RSYNC_SSH_KEY_FILE location : /home/ovlminstall/testHome/working/ovlm-inst
all/artifact/ssh-keys.tar.gz
ENCRYPTION_KEY_FILE location : /home/ovlminstall/testHome/working/ovlm-inst
all/artifact/.installer.png
```

- If you made any configuration changes, upload the related configuration details (for example, the installation YML file) by entering the following command:

```
ovlm-dm configuration create [-i<installation_file>] [-u <installation_user>]
[-secure_stage_traffic <true/false>] [-s <rsync_ssh_key_file>] [-e
<encryption_key_file>] [-c <certificate_file>] [-k <installer_ssh_key_file>]
```

Be aware of the following when you run the configuration create command:

- The `installer_ssh_key_file` must be located at `$HOME/.ssh/id_rsa`
- The `rsync_ssh_key_file` parameter can refer to the `RSYNC_SSH_FILE` location as in the example in step 6.
- The `encryption_key_file` parameter can refer to the `ENCRYPTION_FILE` location as in the example in step 6.
- The `certificate_file` parameter can refer to the `CERTIFICATE_FILE` location as in the example in step 6, and is required only when running Openet Weaver in HTTPS mode.

8. Upgrade the Resource Agent node from v1.9 to v1.10 by entering the following command:

```
ovlm-dm service install -a '[ "<node-1>", "<node-2>", "<node-3>", "<node-4>" ]' -r '{"force_install": "resource_agent"}'
```

In the above command, the `-a` parameter is the list of Resource Agent nodes, which must be in JSON format.

In the next release, when updating from v1.10 to later versions, you can run the following command to update all Resource Agent nodes:

```
ovlm-dm service upgrade -t <resource_agent> -v <weaver_version>
```

Alternatively, you can target specific Resource Agents for upgrade using the following command:

```
ovlm-dm service upgrade -t resource_agent -d '[ "<node-1>", "<node-2>", "<node-3>", "<node-4>" ]' -v <weaver_version>
```

Upgrading Resource Agents from the VNF-M GUI

In order to upgrade a Resource Agent using the VNF-M GUI, you must first perform steps 1 through 7 in Upgrading a Resource Agent from the CLI.

Important: Performing an upgrade using the VNF-M GUI will only upgrade the Resource Agent version. To make configuration changes in addition to upgrading Resource Agents you must upgrade using the CLI.

To upgrade Resource Agents from the VNF-M GUI, follow these steps:

1. Launch the VNF-M GUI. If securityauthorization is enabled, enter your credentials on the Log In page.
2. In the top-right corner of the VNF-M GUI, click the



icon and select **Update**.

The Update Resource Agent page displays.

Nodes
rt1.weaver-lan Version 1.9.4
rt2.weaver-lan Version 1.9.1
rt3.weaver-lan Version 1.9.2
rt4.weaver-lan Version 1.9.1_SM-45900_Request-7820

Figure 10: Update Resource Agent Page

3. Select one or more Resource Agent nodes by checking the check box of each node you are upgrading. Alternatively, check the **Nodes** check box to select all Resource Agent nodes.
4. Click the



(Upgrade) icon.

If you are upgrading one Resource Agent, click the icon on the Resource Agent node card. If you are upgrading multiple Resource Agents, click the icon on the top-right of the screen.

The **Upgrade microservices on** dialog box displays showing all nodes selected for upgrade.

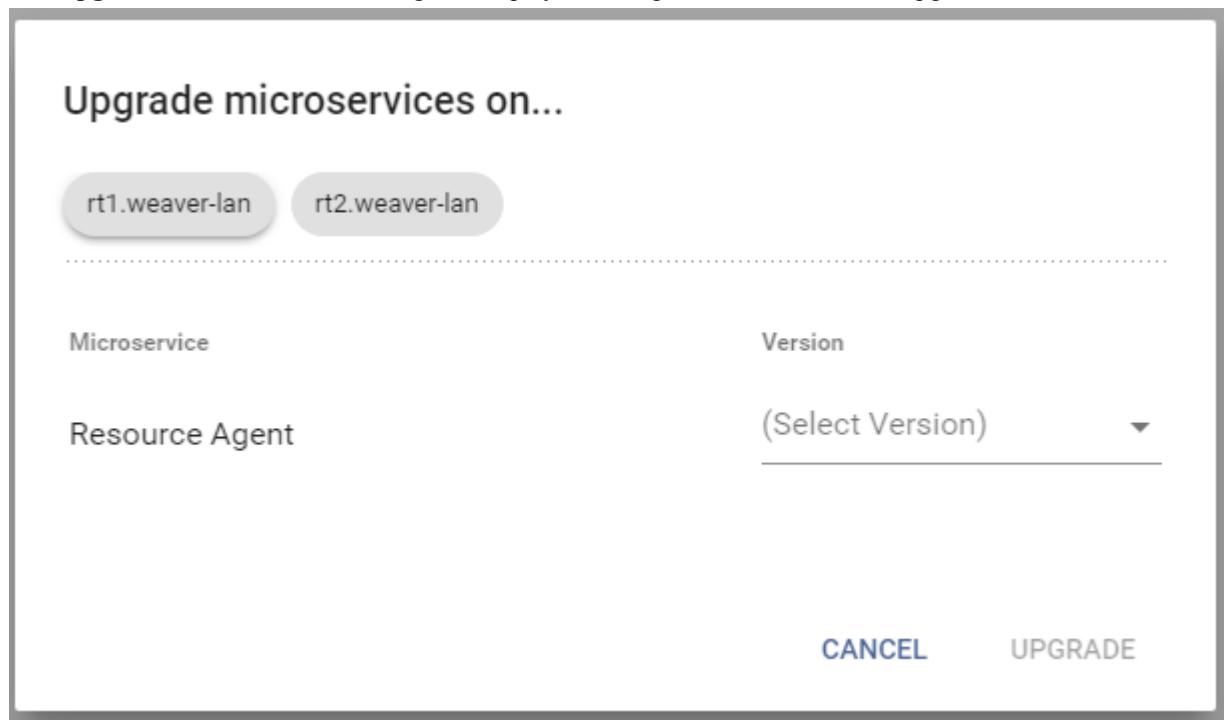


Figure 11: Upgrade microservices on Diaog Box

5. Click an upgrade version from the **Select Version** drop down list.

6. Click **Upgrade**.

The selected Resource Agents are upgraded and the dialog box closes.

Rolling Back Upgraded Resource Agents

This topic describes how to rollback upgraded Resource Agents to a previous version from the CLI or by using the VNF-M GUI. .

Note: In current release, only Resource Agents can be rolled back after upgrade.

Rolling Back Resource Agents from the CLI

To roll back Resource Agents from the CLI, follow these steps:

1. If security authentication(Keycloak) is enabled, log in to the authentication service by entering the following command:

```
eval $(on-auth-client -u <username> -p <password>)
```

Note: Skip this step if security authorization is not enabled.

2. Roll back all Resource Agents to the previous version by entering the following command

```
ovlm-dm service rollback_upgrade -t resource_agent
```

Alternatively, you can roll back specific Resource Agents using the following command:

```
ovlm-dm service rollback_upgrade -t resource_agent -d '[ "<node-1>", "<node-2>", "<node-3>", "<node-4>" ]'
```

Rolling Back Resource Agents Using the VNF-M GUI

To roll back Resource Agents using the VNF-M GUI, follow these steps:

1. Launch the VNF-M GUI. If security authorization is enabled, enter your credentials on the Log In page.
2. In the top-right corner of the VNF-M GUI, click the  icon and select **Update**.

The Update Resource Agent page displays.



Figure 12: Update Resource Agent Page

3. Select one or more Resource Agent nodes by checking the check box of each node you are rolling back. Alternatively, check the **Nodes** check box to select all Resource Agent nodes.
4. Click the 

(Rollback) icon.

If you are rolling back one Resource Agent, click the icon on the Resource Agent node card. If you are rolling back multiple Resource Agents, click the icon on the top-right of the screen.

The **Rollback microservices** dialog box displays showing all nodes selected for rollback.

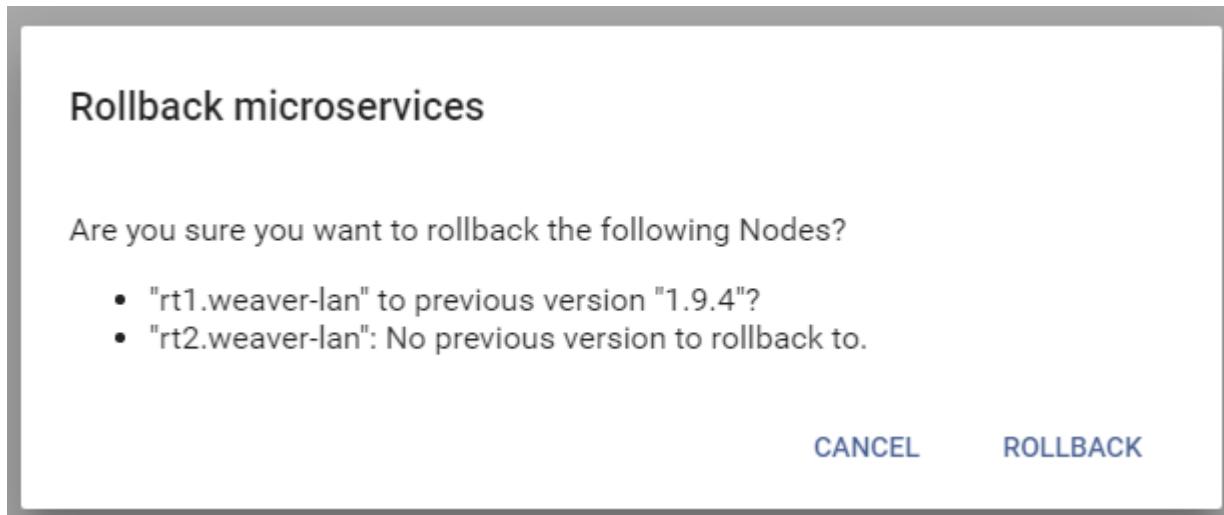


Figure 13: Rollback microservices Diaog Box

5. Click **Rollback**.
The selected Resource Agents are rolled back and the dialog box closes.

Reinstalling any Microservice Except Resource Agents

To reinstall one or more microservices except for Resource Agents, follow these steps:

1. Change directory to ovlm-install

```
cd ovlm-install
```

2. Reinstall specified microservices by entering the following command

```
./ovlm-deploy.sh -i install_properties_file -u --force-install <microservice_1>
<microservice_2>
```

You can specify one or more of the following microservices using the force parameter:

- configuration_manager
- deployment_manager
- resource_manager
- workflow_engine
- frontend
- vnfm_gui
- vim_abstraction_layer
- performance_manager
- instance_inventory_manager
- fault_manager
- influxdb
- grafana
- keepalived
- auth_server

For example, the following command reinstalls all instances of Configuration Manager and Resource Manager in a Deployment

```
./ovlm-deploy.sh -i install.yml -u ovlm_user --force-install
configuration_manager resource_manager
```

Note: If you make changes to the configuration file, the changes are pushed to all other microservice except Resource Agents. However, the new configuration won't take effect on other microservice until those microservices are restarted.

Reinstalling Resource Agents

To reinstall Resource Agents, follow these steps:

1. If security authentication(Keycloak) is enabled, log in to the authentication service by entering the following command:

```
eval $(on-auth-client -u <username> -p <password>)
```

Note: Skip this step if security authorization is not enabled.

2. Reinstall specified Resource Agents by entering the following command

```
ovlm-dm service reinstall -t resource_agent -d '[ "<node-1>", "<node-2>",
"<node-3>", "<node-4>" ]'
```

Note: -d represents a list of Resource Agent host nodes. The list must be in JSON format.

Reinstalling Content Packs

You can reinstall content packs to Workflow Engine using the following command:

```
./ovlm-deploy.sh -u root -i topology_install.yml --install-cp all
```

Updating Out of the Box Procedures

You can update the customized out-of-the-box procedure add-on in Configuration Manager's repository using the following command:

```
./ovlm-deploy.sh -u root -i topology_install.yml --ootb-procedures-addon <OOTB_procedure_addon_tar_file>
```

Granting and Revoking OVLM User Sudo Privileges

You can grant and revoke sudo privileges for the ovlm user as required by updating the installation file then running the installation script. You can also opt to not specify sudo privileges in the installation file, in which case it is assumed that privileges are granted as required by the product.

Use the `sudo_privileges_required` and `sudo_privileges` parameters in the `resource_agent` section of the installation file to specify privileges for the ovlm user. When you do so, Openet Weaver grants access for only those commands listed under sudo privileges when you run the installation script.

The parameters are described in the following table.

Table 25: resource_agent Sudo Privileges parameters

Parameter Name	Default Value	Description
<code>sudo_privileges_required</code>	true / false	Whether sudo_privileges defined in the installation file are used to configure sudo privileges for the ovlm user. When the default value of false is used, all privileges listed under sudo_privileges are ignored.
<code>sudo_privileges</code>	N/A	The list of commands that are sudo privileges for the ovlm user. This section impacts the ovlm user only when <code>sudo_privileges_required</code> is set to true.

Granting sudo privileges for the OVLM User

The following is an example of commands specified for the ovlm in the `resource_agent` section of the installation file. The `sudo_privileges_required` parameter is set to true. The commands the ovlm user has sudo privileges for are listed under `sudo_privileges`.

```
resource_agent:
  hosts:
    - node-1
    - node-2
    - node-3
    - node-4
  properties:
    server:
      port: 28030
    sudo_privileges_required: true
    sudo_privileges:
      - /bin/postgresql-setup
      - /bin/rpm
```

```

- /usr/bin/yum
- /bin/bash
- /bin/sudo *
- /usr/bin/sudo *
- /usr/bin/systemctl status *
- /usr/bin/systemctl start *
- /usr/bin/systemctl stop *
- /usr/bin/systemctl restart *
- /usr/bin/systemctl is-active *
- /usr/bin/systemctl daemon-reload
- /sbin/useradd
- /sbin/userdel
- /sbin/usermod
- /bin/mkdir
- /bin/rm

```

Revoking sudo Privileges for the ovlm User

The following is an example of the sudo privileges commands being used to revoke the ovlm user's sudo privileges. Notice that the sudo_privileges_required parameter is set to true. The sudo_privileges parameter contains an empty list. The effect is that the ovlm user is assigned no sudo privileges.

```

resource_agent:
  hosts:
    - node-1
    - node-2
    - node-3
    - node-4
  properties:
    server:
      port: 28030
    sudo_privileges_required: true
    sudo_privileges: []

```

Not Specifying sudo Privileges for the ovlm user

You have the option of not specifying sudo privileges for the ovlm user, in which case Openet Weaver assumes that you will grant all sudo privileges as required. To do this set sudo_privileges_required to false. This means that all contents of the sudo_privileges list are ignored.

Note: This method is useful for removing privileges temporarily without deleting the list of sudo privileges.

```

resource_agent:
  hosts:
    - node-1
    - node-2
    - node-3
    - node-4
  properties:
    server:
      port: 28030

    sudo_privileges_required: false
    sudo_privileges:
      - /usr/bin/yum
      - /bin/bash

```

Note: You can achieve the same effect as above by not specifying any sudo privileges parameters as in the following example:

```
resource_agent:
  hosts:
    - node-1
    - node-2
    - node-3
    - node-4
  properties:
    server:
      port: 28030
```

If you do not specify sudo privileges in the installation file, then later decide to use Openet Weaver to clean up and grant sudo privileges for the ovlm user, we recommend that you use the command shown below to add the list of commands to granted sudo access in the /etc/sudoers file of each Resource Agent node in addition to specifying these commands in the installation file. The command format is as follows:

```
Cmnd_Alias OVLM_USER_DEFINED_CMD = <comma seperated list of commands to be
used by "ovlm" user>
ovlm  ALL=(ALL) NOPASSWD: OVLM_USER_DEFINED_CMD
```

Updating sudo Privileges for Resource Agents

To update sudo privileges for Resource Agents, follow these steps

1. If security authentication(Keycloak) is enabled, log in to the authentication service by entering the following command:

```
eval $(on-auth-client -u <username> -p <password>)
```

Note: Skip this step if security authorization is not enabled.

2. Upload the installation YML file by entering the following command:

```
ovlm-dm configuration create -i <installation_file>
```

4. Update the sudo privileges of Resource Agent nodes by entering the following command:

```
ovlm-dm service install -a '[ "<node-1>", "<node-2>", "<node-3>", "<node-4>" ]'
```

Scaling Deployments

You can scale out your Openet Weaver deployment as required using the following methods:

- Triggering the VNF instantiation without VIM workflow
- Adding a new Resource Agent node using Deployment Manager

Related tasks

[Creating an Installation File](#) on page 92

[Performing the Initial Installation](#) on page 130

Scaling Out by Triggering a VNF Instantiation without VIM Workflow

To scale out Resource Agents by triggering a VNF instantiation without VIM workflow, follow these steps:

1. If security authentication(Keycloak) is enabled, log in to the authentication service by entering the following command:

```
eval $(on-auth-client -u <username> -p <password>)
```

Note: Skip this step if security authorization is not enabled.

- Download VNF topology file from the Openet Weaver management node by entering the following command:

```
ovlm-cm topology download -t <tenant_id> -v <vnf_id> -i <vnf_instance_id> -f vnf_topology.yml
```

- Update the VNF topology file by adding a block for a new Resource Agents under a VNFC block.
See [Creating an Installation File](#) for more information.
- Upload the updated VNF topology file back to the Openet Weaver management node by entering the following command:

```
ovlm-cm topology upload -t <tenant_id> -v <vnf_id> -i <vnf_instance_id> -f vnf_topology.yml
```

- Trigger a VNF Instantiation without VIM workflow in the Openet Weaver management node using the following command:

```
ovlm-fe workflow submit -t <tenant_id> -v <vnf_id> -i <vnf_instance_id> -y instantiate_vnf_without_vim -p <vnf_template_id>
```

Scaling Out by Adding a Resource Agent Using Deployment Manager

To scale out Resource Agents by adding a Resource Agent using Deployment Manager, follow these steps:

- If security authentication(Keycloak) is enabled, log in to the authentication service by entering the following command:

```
eval $(on-auth-client -u <username> -p <password>)
```

Note: Skip this step if security authorization is not enabled.

- Install a new Resource Agent node by entering the following command:

```
ovlm-dm service install -a '[ "<node-5>" ]'
```

Restarting Microservices

You can restart microservices after post-installation configuration changes are made to Openet Weaver microservices or other system services.

To restart a microservice enter the following command:

```
./ovlm-deploy.sh -restart <microservice> -u root -i topology_install.yml
```

Openet Weaver VNF-M GUI

Openet Weaver comes with a VNF-M graphical user interface (GUI) through which you can perform VNF management tasks. The GUI consists of the following tabbed pages:

- VNF Catalog
- VNF Lifecycle

The VNF-M GUI supports the following web browsers:

- Microsoft Internet Explorer 11
- Mozilla Firefox 55
- Google Chrome 61

Related concepts

[Creating VNFs](#) on page 173

Using the Out of the Box procedures and flows, create a VNF package and deploy it.

[Managing VNFs](#) on page 243

Logging into the VNF-M GUI

The VNF-M GUI is located at `<protocol>://<vnfm_gui_host>:<vnfm_gui_server_port>`. The actual host and port number are specified in the vnfm_gui section of the Openet Weaver installation file. If authentication is enabled, when you connect to the main GUI URL or to a specific page (for example `http://<vnfm_gui_host>:<vnfm_gui_server_port>/#/vnf-lifecycle`), you are redirected to the Login page as shown.

Note: The URL changes to that of the authorization service. You are directed back to the VNF-M GUI upon successful login.

Note: If authentication is not enabled the url will open the VNF-M GUI Catalog Page. For more information about enabling authentication see [Creating an Installation File](#)

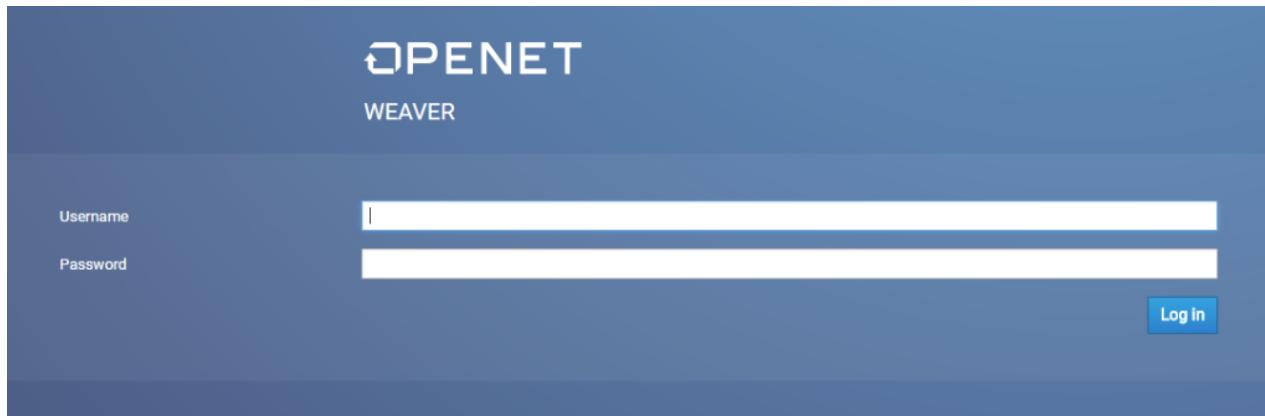


Figure 14: VNF-M Login Page

To log in to the Openet Weaver VNF-M GUI, follow these steps:

1. In the **Username** field enter a valid username.
2. In the **Password** field enter the password for the username.

Note: Usernames and passwords are configured in Keycloak. See the Keycloak documentation for more information.

3. Click **LOG IN**.

You are directed to the VNF-M GUI homepage.

Automatic Log Out

After you log in, the VNF-M GUI is configured to automatically log you out when you let it sit idle for a set amount of time that is configured in the installation file. See [Creating an Installation File](#) for more information about setting the timeout value.

You will also be logged out if an action you take in the GUI triggers an API request where the response is an HTTP status of 401. This can happen when the authorization token retrieved during log in is no longer valid.

In either case you are redirected to the Log In page.

VNF-M GUI Pages

The Openet Weaver VNF-M GUI consists of the following pages:

- Catalog page
- Lifecycle page, which includes the following subpages:
 - VNFC page
 - VNFC Instance page
- Update Resource Agent page

Catalog Page

When you open the GUI, the VNF Catalog page is open by default as shown.

Name	Description	Creation Date	Actions
microblog	A VNF implementing a 3 tier microblog platform	Mar 1, 2017 9:18:38 AM	
microblog_2	A VNF implementing a 3 tier microblog platform	Feb 7, 2017 7:21:52 AM	
microblog_3	A VNF implementing a 3 tier microblog platform	Mar 1, 2017 9:18:21 AM	
microblog_4	A VNF implementing a 3 tier microblog platform	Mar 1, 2017 9:18:39 AM	

Figure 15: Openet Weaver VNF-M GUI VNF Catalog Page

The VNF Catalog page can be used to perform the following tasks:

- Viewing VNFs
- Viewing packages
- Importing packages
- Exporting packages
- Deleting packages
- Creating VNFs

Lifecycle Page

Clicking the **VNF Lifecycle** tab opens that page.

Figure 16: VNF Lifecycle Page

Each VNF instance known to Openet Weaver is represented by a VNF instance card as shown in the following example.

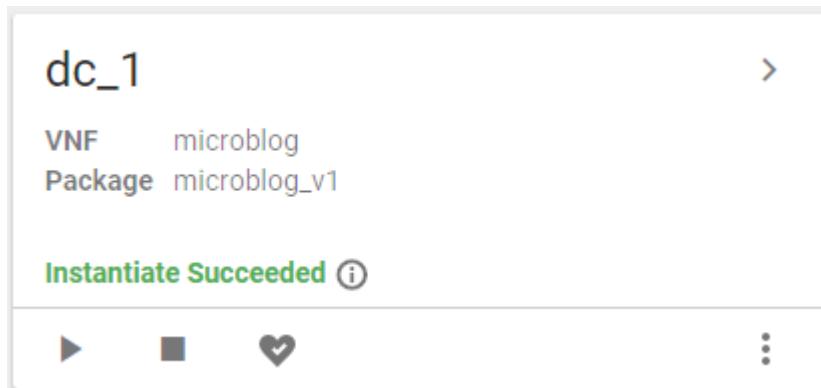


Figure 17: VNF Instance Card

Each VNF card lists the following information in the order shown:

- VNF instance ID
- VNF ID
- VNF package ID
- VNFD ID
- Status

Note: VNFD ID does not appear on VNFs that are not configured for a Virtualized Infrastructure Manager (VIM).

The following table describes the icons that appear on the VNF cards depending on the state they are in.

Table 26: VNF Instance Card Workflow Icons

Icon	Description
	Runs the Instantiate workflow.

Icon	Description
	Runs the Start workflow.
	Runs the Stop workflow.
	Runs the is VNF up workflow.
	Displays a menu with more workflow options.
	Displays workflow status information in a message box. This button is available only when a workflow was invoked previously. Note: Workflow status can also be opened by clicking See Details in the message that displays when a workflow either completes it's run successfully or fails.
	Navigates to the VNFC page.

Cards are arranged on the VNF Lifecycle page in two groups:

- Alerts
- Operating normally

Within those groups cards are sorted by time of last status change, with most recent first.

VNF Status

The status of each VNF instance is indicated by a color code as well as text as described in the following table.

Table 27: VNF Instance Status

Status Message	Status Text Color	Description
<workflow> Failed	Orange	A VNF instance where the previous workflow run against it failed.
Inactive	Black	A VNF instance that has not been instantiated.
Terminated	Black	A VNF instance that was terminated.
<workflow> In-Progress	Blue	A VNF instance that is running and has a workflow currently running against it.
<workflow> Succeeded	Green	An active VNF where the last workflow ran successfully.

All in-progress VNF cards are polled when you first open the Lifecycle page. Polling continues until each workflow succeeds or fails, then the status is updated on the card. The polling interval for in-progress VNF instances is every 15 to 20 seconds. The polling interval for workflows run in the GUI is every five seconds.

At the top right of the screen there are status summary buttons that report on the number of VNF instances in a given state. The buttons can also be used for filtering. When you click on a button, only those VNF instances in that state are displayed. The relationship of buttons to states is described in the following table. You can revert to display all VNF cards by clicking the same button you used to filter or clicking the X button.

Table 28: Status Summary Filter Buttons

Button	State
Orange	Status: failed
Blue	Status: In-Progress
Grey	status: Inactive/terminated
White	Status: successful.

VNFC Page

There is an arrow



in the top-right corner of each active VNF instance card. Clicking that arrow brings you to a page that displays a card for each VNFC in the selected VNF instance as in the example shown below.

The screenshot shows a page titled "microblog - dc_1". Below the title, it says "default > microblog - dc_1". There are three cards displayed side-by-side: "blogserver", "nginx", and "postgres". Each card has an arrow icon in its top-right corner. The "blogserver" card has a green message "Upgrade Succeeded ⓘ". Each card has three small icons below it: a play button, a stop button, and a menu icon.

Figure 18: VNFC Page

Running a workflow from a VNFC card applies to each instance of that VNFC . So for example, if you click the Start icon on the blogserver card, all instances of blogserver are started. Running an action against a VNFC, a status message is displayed similar to the ones for VNFs. In the above example, a successful upgrade was run against all instances of the blogserver VNFC.

The following table describes the icons that appear on the VNFC cards depending on the state they are in

Table 29: VNFC Instance Card Workflow Icons

Icon	Description
▶	Runs the Start workflow if the VNFC is down.
■	Runs the Stop workflow.
⋮	Displays a menu with more workflow options.
>	Navigates to the VNFC Instances page.

Icon	Description
	<p>Displays workflow status information in a message box. The following is an example:</p> <div style="border: 1px solid black; padding: 10px;"> <p>Workflow Status</p> <pre>workflow_instance_status: COMPLETED flow_result: health_status: UP vnfc_instances: - vnfc_instance_id: '1' hostname: node-2 vnfc_id: blogserver exit_code: 0 response: health_status: UP - vnfc_instance_id: '2' vnfc_id: blogserver hostname: node-3 exit_code: 0 response: health_status: UP - vnfc_instance_id: '1' hostname: node-1 vnfc_id: nginx exit_code: 0 response: health_status: UP - vnfc_instance_id: '1' vnfc_id: postgres</pre> <p style="text-align: right;">CLOSE</p> </div>

You can click the back arrow in the top-right corner of a VNFC card to display all instances of that VNFC. The same workflow icons as those on the VNFC-type cards are displayed. on the VNFC instance cards.

Alternatively, from the VNFC page you can toggle to display all VNFC instances in the VNF by clicking the toggle



in the top-right corner of the page as shown below.

The screenshot shows the VNFC Instances page for the 'microblog - dc_1' service. At the top, there's a back arrow and the title 'microblog - dc_1'. Below that, it says 'default > microblog - dc_1'. The page is organized into sections by VNFC type:

- blogserver VNFC Instances:**
 - blogserver 1:** Package microblog_v1, Health Check: Running (info icon). Buttons: play, stop, more.
 - blogserver 2:** Package microblog_v1, Health Check: Running (info icon). Buttons: play, stop, more.
- nginx VNFC Instances:**
 - nginx 1:** Package microblog_v1, Health Check: Running (info icon). Buttons: play, stop, more.
- postgres VNFC Instances:**
 - postgres 1:** Package microblog_v1, Health Check: Running (info icon). Buttons: play, stop, more.

Figure 19: VNFC Instance Page—VNFC Instances

Opening the VNFC Instances page runs the `is_vnf_up` workflow, which results in a status message appearing on each VNFC instance card.

When all VNFC instances are displayed, they are grouped by VNFC. In this example the VNFCs are:

- blogserver
- nginx
- postgres

Clicking the arrow at the top-left of the VNFC Instance page returns you to the lifecycle page.

Update Resource Agent Page

You open the Update Resource Agent page from either the Catalog page or the Lifecycle page by clicking the



icon at the top-right of the page and selecting **Update**.

Figure 20: Update Resource Agent Page

The page displays a card for each node that has a Resource Agent microservice installed. Each card shows the host identifier, which can be a hostname or an IP address, and the Resource Agent version number. Cards are sorted lexicographically so that Resource Agent nodes with hostname identifiers display alphabetically before Resource Agent nodes with IP identifiers.

You can filter Resource Agent nodes by using the [search function](#).

The following table describes the icons that appear on the Resource Agent node cards.

Table 30: Microservice Action Icons

Icon	Description
	Opens the Upgrade microservices on dialog box.
	Opens the Rollback microservices dialog box.
	Select Resource Agent node check box.

Unlike the other pages, the Update Resource Agent page allows you to perform actions on multiple Resource Agent nodes simultaneously. You select multiple nodes by checking two or more Resource Agent node check boxes. In the example below, rt1.weaver-lan and rt2.weaver-lan are selected.

Figure 21: Multiple Resource Agent Node Selection

When you clear a check box by clicking it again it is no longer selected.

Upgrade and Rollback icons appear at the top-right of the page when you select multiple Resource Agent nodes, and actions performed by clicking those icons apply to all selected nodes.

Note: The Upgrade and Rollback icons on the individual cards are disabled when multiple cards are selected.

You can also select all nodes by checking the Nodes checkbox on the left above the first Resource Agent node card.

Clicking the back arrow to the top left of the page returns you to the page from which you launched the Update Resource Agent Page. The arrow appears only when no Resource Agent node cards are selected. Otherwise, the number of cards selected appears at the top-left.

Related tasks

[Rolling Back Upgraded Resource Agents](#) on page 148

[Upgrading Resource Agents](#) on page 145

Selecting a Tenant

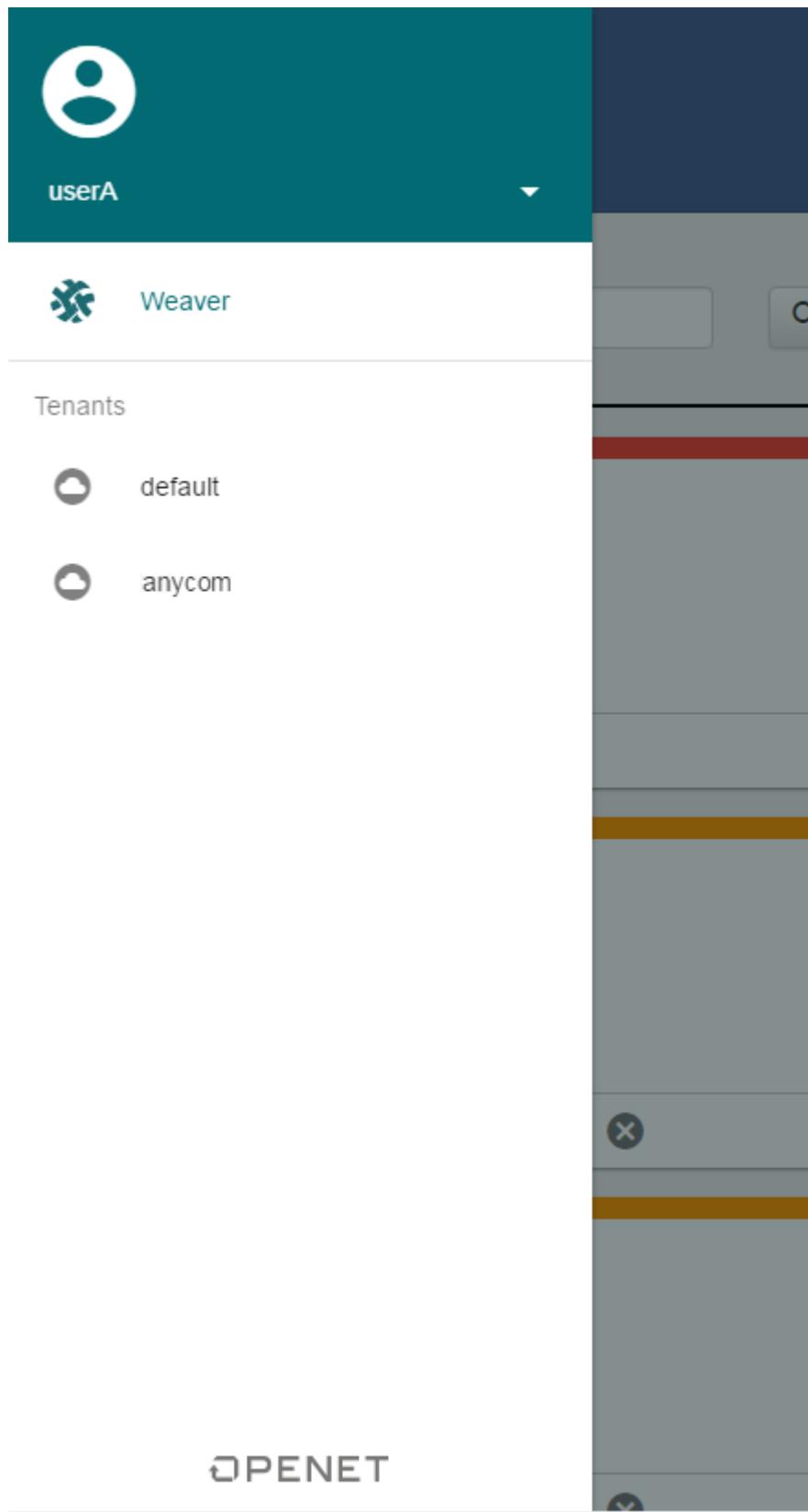
The GUI displays VNF information relevant to the currently selected tenant. To select a tenant, follow these steps:

1. Click the



icon to open the Navigation menu.

The navigation menu displays as shown.



The currently selected tenant is highlighted.

2. Click a tenant to select it.

The Navigation menu closes and the screen it was called from (Catalog or Lifecycle) displays with VNF information specific to the selected tenant.

Using the Search Function

The Openet Weaver VNF-M GUI provides search functionality on most pages. For example, you can search VNFs, VNF instances, or Resource Agent nodes depending on which page you are using. On the **VNF Catalog** page the **VNF Name** search box allows you to search for VNFs. On the **VNF Lifecycle** page the VNF Instance search box allows you to search for VNF instances. Both search boxes work the same way.

To search for a VNF on the VNF Catalog page, follow these steps.

1. Click the search icon



The search field displays.

Name	Description	Creation Date	Actions
microblog	A VNF implementing a 3 tier microblog platform	Mar 1, 2017 9:18:38 AM	
microblog_2	A VNF implementing a 3 tier microblog platform	Feb 7, 2017 7:21:52 AM	
microblog_3	A VNF implementing a 3 tier microblog platform	Mar 1, 2017 9:18:21 AM	
microblog_4	A VNF implementing a 3 tier microblog platform	Mar 1, 2017 9:18:39 AM	

Figure 22: Catalog Page with Search Field

2. Enter a string in the search field

The displayed VNFs are filtered dynamically as you enter the string. The string can be the complete name of the VNF or VNF instance, a partial name, or a string using one or more wildcards. For example, you can filter the VNFs so that only microblog_2 displays by typing any of the following strings into the search field:

- microblog_2
- *2
- m*2
- micro*_2
- m*blog*2

Identifying Mandatory Fields

When you enter data in the Openet Weaver VNF-M GUI be aware that some fields are mandatory. Mandatory fields are indicated by an asterisk (*) to the right of the field name as shown. Validation is performed on incorrect data.

Create VNF Instance

VNF*

Name *

Description *

CANCEL CREATE

Figure 23: Mandatory Fields in a Dialog Box

Checking system Health Status

The System Health Check information box displays the status of the Openet Weaver microservices. To open the information box click the



icon at the top-right of the VNF-M GUI and select **Check system health**.

An example of the system health check information box is shown below.

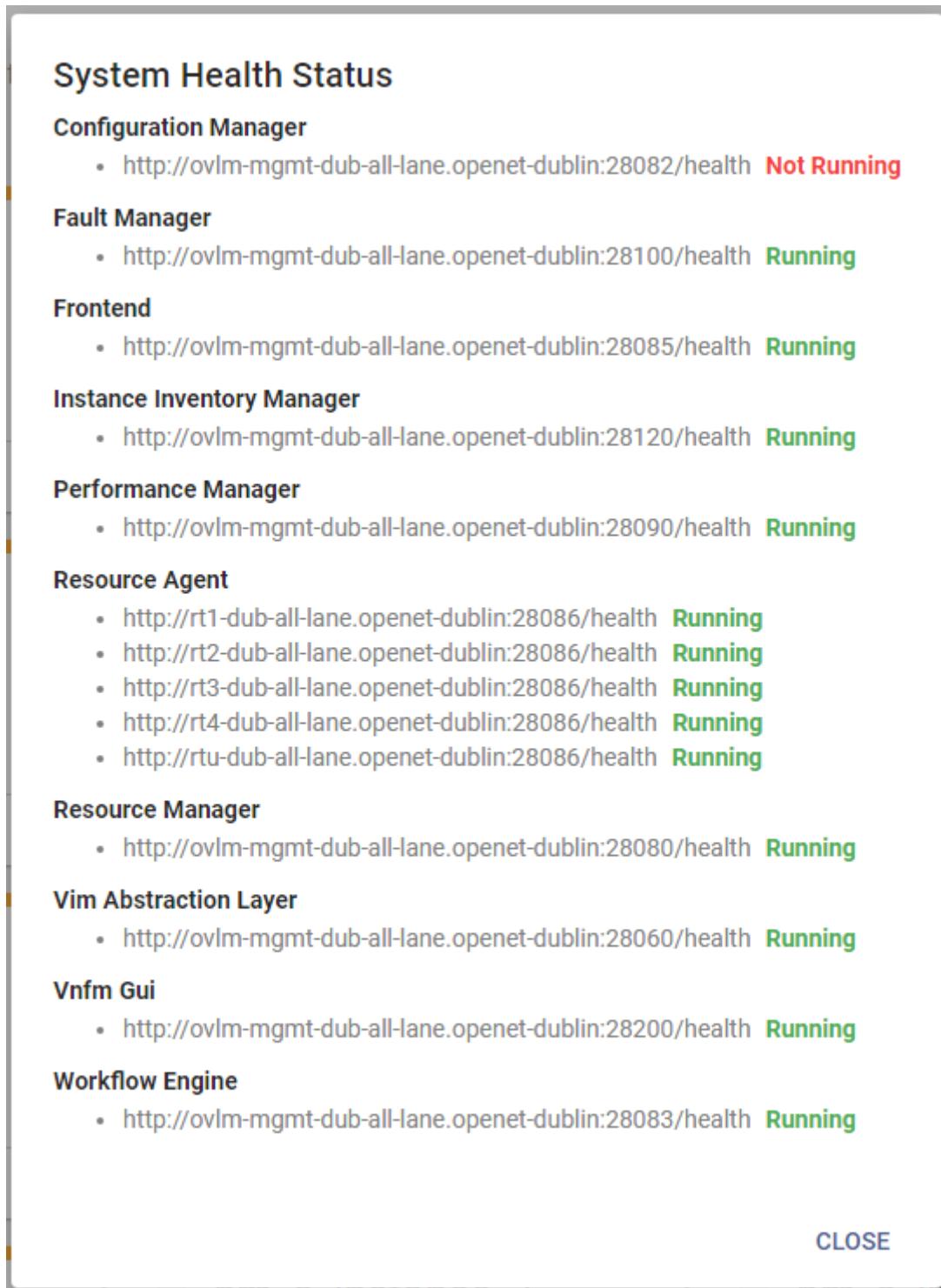


Figure 24: System Health Check Information Box

All running microservices show a status of running in green text. When a microservice is not running the status is shown in red text. In the example shown Configuration Manager is not running.

Logging Out of the VNF-M GUI

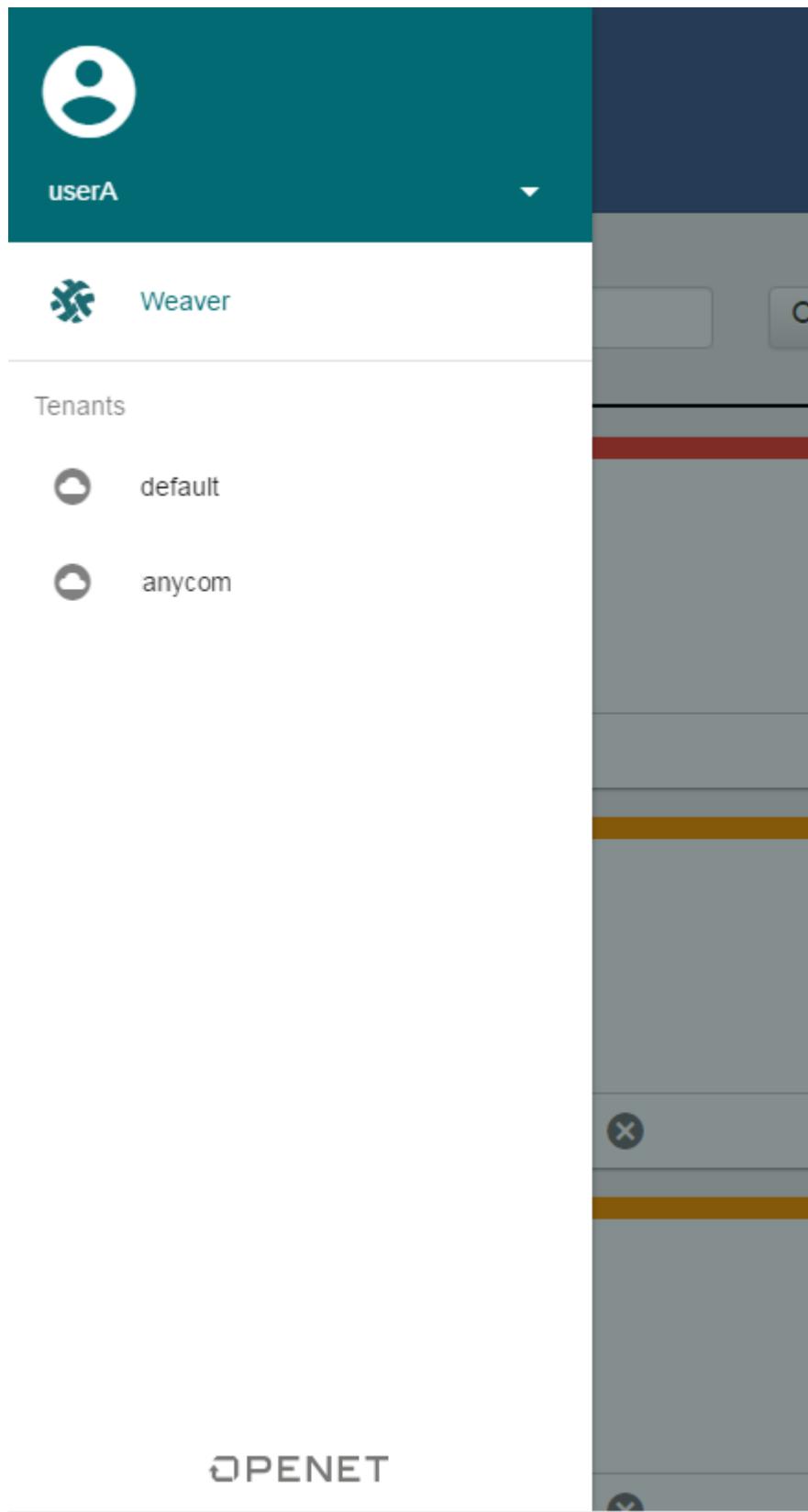
To log out of the Openet Weaver VNF-M GUI, follow these steps.

1. Click the



icon to open the Navigation menu.

The Navigation menu displays as shown.



The currently logged in user (userA in this example) is displayed at the top of the menu.

2. Click the



icon.

3. Click **Sign Out**.

You are directed to the Login page.

Using Openet Weaver in a Secure Environment

This topic describes additional tasks you must perform when you are using Openet Weaver with Keycloak and security is enabled. Specifically, you must:

- Log in to the secure environment
- Get security tokens that must be used in API calls
- Include the tokens in REST API calls

Important: The REST API call examples used elsewhere in this documentation *do not* include tokens. You must add the tokens to the API commands when using those examples in a secure environment

Enabling CLI Calls in a Secure environment

As a system user, you must execute the `on-auth-client` CLI command whenever you start an Openet Weaver session in a secure environment in order to be able to use CLI commands. The `on-auth-client` takes in user credentials and generates a session-based token. This token is passed automatically to all CLIs, so you can run all the CLI commands as specified elsewhere in the documentation during the session.

Before using the `on-auth-client` command, you must ensure the following requirements are met:

- A user was created for you in the Weaver Realm in Keycloak.
- The `weaver.profile` has been sourced using the previously-created user in [Keycloak Security Post Installation setup](#).

Assuming the requirements are met, execute the command as follows:

```
eval $(on-auth-client -u <username> -p <password>)
```

The following example shows the command being used by a user created in Keycloak named `alice` with the password "weaver":

```
eval $(on-auth-client -u alice -p weaver)
```

To verify that a token was created, execute the following command:

```
env | grep ON_AUTH_TOKEN
```

If login was successful the token is printed on screen.

Enabling REST API Calls in a Secure Environment

In order to make API calls in a secure environment you need to retrieve a token that is included in the calls.

Getting a Token

You use the password grant to retrieve a token for use in REST API calls. Details are in the following table:

Item	Description
URL	The token endpoint, for example <code>http://localhost:8080/auth/realms/Weaver/protocol/openid-connect/token</code>)

Item	Description
Method	POST
Headers	content-type: application/x-www-form-urlencoded
Body	grant_type=password&client_id=weaver-ext&scope=openid&username=<username>&password=<password>

Assuming that there is an authorization server URL in the environment, you export a token using the following:

```
export ON_OIDC_TOKEN_URL=http://10.0.8.85:8686/auth/realms/weaver/protocol/openid-connect/token
```

The following is an example of a curl command to export a token:

```
curl -X POST \
$ON_OIDC_TOKEN_URL \
-H 'Content-Type: application/x-www-form-urlencoded' \
-d 'grant_type=password&client_id=weaver-ext&scope=openid&username=dsmyth&password=123openet'
```

The following is an example of a successful response to the command:

```
{
  "access_token": "eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSlldUIiwia21kIIa6ICJNQ21MOTJzTjYta0NXR0txUmpPYk95R29Dc3hwN2JaeVNHCU1qeXVnQVFNI n0.eyJqdGkiOiI1MjI3OTVmZS03MWU2LTQwZjgtYjNhMy1hMmQzODUyNDFiOGMiLCJ leHAIoJE00Tk5NjM5MDUsIm5iZiI6MCviaWF0IjoxNDk5OTYzNjA1LCJpc3MiOjodHRwOi8vMTAuMC44Ljg1Ojg2ODYvYXXV0aC9yZWFBsXMuV2VhdmVyIiwiYXVkJoiid2VhdmVyLWNsaWVudCIsInN1YiI6IjQ1MzFkMzI0LWJKyjctNDAxOS04NDYwLWY1Mzc1NmQ4ODlmYiIsInR5cCI6IkJ1YXJlciIsImF6cCI6IndLYXZlci1jbG11bnQiLCJhdXR oX3RpBWUiOjAsInN1c3Npb25fc3RhGUioiI4OGYwYzI5MC11ZjVjLTQwZjItYmE5NS0zMzNkYmJ1Mjc4NjUiLCJhY3IiOiiXiiwiY2xpZW50X3N1c3Npb24iOiiI5YTAxNDMxMy03Y2MzLTQ1YzAtOTg1Ny02YTU3MGR1NTFiZWIiLCJhbGxvd2VklW9yaWdpbnMiO1tdLCJyZWFsbV9hY2N1c3MiOnsicm9sZXMiolsib2ZmbGluZV9hY2N1c3MiLCJ1bWFfYXXV0aG9yaXphdGlvbiJdfSwicmVzb3VyY2VfYWNjZXNzIjp7ImFjY291bnQiOnsicm9sZXMiolsibWFuYWd1LWFjY291bnQiLCJtYW5hZ2UtYWNjb3VudC1saW5rcyIsInZpZXctcHJvZmlsZSJdfX0sIm5hbWUiOiiIiLCJwcmVmZXJyZWRfdXN1cm5hbWUiOjib2IifQ.FuFm5ra-C4Pclk_CDOI3q04mEmkxz2dV-Vvo6Ww3vTWHJj5cfkRY1X3QpMbZNNv9IWX8cLyRKZrVQSCMH64BFH7uXXBgRakEjPx8qEsM4zPzVZADk0c5os0jQtgQdT hDPRI68FK-jkETM9-icLtAvsOMCAhvNF-F1QtmNBirkCxOIMuvTN3nI80ocfuCGf9ZScE80AwumFPhEN6jvtT3oDIjhI-vz83c1SMXGKRKh73IFo7Z9eGCoCfSB-WHT3b7ZyXa-igUapr9bCk-j4zamMEw0SzDDh2SyR0C6qvBkdWWEA6F1vFnnav5KoOVW7uNqvf04GehB8cPYegYXv1yXlg",
  "expires_in": 300,
  "refresh_expires_in": 0,
  "refresh_token": "eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSlldUIiwia21kIIa6ICJNQ21MOTJzTjYta0NXR0txUmpPYk95R29Dc3hwN2JaeVNHCU1qeXVnQVFNI n0.eyJqdGkiOiI4OWUxYjYxZi03MTM3LTQ1ZTQtODMzOS01MmYyM2YxNWZkMDgiLCJ leHAIoJAsIm5iZiI6MCviaWF0IjoxNDk5OTYzNjA1LCJpc3MiOjodHRwOi8vMTAuMC44Ljg1Ojg2ODYvYXXV0aC9yZWFBsXMuV2VhdmVyIiwiYXVkJoiid2VhdmVyLWNsaWVudCIsInN1YiI6IjQ1MzFkMzI0LWJKyjctNDAxOS04NDYwLWY1Mzc1NmQ4ODlmYiIsInR5cCI6Ik9mZmxpbmUiLCJhenAiOij3ZWF2ZXItY2xpZW50IiwiYXV0aF90aW1Ijo wLCJzzXNzaW9uX3N0YXR1IjoiODhmMGMyOTAtZWY1Yy00MGYyLWJhOTUTMzZGJizTI3ODY1IiwiY2xpZW50X3N1c3Npb24iOii5YTAxNDMxMy03Y2MzLTQ1YzAtOTg1Ny02YTU3MGR1NTFiZWIiLCJyZWFsbV9hY2N1c3MiOnsicm9sZXMiolsib2ZmbGluZV9hY2N1c3MiLCJ1bWFFYXV0aG9yaXphdGlvbiJdfSwicmVzb3VyY2VfYWNjZXNzIjp7ImFjY291bnQiOnsicm9sZXMiolsibWFuYWd1LWFjY291bnQiLCJtYW5hZ2UtYWNjb3VudC1saW5rcyIsInZpZXctcHJvZmlsZSJdfX19.BhjVA53xfNDBLNdfZFh0WX-wXSysmHzzasfJH4WFi7YxPYVimKVXVNpPWBHCeHAnrcc7RCrqYuGghtjENHgoCQWQ0FEPTWlnI76oBgQsZpV32VMDI573QxR815UEDZfDMGSHiuApYsAeAj7ECiQxv_lecX7atEleRo"
```

```

dGuhI9e0wXCDpB8xoP1fbcp1-_JmOEvItZj1kmeRDzps1X9nxPAdrCXyP03rYPD1L4
9n3s6XOzNPGsGMoaz8dwTwRuEvWk525W54CkyA9gPFzBeA-rAXBzcBsT8kj4g3BHd9
bAbSnHoZHRRG51_VFsW9BE5EvP0T4x-OpQ24sIvA5_Enyl0g",
    "token_type": "bearer",
    "id_token": "eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSlldUIiwia21
kIIa6ICJNQ21MOTJzTjYta0NXR0txUmpPYk95R29Dc3hwN2JaeVNHCu1qeXVnQVFNI
n0.eyJqdGkiOiJhYmI5MTQxZi01MjMzLTQyMGItOTgwNi0zNjY4NWRhN2VhYTMiLCJ
leHAiOjE0OTk5NjM5MDUsIm5iZiI6MCwiaWF0IjoxNDk5OTYzNjA1LCJpc3MiOiJod
HRwOi8vMTAuMC44Ljg1Ojg2ODYvYXV0aC9yZWFSbXMvV2VhdmVyIiwiYXVkJoiid2V
hdmVyLWNsaWVudCIsInN1YiI6IjQ1MzFkMzI0LWJkYjctNDAxOS04NDYwLWY1Mzc1N
mQ4ODlmYiIsInR5cCI6Ik1lIiwiYXpwIjoiid2VhdmVyLWNsaWVudCIsImF1dGhfdG1
tZSI6MCwic2Vzc21vb19zdGF0ZSI6Ijg4ZjBjMjkwlWVmNWmtNDBmMi1iYTk1LTMzM
2RiYmUyNzg2NSIsImFjciI6IjEiLCJuYW1lIjoiIiwichJlZmVycmVkJ3VzZXJuYW1
lIjoiYm9iIn0.eSMubExNRX8c2Mv9iCyeWyfktMGNDld8nZDJiZjaQjDNJswt3VKdt
e47Qg68OBv7rqyr4PxgVoec7s4V5omJq9wUba3sFZU1X_jOh1tnOj5tYFMEkyhmSu
SVQCAFSVjj_0kBxuf5Ut7A3Pnw8RNE7idjRyMqiBcKogYecR42faDB4eU5SZDw5ye
Fas1whArilpRSMsS3fP31BkJuVut4IzumkQXOsIwSjv42PN15kXDteRMtSt21esspU
i8U44KdeGwcCij8oc8FQyYy37ZWLNmeP7Eh8sO6XFkMGe7tBoKY7uLPmztsIvAp_IL
eNX3ndH1fZGPhXHxrZvECWJ0w",
    "not-before-policy": 0,
    "session_state": "88f0c290-ef5c-40f2-ba95-333dbbe27865"
}

```

You must extract the `access_token` for use in API calls.

Calling REST APIs in a Secure Environment

When calling REST APIs with security enabled, you must add an Authorization header, which takes the following form:

```
"Authorization: bearer <access_token>"
```

The following is an example of an Authorization header using the token generated in [Getting a Token](#):

```
"Authorization: bearer eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSlldUIiwia21k
IiA6ICJNQ21MOTJzTjYta0NXR0txUmpPYk95R29Dc3hwN2JaeVNHCu1qeXVnQVFNI
0.eyJqdGkiOiI1MjI3OTVmZS03MWU2LTQwZjgtYjNhMy1hMmQzODUyNDFiOGMiLCJ1
eHAiOjE0OTk5NjM5MDUsIm5iZiI6MCwiaWF0IjoxNDk5OTYzNjA1LCJpc3MiOiJodH
RwOi8vMTAuMC44Ljg1Ojg2ODYvYXV0aC9yZWFSbXMvV2VhdmVyIiwiYXVkJoiid2Vh
dmVyLWNsaWVudCIsInN1YiI6IjQ1MzFkMzI0LWJkYjctNDAxOS04NDYwLWY1Mzc1Nm
Q4ODlmYiIsInR5cCI6Ik1lYXJlcIIsImF6CCI6Ind1YXZlci1jbGllbnQiLCJhdXRo
X3RpBWUiOjAsInNlc3Npb25fc3RhGUIoI14OGYwYZi5MC1lZjVjLTQwZjItYmE5NS
0zMznkYmJlMjc4NjUiLCJhY3IiOiiXliwiY2xpZW50X3N1c3Npb24iOii5YTAxNDMx
My03Y2MzLTQ1YzAtOTg1Ny02YTU3MGR1NTFizWtiLCJhbGxvd2VklW9yaWdpbnMi01
tdLCJyZWFsbV9hY2Nlc3MiOnsicm9sZXMiOlsib2ZmbGluZV9hY2Nlc3MiLCJ1bwFF
YXV0aG9yaXphdGlvbiJdfSwicmVzb3Vyy2VfyWNjZXNzIjp7ImFjY291bnQiOnsicm
9sZXMiOlsibWFuYwd1LWFjY291bnQiLCJtYW5hZ2UtYWNjb3VudC1saW5rcyIsInZp
ZXctcHJvZmlsZSJdfX0sIm5hbWUiOiiIiLCJwcmVmZXJyZWRfdXN1cm5hbWUiOjib2
IifQ.FuFm5ra-C4Pclk_CDOI3q04mEmkxz2dV-Vvo6Ww3vTWHj5cfkRY1X3QpMbZN
Nv9IWX8cLyRKZrVQSCMH64BFH7uXXBgRakEjPx8qEsM4zPzVZADk0c5osOjQtgQdTh
DPRiR68FK-jkETM9-icLtAvsOMCAhvNF-F1QtmNbirkCx0IMuvTN3nI80ocfuCGf9Z
ScE80AwumFPhEN6jvt3oDIjhI-vz83c1SMXGKRKh73IFo7Z9eGCoCfSB-WHT3b7ZyX
a-igUapr9bCk-j4zamMEw0SzDDh2SyR0C6qvBkdWWEA6F1vFnav5KoOVW7uNqvf04G
ehB8cPYegVXv1yXlg"
```

The following is an example of an API call to instantiate a VNF that includes an Authorization header:

```
OVLMFE_BASEURL=http://10.3.18.22:28085
tenant_id=default
vnf_id=microblog
vnf_instance_id=dc_1
echo " - instantiate_without_vim - "
```

```

curl \
-S \
-H "Content-Type: application/json" \
-H "Authorization: bearer eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lk
IiA6ICJNQ21MOTJzTjYta0NR0txUmpPYk95R29Dc3hwN2JaeVNhcU1qeXVnQVFNI
0eyJqdGkiOiI1MjI3OTVmZS03MWU2LTQwZjgtYjNhMy1hMmQzODUyNDFiOGMiLCJl
eHAiOjE0OTk5NjM5MDUsIm5iZiI6MCwiaWF0IjoxNDk5OTYzNjA1LCJpc3MiOiJodH
RwOi8vMTAuMC44Ljg1Ojg2ODYvYXV0aC9yZWFSbXMvV2VhdmVyIiwiYXVkJoid2Vh
dmVyLWNsaWVuDCIsInN1YiI6IjQ1MzFkMzI0LWJkYjctNDAxOS04NDYwLWY1Mzc1Nm
Q4ODlmYiIsInR5cCI6IkJ1YXJlcisImF6ccI6IndlYXZlc1jbGllbnQiLCJhdXRo
X3RpBWUiOjAsInN1c3Npb25fc3RhGUIoiI4OGYwYZ15MC11ZjVjLTQwZjItYmE5NS
0zMzNkYmJlMjc4NjUiLCJhY3IiOiiXiwiY2xpZW50X3N1c3Npb24iOiI5YTAxNDMx
My03Y2MzLTQ1YzAtOTg1Ny02YTU3MGRlNTFizWIiLCJhbGxvd2VkLW9yaWdpbnMiO1
tdLCJyZWFSbV9hY2N1c3MiOnsicm9sZXMiOlsib2ZmbGluZV9hY2N1c3MiLCJ1bwFF
YXV0aG9yaXphdGlvbijdfSwicmVzb3VY2VfYWNjZXNzIjp7ImFjY291bnQiOnsicm
9sZXMiOlsibWFuYWd1LWFjY291bnQiLCJtYW5hZ2UtYWNjb3VudC1saW5rcyIsInZp
ZXctcHJvZmlsZSjdfX0sIm5hbWUiOiiLCJwcmVmZXJyZWRfdXN1cm5hbWUiOiJib2
IifQ.FuFm5ra-C4Pclk_CDOI3q04mEmkxz2dV-Vvo6Ww3vTWHJj5cfkRY1X3QpMbZN
Nv9IWX8cLyRKZrVQSCMH64BFH7uXXBgRakEjPx8qEsM4zPzVZADk0c5osOjQtgQdTh
DPRIr68FK-jkETM9-icLtAvsOMCAhvNF-F1QtmNBirkCxOIMuvTN3nI80ocfuCGf9Z
ScE80AwumFPhEN6jvt3oDIjhI-vz83c1SMXGKRKh73IFo7Z9eGCoCfSB-WHT3b7ZyX
a-igUapr9bCk-j4zamMEw0SzDDh2SyR0C6qvBkdWWAE6F1vFnav5KoOVW7uNqvf04G
ehB8cPYegVXvlyXlg" \
-X POST \
-d '{"workflow_type":"instantiate_vnf_without_vim", "vnf_template_id":"mic
roblog_v1"}' \
$OVLMFE_BASEURL/api/v2/tenants/${tenant_id}/vnfs/${vnf_id}/vnf_i
nstances/${vnf_instance_id}/workflow_instances?sync=1|
python -mjson.tool | pygmentize -l json

```

Creating VNFs

Using the Out of the Box procedures and flows, create a VNF package and deploy it.

This section describes how to create VNF packages and instantiate them as VNF instances using Openet Weaver. It also describes how to create custom procedures that can be used in VNF packages, and how to add an application user to run the procedures.

Creating VNF Packages

Using the Out of the Box procedures and flows, create a VNF package and deploy it.

This section describes how to create and instantiate a VNF using Openet Weaver. It covers creating VNFs with Virtualized Infrastructure Manager (VIM) configuration and creating self-managed VNFs without VIM configuration.

VNF packages are created and stored in Openet Weaver Configuration Manager. The process for creating a VNF package is as follows:

1. Create the tenant for the VNF, unless the tenant already exists.

A tenant is a customer or organization that requires common, isolated access to one or more VNFs so that they can manage them in a production or non-production system.

Note: The tenant must exist in the system before you can create the VNF.

2. Create a VNF workspace under the tenant. Each VNF must have its own workspace.
3. Create one or more VNF Components (VNFCs). A VNFC is a container used to associate the files that are required for the VNF.

4. Upload VNFC packages, converting VNFC package configuration files to packages if required for scalability and flexibility in complex deployments.
5. Create and upload the VNFC metadata file that maps the procedures, scripts, and commands related to the VNF lifecycle event the package is intended to manage.
6. Upload VNFC Configuration.
7. Prepare and Upload VNFC procedures, if custom procedures are required.

Note: The use cases discussed in this section rely on predefined procedures that are included with Openet Weaver.

8. Create a VNF instance.
9. If creating a VNF with VIM configuration, create and upload the VNF descriptor VNFD file that describes a VNF in terms of its VIM related deployment and operational behavior.
10. Create the VNF package.
11. If creating a VNF without VIM configuration, create and upload the topology file.
12. Instantiate the VNF.

To illustrate VNF creation, an Apache web server VNF is used as to demonstrate how to prepare the VNF package and deploy it.

The VNF package consists of the following files:

- httpd-2.4.6-405.el7.centos.x86_64.rpm—software package that is installed
- httpd.conf.jinja—the httpd server configuration file that is localized for the runtime node
- meta.yml—a VNFC-level configuration file that defines the procedures and parameters to use when deploying the VNF

Creating a Tenant

Before you create a VNF ensure that the tenant you are creating it for exists. If not, create it. A tenant is a customer or organization that requires common, isolated access to one or more VNFs so that they can manage them in a production or non-production system.

Note: Openet Weaver ships with a pre-created default tenant.

To create a tenant in Openet Weaver, use the command `ovlm-cm tenant create -t tenant_id -d description`

Note: Be sure the baseurl environment variables are set before using CLI commands. There are also API versions of the CLI commands in this topic.

The *tenant_id* must be unique in the system and can only contain alphanumeric characters and underscores only.

When you create a tenant, an underlying directory structure is also created as shown in the following table.

Table 31: Tenant Directory Structure

Directory	Description	Example
<tenant_id>	The main tenant directory named for the id given to the tenant when the command is run.	anycom
<tenant_id>/vnfs	The subdirectory in which VNFs are created.	anycom/vnfs
<tenant_id>/metadata	The subdirectory where metadata for the tenant is stored. A file named tenant.yml resides in this file, which contains information about the tenant.	anycom/metadata/tenant.yml

The following is an example of `tenant create` usage:

```
ovlm-cm tenant create -t anycom -d "anycom tenant"
```

The following is an example response:

```
data:
  method: POST
  request: http://localhost:9090/api/v2/tenants
status: 200
```

To delete a tenant use the following command:

```
ovlm-cm tenant delete -t tenant_id
```

Deleting a tenant removes the tenant from the system. Any related VNFs, files, and directories under the tenant are also deleted.

Note: It is not possible to delete the default tenant from Openet Weaver.

The following is an example of `delete tenant` usage

```
ovlm-cm tenant delete -t anycom
```

The following is an example response:

```
data:
  method: DELETE
  request: http://localhost:9090/api/v2/tenants/anycom
status: 200
```

Related concepts

[Setting Environment Variables](#) on page 133

[Openet Weaver VNF-M API Reference](#) on page 468
Openet Weaver REST API reference guides

[Openet Weaver VNF-M CLI Reference](#) on page 469
Openet Weaver CLI command reference guides

Creating Multi-tenant Flow Mappings

Downloading the flow mappings using the API commands creates a `flow_mappings.yml` file that only has the flow mappings for the default tenant. You must create and add flow mappings for other tenants to the file manually. Then you upload the updated file to the system so the new flow mappings for those tenants can take effect.

Note: Ensure the `baseurl` environment variables are set before using CLI commands. There are also API versions of the CLI commands in this topic.

To create multi-tenant flow mappings, follow these steps:

1. Create the `flow_mappings.yml` file using the following CLI command:

```
ovlm-fe metadata download
```

2. Open the file for editing. The following is an example of the `flow_mappings.yml` file:

```
flow_mappings: # The header
  instantiate_vnf: # The workflow type
    default: # TENANT ID
      default_vnf: # VNF ID
```

```

        flow_name: VnfInstantiationFlow
        flow_path: /Openet/OVLM/User_OOTB_Flows/
start_vnf: # The workflow type
    default: # TENANT
        default_vnf: # VNF ID
            flow_name: VnfStartFlow
            flow_path: /Openet/OVLM/User_OOTB_Flows/
stop_vnf: # The workflow type
    default: # TENANT
        default_vnf: # VNF ID
            flow_name: VnfStopFlow
            flow_path: /Openet/OVLM/User_OOTB_Flows/
is_vnf_up: # The workflow type
    default: # TENANT
        default_vnf: # VNF ID
            flow_name: IsVnfUpFlow
            flow_path: /Openet/OVLM/User_OOTB_Flows/

```

3. Identify a default tenant flow mapping you want to apply to a different tenant and their vnf, then copy and paste the flow mapping. the following is an example:

```

instantiate_vnf: # The workflow type
    default: # DEFAULT TENANT ID – CAN'T BE CHANGED
        default_vnf: # THE DEFAULT FLOW DEFENITION FOR THE TENANT – CAN'T BE
CHANGED
            flow_name: VnfInstantiationFlow
            flow_path: /Openet/OVLM/User_OOTB_Flows/
default: # TENANT ID –DUPLICATED FROM ABOVE
    default_vnf: # VNF ID
        flow_name: VnfInstantiationFlow
        flow_path: /Openet/OVLM/User_OOTB_Flows/

```

4. Update the block of text you paste in step 3 with the tenant ID and the VNF ID for which you are mapping a flow as in the following example:

```

instantiate_vnf: # The workflow type
    default: # TENANT ID
        default_vnf: # THE DEFAULT FLOW DEFENITION FOR THE TENANT – CAN'T BE
CHANGED
            flow_name: VnfInstantiationFlow
            flow_path: /Openet/OVLM/User_OOTB_Flows/
anycom: # NEW TENANT ID
    default_vnf: # THE DEFAULT FLOW DEFENITION FOR THE TENANT – CAN'T BE
CHANGED
        flow_name: VnfInstantiationFlow
        flow_path: /Openet/OVLM/User_OOTB_Flows/

```

5. Repeat steps 3 and 4 until you are satisfied that the flow mapping is complete.
6. Upload the updated flow_mappings.yml file using the following CLI command:

```
ovlm-fe metadata upload -f flow_mappings.yml
```

Note: You can include a path to the file if required.

Tip: Retain a copy of the amended file to facilitate future flow mapping updates.

The required flows are now mapped in Openet Weaver.

Creating a VNF Workspace and VNFC Containers

As part of VNF package creation you need to create a workspace for the VNF. A VNF workspace is an area in the Configuration Manager where you build a repository of VNF artefacts . Each VNF must have its own workspace.

You also need to create one or more VNF containers (VNFCs) within the VNF. VNFCs are used to associate files that are required for a VNF.

You create a VNF using the command `ovlm-cm vnf create -t tenant_id -v vnf_id -d description`, for example:

```
ovlm-cm vnf create -v webserver -t default -d "Sample Webserver"
```

You create a VNFC within the VNF using the command `ovlm-cm vnfc create -t tenant_id -v vnf_id -c vnf_id`, for example:

```
ovlm-cm vnfc create -v webserver -t default -c httpd
```

When you complete this task the VNF is ready to be populated with the metadata, packages, and configuration files, required for the VNF.

Creating and Uploading Metadata for the VNFC

VNFC metadata is defined in YML format. The metadata file describes the lifecycle of a VNF component and enables Openet Weaver to perform the required action when it receives an event. It defines the procedure definition, VNFC package names, service names, configuration files, and the parameters that can be passed to the procedure to be run, It also defines the name of the file to call when executing the procedure.

By providing the name of the file you can call out-of-the-box procedures.

The default out-of-the-box procedures are described in the following table. They are installed in the configuration manager node under /usr/lib64/ovlm/cm/repository-root/ootb_procedures. The procedures are written in python.

Table 32: Predefined Openet Weaver Procedures

Procedure	Description	Customisable	Metadata Configurable
stage	Transfers a VNF package to each Openet Weaver Resource Agent.	No	No
dynamic_config_vnfc	Updates values in all variables and local facts in VNFC configuration packages. Note: You can specify the character encoding to use in the parameters section. The default value is UTF8, which is used when character encoding is not specified.	Yes	Yes
update_config_vnfc	Activates the synced package and applies configuration changes to VNF working folders.	Yes	Yes

Procedure	Description	Customisable	Metadata Configurable
install_vnfc	<p>Installs required software VNFC package(s).</p> <p>Note: The installer provided with Openet Weaver does not support interactive input. For example, you might be prompted for user input as part of an installation process. In such a case you must create a custom script to respond to prompts because you cannot respond manually.</p> <p>This procedure supports RPM and DEB VNFC package-type formats.</p>	Yes	Yes
start_vnfc	Starts the VNFC is started as a Linux service using systemctl.	Yes	Yes
stop_vnfc	Stops an active VNFC.	Yes	Yes
is_vnfc_up	Checks whether the VNFC is up and running as a Linux service using systemctl.	Yes	Yes
is_vnfc_down	Checks whether the VNFC is down.	Yes	Yes
rollback_vnfc	Restores the VNFC backup previously done by update_config_vnfc action.	No	No
clean_up_rollback	Deletes the VNFC backup.	No	No
custom_action_vnfc	Runs a script based on the parameter(s) passed against a specific VNFC instance.	Yes	No
upgrade_vnfc	Uninstalls VNFC packages, then installs newer versions of the same VNFC packages. Uninstall packages or install packages can be null.	Yes	Yes
uninstall_vnfc	Uninstalls VNFC packages.	Yes	Yes
dynamic_config_reference_vnfc	Updates values and all variable and local facts in VNFC configuration.	Yes	Yes
installation_reference_vnfc	Installs required VNFC Packages.	Yes	Yes
update_config_reference_vnfc	Applies configuration changes to VNF working folders	Yes	Yes
post_install_reference_vnfc	<p>Performs user-configured post installation actions, for example cleaning up the environment.</p> <p>Note: This is an optional procedure.</p>	Yes	Yes

When you customize VNFC procedures listed in the table, ensure that implementation does not result in VNFCs being left in a partial state in the event that there is a failure during the execution of a procedure. The state of a VNFC must be restored back to the initial state it was in before the procedure was executed. Leaving a VNFC in a partial state between success and failure has a negative impact on rerunning workflows on that VNFC.

VNFC-Specific Users

In previous releases, all VNF lifecycle management procedures were performed by a single ovlm user. This approach required a set of sudo privileges to be granted to the ovlm user. Although running VNF lifecycle procedure as a specific VNFC user can be handled by these procedures directly, it is hard to change and test.

Using the `run_as` parameter enables you to configure a different user per VNFC as different a stage of VNF lifecycle management as part of creating a template. The VNF lifecycle procedures can be reusable and independent of the `run_as` user. This provides a clean separation between activities internal to Openet Weaver and activities carried out when managing the VNFC such as installing, starting, and stopping, for example.

Prerequisites

Resource Agent nodes need to be configured for the pre-created, specific VNFC user(s) that execute the VNFC-specific actions. These pre-created users must be configured as sudoers with the following configuration:

- NOPASSWD so that ovlm user can switch to specific VNF users without entering a password
- List of commands allowed to be run by the VNF user as required by underlying out-of-the-box procedures.
- requiretty must be disabled for the user

Note: It is recommendation that the VNFC user is set to NOPASSWD: ALL because if user is configured to execute with only a set of predefined commands as a sudoer, then there is no way for Weaver to know and validate the set of commands required by the underlying ootb script and that might result in failure to execute specific ootb procedure.

The following is an example of VNFC users configured in `/etc/sudoers`.

```
microblog  ALL=(ALL)          NOPASSWD:ALL
nginx    ALL=(ALL)          NOPASSWD:ALL

Defaults:microblog !requiretty
Defaults:nginx !requiretty
```

VNFC Actions that Support `run_as`

Configurable vnfc actions which support "run_as" user are as follows

- `upgrade_vnfc`
- `update_config_vnfc`
- `uninstall_vnfc`
- `stop_vnfc`
- `start_vnfc`
- `is_vnfc_up`
- `is_vnfc_down`
- `install_vnfc`

Procedures in a Metadata file

The following is an example a metadatafile. it includes the `run_as` parameter for several procedures to specify a VNFC user for those procedures.

```
# action_type: such as dynamic_config_vnfc # Mandatory
# - procedure_name: Mandatory
# - parameters: Optional

dynamic_config_vnfc:
  timeout: 60
  parameters:
    encoding: utf8
    configurations:
      - configuration/nginx.conf.jinja
      - configuration/others.conf.jinja
  procedure_name: dynamic_config_vnfc_v1.py

install_vnfc:
  run_as: ovlm
```

```
timeout: 60
parameters:
  packages:
    - packages/nginx-1.9.0-1.el6.ngx.x86_64.rpm
procedure_name: install_rpm_v1.py

is_vnfc_down:
  run_as: nginx
  timeout: 60
  parameters:
    services:
      - nginx
  procedure_name: is_vnfc_down_v1.py

is_vnfc_up:
  run_as: nginx
  timeout: 10
  parameters:
    services:
      - nginx
  procedure_name: is_vnfc_up_v1.py

start_vnfc:
  run_as: nginx
  timeout: 10
  parameters:
    services:
      - nginx
  procedure_name: start_vnfc_v1.py

stop_vnfc:
  run_as: nginx
  timeout: 10
  parameters:
    services:
      - nginx
  procedure_name: stop_vnfc_v1.py

update_config_vnfc:
  run_as: nginx
  timeout: 10
  parameters:
    - destFile: /etc/nginx/nginx.conf
      srcFile: configuration/nginx.conf
    - destFile: /etc/nginx/my_nginx.sh
      srcFile: configuration/my_nginx.sh
  procedure_name: update_config_vnfc_v1.py

dynamic_config_reference_vnfc:
  timeout: 60
  parameters:
    configurations:
      - configuration/nginx.conf.jinja
      - configuration/nginx_vnfc_monitoring.sh.jinja
        render_file_with_non_jinja_extension: true
  procedure_name: dynamic_reference_config_vnfc.py

install_reference_vnfc:
  timeout: 120
  parameters:
    packages:
```

```

    - packages/nginx-1.8.1-1.el7.ngx.x86_64.rpm
procedure_name: install_reference_vnfc.py

update_config_reference_vnfc:
parameters:
- destFile: /etc/nginx/nginx.conf
srcFile: configuration/nginx.conf
- destFile: /etc/nginx/weaver_monitor/nginx_vnfc_monitoring.sh
srcFile: configuration/nginx_vnfc_monitoring.sh
- destFile: /var/spool/cron/root
srcFile: configuration/cron_job
procedure_name: update_config_reference_vnfc.py

post_install_reference_vnfc:
timeout: 60
procedure_name: post_install_reference_vnfc.py

```

Using Metadata Parameters in Localizing Configuration

In the metadata file the ServerName parameter is used for dynamic configuration.

To use this parameter to localize the httpd.conf.jinja file, edit the httpd.conf.jinja file and add `ServerName {{ metadata.ServerName }}`. The metadata server name is dynamically substituted with the parameter, for example:

```
ServerName node-3
```

During processing, the `dynamic_config_vnfc` procedure substitutes the parameter in the file.

Uploading Metadata for the VNFC

When you finish creating and configuring the metadata file, upload the file using the command `ovlm-cm vnfc_metadata upload -v vnf_id -t tenant_id -c vnfc_id -f filename`, for example:

```
ovlm-cm vnfc_metadata upload -v webserver -t default -c httpd -f meta.yml
```

Uploading Packages and Configuration Files

In this part of the VNF package creation process you begin to populate the VNFC with the RPMs and configuration information that goes into VNF package.

You upload the RPMs using the command `ovlm-cm vnfc_package upload -v vnf_id -t tenant_id -c vnfc_id -f package`, for example:

```
ovlm-cm vnfc_package upload -v webserver -t default -c httpd -f
httpd-2.4.6-405.el7.centos.x86_64.rpm
```

You must create a configuration file for the VNF, for example the `httpd.conf` file for the Apache web server. This allows you to customize parameters. In this example, set the `ServerName` parameter to be updated with the `ServerName` parameter provided in the metadata file by editing the configuration file, and adding the following line using the `Jinja2` syntax, which are replaced when the file is deployed:

```
ServerName {{ metadata.ServerName }}
```

Save the configuration file, for example `httpd.conf.jinja`, and upload it to the server using the command `ovlm-cm vnfc_configuration upload -v vnf_id -t tenant_id -c vnfc_id -f filename`, for example:

```
ovlm-cm vnfc_configuration upload -v webserver -t default -c httpd -f
httpd.conf.jinja
```

Creating and Uploading a VNF Descriptor (VNFD) File

The VNF descriptor (VNFD) describes a VNF in terms of its VIM infrastructure-related deployment and operational behaviour.

Important: You are required to create and upload a VNFD file only when you are creating and configuring a VNF for VIM. Do not perform this task if you are creating and configuring a VNF without VIM.

The VNFD file is in the YML format. The definition is based on the Topology and orchestration specification for cloud applications (TOSCA) as defined by OASIS.

Note: OASIS also defines a specification for Network Function Virtualization, which is being used for Openet Weaver. In order to support additional requirements for Openet Weaver, for example a floating IP attachment and the IP used for RA node, an extension on top of the TOSCA NFV is developed.

In Openet Weaver, VNFDs are used to process on-boarding and lifecycle management for VNFs with VIM deployment configuration. For example, VNFDs are used to allocate VNF resources and to autogenerate the VNF topology when a VNF is instantiated.

VNFDs are used to define the package for a VNF in addition to virtualization deployment unit (VDU) requirements for instantiated VNFs. A VDU supports the deployment description and operational behaviour a VNF and can be used to specify the following VM requirements for a VNF instance:

- CPU requirements
- Hard disk requirements
- RAM requirements

Note: The scope of what VNFDs can be used to define will increase in later versions of Openet Weaver.

VDUs have a one-to-one correspondence with VMs. In other words, a different set of requirements can be defined for each VM. Because VDUs are also mapped to VNFCs you can potentially define a set of VDU requirements for each instance of a VNF.

The relationship between VNFDs, VDUs and VNFCs is shown in the graphic below.

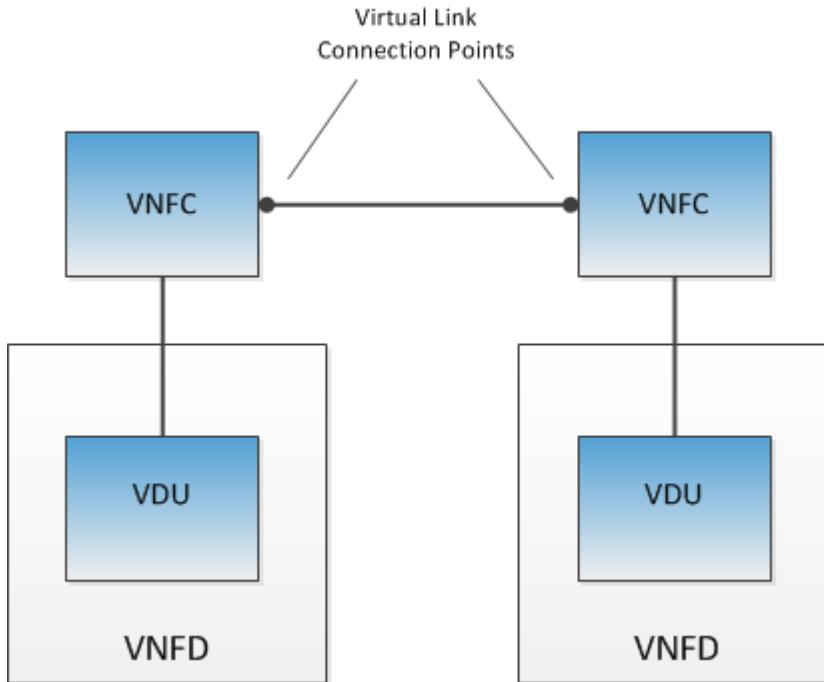


Figure 25: Relationship Between VNFDs, VDUs, and VNFCs

Each instance of a VNF must have a connection point specified that corresponds to a virtualized network interface card on the VM. Each connection point must be associated with a virtual link. Connection points represent physical links to

the VM, for example VM network cards, whereas virtual links are logical links between instances of the same VNF, for example LAN segments that are common to VNF components. Connection points can only communicate with each other when they share the same virtual link.

Connection points and VDUs are defined in a vnfd.yml file.

Creating a VNFD YML File

A VNFD file is defined by the fields described in the tables in this section. The following table describes the basic fields. The attributes for one of those fields, node_templates, are described in [Table 2](#).

Table 33: Basic VNFD Fields

Field	Description
tosca_definitions_version	The version of the TOSCA specification to be used. Currently the only valid value is openet_tosca_nfvi_1_0_0
description	A text description of the VNFD file.
topology_template	Used to define the structure of the service. A topology template consists of a set of node templates and relationship templates that work together to define the topology model of a service as a directed graph.
node_templates	Used to define an instance of a node type as a component of a service. A node type defines the properties of such a component and the operations available to manipulate the component. A node name is defined for each node type. node1 is the node name in the example VNFD file below. Node names must begin with an alphanumeric character and can contain only alphanumeric characters and hyphens, for example: <ul style="list-style-type: none"> • node-policy • policy-1 • 01-policy

The next table describes the node_template attributes.

Table 34: node_template Attributes

node type	node type property	Attribute	Datatype	Mand.	Description
tosca.nodes.nfv.VDU					A Tosca node type that represents a logical VDU entity.

node type	node type property	Attribute	Datatype	Mand.	Description
	capabilities.host	num_cpus	integer	no	Number of CPUs allocated for the node type instance.
		disk_size	scalar-unit.size	no	Size of disk allocated for the node type instance.
		mem_size	scalar-unit.size	no	Size of memory allocated for the node type instance.
	capabilities.os	architecture	string	no	The host operating system architecture.
		type	string	no	The host operating system type.
		distribution	string	no	The host operating system distribution. The following are examples of valid distribution values when the operating system type is "Linux": <ul style="list-style-type: none"> • centos • debian • fedora • rhel • ubuntu
		version	string	no	The host Operating System version.
openet.nodes.nfv.VDU					An extension derived from tosca.nodes.nfv.VDU that is used to support IP assignments.

node type	node type property	Attribute	Datatype	Mand.	Description
	properties	vnfc_id	string	no	The ID of the VNFC associated with the VNFD.
		ovlm_cp_private_ip	string	no	Specifies the connection point from which the Resource Agent uses the private IP address. You can specify a value for ovlm_cp_private_ip or ovlm_cp_floating_ip, but you cannot provide a value for both attributes. Populating both attributes results in a validation error. If neither attribute is assigned a value, the VIM Abstraction Layer checks whether the floating_ip attribute is set to true. If it is, the VDU uses the floating IP address as the Resource Agent connection point. If false, the VDU uses the private IP address.
		ovlm_cp_floating_ip	string	no	Specifies the connection point from which the Resource Agent uses the floating IP address. If more than one network is connected to a VDU this field can be used to select the floating IP address to be used as connection point. This field can be used only if floating_ip is true (enabled), otherwise VNFD validation fails.
		availability_zone	string	no	Name of the availability zone for server placement. See your system administrator for the value to use here.
tosca.nodes.nfv.VL				A Tosca node type that represents a logical virtual link entity.	

node type	node type property	Attribute	Datatype	Mand.	Description
	properties	cidr	string	no	The cidr block of the requested network.
		start_ip	string	no	The IP address to be used as the start of a pool of addresses within the full IP range derived from the cidr block.
		end_ip	string	no	The IP address to be used as the end of a pool of addresses within the full IP range derived from the cidr block.
		vendor	string	yes	Name of the vendor who generated this virtual link.
		network_id	string	no	An identifier of the VIM network instance. The value can be a UUID or the name of the network instance. Note: For OpenStack Kilo version, network_id must be in UUID value. For OpenStack Liberty version, network_id can be a UUID or name value.
	openet.nodes.nfv.VL				
	properties	subnet_id	string	yes	Subnet id of the virtual link connected to the VDU
tosca.nodes.nfv.CP					A Tosca node type that represents a logical connection point entity.
	requirements: virtualLink	node	string	no	The connecting node as defined by tosca.nodes.nfv.VL
		relationship	string	yes	The default value is tosca.relationships.nfv.VirtualLinksTo.
	requirements: virtualBinding	node	string	no	The connection node as defined in tosca.nodes.nfv.VDU
		relationship	string	yes	The default value is tosca.relationships.nfv.VirtualBindsTo.
openet.nodes.nfv.CP					An extension derived from tosca.nodes.nfv.CP that is used to support multiple floating IP assignments per VNFC.

node type	node type property	Attribute	Datatype	Mand.	Description
	properties	security_groups	list	no	<p>One or more entities within OpenStack that define rules of allowed data access based on the following criteria:</p> <ul style="list-style-type: none"> • Direction (in/out) • Port number • Protocol • Source address <p>You can determine which security groups are available for use by opening the OpenStack Dashboard and selecting Project > Compute > Access & Security, then clicking the Security Groups tab.</p>
	use_dynamic_floating_ip		Boolean	No	Whether the Connection Point is associated with floating IP address provided by OpenStack
	static_floating_ip_address		String	No	<p>Whether the Connection Point is associated with the user-provided floating IP address inside the VNFD (Tosca Template)</p> <p>The value for this attribute is the static IP address</p>
	source_network		String	Mandatory if either of the other two properties are defined.	The name of the network that provides the floating IP address allocation.Specifying

The following is an example vnfd.yml file. VDUs are defined on a per-node basis. Connection points (CP) and virtual links (VL) are defined below the node information.

```

tosca_definitions_version: openet_tosca_nfv_1_0_0
description: Template for deploying server with floating IP
topology_template:
  node_templates:
    node1:
      type: openet.nodes.nfv.VDU
      properties:
        vnfc_id: webserver
        ovlm_cp_private_ip : CP12
        availability_zone: nova
      capabilities:
        host:
          properties:
            num_cpus: 1
            disk_size: 10 GB
            mem_size: 1024 MB

```

```
os:

properties:
    architecture: x86_64
    type: linux
    distribution: centos
    version: 7.1

node2:
    type: openet.nodes.nfv.VDU
    properties:
        vnfc_id: database
        ovlm_cp_floating_ip : CP21F
        availability_zone: nova
    capabilities:
        host:
            properties:
                num_cpus: 1
                disk_size: 10 GB
                mem_size: 1024 MB
    os:

properties:
    architecture: x86_64
    type: linux
    distribution: centos
    version: 7.1

node3:
    type: openet.nodes.nfv.VDU
    properties:
        vnfc_id: database
    capabilities:
        host:
            properties:
                num_cpus: 1
                disk_size: 10 GB
                mem_size: 1024 MB
    os:

properties:
    architecture: x86_64
    type: linux
    distribution: centos
    version: 7.1

node4:
    type: openet.nodes.nfv.VDU
    properties:
        vnfc_id: storage_server
    capabilities:
        host:
            properties:
                num_cpus: 1
                disk_size: 10 GB
                mem_size: 1024 MB
    os:

properties:
    architecture: x86_64
    type: linux
    distribution: centos
    version: 7.1
```

```

CP11F:
  type: tosca.nodes.nfv.CP
  properties:
    use_dynamic_floating_ip: true
    source_network: admin_floating_net
  requirements:
    - virtualLink:
        node: VL1
        relationship: tosca.relationships.nfv.VirtualLinksTo
    - virtualBinding:
        node: node1
        relationship: tosca.relationships.nfv.VirtualBindsTo
CP12:
  type: tosca.nodes.nfv.CP
  requirements:
    - virtualLink:
        node: VL2
        relationship: tosca.relationships.nfv.VirtualLinksTo
    - virtualBinding:
        node: node1
        relationship: tosca.relationships.nfv.VirtualBindsTo
CP21F:
  type: tosca.nodes.nfv.CP
  properties:
    use_dynamic_floating_ip: true
    source_network: admin_floating_net
  requirements:
    - virtualLink:
        node: VL1
        relationship: tosca.relationships.nfv.VirtualLinksTo
    - virtualBinding:
        node: node2
        relationship: tosca.relationships.nfv.VirtualBindsTo
CP22:
  type: tosca.nodes.nfv.CP
  requirements:
    - virtualLink:
        node: VL2
        relationship: tosca.relationships.nfv.VirtualLinksTo
    - virtualBinding:
        node: node2
        relationship: tosca.relationships.nfv.VirtualBindsTo
CP3:
  type: tosca.nodes.nfv.CP
  requirements:
    - virtualLink:
        node: VL2
        relationship: tosca.relationships.nfv.VirtualLinksTo
    - virtualBinding:
        node: node3
        relationship: tosca.relationships.nfv.VirtualBindsTo
CP4F:
  type: tosca.nodes.nfv.CP
  properties:
    use_dynamic_floating_ip: true
    source_network: admin_floating_net
  requirements:
    - virtualLink:

```

```

        node: VL1
        relationship: tosca.relationships.nfv.VirtualLinksTo
    - virtualBinding:
        node: node4
        relationship: tosca.relationships.nfv.VirtualBindsTo

VL1:
    type: openet.nodes.nfv.VL
    properties:
        network_id: admin_internal_net
        subnet_id: admin_internal_net_subnet
        vendor: openet

VL2:
    type: tosca.nodes.nfv.VL
    properties:
        cidr: 192.168.0.1/24
        start_ip : 192.168.0.100
        end_ip : 192.168.0.150
        vendor: halo-network

```

Uploading a VNFD File

Before you can upload a VNFD file you need to create a VNFD directory in the target VNF structure on Configuration Manager. You do so using the command `ovlm-cm vnfd create --t tenant_id -v vnf_id -i vnf_id -m template_type -d description`, for example:

```
ovlm-cm vnfd create -t default -v webserver -i test_vnfd -m tosca -d "This VNFD was created for testing."
```

Note: tosca is currently the only supported value for template_type in the create vnfd command.

Then you can upload the VNFD file using the command `ovlm-cm vnfd upload --t tenant_id -v vnf_id -i vnf_id -f vnfd.yml`, for example:

```
ovlm-cm vnfd upload -t default -v webserver -i test_vnfd -f vnfd.yml
```

Specifying Connection Points in a VNFD

This topic describes how to use the VNFD `openet.nodes.nfv.CP` node for creating connection points.

The connection point types are as follows:

- Static Floating IP—a predefined IP address that is accessible from an external network
- Dynamic Floating IP—an automatically assigned IP address that is accessible from an external network
- Static Private IP—a predefined IP address that is accessible only from within OpenStack
- Dynamic Private IP—an automatically assigned IP address that is accessible only from within OpenStack
- Static Private IP on a floating Connection point—a floating IP address associated with a private IP address that complies with the subnet of the connection point target

All connection points declared in a VNFD Template that are connected to a virtual link have a Private IP address. When a Floating IP address is associated with a connection point, another IP address that is accessible from an external network is linked to the existing private IP address. Therefore a connection point that is floating-IP enabled has two IP addresses: a private IP address and a floating IP address.

In the examples used in this topic to illustrate the various connection types, the following are existing networks or network subnets that connect with the external network:

- `admin_internal_net`
- `admin_internal_net_subnet`
- `test_kl_kw`

- `klkwt_subnet`

The floating IP address is provided from `admin_floating_net`.

Assigning a Static Floating IP

This IP address is predefined in the TOSCA Template, and is accessible from outside of OpenStack. You must configure the following properties:

- `static_floating_ip_address`—the IP address to be set to the VM
- `source_network`—the network name that provides the floating IP address
- A virtual link for this connection point that is a network segment that connects to an external network.

The following is an example VNFD with a static floating IP assignment.

```

tosca_definitions_version: openet_tosca_nfv_1_0_0
description: Example template for static floating IP
topology_template:
  node_templates:
    node1:
      type: openet.nodes.nfv.VDU
      properties:
        vnfc_id: webserver
      capabilities:
        host:
          properties:
            num_cpus: 2
            disk_size: 4 GB
            mem_size: 4096 MB
        os:
          properties:
            architecture: x86_64
            type: linux
            distribution: centos
            version: 7.1

    CP11:
      type: openet.nodes.nfv.CP
      properties:
        static_floating_ip_address: 10.0.124.70
        source_network: admin_floating_net
      requirements:
        - virtualLink:
            node: VL_ADMIN_INT
            relationship: tosca.relationships.nfv.VirtualLinksTo
        - virtualBinding:
            node: node1
            relationship: tosca.relationships.nfv.VirtualBindsTo

  VL_ADMIN_INT:
    type: openet.nodes.nfv.VL
    properties:
      network_id: admin_internal_net
      subnet_id: admin_internal_net__subnet
      vendor: openet

```

When defining the IP address for static floating IP, review the OpenStack dashboard (Horizon) Access & Security page to verify that the IP address you specify for the connection point is allocated to the OpenStack project and is currently available. If it is available the Mapped Fixed IP Address column for the IP address is populated as shown below.

	IP Address	Mapped Fixed IP Address	Pool	Status	Actions
1	10.0.124.70	-	admin_floating_net	Down	<button>Associate</button>
2	10.0.124.68	-	admin_floating_net	Down	<button>Associate</button>
3	10.0.124.72	-	admin_floating_net	Down	<button>Associate</button>
4	10.0.124.74	-	admin_floating_net	Down	<button>Associate</button>
5	10.0.124.79	-	admin_floating_net	Down	<button>Associate</button>
6	10.0.124.65	-	admin_floating_net	Down	<button>Associate</button>
7	10.0.124.69	-	admin_floating_net	Down	<button>Associate</button>
8	10.0.124.64	-	admin_floating_net	Down	<button>Associate</button>
9	10.0.124.66	-	admin_floating_net	Down	<button>Associate</button>

Figure 26: Available IP Addresses in OpenStack

Assigning a Dynamic Floating IP

This IP address is assigned automatically by OpenStack. It is accessible from outside of OpenStack. You must configure the following properties:

- `use_dynamic_floating_ip = true`—when set to true, the connection point is assigned as dynamic floating IP
- `source_network`—the network name that provides the floating IP address
- A virtual link for this connection point that is a network segment that connects to an external network.

The following is an example VNFD with a dynamic floating IP assignment.

```

tosca_definitions_version: openet_tosca_nfv_1_0_0
description: Example template for dynamic floating IP
topology_template:
  node_templates:
    node1:
      type: openet.nodes.nfv.VDU
      properties:
        vnfc_id: webserver
      capabilities:
        host:
          properties:
            num_cpus: 2
            disk_size: 4 GB
            mem_size: 4096 MB
        os:
          properties:
            architecture: x86_64
            type: linux
            distribution: centos
            version: 7.1
    CP11:
      type: openet.nodes.nfv.CP
      properties:
        use_dynamic_floating_ip: true
        source_network: admin_floating_net
      requirements:
        - virtualLink:
            node: VL_ADMIN_INT
  
```

```

        relationship: tosca.relationships.nfv.VirtualLinksTo
- virtualBinding:
    node: node1
    relationship: tosca.relationships.nfv.VirtualBindsTo

VL_ADMIN_INT:
type: openet.nodes.nfv.VL
properties:
    network_id: admin_internal_net
    subnet_id: admin_internal_net_subnet
    vendor: openet

```

Assigning a Dynamic Private IP

This IP address is assigned automatically by OpenStack, and the address is accessible only from within OpenStack. There is no property configuration required that is specific to this type of connection point.

Dynamic private IP addressing is provided as a basic feature of OpenStack and TOSCA support. The node type `tosca.nodes.nfv.CP` is sufficient. You are not required to use `openet.nodes.nfv.CP`.

The following is an example VNFD with a dynamic private IP assignment.

```

tosca_definitions_version: openet_tosca_nfv_1_0_0
description: Example template for static private IP
topology_template:
  node_templates:
    node1:
      type: openet.nodes.nfv.VDU
      properties:
        vnfc_id: webserver
      capabilities:
        host:
          properties:
            num_cpus: 2
            disk_size: 4 GB
            mem_size: 4096 MB
      os:
        properties:
          architecture: x86_64
          type: linux
          distribution: centos
          version: 7.1

    CP11:
      type: openet.nodes.nfv.CP
      requirements:
        - virtualLink:
            node: VL1
            relationship: tosca.relationships.nfv.VirtualLinksTo
        - virtualBinding:
            node: node1
            relationship: tosca.relationships.nfv.VirtualBindsTo

    VL1:
      type: tosca.nodes.nfv.VL
      properties:
        cidr: 192.168.1.1/24
        start_ip : 192.168.1.100
        end_ip : 192.168.1.150
        vendor: halo-network

```

Assigning a Static Private IP

This type of connection point is a static IP defined in the VNFD. The IP address is accessible only from within OpenStack. For this type of connection point you configure only the ip_address property with the IP address for the network.

Static private IP addressing is provided as a basic feature of OpenStack and TOSCA support. The node type `tosca.nodes.nfv.CP` is sufficient. You are not required to use `openet.nodes.nfv.CP`.

The following is an example VNFD with a dynamic private IP assignment.

```

tosca_definitions_version: openet_tosca_nfv_1_0_0
description: Example template for static private IP
topology_template:
  node_templates:
    node1:
      type: openet.nodes.nfv.VDU
      properties:
        vnfc_id: webserver
      capabilities:
        host:
          properties:
            num_cpus: 2
            disk_size: 4 GB
            mem_size: 4096 MB
        os:
          properties:
            architecture: x86_64
            type: linux
            distribution: centos
            version: 7.1

    CP11:
      type: tosca.nodes.nfv.CP
      properties:
        ip_address: 192.168.1.101
      requirements:
        - virtualLink:
            node: VL1
            relationship: tosca.relationships.nfv.VirtualLinksTo
        - virtualBinding:
            node: node1
            relationship: tosca.relationships.nfv.VirtualBindsTo

    VL1:
      type: tosca.nodes.nfv.VL
      properties:
        cidr: 192.168.1.1/24
        start_ip : 192.168.1.100
        end_ip : 192.168.1.150
        vendor: halo-network
  
```

Assigning a Static Private IP on a Floating Connection Point

This type of connection point is a floating IP connection point with a predefined Private IP address specified in the VNFD. For this to work, the private IP address must be associated with the subnet to which the connection point is being linked.

In this example, the IP address (192.168.200.22) must be in the range defined by subnet `klkwt_subnet` (192.168.200.2 to 192.168.200.254)

```

tosca_definitions_version: openet_tosca_nfv_1_0_0
description: Example template for static private IP
  
```

```

topology_template:
  node_templates:
    node1:
      type: openet.nodes.nfv.VDU
      properties:
        vnfc_id: webserver
      capabilities:
        host:
          properties:
            num_cpus: 2
            disk_size: 4 GB
            mem_size: 4096 MB
        os:
          properties:
            architecture: x86_64
            type: linux
            distribution: centos
            version: 7.1

    CP22:
      type: openet.nodes.nfv.CP
      properties:
        static_floating_ip_address: 10.0.124.74
        source_network: admin_floating_net
        ip_address: 192.168.200.22
      requirements:
        - virtualLink:
            node: VL_KLKWT
            relationship: tosca.relationships.nfv.VirtualLinksTo
        - virtualBinding:
            node: node1
            relationship: tosca.relationships.nfv.VirtualBindsTo

    VL_KLKWT:
      type: openet.nodes.nfv.VL
      properties:
        network_id: test_kl_kwt
        subnet_id: klkwt_subnet
        vendor: openet
  
```

Review the OpenStack Dashboard (Horizon) Network Details page to verify the range of the subnet. The following shows the klkwt network details with a link to the klkwt_subnet details.

Network Details: test_kl_kwt

Network Overview

Name	test_kl_kwt
ID	2069b398-d204-4288-87db-eb04124fc330
Project ID	25179b8b9534e8a9b58d8de3b0dad5c
Status	Active
Admin State	UP
Shared	No
External Network	No
MTU	Unknown
Provider Network	Network Type: vlan Physical Network: physnet2 Segmentation ID: 1003

Subnets

Name	Network Address	IP Version	Gateway IP	Actions
klkwt_subnet	192.168.200.0/24	IPv4	192.168.200.1	Edit Subnet

Displaying 1 item

Ports

Name	Fixed IPs	Attached Device	Status	Admin State	Actions
(ba40cb02-7f6a)	192.168.200.29	network:dhcp	Active	UP	Edit Port
(9bf54f0-4eec)	192.168.200.1	network:router_interface	Active	UP	Edit Port

Displaying 2 items

Figure 27: klkwt_subnet Network Details

The subnet details are as shown below.

Subnet Details

Subnet Overview

Subnet

Name	klkwt_subnet
ID	69212206-a286-4356-8d1f-f276dc8dbad3
Network ID	2069b398-d204-4288-87db-eb04124fc330
Subnetpool	None
IP version	IPv4
CIDR	192.168.200.0/24
IP allocation pool	Start 192.168.200.2 - End 192.168.200.254
Gateway IP	192.168.200.1
DHCP Enable	Yes
Additional routes	None
DNS name server	None

Figure 28: klkwt_subnet Network Details

Creating a VNF Instance

This section describes how to create a VNF instance. VNF instances exist as a subdirectory below the tenant but outside of the VNF structure to which they are associated

The VNF instance is where the VNF topology is stored, which comes later in the Creating a VNF Package process.

Note: When you create a VNF with WIM configuration, the topology file is generated automatically based on the VNFD as described in Creating and Uploading a VNF Descriptor (VNFD) File,

When you create a VNF without WIM configuration you must create a topology file manually and upload it to the VNF instance as described in Creating and Uploading a VNF topology file.

You can create a VNF instance using CLI or API commands. Alternatively you can use the VNF-M GUI.

Creating a VNF Instance from the CLI

You create the subdirectory structure for the VNF instance using the command `ovlm-cm vnf_instance create -t tenant_id -v vnf_id -i vnf_instance_id -d description`, for example:

```
ovlm-cm vnf_instance create -t default -v webserver -i dublin -d "Dublin
webserver instance"
```

Creating a VNF Instance Using the VNF-M GUI

To create a VNF instance using the VNF-M GUI, follow these steps.

1. In the VNF Lifecycle page, click the floating button



in the bottom right corner of the page shown.

Figure 29: Floating Button in VNF Lifecycle Page

The Create a VNF Instance dialog box is displayed.

The screenshot shows a 'Create VNF Instance' dialog box. At the top, it says 'Create VNF Instance'. Below that is a dropdown menu labeled 'VNF*' with a small downward arrow icon. Underneath is a 'Name *' field with a horizontal line. Below that is a 'Description *' field with a horizontal line. At the bottom right are two buttons: 'CANCEL' and 'CREATE'.

Figure 30: Create a VNF Instance Dialog Box

2. In the Create a VNF Instance dialog box, select a VNF from the **VNF** drop-down list, for example **microblog**.
 3. Enter a name for the VNF Instance in the **Name** field, for example **dublin**.
 4. Enter a description of the VNF instance in the **Description** field.
 5. Click **Create**.
- When the VNF instance is created successfully it displays on a VNF card in the **Operating normally** section VNF Lifecycle page.

Creating the VNF Package

After performing all the tasks prior to this one in this section, you are ready to create the VNF package. To do so, use the command Create a package of the configuration that is ready to deploy using the command `ovlm-cm template create -t tenant_id -v vnf_id -p vnf_template_id`, for example:

```
ovlm-cm template create -t default -v webserver -p httpServer_v1
```

Creating and Uploading the VNF Topology YML File

The VNF topology is defined in YML format and describes which VNFCs are deployed to which nodes in the topology.

This topic describes how to create and upload topology files for VNF packages that are configured without VIM.

Important: You are required to perform this task only when you are creating and configuring VNFs without VIM configuration. Topology files for VNFs that are configured with VIM are generated automatically based on the VNFD.

VNFs are systems that can be made up of multiple modules called VNF components (VNFCs). The function of the VNF instance topology file is to map which VNFCs are deployed to which runtime agent nodes.

The general format of this file is:

```

vnfc_ids:
  <vnfc1>:
    - hostname: <host1.1>
      vnfc_instance_id: 1
      server_name: <hostname>
    [ (...)]
    - hostname: <host1.N>
      vnfc_instance_id: N
      server_name: <hostname>
    [ (...)]
  <vnfcX>:
    - hostname: <hostX.1>
      vnfc_instance_id: 1
      server_name: <hostname>
    [ (...)]
    - hostname: <hostX.M>
      vnfc_instance_id: M]
      server_name: <hostname>

```

In the above, `server_name <hostname>` is the hostname of the VM. This is required when Fault Manager, Performance Manager, or both are used.

It is possible to have one or more VNFCs configured. Each VNFC can be deployed to one or more hosts. The attribute `vnfc_instance_id` must be unique within the same VNFC.

Openet Weaver supports one-to-one mapping between a VNFC instance and a VM, which is the standard deployment model in a cloud environment. Openet Weaver also supports running multiple VNFC instances on the same VM. You have the option of running multiple VNFC instances on a single VM to reduce the performance overhead by having fewer VMs in a private, virtualized environment.

The remainder of this topic provides example topology files for the following use cases:

- A single VNFC instance running on a single VM
- Multiple VNF instances from different VNFs sharing a VM
- Multiple VNFC instances from the same VNF sharing a VM

A Single VNFC Instance Running on a Single VM

The out-of-the-box microblog, shown below, is an example of a topology file with a single VNF instance mapped to a single VM.

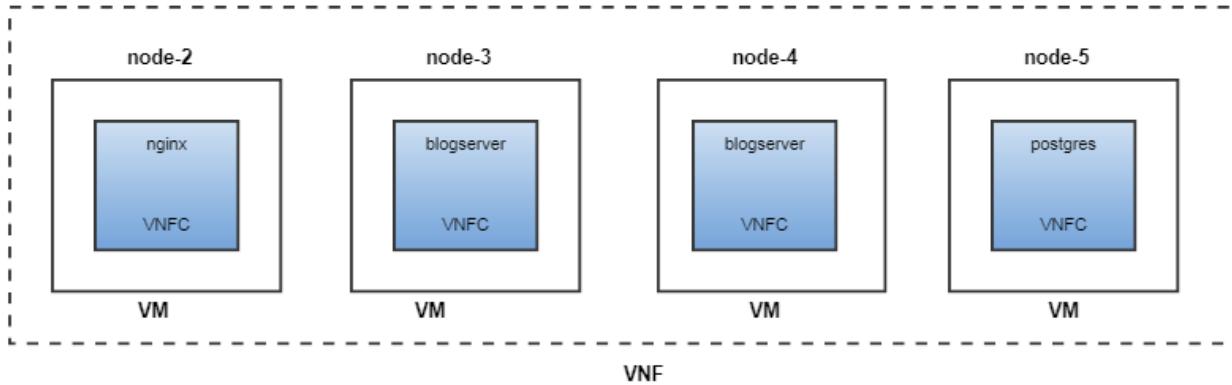
Note: This is the topology file used in the Creating a VNF Without VIM scenario. Save the file as `vnf_topology.yml` and upload it after creating a VNF instance.

```

vnfc_ids:
  nginx:
    - hostname: node-2
      vnfc_instance_id: 1
  blogserver:
    - hostname: node-3
      vnfc_instance_id: 1
    - hostname: node-4
      vnfc_instance_id: 2
  postgres:
    - hostname: node-5
      vnfc_instance_id: 1

```

The following illustrates the topology described above:

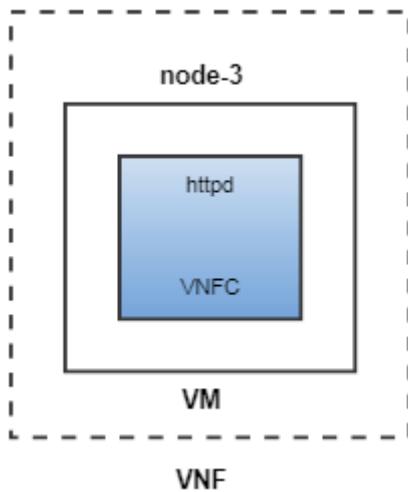


In this example there are three VNFCs: nginx, blogserver and postgres. Nginx is deployed to node-2, blogserver is deployed to node-3 and node-4 and postgres is deployed to node-5

In this example, one VNFC (httpd) is deployed to one host (node-3).

```
vnfc_ids:
  httpd:
    - hostname: node-3
      vnfc_instance_id: 1
```

The following is an illustration of node-3 after the VNFC is deployed:



Multiple VNF Instances from Different VNFs Sharing a VM

In this example, a microblog VNF and a LoadBalancer VNF share the same set of VMs in order to reduce the total number of VMs required to support the deployment.

The microblog VNF has the following configuration:

- VNF—microblog
- VNF template—microblog_v1
- VNF Instance—dc_1
- VNFCs:
 - microblog
 - nginx
 - postgres

In this example, the vnf_topology.yml file for VNF instance dc_1 is as below.

```
vnfc_ids:
  blogserver:
    - hostname: 192.168.10.1
      server_name: node-1
      vnfc_instance_id: 1
    - hostname: 192.168.10.2
      server_name: node-2
      vnfc_instance_id: 2
  nginx:
    - hostname: 192.168.10.3
      server_name: node-3
      vnfc_instance_id: 1
  postgres:
    - hostname: 192.168.10.4
      server_name: node-4
      vnfc_instance_id: 1
```

The LoadBalancer VNF has the following configuration:

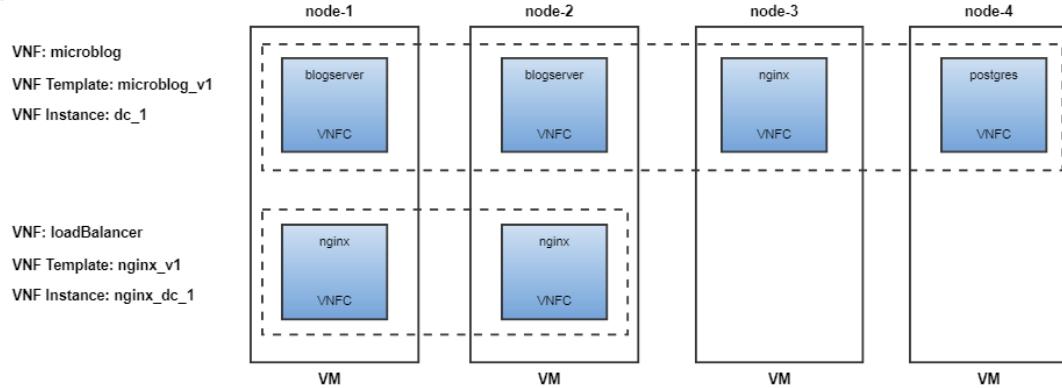
- VNF— LoadBalancer
- VNF template— nginx_v1
- VNF Instance— nginx_dc_1
- VNFC—nginx

In this example, the vnf_topology.yml file for VNF instance nginx_dc_1 is as below.

```
vnfc_ids:
  nginx:
    - hostname: 192.168.10.1
      server_name: node-1
      vnfc_instance_id: 1
    - hostname: 192.168.10.2
      server_name: node-2
      vnfc_instance_id: 2
```

The following illustrates the topology for the microblog and the load balancer:

Figure 3:



Multiple VNFC Instances from the Same VNF Sharing a VM

In this example, two clusters of VNFC instances (blogserver and nginx) from the one microblog VNF share the same set of VMs in order to reduce the total number of VMs required to support the deployment.

The microblog VNF has the following configuration:

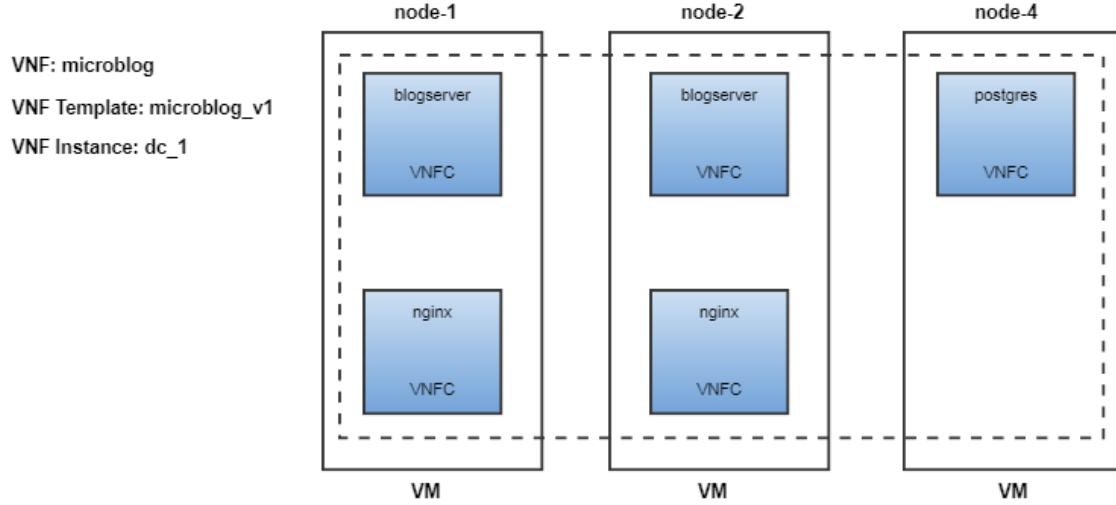
- VNF—microblog
- VNF template—microblog_v1
- VNF Instance—dc_1
- VNFCs:
 - microblog
 - nginx
 - postgres

In this example, the vnf_topology.yml file for VNF instance dc_1 is as below.

```
vnfc_ids:
  blogserver:
    - hostname: 192.168.10.1
      server_name: node-1
      vnfc_instance_id: 1
    - hostname: 192.168.10.2
      server_name: node-2
      vnfc_instance_id: 2
  nginx:
    - hostname: 192.168.10.1
      server_name: node-1
      vnfc_instance_id: 1
    - hostname: 192.168.10.2
      server_name: node-2
      vnfc_instance_id: 2
  postgres:
    - hostname: 192.168.10.4
      server_name: node-4
      vnfc_instance_id: 1
```

The following illustrates the topology described above:

Figure 4:



Note:

Openet Weaver does not support multiple VNFC instances with the same vnfc_id that originate from different VNF instances to share the same VM. Consider the following example based on two VNF instances: dc_1 and dc_2.

The vnf_topology file for dc_1 is as follows:

```
vnfc_ids:
  nginx:
    - hostname: 192.168.10.1
      server_name: node-1
      vnfc_instance_id: 1
  blogserver:
    - hostname: 192.168.10.2
      server_name: node-2
      vnfc_instance_id: 1
    - hostname: 192.168.10.3
      server_name: node-3
      vnfc_instance_id: 2
  postgres:
    - hostname: 192.168.10.4
      server_name: node-4
      vnfc_instance_id: 1
```

The vnf_topology file for dc_1 is as follows:

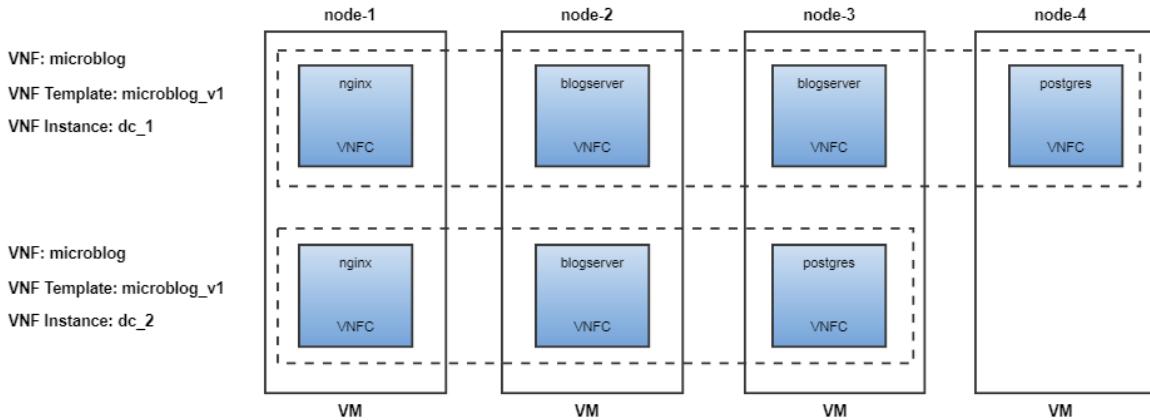
```
vnfc_ids:
  nginx:
    - hostname: 192.168.10.1
      server_name: node-1
      vnfc_instance_id: 1
  blogserver:
    - hostname: 192.168.10.2
      server_name: node-2
      vnfc_instance_id: 1
  postgres:
    - hostname: 192.168.10.3
      server_name: node-3
      vnfc_instance_id: 1
```

Using these topologies, when you attempt to instantiate dc_2 after instantiating dc_1, Openet Weaver generates an error along with the following message:

The VNFC Id "nginx" of VNF template "microblog_v1" has been staged for VNF instance "dc_1". Multiple instantiation of the same VNFC Id in the same target node from different VNF Instances is not supported.

The dc_1 and dc_2 topologies are illustrated below:

Figure 5:



Uploading a Topology File

To upload a topology file for a VNF package, use the command `ovlm-cm topology upload -t tenant_id -v vnf_id -i vnf_instance_id -f topology_file_path [-p vnf_template_id]`, for example:

```
ovlm-cm topology upload -t default -v webserver -i dublin -f vnf_topology.yml
-p httpServer_v1
```

After uploading the topology file it is good practice to [validate the VNFC configuration files](#) to ensure there are no errors that can cause a workflow run against the VNF to fail.

Note: When using the VNF-M GUI, uploading the topology file occurs as part of the instantiation process, not as part of creating an instance.

Instantiating a VNF from a VNF package

When you complete the tasks in this section you are ready to instantiate the VNF from the VNF package. Instantiation is described in detail in the [Managing VNFs](#) section of this documentation.

For instructions about how to instantiate a VNF that does not have VIM Configuration, see [Instantiating a VNF without VIM](#).

For instructions about how to instantiate a VNF that is configured for VIM, see [Instantiating a VNF with VIM](#).

Creating Custom Procedures with OVLM Python API

Openet Weaver provides a Python API library you can use to create custom procedures and scripts that can be invoked as part of a workflow. The library modules are contained in `ovlm-python-api`. It is meant to replace the out-of-the-box libraries that are installed by default and contained in `ovlm_lib`. To use the library, you need to download the package, install it, and migrate from the out-of-the box library to the `ovlm-python-api` library.

Resource Agent nodes have their own Python virtual environment for workflow execution. When Openet Weaver is deployed completely, `ovlm-python-api` is installed under the Resource Agent virtual environment binary path (path_bin). The path_bin location is specified in the installation file. See the Resource Agent section in [Creating an Installation File](#) for more information. For the custom procedure script to access the Openet Weaver Python Module in a Resource Agent, the path of the python interpreter need to be looked up automatically using env as in the following example.

```
#!/usr/bin/env python
<custom procedure script body>
```

If custom procedures require additional python libraries, those libraries must also be installed under the Resource Agent virtual environment. See [Adding Python Packages to the Resource Agent Python Virtual Environment](#) for more information.

Prerequisites

Before installing OVLM Python API, ensure a python environment is installed with a `jinja2` library. You can install the most recent `jinja2` version using `easy_install` or `pip` using one of the following commands:

- `easy_install Jinja2`
- `pip install Jinja2`

Running either of the commands installs a `Jinja2` egg in your Python installation's site-packages directory.

Note: `Jinja2` is an internal dependency for OVLM Python API.

Downloading the RPM

OVLM Python API is packaged with Weaver

1. Extract the TAR files by entering the following command:

```
tar -xvf ovlm-core-install.tar
```

Extracting the files create a directory named ovlm-install

2. Change directory to the ovlm-install directory, where the extracted files are located, by entering the following command:

```
cd ovlm-install
```

3. Change directory to the rpms directory by entering the following command:

```
cd rpms
```

4. Confirm that the OVLM python API exists and is named ovlm-python-api-1.0.0-dev.x86_64.rpm.

Installing the RPM

To install the ovlm-python-api rpm, use the command `sudo rpm -ivh <rpm_name>` as in the following example:

```
sudo rpm -ivh ovlm-python-api-1.0.0-dev.x86_64.rpm
```

During installation the modules are placed under your Python installation's site-packages directory. The library structure is shown below.

```
procedure-python-api
└── ovlm
    ├── __init__.py
    ├── constants.py
    ├── jinja_template.py
    ├── linux_shell.py
    ├── logger.py
    ├── parameters.py
    ├── software_packages.py
    └── systemd_services.py
```

Figure 31: OVLM Python API directory structure.

Installing ovlm-python-api in a User-Defined Location

By default, the ovlm-python-api RPM is installed under the `/usr` directory. To install the RPM to a different location, use the `relocate` parameter in the command `sudo rpm <rpm_name>--relocate /usr=<new location>`

The following example installs ovlm-python-api under the `/opt/ovlm/python-api` directory.

```
$ sudo rpm -ivh ovlm-python-api-1.1.3-dev.x86_64.rpm --relocate
/usr=/opt/ovlm/python-api
```

The following is an example of the contents of the `/opt/ovlm/python-api` directory after running the `rpm` command.

```
$ ls -ltr /opt/ovlm/python-api/lib/python2.7/site-packages/ovlm
total 192
-rw-r--r-- 1 root root 5839 Jul 13 14:18 systemd_services.py
-rw-r--r-- 1 root root 10093 Jul 13 14:18 software_packages.py
-rw-r--r-- 1 root root 326 Jul 13 14:18 parameters_testing.py
-rw-r--r-- 1 root root 4697 Jul 13 14:18 parameters.py
-rw-r--r-- 1 root root 7421 Jul 13 14:18 logger.py
-rw-r--r-- 1 root root 2229 Jul 13 14:18 linux_shell.py
-rw-r--r-- 1 root root 12380 Jul 13 14:18 jinja_template.py
-rw-r--r-- 1 root root 0 Jul 13 14:18 __init__.py
-rw-r--r-- 1 root root 725 Jul 13 14:18 constants.py
-rw-r--r-- 2 root root 5318 Jul 13 14:18 systemd_services.pyo
-rw-r--r-- 2 root root 5318 Jul 13 14:18 systemd_services.pyc
```

```

-rw-r--r-- 2 root root 10107 Jul 13 14:18 software_packages.pyo
-rw-r--r-- 2 root root 10107 Jul 13 14:18 software_packages.pyc
-rw-r--r-- 2 root root 876 Jul 13 14:18 parameters_testing.pyo
-rw-r--r-- 2 root root 876 Jul 13 14:18 parameters_testing.pyc
-rw-r--r-- 2 root root 5239 Jul 13 14:18 parameters.pyo
-rw-r--r-- 2 root root 5239 Jul 13 14:18 parameters.pyc
-rw-r--r-- 2 root root 7117 Jul 13 14:18 logger.pyo
-rw-r--r-- 2 root root 7117 Jul 13 14:18 logger.pyc
-rw-r--r-- 2 root root 2075 Jul 13 14:18 linux_shell.pyo
-rw-r--r-- 2 root root 2075 Jul 13 14:18 linux_shell.pyc
-rw-r--r-- 2 root root 11443 Jul 13 14:18 jinja_template.pyo
-rw-r--r-- 2 root root 11443 Jul 13 14:18 jinja_template.pyc
-rw-r--r-- 2 root root 136 Jul 13 14:18 __init__.pyo
-rw-r--r-- 2 root root 136 Jul 13 14:18 __init__.pyc
-rw-r--r-- 2 root root 829 Jul 13 14:18 constants.pyo
-rw-r--r-- 2 root root 829 Jul 13 14:18 constants.pyc
drwxr-xr-x 2 root root 77 Jul 26 11:02 licence

```

Python API Modules Description

The following table describes the modules.

Note: See the Openet Weaver API reference for more information about the APIs in the Python modules.

Table 35: OVLM Python API Modules

Package/File	Description	Python Class	Original Source Code
systemd_services.py	The source for the APIs that manage systemd services such as systemctl, start, and stop.		utilities/systemd_services.py
software_packages.py	The source for the APIs that manage rpm,debian packages.		software_packages/rpm.py, software_packages/debian.py, software_packages/rpackage_utils.py
parameters.py	The source for the APIs that manage input parameters for Openet Weaver procedures.	Params	utilities/ parameters.py
ovlm	The top level parent package. ovlm is chosen based on source naming in existing java packages in the Openet Weaver code base.		
logger.py	The source for the APIs used in logging by Openet Weaver procedures.		utilities/logger.py
linux_shell.py	The source for the APIs used in shell commands such as running the systemctl stop, start, or mkdir.		utilities/application_utils.py
jinja_template.py	The source for the APIs that manage jinja template files.		jinja/fileutils.py

Package/File	Description	Python Class	Original Source Code
constants.py	The source for constant value definitions.		utilities/ constants.py
__init__.py	The Python standard source file that specifies the package.		

Referencing Configuration Manager Out-of-the-Box Procedures

Model your custom out-of-the-box scripts on the Configuration Manager out-of-the-box Procedures. To reference these you must first install Configuration Manager within the Openet Weaver library.

Post installation, navigate to the cm repository root directory using the following command:

```
cd /usr/lib64/ovlm/cm/repository-root/ootb_procedures
```

The skeleton Out-of-the-Box code is shown below.

```
[ovlminstall@kl-005162-vm01 ~]$ cd /usr/lib64/ovlm/cm/repository-root/
[ovlminstall@kl-005162-vm01 repository-root]$ cd ootb_procedures/
[ovlminstall@kl-005162-vm01 ootb_procedures]$ ls -lrt
total 40
-rwxr-xr-x. 1 ovlm ovlm 1473 Mar 15 04:26 upgrade_vnfc_v2.py
-rwxr-xr-x. 1 ovlm ovlm 1028 Mar 15 04:26 update_config_vnfc_v3.py
-rwxr-xr-x. 1 ovlm ovlm 1108 Mar 15 04:26 stop_vnfc_v2.py
-rwxr-xr-x. 1 ovlm ovlm 1109 Mar 15 04:26 start_vnfc_v2.py
-rwxr-xr-x. 1 ovlm ovlm 1188 Mar 15 04:26 is_vnfc_up_v2.py
-rwxr-xr-x. 1 ovlm ovlm 1195 Mar 15 04:26 is_vnfc_down_v2.py
-rwxr-xr-x. 1 ovlm ovlm 2069 Mar 15 04:26 install_vnfc_v2.py
-rwxr-xr-x. 1 ovlm ovlm 1110 Mar 15 04:26 heal_vnfc_v2.py
-rwxr-xr-x. 1 ovlm ovlm 1579 Mar 15 04:26 dynamic_config_vnfc_v2.py
-rwxr-xr-x. 1 ovlm ovlm 2405 Mar 20 09:34 uninstall_vnfc_v2.py
```

Figure 32: Skeleton Out-of-the-Box Code

OVLM Python API Usage Examples

The following is an example of the is_vnfc_up_v2.py procedure. It invokes the systemd_services API to show how you can get service status and exit from the system if the service is up or down. The script imports the ovlm.constants library for the health status enum that is used to return the status at exit.

```
#!/usr/bin/env python

import sys
import logging
from ovlm import systemd_services
from ovlm.constants import HealthStatus
from ovlm.parameters import Params
from ovlm import logger

if __name__ == "__main__":
    if len(sys.argv) < 2:
        sys.exit('Usage: ' + __file__ + ' YAML_CONFIG_PATH')
    PARAMS = Params(sys.argv[1])
    logging_path = PARAMS.get_param('logging_path')
    logging_level = PARAMS.get_param('logging_level')
    logger.initialize_from_params(PARAMS)
```

```

logger.write(PARAMS.get_dump())
ALLPARAMS = PARAMS.get_all_params()
workflow_type = ALLPARAMS['request'].get('workflow_type')
logging.info('Action triggered for workflow_type: "%s"', workflow_type)
services = PARAMS.get_param('services')
systemd_services.check_services_up(services)

###Generic way to check the status of services to find if they are up/down

detected_down_service = False
service_status = systemd_services.check_services_status(services)
for service, status in service_status.items():
    if status:
        logging.info("Service %s is up." % service)
    else:
        logging.warn("Service %s is down." % service)
        detected_down_service = True
if detected_down_service:
    logging.error("Down services were detected")
    sys.exit(HealthStatus.DOWN)

```

The following is an example of the dynamic_config_vnfc_v2.py procedure. You can perform all template-specific actions by invoking the jinja_template API as shown below.

```

#!/usr/bin/env python

import sys
from ovlm.parameters import Params
from ovlm import logger
from ovlm import jinja_template

if __name__ == "__main__":
    if len(sys.argv) < 2:
        sys.exit('usage: ' + __file__ + ' <PARAMETERS_PATH>')

PARAMS_OBJ = Params(sys.argv[1])
logging_path = PARAMS_OBJ.get_param('logging_path')
logging_level = PARAMS_OBJ.get_param('logging_level')
logger.initialize_from_params(PARAMS_OBJ)
logger.write(PARAMS_OBJ.get_dump())
CONTEXT = PARAMS_OBJ.get_param('template_context')
jinja_template.perform(PARAMS_OBJ)

```

Migrating to the OVLM Python Modules

To migrate from default out-of-the-box scripts to the Python modules, reinstall Configuration Manager as a post installation task as described in [Installing Openet Weaver](#). After reinstalling, the ootb_procedure directory in /cm/repository-root looks similar to the following.

```
-rwxr-xr-x. 1 ovlm ovlm 1473 Mar 15 04:26 upgrade_vnfc_v2.py
-rwxr-xr-x. 1 ovlm ovlm 1028 Mar 15 04:26 update_config_vnfc_v3.py
-rwxr-xr-x. 1 ovlm ovlm 1108 Mar 15 04:26 stop_vnfc_v2.py
-rwxr-xr-x. 1 ovlm ovlm 1109 Mar 15 04:26 start_vnfc_v2.py
-rwxr-xr-x. 1 ovlm ovlm 1188 Mar 15 04:26 is_vnfc_up_v2.py
-rwxr-xr-x. 1 ovlm ovlm 1195 Mar 15 04:26 is_vnfc_down_v2.py
-rwxr-xr-x. 1 ovlm ovlm 2069 Mar 15 04:26 install_vnfc_v2.py
-rwxr-xr-x. 1 ovlm ovlm 1110 Mar 15 04:26 heal_vnfc_v2.py
-rwxr-xr-x. 1 ovlm ovlm 1579 Mar 15 04:26 dynamic_config_vnfc_v2.py
-rwxr-xr-x. 1 ovlm ovlm 2405 Mar 20 09:34 uninstall_vnfc_v2.py
```

Figure 33: ootb_procedure Directory After Migration

Note: The ovlm_lib directory is no longer available in this location. You must verify that all references to ovlm_lib in their out-of-the-box scripts have been replaced with calls to the OVLM Python library. Refer to the section on Skeleton Code for more information.

Adding Python Packages to the Resource Agent Python Virtual Environment

When you need to add additional python packages to support custom procedures, you must add those packages to the Resource Agent virtual environment.

To add a package into the virtual environment follow this procedure. The procedure uses urllib3 as the package name.

1. Login to the Resource Agent node as the service owner user.

Note: The default service owner is ovlm(default: ovlm).

2. Activate the virtual environment as shown in the example below.

In this example, the virtual environment is located at /usr/lib64/ovlm/ra/python_virtualenv.

```
$ ssh ovlm@node-1
ovlm@node-1's password:
Last login: Wed Jul 26 10:08:50 2017 from 192.168.122.1
[ovlm@node-1 ~]$ source /usr/lib64/ovlm/ra/python_virtualenv/bin/activate
(python_virtualenv) [ovlm@node-1 ~]$
```

3. Install the new package using command `pip install <package name>`. The following is an example.

```
(python_virtualenv) [ovlm@node-1 ~]$ pip install urllib3
Collecting urllib3
  Using cached urllib3-1.22-py2.py3-none-any.whl
    Installing collected packages: urllib3
      Successfully installed urllib3-1.22
```

Listing VNF Resources

To review or verify the resources contained in a specified Openet Weaver Configuration Manager repository, you can run the `resource list` CLI command.

```
ovlm-cm resource list -y resource_type -p resource_path
```

Note: Be sure the baseurl environment variables are set before using CLI commands. There is also an API version of this command.

The command lists the specified resources and their resource creation timestamps. The following table describes the resource types that can be listed within a VNF workspace using this command and their paths. [Table 2](#) describes the resource types that can be listed within a VNF package.

Table 36: Resource Types within a VNF Workspace

Resource type	Resource Path	Description	Example
tenant	Resource path is not required for this resource type.	Lists all tenants in the repository.	N/A
vnf	/tenants/ <i>tenant_id</i>	Lists all VNF within the specified tenant.	/tenants/anycom
vnfc	/tenants/ <i>tenant_id</i> /vnfs/ <i>vnf_id</i>	Lists all VNF Components within the specified tenant and VNF.	/tenants/anycom/vnfs/webserver
vnf_metadata	/tenants/ <i>tenant_id</i> /vnfs/ <i>vnf_id</i>	Lists VNF metadata file information for the specified tenant and VNF.	/tenants/anycom/vnfs/webserver
vnfc_configuration	/tenants/ <i>tenant_id</i> /vnfs/ <i>vnf_id</i> /vnfc/ <i>vnfc_id</i>	Lists all VNFC Configuration files within the specified tenant, VNF, and VNF Component.	/tenants/anycom/vnfs/webserver/vnfc/httpd
vnfc_package	/tenants/ <i>tenant_id</i> /vnfs/ <i>vnf_id</i> /vnfc/ <i>vnfc_id</i>	Lists all VNFC Package files within the specified tenant, VNF, and VNF Component.	/tenants/anycom/vnfs/webserver/vnfc/httpd
vnfc_procedure	/tenants/ <i>tenant_id</i> /vnfs/ <i>vnf_id</i> /vnfc/ <i>vnfc_id</i>	Lists all VNFC Procedure files within the specified tenant, VNF, and VNF Component.	/tenants/anycom/vnfs/webserver/vnfc/httpd
vnfc_metadata	/tenants/ <i>tenant_id</i> /vnfs/ <i>vnf_id</i> /vnfc/ <i>vnfc_id</i>	Lists VNFC metadata file information for the specified tenant and VNF.	/tenants/anycom/vnfs/webserver/vnfc/httpd
vnf_instance_topology	/tenants/ <i>tenant_id</i> /vnfs/ <i>vnf_id</i> /vnf_instances/ <i>vnf_instance_id</i>	Lists VNF Instance Topology file information for the specified tenant, VNF, and VNF instance.	/tenants/anycom/vnfs/webserver/vnf_instances/dublin
vnfc_policy	/tenants/ <i>tenant_id</i> /vnfs/ <i>vnf_id</i> /vnfc/ <i>vnfc_id</i>	List all policy files within the specified tenant, VNF, and VNF Component.	/tenants/anycom/vnfs/webserver/vnfc/httpd
vnfc_mib	/tenants/ <i>tenant_id</i> /vnfs/ <i>vnf_id</i> /vnfc/ <i>vnfc_id</i>	List all MIB files within the specified tenant, VNF, and VNF Component.	/tenants/anycom/vnfs/webserver/vnfc/httpd

Resource type	Resource Path	Description	Example
vnf_template	/tenants/ <i>tenant_id</i> /vnfs/vnf_id	List all VNF packages within the specified tenant and VNF.	/tenants/anycom/vnfs/webserver
vnfd	/tenants/ <i>tenant_id</i> /vnfs/vnf_id	List all VNF Descriptors created within the specified tenant and VNF.	/tenants/anycom/vnfs/webserver
vnfd_file	/tenants/ <i>tenant_id</i> /vnfs/vnf_id/vnfd/vnfd_id	List the VNFD file (vnfd.yml) within the specified tenant, VNF, and VNF Descriptor.	/tenants/anycom/vnfs/webserver/vnfd/preprod
vnfd_metadata	/tenants/ <i>tenant_id</i> /vnfs/vnf_id/vnfd/vnfd_id	Retrieve the VNF metadata file information for the specified tenant, VNF, and VNF Descriptor.	/tenants/anycom/vnfs/webserver/vnfd/preprod

The resource types that can be listed within a VNF package are listed in the following table.

Table 37: Resource Types within a VNF package

Resource type	Resource Path	Description	Example
vnfc	/tenants/ <i>tenant_id</i> /vnfs/vnf_id /templates/vnf_template_id	Lists all VNF Components within the specified tenant, VNF, and VNF package.	/tenants/anycom/vnfs/webserver/templates/webserver_v1
vnf_instance	/tenants/ <i>tenant_id</i> /vnfs/vnf_id/templates/vnf_template_id	Lists all VNF Components within the specified tenant, VNF, and VNF package.	/tenants/anycom/vnfs/webserver/templates/webserver_v1
vnf_metadata	/tenants/ <i>tenant_id</i> /vnfs/vnf_id/templates/vnf_template_id	Lists all VNF Components within the specified tenant, VNF, and VNF package.	/tenants/anycom/vnfs/webserver/templates/webserver_v1
vnfc_configuration	/tenants/ <i>tenant_id</i> /vnfs/vnf_id/templates/vnf_template_id /vnfc/vnfc_id	Lists all VNFC Configuration files within the specified tenant, VNF, VNF package, and VNF Component.	/tenants/anycom/vnfs/webserver/templates/webserver_v1/vnfc/httpd
vnfc_package	/tenants/ <i>tenant_id</i> /vnfs/vnf_id/templates/vnf_template_id /vnfc/vnfc_id	Lists all VNFC Package files within the specified tenant, VNF, VNF package, and VNF Component.	/tenants/anycom/vnfs/webserver/templates/webserver_v1/vnfc/httpd

Resource type	Resource Path	Description	Example
vnfc_procedure	/tenants/ <i>tenant_id</i> /vnfs/vnf_id/templates/vnf_template_id/vnfc/vnfc_id	Lists all VNFC Procedure files within the specified tenant, VNF, VNF package, and VNF Component.	/tenants/anycom/vnfs/webserver/templates/webserver_v1/vnfc/httpd
vnfc_metadata	/tenants/ <i>tenant_id</i> /vnfs/vnf_id/templates/vnf_template_id/vnfc/vnfc_id	Lists VNFC metadata file information for the specified tenant VNF, VNF package, and VNF component.	/tenants/anycom/vnfs/webserver/templates/webserver_v1/vnfc/httpd
vnfc_policy	/tenants/ <i>tenant_id</i> /vnfs/vnf_id/templates/vnf_template_id/vnfc/vnfc_id	Lists VNFC policy file information for the specified tenant VNF, VNF package, and VNF component.	/tenants/anycom/vnfs/webserver/templates/webserver_v1/vnfc/httpd
vnfc_mib	/tenants/ <i>tenant_id</i> /vnfs/vnf_id/templates/vnf_template_id/vnfc/vnfc_id	Lists VNFC policy file information for the specified tenant VNF, VNF package, and VNF component.	/tenants/anycom/vnfs/webserver/templates/webserver_v1/vnfc/httpd
vnf_instance_topology	/tenants/ <i>tenant_id</i> /vnfs/vnf_id/templates/vnf_template_id/vnf_instances/vnf_instance_id	Lists VNF Instance Topology file information for the specified tenant, VNF, VNF package, and VNF instance.	/tenants/anycom/vnfs/webserver/templates/webserver_v1/vnf_instances/dublin
vnfd	/tenants/ <i>tenant_id</i> /vnfs/vnf_id/templates/vnf_template_id	List all VNF Descriptors created within the specified tenant, VNF, and VNF package.	/tenants/anycom/vnfs/webserver/templates/webserver_v1
vnfd_file	/tenants/ <i>tenant_id</i> /vnfs/vnf_id/templates/vnf_template_id/vnfd/vnfd_id	List the VNFD file (vnfd.yml) within the specified tenant, VNF, VNF package, and VNF Descriptor.	/tenants/anycom/vnfs/webserver/templates/webserver_v1/vnfd/preprod
vnfd_metadata	/tenants/ <i>tenant_id</i> /vnfs/vnf_id/templates/vnf_template_id/vnfd/vnfd_id	Retrieve the VNF metadata file information for the specified tenant, VNF, and VNF Descriptor.	/tenants/anycom/vnfs/webserver/templates/webserver_v1/vnfd/preprod

Note: Openet Weaver also provides commands for listing specific resources. These are:

- `ovlm-cm tenant list`—lists all the tenants in the repository.
- `ovlm-cm template list -t tenant_id -v vnf_id`—lists all VNF packages within the specified tenant and VNF.
- `ovlm-cm vnf list -t tenant_id`—lists all VNFs within the specified tenant
- `ovlm-cm vnfd list -t tenant_id -v vnf_id`—lists all VNF Descriptors created within the specified tenant and VNF
- `ovlm-cm vnf_instance list -t tenant_id`—lists all VNF instances created within the specified tenant

See the API and CLI ~References for more information about the commands.

The following is an example of the `resource list` command used to list all tenants int the repository:

```
ovlm-cm resource list -y tenant
```

the following is an example of the `resource list` used to list all VNFCs for the `vnf_123` VNF under the default tenant:

```
ovlm-cm resource list -y vnfc -p /tenants/default/vnfs/vnf_123
```

Running the command above results in the following, which lists the three VNFCs for `vnf_123`:

- db
- lb
- app

```
data:
method: GET
request: http://localhost:9090/api/v1/list/vnfc
resource_list:
- creation_time: '2016-04-06 18:56:56'
  resource_id: db
- creation_time: '2016-04-06 18:56:56'
  resource_id: lb
- creation_time: '2016-04-06 18:56:56'
  resource_id: app
  resource_path: /tenants/default/vnfs/vnf_123
  resource_type: vnfc
status: 200
```

Related concepts

[Setting Environment Variables](#) on page 133

[Openet Weaver VNF-M API Reference](#) on page 468

Openet Weaver REST API reference guides

[Openet Weaver VNF-M CLI Reference](#) on page 469

Openet Weaver CLI command reference guides

Understanding Dynamic Configuration

Openet Weaver allows software to be localized on a runtime node using dynamic configuration information.

Dynamic configuration is performed on the runtime nodes when the VNF is instantiated. It is implemented in any kind of ASCII configuration files using jinja2 format. Jinja2 is a template language that allows variable substitution within the files (for more information, see <http://jinja.pocoo.org/docs/dev/templates/>).

Openet Weaver works with dynamic configuration by executing the out of the box procedure `dynamic_config_vnfc`. This procedure takes all the configuration files specified in the VNFC metadata file under the configuration "dynam

`ic_config_vnc:parameters:configurations`, scans their contents and performs substitutions based on the jinja2 template language.

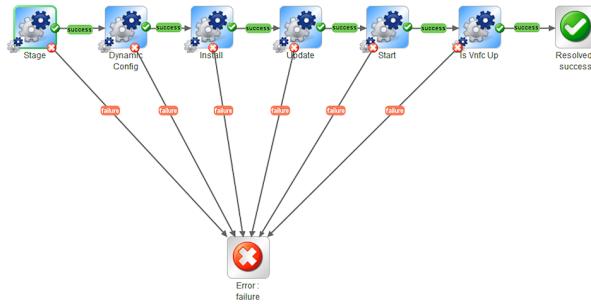
Configuration files must be saved with a `.jinja` extension, for example, `httpd.conf.jinja`. The names of the files for processing are supplied in the metadata file for the `dynamic_config_vnfc` procedure.

```
dynamic_config_vnfc:
  timeout: 60
  parameters:
    configurations:
      - configuration/httpd.conf.jinja
  ServerName: node-3
  procedure_name: dynamic_config_vnfc_v2.py
```

After the `dynamic_config_vnfc` procedure is executed to make the substitutions, the out of the box procedure `update_config_vnfc` must be executed to copy the localized files to their destination directory in the VNFC:

```
update_config_vnfc:
  parameters:
    - destFile: /etc/httpd/conf/httpd.conf
      group: root
      permission: '0644'
      srcFile: configuration/httpd.conf
      user: root
  procedure_name: update_config_vnfc_v2.py
```

Note: Notice that the out of the box `instantiate_vnf` workflow runs both procedures (`dynamic_config_vnfc` and `update_config_vnfc`) for each of the VNFCs configured in the VNF. These operations are represented in the diagram below by the "Dynamic Config" and "Update" boxes respectively. If you create a custom workflow and decide to perform dynamic configuration, it is your responsibility to implement the calls to these procedures correctly or to create your own custom procedures to do so.



There are four levels where dynamic information can be specified and made available to the `dynamic_config_vnfc` procedure:

- Request
- VNF Topology File
- VNFC Meta Data File
- Local Facts

Jinja2 syntax

The developer has a number of options for editing the configuration files, that is, substituting variables.

There are several types of delimiters syntax used by jinja. The default Jinja delimiters are:

- `{% ... %}` for statements.
- `{{ ... }}` for expressions to print to the template output.

- {# ... #} for comments not included in the template output.
- # ... ## for line statements.

Simple Variable Access

This example shows how to access a simple variable for use in a configuration file, where a dot (.) can be used to access attributes of a variable:

```
{{ request.tenant_id }}
{{ topology.vnfc_ids.blogserver.0.hostname }}
{{ metadata.ServerName }}
{{ facts.hostname }}
```

Variable Test

This example shows how the Template Developer can use conditions to define variable substitution rules:

```
{% if topology.vnfc_ids.blogserver.0.vnfc_instance_id == 1 %}
proxy_pass= http://{{ topology.vnfc_ids.blogserver.0.hostname }}:8080
{% endif %}
```

This results in:

```
proxy_pass= http://node-3:8080
```

Variable Looping

This example shows how the Template Developer uses the loop functionality to define variables within the configuration file.

```
{% for app in topology.vnfc_ids.blogserver %}
  <app_host>{{ app.hostname }}</app_host>
{% endfor %}
```

This results in:

```
<app_host>node-3</app_host>
<app_host>node-4</app_host>
```

Request

Request Parameters received over the REST API as attribute/value pairs in the body of the request are available to be used as dynamic configuration. To use dynamic configuration received as a request parameter refer to it as:

```
{{ request.<parameter_name> }}
```

For example, if you use the parameter tenant_id you can refer to it in the configuration jinja file as:

```
System={{ request.tenant_id }}
```

When using the Front End CLI workflow submit command, all the parameters passed to it are available, which are as follows:

- tenant_id
- vnf_id
- vnf_template_id
- vnf_instance_id

- vnf_id
- vnf_instance_id

For example:

```
ovlm-fe workflow submit -t default -v default_vnf -i sl_1 -y instantiate_vnf
-p '{"vnf_template_id": "microblog_v1", \
"vnfd_id": "vnfd_production"}'
```

IN the above example, the available parameters are:

- tenant_id = default
- vnf_id = default_vnf
- vnf_instance_id = sl_1

During the dynamic configuration step in the instantiate_vnf workflow the content in the configuration jinja file would be substituted with:

```
System=default
```

VNF Topology File

The VNF topology file is a YML file containing information about which Runtime Agent nodes holds each of the VNF Components in the VNF architecture. This information is available to be used as dynamic configuration. To use dynamic configuration based on the VNF topology file, refer to it as:

```
{{ topology.<path.to.the.attribute> }}
```

For example, using the following vnf_topology.yml file:

```
vnfc_ids:
  nginx:
    - hostname: node-2
      vnfc_instance_id: 1
  blogserver:
    - hostname: node-3
      vnfc_instance_id: 1
    - hostname: node-4
      vnfc_instance_id: 2
  postgres:
    - hostname: node-5
      vnfc_instance_id: 1
```

You can refer to the attributes in the configuration jinja file as:

```
DATABASE_HOST={{ topology.vnfc_ids.blogserver.0.hostname }}
```

During the dynamic configuration step in the instantiate_vnf workflow the content in the configuration jinja file would be substituted with:

```
DATABASE_HOST=node-3
```

VNFC Meta Data File

The VNFC topology file is a YML file containing metadata information used to map VNF life cycle events to the internal python procedures configured in weaver that executes the life cycle events and the related parameters. The parameters

under dynamic_config_vnfc VNF life cycle events are available to be used as dynamic configuration. To use dynamic configuration based on the VNFC metadata file, refer to it as:

```
{ { metadata.<parameter_name> } }
```

For example, using the following meta.yaml file:

```
dynamic_config_vnfc:
  timeout: 60
  parameters:
    configurations:
      - configuration/postgresql.conf.jinja
      - configuration/pg_hba.conf.jinja
      - configuration/microblog_create_db.sql.jinja
      - configuration/microblog_create_user.sql.jinja
    user: microblog
    pass: supersecret
    db: microblog
    ipv4_net: 0.0.0.0/0
    ipv6_net: ::/0
    listen_addresses: "*"
  procedure_name: dynamic_config_vnfc_v1.py
```

You can refer to the parameters in the configuration jinja file as:

```
DATABASE_PASSWORD={ { metadata.pass } }
```

During the dynamic configuration step in the instantiate_vnf workflow the content in the configuration jinja file would be substituted with:

```
DATABASE_PASSWORD=supersecret
```

Configuration

This refers to the Resource Agent configuration per installation. The available parameters are:

- repository_root
- logging_level
- template_context
- vnfc_root

Local Facts

Openet Weaver uses Facter to gather local facts about the runtime node and makes them available to the dynamic_config_vnfc VNF life cycle events when it is called, for example, IP address, FQDN, etc. More information on facts provided by Facter and a full list of the local facts can be found in https://docs.puppetlabs.com/facter/latest/core_facts.html. To use dynamic configuration based on the VNFC metadata file, refer to it as:

```
{ { facts.<fact_name> } }
```

For example you can refer to the local facts in the configuration jinja file as:

```
DATABASE_HOST={ { facts.hostname } }
```

During the dynamic configuration step in the instantiate_vnf workflow the content in the configuration jinja file would be substituted with (for example):

```
DATABASE_HOST=node-5
```

Validating VNFC Configuration Files

Openet Weaver can be used to trigger a preliminary validation of VNFC Ninja configuration files that are part of a VNF. This validation ensures that Ninja template variable substitution can be performed successfully when it is triggered as part of a workflow.

Important: Validation is performed on a "best effort" basis. Files that pass validation might still encounter failures when Ninja template variable substitution occurs during workflow execution. See [Known Limitations](#) section for more information.

Prerequisites

The following prerequisites must be met to validate VNFC configuration files for a VNF:

- A VNF package must be created
- At least one VNF topology must be uploaded for the VNF

Important: The only files that are validated during the process are those with the .jinja extension in the configuration section defined within the dynamic_config_vnfc procedure of the VNFC metadata.

Validating VNFC configuration files from the CLI

Use the following command to validate VNFC files from the Configuration Manager CLI

```
ovlm-cm template validate -t <tenant_id> -v <vnf_id> -i <vnf_instance_id> -p
<vnf_template_id>
```

A successful validation of the Ninja variables for a VNF package returns status code 200. If there are validation errors status code 400 is returned along with information that contains errmsg that can be divided into the following message severity levels:

Severity Level	Description
WARNING	<p>This level indicates that validation was performed and encountered unknown Ninja variables or invalid Ninja expressions that cannot be substituted in the template.</p> <p>You might see this message due to there being one or more undeclared variables from dynamic input sources such as request.workflow_type.</p>
ERROR	<p>This level indicates that validation on the file could not be completed successfully. In such a case the file was fully validated so it is possible that other Ninja variable or expression issues exist that were not found due to validation terminating early. Therefore you must fix all issues reported at this severity level and run validation again to ensure that there are no further ERROR messages that will cause dynamic configuration to fail during workflow execution.</p> <p>You might see this message if the Ninja file cannot be found, if there is invalid syntax or if there is a Ninja library fatal error during validation process.</p>

The following is an example of a successful response to running the template validate command:

```
status: 200
data:
  request: http://localhost:28010/api/v2/tenants/<tenant_id>/vnfs/<vnf_id>/te
  mplates/validate
  method: POST
```

The following is an example of a response with WARNING and ERROR severity level messages.

```

status: 400
data:
  request: http://localhost:28010/api/v2/tenants/<tenant_id>/vnfs/<vnf_id>/te
  mplates/validate
    method: POST
    error: The specified vnf template is invalid. Check on the ''template'' section in the response. Ignore any Jinja WARNING message due to dynamic input variable such as ''request.*''. Please refer to the user documentation for more details on validation logic.
    error_code: TEMPLATE_VALIDATION_FAILURE
    template:
      vnfc_ids:
        blogserver:
          - file_name: configuration/blogserver.conf.jinja
            error_msg:
              ERROR at line "24" "unexpected '}'"
        nginx:
          - file_name: configuration/nginx.conf.jinja
            error_msg:
              - WARNING Unknown Ninja variable/expression found at line "6"
"server.listen_address"

```

Known Limitations

The following are known limitations of the template validate command:

- L1— All input sources (Request, VNF Topology File, VNFC Meta Data File and Local Facts) for the Jinja files are validated. However, using any dynamic input source returns a WARNING message.
- L2—Any Ninja If - Else statement evaluation might be partially skipped depending on the evaluation of the if condition
- L3—When a Ninja list control structure For statement contains an invalid Ninja variable, the validation continues to evaluate statements within the For loop. Validation on the Ninja variable in the For statement is not performed. However, the validation indicates the non-existence of the derived variable in For statement.
- L4—A Ninja expression that performs an arithmetic operation will return an ERROR message when the wrong data-type value is used for the operation. For example, if subtraction is performed on a string object.
- L5—Applied, built-in filters that relate numeric operations such as abs, int, and round on Ninja variables are not validated.

Examples of the limitations of the template validate command follow:

Limitation Example 1

In this example, validation is performed on the following file set and then evaluated in the table that follows.

The contents of the blogserver/metadata/meta.yml file are as follows:

```

dynamic_config_vnfc:
  timeout: 60
  parameters:
    configurations:
      - configuration/blogserver.conf.jinja
    user: microblog
    pass: supersecret
    db: microblog
    dbport: 5432
    listen_address: 0.0.0.0
    listen_port: 80

```

The contents of the blogserver.conf.jinja file are as follows:

```

{# ----- Database connection information ----- #}
db_secondary_connection_string: {{ request.additional_parameters }}
db_primary_connection_string: user={{ metadata.user }} password={{ metadata.pass
}} dbname={{ metadata.db_test }} host={{ topology.vnfc_ids.postgres.0.hostname
}} port={{ metadata.dbport }}

{# ----- Web server connection information ----- #}
listen_address: {{ server.listen_address }}
{% if metadata.enabled_https %}
listen_port: {{ metadata.listen_port_https }}
{% else %}
listen_port: {{ metadata.listen_port }}
{% endif %}
admin_port: {{ metadata.admin_port + 20 }}
snmp_port: {% if metadata.listen_port is eq 80 %} {{ metadata.listen_port +
20 }} {% else %} {{ metadata.listen_port_https + 50 }} {% endif %}

{# ----- Web server monitoring information ----- #}
{% if metadata.listen_port is gt 100 %}
monitoring_user: {{ metadata.user }}
{% else %}
monitoring_user: {{ metadata.user_admin }}
{% endif %}
display_username: {% filter capitalize %} {{ metadata.user }} {% endfilter
%}

{# ----- Thread pool information ----- #}
thread_count: {{ metadata.thread_count|default(50) }}
predefined_users:
{% for user_name in metadata.predefined_user_list %}
    name: {{ user_name }}
{% endfor %}

```

The contents of the VNF topology file are as follows:

```

topology:
  vnfc_ids:
    blogserver:
      - hostname: 10.1.1.1
        server_name: blog1.testing.com
        vnfc_instance_id: 1
      - hostname: 10.1.1.2
        server_name: blog2.testing.com
        vnfc_instance_id: 2
    nginx:
      - hostname: 10.1.1.3
        server_name: ng.testing.com
        vnfc_instance_id: 1
    postgres:
      - hostname: 10.1.1.4
        server_name: postq.testing.com
        vnfc_instance_id: 1

```

when the template validation command is run against the above files the following response is returned:

```

status: 400
data:
  request: http://localhost:28010/api/v2/tenants/<tenant_id>/vnfs/<vnf_id>/te
  mplates/validate
  method: POST

```

```

error: The specified vnf template is invalid. Check on the ''template'' section in the response. Ignore any Jinja WARNING message due to dynamic input variable such as ''request.*''. Please refer to the user documentation for more details on validation logic.
error_code: TEMPLATE_VALIDATION_FAILURE
template:
vnfc_ids:
blogserver:
- file_name: configuration/blogserver.conf.jinja
error_msg:
- WARNING Unknown Jinja variable/expression found at line "2"
"request.additional_parameters"
- WARNING Unknown Jinja variable/expression found at line "3"
"metadata.db_test"
- WARNING Unknown Jinja variable/expression found at line "6"
"server.listen_address"
- WARNING Unknown Jinja variable/expression found at line "12"
"metadata.admin_port + 20"
- WARNING Unknown Jinja variable/expression found at line "19"
"metadata.user_admin"
- WARNING Unknown Jinja variable/expression found at line "27"
"user_name"

```

The error messages from the response are described in the table below:

Error	Explanation	Action to Resolve
WARNING Unknown Jinja variable/expression found at line "2" "request.additional_parameters"	Request variable is dynamic source that is populated in workflow execution. This variable is not defined when triggering the validation API. Related to L1 limitation.	No fix is required. This message can be ignored safely.
WARNING Unknown Jinja variable/expression found at line "3" "metadata.db_test"	metadata.db_test is a Jinja variable that is not declared in any of the source, for example the metadata or topology file.	You must correct the Jinja file or define the variable source.
WARNING Unknown Jinja variable/expression found at line "6" "server.listen_address"	server.listen_address is a Jinja variable that is not declared in any of the source, for example the metadata or topology file.	You must correct the Jinja file or define the variable source.
WARNING Unknown Jinja variable/expression found at line "12" "metadata.admin_port + 20"	metadata.admin_port is a Jinja variable that is not declared in any of the input source, for example the metadata or topology file. This causes validation failure on the expression.	You must correct the Jinja file or define the variable source.

Error	Explanation	Action to Resolve
WARNING Unknown Jinja variable/expression found at line "19" "metadata.user_admin"	<p>metadata.admin_user is a Jinja variable that is not declared in any of the input source, for example the metadata or topology file. This causes validation failure on the expression.</p> <pre>{% if metadata.listen_port is gt 100 %} monitoring_user: {{ metadata.special_user }} {% else %} monitoring_user: {{ metadata.user_admin }} {% endif %}</pre> <p>Note that expression {% if metadata.listen_port is gt 100 %} evaluated to false, thus the validation goes into the else block. Within the else block, metadata.user_admin is validated as non-existent. However, the validation did not warn about the non-existence of metadata.special_user.</p> <p>Related to limitation L2 .</p> <p>The same explanation applies to evaluating the following block, which is successful although variable metadata.enabled_https is not declared in any of the input source.</p> <pre>{% if metadata.enabled_https %} listen_port: {{ metadata.listen_port_https }} {% else %} listen_port: {{ metadata.listen_port }} {% endif %}</pre>	You must correct the Jinja file or define the variable source.
WARNING Unknown Jinja variable/expression found at line "27" "user_name"	<p>user is a Jinja variable which is part of the for loop expression derived by metadata.predefined_user_list. The real cause of the validation error is that metadata.predefined_user_list is not declared in any of the source, for example the metadata or topology file. Note that the Jinja variable with the For control statement is not returned as WARNING.</p> <p>Related to limitation L4.</p>	You must correct the Jinja file or define the variable source.

Limitation Example 2

This example uses the same metadata and topology files as example 1, however the following expression is modified to have an invalid Jinja name:

```
{% if metadata.listen_port is gt 100 %}
```

In this example, the expression Is changed to:

```
{% if metadata.listen_port is greater 100 %}
```

So now the blogserver.conf.jinja file is as follows:

```

{# ----- Database connection information ----- #}
db_secondary_connection_string: {{ request.additional_parameters }}
db_primary_connection_string: user={{ metadata.user }} password={{ metadata.pass
}} dbname={{ metadata.db_test }} host={{ topology.vnfc_ids.postgres.0.hostname
}} port={{ metadata.dbport }}

{# ----- Web server connection information ----- #}
listen_address: {{ server.listen_address }}
{% if metadata.enabled_https %}
listen_port: {{ metadata.listen_port_https }}
{% else %}
listen_port: {{ metadata.listen_port }}
{% endif %}
admin_port: {{ metadata.admin_port + 20 }}
snmp_port: {% if metadata.listen_port is eq 80 %} {{ metadata.listen_port +
20 }} {% else %} {{ metadata.listen_port_https + 50 }} {% endif %}

{# ----- Web server monitoring information ----- #}
{% if metadata.listen_port is greater 100 %}
monitoring_user: {{ metadata.user }}
{% else %}
monitoring_user: {{ metadata.user_admin }}
{% endif %}
display_username: {% filter capitalize %} {{ metadata.user }} {% endfilter
%}

{# ----- Thread pool information ----- #}
thread_count: {{ metadata.thread_count|default(50) }}
predefined_users:
{% for user_name in metadata.predefined_user_list %}
    name: {{ user_name }}
{% endfor %}

```

When the template validation command is run, the following response is returned:

```

status: 400
data:
  request: http://localhost:28010/api/v2/tenants/<tenant_id>/vnfs/<vnf_id>/te
mplates/validate
  method: POST
  error: The specified vnf template is invalid. Check on the ''template''
section in the response. Ignore any Jinja WARNING message due to dynamic input
variable such as ''request.*''. Please refer to the user documentation for
more details on validation logic.
  error_code: TEMPLATE_VALIDATION_FAILURE
  template:
    vnfc_ids:
      blogserver:
        - file_name: configuration/blogserver.conf.jinja
          error_msg:
            - ERROR at line "20" "no test named 'greater'"
```

The error message from the response is described in the table below:

Error	Explanation	Action to Resolve
ERROR at line "20" "no test named 'greater'"	"greater" is an invalid Jinja name. Note that other Jinja variables/expression within the file are not evaluated and exited immediately with the error. In addition, the reported line number is the end of the if statement containing the error.	Use a valid Jinja name in the Jinja file and re-run validation to check for other possible issues.

Limitation Example 3

This example uses the same metadata and topology files as example 1, however the following expression is modified so that there is a syntax error:

```
thread_count: {{ metadata.thread_count|default(50) }}
```

In this example, the expression is changed to:

```
thread_count: {{ metadata.thread_count|default(50) }}
```

So now the blogserver.conf.jinja file is as follows:

```
{# ----- Database connection information ----- #
db_secondary_connection_string: {{ request.additional_parameters }}
db_primary_connection_string: user={{ metadata.user }} password={{ metadata.pass
}} dbname={{ metadata.db_test }} host={{ topology.vnfc_ids.postgres.0.hostname
}} port={{ metadata.dbport }}

{# ----- Web server connection information ----- #
listen_address: {{ server.listen_address }}
{% if metadata.enabled_https %}
listen_port: {{ metadata.listen_port_https }}
{% else %}
listen_port: {{ metadata.listen_port }}
{% endif %}
admin_port: {{ metadata.admin_port + 20 }}
snmp_port: {% if metadata.listen_port is eq 80 %} {{ metadata.listen_port +
20 }} {% else %} {{ metadata.listen_port_https + 50 }} {% endif %

{# ----- Web server monitoring information ----- #
{% if metadata.listen_port is gt 100 %}
monitoring_user: {{ metadata.user }}
{% else %}
monitoring_user: {{ metadata.user_admin }}
{% endif %}
display_username: {% filter capitalize %} {{ metadata.user }} {% endfilter
%}

{# ----- Thread pool information ----- #
thread_count: {{ metadata.thread_count|default(50) }}
predefined_users:
{% for user_name in metadata.predefined_user_list %}
    name: {{ user_name }}
{% endfor %}
```

When the template validation command is run, the following response is returned:

```
status: 400
data:
  request: http://localhost:28010/api/v2/tenants/<tenant_id>/vnfs/<vnf_id>/te
mplates/validate
```

```

method: POST
error: The specified vnf template is invalid. Check on the ''template'' section in the response. Ignore any Jinja WARNING message due to dynamic input variable such as ''request.*''. Please refer to the user documentation for more details on validation logic.
error_code: TEMPLATE_VALIDATION_FAILURE
template:
vnfc_ids:
blogserver:
- file_name: configuration/blogserver.conf.jinja
error_msg:
- ERROR at line "24" "unexpected '}'"

```

The error message from the response is described in the table below:

Error	Explanation	Action to Resolve
ERROR at line "24" "unexpected '}'"	There is a Jinja syntax error due to a missing bracket ()).	Fix the missing bracket and re-run validation to check for other possible issues.

Limitation Example 4

This example uses the same metadata and topology files as example 1, however the following expression is modified so that there an attempt to perform a mathematical operation on a string:

```
thread_count: {{ metadata.thread_count|default(50) }}
```

In this example, the expression Is changed to:

```
thread_count: {{ metadata.user - 1 }}
```

So now the blogserver.conf.jinja file is as follows:

```

# ----- Database connection information -----
db_secondary_connection_string: {{ request.additional_parameters }}
db_primary_connection_string: user={{ metadata.user }} password={{ metadata.pass }} dbname={{ metadata.db_test }} host={{ topology.vnfc_ids.postgres.0.hostname }} port={{ metadata.dbport }}

# ----- Web server connection information -----
listen_address: {{ server.listen_address }}
{% if metadata.enabled_https %}
listen_port: {{ metadata.listen_port_https }}
{% else %}
listen_port: {{ metadata.listen_port }}
{% endif %}
admin_port: {{ metadata.admin_port + 20 }}
snmp_port: {% if metadata.listen_port is eq 80 %} {{ metadata.listen_port + 20 }} {% else %} {{ metadata.listen_port_https + 50 }} {% endif %}

# ----- Web server monitoring information -----
{% if metadata.listen_port is gt 100 %}
monitoring_user: {{ metadata.user }}
{% else %}
monitoring_user: {{ metadata.user_admin }}
{% endif %}
display_username: {{ filter capitalize }} {{ metadata.user }}  {% endfilter %}

# ----- Thread pool information -----

```

```

thread_count: {{ metadata.user -1 }}
predefined_users:
{% for user_name in metadata.predefined_user_list %}
    name: {{ user_name }}
{% endfor %}

```

When the template validation command is run, the following response is returned:

```

status: 400
data:
  request: http://localhost:28010/api/v2/tenants/<tenant_id>/vnfs/<vnf_id>/te
  mplates/validate
  method: POST
  error: The specified vnf template is invalid. Check on the ''template''
  section in the response. Ignore any Jinja WARNING message due to dynamic input
  variable such as ''request.*''. Please refer to the user documentation for
  more details on validation logic.
  error_code: TEMPLATE_VALIDATION_FAILURE
  template:
    vnfc_ids:
      blogserver:
        - file_name: configuration/blogserver.conf.jinja
          error_msg:
            - ERROR at line "357" "TypeError" "/usr/lib/python2.7/si
              te-packages/ovlm/jinja_template.py" "unsupported operand type(s) for -: 'str'
              and 'int'"

```

The error message from the response is described in the table below:

Error	Explanation	Action to Resolve
ERROR at line "357" "TypeError" "/usr/lib/python2.7/si te-packages/ovlm/jinja _template.py" "unsupported operand type(s) for -: 'str' and 'int'"	metadata.user is a string data type that has a mathematical operation applied to it. This causes Python runtime to fail with the "unsupported operand type(s) for -: 'str' and 'int'". This is related to limitation L4.	Fix the expression in theJinja file so that it does not have a mathematical operation applied to a string. Run validation again to check for other possible issues.

Limitation Example 5

This example uses the same metadata and topology files as example 1, however the following expression is modified so that there is an attempt to perform a Jinja filter operation on a Jinja variable:

```
thread_count: {{ metadata.thread_count|default(50) }}
```

In this example, the expression is changed to:

```
thread_count: {{ metadata.thread_count|int + 20 }}
```

So now the blogserver.conf.jinja file is as follows:

```

# ----- Database connection information -----
db_secondary_connection_string: {{ request.additional_parameters }}
db_primary_connection_string: user={{ metadata.user }} password={{ metadata.pass
}} dbname={{ metadata.db_test }} host={{ topology.vnfc_ids.postgres.0.hostname
}} port={{ metadata.dbport }}

# ----- Web server connection information -----

```

```

listen_address: {{ server.listen_address }}
{% if metadata.enabled_https %}
listen_port: {{ metadata.listen_port_https }}
{% else %}
listen_port: {{ metadata.listen_port }}
{% endif %}
admin_port: {{ metadata.admin_port + 20 }}
snmp_port: {% if metadata.listen_port is eq 80 %} {{ metadata.listen_port + 20 }} {% else %} {{ metadata.listen_port_https + 50 }} {% endif %}

{# ----- Web server monitoring information ----- #}
{% if metadata.listen_port is gt 100 %}
monitoring_user: {{ metadata.user }}
{% else %}
monitoring_user: {{ metadata.user_admin }}
{% endif %}
display_username: {% filter capitalize %} {{ metadata.user }} {% endfilter %}

{# ----- Thread pool information ----- #}
thread_count: {{ metadata.thread_count|int + 20 }}
predefined_users:
{% for user_name in metadata.predefined_user_list %}
  name: {{ user_name }}
{% endfor %}

```

When the template validation command is run, the following response is returned:

```

status: 400
data:
  request: http://localhost:28010/api/v2/tenants/<tenant_id>/vnfs/<vnf_id>/te
mplates/validate
  method: POST
  error: The specified vnf template is invalid. Check on the ''template'' section in the response. Ignore any Jinja WARNING message due to dynamic input variable such as ''request.*''. Please refer to the user documentation for more details on validation logic.
  error_code: TEMPLATE_VALIDATION_FAILURE
  template:
    vnfc_ids:
      blogserver:
        - file_name: configuration/blogserver.conf.jinja
          error_msg:
            - WARNING Unknown Jinja variable/expression found at line "2"
    "request.additional_parameters"
      - WARNING Unknown Jinja variable/expression found at line "3"
    "metadata.db_test"
      - WARNING Unknown Jinja variable/expression found at line "6"
    "server.listen_address"
      - WARNING Unknown Jinja variable/expression found at line "12"
    "metadata.admin_port + 20"
      - WARNING Unknown Jinja variable/expression found at line "19"
    "metadata.user_admin"
      - WARNING Unknown Jinja variable/expression found at line "27"
    "user_name"

```

The error message from the response is described in the table below:

Error	Explanation	Action to Resolve
No ERROR was reported on line 25 <code>thread_count: {{ metadata.thread_count int + 20 }}</code>	The built-inJinja filter related to a numeric operation is omitted from validation and therefore does not return as a WARNING although the variable <code>metadata.thread_count</code> is not defined in any input source. This is related to limitation L5.	Avoid using Jinja filters that perform numeric operations. Instead of using the filter on a known data type, perform the operation using a generic Jinja operation whenever possible, as in the following example: <code>thread_count: {{ metadata.thread_count + 20 }} </code>

Adding an Application User to Openet Weaver

You can add application users to Openet Weaver. Users should have sudo privileges because many commands require those privileges.

You can also configure Resource Agents in the installation file so that other commands require sudo privileges. When you customize sudo privileges in this way, you must assign those privileges to the application user and associate the user/privileges with a specific VNFC in order to run those commands. The following procedure shows how you can update the privileges for a sudo user to run a custom lifecycle event:

1. Create a python script to update user's sudo privileges.

In this example the python script is named `update_user_sudo_privileges.py` and assigns the sudo privileges defined in step 1 to the new sudo user as shown below.

```
#!/usr/bin/python
import sys
from ovlm_lib import utils
def create_sudo_user():
    """Start create a new sudo user """
    result = utils.run_shell_command("sudo useradd pcrf")
    if result.returncode != 0 or result.stderr:
        sys.exit('ERROR Unable to create a user: %s' %( result.stderr))
    utils.run_shell_command("echo pcrf | sudo passwd pcrf --stdin")
def update_user_sudo_privileges():
    """Start update sudo privileges """
    result = utils.run_shell_command('sudo bash -c \'echo \"pcrf ALL=(ALL) NOPASSWD: PCRF_CMD\nCmnd_Alias PCRF_CMD = /opt/pcrf/startApp\" | (EDITOR=\"tee -a\" visudo)\'')
    if result.returncode != 0 or result.stderr:
        sys.exit('ERROR Unable to update user sudo privileges: %s' %( result.stderr))
    else:
        sys.stdout.write('The user sudo privileges were updated successfully');
if __name__ == "__main__":
    create_sudo_user()
    update_user_sudo_privileges()
```

2. Upload the script to the repository using the CLI command `ovlm-cm vnfc_procedure upload -t tenant_id -v vnf_id -c vnfc_id -f update_user_sudo_privileges.py`
3. Create a new package for the related VNF using the CLI command `ovlm-cm template create -t tenant_id -v vnf_id -p vnf_template_id`
4. Instantiate the VNF using the package you created. For example `ovlm-fe workflow submit -t tenant_id -v vnf_id -i vnf_instance_id -y instantiate_vnf_without_vim -p '{vnf_template_id}': "template_id"'`

5. Run the custom lifecycle event using the command `ovlm-fe workflow submit -t tenant_id -v vnf_id -i vnf_instance_id -y custom_action_vnfc -p '{"hostname" : "hostname", "procedure_filename" : "update_user_sudo_privileges.py"}'`

Setting Up a Reference VNF

Openet Weaver provides the functionality to recreate a reference installation node from a working VNF template. The benefit of this feature is to simplify recreating a reference installation node in a new environment, for example from pre-production to a test or production environment. When you recreate a reference installation from a VNF template you are not required to reconfigure the jinja tokens in the reference installation node.

A reference VNF is a VNF template that was installed and configured for the sole purpose of creating templates from the VNFCs. For example, an Openet Policy VNF template that consists of Signaling Manager, Policy Manager and VoltDB.

A reference VNF for this solution has the entire solution installed and configured on a single node. The VNFCs from the reference VNF are then used to create a VNF template using Openet Weaver Integration Module (WIM). After the VNF template is created it can be used to run workflows for the VNF and VNFC instances that are used to implement the customer solution.

Note: For information about how to set up a reference VNF when there is no existing VNF template, see the section *Setting up a Reference Installation* in the *Weaver Integration Module* user guide.

After a reference VNF template is created, it can be used to create another instance of reference installation using the `instantiate_reference_vnf` workflow. The section describes how to create a reference VNF from an existing reference VNF template, which you might want to do for one or more of the following reasons:

1. To duplicate a multiple reference installation from one environment, so it can be installed in another environment for concurrent development and testing. For example, you can clone a reference VNF in a testing environment and install it in a production environment.
2. To move an existing reference VNF to a different node.
3. To set up a reference VNF on a customer site after testing it locally.

Before setting up a reference VNF installation using an existing VNF template, ensure that the `reference_vnf` block in the Openet Weaver installation properties file is configured properly. For more information about the Openet Weaver installation file, see [reference_vnf Settings](#).

Note: You must configure the installation file before installing Openet Weaver.

Before running the `instantiate_reference_vnf` workflow, you must configure the following for the `run_as` user in the Reference Installation Node:

- Passwordless sudo access
- SSH private key to allow password-less SSH access into Reference Installation Node

At this point you can install and configure a reference VNF installation using template by performing the following tasks:

1. Create a VNF Topology that refers to the new target node to install the reference installation
2. Instantiate the reference VNF using the `instantiate_reference_vnf` workflow and the topology created in task 1
3. Install the Weaver Integration Module on the node you are using for the reference VNF you are installing
4. If any changes were made to the WIM files in the initial reference VNF that produced the template you are using to install the new reference VNF, sync-up the WIM changes manually from initial WIM installation to the new node
5. Configure the reference VNF instance you are installing as required

For some use cases Openet Weaver must be installed on a new node. An example of this type of use case is when you are creating a reference VNF from a template in the local test environment and installing the new reference VNF on a customer-hosted virtual machine. In this example, the VNF template and any WIM files changes must be copied to the

customer site before you perform tasks 1 through 4. Then you must install Openet Weaver on the customer site before performing task 5.

Note: To see an illustrated version of this task list see [Illustrated Reference VNF Installation Tasks](#).

After completing the above tasks you can use the reference VNF to create a VNF template to be used with Openet Weaver lifecycle workflows as described in [Managing VNFs](#).

Important: The template from the initial reference VNF must be created using Weaver Integration Module v1.5 or higher for you to succeed with the process described in this section. This is because the following updates were made to the Weaver Integration Module v1.5 and Openet Weaver v1.11:

- Weaver Integration Module:
 - VNFC Metadata updated
 - Python procedure file updated
 - WIM python module updated
- Openet Weaver:
 - New VNF workflow— instantiate_reference_vnf
 - The following actions are now defined in metadata:
 - dynamic_config_reference_vnfc
 - update_config_reference_vnfc
 - install_reference_vnfc
 - post_install_reference_vnfc

Illustrated Reference VNF Installation Tasks

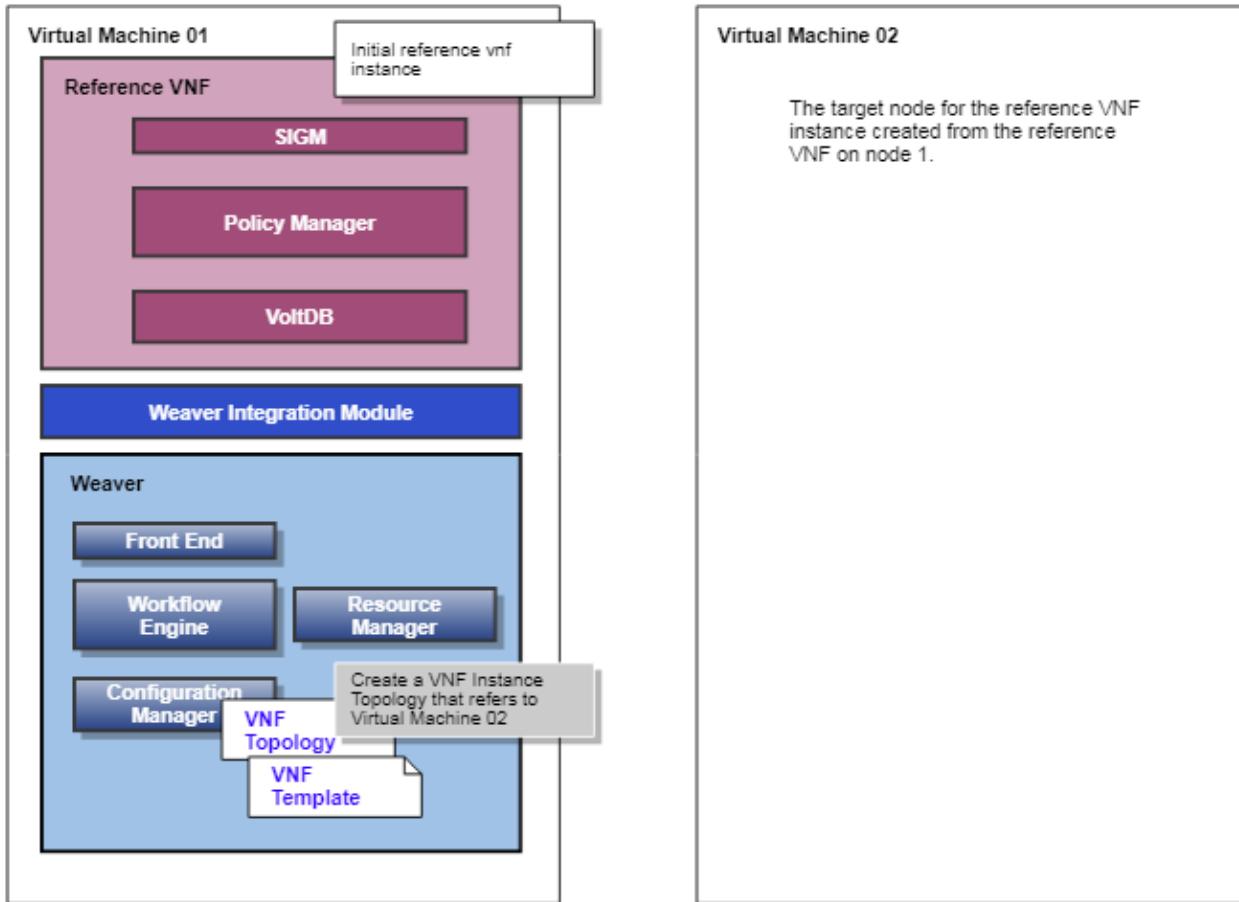


Figure 34: Task 1: Create a VNF Topology that refers to the new target node to install the reference installation

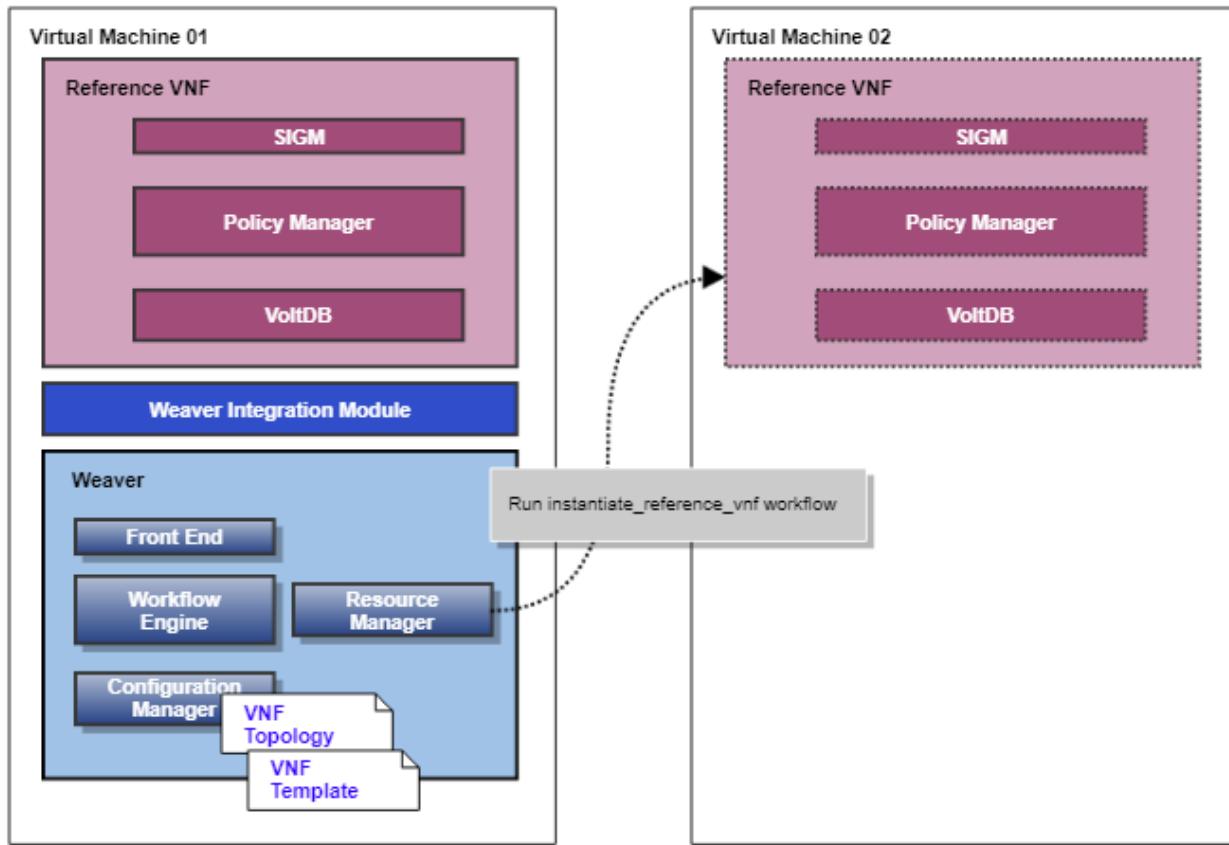


Figure 35: Task 2: Instantiate the Reference VNF using the instantiate_reference_vnf workflow and the topology created in step 1

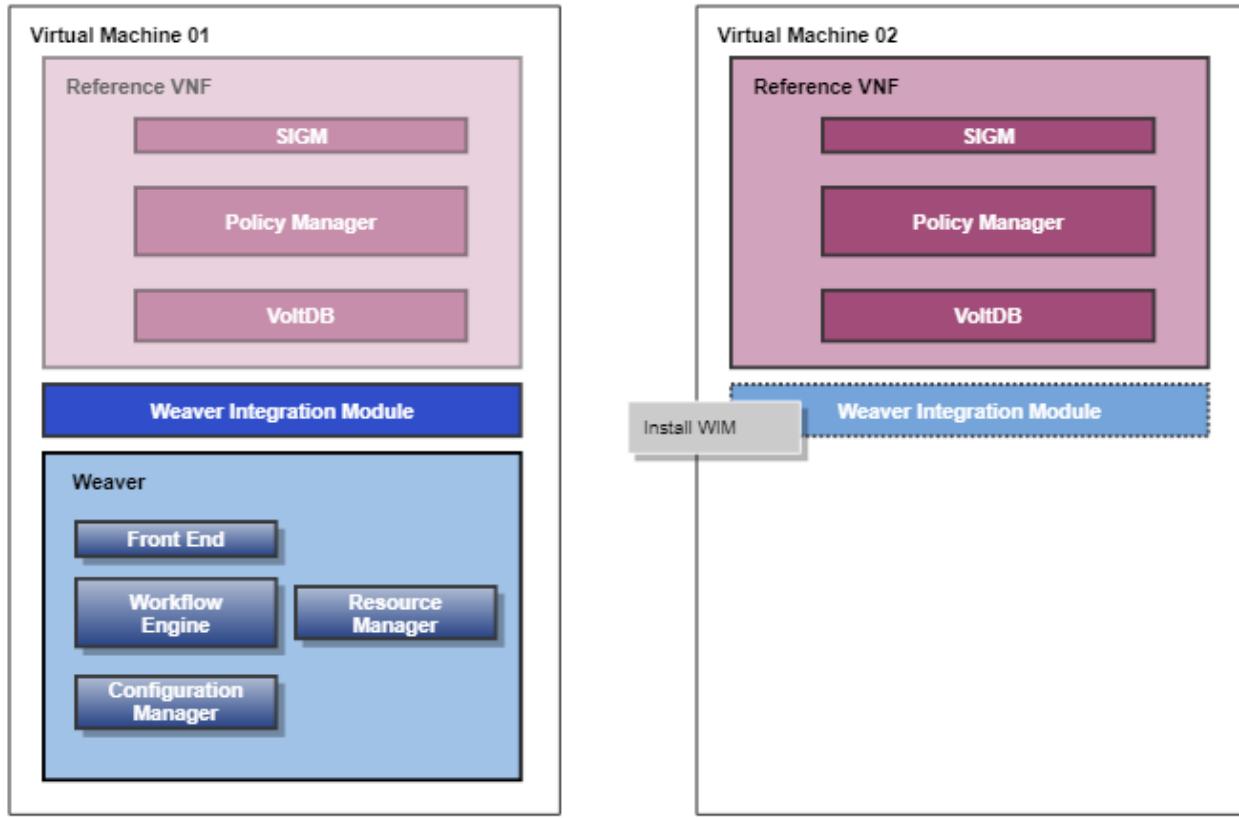


Figure 36: Task 3: Install the Weaver Integration Module on the node you are using for the reference VNF you are installing

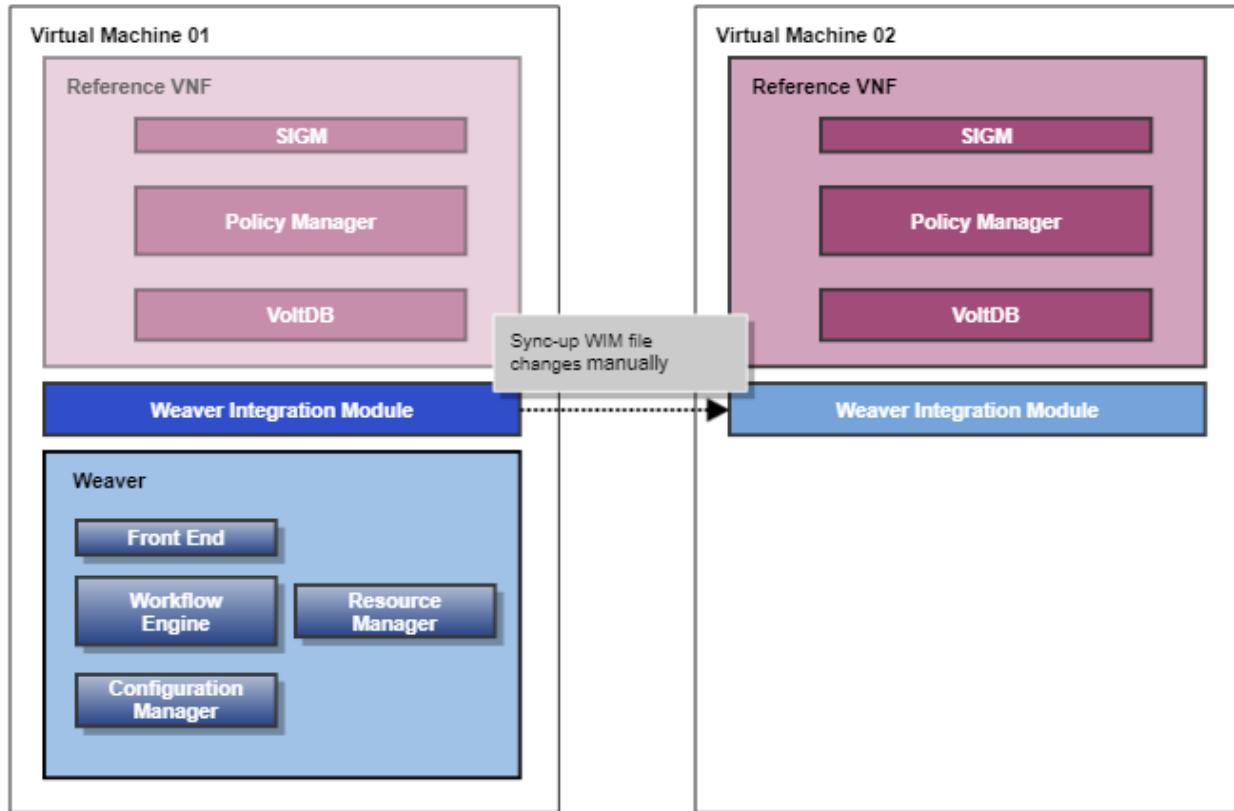


Figure 37: Task 4: If any changes were made to the WIM files in the initial reference VNF that produced the template you are using to install the new reference VNF, sync-up the WIM changes manually from initial WIM installation to the new node

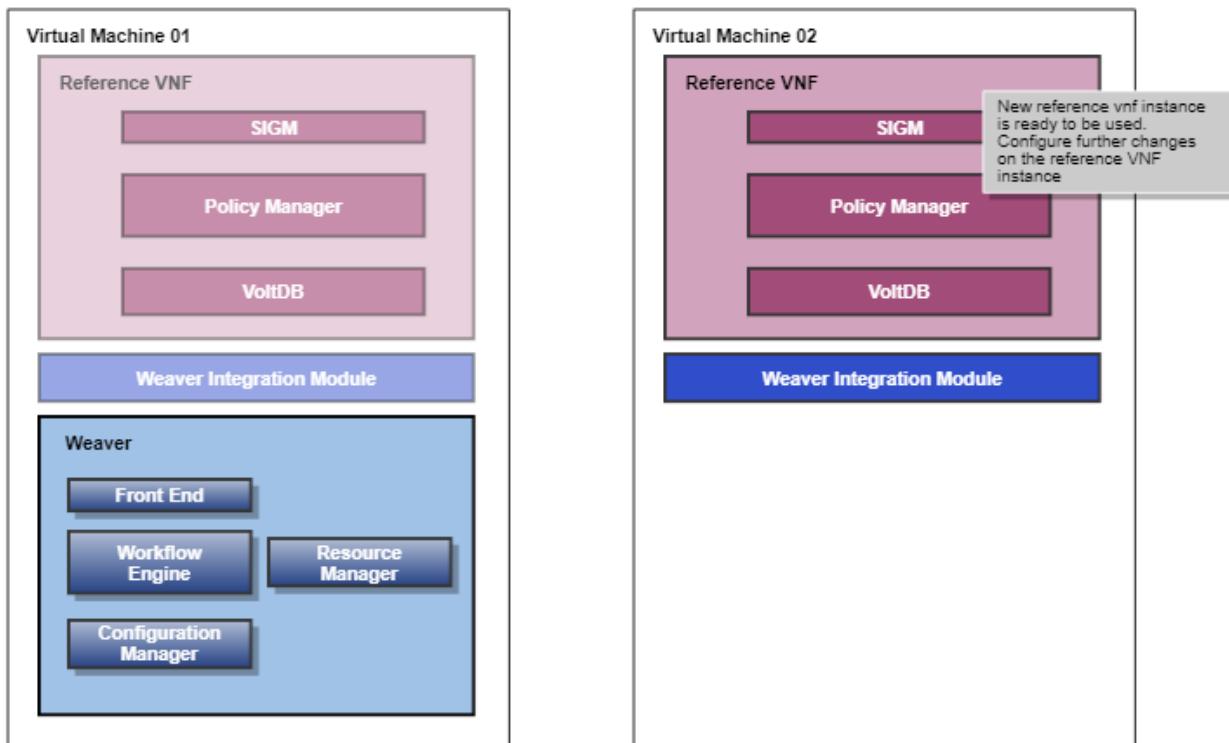


Figure 38: Task 5: Configure the reference VNF instance you are installing as required

After completing the above tasks you can use the reference VNF to create a VNF template to be used with Openet Weaver lifecycle workflows as described in [Managing VNFs](#)

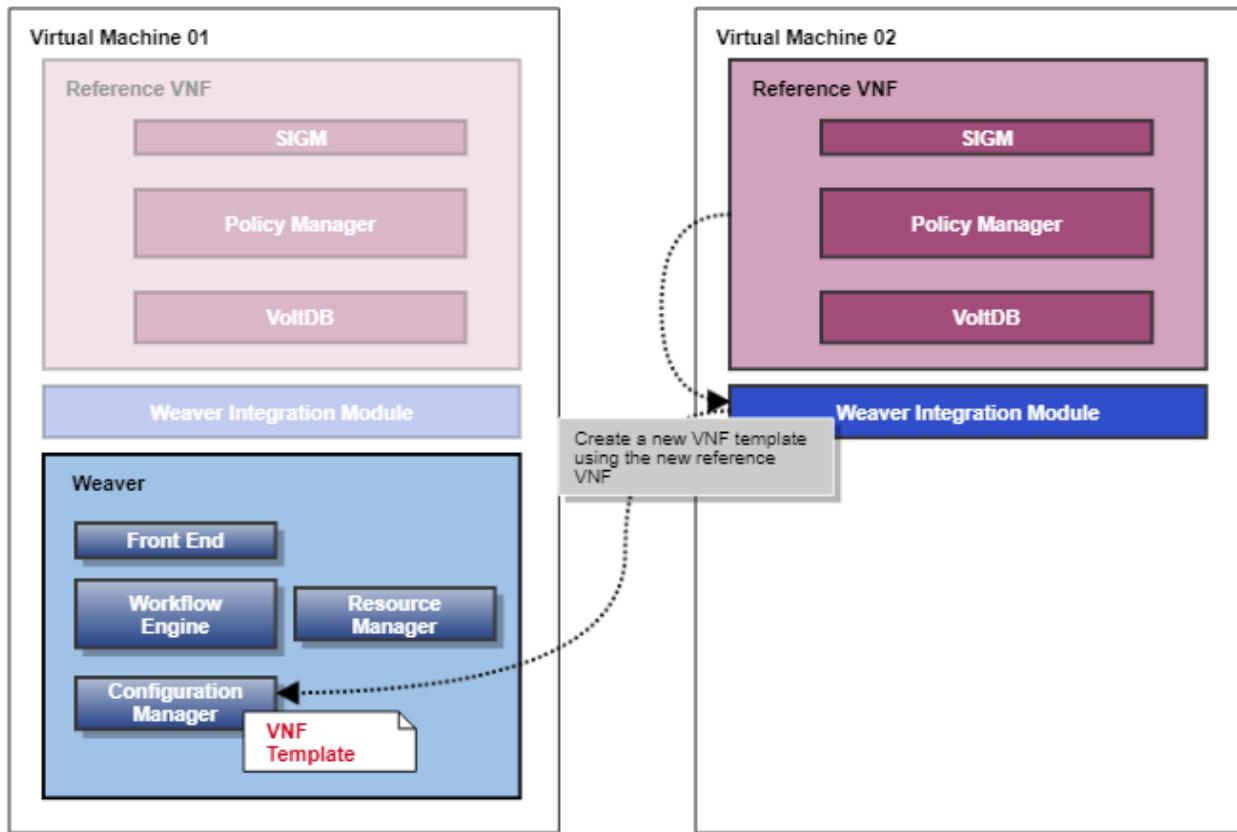


Figure 39:

Creating a VNF Topology File

You must create a topology file for the reference installation VNF. This file describes The VNFCs and identifies the target installation nodes to which they are deployed. For a reference installation VNF, it is possible to deploy all VNFCs to the same host.

The example shown below has all VNFCs has all VNFCs deployed to host 10.0.104.38. In such a case , the FUSIONWORKS_HOME and FUSIONWORKS_PROD directories of each installed product must be unique. This topology file is used with the instantiate_reference_vnf workflow.

```

vnfc_ids:
  policy:
    - hostname: "10.0.104.38"
      vnfc_instance_id: 1
    network:
      vlan_ne:
        private_ip: "10.0.104.1"
        private_hostname: "node1-appvml-ne"
      vlan_appdb:
        private_ip: "10.0.104.7"
        private_hostname: "node1-appvml-appdb"
      vlan_oam:
        private_ip: "10.0.104.38"
  
```

```

private_hostname: "node1-appvm1-oam"
sigm:
- hostname: "10.0.104.38"
  vnfc_instance_id: 1
volt:
- hostname: "10.0.104.38"
  vnfc_instance_id: 1
network:
  volt_internal:
    private_ip: "10.0.151.49"
  volt_external:
    private_ip: "10.0.151.49"

```

For more information about creating a topology file see [Creating and Uploading the VNF Topology File](#).

Instantiating a Reference VNF

When you instantiate a reference VNF, what you are doing is cloning an existing reference VNF from a VNF template. This makes the purpose of instantiation different from the instantiation workflows covered in the *Managing VNFs* section of this user guide.

You can instantiate a reference VNF from the CLI, the API, or the Workflow Engine GUI. A reference VNF is a VNF instance that is used to create a VNF package. The `instantiate_reference_vnf` workflow lets you clone an existing reference VNF from an existing VNF package that might be in another environment. For example, recreating a reference VNF from a testing environment to a Production environment.

Note:

Because it is used as a reference the start option is not applicable when running the workflow.

A successful reference VNF instantiation is shown below in the Workflow Engine GUI.

Step Name	Transition Message
Set up globals	success
Format Additional Parameters	success
Reference Vnrf Instantiation	success
Pre Installation	success
Stage Reference	success
Dynamic Config Reference	success
Install Reference	success
Update Config Reference	success
Post Installation	success
Resolved : success	
Format Success Response	success
Resolved : success	

Step Details

Resolved : success

Run ID: 138200190
Step ID: 72d0a074-5ed3-497d-a2d2-b8f39894cadb
Start Time: 2:48:46 PM
End Time: 2:48:46 PM
Response: N/A
Duration: 0 seconds
Inputs: successMessage [{"status":200,"data":{"success_msg":"Instantiation of Referen"}]}
Primary Result: N/A
Step Results: Result [{"status":200,"data":{"success_msg":"Instantiation of Referen"}]}
Worker Group: N/A
Worker ID:
Transition Message:
Step Persistence: Detailed

Figure 40: Reference VNF Instantiation

The `instantiate_reference_vnf` workflow is shown below.

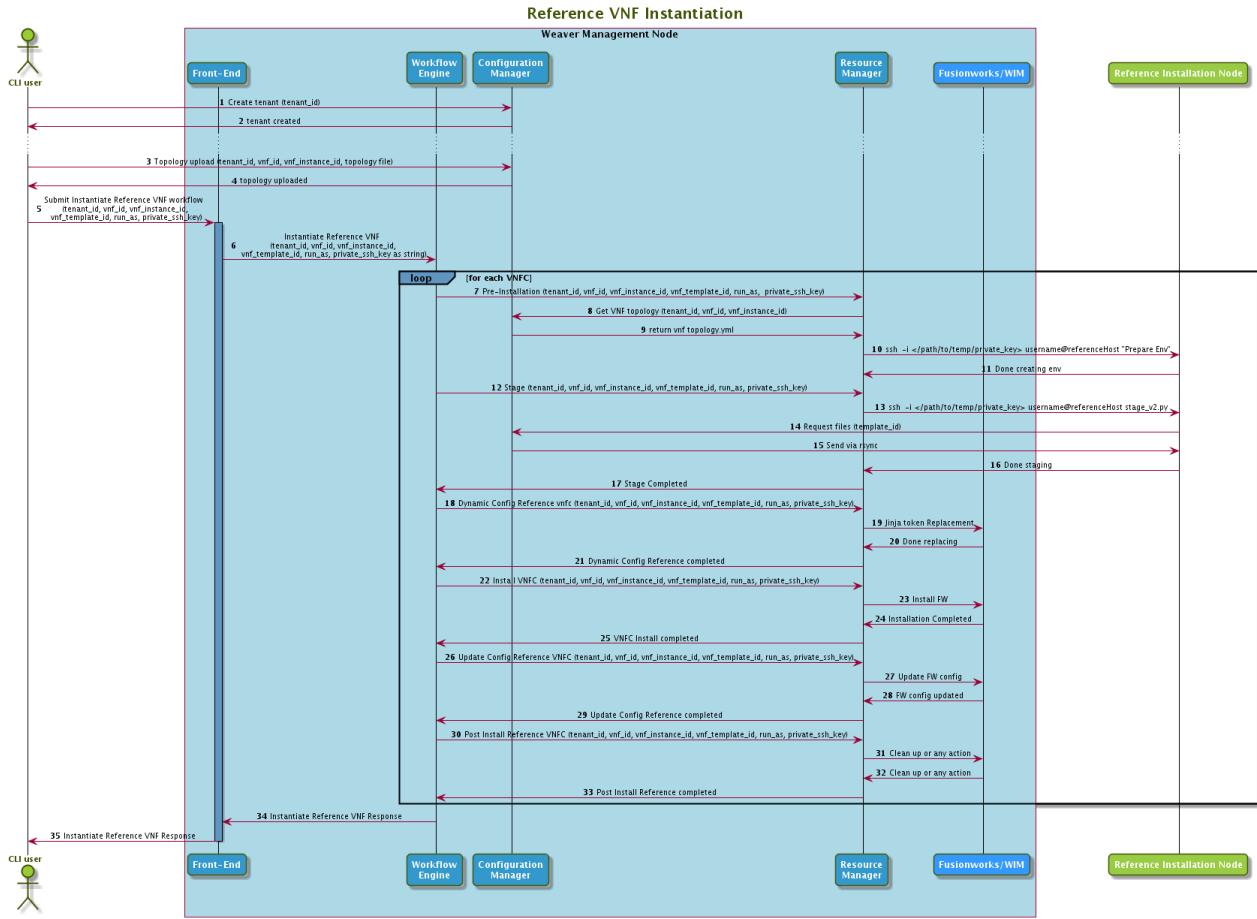


Figure 41: instantiate_reference_vnf Workflow

The following table describes the steps performed for each VNFC in the above diagram:

You use the `instantiate_reference_vnf` workflow to clone an existing reference VNF from a VNF template.

Note: The purpose of the `instantiate_reference_vnf` workflow is to reconstruct an existing VNF template from its previous location, for example from a test environment to a production environment. Therefore the workflow steps are different from other instantiation workflows, for example the `instantiate_vnf_without_vim` workflow.

The supported actions for the `instantiate_reference_vnf` workflow are described in the following table:

Step	Description
Pre-install	<p>Sets up the runtime node by performing the following tasks:</p> <ul style="list-style-type: none"> Creates a reference VNF repository directory structure Installs the Python virtual environment Installs the OVLM Python API in the virtual environment Prepares the secret key file for Rsync connectivity in Stage action <p>The location of the repository root directory can be configured using the <code>repository-root</code> property of the <code>reference_vnf</code> block in the installation file. See Creating an Installation File for more information.</p>
Stage	Transfers the VNF Template from the Configuration Manager microservice to the reference VNF Node.

Step	Description
Dynamic Config Reference	<p>Renders Jinja tokens in the VNFC configuration file. The corresponding configuration is the action type <code>dynamic_config_reference_vnfc</code> in VNFC metadata file.</p> <p>Non-jinja extension files are supported for dynamic configuration if the property <code>render_file_with_non_jinja_extension</code> is set to true in the metadata file. The property is used to indicate whether non-jinja extension files are to be jinjafy or rendered during the the Dynamic Config step. The default value is false.</p> <p>The following is an example of a metadata file with <code>render_file_with_non_jinja_extension</code> set to true:</p> <pre data-bbox="502 544 1383 756">dynamic_config_reference_vnfc: timeout: 60 parameters: configurations: - configuration/nginx.conf.jinja - configuration/nginx_vnfc_monitoring.sh.jinja render_file_with_non_jinja_extension: true procedure_name: dynamic_reference_config_vnfc.py</pre> <p>If configurations field value is empty (not defined), all files in the VNFC configuration directory with .jinja extensions are rendered.</p>
Update Config Reference	Applies the VNFC configuration to VNF working folders.
Post Install Reference	<p>Cleans up the environment and might perform other necessary actions.</p> <p>Note: Post install is an optional step. It is skipped and considered as sucessful if it's not defined in the VNFC metadata. Check the response from the Resource Managers in the Workflow Engine to verify the execution status of this step.</p>

Running the `instantiate_reference_vnf` Workflow

You instantiate a reference VNF by running the workflow submit parameter with the `instantiate_reference_vnf` workflow type. You can instantiate a reference VNF from the CLI, the API, or the Workflow Engine GUI.

The workflow submit parameters are described in the following table. They apply to the CLI an the API versions of the command.

Note: Only the paremeters applicable to the `reference_vnf` workflow type are shown. See [Running VNF Workflows](#) for parameters that apply to other workflow types.

Table 38: workflow submit Parameters

CLI Flag	Parameter	Mn	Description
-t	tenant_id	Yes	The tenant identifier as created in the Configuration Manager.
-v	vnf_id	Yes	The VNF identifier as created in the Configuration Manager.
-i	vnf_instance_id	Yes	The VNF instance identifier as created in the Configuration Manager.
-y	workflow_type	Yes	The Workflow type for the lifecycle event defined in the workflow. Use <code>instantiate_vnf</code> for instantiating a VNF.
-p	vnf_template_id	Yes	The unique identifier for the VNF package.

CLI Flag	Parameter	Mn	Description
-run_as	run_as	Yes	<p>Specifies the user account with which to instantiate the reference VNF on the reference node.</p> <p>This option is applicable only to the instantiate_reference_vnf workflow.</p> <p>This user must be already exist on the reference VNF node and have SSH passwordless sudo access to the VNF Reference node.</p> <p>The following commands must be available in the sudoers file for the run_as user.</p> <ul style="list-style-type: none"> • /bin/mkdir—create directory • /bin/rm—delete file • /bin/chown—change file or directory ownership • /bin/chmod—change file or directory permission • /usr/bin/mv—moving a file or a directory • /bin/bash—access to bash shell • /bin/echo—display message, used in connectivity check • /bin/rsync—file transfer • /usr/bin/ssh—runs the SSH client for remote login • /bin/rpm—builds, installs, queries, verifies, updates, and erases individual software packages • /usr/bin/yum—Performs package installation, removes old packages, and queries the installed and/or available packages, and can be used to perform many other commands and services <p>The following is an example of sudo access in the /etc/sudoers file:</p> <pre>Cmnd_Alias OVLM_REF_VNF = /bin/mkdir, /bin/rm, /bin/chown, /bin/chmod, /usr/bin/mv, \ /bin/bash, /bin/echo, /bin/rsync, /usr/bin/ssh, /bin/rpm, /usr/bin/yum <run_as_user> ALL=(ALL) NOPASSWD:SETENV: OVLM_REF_VNF Defaults:<run_as_user> !requiretty</pre>
-k	private_ssh_key	Yes	<p>Specifies the private SSH key for the user to log in to the target reference installation node. The SSH key-pair must be set up prior to running instantiate_reference_vnf. You set up the SSH key-pair between the node hosting the Resource Manager microservice and the target reference installation node.</p> <p>Applicable only for workflow instantiate_reference_vnf.</p> <p>For CLI commands, run_as_ssh_key points to the location of the private SSH key file to use.</p> <p>For REST API calls, run_as_ssh_key must be the private SSH key value encoded in base64 scheme.</p>
-sync	sync	No	<p>The optional sync parameter to specify whether the call is synchronous (1) or asynchronous (0). The call is asynchronous by default.</p>
-timeout	timeout	No	<p>The optional timeout parameter to specify how long the synchronous call waits for the operation to complete. The default value is 60 seconds.</p> <p>Note: The default timeout value can be specified in the front-end: synchronous-request-timeout attribute in the installation file and propagated to the Front-End microservice using the deploy script.</p>

CLI Flag	Parameter	Mn	Description
-a	additional_parameters	No	<p>Generic parameters in JSON format that are passed through the workflow submit command to the procedural level on Resource Agents. These parameters are used to extend the command's flexibility to support end-user scenarios. The potential impact on the command depends upon which workflow you are running.</p> <p>For example, when running the upgrade workflow, you can take a snapshot of the VNF instance before performing the upgrade. Openet Weaver does not have a defined workflow submit parameter for this action, but a <code>take_a_snapshot</code> parameter might be defined for the actual procedure, and the parameter-value pair can be passed through the workflow submit command to the Resource Agent for processing.</p> <p>You can predefine additional parameters for use with the VNF-M GUI. When you do, the predefined additional parameters display in the dialog box that opens when you select a workflow to run.</p> <p>Note: See Managing Additional Parameters for the VNF-M GUI for more information about predefining additional parameters.</p>

The following topics describe how to instantiate a reference VNF from the CLI or API:

Instantiating a Reference VNF from the CLI

To instantiate a reference VNF from the CLI, use the `instantiate_reference_vnf` `workflow_type` parameter with the command shown below.

```
ovlm-fe workflow submit -t <tenant_id> -v <vnf_id> -i <vnf_instance_id> -y
instantiate_reference_vnf -p <vnf_template_id>
-run_as <run_as_user> -k <private_ssh_key> [-sync <sync>] [-timeout <seconds>]
[-a '{<additional_parameter1, additional_parameter2,...>}']
```

Note: All parameters are described on the main page for this section, [Running VNF Workflows](#).

Important: For the CLI command, `private_ssh_key` value points to the location of the private SSH key file, from which the CLI reads the content automatically and encodes it in base64 scheme before sending the request to Front-End server.

The following CLI command example performs an instantiation of an Openet Policy Manager VNF called `policy_vnf`, with the `vnf_instance_id` set to `dublin`, the tenant set to `default`, the `workflow_type` set to `instantiate_reference_vnf`, the `vnf_template_id` is set to `policy_vnf`, and the `run_as` user set to `admin`.

```
ovlm-fe workflow submit -t default -v policy_vnf -i dublin -y
instantiate_reference_vnf -p fw_v2 -run_as admin -k /home/admin/.ssh/id_rsa
```

Note: Be sure the `OVLMFE_BASEURL` environment variables are set pointing to the Weaver Front End address before using CLI commands.

Assuming the command runs successfully, you should see output similar to the following.

```
data:
  workflow_instance_status: COMPLETED
  success_msg: Instantiation of Reference VNF has completed.
  flow_result: {}
  request: http://node-1:28050/api/v2/tenants/default/vnfs/policy_vnf/vnf_instances/dublin/workflow_instances/113400168
  method: GET
  status: 200
```

The following is an example of a response when one of the VNFCs fails during instantiation.

```

data:
  workflow_instance_status: FAILED
  error_code: EXECUTE_WORKFLOW_FAILED
  error_msg: Instantiation of Reference VNF has failed. Pre-install step has
failed.
  flow_result:
    hosts:
      - hostname: k1-000961-vm01
        server_name: k1-000961-vm01
        exit_code: 1
        response:
          error_msg:
            - Preinstall has failed.
            - Failed to connect to hostname node-5 and servername dc_vm01 with
username admin.
        request: http://node-1:28050/api/v2/tenants/default/vnfs/policy_vnf_rmCompt
est/vnf_instances/comp_dublin/workflow_instances/113400410
        method: GET
    status: 500
  
```

Instantiating a Reference VNF Using REST API

To instantiate a reference VNF from the REST API, use `instantiate_reference_vnf` `workflow_type` parameter with the command shown below.

```

POST http://<ovlm-workflow-fe-url>/api/v2/tenants/<tenant_id>/vnfs/<vnf
_id>/vnf_instances/<vnf_instance_id>/workflow_instances?sync=1&timeout=100
{
  "workflow_type": "instantiate_reference_vnf",
  "vnf_template_id": <package_id>,
  "run_as": <run_as_user>,
  "private_ssh_key": <private_ssh_key>
  [,"additional_parameters": <additional_parameters>]
}
  
```

Note: All parameters are described on the main page for this section, [Running VNF Workflows](#).

Important: For the REST API call, the `private_ssh_key` value must be the content of the SSH key value encoded in base64 scheme.

The following REST API call example performs an instantiation of an Openet Policy Manager VNF called `policy_vnf`, with the `vnf_instance_id` set to `dublin`, the tenant set to `default`, the `workflow_type` set to `instantiate_reference_vnf`, the `vnf_template_id` is set to `policy_v1`, and the `run_as` user set to `admin`.

```

OVLMFE_BASEURL=http://node-1:28050
tenant_id=default
vnf_id=microblog
vnf_instance_id=dc_1
echo " - instantiate_reference_vnf - "
curl \
-S \
-H "Content-Type: application/json" \
-X POST \
-d '{"workflow_type":"instantiate_reference_vnf", "vnf_template_id":"policy_v1",
"run_as": "admin", "private_ssh_key":"LS0tLS1CRUtzFWS0tLS0tCg==" }' \
$OVLMFE_BASEURL/api/v2/tenants/${tenant_id}/vnfs/${vnf_id}/vnf_i
nstances/${vnf_instance_id}/workflow_instances?sync=1|
python -mjson.tool
  
```

The following is an example of a successful response to the API call.

```
{
  "data": {
    "workflow_instance_status": "COMPLETED",
    "success_msg": "Instantiation of Reference VNF has completed.",
    "flow_result": {
    },
    "request": "http://node-1:28050/api/v2/tenants/default/vnfs/policy_vnf/vnf_instances/dublin/workflow_instances",
    "method": "POST"
  },
  "status": 200
}
```

The following is an example of a response when the workflow fails using the same API call as above.

```
data:
  workflow_instance_status: FAILED
  error_code: EXECUTE_WORKFLOW_FAILED
  error_msg: Instantiation of Reference VNF has failed. Dynamic config reference step has failed.
  flow_result:
    vnfc_instances:
      - vnfc_id: policy_server
        vnfc_instance_id: '1'
        hostname: 192.168.122.249
        exit_code: 0
        response:
          output:
            - Dynamic config reference vnfc has completed.
      - vnfc_id: policy_server
        vnfc_instance_id: '2'
        hostname: 192.168.122.248
        exit_code: 0
        response:
          output:
            - Dynamic config reference vnfc has completed.
      - vnfc_id: nginx
        vnfc_instance_id: '1'
        hostname: 192.168.122.34
        exit_code: 1
        response:
          error_msg:
            - Dynamic config reference vnfc has failed.
            - ERROR Can't render template file "configuration/common.xml". 'dict object' has no attribute 'vnfc_ids123'
    request: http://node-1:28050/api/v2/tenants/default/vnfs/policy_vnf/vnf_instances/dublin/workflow_instances
    method: POST
  status: 500
```

Installing the Weaver Integration Module and Configuring the VNF Reference

After running the the instantiate_reference_vnf successfully, you must install the Weaver Integration Module (WIM) into the newly instantiated reference installation node. follow the steps in *Installing the Weaver Integration Module* in the *Weaver Integration Module* user guide. You might need to manually sync-up any files that changed between the existing WIM installation with the new WIM installation. This is because the new WIM installation provides the default content from the existing installation but not any changes made to those files (if any).

You can then configure the new reference installation instance according to your deployment requirements.

Managing VNFs

The Openet Weaver Front End provides a layer of abstraction from the Workflow Engine so that programmable REST API's are provided in a consistent fashion to third party integrators. This results in a unified API that can be used for managing all VNF workflows—the same API can be used by an orchestrator to manage any VNF from any vendor.

Openet Weaver also provides a VNF-M GUI and a CLI interface to the Front End that can be used for managing VNF workflows.

This section of the webhelp includes the following topics:

There are two categories of workflow:

- VNFs configured with a Virtualized Infrastructure Manager (VIM)
- VNFs configured without VIM

Related concepts

[Openet Weaver VNF-M API Reference](#) on page 468

Openet Weaver REST API reference guides

[Openet Weaver VNF-M CLI Reference](#) on page 469

Openet Weaver CLI command reference guides

Managing VNF packages

Managing VNF packages is a part of VNF management. For example, VNF packages are used for instantiation. You can import a package to use in making changes to a VNF, or export a package for archiving it, or to transfer it to another system, for example from the lab environment to a production system. You can delete VNF packages that you no longer have use for.

Related concepts

[Setting Environment Variables](#) on page 133

[Openet Weaver VNF-M API Reference](#) on page 468

Openet Weaver REST API reference guides

[Openet Weaver VNF-M CLI Reference](#) on page 469

Openet Weaver CLI command reference guides

Importing a VNF package

You can import a VNF package using a CLI command, a REST API command, or the VNF-M GUI.

Importing a package from the CLI

To import a VNF package from a tar.gz file use the CLI command:

```
ovlm-cm template import -t tenant_id -v vnf_id -p vnf_template_id -f path/filename.tar.gz --  
overwrite_vnf_workspace value.
```

Note: Use the -o parameter to specify whether you want to overwrite the vnf workspace on import. It accepts the boolean values true and false

You must ensure that the *tenant_id* exists in the Configuration Manager repository to which you are importing. If the *vnf_id* does not exist it is created during import. However, you must ensure that the *vnf_id* specified in the command is the same *vnf_id* in the archive file from which you are importing.

The following is an example of the `template import` command in use:

```
ovlm-cm template import -t default -v webserver -p labell -f  
vnf_archive/labell.tar.gz -o true
```

When the command runs successfully you should see a result similar to the following.

```
data:  
  method: POST  
  request: http://localhost:9090/api/v2/tenants/default/vnfs/webserver/templates/import  
  status: 200
```

In the above example, if a VNF with the same `vnf_id` exists for the tenant workspace in the repository to which you are importing, it is overwritten. To avoid this, set the `-o` value to false.

Importing a VNF package Using the VNF-M GUI

To import a VNF package using the Openet Weaver VNF-M GUI, follow these steps:

1. Open the Navigation menu and select the tenant for the package upload if it is not already selected.

The screenshot shows a user interface for managing VNFs. At the top left is a teal header bar with a user icon and the text "userA". Below this is a white sidebar with a teal header containing the "Weaver" logo and the word "Weaver". The sidebar has a search bar with a magnifying glass icon. Below the search bar is a section titled "Tenants" with two entries: "default" and "anycom". The "anycom" entry is highlighted with a red selection bar. The main content area is a dark grey panel with several horizontal sections. The second section from the top is highlighted with an orange selection bar. A small "X" icon is visible in the middle of this section. The bottom of the main content area has a footer with the text "OPENET".

2. Click the floating



in the bottom-right corner of the screen as shown below.

Name ↑	Description	Creation Date	Actions
▼ microblog	A sample VNF implementing a 3 tier microblog platform	Feb 7, 2017 7:21:52 AM	

Figure 42: VNF Catalog Page

Additional buttons are also displayed, as shown below.

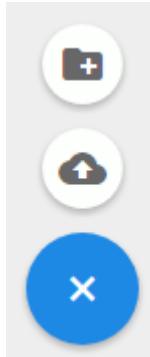


Figure 43: Floating Button Expanded

3. Click the **Import VNF Package** button



The Import VNF Package dialog box displays.

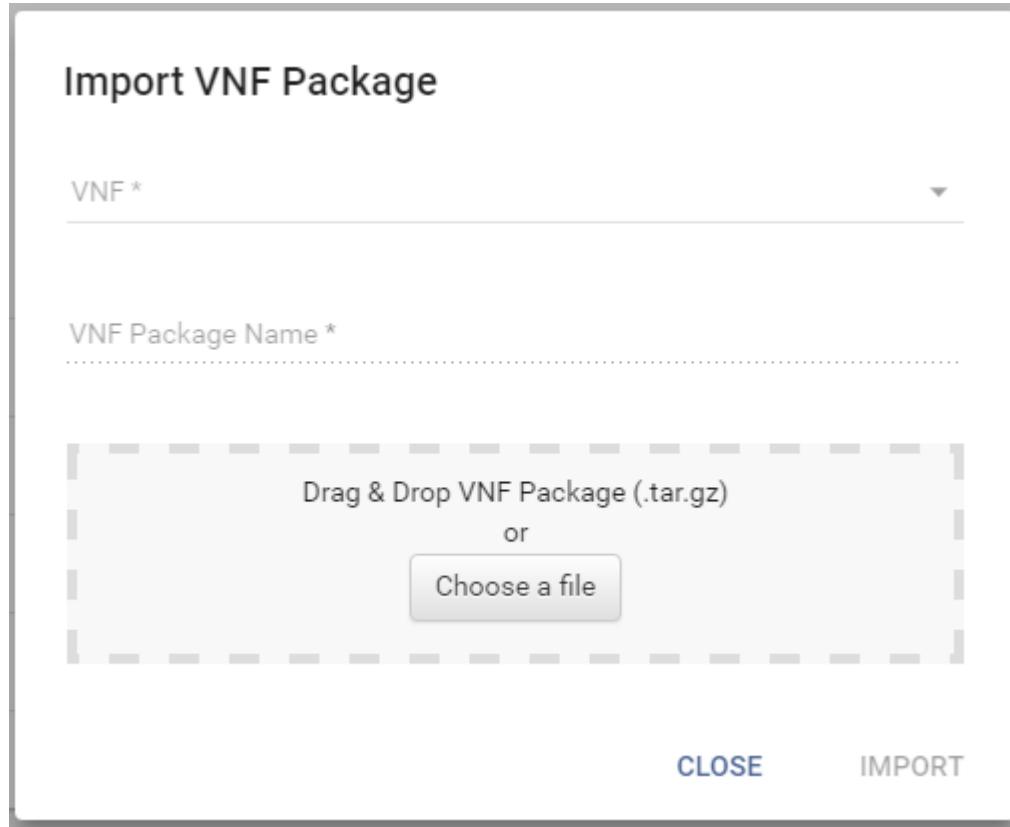


Figure 44: Import VNF Package Dialog Box

4. Select a VNF to import the package to in the VNF drop-down box.
5. Click **Choose a file** then browse to the package file to upload. Alternatively you can drag and drop the file to the Drag & Drop VNF Package (.tar.gz) area.

Note: The import file must be a .tar.gz file.

The name of the file is displayed in the VNF Package Name field.

6. Click **Import**.

Creating a VNF for Package Import

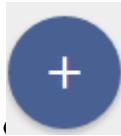
To create a VNF in the Openet Weaver VNF-M GUI, follow these steps:

Note: Before you create a package for import, ensure the VNF exists.

1. Open the Navigation menu and select the tenant for the package upload if it is not already selected.

The screenshot shows the OPENET interface. At the top left is a user icon labeled "userA". Below it is a search bar with a magnifying glass icon and a "Tenants" section. The "Tenants" section lists two entries: "default" and "anycom", each with a circular icon. The "anycom" entry is highlighted with a red selection bar. To the right of the tenants list is a dark sidebar containing several horizontal bars, some of which are highlighted in orange or red. At the bottom of the main area, the word "OPENET" is visible.

2. Click the floating



in the bottom-right corner of the screen as shown below.

Figure 45: Floating button on Catalog Page

Additional buttons appear above the speed dial button.

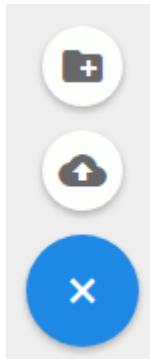


Figure 46: Floating Button Expanded

3. Click the **Create VNF** button



The **Create VNF** dialog box is displayed.

The screenshot shows a 'Create VNF' dialog box. At the top center is the title 'Create VNF'. Below it is a 'Name*' field containing 'vnf1'. Below that is a 'Description *' field containing 'My first VNF'. At the bottom right are two buttons: 'CANCEL' and 'CREATE'.

Figure 47: Create VNF Dialog Box

4. Enter a name for the VNF in the name field.

Note: All fields are mandatory.

5. Enter a description of the VNF in the description field.

The description displays alongside the VNF in the Catalog page of the VNF-M.

6. Click **Create**.

The dialog box closes and the new VNF displays in alphabetic order in the Catalog page.

Exporting a VNF package

You can export a VNF package using a CLI command, an API command, or the VNF-M GUI. The result of each method is the creation of a .tar.gz file.

Exporting a VNF package from the CLI

To export a VNF package use the CLI command `ovlcm-cm template export -t tenant_id -v vnf_id -p vnf_template_id`.

Note: Be sure the baseurl environment variables are set before using CLI commands. There are also API versions of the commands used in this topic.

The command retrieves the specified VNF package including all files and directories from the VNF file repository managed by Weaver Configuration Manager and places them in a file named <vnf-template-id>.tar.gz.

The following is an example of the `template export` command in use:

```
ovlcm-cm template export -t default -v webserver -p label1
```

Running the command results in the creation of a file named label1.tar.gz.

Exporting a VNF package Using the VNF-M

To export a VNF package using the Openet Weaver VNF-M GUI, follow these steps:

1. On the VNF Catalog page, locate the VNF with the package you intend to export and click the down arrow to expand the VNF

Name	Description	Creation Date	Actions
microblog	A sample VNF implementing a 3 tier microblog platform	Feb 7, 2017 7:21:52 AM	
microblog_v3		Mar 8, 2017 10:31:54 AM	
microblog_v1		Mar 8, 2017 10:31:53 AM	
microblog_v2		Mar 8, 2017 10:31:53 AM	

Figure 48: VNF Catalog Page

2. Click the package to export, which in the above illustration is microblog_v1.
3. Click the Export icon



to the far right.

The package downloads to your browser's default download location.

Deleting a VNF package

You can delete a VNF package using a CLI command, an API command, or the VNF-M GUI. The result of each method is to delete all files under `cm/repository-root/tenants/<tenant_id>/vnfs/<vnf_id>/templates`.

Deleting a VNF package from the CLI

To delete a VNF package use the CLI command `ovlm-cm template delete -t tenant_id -v vnf_id -p vnf_template_id`.

Note: Be sure the baseurl environment variables are set before using CLI commands. There are also API versions of the commands used in this topic.

The following is an example of the `template delete` command in use:

```
ovlm-cm template delete -t default -v webserver -p label1
```

When the command runs successfully you should see a result similar to the following.

```
data:
  method: DELETE
  request: http://localhost:9090/api/v2/tenants/default/vnfs/webserver/templates/label1
  status: 200
```

Deleting a VNF package Using the VNF-M

To delete a VNF package using the Openet Weaver VNF-M GUI, follow these steps:

1. On the VNF Catalog page, locate the VNF with the package you intend to delete and click the down arrow to expand it the VNF.

Name	Description	Creation Date	Actions
microblog	A VNF implementing a 3 tier microblog platform	Mar 1, 2017 9:18:38 AM	
microblog_2	A VNF implementing a 3 tier microblog platform	Feb 7, 2017 7:21:52 AM	
microblog_3	A VNF implementing a 3 tier microblog platform	Mar 1, 2017 9:18:21 AM	
microblog_4	A VNF implementing a 3 tier microblog platform	Mar 1, 2017 9:18:39 AM	

Figure 49: VNF Catalog Page

2. Click the package you intend to delete.
3. Click the delete icon



to the far right.

Assuming the deletion is successful a green success banner is displayed at the top of the page and the package is removed from the system.

Note: If deletion is unsuccessful a red banner message is displayed.

Running VNF Workflows

Openet Weaver comes with predefined VNF-level workflows that can be used for out-of-the-box VNF management. Each flow represents a process or group of processes to be run by the VNF manager. Workflows can be initiated by a CLI command at a user interface connected to the Workflow Engine, by an API call, for example through a third-party orchestrator, or by using the VNF-M GUI.

You can use the `workflow submit` command to run any defined VNF workflow.

To run workflow submit from the Front End CLI enter the following command.

```
ovlm-fe workflow submit -t <tenant_id> -v <vnf_id> -i <vnf_instance_id> -y
<workflow_type> -p <vnf_template_id> -vnfd_id <vnfd_id>
[-start <true | false>] [-restart <true | false>] [-sync <sync>] [-timeout
<seconds>]
[-o <execution_order>] [-force <force_value>] [-a <additional_parameter1,
additional_parameter2,...>]
```

Note: Be sure the baseurl environment variables are set before using CLI commands.

The following is an example of running the VNF instantiation workflow from the CLI.

```
ovlm-fe workflow submit -t default -v microblog -i dc_1 -y instantiate_vnf -p
microblog_v1 \
-vnfd_id vnfd_production -restart true
```

To run workflow submit through the Front End API use the following.

```
curl -X POST -H 'Content-Type: application/json' -d
'{"workflow_type":"<workflow_type>","vnfd":{"vnf_template_id": \
"<vnf_package>"}}'
http://<your_api_host>/api/v2/tenants/<tenant_id>/vnfs/<vnf_id>/vnf_instances/<vnf_instance_id>/\
workflow_instances? [sync=<sync>]
```

The optional sync parameter specifies whether the API call is synchronous. The call is synchronous If set to 1, and the API call waits to execute until the end of the current workflow run or until the timeout value specified by the synchronous-request-delay option in the front-end configuration field. If sync is set to 0 the call is asynchronous, which is the default setting..

The following is an example of instantiating a VNF through the API.

```
curl -X POST -H 'Content-Type: application/json' -d '{"workflow_type":"in
stantiate_vnf","vnfd":{"vnf_template_id": \
"microblog_v1", "vnfd_id": "vnfd_production"}}' http://host1/api/v2/te
nants/default/vnfs/microblog/vnf_instances/dc_1/workflow_instances
```

The workflow submit parameters are described in the following table. They apply to the CLI and the API versions of the command.

Table 39: workflow submit Parameters

CLI Flag	Parameter	Mn	Description
-t	tenant_id	Yes	The tenant identifier as created in the Configuration Manager.
-v	vnf_id	Yes	The VNF identifier as created in the Configuration Manager.
-i	vnf_instance_id	Yes	The VNF instance identifier as created in the Configuration Manager.
-y	workflow_type	Yes	<p>The Workflow type for the lifecycle event defined in the workflow. This can be any workflow defined for Openet Weaver. The values for the predefined workflows that come with Openet Weaver are as follows:</p> <ul style="list-style-type: none"> • instantiate_vnf • instantiate_vnf_without_vim • upgrade_vnf • update_vnf • rollback_vnf • start_vnf • stop_vnf • is_vnf_up • terminate_vnf • terminate_vnf_without_vim • instantiated_vnf • is_vnf_healthy <p>Predefined workflows are described in the topics linked at the bottom of this page.</p>
-p	vnf_template_id	Yes	The unique identifier for the VNF package. Mandatory for the following workflows only: <ul style="list-style-type: none"> • instantiate_vnf • instantiate_vnf_without_vim • upgrade_vnf • update_vnf

CLI Flag	Parameter	Mn	Description
-vnfd_id	vnfd_id	No	<p>The unique identifier for the VNFD file associated with the VNF.</p> <p>Used only for the instantiate_vnf workflow, for which it is mandatory.</p>
-start	start	No	<p>Specifies whether a VNF should start as part of instantiation, or be placed in an inactive state to be started later. Possible values are:</p> <ul style="list-style-type: none"> • true (default) • false <p>A value of true means the VNF starts when instantiated. If the parameter is set to false, instantiation omits the start_vnfc and is_vnfc_up steps of the instantiation flow.</p> <p>Note: The start option is not applicable to the instantiate_reference_vnf workflow.</p> <p>The flag is useful when starting a DB cluster, for example. In such a case there is a dependency between VNF instances, and an error can be reported and starting can fail when one instance takes longer to start than others. For example, when the master node starts after a standby node.</p> <p>When the VNFs are instantiated in an inactive state, you can determine the start order manually. For example, you can start the master node of the cluster and wait to start others until it is up.</p> <p>This parameter is used only with instantiate_vnf_without_vim.</p>
-restart	restart	No	<p>A flag used to signify whether the workflow should restart the system. Used only with the vnf_update workflow. When set to false, the workflow does not restart the VNF. The default value of true is used when the restart parameter is not present.</p>
-sync	sync	No	<p>The optional sync parameter to specify whether the call is synchronous (1) or asynchronous (0). The call is asynchronous by default.</p>
-timeout	timeout	No	<p>The optional timeout parameter to specify how long the synchronous call waits for the operation to complete. The default value is 60 seconds.</p> <p>Note: The default timeout value can be specified in the front-end: synchronous-request-timeout attribute in the installation file and propagated to the Front-End microservice using the deploy script.</p>
-o	execution_order	No	<p>The optional execution_order parameter to specify whether VNFC instances are upgraded in parallel or in sequence. Possible values are:</p> <ul style="list-style-type: none"> • sequential • parallel <p>The default value of sequential is used when the parameter is not present. The parallel mode is used for out-of-service execution.</p> <p>The execution_order parameter can have an affect on the following workflows:</p> <ul style="list-style-type: none"> • upgrade_vnf • update_vnf • rollback_vnf <p>The execution_order is typically set to parallel when a workflow applies to VNFC instances that are cluster-aware. These instances must be started simultaneously and stopped simultaneously. System availability might be interrupted while the workflow runs. Therefore a maintenance window is required to run workflows in parallel.</p>

CLI Flag	Parameter	Mn	Description			
-force	force	No	Value	Value Description	Workflow	Effect on Workflow
			0	The default value, which is the value assumed when the -force option is not present in the command.	update_vnf upgrade_vnf terminate_vnf_ without_vim	The VNF template is compared to the existing VNF template during staging. When there is no difference between the two the workflow exits. The workflow exits when the uninstall_vnfc procedure fails.
			1	Applies to all VNFCs associated with the VNF.	update_vnf upgrade_vnf terminate_vnf_ without_vim	Processing continues for a VNFC instance regardless of whether there is a difference between the existing VNF template and the new template to be used. The workflow continues to the clean_up procedure even when the uninstall_vnfc procedure fails. In this event, a warning is returned.
			Comma separated list of VNFCs ("<vnfc_1>", "<vnfc_2>", ...)	Applies to the specified VNFC IDs.	update_vnf upgrade_vnf terminate_vnf_ without_vim	Processing continues for the specified VNFC instances regardless of whether there is a difference between the existing VNF template and the new template to be used. N/A

CLI Flag	Parameter	Mn	Description
-a	additional_parameters	No	<p>Generic parameters in JSON format that are passed through the workflow submit command to the procedural level on Resource Agents. These parameters are used to extend the command's flexibility to support end-user scenarios. The potential impact on the command depends upon which workflow you are running.</p> <p>For example, when running the upgrade workflow, you can take a snapshot of the VNF instance before performing the upgrade. Openet Weaver does not have a defined workflow submit parameter for this action, but a <code>take_a_snapshot</code> parameter might be defined for the actual procedure, and the parameter-value pair can be passed through the workflow submit command to the Resource Agent for processing.</p> <p>You can predefine additional parameters for use with the VNF-M GUI. When you do, the predefined additional parameters display in the dialog box that opens when you select a workflow to run.</p> <p>Note: See Managing Additional Parameters for the VNF-M GUI for more information about predefining additional parameters.</p> <p>Additional parameters can be used with the following workflows:</p> <ul style="list-style-type: none"> • <code>instantiate_vnf</code> • <code>instantiate_vnf_without_vim</code> • <code>upgrade_vnf</code> • <code>update_vnf</code> • <code>rollback_vnf</code> • <code>start_vnf</code> • <code>stop_vnf</code> • <code>is_vnf_up</code> • <code>terminate_vnf</code> • <code>terminate_vnf_without_vim</code>

The predefined workflows that are provided with Openet Weaver are described in the following topics:

Related concepts

[Setting Environment Variables](#) on page 133

[Openet Weaver VNF-M API Reference](#) on page 468

Openet Weaver REST API reference guides

[Openet Weaver VNF-M CLI Reference](#) on page 469

Openet Weaver CLI command reference guides

Instantiating VNFs

Running the instantiation workflow creates a VNF instance, starts it and checks whether it is running. Any updates to VNFC configuration are implemented during the instantiation process.

There are two instantiation workflows:

- `instantiate_vnf` (instantiation with VIM configuration)
- `instantiate_vnf_without_vim`

Running an instantiation workflow creates a workflow_id.

Note: For more information about retry see [Configuring Workflows to Retry Automatically](#).

A successful vnf instantiation is shown below in the Workflow Engine GUI.

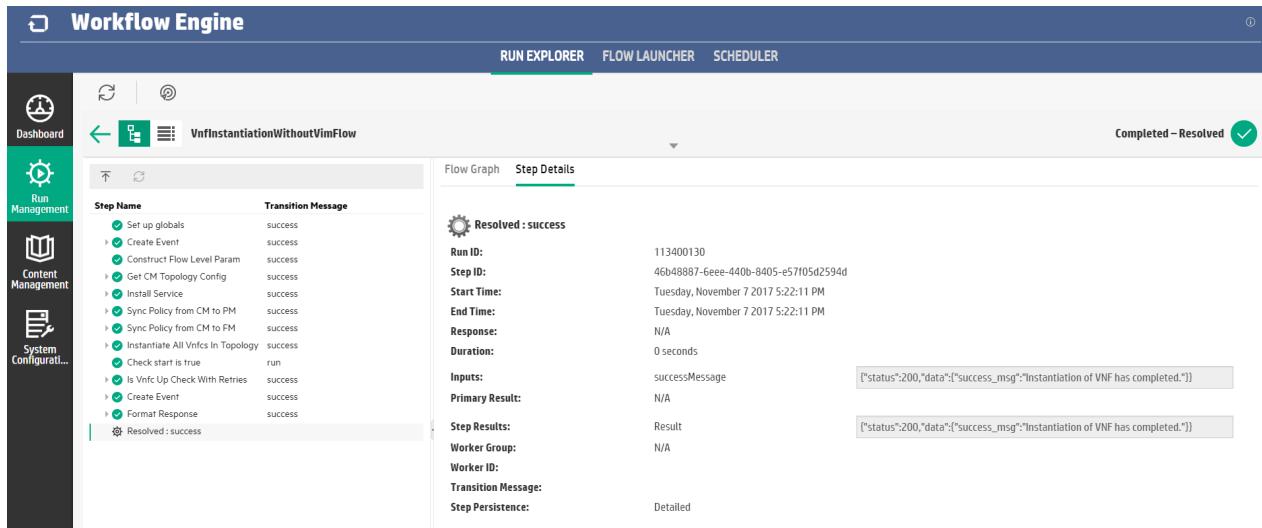


Figure 50: Successful VNF Instantiation in Workflow Engine GUI

The subtopics describe the instantiation workflows in more detail.

Instantiating a VNF Without VIM

You can instantiate a VNF without VIM configuration from the CLI, the API, or from the Openet Weaver VNF-M GUI.

Note: You have the option of instantiating a VNF without starting it, in which case instantiation omits the start_vnfc and is_vnfc_up steps of the workflow.

If no Resource Agent is installed at the time you run the workflow, one is installed by Deployment Manager. The configuration stored in Deployment Manager is used to perform the installation.

The instantiate_vnf_without_vim workflow is shown below.

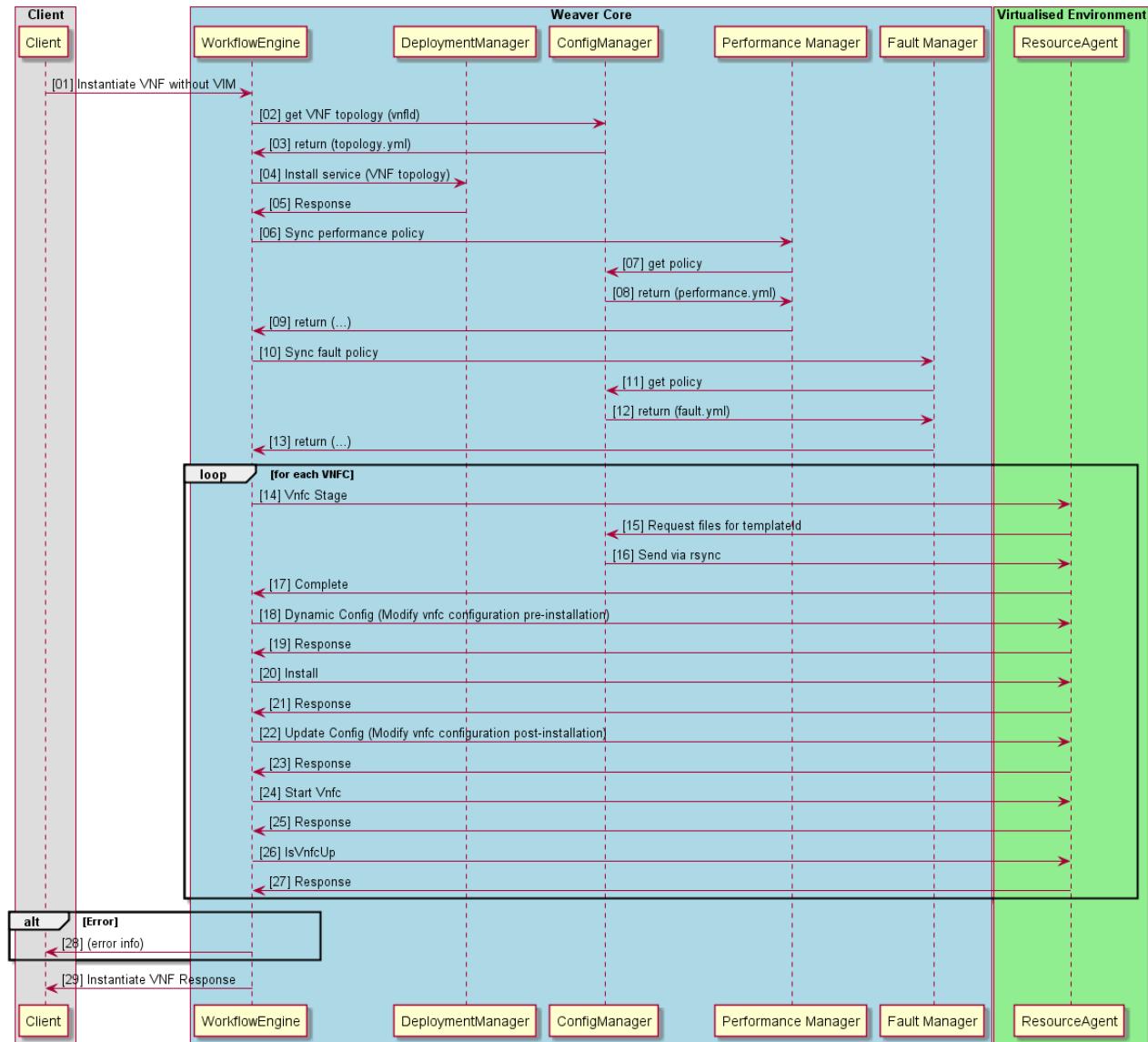


Figure 51: instantiate_vnf_without_vim Process Flow

In the case of failure, the workflow exits and returns an error when the failure is encountered and the current stage for the other VNFCs have completed their run. For example, if one VNFC fails on a dynamic configuration stage while the install stage is running for a second VNFC in the VNF, the install stage completes before the workflow exits. Breaking out of the workflow in this way allows for early failure, with no unnecessary time spent waiting for the workflow to continue for the other VNFCs if one fails. However, if the workflow is configured to retry the process will be repeated before exiting and reporting an error.

Related concepts

[Setting Environment Variables](#) on page 133

[Openet Weaver VNF-M API Reference](#) on page 468

Openet Weaver REST API reference guides

[Openet Weaver VNF-M CLI Reference](#) on page 469

Openet Weaver CLI command reference guides

Instantiating a VNF Without VIM from the CLI

To instantiate a VNF without VIM configuration from the CLI, use the `instantiate_vnf_without_vim` workflow_type parameter with the command shown below.

```
ovlm-fe workflow submit -t <tenant_id> -v <vnf_id> -i <vnf_instance_id> -y
instantiate_vnf_without_vim -p <vnf_template_id>
[-start <true | false>] [-sync <sync>] [-timeout <seconds>]
[-a '{<additional_parameter1, additional_parameter2,...>}']
```

Note: All parameters are described on the main page for this section, [Running VNF Workflows](#).

The following CLI command example performs an instantiation of the out-of-the-box vnf called microblog, with `vnf_instance_id` set to `dc_1`, the tenant set to default, the `workflow_type` set to `instantiate_vnf_without_vim`, and the `vnf_template_id` is set to `microblog_v1`.

```
ovlm-fe workflow submit -t default -v microblog \
-i dc_1 -y instantiate_vnf_without_vim -p microblog_v1
```

Note: Be sure the baseurl environment variables are set before using CLI commands.

Assuming the command runs successfully, you should see output similar to the following.

```
data:
  workflow_instance_status: COMPLETED
  success_msg: Instantiation of VNF has completed.
  flow_result: {}
  request: http://node-1:28050/api/v2/tenants/default/vnfs/microblog/vnf_instances/dc_1/workflow_instances
  method: POST
  status: 200
```

The following is an example of a response when one of the VNFCs fails during instantiation.

```
data:
  workflow_instance_status: FAILED
  error_code: EXECUTE_WORKFLOW_FAILED
  error_msg: 'Instantiation of VNF has failed due to vnfc_id: nginx,
vnfc_instance_id: 1'
  flow_result:
    vnfc_instances:
      - vnfc_id: blogserver
        vnfc_instance_id: '1'
        hostname: 192.168.122.249
        exit_code: '1'
        response:
          error_msg:
            - update_vnfc step has failed.
            - Exiting flow due to other VNFC instance error.
            - Destination directory not found.
      - vnfc_id: blogserver
        vnfc_instance_id: '2'
        hostname: 192.168.122.248
        exit_code: '0'
        response:
          output:
            - Instantiation of VNFC has completed.
      - vnfc_id: nginx
        vnfc_instance_id: '1'
        hostname: 192.168.122.34
        exit_code: '1'
        response:
```

```

    error_msg:
      - stage step has failed.
      - Abort instantiation workflow.
      - Source directory not found.
  - vnfc_id: postgres
    vnfc_instance_id: '1'
    hostname: 192.168.122.66
    exit_code: '0'
    response:
      output:
        - Instantiation of VNFC has completed.
  request: http://node-1:28050/api/v2/tenants/default/vnfs/microblog/vnf_instances/dc_1/workflow_instances
  method: POST
  status: 500

```

Instantiating a VNF Without VIM Using REST API

To instantiate a VNF without VIM configuration using REST API, post a workflow request using the instantiate_vnf_without_vim workflow_type parameter. The API request takes the following format.

```

POST http://<ovlm-workflow-fe-url>/api/v2/tenants
/<tenant_id>/vnfs/<vnf_id>/vnf_instances/<vnf_instance_id>/workflow_instances?sync=1&timeout=100
{
  "workflow_type": "instantiate_vnf_without_vim",
  "vnf_template_id": <package_id>,
  ["start: <true | false>"],
  ["additional_parameters": <additional_parameters>]
}

```

Note: All parameters are described on the main page for this section, [Running VNF Workflows](#).

The following example instantiates the non-VIM microblog VNF using curl. This example is a synchronous API call (sync=1).

Note: When a call is specified as asynchronous or the sync parameter is not present in the call, the status can be queried using a GET status workflow.

```

OVLMFE_BASEURL=http://10.3.18.22:28085
tenant_id=default
vnf_id=microblog
vnf_instance_id=dc_1
echo " - instantiate_without_vim - "
curl \
-S \
-H "Content-Type: application/json" \
-X POST \
-d '{"workflow_type": "instantiate_vnf_without_vim", "vnf_template_id": "microblog_v1"}' \
$OVLMFE_BASEURL/api/v2/tenants/${tenant_id}/vnfs/${vnf_id}/vnf_instances/${vnf_instance_id}/workflow_instances?sync=1 \
python -mjson.tool | pygmentize -l json

```

The following is an example of a successful response to the API call.

```

{
  "data": {
    "workflow_instance_status": "COMPLETED",
    "success_msg": "Instantiation of VNF has completed.",
    "flow_result": {

```

```

        },
        "request": "http://node-1:28050/api/v2/tenants/default/
vnfs/microblog/vnf_instances/dc_1/workflow_instances/125828874",
        "method": "GET"
    },
    "status": 200
}

```

The following is an example of a response when the workflow fails using the same API call as above.

```

{
  "data": {
    "workflow_instance_status": "FAILED",
    "error_code": "EXECUTE_WORKFLOW_FAILED",
    "error_msg": "Instantiation of VNF has failed due to vnfc_id: nginx,
vnfc_instance_id: 1",
    "flow_result": {
      "vnfc_instances": [
        {
          "vnfc_id": "blogserver",
          "vnfc_instance_id": "1",
          "hostname": "192.168.122.249",
          "exit_code": "1",
          "response": {
            "error_msg": ["update_vnfc step has failed.",
              "Exiting flow due to other VNFC instance error.",
              "Destination directory not found."]
          }
        },
        {
          "vnfc_id": "blogserver",
          "vnfc_instance_id": "2",
          "hostname": "192.168.122.248",
          "exit_code": "0",
          "response": {
            "output": ["Instantiation of VNFC has completed."]
          }
        },
        {
          "vnfc_id": "nginx",
          "vnfc_instance_id": "1",
          "hostname": "192.168.122.34",
          "exit_code": "1",
          "response": {
            "error_msg": ["stage step has failed.",
              "Abort instantiation workflow.",
              "Source directory not found."]
          }
        },
        {
          "vnfc_id": "postgres",
          "vnfc_instance_id": "1",
          "hostname": "192.168.122.66",
          "exit_code": "0",
          "response": {
            "output": ["Instantiation of VNFC has completed."]
          }
        }
      ]
    },
    "request": "http://node-1:28050/api/v2/tenants/default/
vnfs/microblog/vnf_instances/dc_1/workflow_instances/125832562",
    "method": "GET"
  },

```

```
        "status": 500
    }
```

Instantiating a VNF Without VIM Using the VNF-M GUI

To instantiate a VNF that is not configured for VIM, follow these steps:

1. On the VNF Lifecycle page, locate the inactive VNF to instantiate and click the instantiate icon



Scroll down the page if required.

Note: Inactive VNFs are grouped above active VNFs.

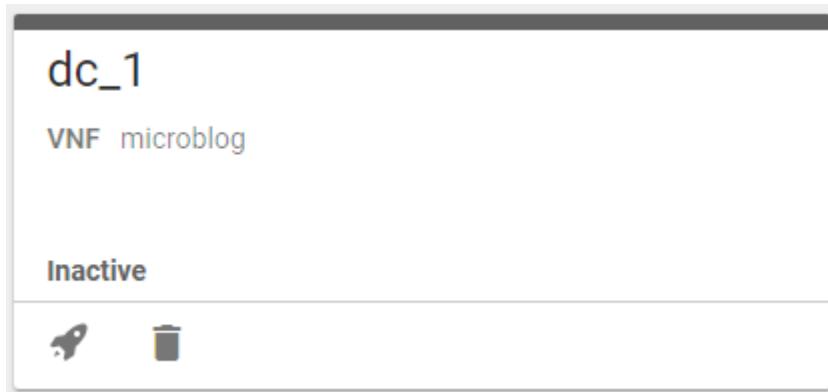


Figure 52: Inactive VNF

The Select a VNF Package dialog box is displayed.

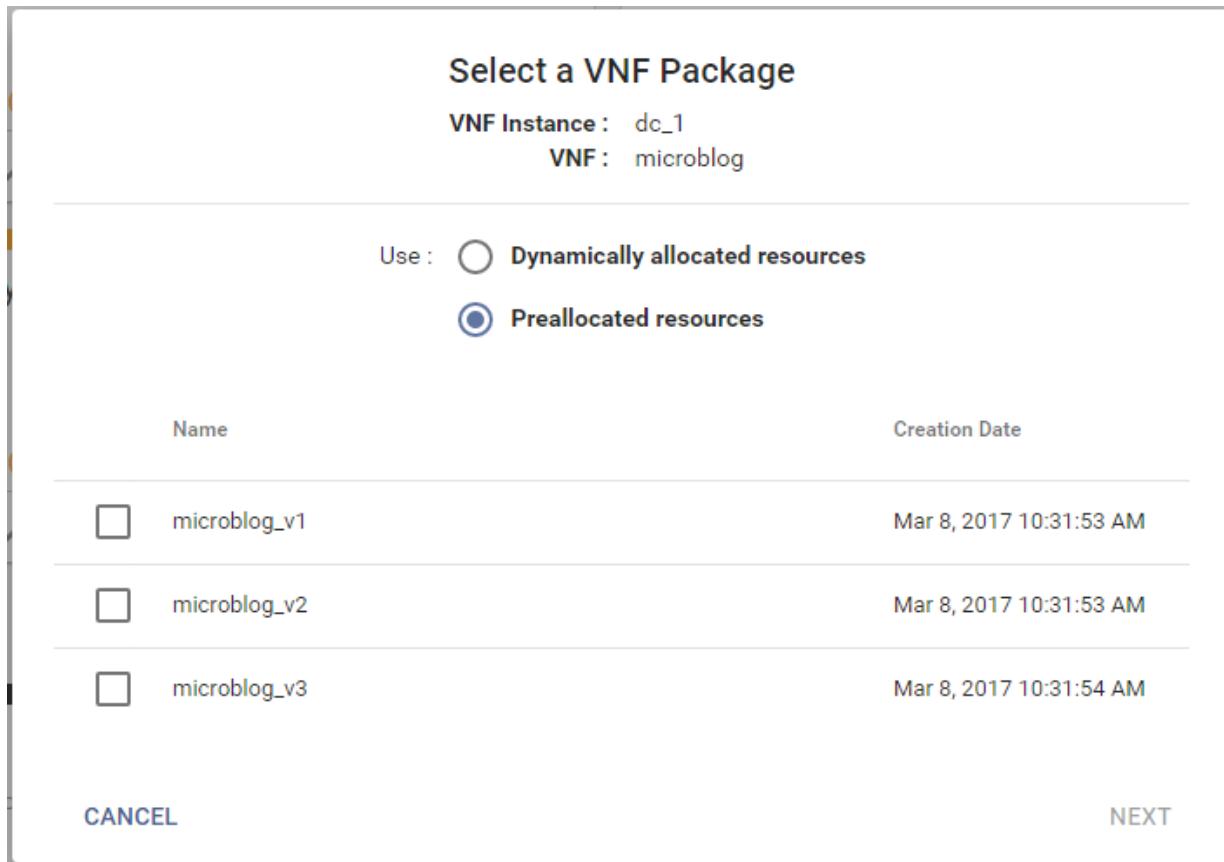


Figure 53: Select a VNF Package Dialog Box

- Click the **Preallocated Resources** radio button because you are instantiating a VNF that has not been configured for VIM.

Note: The radio button defaults to the selection made when used previously.

- Click a VNF package in the **VNF Packages** pane, for example microblog_v1.
- Click **Next**.

The Upload a Topology dialog box displays.



Figure 54: Upload a Topology Dialog Box

- Click **Choose a file** and browse to the YML file with the topology you are using for the VNF.

Alternatively you can drag and drop a file into the **Drag & Drop a Topology File** field.

If the VNF instance already has a topology YML file associated with it, you are given the option of using the existing file or uploading a new file. To use the existing file click the **Yes** radio button. There is a download link to the existing topology file so you can download it.

6. Click **Next**.

The Configure Parameters dialog box displays.

The **Start** check box is selected by default. This means that the VNF is started upon instantiation. To instantiate the VNF without starting it, clear **Start**.

7. To configure additional parameters for a script you are running on the Resource Agent related to instantiating the VNF, click the icon in the top-right of the dialog box and select **Add parameters**.

The dialog box displays a field into which you can enter additional parameters. In some cases there might be predefined parameters, which will display when you select **Add Parameters**. Predefined parameters can be overwritten.

Note: For information about how to predefine Additional Parameters, see *Managing Additional Parameters for the VNF-M GUI*.

Parameters must be entered in JSON format as in the following example.

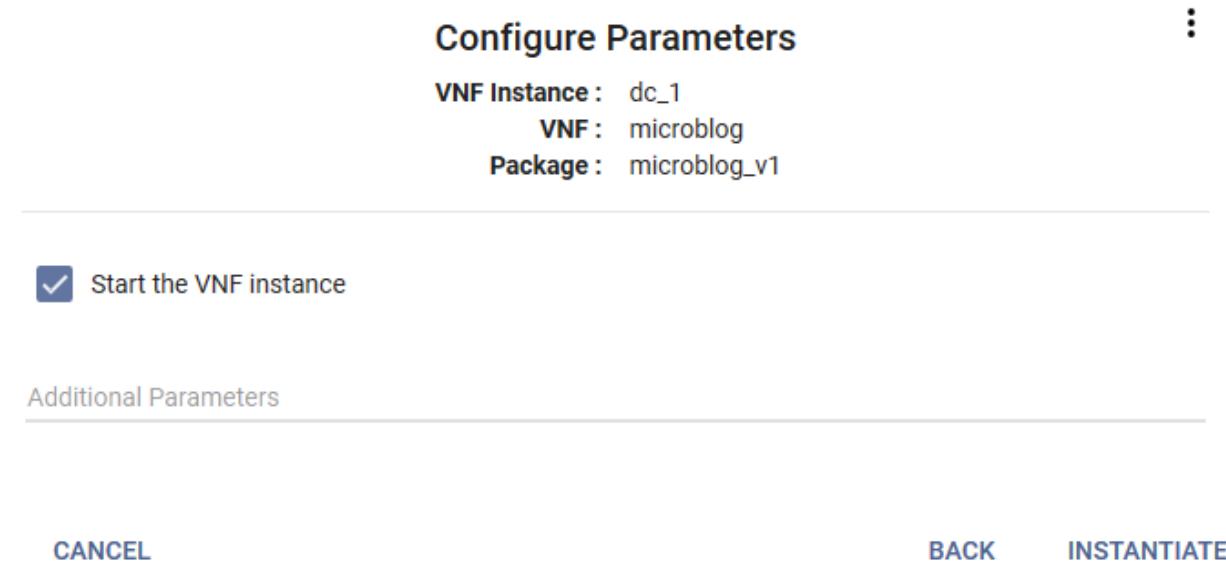


Figure 55: Configure Parameters dialog box

Note: To remove additional parameters after adding them, click the top-right icon again and select **Remove parameters**.

8. Click **Instantiate**.

Assuming that the topology file uploads successfully, an instantiation request is sent and the dialog box closes. The VNF instance card shows its status as "Instantiate In-Progress". After the request is processed completely, the VNF instance card shows the status as "Instantiate Succeeded", assuming that the VNF instantiated successfully.

Note: The status shows as Instantiate Failed if the instantiation request is not successful.

Instantiating a VNF With VIM

You can instantiate VNF a with VIM configuration from the CLI, the API, or from the Openet Weaver VNF-M GUI.

When `instantiate_vnf` is used the workflow performs VNF instantiation based on a predefined VNFD file. A topology file will be created and a workflow instance identifier will be returned on a successful workflow instance creation.

The below flow chart illustrates the process flow that is initiated when the instantiate_vnf workflow is run.

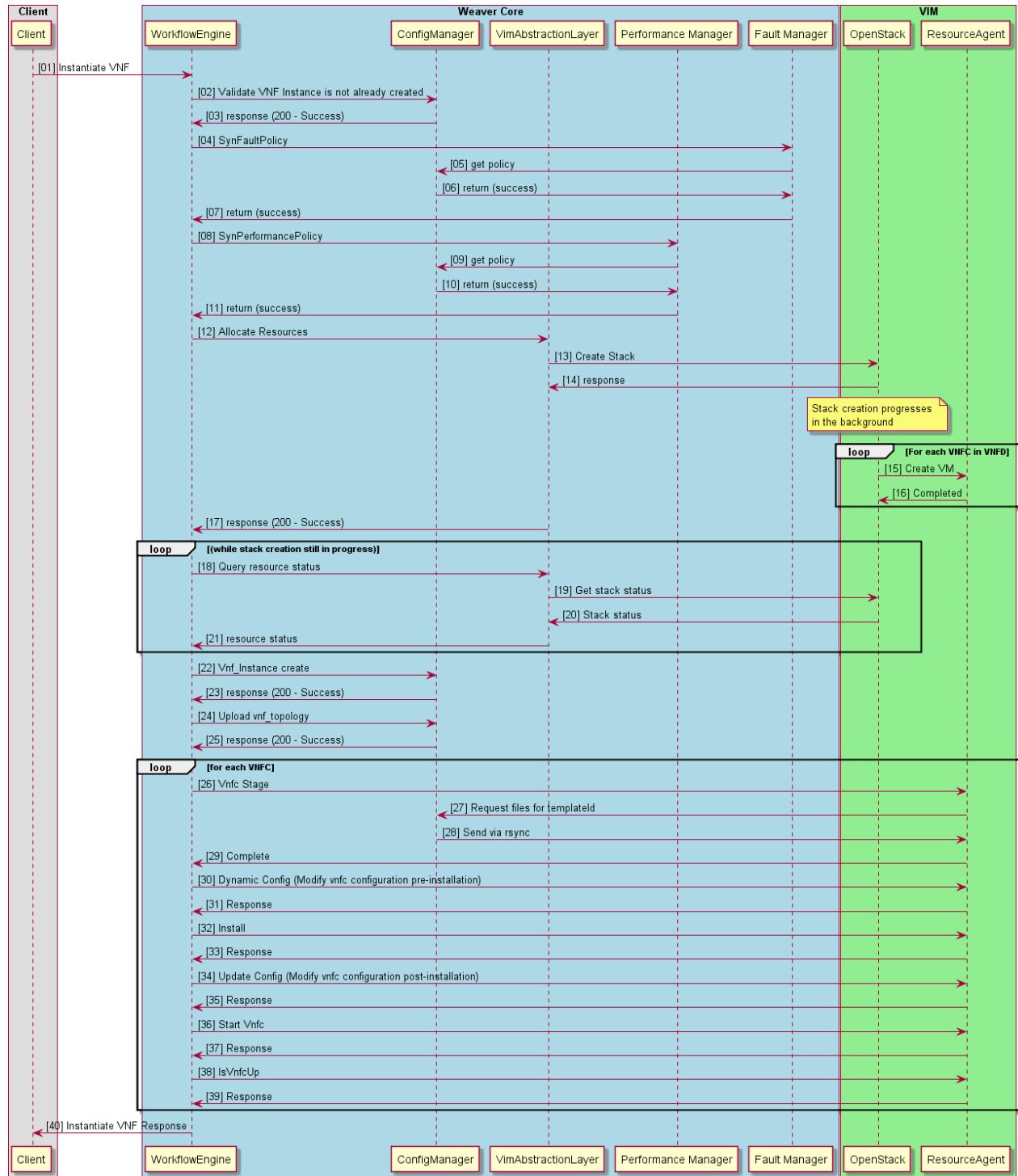


Figure 56: instantiate_vnf Process Flow

In the case of failure, the workflow exits and returns an error when the failure is encountered and the current stage for the other VNFCs have completed their run. For example, if one VNFC fails on a dynamic configuration stage while the install stage is running for a second VNFC in the VNF, the install stage completes before the workflow exits. Breaking out of the workflow in this way allows for early failure, with no unnecessary time spent waiting for the workflow to continue for the other VNFCs if one fails. However, if the workflow is configured to retry the process will be repeated before exiting and reporting an error.

Related concepts

[Setting Environment Variables](#) on page 133

[Openet Weaver VNF-M API Reference](#) on page 468
Openet Weaver REST API reference guides

[Openet Weaver VNF-M CLI Reference](#) on page 469
Openet Weaver CLI command reference guides

Instantiating a VNF With VIM from the CLI

To instantiate a VNF with VIM configuration, instantiate_vnf should be used for the workflow_type parameter with the command shown below.

```
ovlm-fe workflow submit -t <tenant_id> -v <vnf_id> -i <vnf_instance_id> -y
instantiate_vnf -p <vnf_template_id>
[-sync <sync>] [-timeout <seconds>] [-a '{<additional_parameter1,
additional_parameter2,...>}']
```

Note: All parameters are described on the main page for this section, [Running VNF Workflows](#).

The following CLI command example performs an instantiation of the out-of-the-box VNF called microblog, with vnf_instance_id set to dc_1, the tenant set to default, and the workflow_type set to instantiate_vnf, the vnf_template_id is set to microblog_v1, and the vnf_id is set to vnfd_production.

```
ovlm-fe workflow submit -t default -v microblog -i dc_1 -y instantiate_vnf\
-p microblog_v1 -vnfd_id vnfd_production
```

Note: Be sure the baseurl environment variables are set before using CLI commands.

Assuming the command runs successfully, you should see output similar to the following.

```
data:
  workflow_instance_status: COMPLETED
  success_msg: Instantiation of VNF has completed.
  flow_result: {}
  request: http://node-1:28050/api/v2/tenants/default/vnfs/microblog/vnf_instances/dc_2/workflow_instances
  method: POST
  status: 200
```

The following is an example of a response when one of the VNFCs fails during instantiation.

```
data:
  workflow_instance_status: FAILED
  error_code: EXECUTE_WORKFLOW_FAILED
  error_msg: 'Instantiation of VNF has failed due to vnfc_id: nginx,
vnfc_instance_id: 1'
  flow_result:
    vnfc_instances:
    - vnfc_id: blogserver
      vnfc_instance_id: '1'
      hostname: 192.168.122.249
      exit_code: '1'
      response:
        error_msg:
        - update_vnfc step has failed.
        - Exiting flow due to other VNFC instance error.
        - Destination directory not found.
    - vnfc_id: blogserver
      vnfc_instance_id: '2'
      hostname: 192.168.122.248
```

```

    exit_code: '0'
    response:
        output:
            - Instantiation of VNFC has completed.
- vnfc_id: nginx
  vnfc_instance_id: '1'
  hostname: 192.168.122.34
  exit_code: '1'
  response:
      error_msg:
          - stage step has failed.
          - Abort instantiation workflow.
          - Source directory not found.
- vnfc_id: postgres
  vnfc_instance_id: '1'
  hostname: 192.168.122.66
  exit_code: '0'
  response:
      output:
          - Instantiation of VNFC has completed.
request: http://node-1:28050/api/v2/tenants/default/vnfs/microblog/vnf_instances/dc_2/workflow_instances
method: POST
status: 500

```

Instantiating a VNF With VIM Using REST API

To instantiate a VNF with VIM configuration using REST API, post a workflow request using the `instantiate_vnf` `workflow_type` parameter. The API request takes the following format.

```

POST http://<ovlm-workflow-fe-url>/api/v2/tenants
/<tenant_id>/vnfs/<vnf_id>/vnf_instances/<vnf_instance_id>
/workflow_instances?sync=1&timeout=100
{
    "workflow_type": "instantiate_vnf",
    "vnf_template_id": <package_id>,
    "vnfd_id": <vnfd_id>,
    ["additional_parameters": <additional_parameters>]
}

```

Note: All parameters are described on the main page for this section, [Running VNF Workflows](#).

The following example instantiates the VIM-configured microblog VNF using curl. This example is a synchronous API call (`sync=1`).

```

OVLMFE_BASEURL=http://10.3.18.22:28085
tenant_id=default
vnf_id=microblog
vnf_instance_id=dc_2
echo " - instantiate_with_vim - "
curl \
-S \
-H "Content-Type: application/json" \
-X POST \
-d '{"workflow_type": "instantiate_vnf", "vnf_template_id": "microblog_v1", "vnfd_id": "vnfd1"}' \
$OVLMFE_BASEURL/api/v2/tenants/${tenant_id}/vnfs/${vnf_id}/vnf_instances/${vnf_instance_id}/workflow_instances?sync=1 |
python -mjson.tool | pygmentize -l json

```

The following is an example of a successful response to the API call.

```
{
  "data": {
    "workflow_instance_status": "COMPLETED",
    "success_msg": "Instantiation of VNF has completed.",
    "flow_result": {

    },
    "request": "http://node-1:28050/api/v2/tenants/default/
vnfs/microblog/vnf_instances/dc_1/workflow_instances/125828874",
    "method": "GET"
  },
  "status": 200
}
```

The following is an example of a response when the workflow fails using the same API call as above.

```
{
  "data": {
    "workflow_instance_status": "FAILED",
    "error_code": "EXECUTE_WORKFLOW FAILED",
    "error_msg": "Instantiation of VNF has failed due to vnfc_id: nginx,
vnfc_instance_id: 1",
    "flow_result": {
      "vnfc_instances": [
        {
          "vnfc_id": "blogserver",
          "vnfc_instance_id": "1",
          "hostname": "192.168.122.249",
          "exit_code": "1",
          "response": {
            "error_msg": ["update_vnfc step has failed.", "Exiting flow due to other VNFC instance error.", "Destination directory not found."]
          }
        },
        {
          "vnfc_id": "blogserver",
          "vnfc_instance_id": "2",
          "hostname": "192.168.122.248",
          "exit_code": "0",
          "response": {
            "output": ["Instantiation of VNFC has completed."]
          }
        },
        {
          "vnfc_id": "nginx",
          "vnfc_instance_id": "1",
          "hostname": "192.168.122.34",
          "exit_code": "1",
          "response": {
            "error_msg": ["stage step has failed.", "Abort instantiation workflow.", "Source directory not found."]
          }
        },
        {
          "vnfc_id": "postgres",
          "vnfc_instance_id": "1",
          "hostname": "192.168.122.66",
          "exit_code": "0",
          "response": {

```

```
        "output": ["Instantiation of VNFC has completed."]
    }
}
},
"request": "http://node-1:28050/api/v2/tenants/default/
vnfs/microblog/vnf_instances/dc_2/workflow_instances",
"method": "POST"
},
"status": 500
}
```

Instantiating a VNF With VIM Using the VNF-M GUI

To instantiate a VNF that is configured for VIM, follow these steps:

1. On the VNF Lifecycle page, locate the inactive VNF you to instantiate and click the instantiate icon



Scroll down the page if required.

Note: Inactive VNFs are grouped above active VNFs.

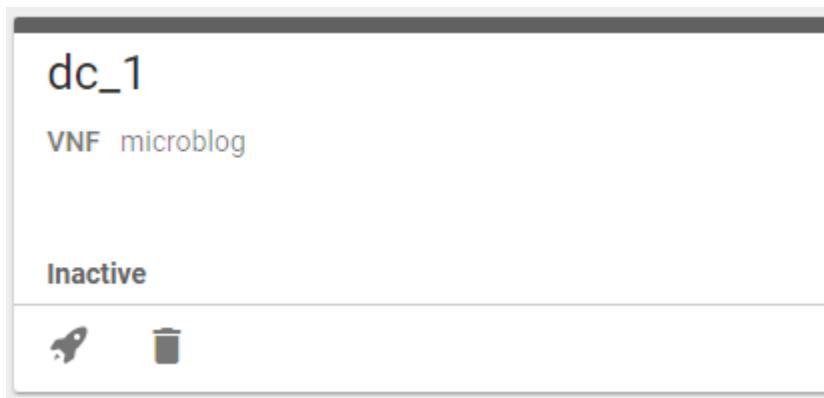


Figure 57: Inactive VNF

The Select a VNF Package dialog box is displayed.

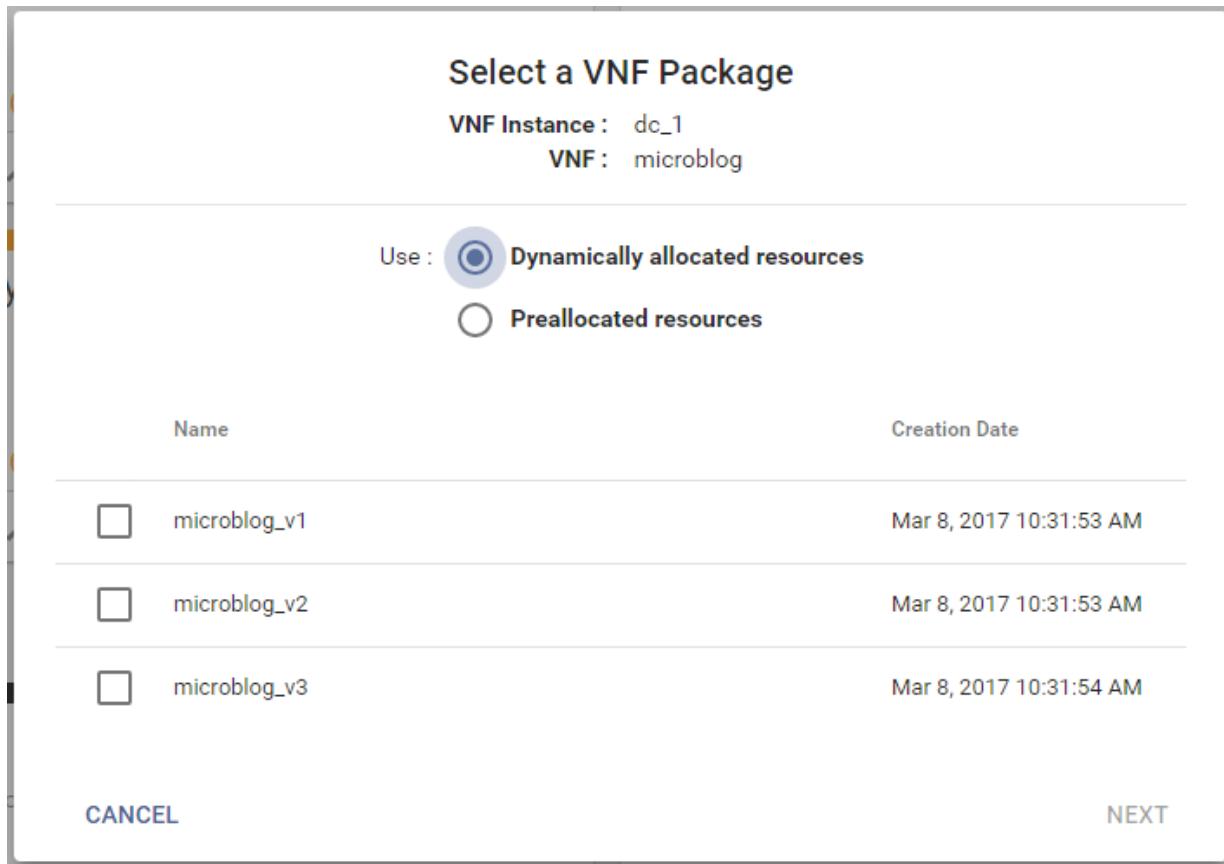


Figure 58: Select a VNF Package Dialog Box

2. Click a VNF package in the **VNF Packages** pane, for example **microblog_v2**.
3. Click the **Dynamically allocated resources** radio button because you are instantiating a VNF with VIM.

Note: The radio button defaults to the selection made when used previously.

4. Click **Next**.
The Select a VNFD dialog box displays.

Select a VNFD

VNF Instance :	dc_1
VNF :	microblog
Package :	microblog_v1

Name	Creation Date
<input type="checkbox"/> vnf1	Feb 7, 2017 7:21:52 AM

CANCEL
BACK
NEXT

Figure 59: Select a VNFD Dialog Box

5. Select a VNFD for the VNF instance, for example vnf1.
6. Click **Next**.

The Select a Point of Presence (PoP) dialog box displays.

Select the Point of Presence (PoP)

VNF Instance :	dc_1
VNF :	microblog
Package :	microblog_v1
VNFD :	vnfd1

Point of Presence *	
default	▼

CANCEL
BACK
INSTANTIATE

Figure 60: Select a Point of Presence (PoP) Dialog Box

This field is for future use so only the default setting is available.

7. Click **Instantiate**.

An instantiation request is sent and the dialog box closes. The VNF instance card shows it's status as Being Instantiated. After the request is processed completely, the VNF instance card shows the status as "Rollback Succeeded" assuming that the VNF instantiated successfully.

Note: The status shows as Instantiate Failed if the instantiation request is not successful.

Upgrading VNFs

To upgrade an instantiated VNF to a newer software version or a new configuration, you run the upgrade_vnf workflow. These workflows can be run only on VNFs that are already instantiated.

You can upgrade a VNF from the CLI, the API, or from the Openet Weaver VNF-M GUI.

The optional additional_parameters parameter allows you to pass generic parameters in JSON format through the workflow submit command to the procedural level on Resource Agents. These parameters are used to extend the command's flexibility to support end-user scenarios.

Note: To ensure a successful upgrade of an instantiated VNF, follow the [VNF Upgrade Procedure](#).

The upgrade_vnf workflow is shown below.



Figure 61: upgrade_vnf Process Flow

Related concepts

[Setting Environment Variables](#) on page 133

[Openet Weaver VNF-M API Reference](#) on page 468

Openet Weaver REST API reference guides

[Openet Weaver VNF-M CLI Reference](#) on page 469

Openet Weaver CLI command reference guides

Upgrading a VNF from the CLI

To upgrade a VNF upgrade_vnf should be used for the workflow_type parameter with the command shown below.

```
ovlm-fe workflow submit -t <tenant_id> -v <vnf_id> -i <vnf_instance_id> -y
upgrade_vnf -p <vnf_template_id> [-restart <true | false>] [-sync <sync>]
[-timeout <seconds>] [-o <execution_order>] [-force <force_value>] [-a
'{<additional_parameter1, additional_parameter2,...>}']
```

Note: All parameters are described on the main page for this section, [Running VNF Workflows](#).

The following CLI command example performs an upgrade of the out-of-the-box VNF called microblog, with vnf_instance_id set to dc_1, the tenant set to default, the workflow_type set to upgrade_vnf, the vnf_template_id set to microblog_v1 and the vnf_id set to microblog. The -a parameter is used to pass a parameter to the Resource Agent so that a snapshot is taken of the VNF before the upgrade.

```
ovlm-fe workflow submit -t default -v microblog -i dc_1 -y upgrade_vnf \
-p microblog_v1 -a '{"take_a_snapshot": "false"}'
```

Note: Be sure the baseurl environment variables are set before using CLI commands.

The optional -a parameter allows you to pass additional, generic parameters in JSON format through the workflow submit command to the procedural level on Resource Agents. These parameters are used to extend the command's flexibility to support end-user scenarios. In this example, the take_a_snapshot parameter is used to take a snapshot of the VNF instance database before applying the update changes.

Upgrading a VNF Using REST API

To upgrade a VNF using REST API, post a workflow request using the upgrade_vnf workflow_type parameter. The API request takes the following format:

```
POST http://<ovlm-workflow-fe-url>
/api/v2/tenants/<tenant_id>/vnfs/<vnf_id>/vnf_instances/<vnf_instance_id>/workflow_instances?sy
nc=1&timeout=100
{
  "workflow_type": "upgrade_vnf",
  "vnf_template_id": <package_id>,
  ["force": <force_value>],
  ["execution_order": <sequential|parallel>],
  ["additional_parameters": <additional_parameters>]
}
```

Note: All parameters are described on the main page for this section, [Running VNF Workflows](#).

The following is an example of upgrading a VNF to the microblog_v2 package using curl.

```
OVLMFE_BASEURL=http://10.3.18.22:28085
tenant_id=default
vnf_id=microblog
vnf_instance_id=dc_1
echo " - upgrade - "
curl -i \
-H "Content-Type: application/json" \
-X POST \
-d '{"workflow_type": "upgrade_vnf", "vnf_template_id": "microblog_v2", "additional_parameters": "{\"take_a_snapshot\": \"true\"}" }' \
```

```
$OVLMFE_BASEURL/api/v2/tenants/${tenant_id}/vnfs/${vnf_id}/vnf_instances/${vnf_instance_id}/workflow_instances?sync=1
```

The following is an example response to the API call.

```
HTTP/1.1 200 OK
Date: Thu, 24 Aug 2017 12:09:02 GMT
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: POST, GET, OPTIONS, DELETE
Access-Control-Max-Age: 3600
Access-Control-Allow-Headers: pragma,data,cache-control, access-control-allow-origin,x-requested-with, authorization, Content-Type, Authorization, credential, X-XSRF-TOKEN
X-Application-Context: application:30001
Content-Type: application/json; charset=UTF-8
Transfer-Encoding: chunked
Server: Jetty(9.2.14.v20151106)
{
  "data": {
    "workflow_instance_status": "COMPLETED",
    "success_msg": "Upgrade of VNF has completed.",
    "flow_result": [
      {
        "vnfc_instances": [
          {
            "vnfc_id": "blogserver",
            "vnfc_instance_id": "1",
            "hostname": "192.168.122.59",
            "exit_code": "0",
            "response": {
              "health_status": "UP",
              "output": ["Upgrade has completed."]
            }
          },
          {
            "vnfc_id": "blogserver",
            "vnfc_instance_id": "2",
            "hostname": "192.168.122.199",
            "exit_code": "0",
            "response": {
              "health_status": "UP",
              "output": ["Upgrade has completed."]
            }
          },
          {
            "vnfc_id": "nginx",
            "vnfc_instance_id": "1",
            "hostname": "192.168.122.199",
            "exit_code": "0",
            "response": {
              "health_status": "UP",
              "output": ["Upgrade has completed."]
            }
          },
          {
            "vnfc_id": "postgres",
            "vnfc_instance_id": "1",
            "hostname": "192.168.122.199",
            "exit_code": "0",
            "response": {
              "health_status": "UP",
              "output": ["Upgrade is not performed as no template changes were found for this VNFC"]
            }
          }
        ]
      }
    ]
  }
}
```

```

        },
        "request": "http://ovlm-vm:8080/api/v2/tenants/default/
vnfs/microblog/vnf_instances/dc_1/workflow_instances",
        "method": "POST"
},
"status": 200
}

```

In the following example the force flag is used to signify that processing the upgrade continues on every VNFC in the VNF, even when there are no changes detected between the previous version and the new version. Additionally, execution_order is set to parallel so that VNFCs are upgraded simultaneously.

```

OVLMFE_BASEURL=http://10.3.18.22:28085
tenant_id=default
vnf_id=microblog
vnf_instance_id=dc_1
echo " - upgrade -"
curl -i \
-H "Content-Type: application/json" \
-X POST \
-d '{"workflow_type":"upgrade_vnf","vnf_template_id":"microblog_v2", "force":
[{"1"}], "execution_order":"parallel", "additional_parameters":
"\"{\\"take_a_snapshot\\\": \\"true\\"}\" }' \
$OVLMFE_BASEURL/api/v2/tenants/${tenant_id}/vnfs/${vnf_id}/vnf_i
nstances/${vnf_instance_id}/workflow_instances?sync=1

```

The following is an example response to the API call.

```

HTTP / 1.1 200 OK
Date: Wed, 23 Aug 2017 12: 37: 41 GMT
Access - Control - Allow - Origin: *
Access - Control - Allow - Methods: POST, GET, OPTIONS, DELETE
Access - Control - Max - Age: 3600
Access - Control - Allow - Headers: pragma, data, cache - control, access -
control - allow - origin, x - requested - with , authorization, Content - Type,
Authorization, credential, X - XSRF - TOKEN
X - Application - Context: application: 28085
Content - Type: application / json;
charset = UTF - 8
Transfer - Encoding: chunked
Server: Jetty(9.2.14.v20151106)
{
  "data": {
    "workflow_instance_status": "COMPLETED",
    "success_msg": "Upgrade VNF has completed.",
    "flow_result": {
      "health_status": "UP",
      "vnfc_instances": [
        {
          "vnfc_id": "blogserver",
          "vnfc_instance_id": "1",
          "hostname": "192.168.122.220",
          "exit_code": "0",
          "response": {
            "health_status": "UP"
          }
        },
        {
          "vnfc_id": "blogserver",
          "vnfc_instance_id": "2",
          "hostname": "192.168.122.105",
          "exit_code": "0",
          "response": {

```

```

        "health_status": "UP"
    }
},
{
    "vnfc_id": "nginx",
    "vnfc_instance_id": "1",
    "hostname": "192.168.122.180",
    "exit_code": "0",
    "response": {
        "health_status": "UP"
    }
},
{
    "vnfc_id": "postgres",
    "vnfc_instance_id": "1",
    "hostname": "192.168.122.72",
    "exit_code": "0",
    "response": {
        "health_status": "UP"
    }
}
]
},
{
    "request": "http://ovlm-vm:8080/api/v2/tenants/default/vnfs/microblog/vnf_instances/dc_1/workflow_instances",
    "method": "POST"
},
{
    "status": 200
}
}

```

You can use -a additional_parameters to extend the functionality of the upgrade workflow as in the following example, where a parameter is passed to take a snapshot of the VNF prior to the upgrade.

```

POST http://<ovlm-workflow-fe-url>/api/v2/tenants/default/vnfs/microblog/vnf_instances/dc_1/workflow_instances
{
    "workflow_type": "upgrade_vnf",
    "additional_parameters": {
    }
}

```

Upgrading a VNF Using the VNF-M GUI

To upgrade a VNF, follow these steps:

1. On the VNF Lifecycle page, locate the active VNF you intend to upgrade, click the more options icon



and select **Upgrade**. Scroll down the page if required.

Note: An active VNF is any VNF that is not inactive, regardless of status.

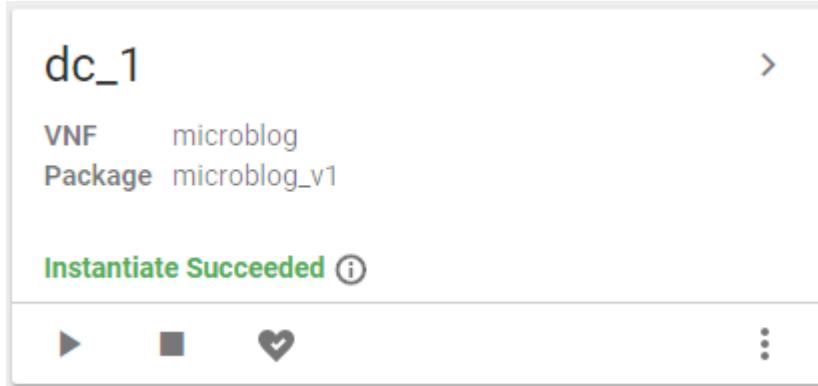


Figure 62: Active VNF

The Select a VNF Package to Upgrade dialog box is displayed.

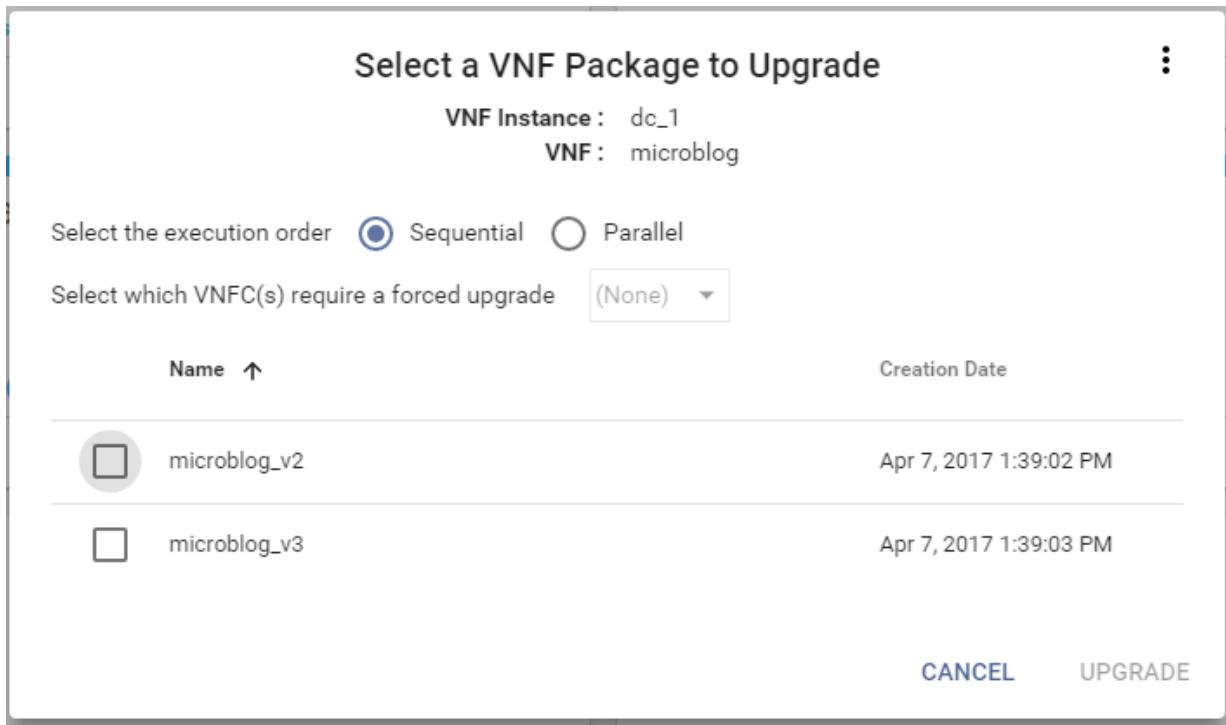


Figure 63: Select a VNF Package to Upgrade Dialog Box

2. Select an execution order by clicking the **Sequential** or **Parallel** radio button.
3. To force an upgrade to a VNFC when the system determines that it results in no change to the VNFC, click the **Select which VNFC(s) require a forced upgrade** drop-down box and select each VNFC to upgrade, or select **(All)** to select all the VNFCs associated with the VNF.
4. Click a VNF package in the **VNF Packages** pane, for example microblog_v2.
5. To configure additional parameters for a script you are running on the Resource Agent related to upgrading the VNF, click the icon in the top-right of the dialog box and select **Add parameters**.

The dialog box displays a field into which you can enter additional parameters. In some cases there might be predefined parameters, which will display when you select **Add Parameters**. Predefined parameters can be overwritten.

Note: For information about how to predefine Additional Parameters, see [Managing Additional Parameters for the VNF-M GUI](#).

Parameters must be entered in JSON format as in the following example.

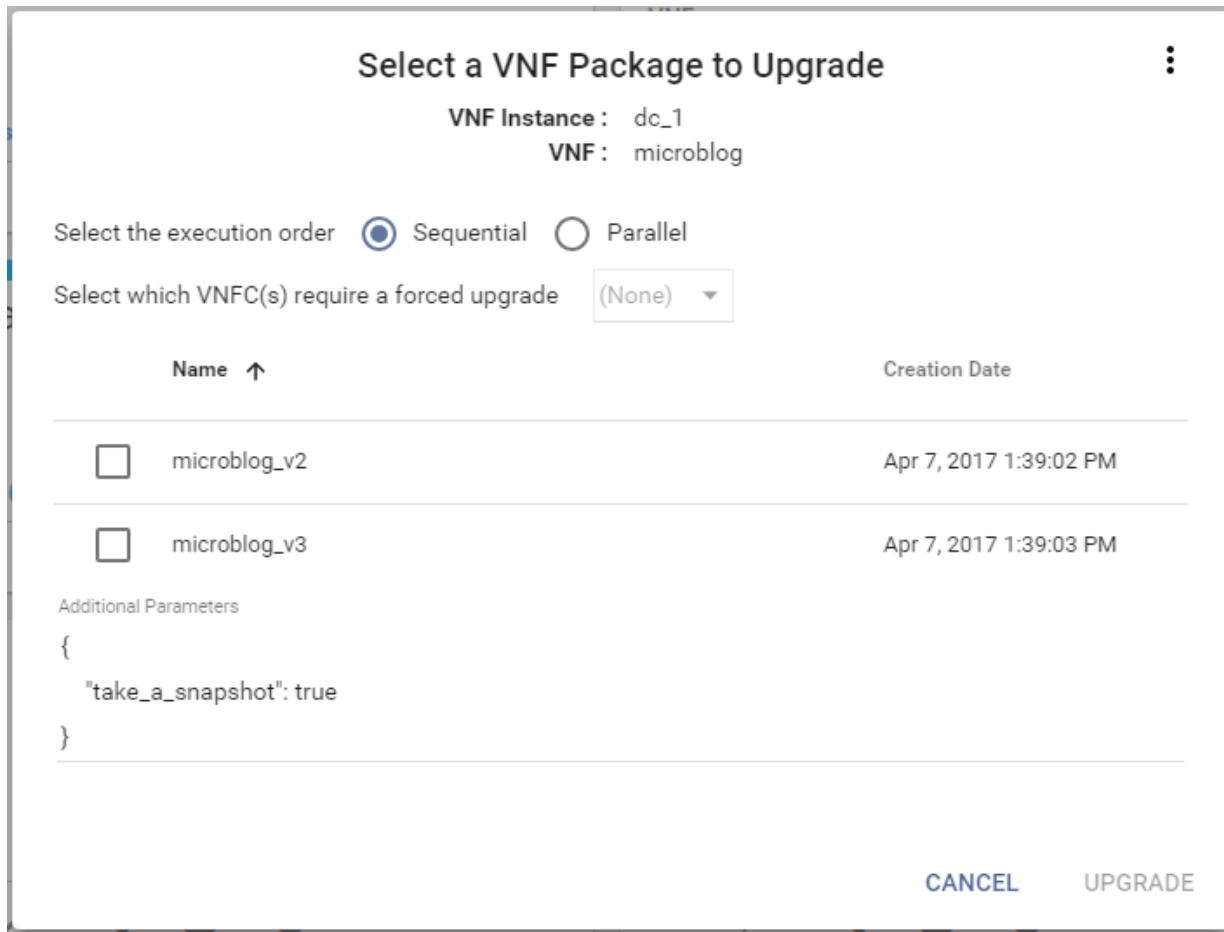


Figure 64: Additional Parameters in Select a VNF Package to Upgrade Dialog Box

Note: To remove additional parameters after adding them, click the top-right icon again and select **Remove parameters**.

6. Click Upgrade.

The dialog box closes, the upgrade begins, and the status changes from green to blue with the text "Upgrade In-Progress".

Note: In the event that the upgrade fails the VNF card is moved to the Alerts section and the status turns orange with the text "Upgrade Failed".

VNF Upgrade Procedure

In order to upgrade an instantiated VNF successfully, create a package to use for the upgrade after making sure that the following files are prepared and uploaded to Configuration Manager so they can be used by the package:

- VNFC packages that must be upgraded or are clean installs
- Configuration files
- The VNF upgrade meta.yml file

The package you create from this procedure can be used with any VNF regardless of whether it was instantiated with VIM.

The command examples in this procedure use the following parameters:

- tenant_id: default
- vnf_id: microblog_v2
- vnf_instance_id: dc_1

- vnf_template_id: upgrade_package
- vnfc_id: nginx

1. Prepare the meta.yml file.

You provide information about the VNFC packages are affected when the upgrade_vnf workflow is run in the upgrade_vnfc section of the meta.yml file. This information includes VNFC packages that are installed (install_packages parameter), VNFC packages that are uninstalled (uninstall_packages parameter), and configuration files that will be removed(configurations parameter). None of the parameters in the upgrade_vnfc section are mandatory. The following is an example of a meta.yml file upgrade_vnfc section:

```
upgrade_vnfc:
  timeout: 60
  parameters:
    services:
      - nginx
    uninstall_packages:
      - packages/nginx-1.8.1-1.el7.ngx.x86_64.rpm
    install_packages:
      - packages/nginx-1.11.3-1.el7.ngx.x86_64.rpm
  procedure_name: upgrade_vnfc_v2.py
```

An example of the complete file would is as follows

```
dynamic_config_vnfc:
  timeout: 60
  parameters:
    configurations:
      - configuration/nginx.conf.jinja
      - configuration/nginx_vnfc_monitoring.sh.jinja
  procedure_name: dynamic_config_vnfc_v2.py
install_vnfc:
  timeout: 60
  parameters:
    packages:
      - packages/nginx-1.8.1-1.el7.ngx.x86_64.rpm
  procedure_name: install_rpm_v2.py
uninstall_vnfc:
  timeout: 60
  parameters:
    configurations:
      - /etc/nginx.conf
    packages:
      - packages/nginx-1.8.1-1.el7.ngx.x86_64.rpm
  procedure_name: uninstall_pkg_v2.py
is_vnfc_down:
  timeout: 10
  parameters:
    services:
      - nginx
  procedure_name: is_vnfc_down_v2.py
is_vnfc_up:
  timeout: 10
  parameters:
    services:
      - nginx
  procedure_name: is_vnfc_up_v2.py
start_vnfc:
  timeout: 10
  parameters:
    services:
      - nginx
```

```

procedure_name: start_vnfc_v2.py
stop_vnfc:
  timeout: 10
parameters:
  services:
    - nginx
procedure_name: stop_vnfc_v2.py
update_config_vnfc:
parameters:
- destFile: /etc/nginx/nginx.conf
  group: root
  permission: '0644'
  srcFile: configuration/nginx.conf
  user: root
- destFile: /etc/nginx/weaver_monitor/nginx_vnfc_monitoring.sh
  group: root
  permission: '0754'
  srcFile: configuration/nginx_vnfc_monitoring.sh
  user: root
- destFile: /var/spool/cron/root
  group: root
  permission: '0644'
  srcFile: configuration/cron_job
  user: root
procedure_name: update_config_vnfc_v2.py
heal_vnfc:
  timeout: 60
parameters:
  services:
    - nginx
procedure_name: heal_vnfc_v2.py
upgrade_vnfc:
  timeout: 60
parameters:
  services:
    - nginx
  uninstall_packages:
    - packages/nginx-1.8.1-1.el7.ngx.x86_64.rpm
  install_packages:
    - packages/nginx-1.11.3-1.el7.ngx.x86_64.rpm
procedure_name: upgrade_vnfc_v2.py

```

- Upload the metadata file to Configuration Manager using the CLI command `ovlm-fe vnfc_metadata upload -t <tenant_id> -v <vnf_id> -c <vnfc_id> -f <metadata_file>`

The following is an example of the command.

```
ovlm-cm vnfc_metadata upload -t default -v microblog_v2 -c nginx -f meta.yml
```

Note: Be sure the baseurl environment variables are set before using CLI commands. There are API versions of the commands in this topic.

The command should produce the following result if successful.

```

Uploading Files:
  meta.yml
data:
  method: POST
  request: http://ovlm-mgmt-dub-all-lane.openet-dublin:28082/api/v2/tenants/default/vnfs/microblog_v2/vnfc/nginx/metadata
status: 200

```

- Upload the required VNFC packages for the package using the command `ovlm-fe vnfc_package upload -t <tenant_id> -v <vnf_id> -c <vnfc_id> -f <package_file(s)>`

The following is an example of the command.

```
ovlm-cm vnfc_package upload -t default -v microblog_v2 -c nginx -f
nginx-1.11.0-1.el6.ngx.x86_64.rpm
```

The command should produce the following result if successful.

```
Uploading Files:
  nginx-1.11.0-1.el6.ngx.x86_64.rpm
data:
  method: POST
  request: http://ovlm-mgmt-dub-all-lane.openet-dublin:28082/api/v2/tenants/d
efault/vnfs/microblog_v2/vnfc_packages
status: 200
```

- Create a package to use in the `upgrade_vnf` workflow using the CLI command `ovlm-fe template create -t <tenant_id> -v <vnf_id> -p <vnf_template_id>`

The following is an example of the command.

```
ovlm-cm template create -t default -v microblog_v2 -p upgrade_package
```

The command should produce the following result if successful.

```
data:
  method: POST
  request: http://ovlm-mgmt-dub-all-lane.openet-dublin:28082/api/v2/tenants/d
efault/vnfs/microblog_v2/templates
status: 200
```

- Run the `upgrade_vnf` workflow using the CLI command `ovlm-fe workflow submit -t <tenant_id> -v <vnf_id> -i <vnf_instance_id> -p <vnf_template_id> -y <workflow_type> [-sync 0|1]`

The following is an example of the command.

```
ovlm-fe workflow submit -t default -v microblog_v2 -i dc_1 -y upgrade_vnf -p
upgrade_package
```

The command should produce the following result if successful.

```
data:
  workflow_instance_status: COMPLETED
  success_msg: Upgrade of VNF has completed.
  flow_result:
    health_status: UP
    vnfc_instances:
      - vnfc_id: blogserver
        vnfc_instance_id: '1'
        hostname: 192.168.122.59
        exit_code: '0'
        response:
          health_status: UP
          output:
            - Upgrade has completed.
      - vnfc_id: blogserver
        vnfc_instance_id: '2'
        hostname: 192.168.122.199
        exit_code: '0'
```

```

response:
  health_status: UP
  output:
    - Upgrade has completed.
- vnfc_id: nginx
  vnfc_instance_id: '1'
  hostname: 192.168.122.199
  exit_code: '0'
  response:
    health_status: UP
    output:
      - Upgrade has completed.
- vnfc_id: postgres
  vnfc_instance_id: '1'
  hostname: 192.168.122.199
  exit_code: '0'
  response:
    health_status: UP
    output:
      - Upgrade is not performed as no template changes were found for this
VNFC
  request: http://ovlm-vm:8080/api/v2/tenants/default/vnfs/microblog/vnf_instances/dc_1/vnfc_instances/workflow_instances
  method: POST
status: 200

```

Related concepts

[Creating VNFs](#) on page 173

Using the Out of the Box procedures and flows, create a VNF package and deploy it.

[Setting Environment Variables](#) on page 133

[Openet Weaver VNF-M API Reference](#) on page 468

Openet Weaver REST API reference guides

[Openet Weaver VNF-M CLI Reference](#) on page 469

Openet Weaver CLI command reference guides

Updating VNFs

To change the configuration of an instantiated VNF you run the update_vnf workflow. This workflow can be run only on VNFs that are already instantiated.

You can update a VNF from the CLI , the API, or from the Openet Weaver VNF-M GUI.

Note: To ensure a successful update of an instantiated VNF, follow the [VNF Update Procedure](#) before running the update_vnf workflow.

For this workflow, the optional force parameter specifies whether an update continues on VNFC instances regardless of whether the update impacts the instance. You can use the parameter to force an update on all VNFCs or on a specified list of VNFCs.

The optional restart parameter lets you choose whether you to restart the VNF after updating it. Since the VNF restarts by default, this parameter is required only when you decide not to restart the VNF.

Note: See the VNF [Update Without Restart Procedure](#) for more information on the steps to follow in addition to the workflow submit command to ensure a successful update.

The update_vnf workflow is shown below.

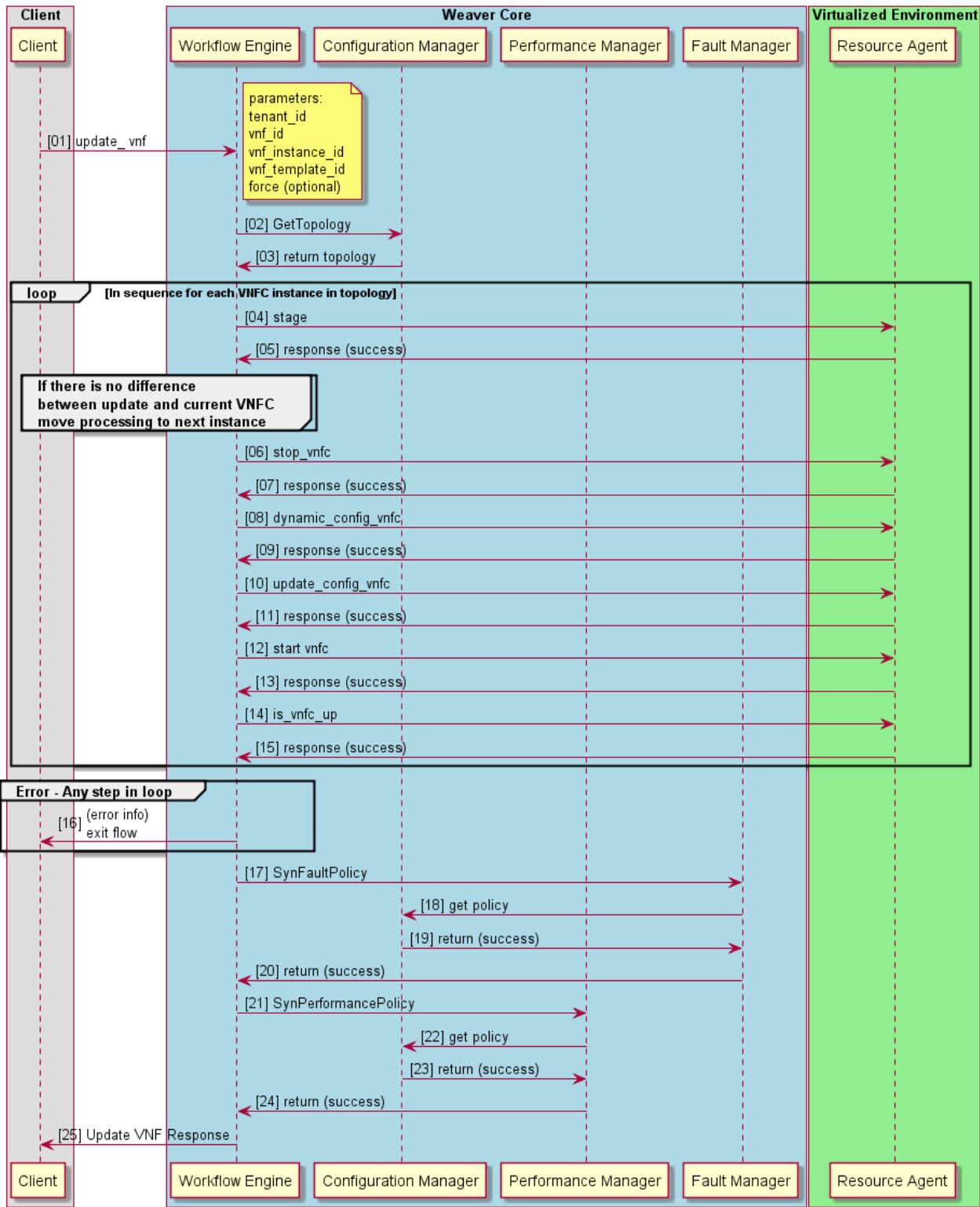


Figure 65: update_vnf Process Flow

Related concepts

[Setting Environment Variables](#) on page 133

[Openet Weaver VNF-M API Reference](#) on page 468

[Openet Weaver REST API reference guides](#)

[Openet Weaver VNF-M CLI Reference](#) on page 469
Openet Weaver CLI command reference guides

Updating a VNF from the CLI

To update a VNF from the CLI, use the update_vnf workflow_type parameter as shown below.

Note: To ensure a successful update of an instantiated VNF, follow the [VNF Update Procedure](#) before running the update_vnf workflow.

```
ovlm-fe workflow submit -t <tenant_id> -v <vnf_id> -i <vnf_instance_id> -y
update_vnf -p <vnf_template_id>
[-o <execution_order>] [-restart <true | false>] [-sync <sync>] [-timeout
<seconds>] [-force <force_value>]
[-a '{<additional_parameter1, additional_parameter2,...>}']
```

Note: All parameters are described on the main page for this section, [Running VNF Workflows](#).

The CLI command in the following example updates the microblog VNF after stopping all VNFC instances on the Resource Agents. In this example the command parameters are the default tenant, the microblog vnf, a vnf_instance_id of dc_1, and a vnf_template_id of microblog_v1. The execution order is set to parallel, so all VNFCs are updated simultaneously, which means the VNF is unavailable until update completes. Assuming the workflow completes successfully, the VNF restarts. The force flag is set to 1, so processing continues for all VNFCs regardless of whether there is a change detected between the update version and the current version during staging.

```
ovlm-fe workflow submit -t default -v microblog -i dc_1 -y update_vnf -p
microblog_v1 -o parallel -restart true -force 1
```

Note: Be sure the baseurl environment variables are set before using CLI commands.

If the -force parameter is not present in the command the default value of 0 is used.

The optional restart parameter lets you choose whether to restart the VNF after updating it. Since the VNF restarts by default, this parameter is required only when decide to restart the VNF.

Note: See the VNF [Update Without Restart Procedure](#) for more information on the steps to follow in addition to the workflow submit command to ensure a successful update.

The output of the command is a workflow_instance_id generated by the system similar to the following example.

```
data:
  method: POST
  request: http://ovlm-vm:8080/api/v2/tenants/default/vnfs/microblog/vnf_instances/dc_1/workflow_instances
  workflow_instance_id: '112600123'
status: 201
```

Updating a VNF Using REST API

To update a VNF using REST API, post a workflow request using the update_vnf workflow_type parameter. The API request takes the following format.

```
POST http://<ovlm-workflow-fe-url>
/api/v2/tenants/<tenant_id>/vnfs/<vnf_id>/vnf_instances/<vnf_instance_id>/workflow_instances?sy
nc=1&timeout=100
{
  "workflow_type": "update_vnf",
  "vnf_template_id": <update_package_id>,
  ["execution_order": <sequential|parallel>],
  ["restart": <false | true>],
  ["force": <force_value>],
```

```

        ["additional_parameters": <additional_parameters>]
}

```

Note: All parameters are described on the main page for this section, [Running VNF Workflows](#).

Note: To ensure a successful update of an instantiated VNF, follow the [VNF Update Procedure](#) before running the update)vnf workflow.

The optional restart parameter lets you choose whether to restart the VNF after updating it. Since the VNF restarts by default, this parameter is required only when you decide to restart the VNF.

Note: See the VNF [Update Without Restart Procedure](#) for more information on the steps to follow in addition to the workflow submit command to ensure a successful update.

The following is an example of updating a VNF using curl. The request specifies that the command runs in synchronous mode, all VNFCs in the VNF are updated simultaneously, and the VNF does not restart after the update.

```

OVLMFE_BASEURL=http://10.3.18.22:28085
tenant_id=default
vnf_id=microblog
vnf_instance_id=dc_1
echo " - update - "
curl -i \
-H "Content-Type: application/json" \
-X POST \
-d '{"workflow_type":"update_vnf","vnf_template_id":"microblog_v1","restart":"false", "execution_order":"parallel"}' \
$OVLMFE_BASEURL/api/v2/tenants/${tenant_id}/vnfs/${vnf_id}/vnf_instances/${vnf_instance_id}/workflow_instances?sync=1

```

The following is an example response to the API call.

```

HTTP / 1.1 200 OK
Date: Mon, 20 Feb 2017 10: 07: 53 GMT
X - Application - Context: application: 28085
Content - Type: application / json;
charset = UTF - 8
Transfer - Encoding: chunked
Server: Jetty(9.2.14.v20151106)
{
  "data": {
    "workflow_instance_status": "COMPLETED",
    "flow_result": {
      "success": "\"VNF Update passed successfully on all servers.
Performance Manager is not enabled thus Sync Policy is skipped.
Fault Manager is not enabled thus Sync Policy is
skipped.\\""
    },
    "request": "http://10.3.18.22:28085/api/v2/tenants/defa
ult/vnfs/microblog/vnf_instances/dc_1/workflow_instances",
    "method": "POST"
  },
  "status": 200
}

```

In the next example the force flag is used to signify that processing the update continues on every VNFC in the VNF, even when there are no changes detected between the previous version and the new version. Additionally, restart is set to true, so the VNF restarts after the update completes.

Note: By default, the VNF restarts after an update so the restart parameter does not need to be specified.

```
OVLMFE_BASEURL=http://10.3.18.22:28085
tenant_id=default
vnf_id=microblog
vnf_instance_id=dc_1
echo " - update - "
curl -i \
-H "Content-Type: application/json" \
-X POST \
-d '{"workflow_type":"update_vnf","vnf_template_id":"microblog_v2","restart":true}' \
$OVLMFE_BASEURL/api/v2/tenants/${tenant_id}/vnfs/${vnf_id}/vnf_instances/${vnf_instance_id}/workflow_instances?sync=1
```

The following is an example response to the API call.

```
HTTP / 1.1 200 OK
Date: Mon, 20 Feb 2017 10: 21: 30 GMT
X - Application - Context: application: 28085
Content - Type: application / json;
charset = UTF - 8
Transfer - Encoding: chunked
Server: Jetty(9.2.14.v20151106)
{
  "data": {
    "workflow_instance_status": "COMPLETED",
    "flow_result": {
      "success": "\"VNF Update passed successfully on all servers.
Performance Manager is not enabled thus Sync Policy is skipped. Fault Manager
is not enabled thus Sync Policy is skipped.\""
    },
    "request": "http://10.3.18.22:28085/api/v2/tenants/defa
ult/vnfs/microblog/vnf_instances/dc_1/workflow_instances",
    "method": "POST"
  },
  "status": 200
}
```

Updating a VNF Using the VNF-M GUI

To update a VNF follow these steps:

1. On the VNF Lifecycle page, locate the active VNF you intend to update, click the more options icon



and select **Update**. Scroll down the page if required.

Note: An active VNF is any VNF that is not inactive, regardless of status.

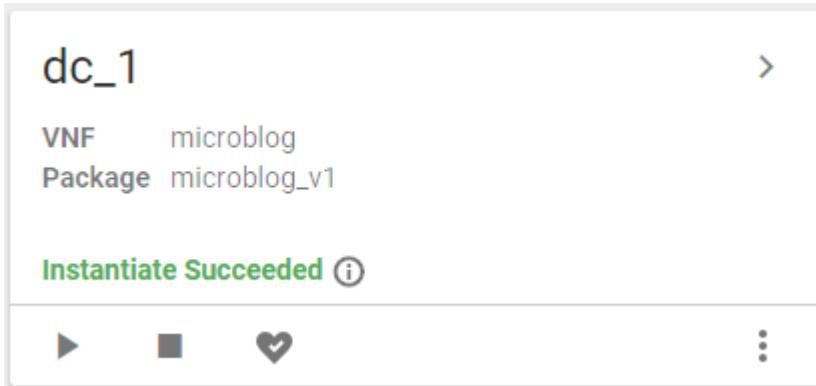


Figure 66: Active VNF

The Select a VNF Package to Update dialog box is displayed.

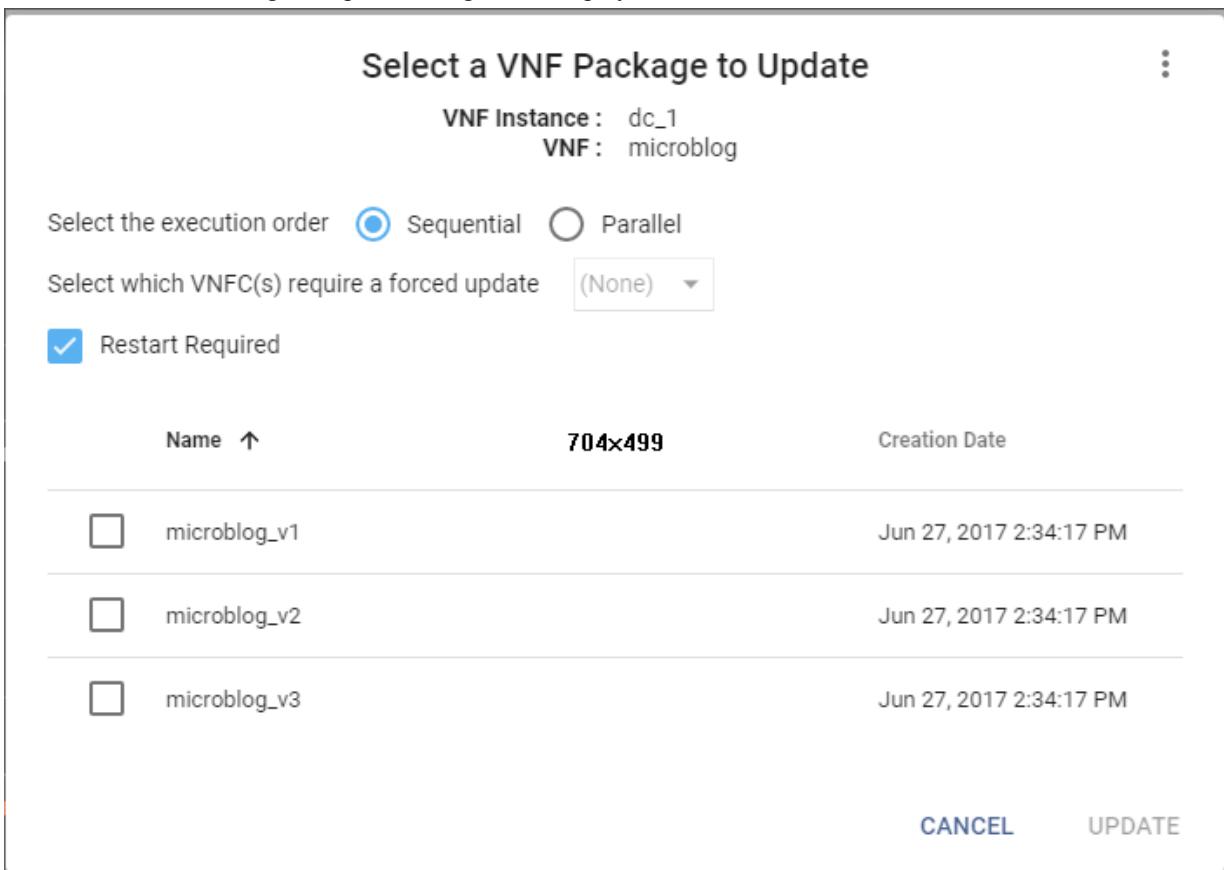


Figure 67: Select a VNF Package to Update Dialog Box

2. Clear the **Restart Required** check box if you intend to prevent the VNF from restarting automatically as part of the update process.
Note: The **Restart Required** check box is selected by default.
3. To force an update to a VNFC when the system determines that it results in no change to the VNFC, click the **Select which VNFC(s) require a forced update** drop-down box and select each VNFC to upgrade, or select **(All)** to select all the VNFCs associated with the VNF.
4. Click a VNF package in the **VNF Packages** pane, for example **microblog_v1**.

- To configure additional parameters for a script you are running on the Resource Agent related to updating the VNF, click the icon in the top-right of the dialog box and select **Add parameters**.

The dialog box displays a field into which you can enter additional parameters. In some cases there might be predefined parameters, which will display when you select **Add Parameters**. Predefined parameters can be overwritten.

Note: For information about how to predefine Additional Parameters, see *Managing Additional Parameters for the VNF-M GUI*.

Parameters must be entered in JSON format as in the following example.

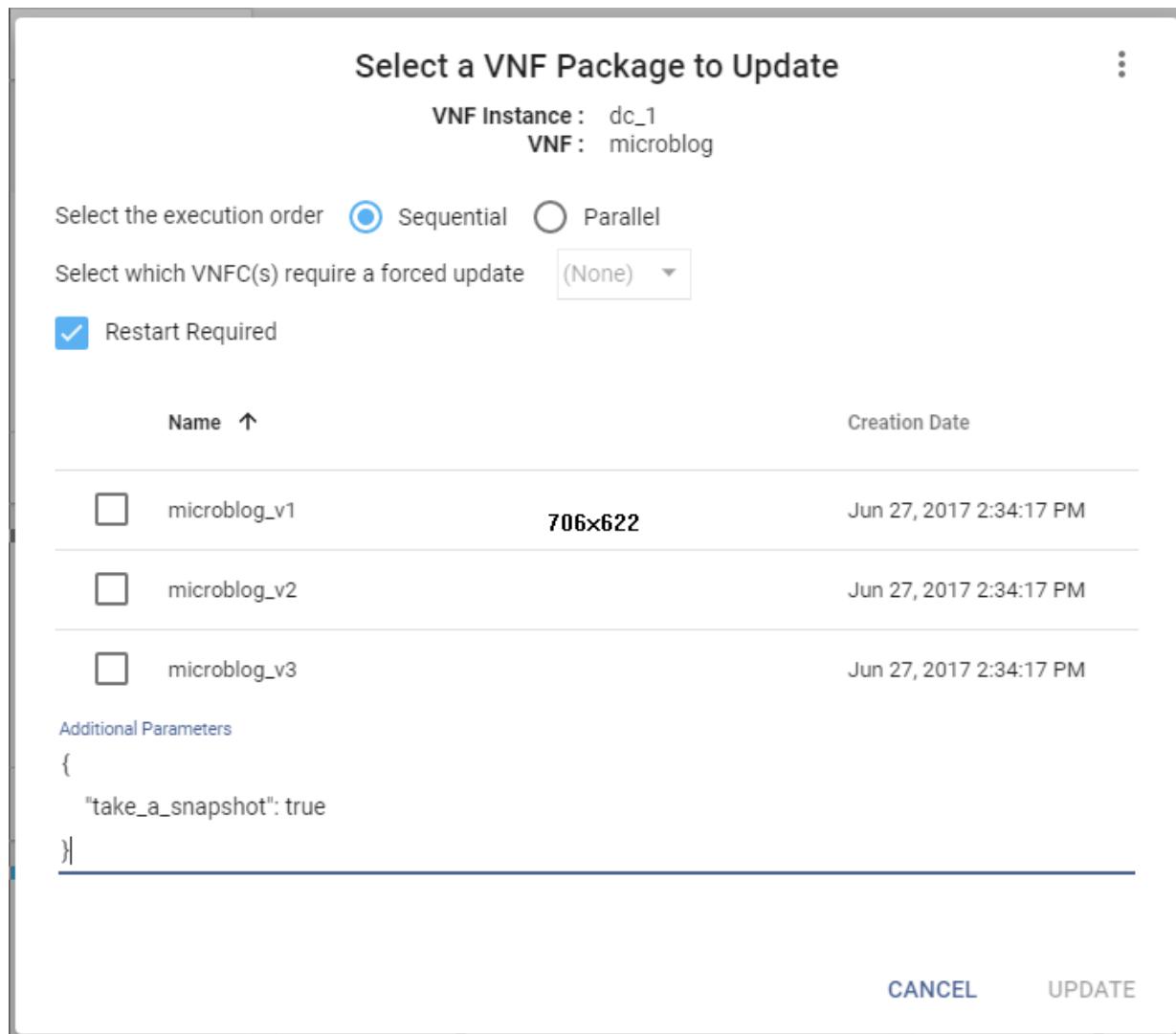


Figure 68: Additional Parameters in Select a VNF Package to Update Dialog Box

Note: To remove additional parameters after adding them, click the top-right icon again and select **Remove parameters**.

- Click **Update**.

The dialog box closes, the update begins and the status changes from green to blue with the text "Update In-Progress".

Note: In the event that the update fails the VNF card is moved to the Alerts section and the status turns orange with the text "Update Failed".

VNF Update Without Restart Procedure

It is possible to update a VNF without restarting the VNF. When you do, the update workflow omits the stop_vnfc and start_vnfc stages. The following stages run as specified in the workflow:

- stage
- dynamic_config
- update_config
- is_vnfc_up

Before you run the workflow, prepare the configuration data and meta.yml files as described below, then run the update_vnf workflow as shown in the final step. This is slightly different from the main [VNF Update Procedure](#).

In order to update an instantiated VNF successfully, create a package to use for the update after making sure configuration files are prepared and uploaded to Configuration Manager so they can be used by the package.

The package you create from this procedure can be used with any VNF regardless of whether it was instantiated with VIM.

The command examples in this procedure use the following parameters:

- tenant_id: default
- vnf_id: microblog
- vnf_instance_id: dc_1
- vnf_template_id: update_nginx_package
- vnfc_id: nginx

In the example, the VNF is being updated for Nginx listening ports.

1. Prepare the configuration file.

In this example the contents of the nginx_update.conf.jinja file is as follows:

```
events {
    worker_connections 1024;
}
error_log /var/log/nginx/error.log;
http {
    upstream microblog {
        server rt2-dub-all-lane.anycom ;
        server rt3-dub-all-lane.anycom ;
    }
    server {
        listen 0.0.0.0:80;
        listen 0.0.0.0:8080;
        location / {
            proxy_pass http://microblog;
        }
    }
}
```

2. Prepare the update_config_vnfc section of the metadata/meta.yml file.

In this example the contents of the update_config_vnfc section is configured as follows:

```
update_config_vnfc:
parameters:
- destFile: /etc/nginx/nginx.conf
  group: root
  permission: '0644'
  srcFile: configuration/nginx_update.conf
  user: root
procedure_name: update_config_vnfc_v2.py
```

3. Upload the configuration file to Configuration Manager using the CLI command `ovlm-cm vnfc_configuration upload -t <tenant_id> -v <vnf_id> -c <vnfc_id> -f <configuration_file>`
The following is an example of the command.

```
ovlm-cm vnfc_configuration upload -t default -v microblog -c nginx -f
nginx_update.conf.jinja
```

Note: Be sure the baseurl environment variables are set before using CLI commands. There are API versions of the commands in this topic.

The command should produce the following result if successful.

```
Uploading Files:
    nginx_update.conf.jinja
data:
  method: POST
  request: http://ovlm-mgmt-dub-all-lane.anycom:28082/api/v2/tenants/default/
vnfs/microblog/vnfcns/nginx/configuration
status: 200
```

4. Upload the meta.yml file to Configuration Manager using the CLI command `ovlm-cm vnfc_metadata upload -t <tenant_id> -v <vnf_id> -c <vnfc_id> -f <metadata_file>`

The following is an example of the command.

```
ovlm-cm vnfc_metadata upload -t default -v microblog -c nginx -f meta.yml
```

Note: Be sure the baseurl environment variables are set before using CLI commands. There are API versions of the commands in this topic.

The command should produce the following result if successful.

```
Uploading Files:
    meta.yml
data:
  method: POST
  request: http://ovlm-mgmt-dub-all-lane.openet-dublin:28082/api/v2/tenants/d
efault/vnfs/microblog/vnfcns/nginx/meta
status: 200
```

5. Create a package to use in the update_vnf workflow using the CLI command `ovlm-cm template create -t tenant_id -v vnf_id -p vnf_template_id`

The following is an example of the command.

```
ovlm-cm template create -t default -v microblog -p update_nginx_config_only
```

The command should produce the following result if successful.

```
data:
  method: POST
  request: http://ovlm-mgmt-dub-all-lane.anycom:28082/api/v2/tenants/default/
vnfs/microblog/templates
status: 200
```

6. Run the update_vnf workflow using the CLI command `ovlm-fe workflow submit -t tenant_id -v vnf_id -i vnf_instance_id -p vnf_id -y workflow_type [-sync 0|1]`

For the -p parameter you specify the VNFD as in the standard update procedure then include "restart" : "false" to indicate that the VNF is not to be restarted. The following is an example of the command.

```
ovlm-fe workflow submit -t default -v microblog -i dc_1 -y update_vnf -p
update_nginx_package -restart false
```

The command should produce the following result if successful.

```
ata:
  method: POST
  request: http://ovlm-vm:8080/api/v2/tenants/default/v
nfs/microblog/vnf_instances/dc_1/workflow_instances
  workflow_instance_id: '112600123'
  status: 201
```

VNF Update Procedure

In order to update an instantiated VNF successfully, create a package to use for the update after making sure configuration files are prepared and uploaded to Configuration Manager so they can be used by the package.

The package you create from this procedure can be used with any VNF regardless of whether it was instantiated with VIM.

The command examples in this procedure use the following parameters:

- tenant_id: default
- vnf_id: microblog
- vnf_instance_id: dc_1
- vnf_template_id: update_nginx_package
- vnfc_id: nginx

In the example, the VNF is being updated for Nginx listening ports.

1. Prepare the configuration file.

In this example the contents of the nginx.conf.jinja file is as follows:

```
events {
    worker_connections 1024;
}

error_log /var/log/nginx/error.log;

http {
    upstream microblog {

        {% for srv in topology.vnfc_ids.blogserver %}
            server {{ srv.hostname }} ;
        {% endfor %}

    }

    server {
        listen 0.0.0.0:80;
        listen 0.0.0.0:8080;

        location / {
            proxy_pass http://microblog;
        }
    }
}
```

The listening ports are being updated to use 0.0.0.0:443 and 0.0.0.0:8443. So the amended configuration file is now as follows:

```

events {
    worker_connections 1024;
}

error_log /var/log/nginx/error.log;

http {
    upstream microblog {

        {% for srv in topology.vnfc_ids.blogserver %}
            server {{ srv.hostname }} ;
        {% endfor %}

    }

    server {
        listen 0.0.0.0:443;
        listen 0.0.0.0:8443;

        location / {
            proxy_pass http://microblog;
        }
    }
}

```

- Upload the configuration file to Configuration Manager using the CLI command `ovlm-cm vnfc_configuration upload -t <tenant_id> -v <vnf_id> -c <vnfc_id> -f <configuration_file>`
The following is an example of the command.

```
ovlm-cm vnfc_configuration upload -t default -v microblog -c nginx -f nginx.conf.jinja
```

Note: Be sure the baseurl environment variables are set before using CLI commands. There are API versions of the commands in this topic.

The command should produce the following result if successful.

```

Uploading Files:
meta.yml
data:
method: POST
request: http://ovlm-mgmt-dub-all-lane.openet-dublin:28082/api/v2/tenants/default/vnfs/microblog/vnfcs/nginx/configuration
status: 200

```

- Create a package to use in the update_vnf workflow using the CLI command `ovlm-cm template create -t <tenant_id> -v <vnf_id> -p <vnf_template_id>`
The following is an example of the command.

```
ovlm-cm template create -t default -v microblog -p update_nginx_package
```

The command should produce the following result if successful.

```

data:
method: POST
request: http://ovlm-mgmt-dub-all-lane.openet-dublin:28082/api/v2/tenants/d

```

```
efault/vnfs/microblog/templates
status: 200
```

4. Run the update_vnf workflow using the CLI command `ovlm-fe workflow submit -t <tenant_id> -v <vnf_id> -i <vnf_instance_id> -p <vnfd> -y <workflow_type> [-sync 0|1]`

The following is an example of the command.

```
ovlm-fe workflow submit -t default -v microblog -i dc_1 -y update_vnf -p
update_nginx_package
```

The command should produce the following result if successful.

```
ata:
  method: POST
  request: http://ovlm-vm:8080/api/v2/tenants/default/v
nfs/microblog/vnf_instances/dc_1/workflow_instances
  workflow_instance_id: '112600123'
  status: 201
```

[Rolling Back VNFs](#)

To roll back a VNF to its previous version you run the rollback_vnf workflow. This workflow can be run only on VNFs that were upgraded. A VNF can be rolled back to the most recent previous version only.

You can roll back a VNF from the CLI, the API, or from the Openet Weaver VNF-M GUI.

Important: The rollback_vnf workflow can be used to roll back only those VNFCs that have been upgraded using the upgrade_vnf workflow.

The workflow reverts all changes introduced when the most recent upgrade_vnf workflow was run against the VNF.

The steps that are run in the workflow shown below depend on the result of the most recent upgrade_vnf workflow run against the VNF.

The rollback_vnf workflow runs all the step in the following circumstances:

- The most recent upgrade ran successfully
- An error occurred during upgrade *after* the update_config_vnfc step when running the upgrade_vnf workflow.
- The VNFC was rolled back previously with a failure.

The rollback_vnf workflow will run only is_vnfc_up and continue with start_vnfc step (if the VNFC is not running) in the following circumstances:

- An error occurred *before* or *during* the update_config_vnfc step during when running the upgrade_vnf workflow.
- The VNFC was successfully rolled back previously.

In all cases, troubleshoot the errors on the Resource Agent to prevent the error from reoccurring.

In the unlikely event that you require more information to troubleshoot the problem, you can try running the rollback_vnf workflow through the Front End GUI, which might generate a different error message.

Important: The rollback_vnf workflow will not work if you ran the update flow against the VNF since the last upgrade.

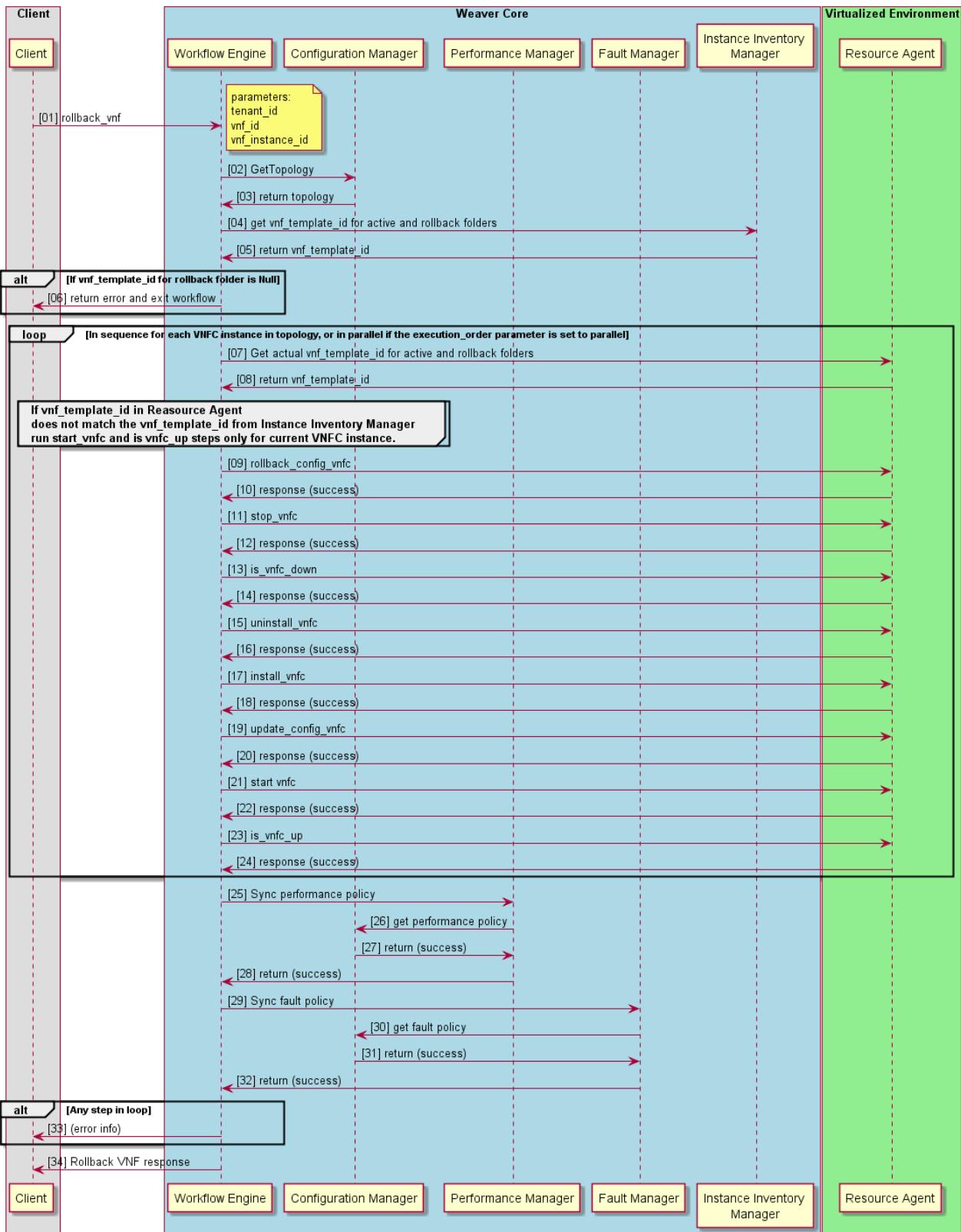


Figure 69: `rollback_vnf` Process Flow

Rolling Back a VNF from the CLI

To roll back a VNF from the CLI, use the `rollback_vnf` workflow_type parameter in the `workflow_submit` command as shown below.

```
ovlm-fe workflow submit -t <tenant_id> -v <vnf_id> -i <vnf_instance_id> -y
rollback_vnf [-restart <true | false>] [-sync <sync>]
[-timeout <seconds>] [-o <execution_order>] [-a '{<additional_parameter1,
additional_parameter2,...>}']
```

Note: All parameters are described on the main page for this section, [Running VNF Workflows](#).

The CLI command in the following example rolls back a specified VNF by reinstalling and restarting the previous version on all VNFC instances on the Resource Agents. In this example the command parameters are the default tenant, the microblog vnf, and a vnf_instance_id of dc_1. The execution_order option is set to parallel, which means that all VNFC instances are upgraded simultaneously.

Note: In previous versions of Openet Weaver the template_id was required as a parameter to the rollback workflow. That parameter is no longer required.

```
ovlm-fe workflow submit -t default -v microblog -i dc_1 -y rollback_vnf -o
parallel
```

Note: Be sure the baseurl environment variables are set before using CLI commands.

The output of the command is a workflow_instance_id generated by the system similar to the following example.

```
data:
  method: POST
  request: http://ovlm-vm:8080/api/v2/tenants/default/vnfs/microblog/vnf_instances/dc_1/workflow_instances
  workflow_instance_id: '112600123'
  status: 200
```

Rolling Back a VNF Using REST API

To roll back a VNF using REST API, post a workflow request using the `rollback_vnf` workflow_type parameter. The API request takes the following format.

```
POST http://<ovlm-workflow-fe-url>
/api/v2/tenants/<tenant_id>/vnfs/<vnf_id>/vnf_instances/<vnf_instance_id>/
  workflow_instances?sync=1&timeout=100
{
  "workflow_type": "rollback_vnf",
  ["force": <force_value>],
  ["additional_parameters": <additional_parameters>]
}
```

Note: All parameters are described on the main page for this section, [Running VNF Workflows](#).

The following is an example of rolling back a VNF using curl. The request specifies that the workflow run in synchronous mode and that VNFCs are rolled back sequentially.

Note: The order_execution value (parallel or sequential) must be the same as that used for upgrading the VNF. For example, if the upgrade was sequential, then rollback must be sequential.

```
OVLMFE_BASEURL=http://10.3.18.22:28085
tenant_id=default
vnf_id=microblog
vnf_instance_id=dc_1
echo " - rollback - "
curl -i \
```

```
-H "Content-Type: application/json" \
-X POST \
-d '{"workflow_type":"rollback_vnf", "execution_order": "sequential", "force": \
["1"]}' \
$OVLMFE_BASEURL/api/v2/tenants/${tenant_id}/vnfs/${vnf_id}/vnf_i
nstances/${vnf_instance_id}/workflow_instances?sync=1
```

The following is an example response to the API call.

```
HTTP/1.1 200 OK
Date: Thu, 24 Aug 2017 12:11:31 GMT
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: POST, GET, OPTIONS, DELETE
Access-Control-Max-Age: 3600
Access-Control-Allow-Headers: pragma,data,cache-control, access-control-allow-o
rigin,x-requested-with, authorization, Content-Type, Authorization, credential,
X-XSRF-TOKEN
X-Application-Context: application:30001
Content-Type: application/json; charset=UTF-8
Transfer-Encoding: chunked
Server: Jetty(9.2.14.v20151106)
{
  "data": {
    "workflow_instance_status": "COMPLETED",
    "success_msg": "Rollback of VNF has completed.",
    "flow_result": [
      {
        "vnfc_instances": [
          {
            "vnfc_id": "blogserver",
            "vnfc_instance_id": "1",
            "hostname": "192.168.122.59",
            "exit_code": "0",
            "response": {
              "health_status": "UP",
              "output": ["Rollback has completed."]
            }
          },
          {
            "vnfc_id": "blogserver",
            "vnfc_instance_id": "2",
            "hostname": "192.168.122.199",
            "exit_code": "0",
            "response": {
              "health_status": "UP",
              "output": ["Rollback has completed."]
            }
          },
          {
            "vnfc_id": "nginx",
            "vnfc_instance_id": "1",
            "hostname": "192.168.122.199",
            "exit_code": "0",
            "response": {
              "health_status": "UP",
              "output": ["Rollback has completed."]
            }
          },
          {
            "vnfc_id": "postgres",
            "vnfc_instance_id": "1",
            "hostname": "192.168.122.199",
            "exit_code": "0",
            "response": {
              "health_status": "UP",
              "output": ["Rollback has completed (skipped to"]
            }
          }
        ]
      }
    ]
  }
}
```

```

        start_vnfc)."]
            }
        ]
    },
    "request": "http://ovlm-vm:8080/api/v2/tenants/default/
vnfs/microblog/vnf_instances/dc_1/workflow_instances",
    "method": "POST"
},
"status": 200
}

```

In the next example an rollback_vnf API request is made against a VNF that was not upgraded.

```

OVLMFE_BASEURL=http://10.3.18.22:28085
tenant_id=default
vnf_id=microblog
vnf_instance_id=dc_1
echo " - rollback - "
curl -i \
-H "Content-Type: application/json" \
-X POST \
-d '{"workflow_type":"rollback_vnf", "execution_order": "sequential", "force": \
["1"]}' \
$OVLMFE_BASEURL/api/v2/tenants/${tenant_id}/vnfs/${vnf_id}/vnf_i
nstances/${vnf_instance_id}/workflow_instances?sync=1

```

The command fails because the API was not upgraded and a 500 status error is returned as shown in the example response below.

```

HTTP / 1.1 500 Server Error
Date: Mon, 20 Feb 2017 10: 50: 16 GMT
X - Application - Context: application: 28085
Content - Type: application / json;
charset = UTF - 8
Transfer - Encoding: chunked
Server: Jetty(9.2.14.v20151106)
{
"data": {
    "workflow_instance_status": "FAILED",
    "error": "{\"Result\":{\"status\":500,\"data\":{\"request\":\"http://10.3.18.22:28085/api/v2/tenants/default/vnfs/microblog/vnf_instances/dc_1/workflow_instances\", \"method\": \"POST\"}}}",
    "error_code": "EXECUTE_WORKFLOW_FAILED",
    "request": "http://10.3.18.22:28085/api/v2/tenants/default/vnfs/microblog/vnf_instances/dc_1/workflow_instances",
    "method": "POST"
},
"status": 500
}

```

Rolling Back a VNF Using the VNF-M GUI

To roll back a VNF follow these steps:

- On the VNF Lifecycle page, locate the active VNF to roll back, click the more options icon



and select **Rollback**. Scroll down the page if required.

Note: An active VNF is any VNF that is not inactive, regardless of status.

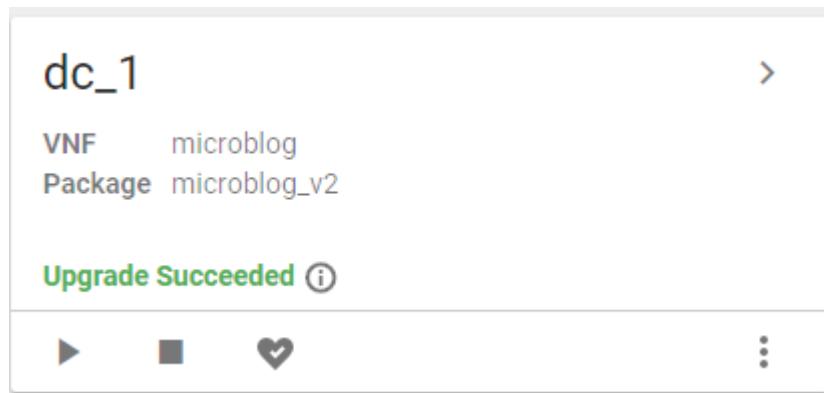


Figure 70: Active VNF

The Rollback VNF dialog box is displayed.



Figure 71: Rollback VNF Confirmation Box

Note: The VNF can be rolled back to the previous version only.

- Select an execution order by clicking the **Sequential** or **Parallel** radio button.
- To configure additional parameters for a script you are running on the Resource Agent related to rolling back the VNF, click the icon in the top-right of the dialog box and select **Add parameters**.

The dialog box displays a field into which you can enter additional parameters. In some cases there might be predefined parameters, which will display when you select **Add Parameters**. Predefined parameters can be overwritten.

Note: For information about how to predefine Additional Parameters, see [Managing Additional Parameters for the VNF-M GUI](#).

```
{
    "take_a_snapshot": true
}
```

Note: To remove additional parameters after adding them, click the top-right icon again and select **Remove parameters**.

- Click **Rollback** to roll back and close the dialog box.

The dialog box closes, the update begins and the status changes from green to blue with the text "Rollback In-Progress".

Note: When rollback completes successfully the VNF status shows as "Running". In the event that the rollback fails the VNF card is moved to the Alerts section and the status turns orange with the text "Rollback Failed".

Starting VNFs

To start a VNF instance, use the `start_vnf` workflow_type parameter.

You can start a VNF instance from the CLI, the API, or from the Openet Weaver VNF-M GUI.

The `start_vnf` workflow is shown below.

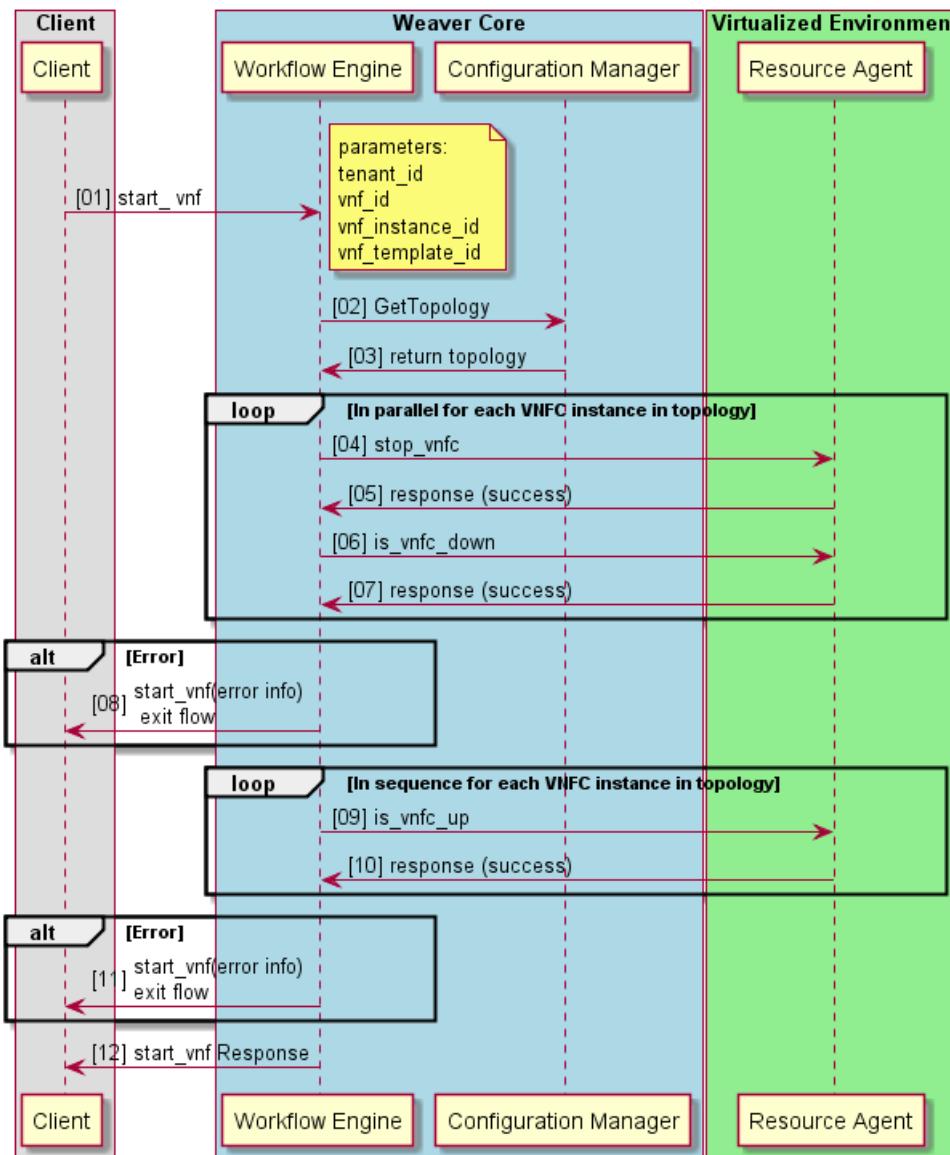


Figure 72: `start_vnf` Process Flow

Related concepts

[Setting Environment Variables](#) on page 133

[Openet Weaver VNF-M API Reference](#) on page 468

Openet Weaver REST API reference guides

[Openet Weaver VNF-M CLI Reference](#) on page 469

Openet Weaver CLI command reference guides

Starting a VNF Instance from the CLI

To start a VNF instance from the CLI, run the workflow submit command using the start_vnf workflow with the workflow submit command as shown below.

```
ovlm-fe workflow submit -t <tenant_id> -v <vnf_id> -i <vnf_instance_id> -y
start_vnf [-sync <sync>] [-timeout <seconds>] [-a '{<additional_parameter1,
additional_parameter2,...>}']
```

Note: All parameters are described on the main page for this section, [Running VNF Workflows](#).

The command in the following example starts a VNF instance with the default tenant, the microblog vnf, and a vnf_instance_id of dc_1. It also checks whether that instance is the master in an HA environment.

```
ovlm-fe workflow submit -t default -v microblog -i dc_1 -y start_vnf -a
'{"active_master": "true", "take_a_snapshot": "false"}'
```

Note: Be sure the baseurl environment variables are set before using CLI commands.

The output of the command is a workflow_instance_id generated by the system similar to the following example.

```
data:
  method: POST
  request: http://ovlm-vm:8080/api/v2/tenants/default/vnfs/microblog/vnf_instances/dc_1/workflow_instances
  workflow_instance_id: '112600278'
status: 200
```

Starting a VNF Instance Using REST API

To start a VNF instance using REST API, post a workflow request using the start_vnf workflow_type parameter. The API request takes the following format.

```
POST http://<ovlm-workflow-fe-url>
/api/v2/tenants/<tenant_id>/vnfs/<vnf_id>/vnf_instances/<vnf_instance_id>/workflow_instances?sync=1
{
  "workflow_type": "start_vnf",
  ["additional_parameters": <additional_parameters>]
}
```

Note: All parameters are described on the main page for this section, [Running VNF Workflows](#).

The following is an example of starting a VNF instance using curl. The request specifies that the workflow run in synchronous mode.

```
OVLMFE_BASEURL=http://10.3.18.22:28085
tenant_id=default
vnf_id=microblog
vnf_instance_id=dc_1
echo " - start_vnf - "
curl -i \
-H "Content-Type: application/json" \
-X POST \
-d '{"workflow_type": "start_vnf"}' \
$OVLMFE_BASEURL/api/v2/tenants/${tenant_id}/vnfs/${vnf_id}/vnf_i
```

```
instances/${vnf_instance_id}/workflow_instances?sync=1
#ovlm-fe workflow submit -t default -v microblog -i dc_1 -y start_vnf
```

The following is an example response to the API call.

```
HTTP / 1.1 200 OK
Date: Mon, 20 Feb 2017 11: 17: 34 GMT
X - Application - Context: application: 28085
Content - Type: application / json;
charset = UTF - 8
Transfer - Encoding: chunked
Server: Jetty(9.2.14.v20151106)
{
  "data": {
    "workflow_instance_status": "COMPLETED",
    "flow_result": {
      "success": "\"VNF Start passed successfully on all servers.\""
    },
    "request": "http://10.3.18.22:28085/api/v2/tenants/default/vnfs/microblog/vnf_instances/dc_1/workflow_instances",
    "method": "POST"
  },
  "status": 200
}
```

You can use -a additional_parameters to extend the functionality of the start_vnf workflow. In the following example there is a deployment of database clusters spread across data centers with a co-located Openet Weaver installation in each center. The database is set up in active/passive node. In this example there is different criteria for determining whether a VNFC is up depending on which cluster you are starting . When starting the first cluster, the cluster is allowed to start and is considered up even though there is currently no second cluster to which to replicate. When starting the second cluster, it is only considered up if it is replicating successfully to the first cluster. To support this behavior you can pass a parameter that indicates which cluster you are starting as in the following command.

```
POST http://<ovlm-workflow-fe-url>/api/v2/tenants/default/vnfs/microblog/vnf_instances/dc_1/workflow_instances
{
  "workflow_type": "start_vnf",
  "additional_parameters": "{\"active_master\": \"true\", \"take_a_snapshot\": \"false\"}"
}
```

Starting a VNF Instance Using the VNF-M GUI

To start a VNF follow these steps:

1. On the VNF Lifecycle page, locate the VNF you intend to start and click the start icon



Scroll down the page to find the VNF if necessary.

Note: The Start icon is the same as the instantiate icon, but the behavior is different depending on whether the VNF is inactive or in a down state. VNFs with a Down status are grouped in the Alerts section.

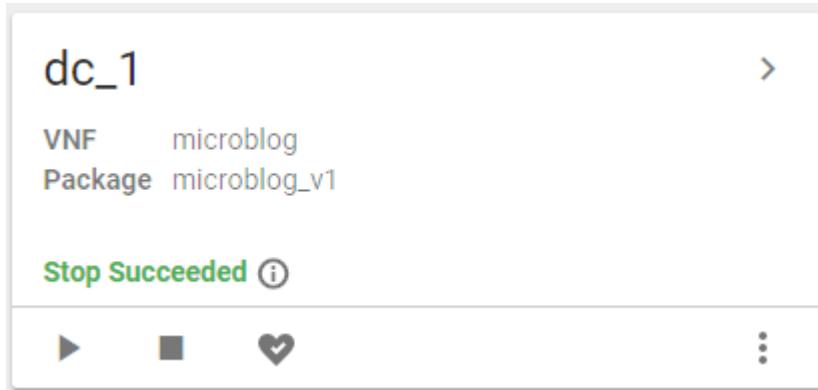


Figure 73: VNF Instance with Down Health Status

The Start VNF Instance confirmation box is displayed.



Figure 74: Start VNF Instance Confirmation Box

2. To configure additional parameters for a script you are running on the Resource Agent related to starting the VNF, click the icon in the top-right of the confirmation box and select **Add parameters**.

The dialog box displays a field into which you can enter additional parameters. In some cases there might be predefined parameters, which will display when you select **Add Parameters**. Predefined parameters can be overwritten.

Note: For information about how to predefine Additional Parameters, see [Managing Additional Parameters for the VNF-M GUI](#).

Parameters must be entered in JSON format as in the following example.

```
{
    "take_a_snapshot": true
}
```

Note: To remove additional parameters after adding them, click the top-right icon again and select **Remove parameters**.

3. Click **Start**.
The VNF instance starts and its status is displayed at the bottom of the card.

Stopping VNFs

To stop a VNF instance, use the `stop_vnf workflow_type` parameter.

You can stop a VNF instance from the CLI, the API, or from the Openet Weaver VNF-M GUI.

The stop_vnf workflow is shown below.

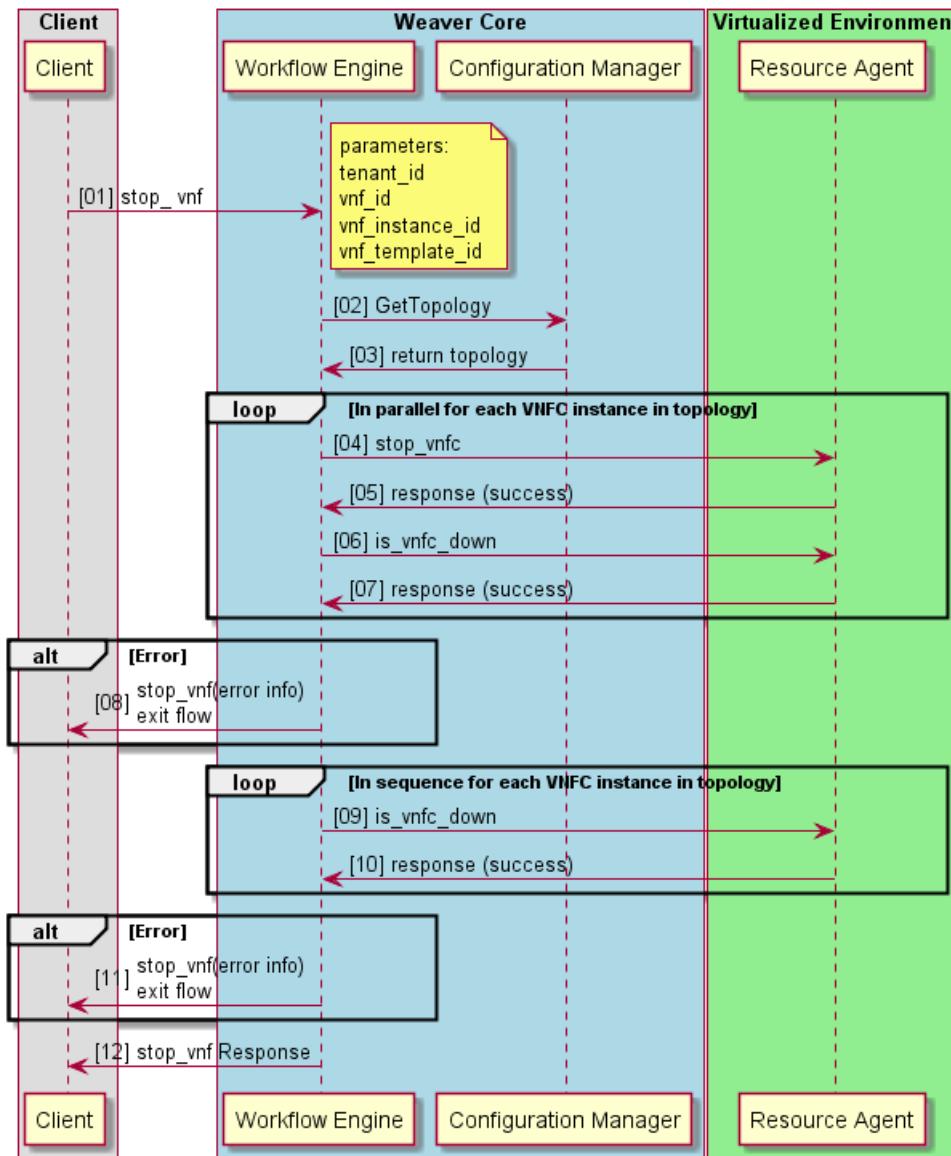


Figure 75: stop_vnf Process Flow

Related concepts

[Setting Environment Variables](#) on page 133

[Openet Weaver VNF-M API Reference](#) on page 468

Openet Weaver REST API reference guides

[Openet Weaver VNF-M CLI Reference](#) on page 469

Openet Weaver CLI command reference guides

Stopping a VNF Instance from the CLI

You can stop a VNF from the CLI by running the workflow submit command with the stop_vnf workflow as shown below.

```
ovlm-fe workflow submit -t <tenant_id> -v <vnf_id> -i <vnf_instance_id> -y
stop_vnf [-sync <sync>] [-timeout <seconds>] [-a '{<additional_parameter1,
additional_parameter2,...>}']
```

Note: All parameters are described on the main page for this section, [Running VNF Workflows](#).

The command in the following example stops a VNF instance with the default tenant, the microblog vnf, and a vnf_instance_id of dc_1.

```
ovlm-fe workflow submit -t default -v microblog -i dc_1 -y stop_vnf
```

Note: Be sure the baseurl environment variables are set before using CLI commands. There is also an API version of this command.

The output of the command is a workflow_instance_id generated by the system similar to the following example.

```
data:
  method: POST
  request: http://ovlm-vm:8080/api/v2/tenants/default/vnfs/microblog/vnf_instances/dc_1/workflow_instances
  workflow_instance_id: '112600351'
  status: 200
```

Stopping a VNF Instance Using REST API

To stop a VNF instance using REST API, post a workflow request using the stop_vnf workflow_type parameter. The API request takes the following format.

```
POST http://<ovlm-workflow-fe-url>
/api/v2/tenants/<tenant_id>/vnfs/<vnf_id>/vnf_instances/<vnf_instance_id>/workflow_instances?sync=1
{
  "workflow_type": "stop_vnf",
  ["additional_parameters": <additional_parameters>]
}
```

Note: All parameters are described on the main page for this section, [Running VNF Workflows](#).

The following is an example of stopping a VNF instance using curl. The request specifies that the workflow run in synchronous mode.

```
OOVLMFE_BASEURL=http://10.3.18.22:28085
tenant_id=default
vnf_id=microblog
vnf_instance_id=dc_1
echo " - stop - "
curl -i \
-H "Content-Type: application/json" \
-X POST \
-d '{"workflow_type": "stop_vnf"}' \
$OOVLMFE_BASEURL/api/v2/tenants/${tenant_id}/vnfs/${vnf_id}/vnf_instances/${vnf_instance_id}/workflow_instances?sync=1
#ovlm-fe workflow submit -t default -v microblog -i dc_1 -y stop_vnf
```

The following is an example response to the API call.

```

HTTP / 1.1 200 OK
Date: Mon, 20 Feb 2017 11: 30: 56 GMT
X - Application - Context: application: 28085
Content - Type: application / json;
charset = UTF - 8
Transfer - Encoding: chunked
Server: Jetty(9.2.14.v20151106)
{
  "data": {
    "workflow_instance_status": "COMPLETED",
    "flow_result": {
      "success": "\"VNF Stop passed successfully on all servers.\""
    },
    "request": "http://10.3.18.22:28085/api/v2/tenants/default/vnfs/microblog/vnf_instances/dc_1/workflow_instances",
    "method": "POST"
  },
  "status": 200
}

```

Stopping a VNF Instance Using the VNF-M GUI

To stop a VNF follow these steps:

1. On the VNF Lifecycle page, locate the active VNF you intend to stop and click the stop icon



Scroll down the page to find the VNF if necessary.

Note: Active VNFs are grouped at the bottom of the VNF Lifecycle page.

dc_1 >

VNF microblog
Package microblog_v1

Instantiate Succeeded ⓘ

▶ ■ ❤ ⋮

Figure 76: Active VNF

The Stop VNF Instance confirmation box is displayed.



Figure 77: Stop VNF Instance Confirmation Box

2. To configure additional parameters for a script you are running on the Resource Agent related to Stopping the VNF, click the icon in the top-right of the dialog box and select **Add parameters**.

The dialog box displays a field into which you can enter additional parameters. In some cases there might be predefined parameters, which will display when you select **Add Parameters**. Predefined parameters can be overwritten.

Note: For information about how to predefine Additional Parameters, see *Managing Additional Parameters for the VNF-M GUI*.

Parameters must be entered in JSON format as in the following example.

```
{
    "take_a_snapshot": true
}
```

Note: To remove additional parameters after adding them, click the top-right icon again and select **Remove parameters**.

3. Click **Stop**.

The VNF instance is stopped with a status of Stop Succeeded.

Checking That a VNF is Up

To check whether a VNF is up use the `is_vnf_up` workflow_type parameter.

In addition to VNF status, this workflow returns the health status of each VNFC instance. The table below contains a description of each VNF health status and a description

VNF Health Status	Description
UP	All VNFC instances are running with an UP health status.
DOWN	At least one of the VNFC instances has a DOWN health status, and none of the VNFC instances has an UNKNOWN health status.
UNKNOWN	At least one of the VNFC instances has an UNKNOWN health status. There might also be one or more DOWN status VNFCs.

As part of the workflow, each VNFC is checked and returns one of the VNFC health status responses listed in the following table.

VNFC Health Status	Description	Exit Code	OVLM Python API Constants
UP	The VNFC is running successfully.	0	HealthStatus.UP
DOWN	The VNFC is not running.	100	HealthStatus.DOWN
UNKNOWN	The VNFC status cannot be determined. For example, there might be a: <ul style="list-style-type: none">• Script syntax error• Script execution error• Script timeout	Any exit code value other than 0 and 100	Not defined

You can check whether a VNF instance is up from the CLI, the API, or from the Openet Weaver VNF-M GUI.

The `is_vnf_up` workflow is shown below.

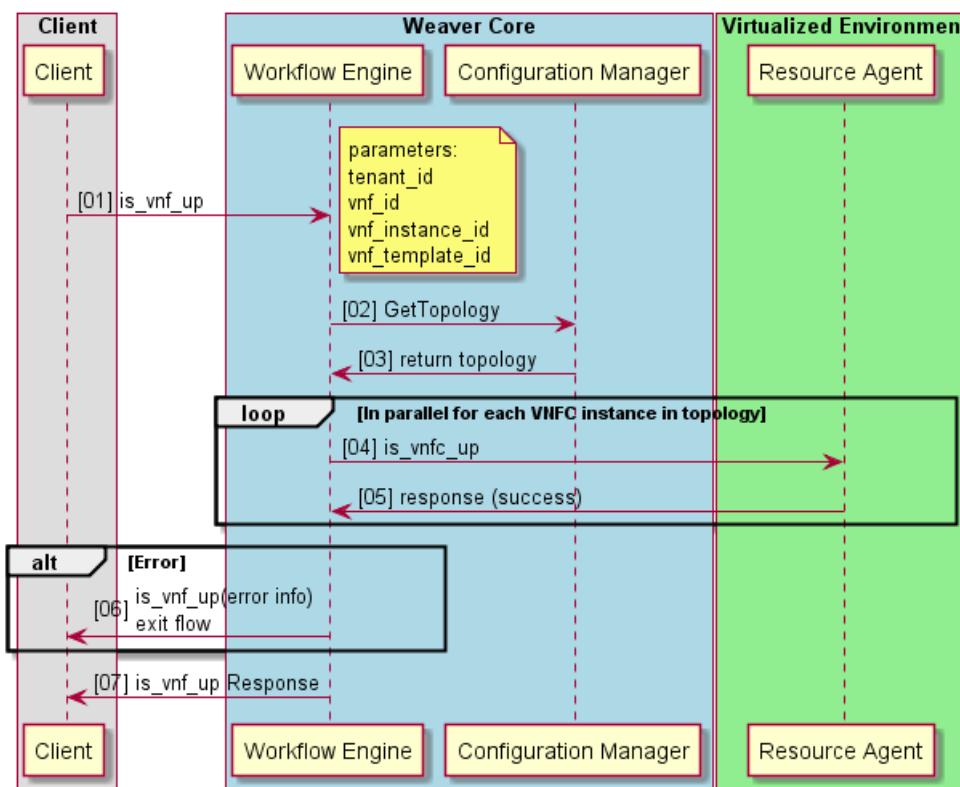


Figure 78: `is_vnf_up` Process Flow

Related concepts

[Setting Environment Variables](#) on page 133

[Openet Weaver VNF-M API Reference](#) on page 468

Openet Weaver REST API reference guides

[Openet Weaver VNF-M CLI Reference](#) on page 469

Openet Weaver CLI command reference guides

Checking Whether a VNF is Up from the CLI

To check whether a VNF is up and the health status of each VNFC instance from the CLI, use the `is_vnf_up` workflow_type parameter with the workflow submit command as shown below.

```
ovlm-fe workflow submit -t <tenant_id> -v <vnf_id> -i <vnf_instance_id> -y
is_vnf_up [-sync <sync>] [-timeout <seconds>] [-a <additional_parameter1,
additional_parameter2,...>]
```

Note: All parameters are described on the main page for this section, [Running VNF Workflows](#). For a description of each health status see [Checking That a VNF is Up](#).

The CLI command in the following example checks whether a specified VNF is up. In this example the VNF parameters use the default tenant, the microblog vnf, and a vnf_instance_id of dc_1.

```
ovlm-fe workflow submit -t default -v microblog -i dc_1 -y is_vnf_up
```

Note: Be sure the baseurl environment variables are set before using CLI commands. There is also an API version of this command.

The following is an example of a successful response when all VNFCs have an UP health status.

```
data:
  workflow_instance_status: COMPLETED
  flow_result:
    health_status: UP
    vnfc_instances:
      - vnfc_id: blogserver
        vnfc_instance_id: '1'
        hostname: 10.3.12.32
        exit_code: '0'
        response:
          health_status: UP
      - vnfc_id: blogserver
        vnfc_instance_id: '2'
        hostname: 10.3.12.144
        exit_code: '0'
        response:
          health_status: UP
      - vnfc_id: nginx
        vnfc_instance_id: '1'
        hostname: 10.3.12.85
        exit_code: '0'
        response:
          health_status: UP
      - vnfc_id: postgres
        vnfc_instance_id: '1'
        hostname: 10.3.11.44
        exit_code: '0'
        response:
          health_status: UP
  request: http://ovml:28085/api/v2/tenants/default/vnfs/microblog/vnf_instances/dc_1/workflow_instances
  method: POST
  status: 200
```

The following is an example of a successful response when several of the VNFC instances have a DOWN health status and none of the VNFCs has an UNKNOWN health status.

```
data:
  workflow_instance_status: COMPLETED
  flow_result:
```

```

health_status: DOWN
vnfc_instances:
- vnfc_id: blogserver
  vnfc_instance_id: '1'
  hostname: 10.3.12.32
  exit_code: '100'
  response:
    health_status: DOWN
    error_msg:
      - 'ERROR The following service(s) are inactive: blogserver'
- vnfc_id: blogserver
  vnfc_instance_id: '2'
  hostname: 10.3.12.144
  exit_code: '100'
  response:
    health_status: DOWN
    error_msg:
      - 'ERROR The following service(s) are inactive: blogserver'
- vnfc_id: nginx
  vnfc_instance_id: '1'
  hostname: 10.3.10.153
  exit_code: '0'
  response:
    health_status: UP
- vnfc_id: postgres
  vnfc_instance_id: '1'
  hostname: 10.3.11.44
  exit_code: '0'
  response:
    health_status: UP
request: http://ovm1:28085/api/v2/tenants/default/vnfs/microblog/vnf_instances/dc_1/workflow_instances
method: POST
status: 200

```

The following is an example of a successful response when several of the VNFC instances have an UNKNOWN health status and one of the VNFCs has a DOWN health status.

```

data:
  workflow_instance_status: COMPLETED
  flow_result:
    health_status: UNKNOWN
    vnfc_instances:
- vnfc_id: blogserver
  vnfc_instance_id: '1'
  hostname: 10.3.12.32
  exit_code: '0'
  response:
    health_status: UP
- vnfc_id: blogserver
  vnfc_instance_id: '2'
  hostname: 10.3.12.144
  exit_code: '1'
  response:
    health_status: UNKNOWN
    error_msg:
      - 'File "./is_vnfc_up_v2.py", line 46'
      - ' '
      - '^'
      - 'SyntaxError: invalid syntax'
- vnfc_id: nginx
  vnfc_instance_id: '1'

```

```

hostname: 10.3.10.153
exit_code: ''
response:
  health_status: UNKNOWN
  error_msg:
    - 'I/O error on PUT request for "http://10.3.10.153:28086/api/v1/tenants/default/vnfs/microblog/vnf_instances/dc_1/vnfc_instances/1/action": No route to host; nested exception is java.net.NoRouteToHostException: No route to host'
    - vnfc_id: postgres
      vnfc_instance_id: '1'
      hostname: 10.3.11.44
      exit_code: '100'
      response:
        health_status: DOWN
        error_msg:
          - 'ERROR The following service(s) are inactive: postgres'
request: http://ovml:28085/api/v2/tenants/default/vnfs/microblog/vnf_instances/dc_1/workflow_instances
method: GET
status: 200

```

Checking Whether a VNF is Up Using Rest API

To check whether a VNF is up and the health status of each VNFC instance in the VNF using REST API, post a workflow request using the `is_vnf_up` workflow_type parameter. The API request takes the following format.

```

POST http://<ovlm-workflow-fe-url>/api/v2/tenants/<tenant_id>/vnfs/<vnf_id>/vnf_instances/<vnf_instance_id>/workflow_instances?sync=1&timeout=2000
{
  "workflow_type": "is_vnf_up",
  ["additional_parameters": <additional_parameters>]
}

```

Note: All parameters are described on the main page for this section, [Running VNF Workflows](#). For a description of each health status see [Checking That a VNF is Up](#).

The following is an example of checking whether a VNF instance is up using curl. The request specifies that the workflow run in synchronous mode.

```

OVLMFE_BASEURL=http://10.3.18.22:28085
tenant_id=default
vnf_id=microblog
vnf_instance_id=dc_1
echo " - is_vnf_up - "
curl -i \
-H "Content-Type: application/json" \
-X POST \
-d '{"workflow_type": "is_vnf_up"}' \
$OVLMFE_BASEURL/api/v2/tenants/${tenant_id}/vnfs/${vnf_id}/vnf_instances/${vnf_instance_id}/workflow_instances?sync=1
#ovlm-fe workflow submit -t default -v microblog -i dc_1 -y is_vnf_up

```

The following is an example of a successful response with all VNFCs having an UP health_status.

```
{
  "data": {
    "workflow_instance_status": "COMPLETED",
    "flow_result": {
      "health_status": "UP",
      "vnfc_instances": [

```

```
{
  "vnfc_id": "blogserver",
  "vnfc_instance_id": "1",
  "hostname": "10.3.12.32",
  "exit_code": "0",
  "response": {
    "health_status": "UP"
  }
},
{
  "vnfc_id": "blogserver",
  "vnfc_instance_id": "2",
  "hostname": "10.3.12.144",
  "exit_code": "0",
  "response": {
    "health_status": "UP"
  }
},
{
  "vnfc_id": "nginx",
  "vnfc_instance_id": "1",
  "hostname": "10.3.12.85",
  "exit_code": "0",
  "response": {
    "health_status": "UP"
  }
},
{
  "vnfc_id": "postgres",
  "vnfc_instance_id": "1",
  "hostname": "10.3.11.44",
  "exit_code": "0",
  "response": {
    "health_status": "UP"
  }
}
]
},
  "request": "http://10.3.12.24:28085/api/v2/tenants/default/vnfs/microblog/vnf_instances/dc_1/workflow_instances",
  "method": "POST"
},
  "status": 200
}
```

The following is an example of a successful response with two of the VNFCs having a DOWN health_status and none of the VNFCs having an UNKNOWN health status.

```
{
  "data": {
    "workflow_instance_status": "COMPLETED",
    "flow_result": {
      "health_status": "DOWN",
      "vnfc_instances": [
        {
          "vnfc_id": "blogserver",
          "vnfc_instance_id": "1",
          "hostname": "10.3.12.32",
          "exit_code": "100",
          "response": {
            "health_status": "DOWN",
            "error_msg": [

```

```

        "ERROR The following service(s) are inactive: blogserver"
    ]
}
{
    "vnfc_id": "blogserver",
    "vnfc_instance_id": "2",
    "hostname": "10.3.12.144",
    "exit_code": "100",
    "response": {
        "health_status": "DOWN",
        "error_msg": [
            "ERROR The following service(s) are inactive: blogserver"
        ]
    }
},
{
    "vnfc_id": "nginx",
    "vnfc_instance_id": "1",
    "hostname": "10.3.10.153",
    "exit_code": "0",
    "response": {
        "health_status": "UP"
    }
},
{
    "vnfc_id": "postgres",
    "vnfc_instance_id": "1",
    "hostname": "10.3.11.44",
    "exit_code": "0",
    "response": {
        "health_status": "UP"
    }
}
]
},
"request": "http://10.3.12.24:28085/api/v2/tenants/default/vnfs/microblog/vnf_instances/dc_1/workflow_instances",
"method": "GET"
},
"status": 200
}

```

The following is an example of a successful response with two of the VNFCs having an UNKNOWN health_status and one of the VNFCs having a DOWN health status.

```
{
  "data": {
    "workflow_instance_status": "COMPLETED",
    "flow_result": {
      "health_status": "UNKNOWN",
      "vnfc_instances": [
        {
          "vnfc_id": "blogserver",
          "vnfc_instance_id": "1",
          "hostname": "10.3.12.32",
          "exit_code": "0",
          "response": {
            "health_status": "UP"
          }
        },
        {

```

```

    "vnfc_id": "blogserver",
    "vnfc_instance_id": "2",
    "hostname": "10.3.12.144",
    "exit_code": "1",
    "response": {
        "health_status": "UNKNOWN",
        "error_msg": [
            "  File './is_vnfc_up_v2.py', line 46",
            "  ^",
            "  ^",
            "SyntaxError: invalid syntax"
        ]
    }
},
{
    "vnfc_id": "nginx",
    "vnfc_instance_id": "1",
    "hostname": "10.3.10.153",
    "exit_code": "",
    "response": {
        "health_status": "UNKNOWN",
        "error_msg": [
            "I/O error on PUT request for \"http://10.3.10.153:28086/api/v1/tenants/default/vnfs/microblog/vnf_instances/dc_1/vnfc_instances/nginx/vnfc_instances/1/action\": No route to host; nested_exception is java.net.NoRouteToHostException: No route to host"
        ]
    }
},
{
    "vnfc_id": "postgres",
    "vnfc_instance_id": "1",
    "hostname": "10.3.11.44",
    "exit_code": "100",
    "response": {
        "health_status": "DOWN",
        "error_msg": [
            "ERROR The following service(s) are inactive: postgres"
        ]
    }
},
{
    "request": "http://10.3.12.24:28085/api/v2/tenants/default/vnfs/microblog/vnf_instances/dc_1/workflow_instances",
    "method": "GET"
},
{
    "status": 200
}

```

Checking Whether a VNF is Up Using the VNF-M GUI

To check whether a VNF is up and the health status of each VNFC instance, follow these steps:

1. On the VNF Lifecycle page, locate the active VNF to check and click the Health Check icon



Scroll down the page to find the VNF if necessary.

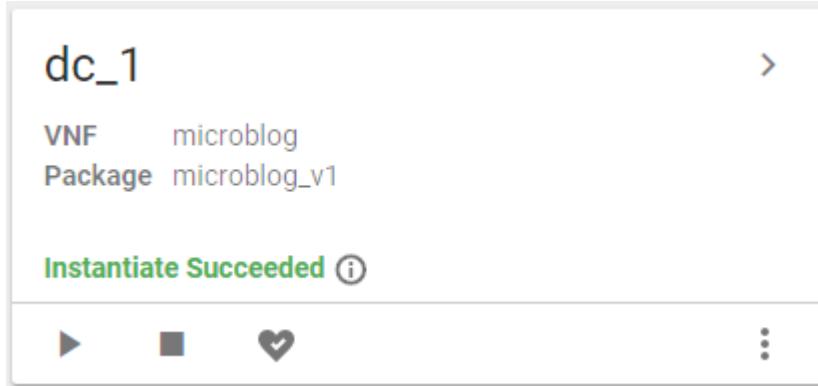


Figure 79: Active VNF

The VNF Instance Health Status message box is displayed. It displays details about each VNFC in the VNF instance, including the health status, as shown in the example below.

Note: For a description of each health status see [Checking That a VNF is Up](#).



Figure 80: VNF Instance Health Status Message Box

2. Click **Close** to close the message box.

The message box closes and the VNF card status is changed to "Health Check: Running".

Terminating a VNF

Terminating a VNF stops all VNFCs in a VNF instance, uninstalls the VNF software and removes all active and inactive packages. A termination workflow can be run on VNFs that are already instantiated.

When terminating from the CLI or using the API, there are two workflows:

- `terminate_vnf_without_vim`

- terminate_vnf (with VIM)

The VNF-M GUI has only one terminate function because it can determine whether the target VNF is configured for VIM.

[Terminating a VNF Without VIM](#)

To terminate a VNF that was configured without VIM, use the terminate_vnf_without_vim workflow_type parameter.

You can terminate a VNF instance from the CLI, the API, or from the Openet Weaver VNF-M GUI. The VNF-M GUI has one terminate function that can be used on VNFs with or without VIM as described in [Terminating a VNF Using the VNF-M GUI](#).

The terminate_vnf workflow without VIM is shown below.

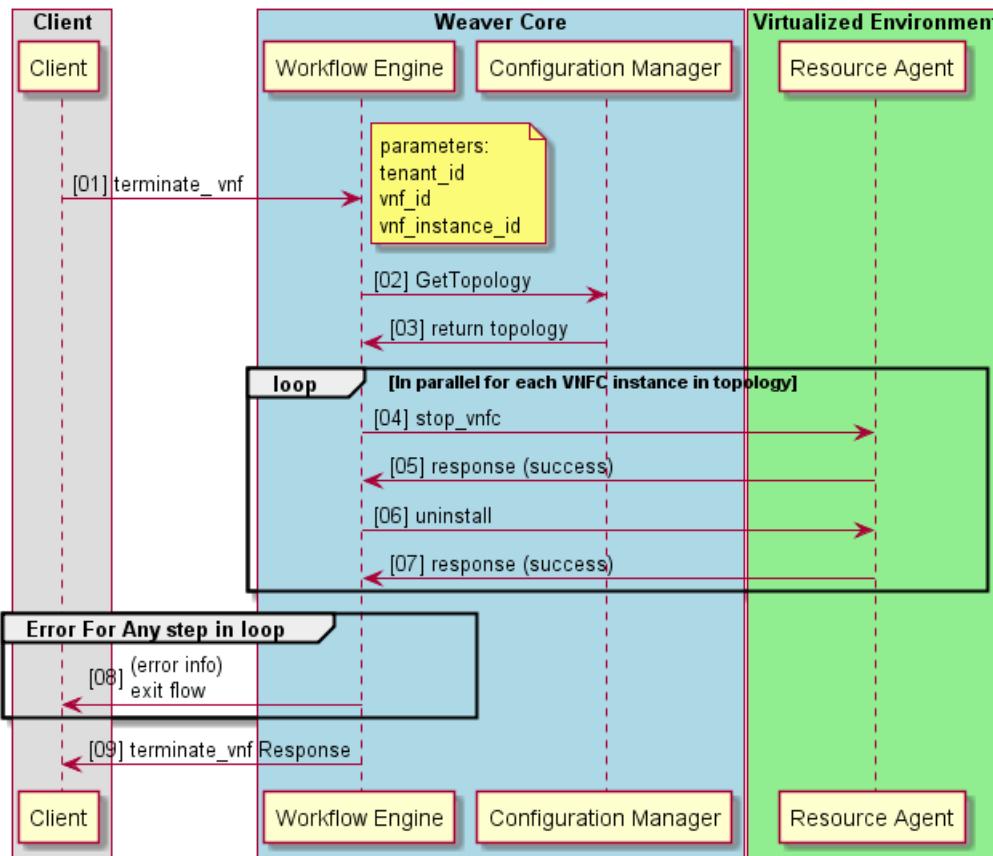


Figure 81: terminate_vnf_without_vim Process Flow

Related concepts

[Setting Environment Variables](#) on page 133

[Openet Weaver VNF-M API Reference](#) on page 468

Openet Weaver REST API reference guides

[Openet Weaver VNF-M CLI Reference](#) on page 469

Openet Weaver CLI command reference guides

[Terminating a VNF Without VIM from the CLI](#)

You can terminate a VNF from the CLI using the workflow submit command with the terminate_vnf_without_vim workflow as shown below.

```
ovlm-fe workflow submit -t <tenant_id> -v <vnf_id> -i <vnf_instance_id> -y
terminate_vnf_without_vim [-sync <sync>] [-timeout <seconds>] [-force]
```

```
<force_value>
[-a '<additional_parameter1, additional_parameter2,...>']
```

Note: All parameters are described on the main page for this section, [Running VNF Workflows](#).

The CLI command in the following example terminates a specified VNF after stopping all VNFC instances on the Resource Agents. In this example the command parameters are the default tenant, the microblog vnf, and a vnf_instance_id of dc_1. The force flag has a value of 1, which specifies that the clean up procedure for the VNFC s is triggered even if there is an error in uninstalling VNFCs.

```
ovlm-fe workflow submit -t default -v microblog -i dc_1 -y
terminate_vnf_without_vim -force 1
```

Note: Be sure the baseurl environment variables are set before using CLI commands. There is also an API version of this command.

The output of the command is a workflow_instance_id generated by the system similar to the following example.

```
data:
  method: POST
  request: http://ovlm-vm:8080/api/v2/tenants/default/vnfs/microblog/vnf_instances/dc_1/workflow_instances
  workflow_instance_id: '112600123'
status: 201
```

Terminating a VNF Without VIM Using REST API

To terminate a VNF that is not configured for VIM using REST API, post a workflow request using the terminate_vnf_without_vim workflow_type parameter. The API request takes the following format.

```
POST http://<ovlm-workflow-fe-url>
/api/v2/tenants/<tenant_id>/vnfs/<vnf_id>/vnf_instances/<vnf_instance_id>/workflow_instances?sync=1
{
  "workflow_type": "terminate_vnf_without_vim",
  ["force": [<force_value>], ],
  ["additional_parameters": <additional_parameters>]
}
```

Note: All parameters are described on the main page for this section, [Running VNF Workflows](#).

The following is an example of terminating a VNF without VIM using curl. The request specifies that the workflow run in synchronous mode. Additionally, the force flag has a value of 1, which specifies that the clean up procedure for the VNFC s is triggered even if there is an error in uninstalling VNFCs.

```
OVLMFE_BASEURL=http://10.3.18.22:28085
tenant_id=default
vnf_id=microblog
vnf_instance_id=dc_1
echo " - terminate - "
curl -i \
-H "Content-Type: application/json" \
-X POST \
-d '{"workflow_type": "terminate_vnf_without_vim"}' \
$OVLMFE_BASEURL/api/v2/tenants/${tenant_id}/vnfs/${vnf_id}/vnf_instances/${vnf_instance_id}/workflow_instances?sync=1
#ovlm-fe workflow submit -t default -v microblog -i dc_1 -y
terminate_vnf_without_vim -force 1
```

The following is an example response to the API call.

```

HTTP / 1.1 200 OK
Date: Mon, 20 Feb 2017 11: 42: 09 GMT
X - Application - Context: application: 28085
Content - Type: application / json;
charset = UTF - 8
Transfer - Encoding: chunked
Server: Jetty(9.2.14.v20151106)
{
  "data": {
    "workflow_instance_status": "COMPLETED",
    "flow_result": {
      "success": "\"VNF Termination passed successfully on all servers.\\""
    },
    "request": "http://10.3.18.22:28085/api/v2/tenants/default/vnfs/microblog/vnf_instances/dc_1/workflow_instances",
    "method": "POST"
  },
  "status": 200
}

```

Terminating a VNF Using the VNF-M GUI

To terminate a VNF follow these steps:

1. On the VNF Lifecycle page, locate the active VNF you intend to terminate, click the more options icon



and select **Terminate**. Scroll down the page if required.

Note: An active VNF is any VNF that is not inactive, regardless of status.

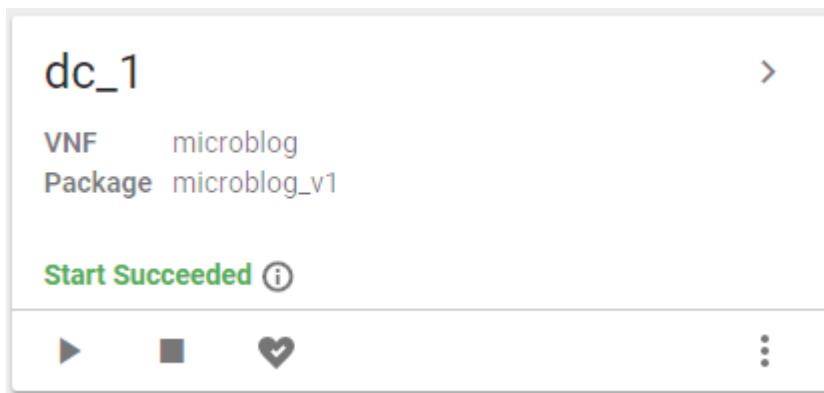


Figure 82: Inactive VNF

The Terminate VNF Instance confirmation box is displayed.



Figure 83: Terminate VNF Instance Confirmation Box

2. To configure additional parameters for a script you are running on the Resource Agent related to terminating the VNF, click the icon in the top-right of the confirmation box and select **Add parameters**.

The dialog box displays a field into which you can enter additional parameters. In some cases there might be predefined parameters, which will display when you select **Add Parameters**. Predefined parameters can be overwritten.

Note: For information about how to predefine Additional Parameters, see [Managing Additional Parameters for the VNF-M GUI](#).

Parameters must be entered in JSON format as in the following example.

```
{
    "take_a_snapshot": true
}
```

Note: To remove additional parameters after adding them, click the top-right icon again and select **Remove parameters**.

3. Click **Terminate**.

The dialog box closes, the VNF is terminated, and the VNF card status is changed to "Terminated" in black text among the inactive VNF cards.

Note: In the event that terminating the VNF fails, the VNF card is moved to the Alerts section with the status "Terminate Failed" in orange text.

Terminating a VNF with VIM

To terminate a VNF that was configured for VIM, use the terminate_vnf workflow_type parameter.

You can terminate a VNF instance from the CLI, the API, or from the Openet Weaver VNF-M GUI. The VNF-M GUI has one terminate function that can be used on VNFs with or without VIM as described in [Terminating a VNF Using the VNF-M GUI](#).

The terminate_vnf workflow is shown below.

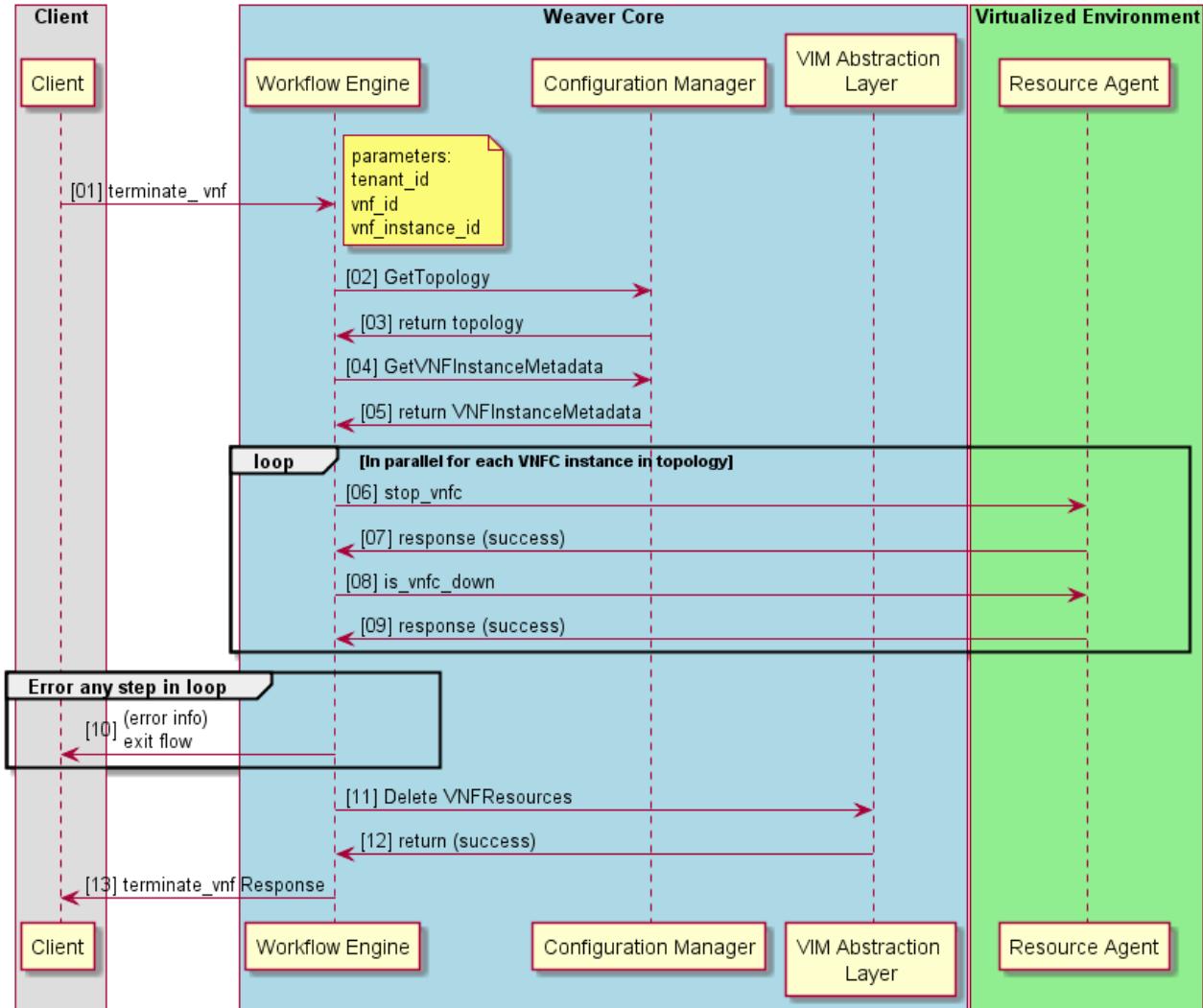


Figure 84: terminate_vnf Process Flow

Related concepts

[Setting Environment Variables](#) on page 133

[Openet Weaver VNF-M API Reference](#) on page 468
Openet Weaver REST API reference guides

[Openet Weaver VNF-M CLI Reference](#) on page 469
Openet Weaver CLI command reference guides

Terminating a VNF With VIM from the CLI

You can terminate a VNF with VIM configuration from the CLI using the workflow submit command with the terminate_vnf workflow as shown below.

```
ovlm-fe workflow submit -t <tenant_id> -v <vnf_id> -i <vnf_instance_id> -y
    terminate_vnf [-sync <sync>] [-timeout <seconds>]
    [-a '{<additional_parameter1, additional_parameter2,...>}']
```

Note: All parameters are described on the main page for this section, [Running VNF Workflows](#).

You can terminate a VNF with VIM configuration from the CLI using the workflow submit command with the terminate_vnf workflow. The command in the following example terminates a specified VNF after stopping all VNFC

instances on the Resource Agents. In this example the command parameters are the default tenant, the microblog vnf, and a vnf_instance_id of dc_1.

```
ovlm-fe workflow submit -t default -v microblog -i dc_1 -y terminate_vnf
```

Note: Be sure the baseurl environment variables are set before using CLI commands. There is also an API version of this command.

The output of the command is a workflow_instance_id generated by the system similar to the following example.

```
data:
  method: POST
  request: http://ovlm-vm:8080/api/v2/tenants/default/vnfs/microblog/vnf_instances/dc_1/workflow_instances
  workflow_instance_id: '112600123'
status: 201
```

Terminating a VNF with VIM Using REST API

To terminate a VNF that is configured for VIM using REST API, post a workflow request using the terminate_vnf workflow_type parameter. The API request takes the following format.

```
POST http://<ovlm-workflow-fe-url>
/api/v2/tenants/<tenant_id>/vnfs/<vnf_id>/vnf_instances/<vnf_instance_id>/workflow_instances?sync=1
{
  "workflow_type": "terminate_vnf",
  ["additional_parameters": <additional_parameters>]
}
```

Note: All parameters are described on the main page for this section, [Running VNF Workflows](#).

The following is an example of terminating a VNF with VIM using curl. The request specifies that the workflow run in synchronous mode. Additionally, the force parameter has a value of 1, which means that the VNF is terminated even if there is a failure when stopping or uninstalling the associated VNFCs.

```
OVLMFE_BASEURL=http://10.3.18.22:28085
tenant_id=default
vnf_id=microblog
vnf_instance_id=dc_2
echo " - terminate with vim- "
curl -i \
-H "Content-Type: application/json" \
-X POST \
-d '{"workflow_type": "terminate_vnf"}' \
$OVLMFE_BASEURL/api/v2/tenants/${tenant_id}/vnfs/${vnf_id}/vnf_instances/${vnf_instance_id}/workflow_instances?sync=1
```

The following is an example response to the API call.

```
HTTP / 1.1 200 OK
Date: Tue, 21 Feb 2017 04: 17: 20 GMT
X - Application - Context: application: 28085
Content - Type: application / json;
charset = UTF - 8
Transfer - Encoding: chunked
Server: Jetty(9.2.14.v20151106)
{
  "data": {
    "workflow_instance_status": "COMPLETED",
    "flow_result": {
```

```

        "success": "\"VNF Termination passed successfully on all servers.\""
    },
    "request": "http://10.3.18.22:28085/api/v2/tenants/default/vnfs/microblog/vnf_instances/dc_2/workflow_instances",
    "method": "POST"
},
"status": 200
}

```

Terminating a VNF With VIM Using the VNF-M GUI

To terminate a VNF with VIM, follow these steps:

1. On the VNF Lifecycle page, locate the active VNF you intend to terminate, click the more options icon



and select **Terminate**. Scroll down the page if required.

Note: An active VNF is any VNF that is not inactive, regardless of status.

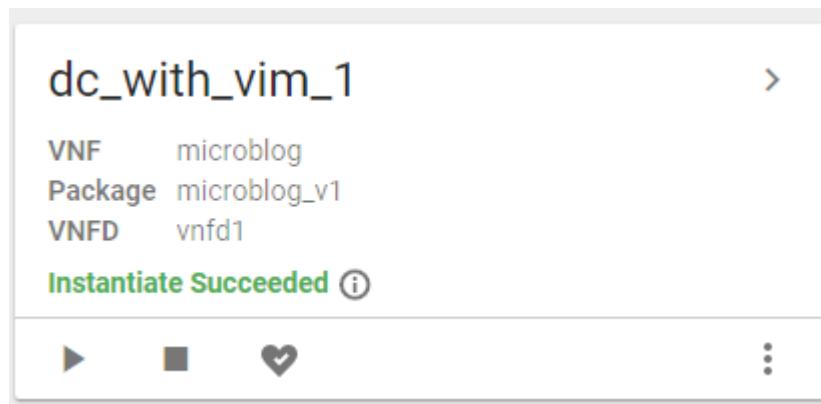


Figure 85: Active VNF

The Terminate VNF Instance confirmation box is displayed.

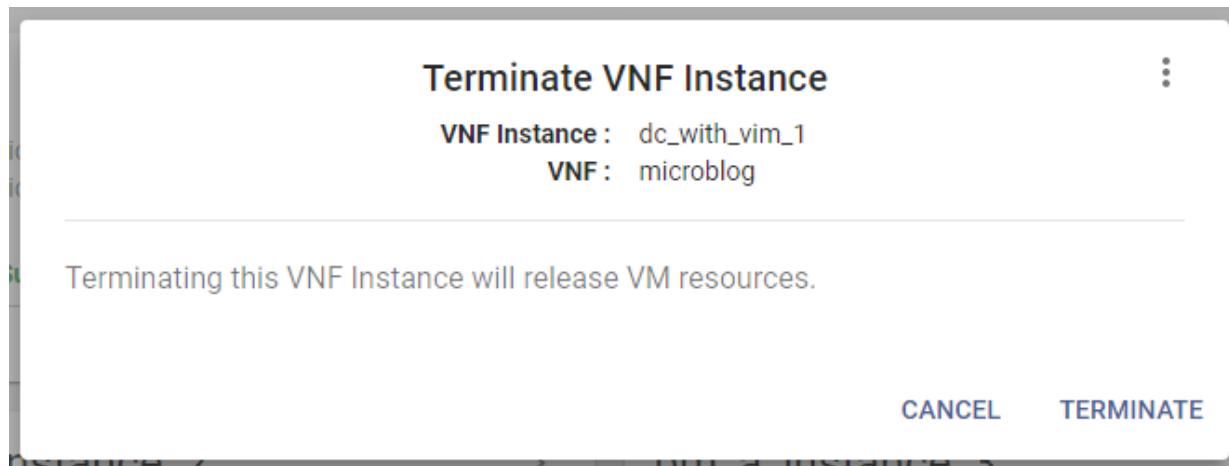


Figure 86: Terminate VNF Instance Confirmation Box

2. To configure additional parameters for a script you are running on the Resource Agent related to terminating the VNF, click the icon in the top-right of the confirmation box and select **Add parameters**.

The dialog box displays a field into which you can enter additional parameters. In some cases there might be predefined parameters, which will display when you select **Add Parameters**. Predefined parameters can be overwritten.

Note: For information about how to predefine Additional Parameters, see [Managing Additional Parameters for the VNF-M GUI](#).

Parameters must be entered in JSON format as in the following example.

```
{
    "take_a_snapshot": true
}
```

Note: To remove additional parameters after adding them, click the top-right icon again and select **Remove parameters**.

3. Click **Terminate**.

The dialog box closes, the VNF is terminated, and the VNF card status is changed to "Terminated" in black text among the inactive VNF cards.

Note: In the event that terminating the VNF fails, the VNF card is moved to the Alerts section with the status "Terminate Failed" in orange text.

Running VNFC Workflows

Openet Weaver comes with predefined VNFC-level workflows that can be used for out-of-the-box VNFC management. Each flow represents a process or group of processes to be run by the VNF manager. Workflows can be initiated by a CLI command at a user interface connected to the Workflow Engine, by an API call, for example through a third-party orchestrator, or by using the VNF-M GUI.

You can use the `vnfc_workflow submit` command to run any defined VNFC workflow.

To run `vnfc_workflow submit` from the Front End CLI enter the following command.

```
ovlm-fe vnfc_workflow submit -t <tenant_id> -v <vnf_id> -i <vnf_instance_id>
-y <workflow_type> -p <vnfd_template_id>
-c <vnfc_id> [-n <vnf_instance_id>] [-restart <true | false>] [-hostname
<host_name>] [-sync <sync>] [-timeout <seconds>] [-o <execution_order>] [-force
<force_value>] [-start <true | false>] [-a <additional_parameter1,
additional_parameter2,...>]
```

Note: Be sure the baseurl environment variables are set before using CLI commands.

To run workflow submit through the Front End API use the following.

```
curl -X POST -H 'Content-Type: application/json' -d
'{"workflow_type":"<workflow_type>","vnfd":{"vnfd_template_id": \
"<vnfd_package>"}}'
http://<your_api_host>/api/v2/tenants/<tenant_id>/vnfs/<vnf_id>/vnf_instances/<vnf_instance_id>/\
workflow_instances? [sync=<sync>]
```

Note: The subpages to this topic contain CLI and API command examples for specific workflows.

```
curl -X POST -H 'Content-Type: application/json' -d '{"workflow_type":"instantiate_vnf","vnfd":{"vnfd_template_id": \
"microblog_v1", "vnfd_id": "vnfd_production"}}' http://host1/api/v2/tenants/default/vnfs/microblog/vnf_instances/dc_1/workflow_instances
```

The workflow submit parameters are described in the following table. They apply to the CLI and the API versions of the command.

Table 40: vnfc_workflow submit Parameters

CLI Flag	Parameter	Mand.	Description
-t	tenant_id	Yes	The tenant identifier as created in the Configuration Manager.
-v	vnf_id	Yes	The VNF identifier as created in the Configuration Manager.
-i	vnf_instance_id	Yes	The VNF instance identifier as created in the Configuration Manager.
-y	workflow_type	Yes	<p>The Workflow type for the lifecycle event defined in the workflow. This can be any workflow defined for Openet Weaver. The values for the predefined workflows that come with Openet Weaver are as follows:</p> <ul style="list-style-type: none"> • instantiate_vnfc_without_vim • upgrade_vnfc • update_vnfc • rollback_vnfc • heal_vnfc • start_vnfc • stop_vnfc • custom_action_vnfc <p>Predefined workflows are described in the topics linked at the bottom of this page.</p>
-p	vnf_template_id	Yes for upgrade_vnfc and update_vnfc, otherwise no	<p>The unique identifier for the VNF package. Mandatory for the following workflows only:</p> <ul style="list-style-type: none"> • upgrade_vnfc • update_vnfc <p>Note: When this parameter is used with instantiate_vnfc_without_vim, returns the specified template. When the parameter is omitted from the workflow submit command for this workflow, a template is used based on information in IIM.</p>
-c	vnfc_id	Yes for VNFC workflows	The unique identifier for the VNFC as created in Configuration Manager
-n	vnfc_instance_id	Yes for instantiate_vnfc_without_vim otherwise no	The unique identifier for the VNFC instance. Used in VNFC workflows to target specific VNFCs.
-reinstall_ra	reinstall_ra	No	<p>Whether the Resource Agent is reinstalled during the workflow. Used only with instantiate_vnfc_without_vim. Possible values are:</p> <ul style="list-style-type: none"> • true—the Resource Agent is reinstalled • false—the Resource Agent is not reinstalled, but it must be available for the workflow to succeed. <p>The default value is true.</p>

CLI Flag	Parameter	Mand.	Description
-start	start	No	<p>A flag used to specify whether the workflow should start the VNFC after instantiation. Used only with the vnf_instantiate_without_vim workflow. Possible values are:</p> <ul style="list-style-type: none"> • true—the VNFC is started when instantiated • false—the VNFC is not started when instantiated <p>The default value is true.</p>
-restart	restart	No	<p>A flag used to signify whether the workflow should restart the system. Used only with the vnf_update workflow. When set to false, the workflow does not restart the VNF. The default value of true is used when the restart parameter is not present.</p>
-hostname	hostname	No	<p>The hostname on which the target VNFC is located. Used only with the heal_vnfc flow.</p>
-sync	sync	No	<p>The optional sync parameter to specify whether the call is synchronous (1) or asynchronous (0). The call is asynchronous by default.</p>
-timeout	timeout	No	<p>The optional timeout parameter to specify how long the synchronous call waits for the operation to complete. The default value is 60 seconds.</p> <p>Note: The default timeout value can be specified in the front-end: synchronous-request-timeout attribute in the installation file and propagated to the Front-End microservice using the deploy script.</p>
-o	execution_order	No	<p>The optional execution_order parameter to specify whether VNFC instances are upgraded in parallel or in sequence. Possible values are:</p> <ul style="list-style-type: none"> • sequential • parallel <p>The default value of sequential is used when the parameter is not present. The parallel mode is used for out-of-service execution.</p> <p>The execution_order parameter can have an affect on the following workflows:</p> <ul style="list-style-type: none"> • upgrade_vnf • update_vnfc • rollback_vnf <p>The execution_order is typically set to parallel when a workflow applies to VNFC instances that are cluster-aware. These instances must be started simultaneously and stopped simultaneously. System availability might be interrupted while the workflow runs. Therefore a maintenance window is required to run workflows in parallel.</p>

CLI Flag	Parameter	Mand.	Description			
-force	force	No	Value	Value Description	Workflow	Effect on Workflow
			0	The default value, which is the value assumed when the -force option is not present in the command.	update_vnf upgrade_vnf	The VNF template is compared to the existing VNF template during staging. When there is no difference between the two the workflow exits.
			1	Applies to all VNFCs associated with the VNF.	update_vnf upgrade_vnf	Processing continues for a VNFC instance regardless of whether there is a difference between the existing VNF template and the new template to be used.
					terminate_vnf_without_vim	The workflow continues to the clean_up procedure even when the uninstall_vnfc procedure fails. In this event, a warning is returned.
			Comma-separated list of VNFCs ("<vnfc_1>", "<vnfc_2>", ...)	Applies to the specified VNFC IDs.	update_vnf upgrade_vnf	Processing continues for the specified VNFC instances regardless of whether there is a difference between the existing VNF template and the new template to be used.
					terminate_vnf_without_vim	N/A
-continue_on_failure	continue_on_failure	No	<p>An optional parameter to indicate whether the rollback workflow continues regardless of the presence of a VNFC rollback directory. Used only with the vnfc_rollback workflow.</p> <p>The default value is false. The default behavior is that when a VNFC rollback directory does not exist the vnfc_rollback workflow fails due to the missing directory. If the parameter is set to true provided with value set to true, rollback continues even when the rollback directory is not found.</p>			

CLI Flag	Parameter	Mand.	Description
-a	additional_parameters	No	<p>Generic parameters in JSON format that are passed through the workflow submit command to the procedural level on Resource Agents. These parameters are used to extend the command's flexibility to support end-user scenarios. The potential impact on the command depends upon which workflow you are running.</p> <p>For example, when running the upgrade workflow, you can take a snapshot of the VNF instance before performing the upgrade. Openet Weaver does not have a defined workflow submit parameter for this action, but a <code>take_a_snapshot</code> parameter might be defined for the actual procedure, and the parameter-value pair can be passed through the workflow submit command to the Resource Agent for processing.</p> <p>You can predefine additional parameters for use with the VNF-M GUI. When you do, the predefined additional parameters display in the dialog box that opens when you select a workflow to run.</p> <p>Note: See Managing Additional Parameters for the VNF-M GUI for more information about predefining additional parameters.</p> <p>Additional parameters can be used with the following workflows:</p> <ul style="list-style-type: none"> • <code>instantiate_vnfc_without_vim</code> • <code>upgrade_vnfc</code> • <code>update</code> • <code>rollback_vnfc</code> • <code>start_vnfc</code> • <code>stop_vnfc</code> • <code>custom_action_vnfc</code>

The predefined workflows that are provided with Openet Weaver are described in the following topics:

Reinstantiating VNFCs without VIM

When a VNF is instantiated, the workflow instantiates all VNFCs in the VNF. However, there are some instances when you might want to reinstantiate a VNFC that has no VIM configuration in an already instantiated VNF, for example:

- When a VM fails or is down and a second VM is switched on to replace it
- When a broken VNFC instance is detected and must be re-instantiated without affecting any other VNFCs in the VNF
- When there are topology changes in the VNF topology that affect specific VNFC Instances

Note: There is currently no similar workflow for reinstantiating VNFCs with VIM.

Note: For more information about instantiating VNFs without VIM see [Instantiating a VNF without VIM](#).

The following restrictions and limitations apply to running `instantiate_vnfc_without_vim`:

- You cannot instantiate a VNFC that belongs to an uninstantiated VNF
- There is no check performed to verify whether the VNFC is already instantiated, so it is possible to reinstantiate an already instantiated VNFC
- In the event that an upgrade was performed before VNFC instantiation, rollback directories are no longer available after VNFC instantiation, therefore, rollback cannot be performed after a VNFC instantiation
- The VNFC is not terminated (uninstalled) prior to instantiation
- Only one VNFC at a time can be instantiated

The `instantiate_vnfc_without_vim` workflow is shown below.

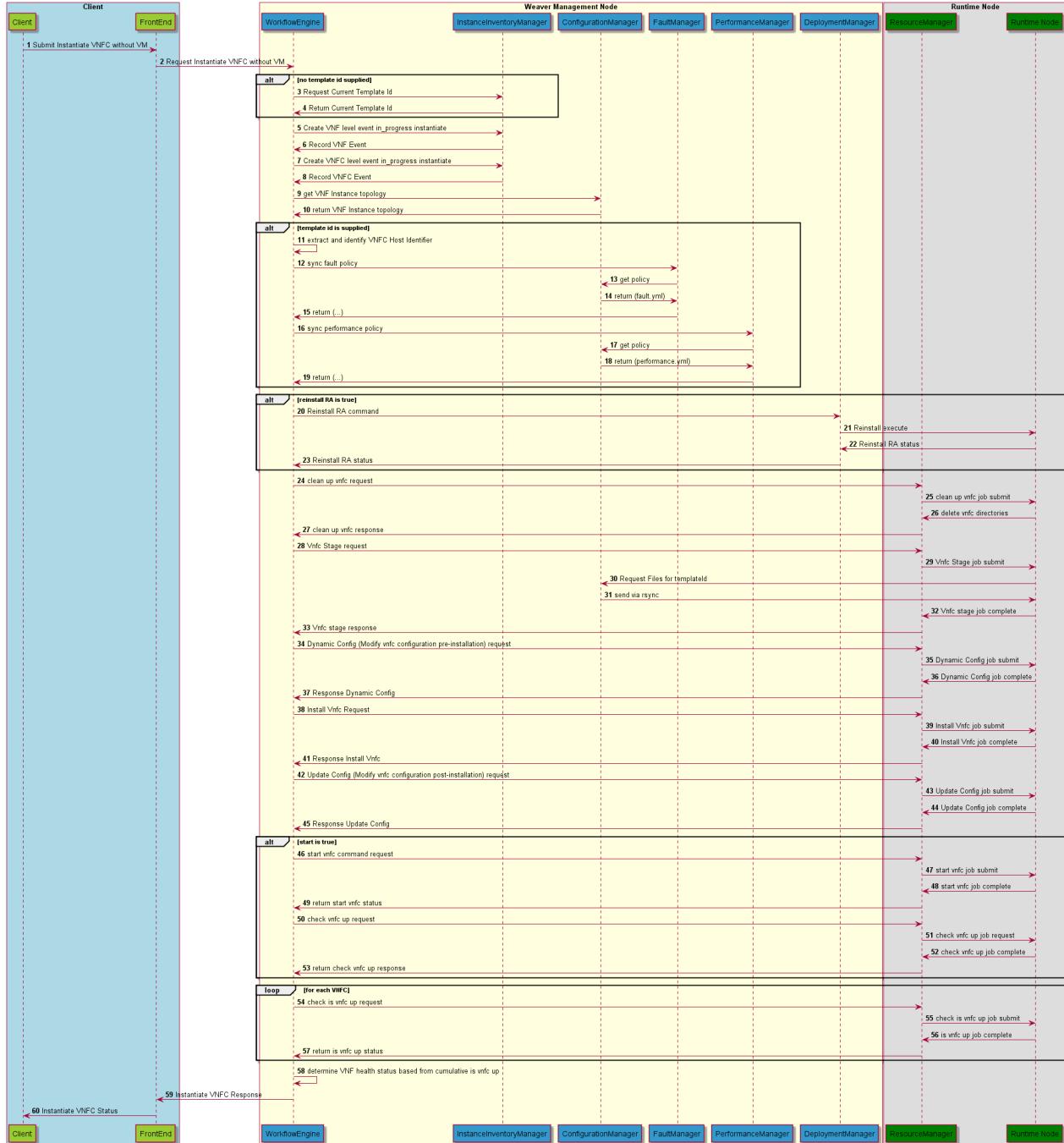


Figure 87: Reinstantiate VNFC without VIM

Reinstantiating a VNFC from the CLI

To reinstantiate a VNFC with no VIM configuration from the CLI, use the `instantiate_vnfc_without_vim` workflow_type parameter in the `vnfc_workflow submit` command as shown below.

```
ovlm-fe vnfc_workflow submit -y instantiate_vnfc_without_vim -t <tenant_id>
-v <vnf_id> -i <vnf_instance_id> -c <vnfc_id> -n <vnfc_instance_id>
[-a '{<additional_parameter1>, <additional_parameter2>, ...}'] [-reinstall_ra
<true/false>]
[-p <vnf_template_id>] [-start <true | false>] [-sync <1/0>]
```

Note: All parameters are described on the main page for this section, [Running VNFC Workflows](#).

The command reinstatiates a VNFC that is part of an already instantiated VNF. When the optional `reinstall_ra` parameter is not used or is set to true, the Resource Agent is reinstalled by default. When the optional `-p` parameter is not used or is set to false, IIM finds the associated VNF template in the VNFC instance if it exists. If it does not exist then the VNF-level `template_id` is used.

Only one VNFC can be instantiated at a time.

The following is an example of instantiating a VNFC:

```
ovlm-fe vnfc_workflow submit -sync 1 -t default -v microblog -i dc_1 -c nginx
-n 1 -y instantiate_vnfc_without_vim
-p microblog_v1
```

Note: Be sure the baseurl environment variables are set before using CLI commands.

The following is an example of a successful response to running the command.

```
data:
  workflow_instance_status: COMPLETED
  success_msg: Instantiation of VNFC has completed.
  flow_result:
    health_status: UP
    vnfc_instances:
      - vnfc_id: blogserver
        vnfc_instance_id: '1'
        hostname: 192.168.122.165
        exit_code: ''
        response:
          health_status: UP
          output:
            - Instantiation is not performed.
      - vnfc_id: blogserver
        vnfc_instance_id: '2'
        hostname: 192.168.122.48
        exit_code: ''
        response:
          health_status: UP
          output:
            - Instantiation is not performed.
      - vnfc_id: nginx
        vnfc_instance_id: '1'
        hostname: 192.168.122.118
        exit_code: '0'
        response:
          health_status: UP
          output:
            - Instantiation has completed.
      - vnfc_id: postgres
        vnfc_instance_id: '1'
        hostname: 192.168.122.242
        exit_code: ''
        response:
          health_status: UP
          output:
            - Instantiation is not performed.
  request: http://kl-005006-vm05.openet-telecom.lan:28085/api/v2/tenants/default/vnfs/microblog/vnf_instances/dc_1/vnfc_instances/workflow_instances
  method: POST
  status: 200
```

Reinstantiating a VNFC Using REST API

To reinstantiate a VNFC with no VIM configuration using REST API , post a workflow request using the instantiate_vnfc_without_vim workflow_type parameter.

Note: The associated VNF must already instantiated.

The API request takes the following format.

```
POST http://<ovlm-workflow-fe-url>/api/v2/tenants/<tenant_id>/vnfs/<vnf_id>/vnf_instances/<vnf_instance_id>/vnfc_instances/workflow_instances?sync=1&timeout=2000
{
    "workflow_type": "instantiate_vnfc_without_vim",
    "vnfc_id": <vnfc_id>,
    "vnfc_instance_id": <vnfc_instance_id>,
    ["vnf_template_id": <vnf_template_id>],
    ["additional_parameters": <additional_parameters>],
    ["start": <true|false>],
    ["reinstall_ra": <true|false>]
}
```

Note: All parameters are described on the main page for this section, [Running VNFC Workflows](#).

When the optional reinstall_ra parameter is not used or is set to true, the Resource Agent is reinstalled by default. When the optional -p parameter is not used or is set to false, IIM finds the associated VNF template in the VNFC instance if it exists. If it does not exist then the VNF-level template_id is used.

The following is an example of instantiating a VNFC :

```
curl -s -X POST -H Content-Type: application/json -d {"workflow_type": "instantiate_vnfc_without_vim", "vnfc_id": "nginx", "vnfc_instance_id": "1", "vnf_template_id": "microblog_v1", "additional_parameters": ""} http://10.3.18.22:28085/api/v2/tenants/default/vnfs/microblog/vnf_instances/dc_1/vnfc_instances/workflow_instances?sync=1&timeout=2000
```

The following is an example of a successful response.

```
{
  "data": {
    "workflow_instance_status": "COMPLETED",
    "success_msg": "Instantiation of VNFC has completed.",
    "flow_result": {
      "health_status": "UP",
      "vnfc_instances": [
        {
          "vnfc_id": "blogserver",
          "vnfc_instance_id": "1",
          "hostname": "192.168.122.165",
          "exit_code": "",
          "response": {
            "health_status": "UP",
            "output": ["Instantiation is not performed."]
          }
        },
        {
          "vnfc_id": "blogserver",
          "vnfc_instance_id": "2",
          "hostname": "192.168.122.48",
          "exit_code": "",
          "response": {
            "health_status": "UP",
            "output": ["Instantiation is not performed."]
          }
        }
      ]
    }
  }
}
```

```

    "vnfc_id": "nginx",
    "vnfc_instance_id": "1",
    "hostname": "192.168.122.118",
    "exit_code": "0",
    "response": {
        "health_status": "UP",
        "output": ["Instantiation has completed."]
    }
},
{
    "vnfc_id": "postgres",
    "vnfc_instance_id": "1",
    "hostname": "192.168.122.242",
    "exit_code": "",
    "response": {
        "health_status": "UP",
        "output": ["Instantiation is not performed."]
    }
}
],
"request": "http://10.3.18.22:28085/api/v2/tenants/default/vnfs/microblog/vnf_instances/dc_1/vnfc_instances/workflow_instances",
"method": "POST"
},
"status": 200
}

```

Upgrading VNFCs

To upgrade one or more VNFCs in an instantiated VNF to a newer software version or a new configuration, you run the `upgrade_vnfc` workflow. You can run the workflow against all VNFCs of the same type or a specific VNFC instance. The workflow can be run only on VNFs that are already instantiated.

You can upgrade VNF from the CLI, the API, or from the Openet Weaver VNF-M GUI.

The optional `additional_parameters` parameter allows you to pass parameters in JSON format through the workflow `submit` command to the procedural level on Resource Agents. These parameters are available as input parameters for the procedures.

Upgrading a VNFC from the CLI

To perform a partial VNF upgrade from the CLI, you can upgrade specific VNFCs using `upgrade_vnfc` as the `workflow_type` parameter with the `vnfc_workflow submit` command as shown below.

```

ovlm-fe vnfc_workflow submit -t <tenant_id> -v <vnf_id> -i <vnf_instance_id>
-y upgrade_vnfc -c <vnfc_id>
[-n <vnfc_instance_id>] [-restart <true | false>] [-sync <sync>] [-timeout
<seconds>] [-o <execution_order>]
[-force <force_value>] [-a '{<additional_parameter1, additional_parameter2,
...>}']

```

Note: All parameters are described on the main page for this section, [Running VNFC Workflows](#).

The command upgrades all VNFC instances with the specified `vnfc_id`. When the optional `vnfc_instance_id` parameter is also used in the command, only that VNFC instance is upgraded.

Example: Upgrading all VNFCs with the Same `vnfc_id`

This example shows a partial VNF upgrade of all VNFCs with the same `vnfc_id` within in an instance of a microblog VNF named `dc_1`.

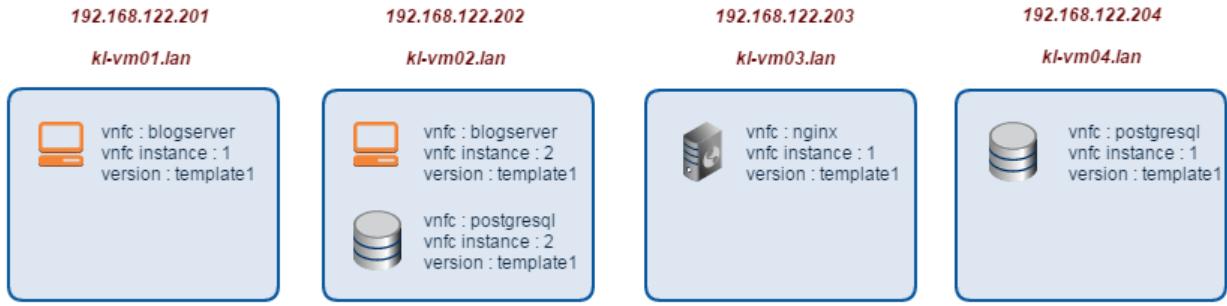


Figure 88: Microblog VNF

The following CLI command upgrades all VNFCs with the same `vnfd_id` (blogserver) within `dc_1`.

```
ovlm-fe vnfc_workflow submit -y upgrade_vnfc -t Default -v microblog -i dc_1
-c
blogserver -p template2 -a '{"take_a_snapshot": "true"}'
```

Note: Be sure the `baseurl` environment variables are set before using CLI commands.

The following is an example of a successful response to running the command.

```
ovlm-fe vnfc_workflow submit -y upgrade_vnfc -t default -v microblog -i dc_1
-c blogserver -p microblog_v2 -sync 1
data:
  workflow_instance_status: COMPLETED
  success_msg: Upgrade of VNFC has completed.
  flow_result:
    health_status: UP
    vnfc_instances:
      - vnfc_id: blogserver
        vnfc_instance_id: '1'
        hostname: 192.168.122.220
        exit_code: '0'
        response:
          health_status: UP
          output:
            - Upgrade has completed.
      - vnfc_id: blogserver
        vnfc_instance_id: '2'
        hostname: 192.168.122.105
        exit_code: '0'
        response:
          health_status: UP
          output:
            - Upgrade has completed.
      - vnfc_id: nginx
        vnfc_instance_id: '1'
        hostname: 192.168.122.180
        exit_code: ''
        response:
          health_status: UP
          output:
            - Upgrade is not performed.
      - vnfc_id: postgres
        vnfc_instance_id: '1'
        hostname: 192.168.122.72
        exit_code: ''
        response:
```

```

health_status: UP
output:
- Upgrade is not performed.
request: http://ovlm-vm:8080/api/v2/tenants/default/vnfs/microblog/vnf_instances/dc_1/vnfc_instances/workflow_instances
method: POST
status: 200

```

As a result of the successful call, the blogserver VNFCs are upgraded to template2 while the other VNFCs remain on template1 as shown.

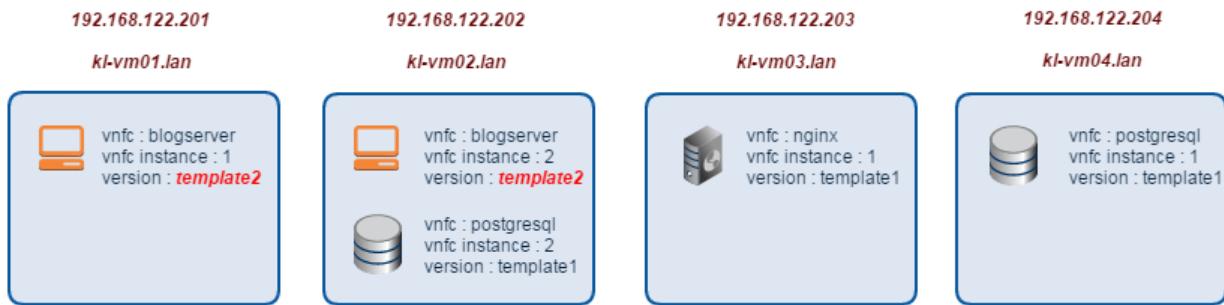


Figure 89: Microblog VNF After partial Upgrade

Example: Upgrading a Specific VNFC Instance

This example shows a partial VNF upgrade of a specific VNFC within in an instance of a microblog VNF named dc_1.

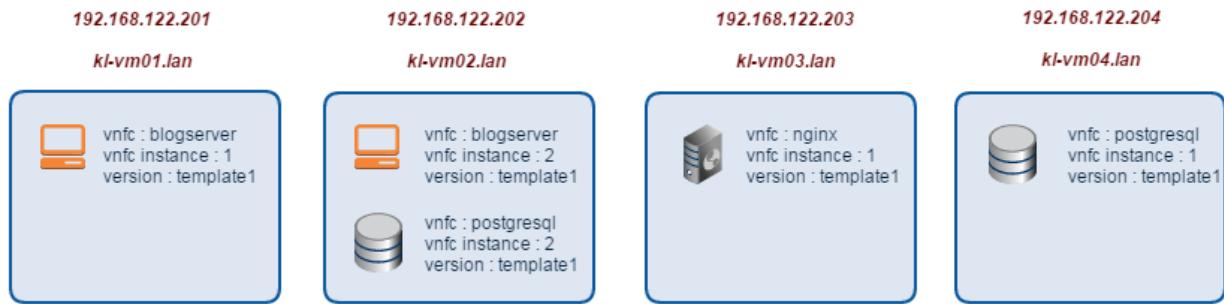


Figure 90: Microblog VNF

The following CLI command example upgrades a specific instance (2) of the blogserver VNFC in an instance of the microblog VNF.

```

ovlm-fe vnfc_workflow submit -y upgrade_vnfc -t Default -v microblog -i dc_1
-c blogserver -n 2
blogserver -p template2

```

The following is an example of a successful response to running the command.

```

data:
workflow_instance_status: COMPLETED
success_msg: Upgrade of VNFC has completed.
flow_result:
  health_status: UP
  vnfc_instances:
    - vnfc_id: blogserver

```

```

vnfc_instance_id: '1'
hostname: 192.168.122.220
exit_code: ''
response:
    health_status: UP
    output:
        - Upgrade is not performed.
- vnfc_id: blogserver
vnfc_instance_id: '2'
hostname: 192.168.122.105
exit_code: '0'
response:
    health_status: UP
    output:
        - Upgrade has completed.
- vnfc_id: nginx
vnfc_instance_id: '1'
hostname: 192.168.122.180
exit_code: ''
response:
    health_status: UP
    output:
        - Upgrade is not performed.
- vnfc_id: postgres
vnfc_instance_id: '1'
hostname: 192.168.122.72
exit_code: ''
response:
    health_status: UP
    output:
        - Upgrade is not performed.
request: http://ovlm-vm:8080/api/v2/tenants/default/vnfs/microblog/vnf_instances/dc_1/vnfc_instances/workflow_instances
method: POST
status: 200

```

As a result of the successful call, instance 2 of the blogserver VNFC is upgraded to template2 while the other VNFCs remain on template1 as shown.

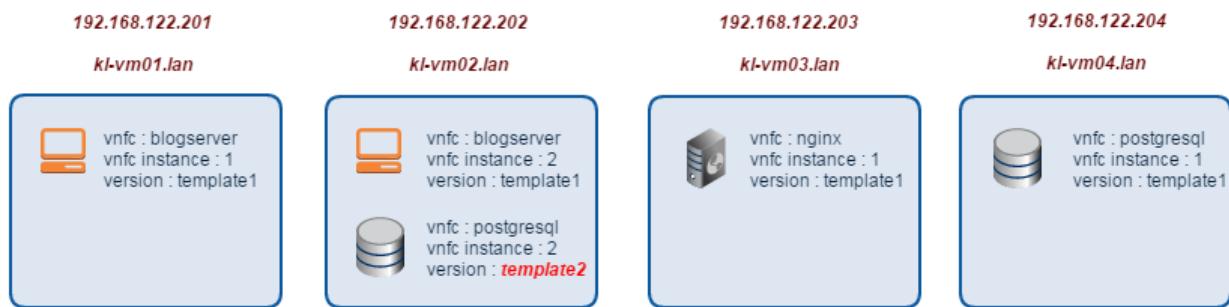


Figure 91: Microblog VNF After partial Upgrade

Upgrading a VNFC Using REST API

To perform a partial upgrade of a VNF using REST API, you can upgrade specific VNFCs using the upgrade_vnfc_workflow_type parameter. The API request takes the following format.

```

POST http://<ovlm-workflow-fe-url>/api/v2/tenants/<tenant_id>/vnfs/<vnf_id>/vnf_instances/<vnf_instance_id>/vnfc_instances/workflow_instances?sync=1&timeout=2000

```

```
{
    "workflow_type": "upgrade_vnfc",
    "vnfc_id": <vnfc_id>,
    ["vnfc_instance_id": <vnfc_instance_id>],
    "vnf_template_id": <vnf_template_id>,
    ["additional_parameters": <additional_parameters>],
    ["execution_order": <sequential|parallel>]
}
```

Note: All parameters are described on the main page for this section, [Running VNFC Workflows](#).

The vnfc_instance_id parameter is optional. When you do not use it, all of the VNFCs with the same vnfc_id within a VNF are upgraded. When you do use the parameter, only the VNFC with the specified instance ID is upgraded.

Example: Upgrading all VNFCs with the Same vnfc_id

In this example of a partial VNF upgrade, both VNFCs with the same vnfc_id (blogserver) are upgraded in an instance of the microblog VNF. The VNF instance name is dc_1.

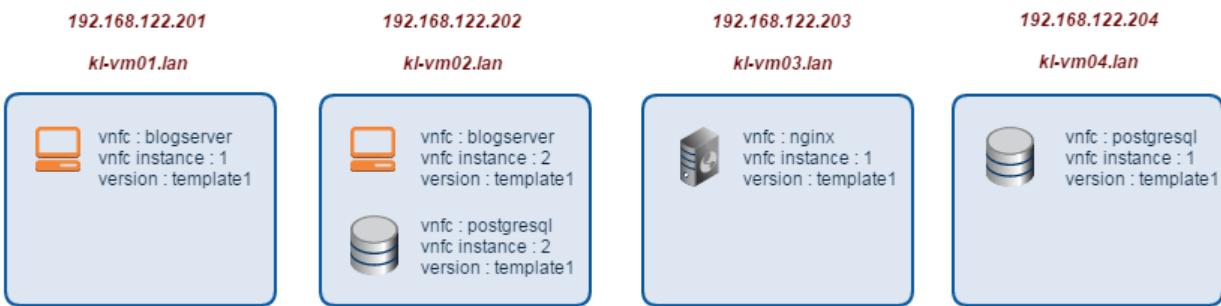


Figure 92: Microblog VNF

The API call to upgrade the blogserver VNFCs using curl is shown below. In this case, vnfc_instance_id is not used in the call.

```
upgrade_vnfc_type.sh default microblog dc_1 blogserver microblog_v2 '{
  \"passedKey1\": \"passedKeyValue1\"
}' http://10.3.18.22:28085
- upgrade vnfc type -
. tenant_id           = default
. vnf_id               = microblog
. vnf_instance_id     = dc_1
. vnfc_id              = blogserver
. vnf_template_id      = template2
}

===== sample script =====

OVLMFE_BASEURL=http://10.3.18.22:28085
echo $OVLMFE_BASEURL
if [ $# -lt 5 ]; then
  echo "usage : $0 <tenant_id> <vnf_id> <vnf_instance> <vnfc_id>
<vnf_template_id> [<additional_parameters>]"
  exit 1
fi
tenant_id=$1
vnf_id=$2
vnf_instance_id=$3
vnfc_id=$4
vnf_template_id=$5
```

```

echo " - upgrade vnfc type - "
echo ". tenant_id" = $tenant_id"
echo ". vnf_id" = $vnf_id"
echo ". vnf_instance_id" = $vnf_instance_id"
echo ". vnfc_id" = $vnfc_id"
echo ". vnf_template_id" = $vnf_template_id"

body={"\\"workflow_type\\": \\"upgrade_vnfc\\", \\"vnfc_id\\": \"$vnfc_id\", \\"vnf_template_id\\": \"$vnf_template_id\", \\"additional_parameters\\": \"$additional_parameters\", \\"execution_order\\": \\"parallel\\\" }"

curl -s -X POST -H "Content-Type: application/json" \
-d "$body" \
"${OVLME_BASEURL}/api/v2/tenants/${tenant_id}/vnfs/${vnf_id}/vnf_instances/${vnf_instance_id}/vnfc_instances/workflow_instances?sync=1&timeout=2000"
| python -mjson.tool | pygmentize -l json

```

The following is an example of a successful response.

```
{
  "data": {
    "workflow_instance_status": "COMPLETED",
    "success_msg": "Upgrade of VNFC has completed.",
    "flow_result": [
      {
        "health_status": "UP",
        "vnfc_instances": [
          {
            "vnfc_id": "blogserver",
            "vnfc_instance_id": "1",
            "hostname": "192.168.122.59",
            "exit_code": "0",
            "response": {
              "health_status": "UP",
              "output": ["Upgrade has completed."]
            }
          },
          {
            "vnfc_id": "blogserver",
            "vnfc_instance_id": "2",
            "hostname": "192.168.122.199",
            "exit_code": "0",
            "response": {
              "health_status": "UP",
              "output": ["Upgrade has completed."]
            }
          },
          {
            "vnfc_id": "nginx",
            "vnfc_instance_id": "1",
            "hostname": "192.168.122.199",
            "exit_code": "",
            "response": {
              "health_status": "UP",
              "output": ["Upgrade is not performed."]
            }
          },
          {
            "vnfc_id": "postgres",
            "vnfc_instance_id": "1",
            "hostname": "192.168.122.199",
            "exit_code": "",
            "response": {

```

```

        "health_status": "UP",
        "output": ["Upgrade is not performed."]
    }
}
],
"request": "http://ovlm-vm:8080/api/v2/tenants/default/
vnfs/microblog/vnf_instances/dc_1/vnfc_instances/workflow_instances",
"method": "POST"
},
"status": 200
}

```

As a result of the successful call, the blogserver VNFCs are upgraded to template2 while the other VNFCs remain on template1 as shown.

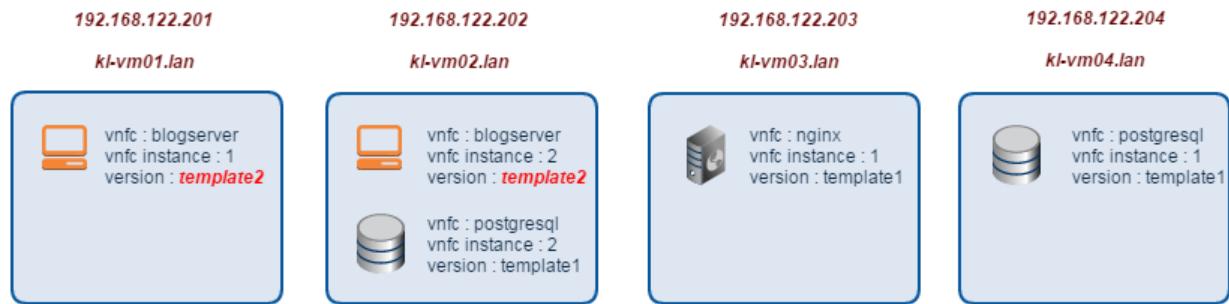


Figure 93: Microblog VNF After partial Upgrade

Example: Upgrading a Specific VNFC Instance

In this example of a partial VNF upgrade, a specific instance of the blogserver VNFC within the microblog VNF is upgraded. The VNFC instance name is 2.

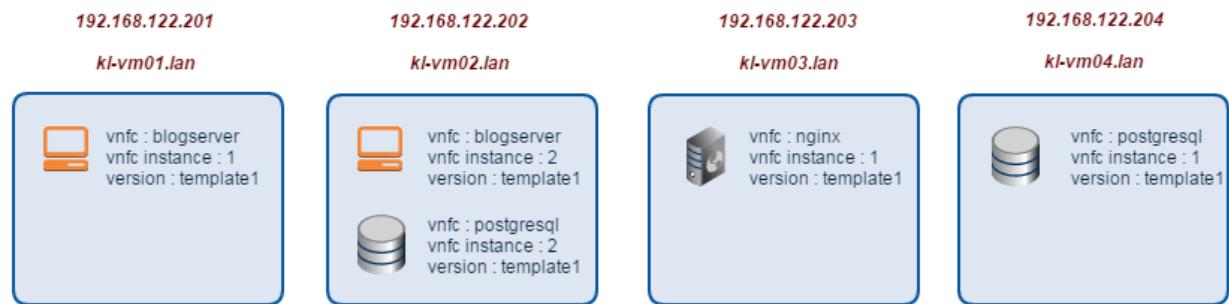


Figure 94: Microblog VNF

The API call to upgrade the blogserver VNFCs using curl is shown below. In this case, vnf_instance_id is used to target a specific instance of the blogserver VNFC .

```

./upgrade_vnfc_instance.sh default microblog dc_1 blogserver 1 microblog_v2
'{ \"passedKey1\": \"passedKeyValue1\" }'
http://10.3.18.22:28085
- upgrade vnfc type -
. tenant_id           = default
. vnf_id               = microblog
. vnf_instance_id     = dc_1
. vnfc_id              = blogserver

```

```

. vnfc_instance_id      = 2
. vnf_template_id       = microblog_v2

===== sample script =====

#!/bin/bash
OVLMFE_BASEURL=http://10.3.18.22:28085
echo $OVLMFE_BASEURL
if [ $# -lt 6 ]; then
    echo "usage : $0 <tenant_id> <vnf_id> <vnf_instance> <vnfc_id>
<vnfc_instance_id> <vnf_template_id> [<additional_parameters>]"
    exit 1
fi
tenant_id=$1
vnf_id=$2
vnf_instance_id=$3
vnfc_id=$4
vnfc_instance_id=$5
vnf_template_id=$6
echo " - upgrade vnfc type - "
echo ". tenant_id          = $tenant_id"
echo ". vnf_id              = $vnf_id"
echo ". vnf_instance_id     = $vnf_instance_id"
echo ". vnfc_id              = $vnfc_id"
echo ". vnfc_instance_id    = $vnfc_instance_id"
echo ". vnf_template_id      = $vnf_template_id"

body={"\\"workflow_type\\": \\"upgrade_vnfc\\", \\"vnfc_id\\": \\"$vnfc_id\\",
\\"vnfc_instance_id\\": \\"$vnfc_instance_id\\", \\"vnf_template_id\\":
\\"$vnf_template_id\\", \\"additional_parameters\\": \"${additional_params}\"}
curl -s -X POST -H "Content-Type: application/json" \
-d "$body" \
"${OVLMFE_BASEURL}/api/v2/tenants/${tenant_id}/vnfs/${vnf_id}/vnf_
instances/${vnf_instance_id}/vnfc_instances/workflow_instances?syn
c=1&timeout=20000" | python -mjson.tool | pygmentize -l json

```

The following is an example of a successful response.

```
{
  "data": {
    "workflow_instance_status": "COMPLETED",
    "success_msg": "Upgrade of VNFC has completed.",
    "flow_result": {
      "health_status": "UP",
      "vnfc_instances": [
        {
          "vnfc_id": "blogserver",
          "vnfc_instance_id": "1",
          "hostname": "192.168.122.59",
          "exit_code": "",
          "response": {
            "health_status": "UP",
            "output": ["Upgrade is not performed."]
          }
        },
        {
          "vnfc_id": "blogserver",
          "vnfc_instance_id": "2",
          "hostname": "192.168.122.199",
          "exit_code": "0",
          "response": {
            "health_status": "UP",

```

```

        "output": ["Upgrade has completed."]
    }
},
{
    "vnfc_id": "nginx",
    "vnfc_instance_id": "1",
    "hostname": "192.168.122.199",
    "exit_code": "",
    "response": {
        "health_status": "UP",
        "output": ["Upgrade is not performed."]
    }
},
{
    "vnfc_id": "postgres",
    "vnfc_instance_id": "1",
    "hostname": "192.168.122.199",
    "exit_code": "",
    "response": {
        "health_status": "UP",
        "output": ["Upgrade is not performed."]
    }
}
],
},
"request": "http://ovlm-vm:8080/api/v2/tenants/default/
vnfs/microblog/vnf_instances/dc_1/vnfc_instances/workflow_instances",
"method": "POST"
},
"status": 200
}

```

As a result of the successful call, instance 2 of the blogserver VNFC is upgraded to template2 while the other VNFCs remain on template1 as shown.

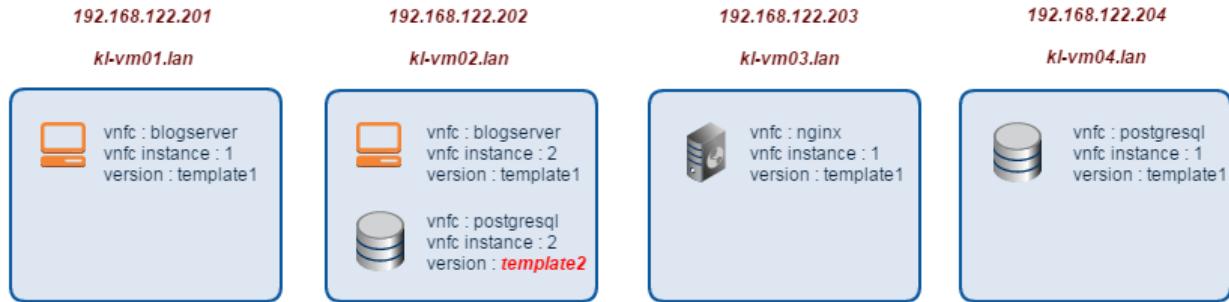


Figure 95: Microblog VNF After partial Upgrade

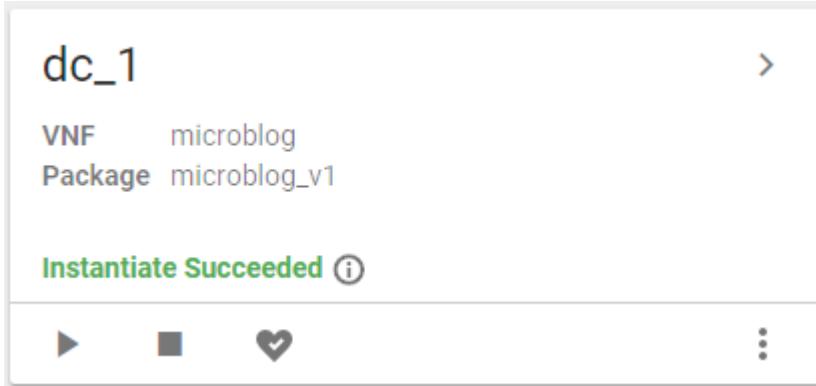
Upgrading VNFCs Using the VNF-M GUI

You can use the VNF-M GUI to upgrade a VNFC, which upgrades all instances of the VNFC, or to upgrade a specific VNFC instance.

Upgrading All Instances of a VNFC

To upgrade all instances of a VNFC, follow these steps

1. On the VNF Lifecycle page, locate the active VNF with the VNFC you intend to upgrade and click the arrow in the top-right corner of the VNF instance card.



Scroll down the page to find the VNF if necessary.

Note: Active VNFs are grouped at the bottom of the VNF Lifecycle page.

Figure 96: Active VNF

The VNFC Page for the selected VNF is displayed with a card for each VNFC.

Figure 97: VNFC Page

2. Click the more options icon



and select **Upgrade** on the VNFC for which you intend to upgrade all instances, which in this example is blogserver. The Upgrade VNFC dialog box is displayed.

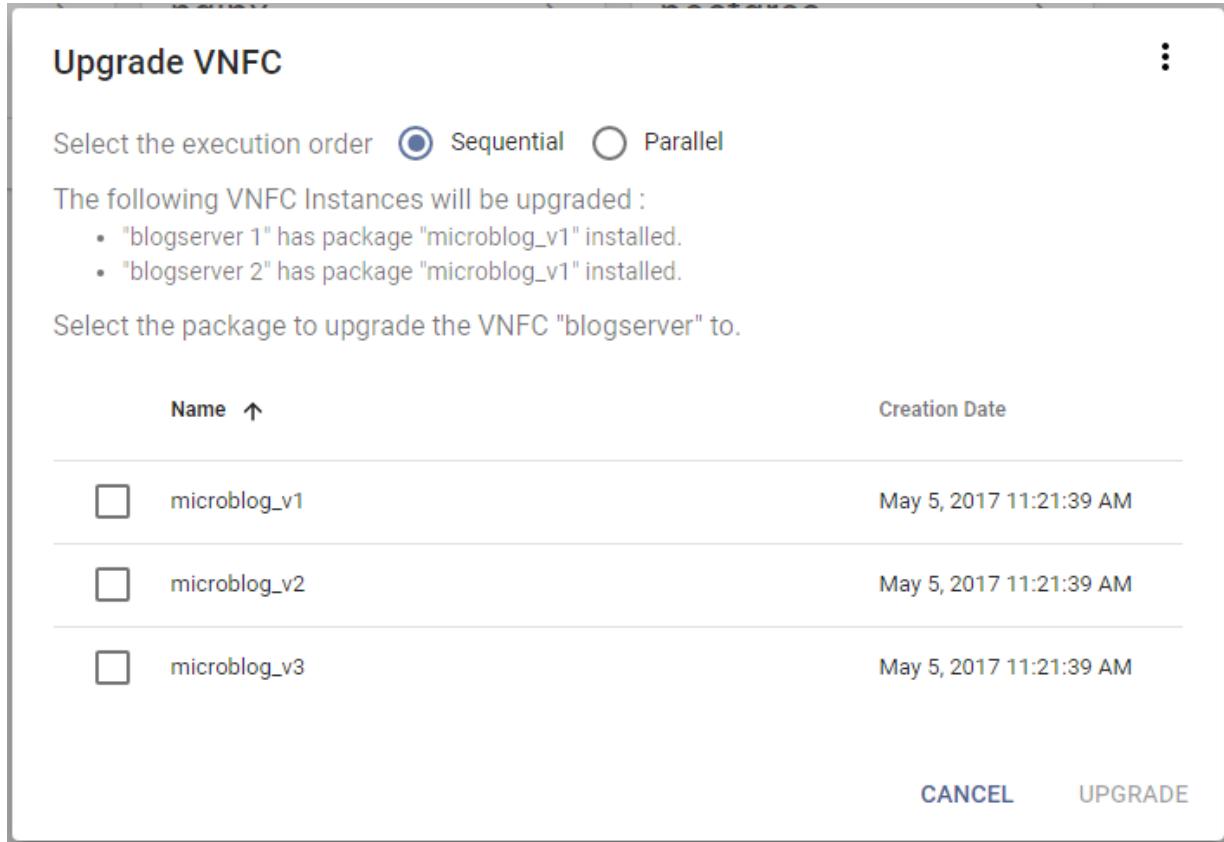


Figure 98: Upgrade VNFC Dialog Box

3. In the **Select the execution order** field, click a radio button to specify whether the update will be made to each VNFC instance in series or in parallel.
4. Select a VNF package for the upgrade by selecting it, for example microblog_v2.
5. To configure additional parameters for a script you are running on the Resource Agent related to upgrading the VNFC, click the icon in the top-right of the confirmation box and select **Add parameters**.

The dialog box displays a field into which you can enter additional parameters. In some cases there might be predefined parameters, which will display when you select **Add Parameters**. Predefined parameters can be overwritten.

Note: For information about how to predefine Additional Parameters, see [Managing Additional Parameters for the VNF-M GUI](#).

Parameters must be entered in JSON format as in the following example.

Upgrade VNFC

Select the execution order Sequential Parallel

The following VNFC Instances will be upgraded :

- "blogserver 1" has package "microblog_v1" installed.
- "blogserver 2" has package "microblog_v1" installed.

Select the package to upgrade the VNFC "blogserver" to.

Name ↑	Creation Date
<input type="checkbox"/> microblog_v1	May 5, 2017 11:21:39 AM
<input checked="" type="checkbox"/> microblog_v2	May 5, 2017 11:21:39 AM
<input type="checkbox"/> microblog_v3	May 5, 2017 11:21:39 AM

Additional Parameters

```
{
  "take_a_snapshot": true
}
```

[CANCEL](#) [UPGRADE](#)

Figure 99: Additional Parameters

Note: To remove additional parameters after adding them, click the top-right icon again and select **Remove parameters**.

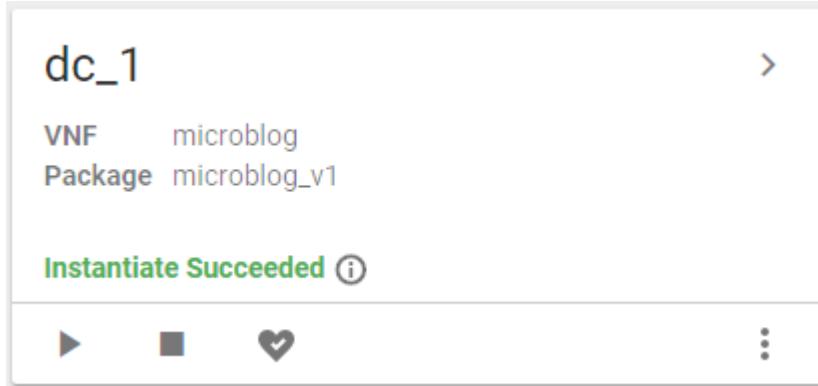
6. Click **Upgrade**.

The VNFC instances are started with a status of Upgrade Succeeded.

[Upgrading a Specific Instance of a VNFC](#)

To Upgrade a specific VNFC instance, follow these steps:

1. On the VNF Lifecycle page, locate the active VNF with the VNFC you intend to upgrade and click the arrow in the top-right corner of the VNF instance card.



Scroll down the page to find the VNF if necessary.

Note: Active VNFs are grouped at the bottom of the VNF Lifecycle page.

Figure 100: Active VNF

The VNFC Page for the selected VNF is displayed with a card for each VNFC.

← microblog - dc_1

default > microblog - dc_1

blogserver >
nginx >
postgres >

▶ ■ ⋮ ▶ ■ ⋮ ▶ ■ ⋮

Figure 101: VNFC Page

2. Click the arrow in the top-right corner of the blogserver VNFC card.
All instances of the blogserver VNFC are displayed.

← blogserver

default > microblog - dc_1 > blogserver

🌐 blogserver VNFC Instances

blogserver 1
Package microblog_v1
Health Check: Running ⓘ

blogserver 2
Package microblog_v1
Health Check: Running ⓘ

▶ ■ ⋮ ▶ ■ ⋮ ▶ ■ ⋮

Figure 102: Blogserver VNFC Instances

3. Click the more options icon



and select **Upgrade** on the VNFC you intend to upgrade.
The Upgrade VNFC Instance dialog box is displayed.

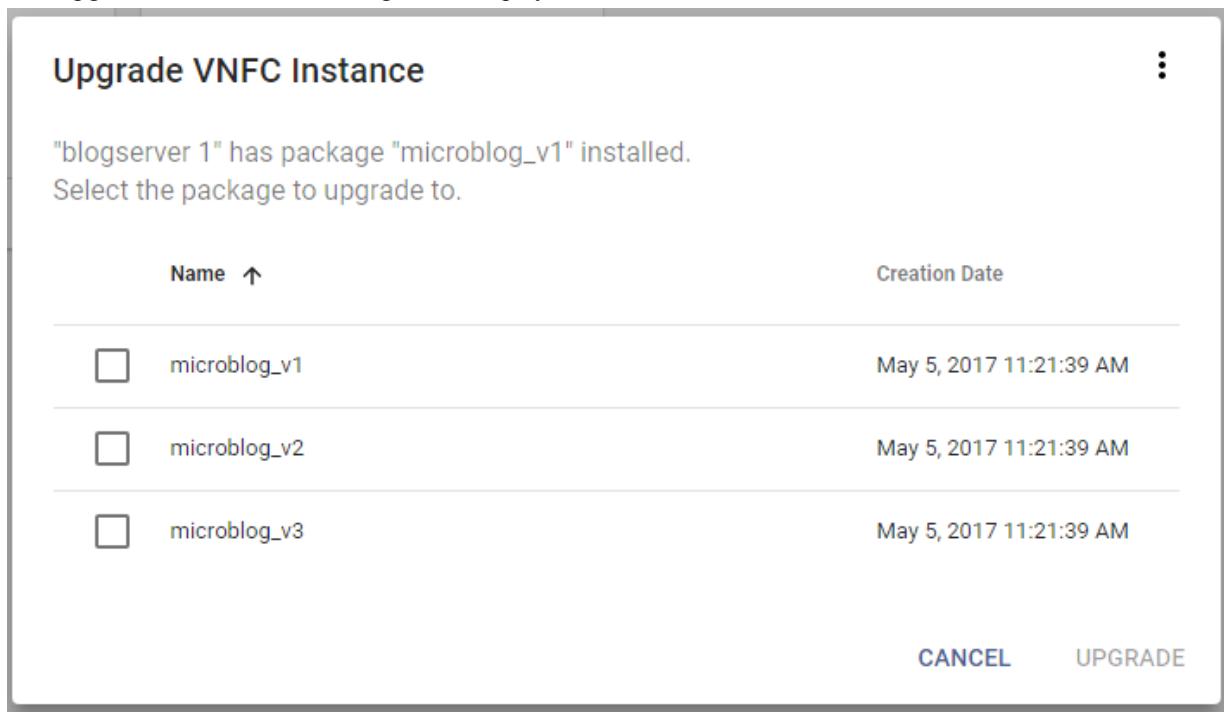


Figure 103: Upgrade VNFC Instance Dialog Box

4. Select a VNF package for the upgrade by selecting it, for example `microblog_v2`.
5. To configure additional parameters for a script you are running on the Resource Agent related to upgrading the VNFC, click the icon in the top-right of the dialog box card and select **Add parameters**.

The dialog box displays a field into which you can enter additional parameters. In some cases there might be predefined parameters, which will display when you select **Add Parameters**. Predefined parameters can be overwritten.

Note: For information about how to predefine Additional Parameters, see [Managing Additional Parameters for the VNF-M GUI](#).

Parameters must be entered in JSON format as in the following example.

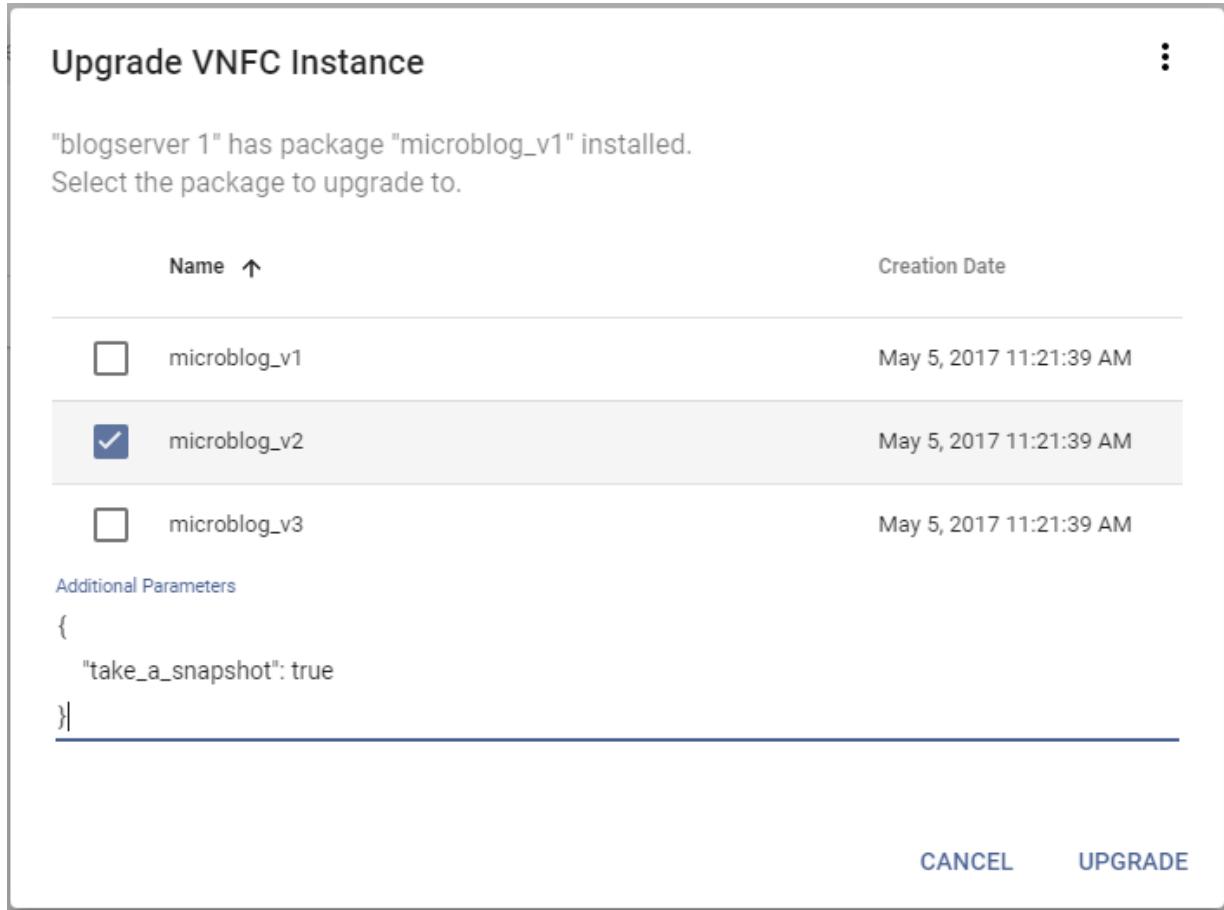


Figure 104: Additional Parameters

Note: To remove additional parameters after adding them, click the top-right icon again and select **Remove parameters**.

6. Click Upgrade.

The VNFC instance is Upgraded with a status of Upgrade Succeeded.

Updating VNFCs

To change the configuration of an instantiated VNFC you run the update_vnfc workflow. You can run the workflow against all VNFCs of the same type or a specific VNFC instance. This workflow can be run only on VNFCs that are already instantiated.

You can update a VNF from the CLI , the API, or from the Openet Weaver VNF-M GUI.

For this workflow, the optional force parameter specifies whether an update continues on VNFC instances regardless of whether the update impacts the instance. You can use the parameter to force an update on all VNFCs or on a specified list of VNFCs.

The optional restart parameter lets you choose whether to restart the VNFC after updating it. Since the VNFC restarts by default, this parameter is required only when you decide to restart the VNF.

Updating a VNFC from the CLI

To perform a partial VNFC update from the CLI, you can update specific VNFCs using `update_vnfc` as the `workflow_type` parameter with the `vnfc_workflow submit` command as shown below.

```
ovlm-fe vnfc_workflow submit -t <tenant_id> -v <vnfc_id> -i <vnf_instance_id>
-y update_vnfc -c <vnfc_id>
[-n <vnfc_instance_id>] [-restart <true | false>] [-sync <sync>] [-timeout
<seconds>] [-o <execution_order>]
[-force <force_value>] [-a '{<additional_parameter1, additional_parameter2,
...>}']
```

Note: All parameters are described on the main page for this section, [Running VNFC Workflows](#).

The command updates all VNFC instances with the specified `vnfc_id`. When the optional `vnfc_instance_id` parameter is also used in the command, only that VNFC instance is updated.

Example: Updating all VNFCs with the Same `vnfc_id`

This example shows a partial VNF update of all VNFCs with the same `vnfc_id` within in an instance of a microblog VNF named `dc_1`.

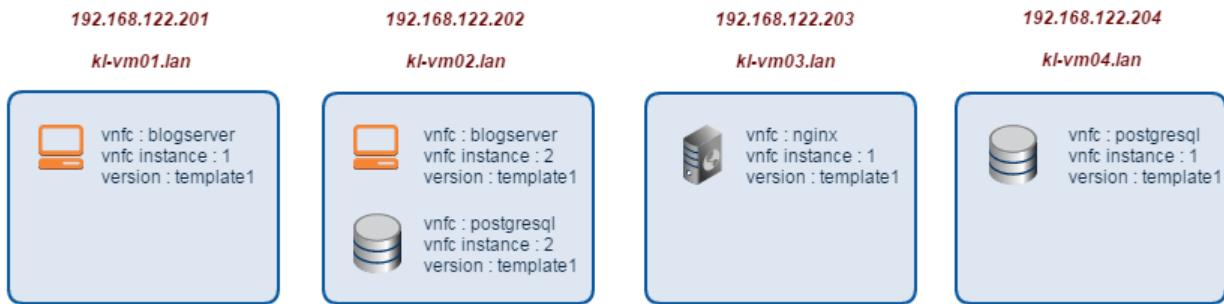


Figure 105: Microblog VNF

The following CLI command updates all VNFCs with the same `vnfd_id` (`blogserver`) within `dc_1`.

```
ovlm-fe vnfc_workflow submit -tdefault-v microblog -i dc_1 -y update_vnfc -c
blogserver -p template2 -a'{"take_a_snapshot": "true"}'
```

Note: Be sure the `baseurl` environment variables are set before using CLI commands.

The following is an example of a successful response to running the command.

```
data:
  flow_result:
    success: '"VNFC Update passed successfully on all servers."'
method: POST
  request: http://10.3.18.22:28085/api/v2/tenants/default/vnfs/microblog/vnf_
instances/dc_1/vnfc_instances/workflow_instances
  workflow_instance_status: COMPLETED
status: 200
```

As a result of the successful call, the `blogserver` VNFCs are updated to `template2` while the other VNFCs remain on `template1` as shown.

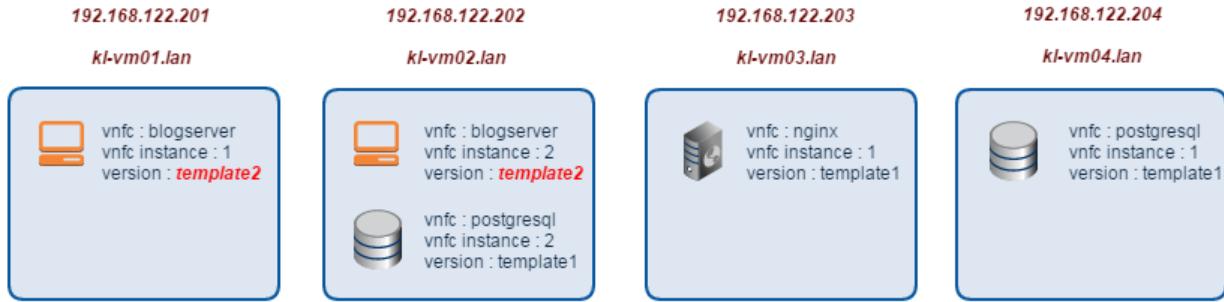


Figure 106: Microblog VNF After partial Update

Example: Updating a Specific VNFC Instance

This example shows a partial VNF update of a specific VNFC within in an instance of a microblog VNF named dc_1.

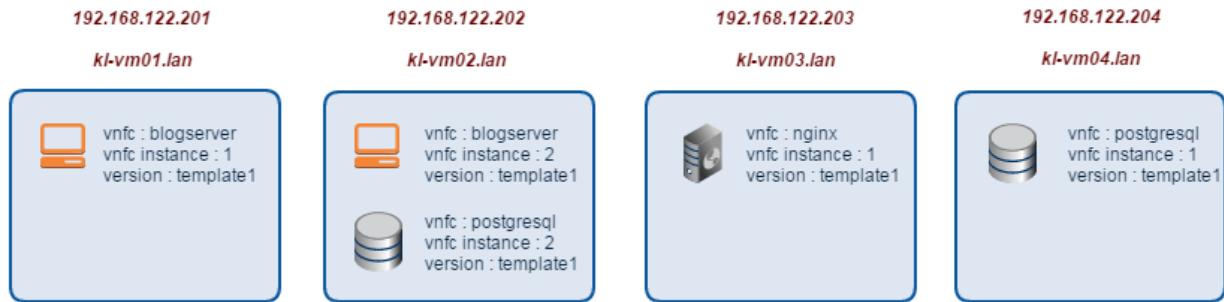


Figure 107: Microblog VNF

The following CLI command example updates a specific instance (2) of the blogserver VNFC in an instance of the microblog VNF.

```
ovlm-fe vnfc_workflow submit -tdefault-v microblog -i dc_1 -y update_vnfc -c
blogserver -n 2 -p template2 -a '{"take_a_snapshot": "true"}'
```

The following is an example of a successful response to running the command.

```
data:
  flow_result:
    success: '"VNFC Update passed successfully on all servers."'
  method: POST
  request: http://10.3.18.22:28085/api/v2/tenants/default/vnfs/microblog/vnf_
instances/dc_1/vnfc_instances/workflow_instances
  workflow_instance_status: COMPLETED
  status: 200
```

As a result of the successful call, instance 2 of the blogserver VNFC is updated to template2 while the other VNFCs remain on template1 as shown.

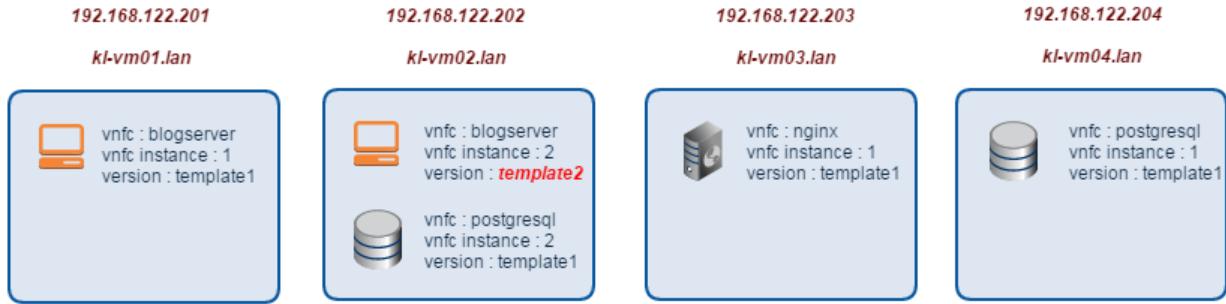


Figure 108: Microblog VNF After partial Update

Updating a VNFC Using REST API

To perform a partial update of a VNF using REST API, you can update specific VNFCs using the update_vnfc workflow_type parameter. The API request takes the following format.

```
POST http://<ovlm-workflow-fe-url>/api/v2/tenants/<tenant_id>/vnfs/<vnf_id>/vnf_instances/<vnf_instance_id>/vnfc_instances/workflow_instances?sync=1&timeout=2000
{
    "workflow_type": "update_vnfc",
    "vnfc_id": <vnfc_id>,
    ["vnfc_instance_id": <vnfc_instance_id>],
    "vnf_template_id": <vnf_template_id>,
    ["additional_parameters": <additional_parameters>],
    ["execution_order": <sequential|parallel>]
    ["restart": <true|false>],
}
```

Note: All parameters are described on the main page for this section, [Running VNFC Workflows](#).

The vnf_instance_id parameter is optional. When you do not use it, all of the VNFCs with the same vnfc_id within a VNF are updated. When you do use the parameter, only the VNFC with the specified instance ID is updated.

Example: Updating all VNFCs with the Same vnfc_id

In this example of a partial VNF update, both VNFCs with the same vnfc_id (blogserver) are updated in an instance of the microblog VNF. The VNF instance name is dc_1.

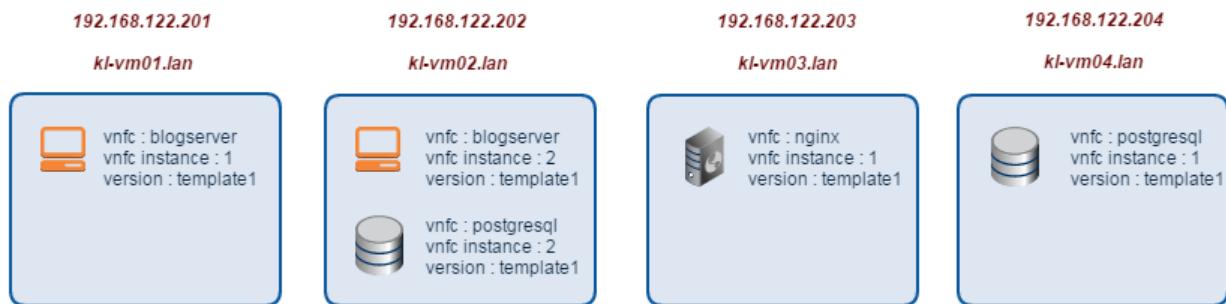


Figure 109: Microblog VNF

The API call to update the blogserver VNFCs using curl is shown below. In this case, vnfc_instance_id is not used in the call.

```
update_vnfc_type.sh default microblog dc_1 blogserver microblog_v2 false '{
  \"passedKey1\": \"passedKeyValue1\" }'
http://10.3.18.22:28085
- update vnfc type -
. tenant_id           = default
. vnf_id               = microblog
. vnf_instance_id     = dc_1
. vnfc_id              = blogserver
. vnf_template_id      = microblog_v2
. restart              = false
}

===== sample script =====

OVLMFE_BASEURL=http://10.3.18.22:28085
if [ $# -lt 7 ]; then
    echo "usage : $0 <tenant_id> <vnf_id> <vnf_instance> <vnfc_id>
<vnf_template_id> <restart> [<additional_parameters>]"
    exit 1
fi
tenant_id=$1
vnf_id=$2
vnf_instance_id=$3
vnfc_id=$4
vnf_template_id=$5
restart=$6
additional_parameters=$7
echo " - update vnfc type - "
echo " . tenant_id          = $tenant_id"
echo " . vnf_id              = $vnf_id"
echo " . vnf_instance_id     = $vnf_instance_id"
echo " . vnfc_id              = $vnfc_id"
echo " . vnf_template_id      = $vnf_template_id"
echo " . restart              = $restart"
echo " . additional_parameters = $additional_parameters"
body="{"workflow_type": "update_vnfc", "vnfc_id": "${vnfc_id}\",
"vnf_template_id": "${vnf_template_id}\", \"additional_parameters\":
\"${additional_parameters}\", \"execution_order\": \"parallel\", \"restart\":
\"${restart}\" }"

curl -s -X POST -H "Content-Type: application/json" \
-d "$body" \
"${OVLMFE_BASEURL}/api/v2/tenants/${tenant_id}/vnfs/${vnf_id}/vnf_
instances/${vnf_instance_id}/vnfc_instances/workflow_instances?syn
c=1&timeout=2000"
```

The following is an example of a successful response.

```
{
  "data": {
    "flow_result": {
      "success": "\"VNFC Update passed successfully on all servers.\""
    },
    "method": "POST",
    "request": "http://10.3.18.22:28085/api/v2/tenants/defa
ult/vnfs/microblog/vnf_instances/dc_1/vnfc_instances/workflow_instances",
    "workflow_instance_status": "COMPLETED"
  },
}
```

```

        "status": 200
    }
}

```

As a result of the successful call, the blogserver VNFCs are updated to template2 while the other VNFCs remain on template1 as shown.

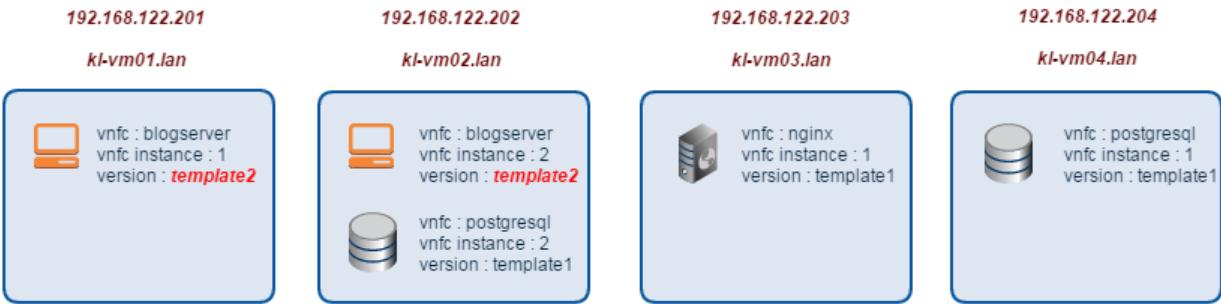


Figure 110: Microblog VNF After partial Update

Example: Updating a Specific VNFC Instance

In this example of a partial VNF update, a specific instance of the blogserver VNFC within the microblog VNF is updated. The VNFC instance name is 2.

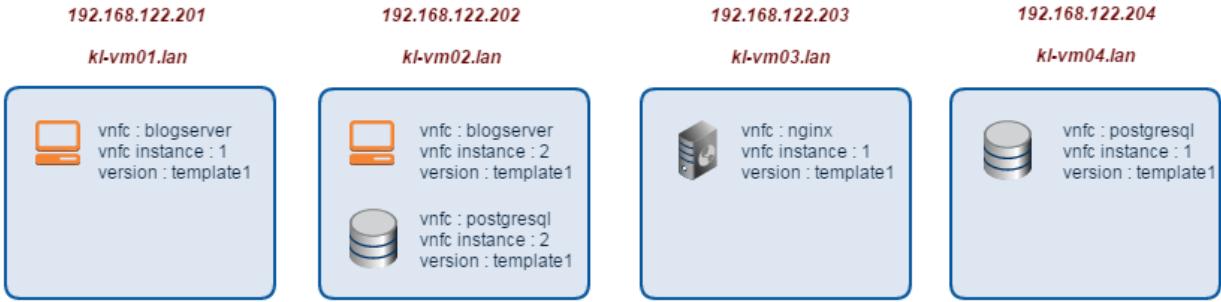


Figure 111: Microblog VNF

The API call to update the blogserver VNFCs using curl is shown below. In this case, vnf_instance_id is used to target a specific instance of the blogserver VNFC .

```

update_vnfc_instance.sh default microblog dc_1 blogserver 2 microblog_v2 false
'{"passedKey1": "passedKeyValue1"}'
http://10.3.18.22:28085
- update vnfc instance -
. tenant_id           = default
. vnf_id               = microBlog
. vnf_instance_id     = dc_1
. vnf_id               = blogserver
. vnf_instance_id     = 2
. vnf_template_id     = microblog_v2
. restart              = false
}

===== sample script =====

OVLMFE_BASEURL=http://10.3.18.22:28085
if [ $# -lt 8 ]; then
    echo "usage : $0 <tenant_id> <vnf_id> <vnf_instance> <vnfc_id>

```

```

<vnfc_instance_id> <vnf_template_id> <restart> [<additional_parameters>]"
    exit 1
fi
tenant_id=$1
vnf_id=$2
vnf_instance_id=$3
vnfc_id=$4
vnfc_instance_id=$5
vnf_template_id=$6
restart=$7
additional_parameters=$8
echo " - update vnfc instance - "
echo ". tenant_id = $tenant_id"
echo ". vnf_id = $vnf_id"
echo ". vnf_instance_id = $vnf_instance_id"
echo ". vnfc_id = $vnfc_id"
echo ". vnfc_instance_id = $vnfc_instance_id"
echo ". vnf_template_id = $vnf_template_id"
echo ". restart = $restart"
echo ". additional_parameters = $additional_parameters"
body="{"workflow_type": "update_vnfc", "vnfc_id": "${vnfc_id}\",
"vnfc_instance_id": "${vnfc_instance_id}\", "vnf_template_id": "${vnf_template_id}\",
"additional_parameters": "${additional_parameters}\",
"execution_order": "parallel", "restart": "${restart}" }"

curl -s -X POST -H "Content-Type: application/json" \
-d "$body" \
"${OVLME_BASEURL}/api/v2/tenants/${tenant_id}/vnfs/${vnf_id}/vnf_
instances/${vnf_instance_id}/vnfc_instances/workflow_instances?syn
c=1&timeout=2000"

```

The following is an example of a successful response.

```
{
  "data": {
    "flow_result": {
      "success": "\"VNFC Update passed successfully on all servers.\""
    },
    "method": "POST",
    "request": "http://10.3.18.22:28085/api/v2/tenants/defa
ult/vnfs/microblog/vnf_instances/dc_1/vnfc_instances/workflow_instances",
    "workflow_instance_status": "COMPLETED"
  },
  "status": 200
}
```

As a result of the successful call, instance 2 of the blogserver VNFC is updated to template2 while the other VNFCs remain on template1 as shown.

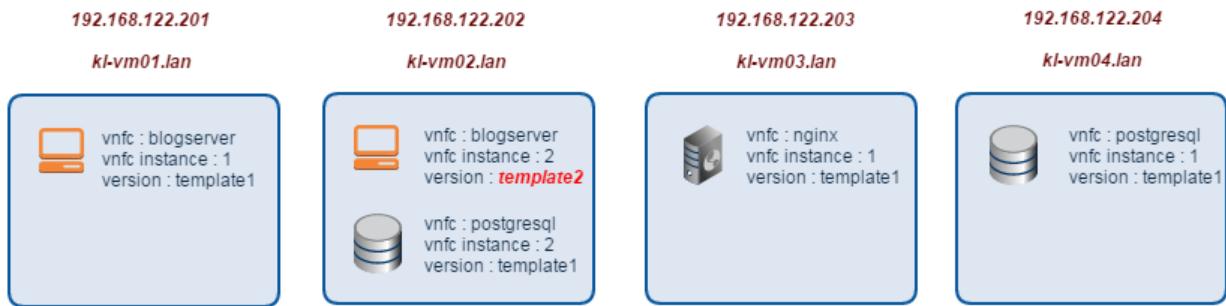


Figure 112: Microblog VNF After partial Update

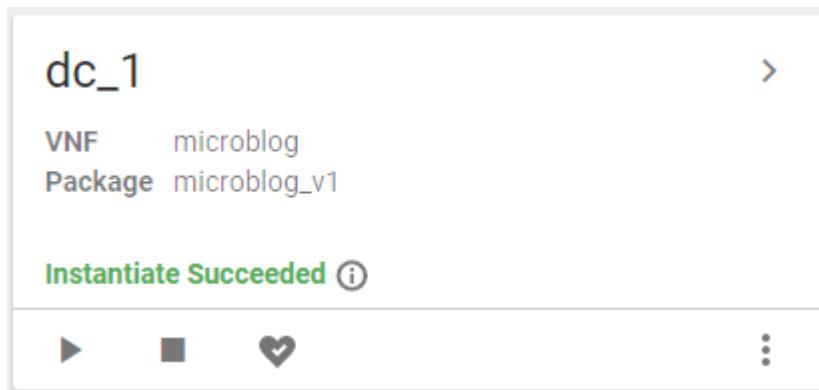
Updating VNFCs Using the VNF-M GUI

You can use the VNF-M GUI to update a VNFC, which updates all instances of the VNFC, or to update a specific VNFC instance.

Updating All Instances of a VNFC

To update all instances of a VNFC, follow these steps

1. On the VNF Lifecycle page, locate the active VNF with the VNFC you intend to update and click the arrow in the top-right corner of the VNF instance card.



Scroll down the page to find the VNF if necessary.

Note: Active VNFs are grouped at the bottom of the VNF Lifecycle page.

Figure 113: Active VNF

The VNFC Page for the selected VNF is displayed with a card for each VNFC.

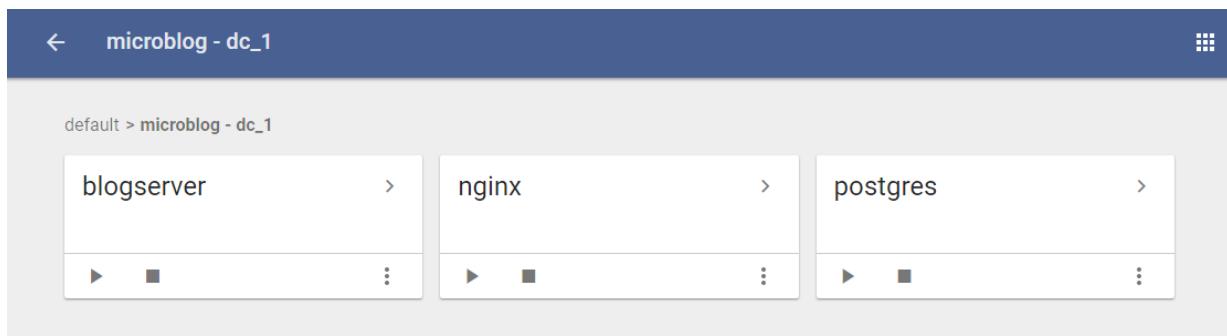


Figure 114: VNFC Page

2. Click the more options icon



and select **Update** on the VNFC for which you intend to update all instances, which in this example is blogserver. The update VNFC dialog box is displayed.

Update VNFC

⋮

Select the execution order Sequential Parallel

The following VNFC Instances will be updated :

- "blogserver 1" has package "microblog_v1" installed.
- "blogserver 2" has package "microblog_v1" installed.

Select the package to update the VNFC "blogserver" to.

Name ↑	Creation Date
<input type="checkbox"/> microblog_v1	May 5, 2017 11:21:39 AM
<input type="checkbox"/> microblog_v2	May 5, 2017 11:21:39 AM
<input type="checkbox"/> microblog_v3	May 5, 2017 11:21:39 AM

CANCEL
UPDATE

Figure 115: update VNFC Dialog Box

3. In the **Select the execution order** field, click a radio button to specify whether the update will be made to each VNFC instance in series or in parallel.
4. Select a VNF package for the update by checking it, for example microblog_v2.
5. To configure additional parameters for a script you are running on the Resource Agent related to updating the VNFC, click the icon in the top-right of the confirmation box and select **Add parameters**.

The dialog box displays a field into which you can enter additional parameters. In some cases there might be predefined parameters, which will display when you select **Add Parameters**. Predefined parameters can be overwritten.

Note: For information about how to predefine Additional Parameters, see [Managing Additional Parameters for the VNF-M GUI](#).

Parameters must be entered in JSON format as in the following example.

Update VNFC

Select the execution order Sequential Parallel

The following VNFC Instances will be updated :

- "blogserver 1" has package "microblog_v1" installed.
- "blogserver 2" has package "microblog_v1" installed.

Select the package to update the VNFC "blogserver" to.

Name ↑	Creation Date
<input type="checkbox"/> microblog_v1	May 5, 2017 11:21:39 AM
<input checked="" type="checkbox"/> microblog_v2	May 5, 2017 11:21:39 AM
<input type="checkbox"/> microblog_v3	May 5, 2017 11:21:39 AM

Additional Parameters

```
{
  "take_a_snapshot": true
}
```

CANCEL UPDATE

Figure 116: Additional Parameters

Note: To remove additional parameters after adding them, click the top-right icon again and select **Remove parameters**.

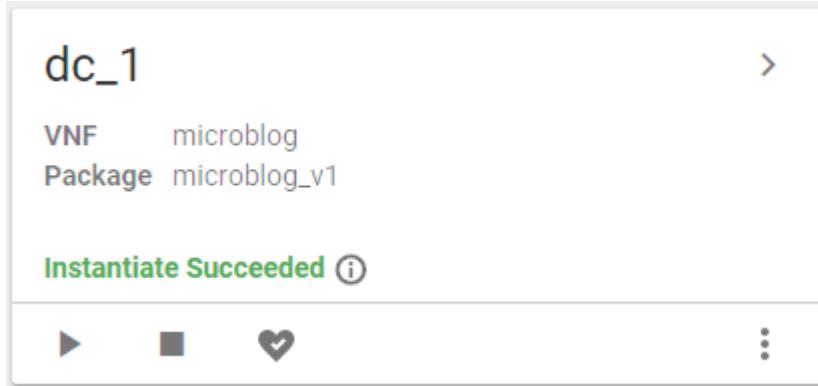
6. Click **update.**

The VNFC instances are started with a status of update Succeeded.

Updating a Specific Instance of a VNFC

To update a specific VNFC instance, follow these steps:

1. On the VNF Lifecycle page, locate the active VNF with the VNFC you intend to update and click the arrow in the top-right corner of the VNF instance card.



Scroll down the page to find the VNF if necessary.

Note: Active VNFs are grouped at the bottom of the VNF Lifecycle page.

Figure 117: Active VNF

The VNFC Page for the selected VNF is displayed with a card for each VNFC.

Figure 118: VNFC Page

2. Click the arrow in the top-right corner of the blogserver VNFC card.
All instances of the blogserver VNFC are displayed.

Figure 119: Blogserver VNFC Instances

3. Click the more options icon



and select **Update** on the VNFC you intend to update.
The update VNFC Instance dialog box is displayed.

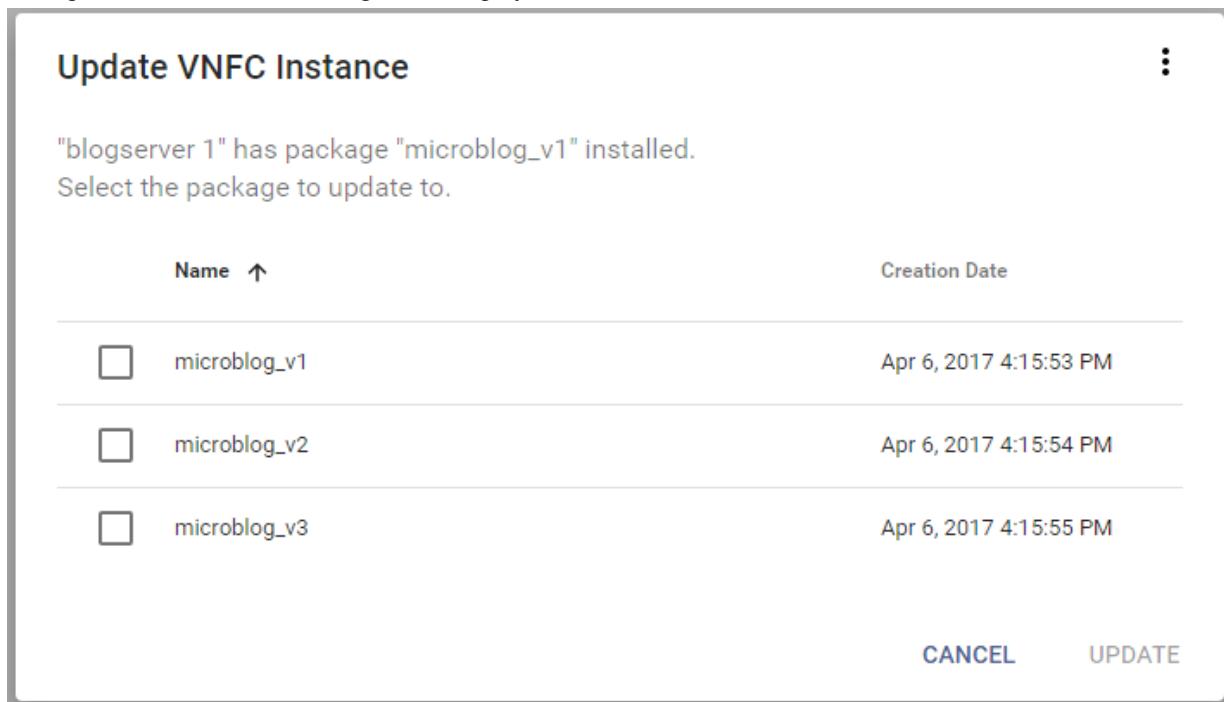


Figure 120: update VNFC Instance Dialog Box

4. Select a VNF package for the update by checking it, for example `microblog_v2`.
5. To configure additional parameters for a script you are running on the Resource Agent related to updating the VNFC, click the icon in the top-right of the dialog box card and select **Add parameters**.

The dialog box displays a field into which you can enter additional parameters. In some cases there might be predefined parameters, which will display when you select **Add Parameters**. Predefined parameters can be overwritten.

Note: For information about how to predefine Additional Parameters, see [Managing Additional Parameters for the VNF-M GUI](#).

Parameters must be entered in JSON format as in the following example.

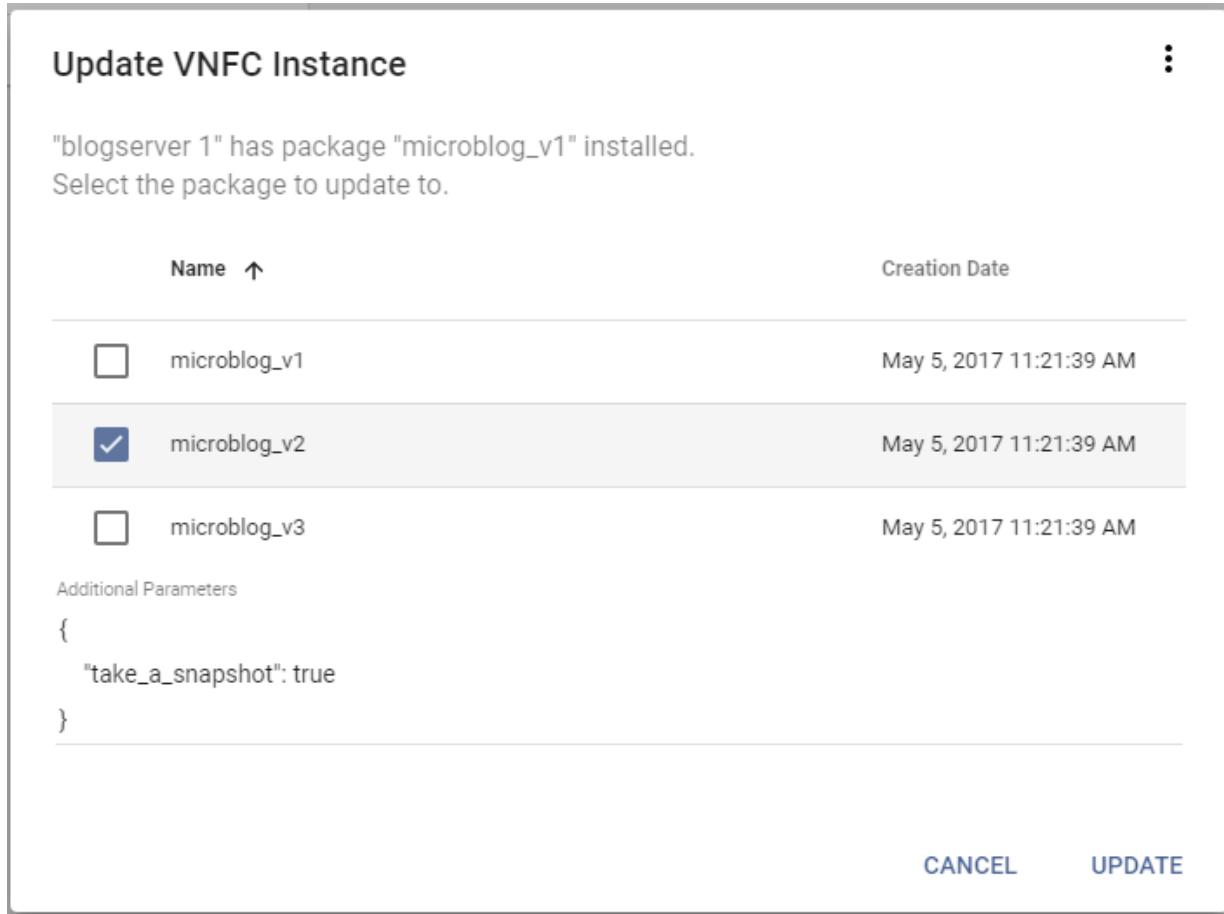


Figure 121: Additional Parameters

Note: To remove additional parameters after adding them, click the top-right icon again and select **Remove parameters**.

6. Click **update**.

The VNFC instance is updated with a status of update Succeeded.

Rolling Back VNFCs

You can roll back one or more VNFCs in a VNF to the version prior to an upgrade by using the `rollback_vnfc` workflow. You can run the rollback workflow against all VNFCs of the same type or a specific VNFC instance. This rollback workflow can be run only on a VNF instance where the VNFC was previously upgraded. A VNF can be rolled back only to the version just prior to the most recent upgrade.

You can roll back VNFCs from the CLI, the API, or from the Openet Weaver VNF-M GUI.

Important: The `rollback_vnfc` workflow can be used to roll back only those VNFCs that have been upgraded using the `upgrade_vnfc` workflow.

The workflow reverts all changes introduced when the most recent `upgrade_vnfc` workflow was run against the VNF.

The steps that are run in the workflow depend on the result of the most recent `upgrade_vnfc` workflow run against the VNFC.

The `rollback_vnf` workflow runs all the step in the following circumstances:

- The most recent upgrade ran successfully
- An error occurred during upgrade *after* the `update_config_vnfc` step when running the `upgrade_vnf` workflow.

- The VNFC was rolled back previously with a failure.

The rollback_vnf workflow will run only is_vnfc_up and continue with start_vnfc step (if the VNFC is not running) in the following circumstances:

- An error occurred *before* or *during* the update_config_vnfc step during when running the upgrade_vnf workflow.
- The VNFC was successfully rolled back previously.

There might be other reasons that cause rollback failure depending on the complexity of the VNF deployment. If rollback fails, you might need to perform self-troubleshooting based on Resource Agent and procedures logs in order to apply proper recovery.

In the unlikely event that you require more information to troubleshoot the problem, you can try running the rollback_vnfc workflow through the Front End CLI, which might generate a different error message.

Important: The rollback_vnfc workflow will not work if you ran the update flow against the VNF since the last upgrade.

Rolling Back a VNFC from the CLI

To roll back a VNFC from the CLI, use the rollback_vnfc workflow_type parameter in the vnfc_workflow submit command as shown below.

```
ovlm-fe vnfc_workflow submit -t <tenant_id> -v <vnfd_id> -i <vnfc_instance_id>
-y rollback_vnfc -c <vnfc_id> [-n <vnfc_instance_id>] [-restart <true | false>]
[-sync <sync>]
[-timeout <seconds>] [-o <execution_order>] [-a '{<additional_parameter1,
additional_parameter2,...>}'] [-continue_on_failure <true|false>]
```

Note: All parameters are described on the main page for this section, [Running VNFC Workflows](#).

The command rolls back all VNFC instances with the specified vnfc_id. When the optional vnfc_instance_id parameter is also used in the command, only that VNFC instance is rolled back.

This workflow validates the presence of the VNFC Rollback directory for the VNFC. By default, the workflow fails if the VNFC rollback directory is missing. However, you can override this behavior by setting the continue_on_failure parameter to true.

Example: Rolling Back all VNFCs with the Same vnfc_id

This example shows a partial VNF rollback where all VNFCs with the same vnfc_id within in an instance of a microblog VNF named dc_1.

The following CLI command rolls back all VNFCs with the same vnfd_id (blogserver) within dc_1.

```
ovlm-fe vnfc_workflow submit -y rollback_vnfc -t Default -v microblog -i dc_1
-c
blogserver
```

Note: Be sure the baseurl environment variables are set before using CLI commands.

The following is an example of a successful response to running the command.

```
data:
  workflow_instance_status: COMPLETED
  success_msg: Rollback of VNFC has completed.
  flow_result:
    health_status: UP
    vnfc_instances:
      - vnfc_id: blogserver
        vnfc_instance_id: '1'
        hostname: 192.168.122.216
```

```

exit_code: '0'
response:
  health_status: UP
  output:
    - Rollback has completed.
- vnfc_id: blogserver
  vnfc_instance_id: '2'
  hostname: 192.168.122.74
  exit_code: '0'
  response:
    health_status: UP
    output:
      - Rollback has completed.
- vnfc_id: nginx
  vnfc_instance_id: '1'
  hostname: 192.168.122.118
  exit_code: ''
  response:
    health_status: UP
    output:
      - Rollback is not performed.
- vnfc_id: postgres
  vnfc_instance_id: '1'
  hostname: 192.168.122.29
  exit_code: ''
  response:
    health_status: UP
    output:
      - Rollback is not performed.
request: http://localhost:28085/api/v2/tenants/default/vnfs/microblog/vnf_instances/dc_1/vnfc_instances/workflow_instances
method: POST
status: 200

```

Example: Rolling Back a Specific VNFC Instance

This example shows a partial VNF rollback of a specific VNFC within in an instance of a microblog VNF named dc_1.

The following CLI command example rolls back a specific instance (2) of the blogserver VNFC in an instance of the microblog VNF.

```
ovlm-fe vnfc_workflow submit -y rollback_vnfc -t Default -v microblog -i dc_1
-c blogserver -n 1
```

The following is an example of a successful response to running the command.

```

data:
  workflow_instance_status: COMPLETED
  success_msg: Rollback of VNFC has completed.
  flow_result:
    health_status: UP
    vnfc_instances:
      - vnfc_id: blogserver
        vnfc_instance_id: '1'
        hostname: 192.168.122.216
        exit_code: '0'
        response:
          health_status: UP
          output:
            - Rollback has completed.
- vnfc_id: blogserver

```

```

vnfc_instance_id: '2'
hostname: 192.168.122.74
exit_code: ''
response:
  health_status: UP
  output:
    - Rollback is not performed.
- vnfc_id: nginx
  vnfc_instance_id: '1'
  hostname: 192.168.122.118
  exit_code: ''
  response:
    health_status: UP
    output:
      - Rollback is not performed.
- vnfc_id: postgres
  vnfc_instance_id: '1'
  hostname: 192.168.122.29
  exit_code: ''
  response:
    health_status: UP
    output:
      - Rollback is not performed.
request: http://localhost:28085/api/v2/tenants/default/vnfs/microblog/vnf_instances/dc_1/vnfc_instances/workflow_instances
method: POST
status: 200

```

As a result of the successful call, instance 2 of the blogserver VNFC is upgraded to template2 while the other VNFCs remain on template1 as shown.

Rolling Back a VNFC Using REST API

To perform a partial VNF roll back using REST API, you can roll back specific VNFCs by posting a workflow request using the rollback_vnfc workflow_type parameter. The API request takes the following format.

```

POST http://<ovlm-workflow-fe-url>/api/v2/tenants/<tenant_id>/vnfs/<vnf_id>/vnfc_instances/<vnfc_instance_id>/vnfc_instances/workflow_instances?sync=1&timeout=2000
{
  "workflow_type": "rollback_vnfc",
  "vnfc_id": <vnfc_id>,
  ["vnfc_instance_id": <vnfc_instance_id>],
  ["additional_parameters": <additional_parameters>]
}

```

Note: All parameters are described on the main page for this section, [Running VNFC Workflows](#).

The vnfc_instance_id parameter is optional. When you do not use it, all of the VNFCs with the same vnfc_id within a VNF are rolled back. When you do use the parameter, only the VNFC with the specified instance ID is rolled back.

Example: Rolling Back all VNFCs with the Same vnfc_id

In this example of a partial VNF rollback, both VNFCs with the same vnfc_id (blogserver) are rolled back in a microblog VNF instance. The VNF instance name is dc_1. The API call to roll back the blogserver VNFCs using curl is shown below. In this case, vnfc_instance_id is not used in the call.

```

./rollback_vnfc_type.sh default microblog dc_1 blogserver '{ \"passedKey1\": \"passedKeyValue1\" }'
http://10.3.18.22:28085
- rollback vnfc type -

```

```

. tenant_id           = default
. vnf_id              = microblog
. vnf_instance_id     = dc_1
. vnfc_id              = blogserver

===== sample script =====

#!/bin/bash
OVLMFE_BASEURL=http://10.3.18.22:28085
echo $OVLMFE_BASEURL
if [ $# -lt 4 ]; then
    echo "usage : $0 <tenant_id> <vnf_id> <vnf_instance> <vnfc_id>
[<additional_parameters>]"
    exit 1
fi
tenant_id=$1
vnf_id=$2
vnf_instance_id=$3
vnfc_id=$4
additional_parameters=$5
echo " - rollback vnfc type - "
echo ". tenant_id          = $tenant_id"
echo ". vnf_id              = $vnf_id"
echo ". vnf_instance_id     = $vnf_instance_id"
echo ". vnfc_id              = $vnfc_id"
body={"\\"workflow_type\\": \\"rollback_vnfc\\", \\"vnfc_id\\": \"${vnfc_id}\", \
\\"additional_parameters\\": \"${additional_parameters}\", \\"execution_order\\": \
\\"parallel\\\" }
curl -s -X POST -H "Content-Type: application/json" \
-d "$body" \
"${OVLMFE_BASEURL}/api/v2/tenants/${tenant_id}/vnfs/${vnf_id}/vnf_\
instances/${vnf_instance_id}/vnfc_instances/workflow_instances?syn\
c=1&timeout=2000" | python -mjson.tool | pygmentize -l json

```

The following is an example of a successful response.

```
{
  "data": {
    "workflow_instance_status": "COMPLETED",
    "success_msg": "Rollback of VNFC has completed.",
    "flow_result": [
      {
        "health_status": "UP",
        "vnfc_instances": [
          {
            "vnfc_id": "blogserver",
            "vnfc_instance_id": "1",
            "hostname": "192.168.122.216",
            "exit_code": "0",
            "response": {
              "health_status": "UP",
              "output": ["Rollback has completed."]
            }
          },
          {
            "vnfc_id": "blogserver",
            "vnfc_instance_id": "2",
            "hostname": "192.168.122.74",
            "exit_code": "0",
            "response": {
              "health_status": "UP",
              "output": ["Rollback has completed."]
            }
          }
        ]
      }
    ]
  }
}
```

```

        "vnfc_id": "nginx",
        "vnfc_instance_id": "1",
        "hostname": "192.168.122.118",
        "exit_code": "",
        "response": {
            "health_status": "UP",
            "output": ["Rollback is not performed."]
        }
    },
    {
        "vnfc_id": "postgres",
        "vnfc_instance_id": "1",
        "hostname": "192.168.122.29",
        "exit_code": "",
        "response": {
            "health_status": "UP",
            "output": ["Rollback is not performed."]
        }
    }
],
{
    "request": "http://localhost:28085/api/v2/tenants/default/vnfs/microblog/vnf_instances/dc_1/vnfc_instances/workflow_instances",
    "method": "POST"
},
"status": 200
}

```

Example: Rolling Back a specific VNFC

In this example of a partial VNF rollback, a specific VNFC instance of the blogserver VNFC within the microblog VNF is rolled back. The VNFC instance name is 1. The API call to roll back the blogserver VNFCs using curl is shown below. In this case, vnfc_instance_id is used to target a specific instance of the blogserver VNFC .

```

./rollback_vnfc_instance.sh default microblog dc_1 blogserver 1 '{
\"passedKey1\": \"passedKeyValue1\" }'
http://10.3.18.22:28085
- rollback vnfc instance -
. tenant_id           = default
. vnf_id               = microblog
. vnf_instance_id     = dc_1
. vnfc_id              = blogserver
. vnfc_instance_id    = 1
. additional_parameters = { \"passedKey1\": \"passedKeyValue1\" }

=====
sample script =====

#!/bin/bash
OVLMFE_BASEURL=http://10.3.18.22:28085
echo $OVLMFE_BASEURL
if [ $# -lt 5 ]; then
    echo "usage : $0 <tenant_id> <vnf_id> <vnf_instance> <vnfc_id>
<vnfc_instance_id>[<additional_parameters>]"
    exit 1
fi
tenant_id=$1
vnf_id=$2
vnf_instance_id=$3
vnfc_id=$4
vnfc_instance_id=$5

```

```

additional_parameters=$6
echo " - rollback vnfc instance - "
echo ". tenant_id = $tenant_id"
echo ". vnf_id = $vnfc_id"
echo ". vnf_instance_id = $vnfc_instance_id"
echo ". vnfc_id = $vnfc_id"
echo ". vnfc_instance_id = $vnfc_instance_id"
echo ". additional_parameters = $additional_parameters"
body={"\\"workflow_type\": \"rollback_vnfc\", \\"vnfc_id\": \"$vnfc_id\", \\"vnfc_instance_id\": \"$vnfc_instance_id\", \\"additional_parameters\": \"$additional_parameters\" }"
curl -s -X POST -H "Content-Type: application/json" \
-d "$body" \
"${OVLME_BASEURL}/api/v2/tenants/${tenant_id}/vnfs/${vnf_id}/vnf_instances/${vnf_instance_id}/vnfc_instances/workflow_instances?sync=1&timeout=2000" | python -mjson.tool | pygmentize -l json

```

The following is an example of a successful response.

```
{
  "data": {
    "workflow_instance_status": "COMPLETED",
    "success_msg": "Rollback of VNFC has completed.",
    "flow_result": [
      {
        "health_status": "UP",
        "vnfc_instances": [
          {
            "vnfc_id": "blogserver",
            "vnfc_instance_id": "1",
            "hostname": "192.168.122.216",
            "exit_code": "0",
            "response": {
              "health_status": "UP",
              "output": ["Rollback has completed."]
            }
          },
          {
            "vnfc_id": "blogserver",
            "vnfc_instance_id": "2",
            "hostname": "192.168.122.74",
            "exit_code": "",
            "response": {
              "health_status": "UP",
              "output": ["Rollback is not performed."]
            }
          },
          {
            "vnfc_id": "nginx",
            "vnfc_instance_id": "1",
            "hostname": "192.168.122.118",
            "exit_code": "",
            "response": {
              "health_status": "UP",
              "output": ["Rollback is not performed."]
            }
          },
          {
            "vnfc_id": "postgres",
            "vnfc_instance_id": "1",
            "hostname": "192.168.122.29",
            "exit_code": "",
            "response": {
              "health_status": "UP",
              "output": ["Rollback is not performed."]
            }
          }
        ]
      }
    ]
  }
}
```

```

        ],
      },
      "request": "http://localhost:28085/api/v2/tenants/default/vnfs/microblog/vnf_instances/dc_1/vnfc_instances/workflow_instances",
      "method": "POST"
    },
    "status": 200
}

```

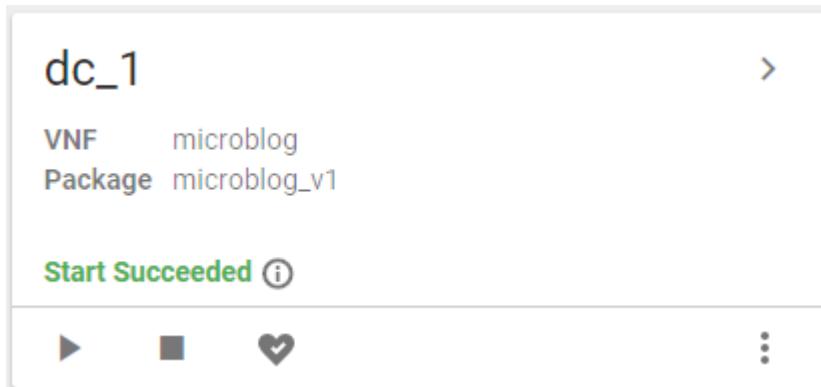
Rolling Back VNFCs Using the VNF-M GUI

You can use the VNF-M GUI to roll back a VNFC to the version prior to the VNFC upgrade, which rolls back all instances of the VNFC. Alternatively, you can roll back a specific VNFC instance.

Rolling Back All Instances of a VNFC

To roll back all instances of a VNFC to the version prior to the VNFC upgrade, follow these steps

1. On the VNF Lifecycle page, locate the active VNF with the VNFC to roll back and click the arrow in the top-right corner of the VNF instance card.



Scroll down the page to find the VNF if necessary.

Note: Active VNFs are grouped at the bottom of the VNF Lifecycle page.

Figure 122: Active VNF

The VNFC Page for the selected VNF is displayed with a card for each VNFC.

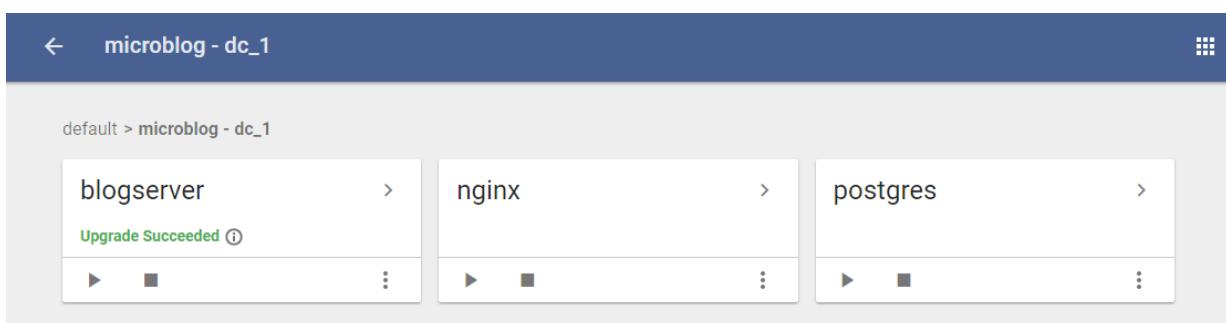


Figure 123: VNFC Page

2. Click the more options icon



and select **Rollback** on the VNFC for which you intend to roll back all instances, which in this example is blogserver. The Rollback VNFC confirmation box is displayed.



Figure 124: Rollback VNFC Confirmation Box

- In the **Select the execution order** field, click a radio button to specify whether the update will be made to each VNFC instance in series or in parallel.
- To configure additional parameters for a script you are running on the Resource Agent related to rolling back the VNFC, click the icon in the top-right of the confirmation box and select **Add parameters**.

The dialog box displays a field into which you can enter additional parameters. In some cases there might be predefined parameters, which will display when you select **Add Parameters**. Predefined parameters can be overwritten.

Note: For information about how to predefine Additional Parameters, see [Managing Additional Parameters for the VNF-M GUI](#).

Parameters must be entered in JSON format as in the following example.

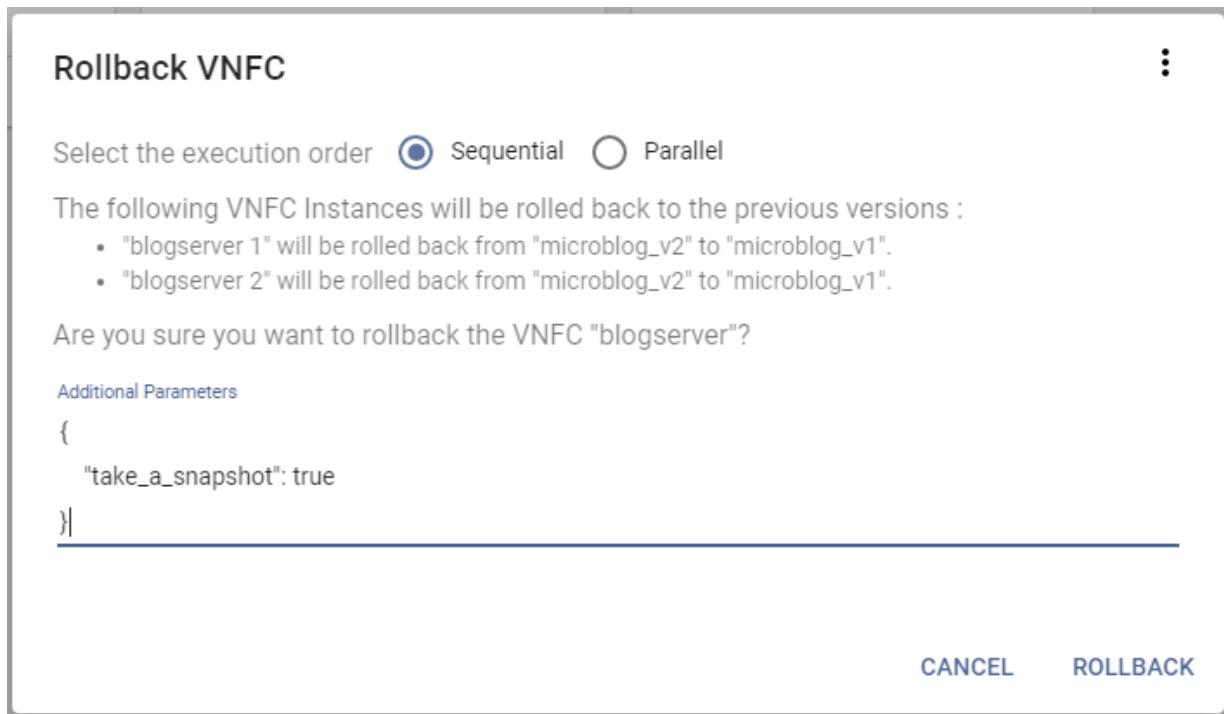


Figure 125: Additional Parameters

Note: To remove additional parameters after adding them, click the top-right icon again and select **Remove parameters**.

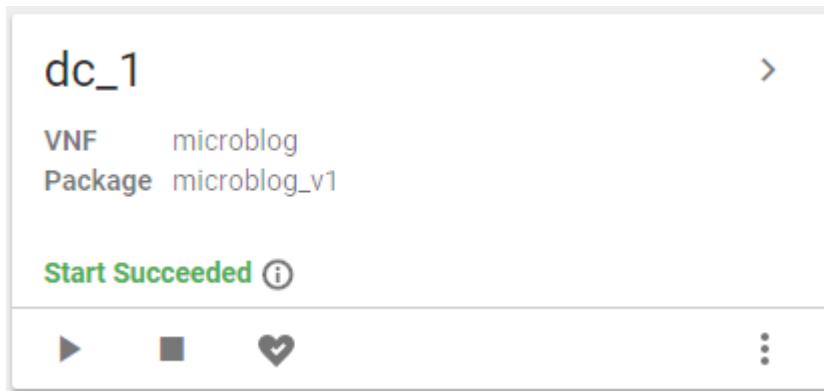
5. Click **Rollback**.

The VNFC instances are rolled back with a status of Rollback Succeeded.

Rolling Back a Specific Instance of a VNFC

To roll back a specific VNFC instance to the version prior to the VNFC upgrade, follow these steps:

1. On the VNF Lifecycle page, locate the active VNF with the VNFC to roll back and click the arrow in the top-right corner of the VNF instance card.



Scroll down the page to find the VNF if necessary.

Note: Active VNFs are grouped at the bottom of the VNF Lifecycle page.

Figure 126: Active VNF

The VNFC Page for the selected VNF is displayed with a card for each VNFC.

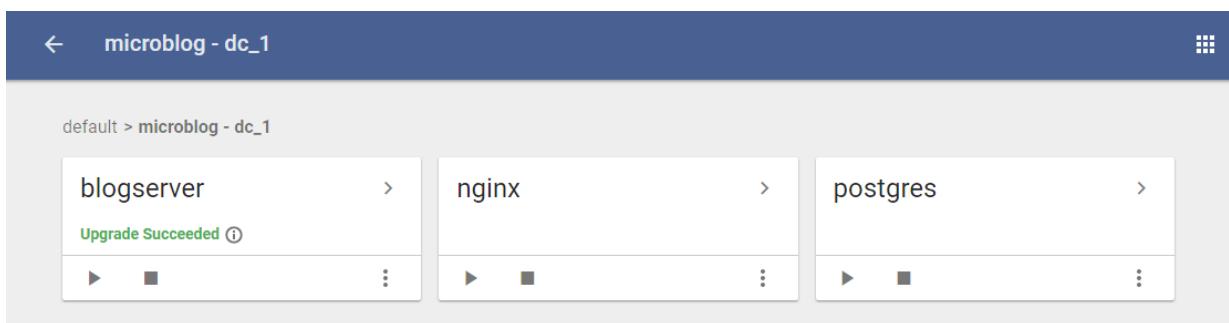


Figure 127: VNFC Page

2. Click the arrow in the top-right corner of the blogserver VNFC card.
All instances of the blogserver VNFC are displayed.

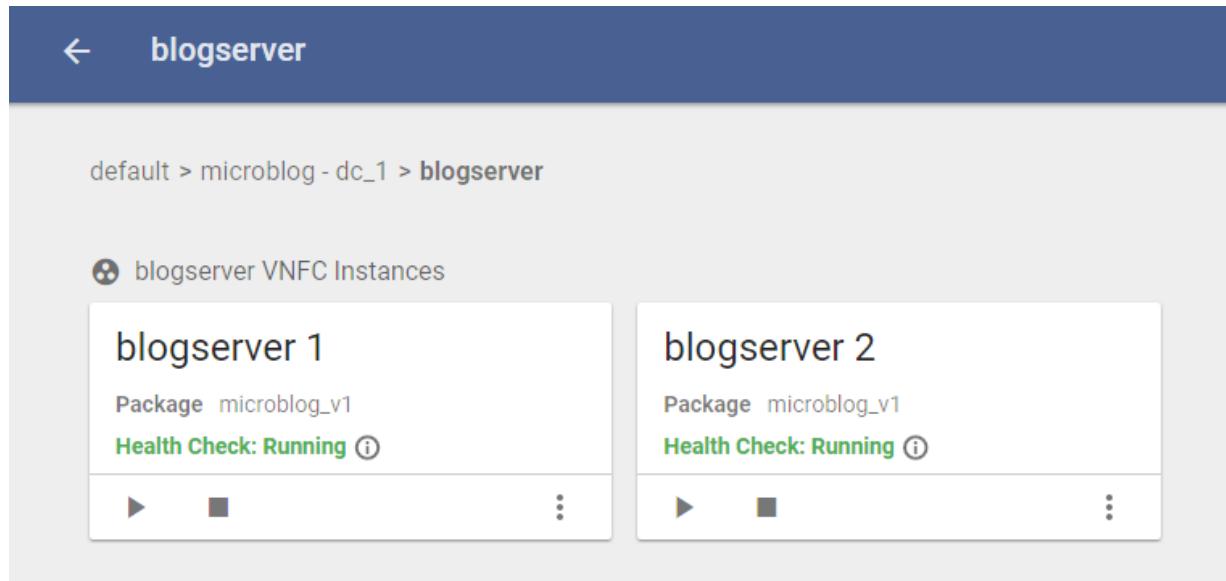


Figure 128: Blogserver VNFC Instances

- Click the more options icon



and select **Rollback** on the VNFC to roll back, which in this case is blogserver 1. The Rollback VNFC Instance confirmation box is displayed.

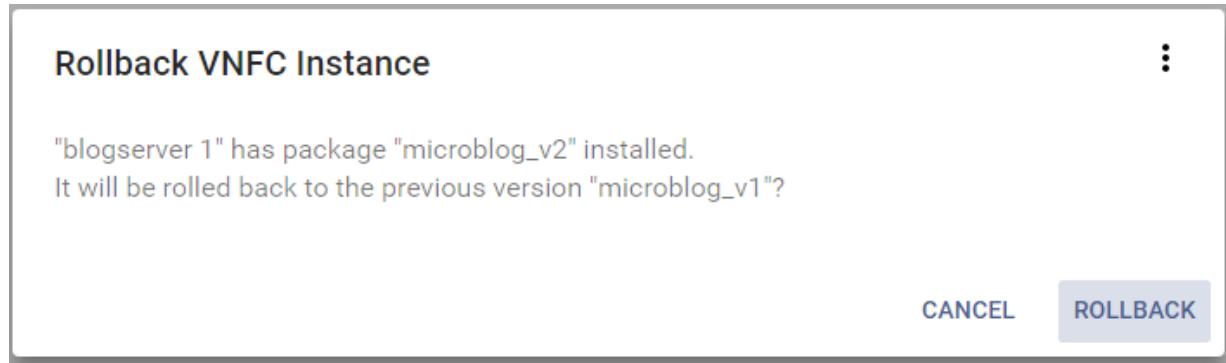


Figure 129: Rollback VNFC Instance Confirmation Box

- To configure additional parameters for a script you are running on the Resource Agent related to rolling back the VNFC, click the icon in the top-right of the confirmation box and select **Add parameters**.

The dialog box displays a field into which you can enter additional parameters. In some cases there might be predefined parameters, which will display when you select **Add Parameters**. Predefined parameters can be overwritten.

Note: For information about how to predefine Additional Parameters, see [Managing Additional Parameters for the VNF-M GUI](#).

Parameters must be entered in JSON format as in the following example.



Figure 130: Additional Parameters

Note: To remove additional parameters after adding them, click the top-right icon again and select **Remove parameters**.

5. Click **Rollback**.
The VNFC instance is rolled back with a status of Rollback Succeeded.

Healing a VNFC

To run Healing VNFC flow, use the heal_vnfc workflow_type parameter.

You can run the heal_vnfc workflow from the CLI, the API, or from the Openet Weaver VNF-M GUI.

This workflow can be run only on VNFs that are already instantiated.

The heal_vnfc workflow is shown below.

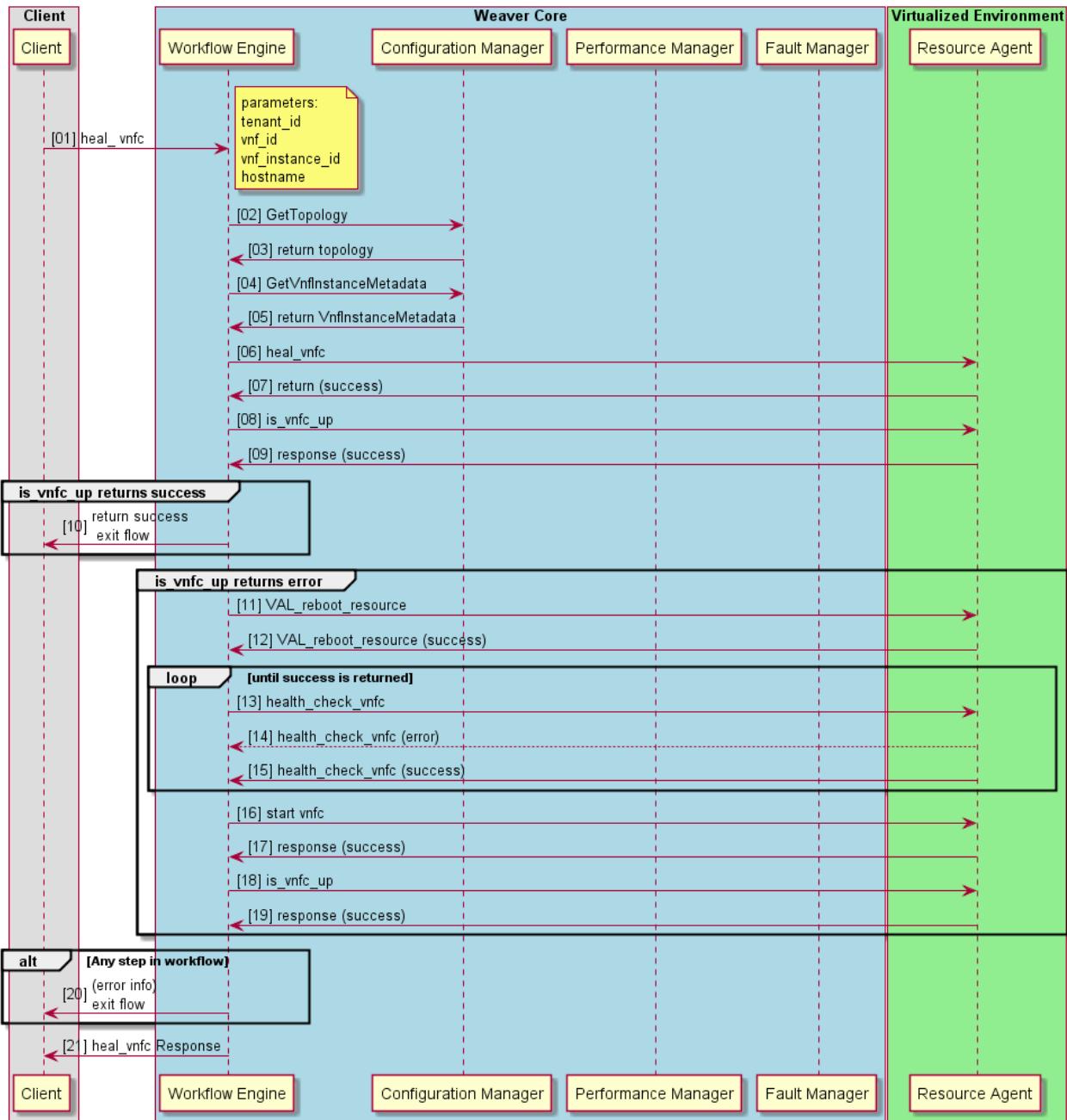


Figure 131: Healing VNFC Flow

Related concepts

[Setting Environment Variables](#) on page 133

[Openet Weaver VNF-M API Reference](#) on page 468

Openet Weaver REST API reference guides

[Openet Weaver VNF-M CLI Reference](#) on page 469

Openet Weaver CLI command reference guides

Healing a VNFC Instance from the CLI

To run the healing workflow from the CLI, run the workflow submit command using the heal_vnfc workflow as shown below.

```
ovlm-fe vnfc_workflow submit -t <tenant_id> -v <vnf_id> -i <vnf_instance_id>
-y heal_vnfc[-restart <true | false>]
[-hostname <host_name>] [-sync <sync>] [-timeout <seconds>] [-a
'<additional_parameter1, additional_parameter2,...>']
```

Note: All parameters are described on the main page for this section, [Running VNFC Workflows](#).

The command in the following example heals a specified host of a VNF instance (Resource Agent) that has failed or has stopped. In this example the command parameters are the default tenant, the microblog vnf, a vnf_instance_id of dc_1 and a hostname of node-1.

```
ovlm-fe workflow submit -t default -v microblog -i dc_1 -y heal_vnfc -hostname
node-1
```

Note: Be sure the baseurl environment variables are set before using CLI commands.

The output of the command is a workflow_instance_id generated by the system similar to the following example.

```
data:
  method: POST
  request: http://ovlm-vm:8080/api/v2/tenants/default/v
nfs/microblog/vnf_instances/dc_1/workflow_instances
    workflow_instance_id: '142600153'
  status: 201
```

Healing a VNFC Using REST API

To heal a VNF using REST API, post a workflow request using the heal_vnfc workflow_type parameter. The API request takes the following format.

```
POST http://<ovlm-workflow-fe-url>
/api/v2/tenants/<tenant_id>/vnfs/<vnf_id>/vnf_instances/<vnf_instance_id>/workflow_instances?sy
nc=1
{
  "workflow_type": "heal_vnfc",
  "hostname": <hostname indicated in vnf_topology.yml>
}
```

Note: All parameters are described on the main page for this section, [Running VNFC Workflows](#).

The following is an example of running the heal_vnfc workflow using curl. The request specifies that the workflow run in synchronous mode.

```
OVLMFE_BASEURL=http://10.3.18.22:28085
tenant_id=default
vnf_id=microblog
vnf_instance_id=dc_1
echo " - heal - "
curl -i \
-H "Content-Type: application/json" \
-X POST \
-d '{"workflow_type": "heal_vnfc", "hostname": "192.168.122.180"}' \
$OVLMFE_BASEURL/api/v2/tenants/${tenant_id}/vnfs/${vnf_id}/vnf_i
nstances/${vnf_instance_id}/workflow_instances?sync=1
```

The following is an example response to the API call.

```

HTTP / 1.1 200 OK
Date: Mon, 20 Feb 2017 11: 01: 41 GMT
X - Application - Context: application: 28085
Content - Type: application / json;
charset = UTF - 8
Transfer - Encoding: chunked
Server: Jetty(9.2.14.v20151106)
{
  "data": {
    "workflow_instance_status": "COMPLETED",
    "flow_result": {
      "success": "\"VNFC Healing passed successfully on the VNFC with a
hostname=192.168.122.180.\\""
    },
    "request": "http://10.3.18.22:28085/api/v2/tenants/defa
ult/vnfs/microblog/vnf_instances/dc_1/workflow_instances",
    "method": "POST"
  },
  "status": 200
}

```

Healing a VNFC Using the VNF-M GUI

To heal a VNF that is not configured for VIM, follow these steps:

1. On the VNF Lifecycle page, locate the active VNF to heal, click the more options icon



and select **Heal**. Scroll down the page if required.

Note: An active VNF is any VNF that is not inactive, regardless of status.

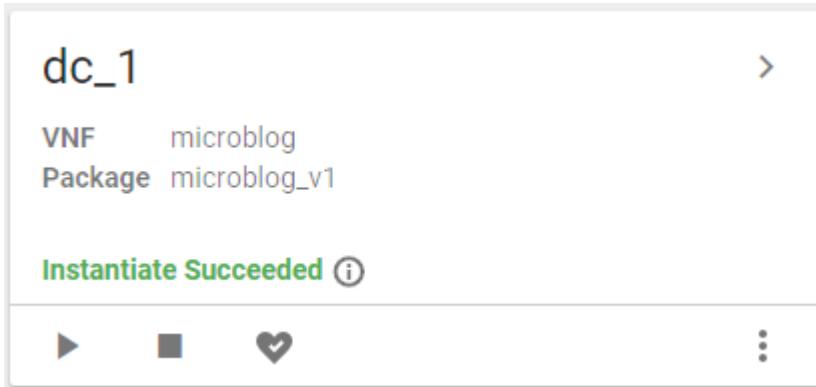


Figure 132: Active VNF

The Select a VNFC Instance to Heal dialog box is displayed.

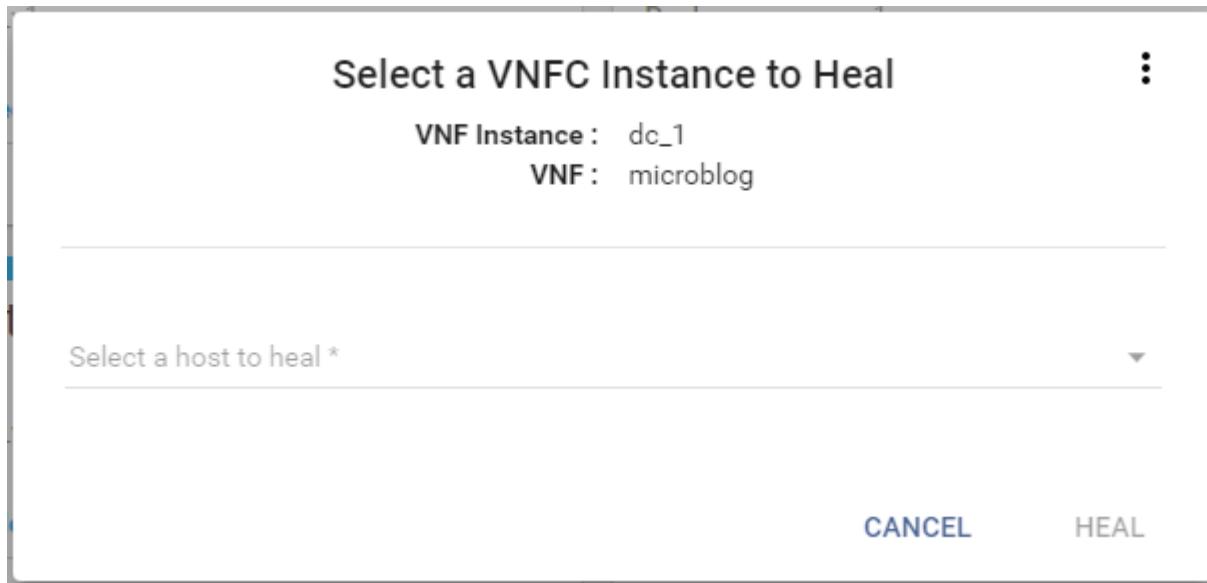


Figure 133: Select a VNFC Instance to Heal Dialog Box

2. Select a host from the **Select a host to heal** drop-down box.
3. To configure additional parameters for a script you are running on the Resource Agent related to healing the VNFC, click the icon in the top-right of the dialog box and select **Add parameters**.

The dialog box displays a field into which you can enter additional parameters. In some cases there might be predefined parameters, which will display when you select **Add Parameters**. Predefined parameters can be overwritten.

Note: For information about how to predefine Additional Parameters, see [Managing Additional Parameters for the VNF-M GUI](#).

Parameters must be entered in JSON format as in the following example.

```
{
    "take_a_snapshot": true
}
```

Note: To remove additional parameters after adding them, click the top-right icon again and select **Remove parameters**.

4. Click **Heal**.

The dialog box closes, the healing process begins, and the status changes from green to blue with the text "Heal In-Progress".

Note: In the event that the healing process fails the VNF card is moved to the Alerts section and the status turns orange with the text "Heal Failed".

Starting VNFCs

To start one or more VNFC instances, use the `start_vnfc` workflow_type parameter. You can run the workflow against all VNFCs of the same type or a specific VNFC instance.

You can start VNFC instance from the CLI, the API, or from the Openet Weaver VNF-M GUI.

Starting VNFCs from the CLI

To perform a partial start of a VNF, you can start specific VNFCs using `start_vnfc` as the `workflow_type` parameter with the `vnfc_workflow submit` command shown below.

```
ovlm-fe vnfc_workflow submit -t <tenant_id> -v <vnf_id> -i <vnf_instance_id>
-y start_vnfc -c <vnfc_id> [-n <vnfc_instance_id>] [-sync <sync>]
[-timeout <seconds>] [-a '{<additional_parameter1>, additional_parameter2, ...}' ]
```

Note: All parameters are described on the main page for this section, [Running VNFC Workflows](#).

The command starts all VNFC instances with the specified `vnfc_id`. When the optional `vnfc_instance_id` parameter is also used in the command, only that VNFC instance is started.

Example: Starting all VNFCs with the Same `vnfc_id`

In this example of a partial VNF start, the command starts all VNFCs with the same `vnfc_id` (`blogserver`) within in an instance of a microblog VNF named `dc_1`.

```
ovlm-fe vnfc_workflow submit -y start_vnfc -t Default -v microblog -i dc_1 -c
blogserver
```

Note: Be sure the `baseurl` environment variables are set before using CLI commands.

The following is an example of a successful response to running the command.

```
data:
method: POST
  request: http://10.3.18.22:28085/api/v2/tenants/default/vnfs/microblog/vnf_
instances/dc_1/vnfc_instances/workflow_instances
    workflow_instance_id: '112602008'
status: 201
```

Example: Start a Specific VNFC Instance

In this example of a partial VNF start, the command starts a specific VNFC within in an instance (2) of a microblog VNF named `dc_1`.

```
ovlm-fe vnfc_workflow submit -y start_vnfc -t Default -v microblog -i dc_1 -c
blogserver -n 1
blogserver
```

The following is an example of a successful response to running the command.

```
data:
method: POST
  request: http://10.3.18.22:28085/api/v2/tenants/default/vnfs/microblog/vnf_
instances/dc_1/vnfc_instances/workflow_instances
    workflow_instance_id: '112602312'
status: 201
```

As a result of the successful call, instance 2 of the `blogserver` VNFC is upgraded to template2 while the other VNFCs remain on template1 as shown.

Starting VNFCs Using REST API

To perform a partial start of a VNF using REST API, you can start VNFCs by posting a workflow request using the start_vnfc workflow_type parameter. The API request takes the following format.

```
POST http://<ovlm-workflow-fe-url>/api/v2/tenants/<tenant_id>/vnfs/<vnf_id>/vnf_instances/<vnf_instance_id>/vnfc_instances/workflow_instances?sync=1&timeout=2000
{
    "workflow_type": "start_vnfc",
    "vnfc_id": <vnfc_id>,
    ["vnfc_instance_id": <vnfc_instance_id>],
    ["additional_parameters": <additional_parameters>]
}
```

Note: All parameters are described on the main page for this section, [Running VNFC Workflows](#).

The vnfc_instance_id parameter is optional. When you do not use it, all the VNFCs with the same vnfc_id within a VNF are started. When you do use the parameter, only the VNFC with the specified instance ID is started.

Example: Starting all VNFCs with the Same vnfc_id

In this example of a partial VNF start, both VNFCs with the same vnfc_id (blogserver) are started in an instance of the microblog VNF. The instance name is dc_1. The API call to start the blogserver VNFCs using curl is shown below. In this case, vnfc_instance_id is not used in the call.

```
./start_vnfc_type.sh default microblog dc_1 blogserver '{ \"passedKey1\": \"passedKeyValue1\" }'
http://10.3.18.22:28085
- start vnfc type -
. tenant_id          = default
. vnf_id              = microblog
. vnf_instance_id     = dc_1
. vnfc_id              = blogserver

===== sample script =====

#!/bin/bash
OVLMFE_BASEURL=http://10.3.18.22:28085
echo $OVLMFE_BASEURL
if [ $# -lt 4 ]; then
    echo "usage : $0 <tenant_id> <vnf_id> <vnf_instance> <vnfc_id>
[<additional_parameters>]"
    exit 1
fi
tenant_id=$1
vnf_id=$2
vnf_instance_id=$3
vnfc_id=$4

echo " - start vnfc type - "
echo " . tenant_id          = $tenant_id"
echo " . vnf_id              = $vnf_id"
echo " . vnf_instance_id     = $vnf_instance_id"
echo " . vnfc_id              = $vnfc_id"

body="{\"workflow_type\": \"start_vnfc\", \"vnfc_id\": \"$vnfc_id\", \"additional_parameters\": \"$additional_parameters\" }"
curl -s -X POST -H "Content-Type: application/json" \
-d "$body" \
"${OVLMFE_BASEURL}/api/v2/tenants/${tenant_id}/vnfs/${vnf_id}/vnf_
```

```
instances/${vnf_instance_id}/vnfc_instances/workflow_instances?syn
c=1&timeout=2000" | python -mjson.tool | pygmentize -l json
```

The following is an example of a successful response.

```
{
  "data": {
    "flow_result": {
      "success": "\"VNFC Start passed successfully on all servers.\"",
    },
    "method": "POST",
    "request": "http://10.3.18.22:28085/api/v2/tenants/default/vnfs/microblog/vnf_instances/dc_1/vnfc_instances/workflow_instances",
    "workflow_instance_status": "COMPLETED"
  },
  "status": 200
}
```

Example: Starting a specific VNFC

In this example of a partial VNF start, a specific instance of the blogserver VNFC within the microblog VNF is started. The instance name is 1. The API call to start the blogserver VNFCs using curl is shown below. In this case, vnfc_instance_id is used to target a specific instance of the blogserver VNFC .

```
./start_vnfc_instance.sh default microblog dc_1 blogserver 1 '{ \"passedKey1\":
\"passedKeyValue1\" }'
http://10.3.18.22:28085
- start vnfc instance -
. tenant_id           = default
. vnf_id               = microblog
. vnf_instance_id     = dc_1
. vnfc_id              = blogserver
. vnfc_instance_id    = 1

===== sample script =====

#!/bin/bash
OVLMFE_BASEURL=http://10.3.18.22:28085
echo $OVLMFE_BASEURL
if [ $# -lt 5 ]; then
  echo "usage : $0 <tenant_id> <vnf_id> <vnf_instance> <vnfc_id>
<vnfc_instance_id>[<additional_parameters>]"
  exit 1
fi
tenant_id=$1
vnf_id=$2
vnf_instance_id=$3
vnfc_id=$4
vnfc_instance_id=$5

echo " - start vnfc instance - "
echo " . tenant_id           = $tenant_id"
echo " . vnf_id               = $vnf_id"
echo " . vnf_instance_id     = $vnf_instance_id"
echo " . vnfc_id              = $vnfc_id"
echo " . vnfc_instance_id    = $vnfc_instance_id"

body="{\"workflow_type\": \"start_vnfc\", \"vnfc_id\": \"${vnfc_id}\",
\"vnfc_instance_id\": \"${vnfc_instance_id}\", \"additional_parameters\":
\"${additional_parameters}\" }"
```

```
curl -s -X POST -H "Content-Type: application/json" \
-d "$body" \
"${OVLME_BASEURL}/api/v2/tenants/${tenant_id}/vnfs/${vnf_id}/vnf_instances/${vnf_instance_id}/vnfc_instances/workflow_instances?sync=1&timeout=2000" | python -mjson.tool | pygmentize -l json
```

The following is an example of a successful response.

```
{
    "data": {
        "flow_result": {
            "success": "\"VNFC Rollback passed successfully on all servers.\"",
            },
        "method": "POST",
        "request": "http://10.3.18.22:28085/api/v2/tenants/default/vnfs/microblog/vnf_instances/dc_1/vnfc_instances/workflow_instances",
        "workflow_instance_status": "COMPLETED"
    },
    "status": 200
}
```

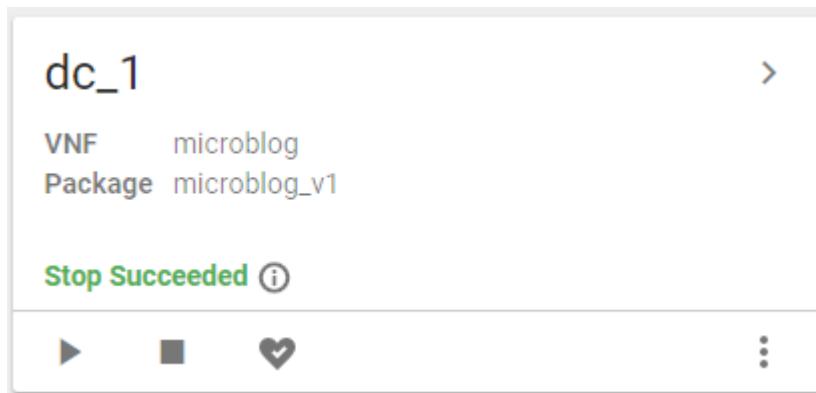
Starting VNFCs Using the VNF-M GUI

You can use the VNF-M GUI to start a VNFC, which starts all instances of the VNFC, or to start a specific VNFC instance.

Starting All instances of a VNFC

To start all instances of a VNFC, follow these steps

1. On the VNF Lifecycle page, locate the active VNF with the VNFC you intend to start and click the arrow in the top-right corner of the VNF instance card.



Scroll down the page to find the VNF if necessary.

Figure 134: Stopped VNF Instance

The VNFC Page for the selected VNF is displayed with a card for each VNFC.

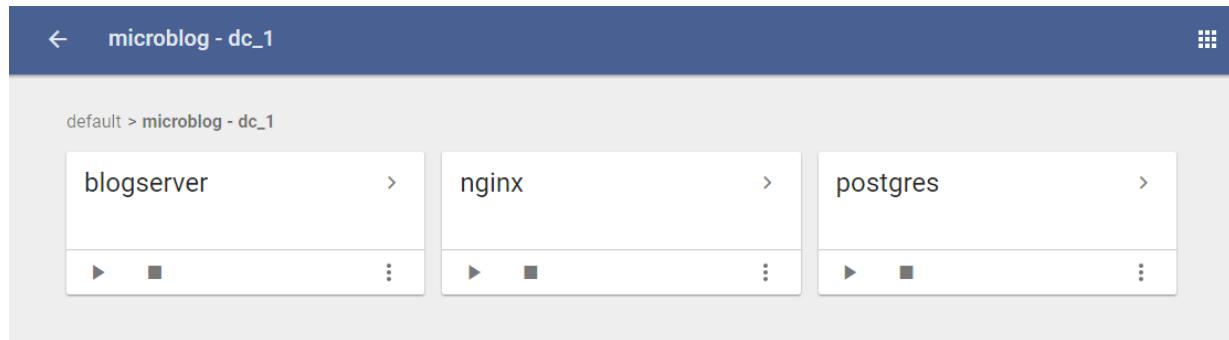


Figure 135: VNFC Page

2. Click the start icon



on the VNFC for which you are starting all instances, which in this example is blogserver. The Start VNFC confirmation box is displayed.

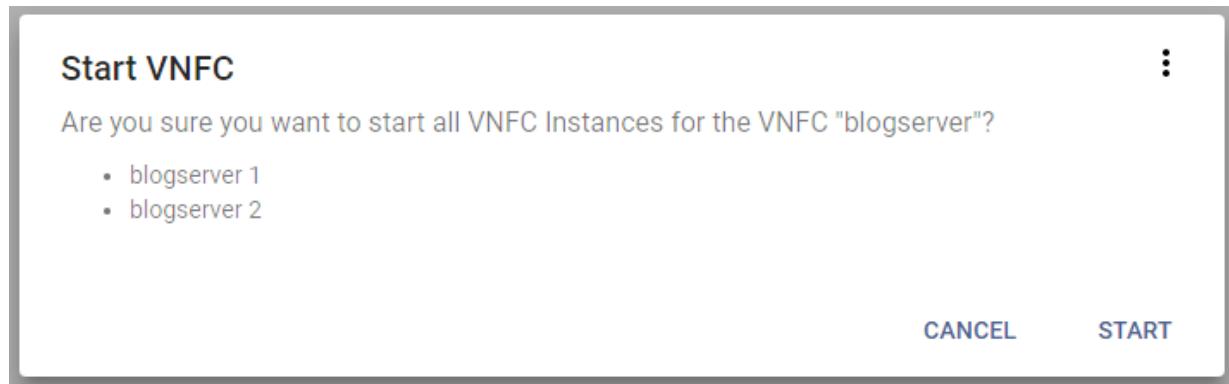


Figure 136: Start VNFC Confirmation Box

3. To configure additional parameters for a script you are running on the Resource Agent related to starting the VNFC, click the icon in the top-right of the confirmation box and select **Add parameters**.

The dialog box displays a field into which you can enter additional parameters. In some cases there might be predefined parameters, which will display when you select **Add Parameters**. Predefined parameters can be overwritten.

Note: For information about how to predefine Additional Parameters, see [Managing Additional Parameters for the VNF-M GUI](#).

Parameters must be entered in JSON format as in the following example.



Figure 137: Additional Parameters

Note: To remove additional parameters after adding them, click the top-right icon again and select **Remove parameters**.

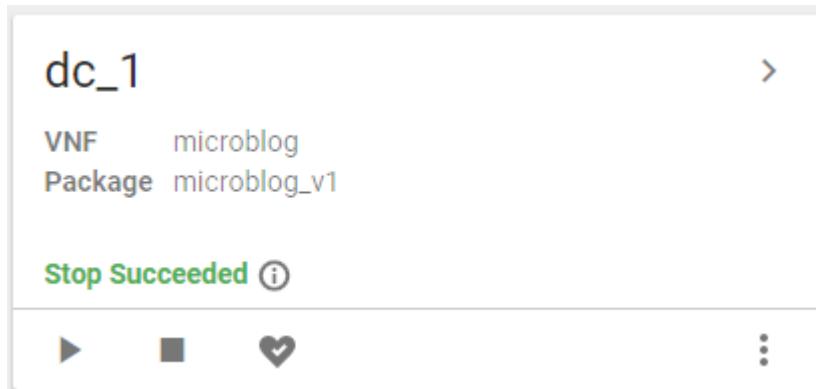
4. Click Start.

The VNFC instances are started with a status of Start Succeeded.

Starting a Specific VNFC Instance

To Start a specific VNFC instance, follow these steps:

1. On the VNF Lifecycle page, locate the active VNF with the VNFC you intend to start and click the arrow in the top-right corner of the VNF instance card.



Scroll down the page to find the VNF if necessary.

Note: Active VNFs are grouped at the bottom of the VNF Lifecycle page.

Figure 138: Active VNF

The VNFC Page for the selected VNF is displayed with a card for each VNFC.

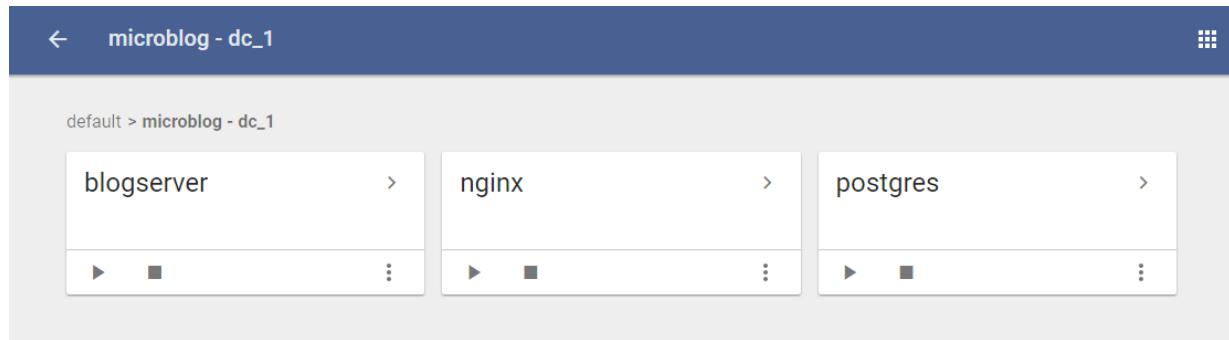


Figure 139: VNFC Page

- Click the arrow in the top-right corner of the blogserver VNFC card.
All instances of the blogserver VNFC are displayed.

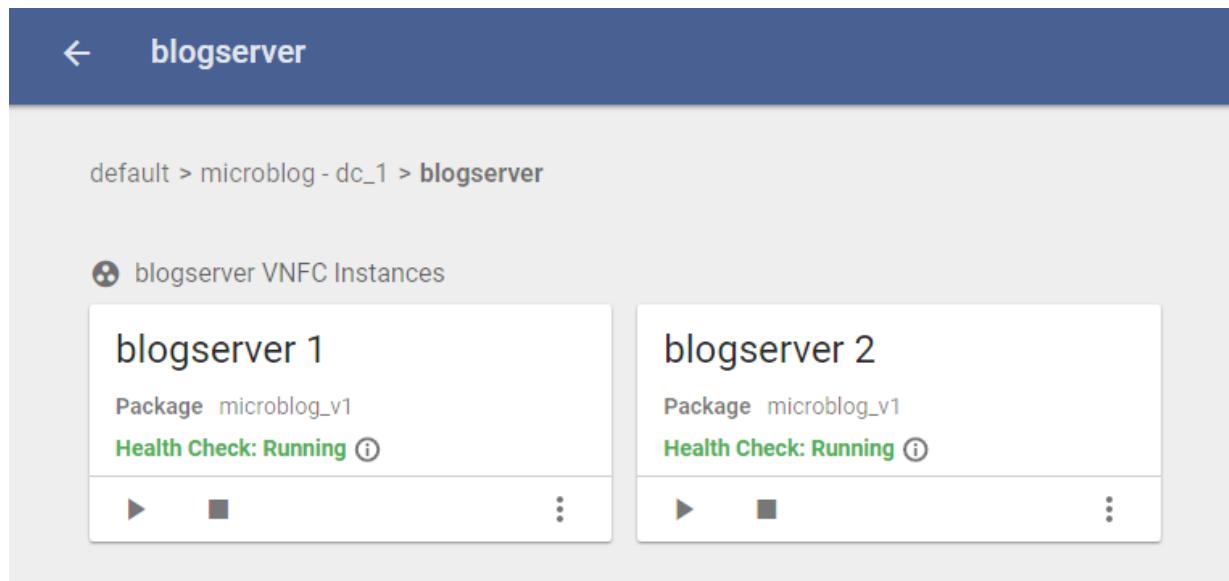


Figure 140: Blogserver VNFC Instances

- Click the start icon



on the VNFC instance to start, which in this example is blogserver 1.
The Start VNFC Instance confirmation box is displayed.

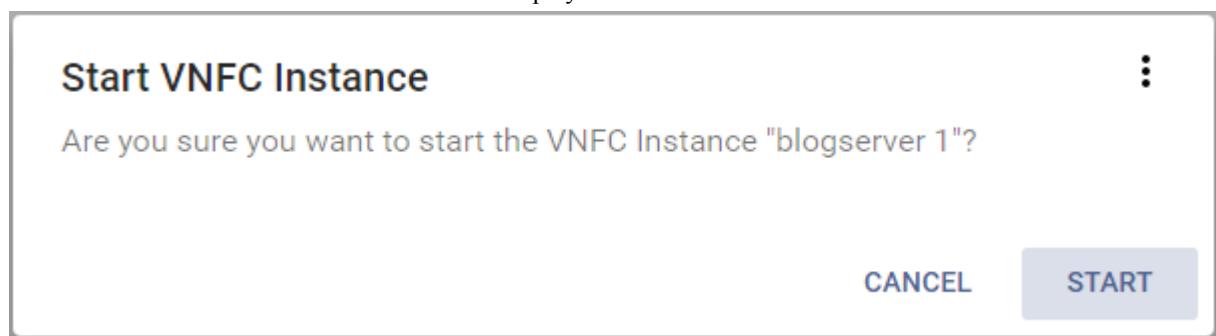


Figure 141: Start VNFC Instance Confirmation Box

- To configure additional parameters for a script you are running on the Resource Agent related to starting the VNFC, click the icon in the top-right of the confirmation box and select **Add parameters**.

The dialog box displays a field into which you can enter additional parameters. In some cases there might be predefined parameters, which will display when you select **Add Parameters**. Predefined parameters can be overwritten.

Note: For information about how to predefine Additional Parameters, see [Managing Additional Parameters for the VNF-M GUI](#).

Parameters must be entered in JSON format as in the following example.

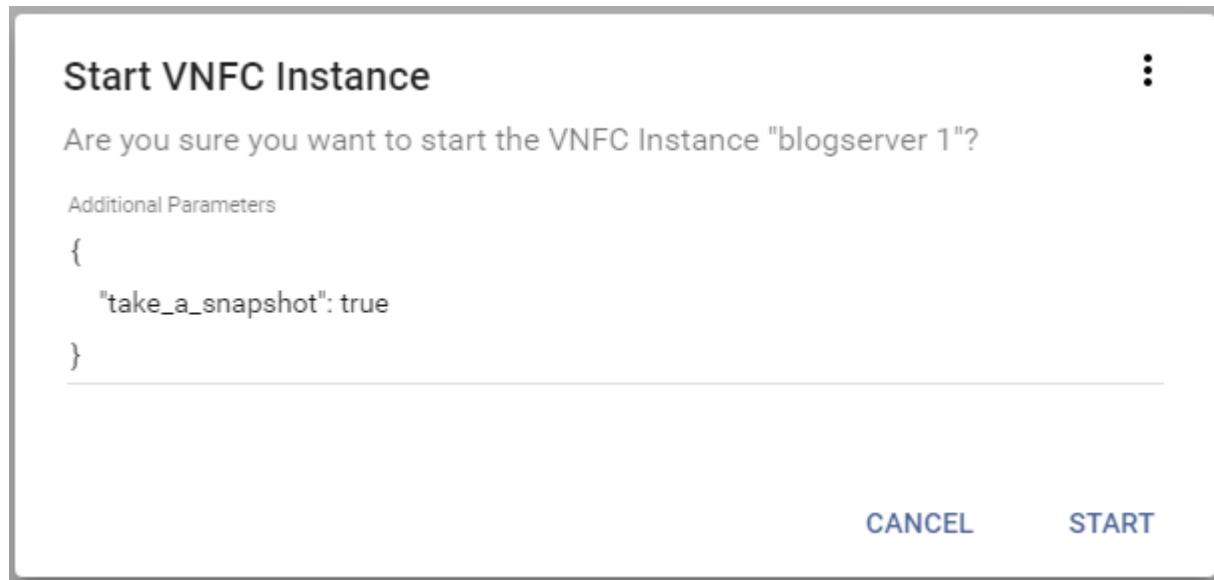


Figure 142: Additional Parameters

Note: To remove additional parameters after adding them, click the top-right icon again and select **Remove parameters**.

- Click **Start**.

The VNFC instance is started with a status of Start Succeeded.

Stopping VNFCs

To stop one or more VNFC instances, use the `stop_vnf` workflow_type parameter. You can run the workflow against all VNFCs of the same type or a specific VNFC instance.

You can stop VNFC instance from the CLI, the API, or from the Openet Weaver VNF-M GUI.

Stopping VNFCs from the CLI

To perform a partial stop of a VNF, you can stop specific VNFCs using `stop_vnfc` as the workflow_type parameter with the `vnfc_workflow submit` command.

```
ovlm-fe vnfc_workflow submit -t <tenant_id> -v <vnf_id> -i <vnf_instance_id>
-y stop_vnfc -c <vnfc_id> [-n <vnfc_instance_id>] [-sync <sync>]
[-timeout <seconds>] [-a '{<additional_parameter1, additional_parameter2,...>}']
```

Note: All parameters are described on the main page for this section, [Running VNFC Workflows](#).

The command stops all VNFC instances with the specified `vnfc_id`. When the optional `vnfc_instance_id` parameter is also used in the command, only that VNFC instance is stopped.

Example: Stopping all VNFCs with the Same vnfc_id

In this example of a partial VNF start, the command stops all VNFCs with the same `vnfc_id` (blogserver) within in an instance of a microblog VNF named `dc_1`.

```
ovlm-fe vnfc_workflow submit -y stop_vnfc -t Default -v microblog -i dc_1 -c
blogserver
```

Note: Be sure the baseurl environment variables are set before using CLI commands.

The following is an example of a successful response to running the command.

```
data:
  method: POST
  request: http://10.3.18.22:28085/api/v2/tenants/default/vnfs/microblog/vnf_
instances/dc_1/vnfc_instances/workflow_instances
    workflow_instance_id: '112602008'
status: 201
```

Example: Stop a Specific VNFC Instance

In this example of a partial VNF stop, the command stops a specific VNFC within in an instance (2) of a microblog VNF named `dc_1`.

```
ovlm-fe vnfc_workflow submit -y stop_vnfc -t Default -v microblog -i dc_1 -c
blogserver -n 1
blogserver
```

The following is an example of a successful response to running the command.

```
data:
  method: POST
  request: http://10.3.18.22:28085/api/v2/tenants/default/vnfs/microblog/vnf_
instances/dc_1/vnfc_instances/workflow_instances
    workflow_instance_id: '112602312'
status: 201
```

As a result of the successful call, instance 2 of the blogserver VNFC is upgraded to template2 while the other VNFCs remain on template1 as shown.

Stopping VNFCs Using REST API

To perform a partial stop of a VNF using REST API, you can stop VNFCs by posting a workflow request using the `stop_vnfc` `workflow_type` parameter. The API request takes the following format.

```
POST http://<ovlm-workflow-fe-url>/api/v2/tenants/<tenant_id>/vnfs/<vnf
_id>/vnf_instances/<vnf_instance_id>/vnfc_instances/workflow_insta
nces?sync=1&timeout=2000
{
  "workflow_type": "stop_vnfc",
  "vnfc_id": <vnfc_id>,
  ["vnfc_instance_id": <vnfc_instance_id>],
  ["additional_parameters": <additional_parameters>]
}
```

Note: All parameters are described on the main page for this section, [Running VNFC Workflows](#).

The `vnfc_instance_id` parameter is optional. When you do not use it, all the VNFCs with the same `vnfc_id` within a VNF are stopped. When you do use the parameter, only the VNFC with the specified instance ID is stopped.

Example: Stopping all VNFCs with the Same vnfc_id

In this example of a partial VNF stop, both VNFCs with the same vnfc_id (blogserver) are stopped in an instance of the microblog VNF. The instance name is dc_1. The API call to stop the blogserver VNFCs using curl is shown below. In this case, vnfc_instance_id is not used in the call.

```
./stop_vnfc_type.sh default microblog dc_1 blogserver '{ \"passedKey1\": \"passedKeyValue1\" }'
http://10.3.18.22:28085
- stop vnfc type -
. tenant_id           = default
. vnf_id               = microblog
. vnf_instance_id     = dc_1
. vnfc_id              = blogserver

===== sample script =====

#!/bin/bash
OVLMFE_BASEURL=http://10.3.18.22:28085
echo $OVLMFE_BASEURL
if [ $# -lt 4 ]; then
    echo "usage : $0 <tenant_id> <vnf_id> <vnf_instance> <vnfc_id>
[<additional_parameters>]"
    exit 1
fi
tenant_id=$1
vnf_id=$2
vnf_instance_id=$3
vnfc_id=$4

echo " - stop vnfc type - "
echo " . tenant_id           = $tenant_id"
echo " . vnf_id               = $vnf_id"
echo " . vnf_instance_id     = $vnf_instance_id"
echo " . vnfc_id              = $vnfc_id"

body="{"workflow_type": "stop_vnfc", "vnfc_id": "${vnfc_id}", "additional_parameters": "${additional_parameters}" }"
curl -s -X POST -H "Content-Type: application/json" \
-d "$body" \
"${OVLMFE_BASEURL}/api/v2/tenants/${tenant_id}/vnfs/${vnf_id}/vnf_instances/${vnf_instance_id}/vnfc_instances/workflow_instances?sync=1&timeout=2000" | python -mjson.tool | pygmentize -l json
```

The following is an example of a successful response.

```
{
  "data": {
    "flow_result": {
      "success": "\"VNFC Stop passed successfully on all servers.\\"", },
      "method": "POST",
      "request": "http://10.3.18.22:28085/api/v2/tenants/default/vnfs/microblog/vnf_instances/dc_1/vnfc_instances/workflow_instances",
      "workflow_instance_status": "COMPLETED"
    },
    "status": 200
  }
```

Example: Stopping a specific VNFC

In this example of a partial VNF stop, a specific instance of the blogserver VNFC within the microblog VNF is stopped. The instance name is 1. The API call to stop the blogserver VNFCs using curl is shown below. In this case, vnfc_instance_id is used to target a specific instance of the blogserver VNFC .

```
./stop_vnfc_instance.sh default microblog dc_1 blogserver 1 '{ \"passedKey1\":
  \"passedKeyValue1\" }'
http://10.3.18.22:28085
- stop vnfc instance -
. tenant_id          = default
. vnf_id              = microblog
. vnf_instance_id    = dc_1
. vnfc_id              = blogserver
. vnfc_instance_id    = 1

=====
sample script =====

#!/bin/bash
OVLMFE_BASEURL=http://10.3.18.22:28085
echo $OVLMFE_BASEURL
if [ $# -lt 5 ]; then
    echo "usage : $0 <tenant_id> <vnf_id> <vnf_instance> <vnfc_id>
<vnfc_instance_id>[<additional_parameters>]"
    exit 1
fi
tenant_id=$1
vnf_id=$2
vnf_instance_id=$3
vnfc_id=$4
vnfc_instance_id=$5

echo " - stop vnfc instance - "
echo " . tenant_id          = $tenant_id"
echo " . vnf_id              = $vnf_id"
echo " . vnf_instance_id    = $vnf_instance_id"
echo " . vnfc_id              = $vnfc_id"
echo " . vnfc_instance_id    = $vnfc_instance_id"

body="{"workflow_type": "stop_vnfc", "vnfc_id": "${vnfc_id}", "vnfc_instance_id": "${vnfc_instance_id}", "additional_parameters": "${additional_parameters}"}
curl -s -X POST -H "Content-Type: application/json" \
-d "$body" \
"${OVLMFE_BASEURL}/api/v2/tenants/${tenant_id}/vnfs/${vnf_id}/vnf_instances/${vnf_instance_id}/vnfc_instances/workflow_instances?sync=1&timeout=2000" | python -mjson.tool | pygmentize -l json
```

The following is an example of a successful response.

```
{
  "data": {
    "flow_result": {
      "success": "\"VNFC Rollback passed successfully on all servers.\"",

    },
    "method": "POST",
    "request": "http://10.3.18.22:28085/api/v2/tenants/default/vnfs/microblog/vnf_instances/dc_1/vnfc_instances/workflow_instances",
    "workflow_instance_status": "COMPLETED"
```

```

    },
  "status": 200
}

```

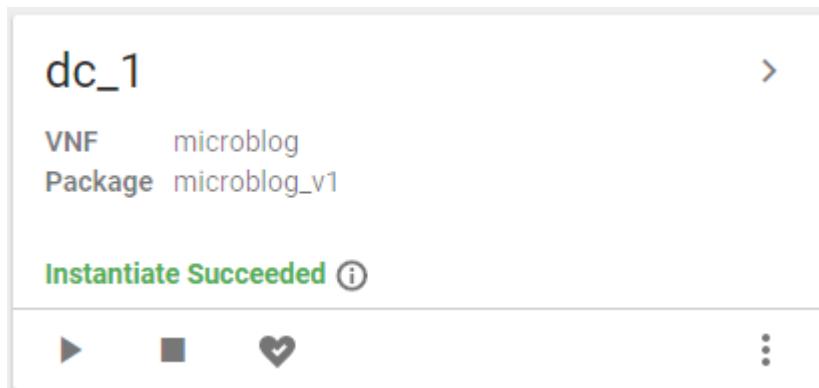
Stopping VNFCs Using the VNF-M GUI

You can use the VNF-M GUI to stop a VNFC, which stops all instances of the VNFC, or to stop a specific VNFC instance.

Stopping All Instances of a VNFC

To stop all instances of a VNFC, follow these steps

1. On the VNF Lifecycle page, locate the active VNF with the VNFC you intend to stop and click the arrow in the top-right corner of the VNF instance card.



Scroll down the page to find the VNF if necessary.

Note: Active VNFs are grouped at the bottom of the VNF Lifecycle page.

Figure 143: Active VNF

The VNFC Page for the selected VNF is displayed with a card for each VNFC.

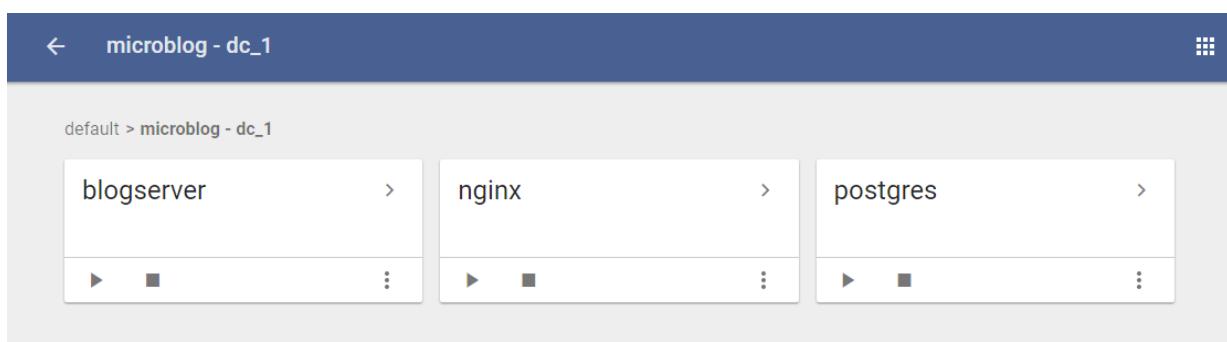


Figure 144: VNFC Page

2. Click the Stop icon



on the VNFC for which you intend to stop all instances, which in this example is blogserver. The Stop VNFC confirmation box is displayed.

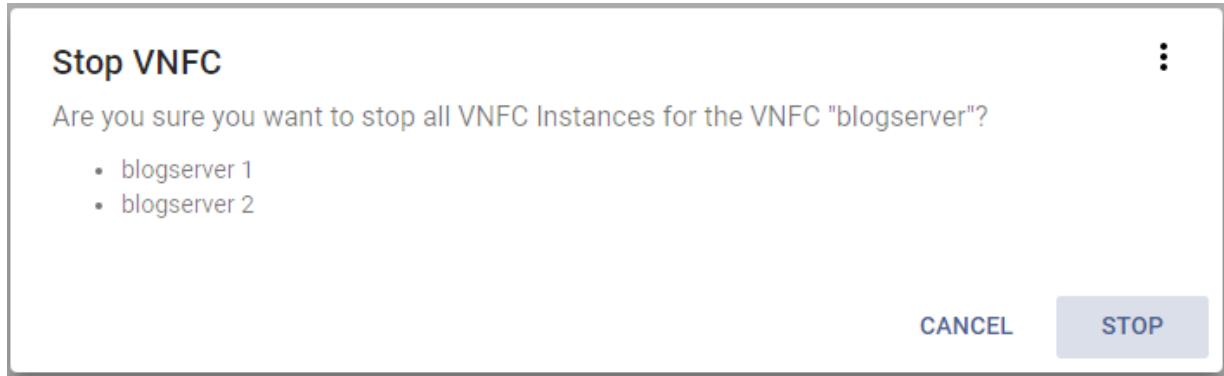


Figure 145: Stop VNFC Confirmation Box

3. To configure additional parameters for a script you are running on the Resource Agent related to stopping the VNFC, click the icon in the top-right of the confirmation box and select **Add parameters**.

The dialog box displays a field into which you can enter additional parameters. In some cases there might be predefined parameters, which will display when you select **Add Parameters**. Predefined parameters can be overwritten.

Note: For information about how to predefine Additional Parameters, see *Managing Additional Parameters for the VNF-M GUI*.

Parameters must be entered in JSON format as in the following example.



Figure 146: Additional Parameters

Note: To remove additional parameters after adding them, click the top-right icon again and select **Remove parameters**.

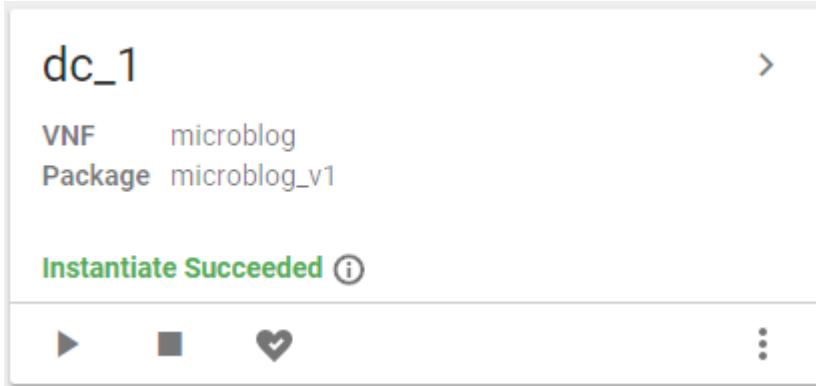
4. Click **Stop**.

The VNFC instances are started with a status of Stop Succeeded.

Stopping a Specific VNFC Instance

To stop a specific VNFC instance follow these steps:

1. On the VNF Lifecycle page, locate the active VNF with the VNFC you intend to stop and click the arrow in the top-right corner of the VNF instance card.



Scroll down the page to find the VNF if necessary.

Note: Active VNFs are grouped at the bottom of the VNF Lifecycle page.

Figure 147: Active VNF

The VNFC Page for the selected VNF is displayed with a card for each VNFC.

Figure 148: VNFC Page

2. Click the arrow in the top-right corner of the blogserver VNFC card.
All instances of the blogserver VNFC are displayed.

Figure 149: Blogserver VNFC Instances

- Click the Stop icon



on the VNFC to stop.

The Stop VNFC Instance confirmation box is displayed.

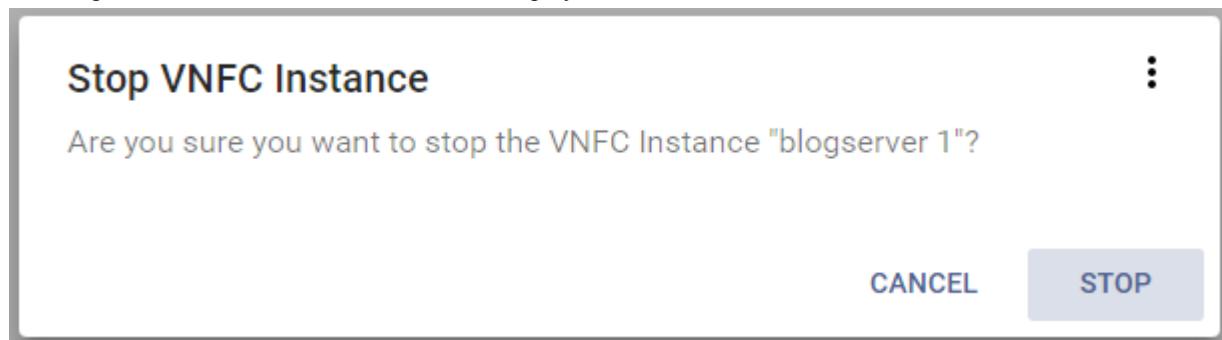


Figure 150: Stop VNFC Instance Confirmation Box

- To configure additional parameters for a script you are running on the Resource Agent related to stopping the VNFC, click the icon in the top-right of the dialog box and select **Add parameters**.

The dialog box displays a field into which you can enter additional parameters. In some cases there might be default parameters and values configured, which will display when you select **Add Parameters**. The default parameters can be overwritten.

The dialog box displays a field into which you can enter additional parameters. In some cases there might be predefined parameters, which will display when you select **Add Parameters**. Predefined parameters can be overwritten.

Note: For information about how to predefine Additional Parameters, see [Managing Additional Parameters for the VNF-M GUI](#).

Parameters must be entered in JSON format as in the following example.

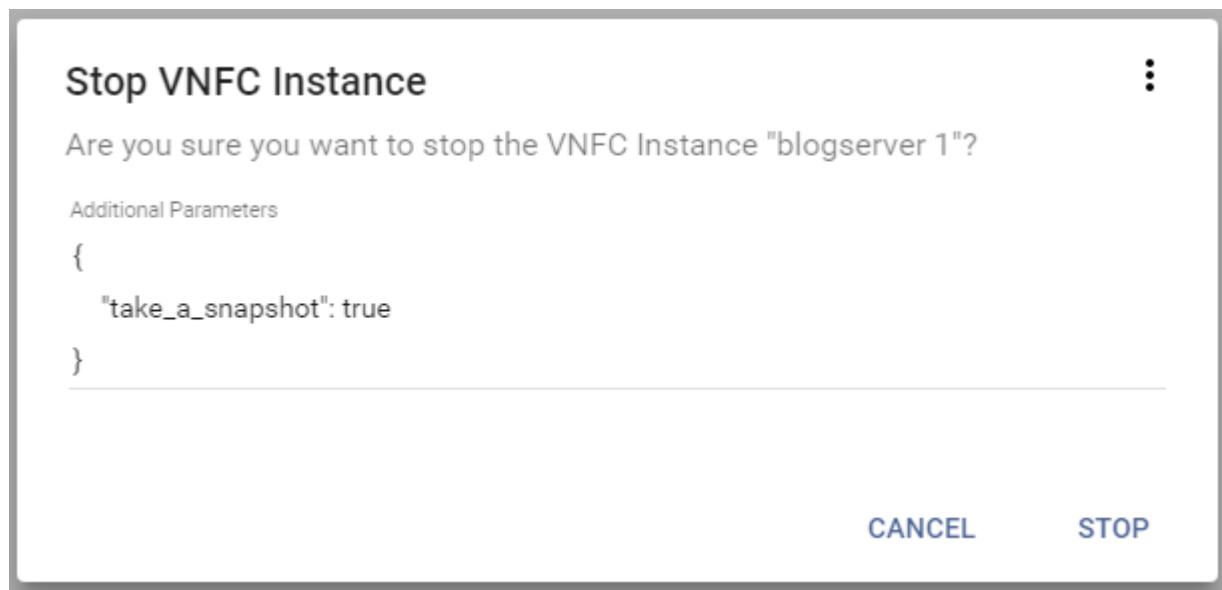


Figure 151: Additional Parameters

Note: To remove additional parameters after adding them, click the top-right icon again and select **Remove parameters**.

- Click **Stop**.

The VNFC instance is stopped with a status of Stop Succeeded.

Running a custom_action_vnfc Workflow

To run a custom lifecycle event based on a script created specifically for the event, you run the custom_action_vnfc workflow. To support this type of workflow, the workflow submit command accepts some different parameters than it does for running other workflows. The following table describes the supported parameters.

Table 41: custom_action_vnfc workflow submit parameters

Flag	Parameter	Mand.	Description
-t	tenant_id	Yes	The tenant identifier as created in the Configuration Manager.
-v	vnf_id	Yes	The VNF identifier as created in the Configuration Manager.
-i	vnf_instance_id	Yes	The VNF instance identifier as created in the Configuration Manager.
-c	vnfc_id	Yes	The VNFC identifier
-n	vnfc_instance_id	No	The VNFC instance identifier
-procedure_name	procedure_filename	Yes	The name of the script to be invoked. When this parameter is used, the custom action is triggered for that VNFC only. Otherwise the command applies to all VNFCs in the VNF.
-script_timeout	script_timeout	No	The timeout (in second) to complete invocation of the procedure. The default value is 60.
-a	additional_parameters	No	The parameters to be passed into the script for the event. Parameters must be in JSON format and in the correct, expected sequence during script invocation.
-o	execution_order	No	When there are multiple VNFC instances, this parameter specifies whether the custom action is applied to the instances in parallel or in sequence. Possible values are: <ul style="list-style-type: none"> • sequential (default) • parallel
-run_as	run_as	No	Enables you to specify whether to run the custom_action_vnfc workflow as a specific user other than the ovlm user.

The process flow that is initiated when the custom_action_vnfc workflow is run is shown below.

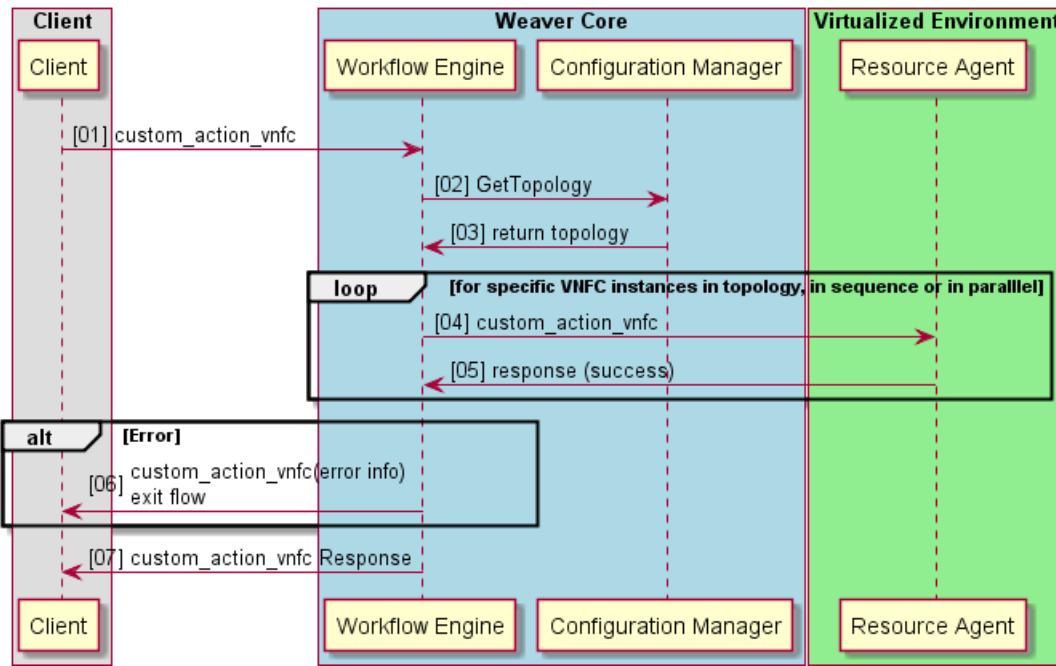


Figure 152: custom_action_vnfc Process Flow

Running a custom_action_vnfc Workflow from the CLI

To run a custom action on a VNFC from the CLI, use the `custom_action_vnfc` workflow_type parameter. When running this workflow type, the workflow submit command takes the following format:

```

ovlm-fe vnfc_workflow submit -t <tenant_id> -v <vnf_id> -i <vnf_instance_id>
  -c <vnfc_id> -n <vnfc_instance_id> \
  -y custom_action_vnfc -procedure_filename "<procedure_filename>" -script_timeout
    <script_timeout> -o "<execution_order>" \
  -a <additional_parameters> \
  -run_as <user_name>

```

In the following example the command is used to run `custom_script.sh` on an instance of the microblog `httpd` VNFC for the default tenant. A username and password are passed in as parameters. The `execution_order` option is set to `parallel`, which means each VNFC instance is processed simultaneously.

```

ovlm-fe vnfc_workflow submit -t default -v microblog -i dc_1 -c nginx -n nginx_1
  -y custom_action_vnfc -procedure_filename "custom_script.sh" \
  -script_timeout 200 -o "parallel" -a '{"username": "admin", "password": "123Xyz"}'

```

The following is an example of a successful response to running the CLI command with `custom_action_vnfc`.

```

data:
  workflow_instance_status: COMPLETED
  flow_result:
    vnfc_instances:
      - vnfc_id: blogserver
        vnfc_instance_id: '1'
        hostname: 10.3.12.144
        exit_code: '0'
        response:
          output: The script is executed successfully
      - vnfc_id: blogserver

```

```

vnfc_instance_id: '2'
hostname: 10.3.12.143
exit_code: '0'
response:
  stats:
    - name: cpu_usage
      value: '11'
    - name: cpu_5min_usage
      value: '16'
    - name: memory_usage
      value: '53'
request: http://ovml:28085/api/v2/tenants/default/vnfs/microblog/vnf_instances/dc_1/workflow_instances
method: GET
status: 200

```

The following is an example of a failure response for the custom_action_vnfc workflow.

```

data:
  workflow_instance_status: FAILED
  error_code: EXECUTE_WORKFLOW_FAILED
  error_msg: VNFC Custom Action Invocation Failed
  flow_result:
    vnfc_instances:
      - vnfc_id: blogserver
        vnfc_instance_id: '1'
        hostname: 10.3.12.32
        exit_code: '1'
        response:
          error:
            - '  File "./test.py", line 19'
            - '    :     sys.exit(0)'
            - '    ^'
            - 'SyntaxError: invalid syntax'
      - vnfc_id: blogserver
        vnfc_instance_id: '2'
        hostname: 10.3.12.144
        exit_code: '1'
        response:
          error: Script '/usr/lib64/ovlm/ra/repository/active/tenants/default/vnfs/microblog/templates/microblog_v1/vnfcs/blogserver/procedures/test.py' not found
request: http://ovml:28085/api/v2/tenants/default/vnfs/microblog/vnf_instances/dc_1/workflow_instances
method: GET
status: 500

```

Running a custom_action_vnfc Workflow Using REST API

To run a custom action on a VNFC using REST API, post a workflow request using the custom_action_vnfc workflow_type parameter. The API request takes the following format.

```

POST http://<ovlm-workflow-fe-url>
/api/v2/tenants/<tenant_id>/vnfs/<vnf_id>/vnf_instances/<vnf_instance_id>
  /vnfc_instances/workflow_instances
{
  "workflow_type": "custom_action_vnfc",
  "vnfc_id": <vnfc_id>,
  "vnfc_instance_id": <vnfc_instance_id>,
  "procedure_filename": "<Procedure_Filename>",
  "script_timeout": <Timeout_In_Seconds> ,

```

```

    "additional_parameters": <additional_parameters>,
    "execution_order": <execution_order>
    "run_as": <user_name>
}

```

The following is an example of running a custom action using curl. The command is used to run `custom_script.sh` on an instance of the microblog `httpd` VNFC for the default tenant. A username and password are passed in as parameters. The execution order is set to process each VNFC instance in sequence.

```

POST http://mgmt-k1-005114-vm01.openet-telecom.lan:28050/api/v2/tenants
/default/vnfs/microblog/vnf_instances/dc_1/vnfc_instances/workflow_instances
{
    "workflow_type": "custom_action_vnfc",
    "vnfc_id": "blogserver",
    "vnfc_instance_id": 1,
    "procedure_filename": "test.sh",
    "script_timeout": 80,
    "additional_parameters": "[\"username\": \"administrator\", \"password\": \"password123\"]",
    "execution_order": "parallel"
}

```

The following is an example response to the API call to running the `custom_vnfc_action` workflow.

```

{
  "data": {
    "workflow_instance_status": "COMPLETED",
    "flow_result": {
      "vnfc_instances": [
        {
          "vnfc_id": "blogserver",
          "vnfc_instance_id": 1,
          "hostname": "10.3.12.32",
          "exit_code": "1",
          "response": {
            "output": "The script is executed successfully"
          }
        },
        {
          "vnfc_id": "blogserver",
          "vnfc_instance_id": 2,
          "hostname": "10.3.12.144",
          "exit_code": "0",
          "response": {
            "stats": [
              {
                "name": "cpu_usage",
                "value": "11"
              },
              {
                "name": "cpu_5min_usage",
                "value": "16"
              },
              {
                "name": "memory_usage",
                "value": "53"
              }
            ]
          }
        }
      ]
    }
  }
}

```

```

},
  "request": "http://10.3.12.24:28085/api/v2/tenants/default/vnfs/microblog/vnf_instances/dc_1/workflow_instances",
  "method": "GET"
},
"status": 200
}

```

The following is an example failure response to running the custom_vnfc_action workflow.

```

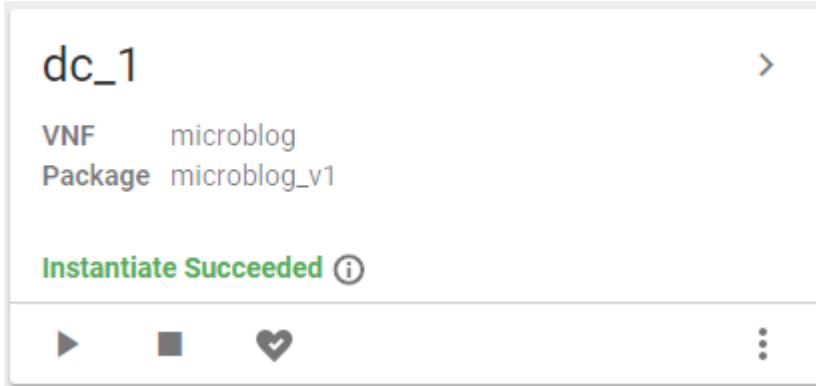
{
  "data": {
    "workflow_instance_status": "FAILED",
    "error_code": "EXECUTE_WORKFLOW_FAILED",
    "error_msg": "VNFC Custom Action Invocation Failed.",
    "flow_result": {
      "vnfc_instances": [
        {
          "vnfc_id": "blogserver",
          "vnfc_instance_id": "1",
          "hostname": "10.3.12.32",
          "exit_code": "1",
          "response": {
            "error": [
              "  File './test.py\\'', line 19",
              "    : sys.exit(0)",
              "  ^",
              "SyntaxError: invalid syntax"
            ]
          }
        },
        {
          "vnfc_id": "blogserver",
          "vnfc_instance_id": "2",
          "hostname": "10.3.12.144",
          "exit_code": "1",
          "response": {
            "error": "Script '/usr/lib64/ovlm/ra/repository/active/tenants/default/vnfs/microblog/templates/microblog_v1/vnfcs/blogserver/procedures/test.py' not found"
          }
        }
      ],
      "request": "http://10.3.12.24:28085/api/v2/tenants/default/vnfs/microblog/vnf_instances/dc_1/workflow_instances",
      "method": "GET"
    },
    "status": 500
  }
}

```

[Running a custom_action_vnfc Workflow Using the VNF-M GUI](#)

You can use the VNF-M GUI to run a custom action on a VNFC. To do so, follow these steps:

1. On the VNF Lifecycle page, locate the active VNF with the VNFC you intend to run the custom action against and click the arrow in the top-right corner of the VNF instance card.



Scroll down the page to find the VNF if necessary.

Note: Active VNFs are grouped at the bottom of the VNF Lifecycle page.

Figure 153: Active VNF

The VNFC Page for the selected VNF is displayed with a card for each VNFC.

Figure 154: VNFC Page

2. Click the more options icon



on the bottom-right of the target VNFC, which in this example is blogserver, and select Custom Action. The Run Custom Action dialog box is displayed.

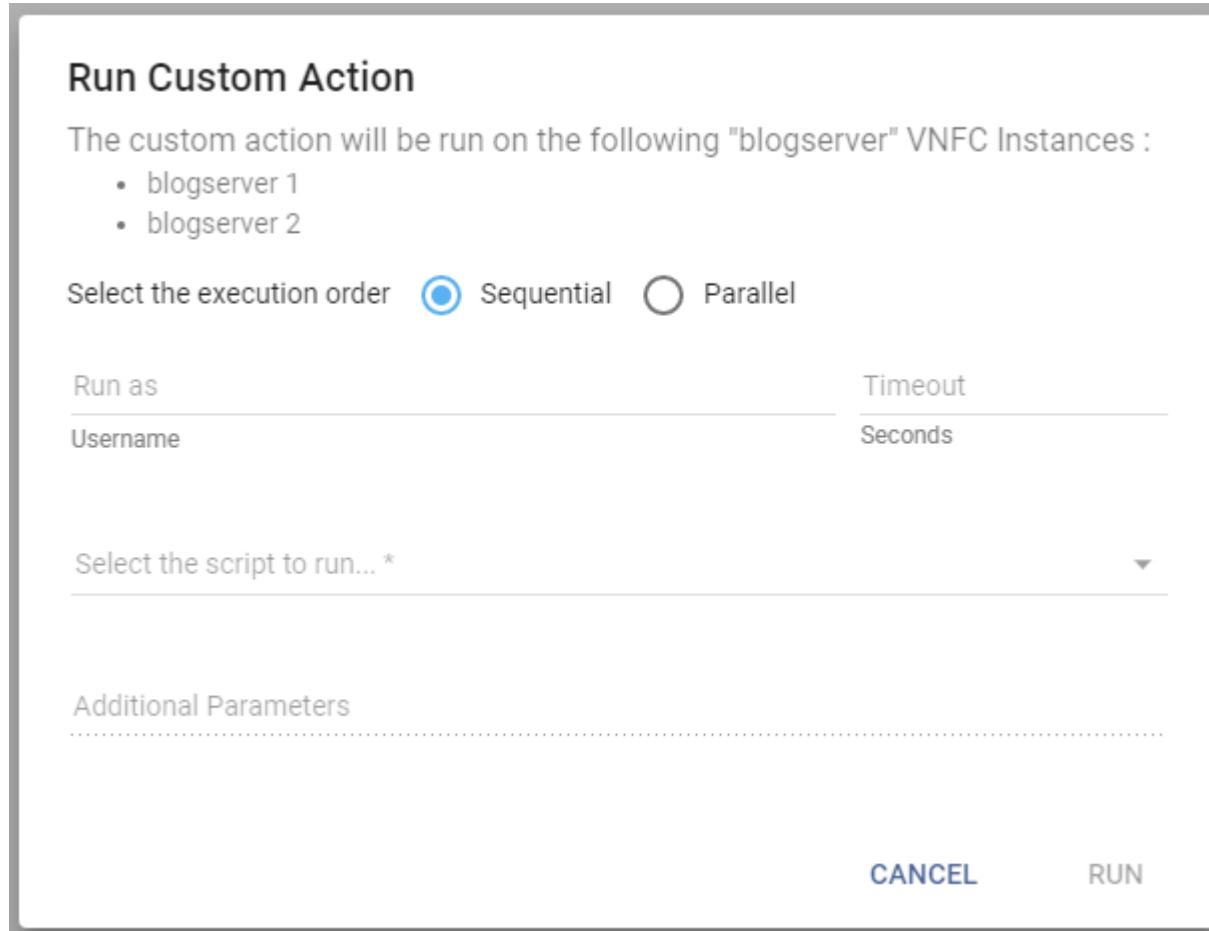


Figure 155: Run Custom Action Dialog Box

Note: In the event that there are no custom actions defined in the metadata for the VNFC, a message box displays stating that fact.

3. In the **Select the execution order** field, click a radio button to specify whether the update will be made to each VNFC instance in series or in parallel.
4. Click **Run as**, and enter a username with permission to run a custom action on the VNFC.

Note: For more information about the run as parameter, see [Creating the Metadata for the VNFC](#).

5. Click **Timeout**, then enter the amount of time, in seconds, that can elapse before the custom action times out from a failed run.
6. Click **Script**, then select the script to run for the custom action
If there are predefined Additional Parameters, they display after a script is selected. The predefined parameters can be overwritten.

Note: For information about how to predefine Additional Parameters, see [Managing Additional Parameters for the VNF-M GUI](#).

7. Click **Run** to run the custom action.

[Creating and invoking a Custom Script for the custom_action_vnfc Workflow](#)

This topic describes the process for creating a script for a custom lifecycle event and running it in a custom_action_vnfc workflow. The examples in the procedure use the default tenant, the microblog vnf, a vnf_instance_id of dc_1, and the blogserver vnfc for parameter values. To do so, follow these steps:

1. Create the script.

In this example the script is named `create_credential_file.py`, which is a shell script that creates a file with a username and password when the script is run. The file contents are shown below.

```

#!/usr/bin/python
import yaml
import json
import sys
import logging
from ovlm import systemd_services
from ovlm.parameters import Params
from ovlm import logger
from ovlm import linux_shell
if __name__ == '__main__':
    if len(sys.argv) < 2:
        sys.exit('Usage: ' + __file__ + ' YAML_CONFIG_PATH')
    PARAMS = Params(sys.argv[1])
    logging_path = PARAMS.get_param('logging_path')
    logging_level = PARAMS.get_param('logging_level')
    logger.initialize(logging_path, logging_level)
    logger.write(PARAMS.get_dump())
    try:
        ## Read all parameters from YAML file
        ALLPARAMS = PARAMS.get_all_params()

        ## Retrieve value of additional_parameters
        additional_param = ALLPARAMS['request'].get('additional_parameters')

        ## Parse value of additional_parameters which is in JSON format
        jObject = json.loads(additional_param)

        ## Retrieve value of username
        username = jObject['username']

        ## Retrieve value of password
        password = jObject['password']
    except KeyError as err:
        error_msg = 'Parameter ' + str(err) + ' is missed in the yaml file'
        sys.exit('ERROR ' + error_msg)
    if not username:
        logging.error('User name is empty')
        sys.exit("User name is empty")
    if not password:
        logging.error('Password is empty')
        sys.exit("Password is empty")

    # create a file with a content of username and password
    cmd = "echo username=" + username + ", password=" + password + " >
/tmp/test.out"
    result = linux_shell.run_shell_command(cmd)
    if result.returncode != 0:
        logging.error('ERROR Unable to execute command ' + cmd)
        sys.exit('Error ' + result.stderr)
    sys.stdout.write("The script is executed successfully")

```

In this example, the script contains configured responses to running the script. Two types of responses can be returned: standard output and standard error. The Resource Agent is able to capture both of these responses during script execution and return them to the Front-End microservice. Response methods must be appropriate to the script type, which can be shell script or python script. The following table describes the response methods

Table 42: Response Methods

Script Type	Method		Examples
	Standard Output	Standard Error	
Shell Script	Redirect a response by using ">&1"	Redirect a response by using ">&2"	echo "Successfully executed" > &1 echo "Error: invalid value" >&2
Python Script	Call python library sys.stdout.write()	Call python library sys.stderr.write() or sys.exit()	sys.stdout.write("Successfully executed") sys.stderr.write("Error: invalid value") sys.exit("Error: invalid value")

Note:

All Openet Weaver out-of-the-box procedure outputs are converted to JSON objects by default (Stdout/stderr). As a VNF developer, you are responsible for converting your custom Openet Weaver procedure outputs to JSON format. For example:

- Multiple strings separated by "\n" are converted to a JSON array of strings
- CSV formatted strings will be converted to a JSON object

You must use single quotations instead of double quotations in a String object.

The following is an example of a CSV formatted strings:

```
"name","value"
"cpu_usage",11
"cpu_5min_usage",16
"memory_usage",53
```

The following is an example of a python script used to convert the CSV formatted strings to JSON format:

```
#!/usr/bin/python
import sys
import json
import csv

if __name__ == "__main__":
    csvfile = open('file.csv', 'r')
    statItem = []
    for row in csv.DictReader(csvfile):
        statItem.append(row)
    stats = {"stats":statItem}
    sys.stdout.write(json.dumps(stats) + '\n')
    sys.stdout.write('The script is executed successfully\n')
```

2. Upload the script to Configuration Manager, in the blogserver VNFC of the default tenant microblog VNF, using the following CLI command.

```
ovlm-cm vnfc_procedure upload -t default -v microblog -c blogserver -f
create_credential_file.sh
```

3. Create a new package with the name new_package for the custom lifecycle event using the following CLI command.

```
ovlm-cm template create -t default -v microblog -p new_package
```

4. Instantiate the VNF with the new package using the following command.

```
ovlm-fe workflow submit -t default -v microblog -i dc1 -y
instantiate_vnf_without_vim -p new_package
```

5. Run the custom_action_vnfc workflow, which runs the create_credential_file.sh script, using the following command, and passing in the username and password that are placed in the credentials file.

```
ovlm-fe workflow submit -t default -v microblog -i dc_1 -c blogserver -y
custom_action_vnfc -procedure_filename create_credential_file.py \n
-script_timeout 30 -a '{"username": "administrator", "password": "password123"}'
```

The following is an example response to running the command successfully.

```
data:
  workflow_instance_status: COMPLETED
  flow_result:
    vnfc_instances:
      - vnfc_id: blogserver
        vnfc_instance_id: '2'
        hostname: 10.3.12.143
        exit_code: '0'
        response:
          stats:
            - name: cpu_usage
              value: '11'
            - name: cpu_5min_usage
              value: '16'
            - name: memory_usage
              value: '53'
          output: The script is executed successfully
        request: http://ovml1:28085/api/v2/tenants/default/vnfs/microblog/vnf_instances/dc_1/workflow_instances
        method: GET
      status: 200
```

Cancelling a Workflow

You can cancel a running workflow using the command `ovlm-fe workflow cancel -t tenant_id -v vnf_id -i vnf_instance_id -w workflow_instance_id`.

Note: A `workflow_instance_id` is returned in the response when you run a workflow.

 **Warning:** Cancelling a workflow can result in the VNF instance being placed in an inconsistent state. Therefore, do not cancel workflows in a production environment. Cancelling a running workflow may also cause the VNF instance to become locked, which means no other workflow can be run against the VNF instance. See the troubleshooting section [Problem: Locked VNF Instance](#) for how to resolve this issue.

This operation is intended for test environment usage only. Use it with caution. The cancel action is not a supported state in Instance Inventory Manager.

The following is an example of a workflow being canceled that is running against the dc_1 instance of the microblog vnf.

```
ovlm-fe workflow cancel -t default -v microblog -i dc_1 -w 1722606007
```

Note: Be sure the baseurl environment variables are set before using CLI commands. There is also an API version of this command.

The following is an example of a successful response from running the command.

```
data:
  method: PUT
  request: http://localhost:8080/api/v2/tenants/default/vnfs/microblog/vnf_instances/dc_1/workflow_instances/1722606007/cancel
  status: 200
```

Related concepts

[Openet Weaver VNF-M API Reference](#) on page 468
Openet Weaver REST API reference guides

[Openet Weaver VNF-M CLI Reference](#) on page 469
Openet Weaver CLI command reference guides

[Setting Environment Variables](#) on page 133

Managing Additional Parameters for the VNF-M GUI

You can predefine the additional_parameters parameter for any VNF or VNFC workflow that supports the parameter.

Note: See [Running VNF Workflows](#) and [Running VNFC Workflows](#) for a list of which workflows support additional_parameters.

You configure those values in a VNF-level metadata file. You upload that file to a specific VNF for a specific tenant, so the predefined values apply only to that VNF. You can download the file to update it as required.

The metadata file contains an overview section for general information about the VNF and who created the metadata file. Then there is a section for each workflow where you are predefining additional_parameter values.

When you run a workflow in the Openet Weaver VNF-M GUI, the predefined values display in the dialog box for the workflow.

Creating a VNF Metadata File

This topic describes how to configure a VNF metadata file.

A VNF metadata file has two main parts:

- VNF Overview
- Predefined additional_parameters sections

VNF Overview

The VNF overview consists of the attributes shown in the following table:

Attribute	Description	Editable	Example
vnf_meta_version	The version of the vnf meta schema used. Only the following characters are allowed: <ul style="list-style-type: none"> • Alphanumeric • Dash • Underscore • Dot 	No	1.0
vnf_id	The vnf_id of the VNF to which the metadata file is to be uploaded. Only the following characters are allowed: <ul style="list-style-type: none"> • Alphanumeric • Underscore 	No	microblog_nginx1
creation_date	The date and time when the VNF was on-boarded. The date and time is specified automatically by Openet Weaver.	No	2016-07-20 14:48:17.00197+0800
vendor	The name of the vendor. Only the following characters are allowed: <ul style="list-style-type: none"> • Alphanumeric • Dash • Underscore • Dot 	Yes	acme-apac
contact	The person or organization that created the VNF.	Yes	Acme Office
description	A text description of the VNF.	Yes	Simple VNF with webserver and database for VNF lifecycle demonstration

Uploading the VNF metadata file does not affect non-editable attributes. The original value of those attributes is preserved when new values would be different.

Workflow Configurations

The vnf_workflow_configurations and vnfc_workflow_configurations sections are optional. They are used to specify predefined data for use in the VNFM-GUI, so that default parameters can be defined and displayed in the GUI, reducing the need for you to configure JSON-formatted parameters in the GUI.

The additional_parameters parameter can be defined each VNF-level and VNFC-level workflow that supports it. Furthermore, VNFC-level parameters can be predefined for specific VNFCs.

The following shows the Workflow Configurations structures:

```

vnf_workflow_configurations:
  - <workflow parameter configuration 1>
  - <workflow parameter configuration 2>
  ...
vnfc_workflow_configurations:
  - vnfc_id: <vnfc id 1>
    configurations:
      - <workflow parameter configuration 1 for vnfc id 1>
      - <workflow parameter configuration 2 for vnfc id 1>
      ...
  - vnfc_id: <vnfc id 2>
    configurations:

```

```

- <workflow parameter configuration 1 for vnfc id 2>
- <workflow parameter configuration 2 for vnfc id 2>
...

```

The following is the structure of a workflow parameter configuration for a VNFC:

```

workflow_type: <workflow type>
additional_parameters_set:
- key: <key name>
  value: <json object value>

```

The json_object value can be a simple JSON type or complex JSON object.

Example VNF Metadata File

The following is an example of a VNF metadata file:

```

vnf_id: demo_vnf
creation_date: 2017-05-19 01:22:05.124+0800
description: A sample VNF with predefined parameter
vnf_meta_version: "1.0"

##
## vnf workflow level configurations.
##
vnf_workflow_configurations:
- workflow_type: "update_vnf"
  additional_parameters_set:
    - key: "default"
      value:
        contact:
          person:
            first_name : John
            family_name : Doe
          address:
            lines: |
              Orchid rd.
              Suite #292
            city : Royal Oak
            state : MI
            postal : 48046
    - workflow_type: "start_vnf"
      additional_parameters_set:
        - key: "openet"
          value:
            take_a_snapshot: true
            timeout: 300

##
## vnfc workflow level configurations.
##
vnfc_workflow_configurations:
- vnfc_id: "blogserver"
  configurations:
    - workflow_type: "upgrade_vnfc"

  additional_parameters_set:
    - key: "default"
      value:
        database:

```

```

        - server_name : db-1-svr
          username : dbuser
          password : dbpassword
        - server_name : db-2-svr
          username : dbmaster
          password : dbpassword
      - workflow_type: "custom_action_vnfc"
        additional_parameters_set:
          - key: "default"
            value:
              vm:
                - arch: x86_64
                  os: redhat
                  ram: 2 GB
                  host: vm1
                - arch: x86_64
                  os: opensuse
                  ram: 2 GB
                  host: vm2
          - key: "timeout param"
            value:
              process_timeout: 15
              num_of_retry: 10
    - vnfc_id: "nginx"
      configurations:
        - workflow_type: "start_vnfc"
          additional_parameters_set:
            - key: "default"
              value:
                customer_name: "voltDBMaster"
                take_a_snapshot: true
                timeout: 200
        - workflow_type: "custom_action_vnfc"
          additional_parameters_set:
            - key: "default"
              value:
                customer_name: "voltDBSlave"
                take_a_snapshot: false
                timeout: 500
                protocol: http
            - key: "secure"
              value:
                customer_name: "voltDBMaster"
                take_a_snapshot: true
                timeout: 300
                protocol: https

```

Downloading and Uploading VNF Metadata

This topic describes how to download and upload VNF metadata

Downloading VNF Metadata

To update a VNF metadata file with predefined additional_parameters, you first need to download the file. You can download the file as either JSON or a YML file, and you can do it from the template or the workspace. The syntax for downloading from the CLI is as follows:

```
ovlm-cm vnf_metadata download -t <tenant_id> -v <vnf_id> -f<vnf.json|vnf.yml>
[ -p <vnf_template_id> ]
```

The syntax for downloading using REST API is as shown below:

```
GET http://<ovlm-workflow-cm-url>/api/v2/tenants/<tenant_id>/vnfs/<vnf_id>/metadata/<vnf.json|vnf.yml>[?vnf_template_id=<vnf_template_id>]
{
}
```

Example: Downloading VNF Metadata as a JSON Formatted File from the Workspace

The following is an example of downloading VNF metadata as a JSON formatted file from the workspace using the CLI:

```
ovlm-cm vnf_metadata download -t default -v abc -f vnf.json
```

The following is an example of downloading VNF metadata as a JSON formatted file from the workspace using REST API:

```
curl -s -H "Content-Type: application/json" -X GET $OVLMCM_BASEURL/api/v2/tenants/${tenant_id}/vnfs/${vnf_id}/metadata/vnf.json
```

Example: Downloading VNF Metadata as a YML Formatted File from the Workspace

The following is an example of downloading VNF metadata as a YML formatted file from the workspace using the CLI:

```
ovlm-cm vnf_metadata download -t default -v abc -f vnf.yml
```

The following is an example of downloading VNF metadata as a YML formatted file from the workspace using REST API:

```
curl -s -H "Content-Type: application/json" -X GET $OVLMCM_BASEURL/api/v2/tenants/${tenant_id}/vnfs/${vnf_id}/metadata/vnf.yml
```

Example: Downloading VNF Metadata as a JSON Formatted File from the Template

The following is an example of downloading VNF metadata as a JSON formatted file from the template using the CLI:

```
ovlm-cm vnf_metadata download -t default -v abc -p microblog_v1 -f vnf.json
```

The following is an example of downloading VNF metadata as a JSON formatted file from the template using REST API:

```
curl -s -H "Content-Type: application/json" -X GET $OVLMCM_BASEURL/api/v2/tenants/${tenant_id}/vnfs/${vnf_id}/metadata/vnf.json?vnf_template_id=${vnf_template_id}
```

Example: Downloading VNF Metadata as a YML Formatted File from the Template

The following is an example of downloading VNF metadata as a YML formatted file from the template using the CLI:

```
ovlm-cm vnf_metadata download -t default -v abc -p microblog_v1 -f vnf.yml
```

The following is an example of downloading VNF metadata as a YML formatted file from the template using REST API:

```
curl -s -H "Content-Type: application/json" -X GET $OVLMCM_BASEURL/api/v2/tenants/${tenant_id}/vnfs/${vnf_id}/metadata/vnf.yml?vnf_template_id=${vnf_template_id}
```

Uploading VNF Metadata

After updating the VNF metadata file to include predefined additional parameters, you must upload it back to the correct tenant and VNF.

The syntax for uploading VNF metadata from the CLI is as follows:

```
ovlm-cm vnf_metadata upload -t <tenant_id> -v <vnf_id> -f<filename>
```

The syntax for uploading using REST API is as shown below:

```
POST http://<ovlm-workflow-cm-url>/api/v2/tenants/<tenant_id>/vnfs/<vnf_id>/metadata
{
    "metadata_file": "<filename>",
}
```

Example: Uploading a VNF Metadata File from the CLI

The following is an example of uploading a VNF metadata file from the CLI:

```
ovlm-cm vnf_metadata upload -t default -v abc -f vnf_sample.yml
```

In this case, a successful response should look similar to the following:

```
Uploading Files:
    vnf_sample.yml
data:
    method: POST
    request: http://10.3.18.22:9090/api/v2/tenants/default/vnfs/abc/metadata
    warning: 'The following attribute(s) has been reverted to its original value:
        creation_date, vnf_id.'
status: 201
```

Example: Uploading a VNF Metadata File Using REST API

The following is an example of uploading a VNF metadata file using REST API:

```
curl -s -XPOST -F "metadata_file=@${filename}" $OVLMCM_BASEURL/api/v2
/tenants/${tenant_id}/vnfs/${vnf_id}/metadata
```

JSON Schema for VNF Metadata

The following shows the JSON schema for VNF metadata:

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "Weaver VNF meta Schema",
    "description": "The workflow additional parameter schema specifies the
structure of additional parameters a VNF developer can specify for GUI to pick
it up.\n",
    "definitions": {
        "workflow_configurations": {
            "type": "array",
            "items": {
                "type": "object",
                "properties": {
                    "workflow_type": {
                        "type": "string"
                    },
                    "additional_parameters_set": {
                        "type": "array"
                    }
                }
            }
        }
    }
}
```

```

        "type": "array",
        "items": {
            "type": "object",
            "properties": {
                "key": {
                    "type": "string"
                },
                "value": {
                    "type": "object"
                }
            }
        },
        "required": [
            "key",
            "value"
        ],
        "additionalProperties": true,
        "uniqueItems": true
    }
},
"required": [
    "workflow_type",
    "additional_parameters_set"
],
"additionalProperties": true,
"uniqueItems": true
},
"additionalProperties": true,
"uniqueItems": true,
"minItems": 0
}
},
"type": "object",
"properties": {
    "vnf_meta_version": {
        "description": "The version of the vnf meta schema used.",
        "type": "string",
        "pattern": "^[A-Za-z0-9\\-_.]+$"
    },
    "vendor": {
        "description": "The vendor identifier.",
        "type": "string",
        "pattern": "^[A-Za-z0-9\\-_.]+$"
    },
    "vnf_id": {
        "description": "The vnf id that identifies the vnf without its
version.",
        "type": "string",
        "pattern": "^[A-Za-z0-9_]+$"
    },
    "contact": {
        "description": "The person or organization that created the Vnf.",
        "type": "string"
    },
    "creation_date": {
        "description": "A date time when the vnf is on-boarded and the
value will be filled by the Weaver automatically",
        "type": "string"
    },
    "description": {
        "description": "An arbitrary description of the VNF."
    }
}

```

```

        "type": "string"
    },
    "vnf_workflow_configurations": {
        "$ref": "#/definitions/workflow_configurations"
    },
    "vnfc_workflow_configurations": {
        "description": "A list of configurations used to manage this VNFC per vnfc workflow.",
        "type": "array",
        "items": {
            "type": "object",
            "properties": {
                "vnfc_id": {
                    "description": "an unique vnfc identifier per vnf.",
                    "type": "string"
                },
                "configurations": {
                    "$ref": "#/definitions/workflow_configurations"
                }
            }
        }
    }
},
"required": [
    "vnf_id",
    "creation_date",
    "vnf_meta_version"
],
"additionalProperties": true
}

```

Configuring Workflows to Retry Automatically

This topic describes how to change workflow retry setting using the Workflow Engine Dashboard.

Some Openet Weaver has a retry mechanism for running workflows that allows you to specify that applicable workflows are retried automatically at run time. This is a global setting, and by default the value is set so that workflows are not retried. That is because In most cases, workflow errors are caused by a configuration issue, so there is little point of performing a retry before reviewing where the error occurred. However, there can be instances when a workflow error is caused for other reasons. For example, a workflow error might occur because a VM is slow to restart after a timeout, but might be up at the time the retry is run.

The trade-off is that it takes longer for a workflow to fail and retry with subsequent failures than it does for a workflow to fail once and terminate. You need to determine which option is more advantageous in your deployment.

The following table shows the applicable workflows support the retry mechanism. It also shows the workflows that are hard-coded to retry once. For the configurable workflows, the retry value can be set from 0 through 9, which specifies how many times a workflow will retry before terminating. So, for example, setting retry to 1 means that applicable workflows will potentially try to run twice—the original attempt and one retry.

Workflow Type	Default Value	Configurable
is_vnf_up	1	No
is_vnf_healthy	1	No
start_vnfc	0	Yes
stop_vnfc	0	Yes
instantiate_vnf	0	Yes

Workflow Type	Default Value	Configurable
instantiate_vnf_without_vim	0	Yes
start_vnf	0	Yes
stop_vnf	0	Yes
terminate_vnf	0	Yes
terminate_vnf_without_vim	0	Yes

To change the retry value for the configurable workflows listed in the above table, follow these steps:

1. Open the Workflow Engine Dashboard in a web browser. The url is: `http://<workflow_engine_host>:<workflow_engine_port>`

If you are not sure of the host and port settings, refer to the Workflow Engine Configuration section of the installation file. The following is an example:

```
# Workflow Engine Configuration
workflow_engine:
  hosts:
    - ovlm-mgmt-dub-all-lane.openet-dublin
  properties:
    server:
      port: 28083
      protocol: http
```

Note: See [Creating an Installation File](#) for more information about the installation file.

2. Select Content Management in the Navigation pane.

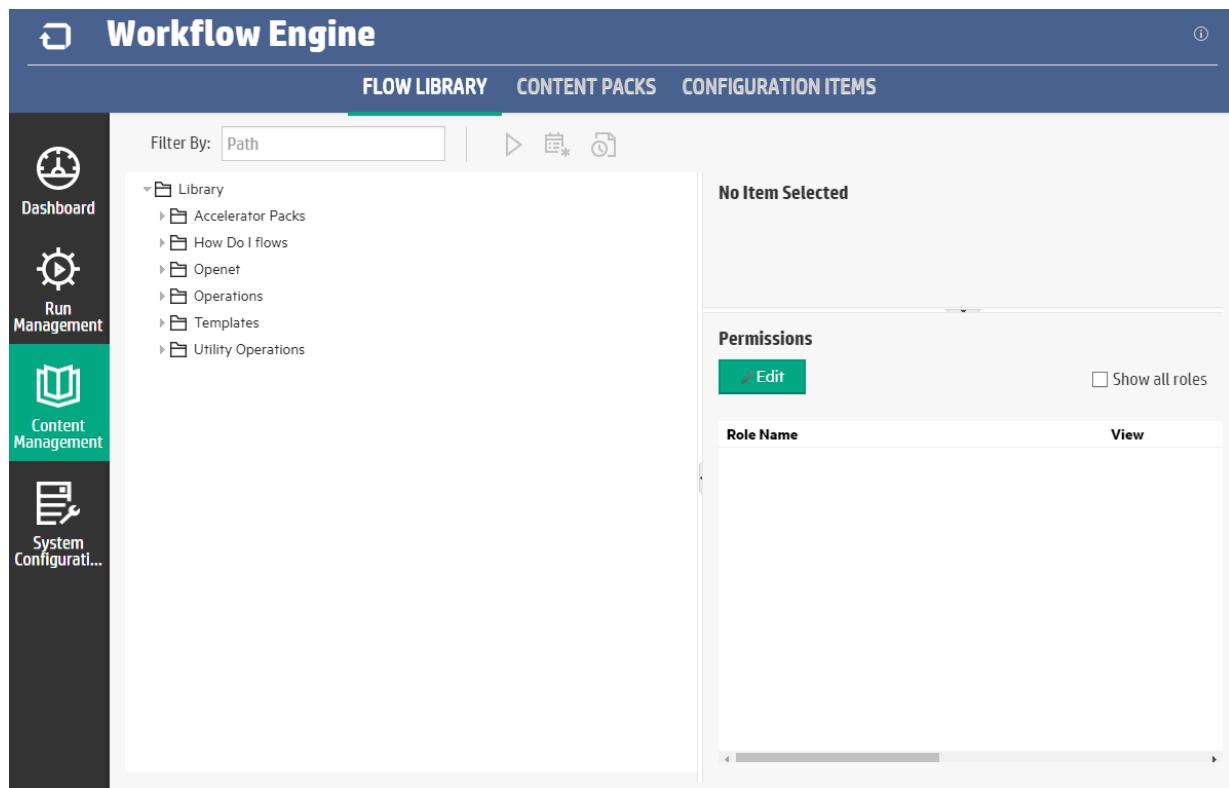


Figure 156: Content Management in Workflow Engine Navigation Pane

3. click Configuration Items from the top menu.

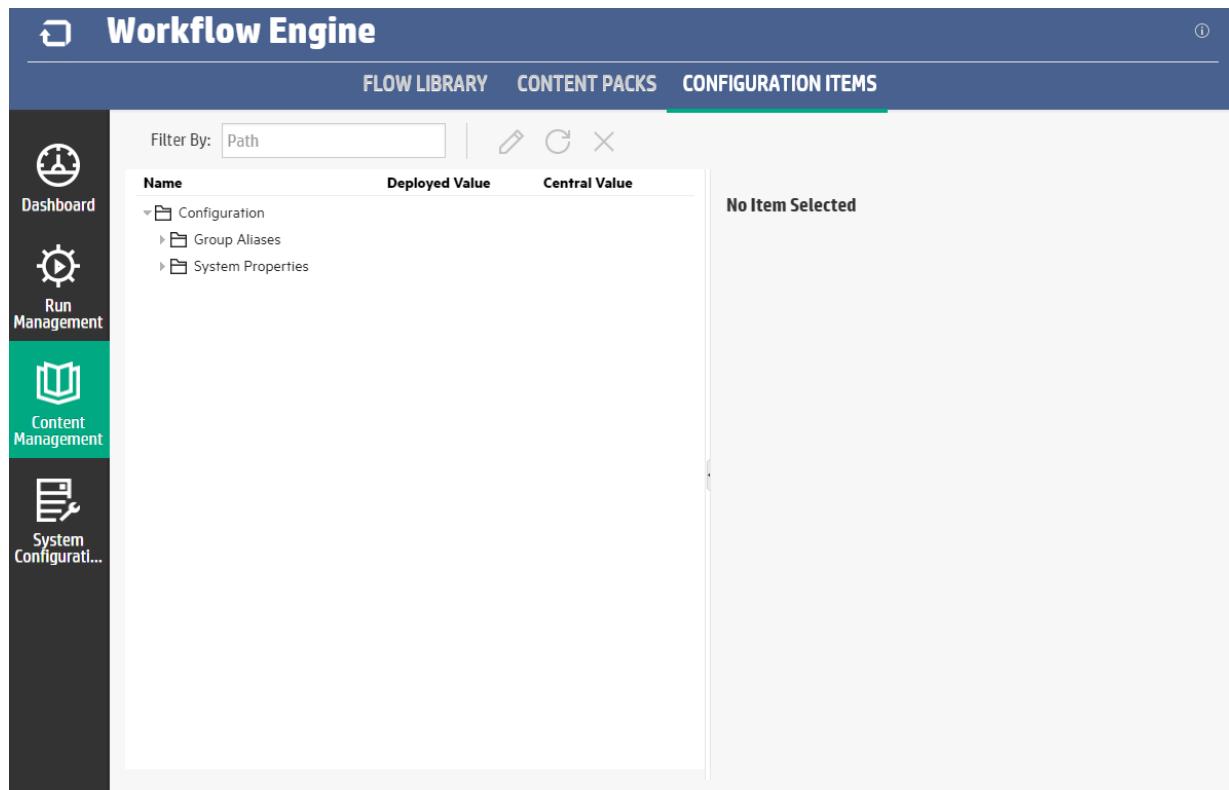


Figure 157: Configuration Items Menu

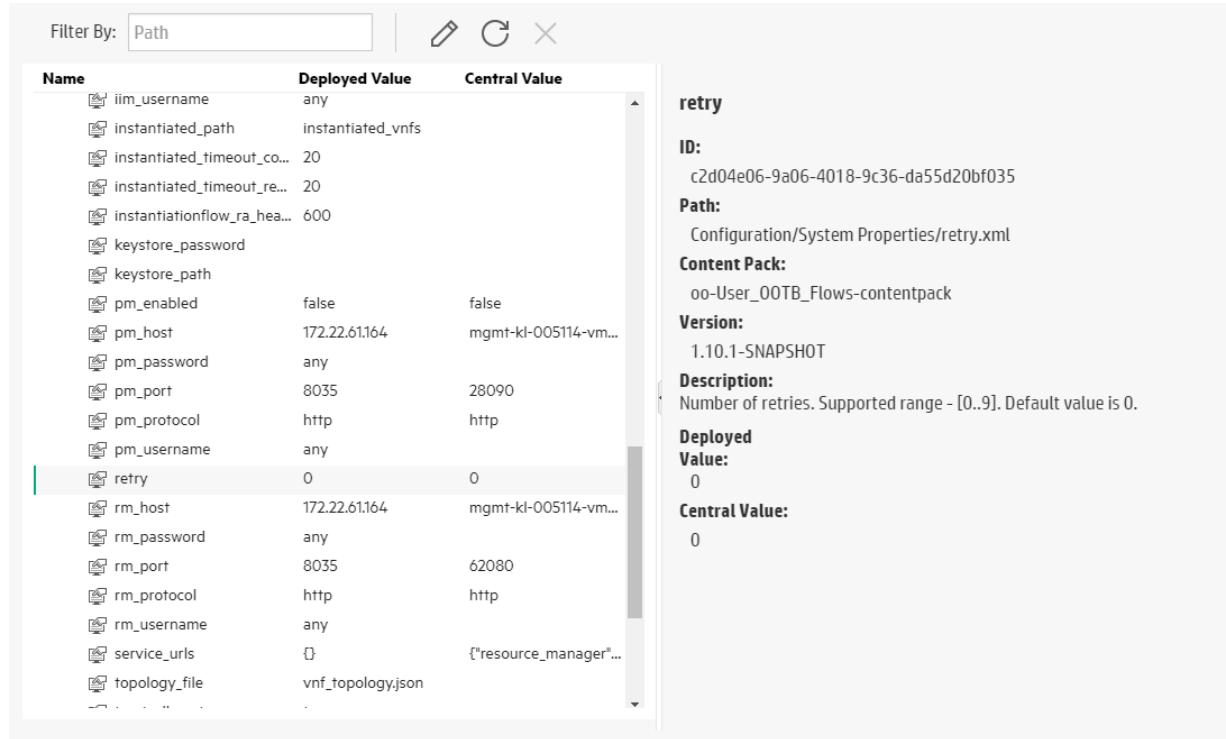
4. Expand **System Properties** by clicking the arrow to the left of it.

Name	Deployed Value	Central Value
Configuration		
Group Aliases		
System Properties		
iim		
ra		
val		
we		
cm_host	172.22.61.164	10.3.18.16
cm_password	any	
cm_port	8035	62082
cm_protocol	http	http
cm_rsync_port	873	62081
cm_username	any	
debug	OFF	
dm_host	172.22.61.164	10.3.18.16
dm_password	any	
dm_port	8035	28130
dm_protocol	http	http
dm_username	any	
fa_enabled	false	
fa_host	172.22.61.164	

Figure 158: System Properties Expanded

5. Locate and select retry under System Properties.

You might need to scroll to find the selection.



The screenshot shows a system properties grid with a toolbar at the top. The grid has columns for Name, Deployed Value, and Central Value. A vertical scroll bar is visible on the right. The 'retry' property is selected, highlighted with a blue border. To the right of the grid, detailed information about the 'retry' property is displayed:

- retry**
- ID:** c2d04e06-9a06-4018-9c36-da55d20bf035
- Path:** Configuration/System Properties/retry.xml
- Content Pack:** oo-User_OOTB_Flows-contentpack
- Version:** 1.10.1-SNAPSHOT
- Description:** Number of retries. Supported range - [0..9]. Default value is 0.
- Deployed Value:** 0
- Central Value:** 0

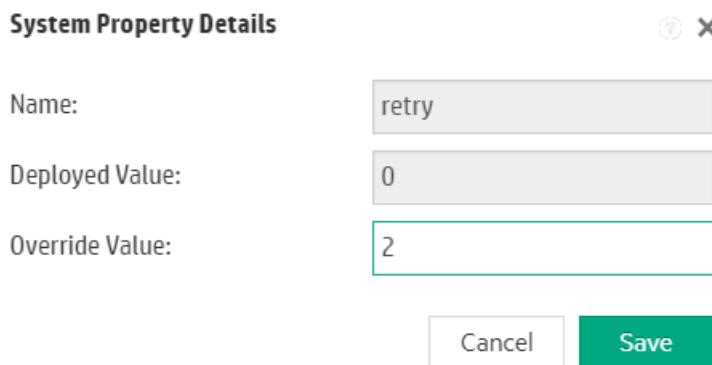
Figure 159: Selecting Retry

6. Click the Edit icon



in the tool bar.

The **System Property Details** dialog box is displayed.



The dialog box has fields for Name, Deployed Value, and Override Value. The 'Name' field contains 'retry', 'Deployed Value' contains '0', and 'Override Value' contains '2'. The 'Override Value' field is highlighted with a green border. At the bottom are 'Cancel' and 'Save' buttons.

Name:	retry
Deployed Value:	0
Override Value:	2

Figure 160: System Property Details Dialog Box

7. In the **Override Value field, enter a new value for the number of retries the workflow performs when the first attempt to run the workflow fails.**

The following rules apply to values entered in the **Override Value** field:

- if the field has no value (blank), the default value 0 is used
- if the value entered is less than 0, the default value 0 is used
- if the value entered is a numeric value greater than 9, the value 9 is used

- if the value entered is a non-numeric value, the default value 0 is used.

8. Click Save.

Synchronizing Performance Manager and Fault Manager Manually

Under normal circumstance are not required to synchronize Performance Manager or Fault Manager manually. They sync automatically as part of running any of the following workflows:

- Instantiation
- Upgrade
- Update

However, in some cases synchronization might fail and an error message is returned. The following is an example of a message relating to a Performance Manager synchronization error.

```
{
  "status":201,
  "data":{
    "request":"http://ovlm-mgmt-dub-all-lane.openet-dublin:28085/api/v2/tenants/default/vnfs/microblog_v2/vnf_instances/dc_3/workflow_instances",
    "method":"POST",
    "result":"VNF Instantiation passed successfully on all servers.",
    "warning":"Sync Performance Policy completed with warning:
[{\\"vnfc_id\\":\\"blogserver\\",\"code\":\"NO_CHANGE_APPLIED\",\"message\":\"There are no policy changes applied to the vnfc\"}, {\\"vnfc_id\\":\\"postgres\\",\"code\":\"NO_CHANGE_APPLIED\",\"message\":\"There are no policy changes applied to the vnfc\"}, {\\"vnfc_id\\":\\"nginx\\",\"code\":\"NO_CHANGE_APPLIED\",\"message\":\"There are no policy changes applied to the vnfc\"}]"
  }
}
```

The following is an example of a Performance Manager synchronization error.

```
{
  "status":201,
  "data":{
    "request":"http://ovlm-mgmt-dub-all-lane.openet-dublin:28085/api/v2/tenants/default/vnfs/microblog_v2/vnf_instances/dc_3/workflow_instances",
    "method":"POST",
    "result":"VNF Instantiation passed successfully on all servers.",
    "warning":"Sync Fault Policy completed with warning:
{\\"nginx\\": [{\"error_code\":\"NO_CHANGE_APPLIED\", \"error\":\"There are no policy changes applied to the vnfc\"}], \\"blogserver\\": [{\"error_code\":\"NO_CHANGE_APPLIED\", \"error\":\"There are no policy changes applied to the vnfc\"}], \\"postgres\\\": [{\"error_code\":\"NO_CHANGE_APPLIED\", \"error\":\"There are no policy changes applied to the vnfc\"}]}"
  }
}
```

When you see either of these errors, synchronize the corresponding microservice manually so that it is part of the VNF lifecycle. The steps you take depend on which microservice you are synchronizing.

Synchronizing Performance Manager Manually

To synchronize Performance Manager manually with a VNF instance follow these steps:

Note: Be sure the baseurl environment variables are set before using CLI commands. There are also API versions of the commands used in this topic.

1. Create a performance policy file with the name policy.yml. Include the following content.

```
SimpleScalingPolicy:
    - policyName: UpperCPUThresholdExceeded
      metricName: cpu_usage
      timePeriod: 360
      coolDownPeriod: 730
      averageUpperThreshold: 80
      triggerWorkflowID: scaleOutVNF
      workflowInputTags:
        - scaleFactor: 2
      healthStatusPriority: high

    - policyName: UpperMemoryThresholdExceeded
      metricName: memory_usage
      timePeriod: 60
      coolDownPeriod: 130
      averageUpperThreshold: 70
      triggerWorkflowID: scaleOutVNF
      workflowInputTags:
        - scaleFactor: 2
      healthStatusPriority: high

    - policyName: UpperMemoryThresholdExceeded
      metricName: memory_usage
      timePeriod: 60
      coolDownPeriod: 120
      averageUpperThreshold: 60
      triggerWorkflowID: no
      workflowInputTags:
        - scaleFactor: 2
      healthStatusPriority: medium

    - policyName: LowerCPUThresholdExceeded
      metricName: cpu_usage
      timePeriod: 360
      coolDownPeriod: 720
      averageLowerThreshold: 30
      triggerWorkflowID: scaleInVNF
      workflowInputTags:
        - scaleFactor: 1
      healthStatusPriority: low

    - policyName: LowerCPUThresholdExceeded
      metricName: memory_usage
      timePeriod: 60
      coolDownPeriod: 180
      averageLowerThreshold: 10
      triggerWorkflowID: scaleInVNF
      workflowInputTags:
        - scaleFactor: 1
      healthStatusPriority: low
```

2. Upload the policy.yml file you created in [step 1](#) to Configuration Manager in the appropriate VNFC workspace using the command `ovlm-cm policy upload -t tenant_id -v vnf_id -c vnfc_id -y policy_type -f policy_file`

The following example uploads a performance policy file to the nginx VNFC for the default tenant microblog_v2 VNF.

```
ovlm-cm policy upload -t default -v microblog_v2 -c nginx -y performance -f performance.yml.
```

3. Create a package using the command `ovlm-fe template create -t tenant_id -v vnf_id -p vnf_template_id`

The package is used in the sync command you run in [step 4](#).

The following example creates a package for the default tenant microblog_v2 VNF.

```
ovlm-cm template create -t default -v microblog_v2 -p pm_policy_on_package
```

4. Synchronize Performance Manager with a VNF instance using the command `ovlm-pm configuration sync -t tenant_id -v vnf_id -i vnf_instance_id -p vnf_template_id`

The following command synchronizes performance manager with the dc_i instance of the microblog_v2 VNF.

```
ovlm-pm configuration sync -t default -v microblog_v2 -i dc_1 -p pm_policy_on_package
```

The following is an example of the successful result of running the command.

```
data:
  method: POST
  request: http://ovlm-mgmt-dub-all-lane.openet-dublin:28090/api/v2/tenants/default/vnfs/microblog_v2-sync
  status: 200
```

Related concepts

[Setting Environment Variables](#) on page 133

[Openet Weaver VNF-M API Reference](#) on page 468

Openet Weaver REST API reference guides

[Openet Weaver VNF-M CLI Reference](#) on page 469

Openet Weaver CLI command reference guides

Synchronizing Fault Manager Manually

To synchronize Fault Manager manually with a VNF instance follow these steps:

Note: Be sure the baseurl environment variables are set before using CLI commands. There are also API versions of the commands used in this topic.

1. Create a fault policy file with the name fault.yml. Include the following content.

```
SimpleFaultPolicies:
  - policyName : VNFCDown
    triggerWorkflowID : heal_vnfc
    workflowInputTags :
      hostname: "${hostname}"
    healthStatusPriority : high
    gracePeriod : 90
    notificationType: "1.3.6.1.4.1.7898.1.8.5"
    rules:
      - oid: '1.3.6.1.4.1.2021.7898.10.2.1'
        value : "is_vnfc_up"
      - oid: '1.3.6.1.4.1.2021.7898.10.4.1'
        value : "1"
```

2. Upload the fault.yml file you created in [step 1](#) to Configuration Manager in each VNFC for the tenant VNF using the command `ovlm-cm policy upload -t tenant_id -v vnf_id -c vnfc_id -y policy_type -f policy_file`

The following example uploads a fault policy file to the nginx VNFC for the default tenant microblog_v2 VNF.

```
ovlm-cm policy upload -t default -v microblog_v2 -c nginx -y fault -f fault.yml
```

3. Create a package using the command `ovlm-fe template create -t tenant_id -v vnf_id -p vnf_template_id`

The package is used in the sync command you run in [step 4](#).

The following example creates a package for the default tenant microblog_v2 VNF.

```
ovlm-cm template create -t default -v microblog_v2 -p fm_policy_on_package
```

4. Synchronize Fault Manager with a VNF instance using the command `ovlm-pm configuration sync -t tenant_id -v vnf_id -i vnf_instance_id -p vnf_template_id`

The following command synchronizes fault manager with the dc_i instance of the microblog_v2 VNF.

```
oovlm-fm configuration sync -t default -v microblog_v2 -i dc_1 -p fm_policy_on_package
```

The following is an example of the successful result of running the command.

```
data:
  method: POST
  request: http://ovlm-mgmt-dub-all-lane.openet-dublin:28100/api/v2/tenants/default/vnfs/microblog_v2-sync
  status: 200
```

Related concepts

[Setting Environment Variables](#) on page 133

[Openet Weaver VNF-M API Reference](#) on page 468

Openet Weaver REST API reference guides

[Openet Weaver VNF-M CLI Reference](#) on page 469

Openet Weaver CLI command reference guides

Listing Workflows

You can display a list of workflows per tenant using the following command:

```
ovlm-fe metadata list_workflow_type -t tenant_id
```

Note: Be sure the baseurl environment variables are set before using CLI commands. There is also an API version of this command.

The following is an example result from running the `metadata list_workflow_type` command:

```
data:
  method: GET
  request: http://localhost:28050/api/v2/metadata/workflow_type
  workflow_type_list:
    - custom_action_vnfc
    - heal_vnfc
    - health_status
    - instantiate_vnf
    - instantiate_vnf_without_vim
```

```

- instantiated_vnf_list
- is_vnf_healthy
- is_vnf_up
- rollback_vnf
- scale_in_vnf
- scale_out_vnf
- start_vnf
- stop_vnf
- terminate_vnf
- terminate_vnf_without_vim
- update_vnf
- upgrade_vnf
status: 200

```

Related concepts

[Setting Environment Variables](#) on page 133

[Openet Weaver VNF-M API Reference](#) on page 468
Openet Weaver REST API reference guides

[Openet Weaver VNF-M CLI Reference](#) on page 469
Openet Weaver CLI command reference guides

Checking Workflow Status

You use the workflow-status command to get the status of an executed flow. The output of the command is a summary of current status of running flow or, if the run is stopped or complete, the final state of the workflow. The CLI command syntax is `ovlm-fe workflow status -t <tenant_id> -v <vnf_id> -i <vnf_instance_id> -w <workflow_instance_id>`

Note: Be sure the baseurl environment variables are set before using CLI commands. There is also an API version of this command.

The workflow-status parameters are described in the following table.

Table 43: workflow status Parameters

CLI Flag	Parameter	Mand.	Description
-t	tenant_id	Yes	The tenant identifier as created in the Configuration Manager.
-v	vnf_id	Yes	The VNF identifier as created in the Configuration Manager.
-i	vnf_instance_id	Yes	The VNF instance identifier as created in the Configuration Manager.
-w	workflow_instance_id	Yes	The workflow identifier.

The following is an example of a workflow status command run from the front end CLI:

```
ovlm-fe workflow status -t default -v microblog -i dc_1 -w 101100291
```

The API equivalent of the get status command syntax is as follows:

```
curl -X GET http://<apiHost:port>/api/v2/tenants/<tenant_id>/vnfs/<vnf_id>/vnf_instances/<vnf_instance_id>/\ workflow_instances/<workflow_instance_id>
```

Where apiHost:port is the host and port where the API is deployed.

The following is an example of an API call to check the status of a workflow: url -X GET http://server01:28888/api/v2/tenants/default/vnfs/microblog/vnf_instances/dc_1/workflow_instances/101100291

The resulting output from running the workflow status command contains a workflow_instance_status which might be one of the following:

- RUNNING—the workflow is still running.
- COMPLETED—the workflow completed successfully.
- FAILED—the workflow completed with errors. Error information is provided in the Detail field.
- CANCELLED—the workflow was cancelled by the user. The Detail field shows a general error and the Error Status field shows 500. You can run the CLI command with the -raw parameter to obtain further detail about the state.
- PAUSED—the workflow was paused by Workflow Engine. this might be due to missing parameters or it might mean that other action is required, The Detail field shows general error and Error Status field shows 500. You can run the CLI command with the -raw parameter to obtain further detail about the state.

Example workflow status Result for Running Workflow

```
data:
  method: GET
  request: http://localhost:8080/api/v2/tenants/default
/vnfs/default_vnf/vnf_instances/ResourceManager/workflow_instances
/101100291
  workflow_instance_status: RUNNING
status: 200
```

Example workflow status Result for Cancelled Workflow

```
{"data": {"workflow_instance_status": "CANCELED", "error": "The
execution of the specified operation or workflow failed.
Please contact your system administrators", "error
_code": "EXECUTE_WORKFLOW_FAILED", "request": "http://localhost:8080/
api/v2/tenants/
default/vnfs/microblog/vnf_instances/dc_1/workflow
_instances/112600094", "method": "GET"}, "status": 500}
Error Status: 500.
Error Code: EXECUTE_WORKFLOW_FAILED.
Detail: The execution of the specified operation or workflow failed.
Please contact your system administrators
```

Related concepts

[Setting Environment Variables](#) on page 133

[Openet Weaver VNF-M API Reference](#) on page 468

Openet Weaver REST API reference guides

[Openet Weaver VNF-M CLI Reference](#) on page 469

Openet Weaver CLI command reference guides

Listing Instantiated VNFs

You can display all instantiated instances of a VNF using the following CLI command `ovlm-fe instantiated_vnf list -t tenant -v vnf_id`.

Note: Be sure the baseurl environment variables are set before using CLI commands. There is also an API version of this command.

The command returns a list of vnf instances that have been instantiated and the running status of each. The following is an example of the command in use:

```
ovlm-fe instantiated_vnf list -t default -v microblog
```

Running the command results in the following:

```
data:
  flow_result:
    Result:
      hosts:
        rt1-ovlm-vm:
          vnf_instance_id: dc_1
          vnf_template_id: multi_nodes_template
          vnfc_id: nginx
          vnfc_instance_id: '1'
        rt2-ovlm-vm:
          vnf_instance_id: dc_1
          vnf_template_id: multi_nodes_template
          vnfc_id: blogserver
          vnfc_instance_id: '2'
        rt3-ovlm-vm:
          vnf_instance_id: dc_1
          vnf_template_id: multi_nodes_template
          vnfc_id: blogserver
          vnfc_instance_id: '3'
        rt4-ovlm-vm:
          vnf_instance_id: dc_1
          vnf_template_id: multi_nodes_template
          vnfc_id: postgres
          vnfc_instance_id: '4'
        tenant_id: default
        vnf_id: microblog
    method: GET
    request: http://ovlm-vm:8080/api/v2/health/status
    workflow_instance_status: COMPLETED
  status: 200
```

Related concepts

[Setting Environment Variables](#) on page 133

[Openet Weaver VNF-M API Reference](#) on page 468
Openet Weaver REST API reference guides

[Openet Weaver VNF-M CLI Reference](#) on page 469
Openet Weaver CLI command reference guides

Deleting a VNF

You can delete a VNF including all its files and directories from Configuration Manager from the CLI, The VNF-M GUI or by using an API command

You can delete a VNF using the CLI command `ovlm-cm vnf delete -t tenant_id -v vnf_id`

Running the `vnf delete` command removes the VNF including all its files and directories from Configuration Manager. the following is an example of the command.

```
ovlm-cm vnf delete -t default -v webserver
```

Note: Be sure the baseurl environment variables are set before using CLI commands. There is also an API version of this command.

The output of the command should be similar to the following.

```
data:
  method: DELETE
  request: http://localhost:9090/api/v2/tenants/default/vnfs/webserver
status: 200
```

To Delete A VNF using the VNF-M GUI follow these steps:

1. On the VNF Lifecycle page, locate the inactive VNF you intend to delete and click the delete icon



Scroll down the page if required.

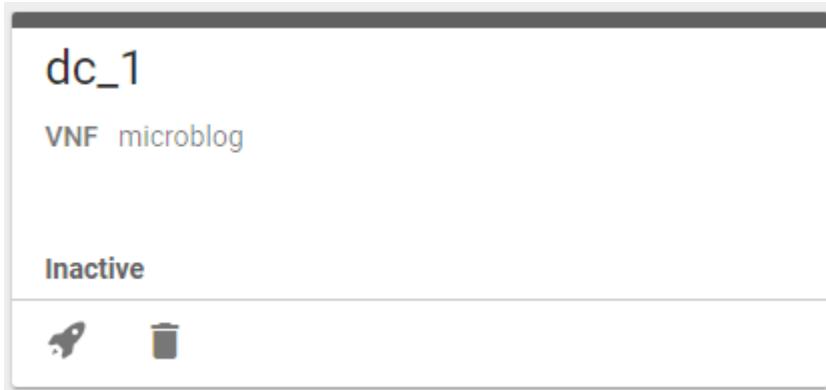


Figure 161: Inactive VNF

The delete VNF confirmation box is displayed.

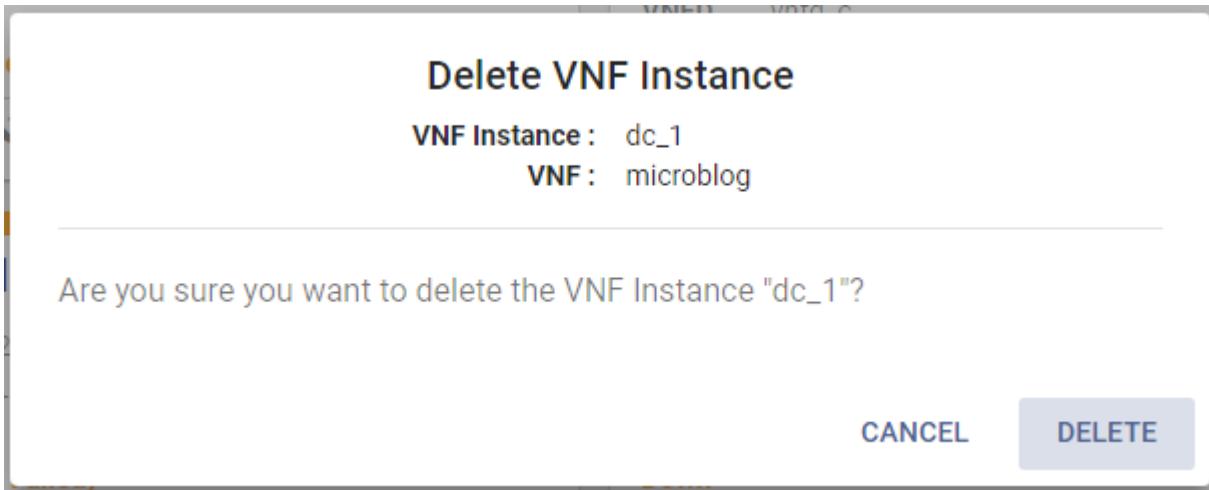


Figure 162: Delete VNF Confirmation Box

2. Click **Confirm** to delete the VNF and close the confirmation box.

Managing Performance

The Openet Weaver Performance Manager microservice provides performance management functionality, for example:

- Collecting Performance metrics such as CPU and memory usage
- Reporting on VNF performance health

- Managing performance issues, for example triggering scalability flows when a CPU or memory threshold set in the performance metadata is breached

Note: Scalability flows and their triggering will be available in a future release of Openet Weaver when the VIM Abstraction Layer is integrated.

The following shows the interaction between the Openet Weaver Performance Manager, Workflow Engine, Configuration Manager, and a Resource Agent.

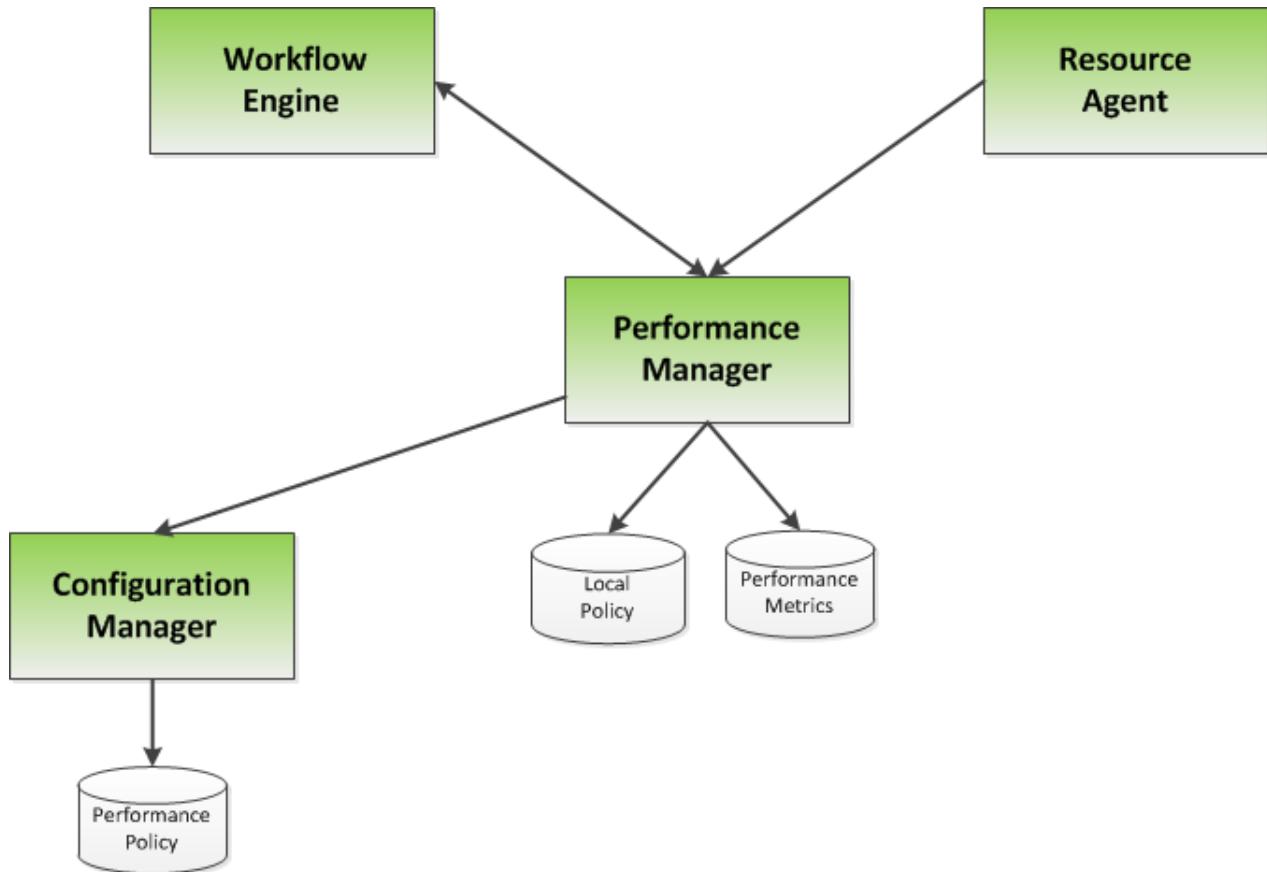


Figure 163: Performance Manager Interaction

Note: To enable Performance Manager, the following Resource Agent attributes must be enabled and configured in the installation file either before deployment or before running the deployment script for configuration changes to Openet Weaver.

The microservices communicate with each other through API calls.

Each VNF has its own set of performance policies associated with it. The policies are stored in YML metadata files within the VNF package structure in Configuration Manager. Openet Weaver provides CLI and API commands to upload and remove performance policy metadata files to and from VNFs. Performance metrics are stored in Performance Manager.

Managing Performance Policies

Performance policy files are stored and managed through Configuration Manager. Openet Weaver provides API and CLI commands that allow you to perform the following performance policy file management tasks:

Uploading a Performance Policy

You can upload a performance policy file to a VNF in Configuration Manager using the command `ovlm-cm policy upload -y policy_type -t tenant_id -v vnf_id -c vnfcs_id -f policy_file` command, for example.

```
ovlm-cm policy upload -y performance -t anycom -v webserver -c httpd -f
performance.yml
```

The following is an example of a successful response.

```
data:
  method: POST
  request: http://localhost:9090/api/v2/tenants/anycom/vnfs/webserver/vnfcs/h
  ttpd/performance
  status: 200
```

Performance policies in a performance policy metadata file have the following structure.

```
<PolicyGroupName>
  - policyName : <policyNameString>
  metricName : <metricNameString>
  timePeriod : <NumberOfSeconds>
  averageUpperThreshold or averageLowerThreshold : <MetricValue>
  triggerWorkflowID : <WorkflowIDString>
  healthStatusPriority : "high|medium|low"
```

The following table lists the performance policy parameters. All parameters are mandatory.

Table 44: Performance Policy Parameters

Parameter	Type	Unit of Measurement	Restrictions	Description
PolicyGroupName	String			The group to which the policy belongs.
policyName	String			The name of the policy. The value of this parameter must be unique.
metricName	String			The metric to analyze. The metric name must be one of the supported names listed in Table 2 .
timePeriod	String	seconds	Value must be greater than 0.	The time period over which an average upper or lower threshold is breached before a workflow is triggered.
averageUpperThreshold averageLowerThreshold	String	value	Value must be greater than 0.	The average upper or lower threshold for the performance policy, depending on which parameter is used. Only one of these parameters is used per performance policy. When this value is breached for the amount of time specified in the timePeriod parameter, the workflow specified in triggerWorkflowID is triggered.
triggerWorkflowID	String			Name of existing Openet Weaver workflow to be triggered, when the policy is breached. Note: Performance Manager will be capable of triggering workflows in future releases of Openet Weaver
healthStatusPriority	String		high medium low	The priority of the rule.

Parameter	Type	Unit of Measurement	Restrictions	Description
coolDownPeriod	String	seconds	> timePeriod * 2	<p>Specifies the amount of time, in seconds, to wait before another workflow can be triggered. However, Performance Manager can not trigger scaling workflows during the cool down period.</p> <p>The value should be twice the length of the value of the timePeriod parameter.</p>

The following table lists the supported metrics that can be measured currently by Performance Manager.

Table 45: Supported Performance Manager Metrics

Metric Name	Description
cpu_usage	CPU usage over the last 0.1 second.
cpu_5min_usage	CPU usage over the last 5 minutes.
memory_usage	Instant system memory usage.
swap_usage	OS swap usage.
disk_usage	Disk usage: applied to "/" location.

The following is an example of a performance policy metadata file:

```

SimpleScalingPolicy:
  - policyName : UpperCPUThresholdExceeded
    metricName : cpu_usage
    timePeriod: 360
    averageUpperThreshold: 80
    triggerWorkflowID : scaleOutVNF
    coolDownPeriod: 10000
    workflowInputTags :
      scaleFactor : 2
    healthStatusPriority : high

  - policyName : UpperMemoryThresholdExceeded
    metricName : memory_usage
    timePeriod: 60
    averageUpperThreshold: 70
    triggerWorkflowID : scaleOutVNF
    coolDownPeriod: 10000
    workflowInputTags :
      scaleFactor : 2
    healthStatusPriority : high

  - policyName : UpperMemoryThresholdExceeded
    metricName : memory_usage
    timePeriod: 60
    averageUpperThreshold: 60
    triggerWorkflowID : no
    coolDownPeriod: 10000
    workflowInputTags :
      scaleFactor : 2
    healthStatusPriority : medium

  - policyName : LowerCPUThresholdExceeded
    metricName : cpu_usage
  
```

```

timePeriod: 360
averageLowerThreshold: 30
triggerWorkflowID : scaleInVNF
workflowInputTags :
  scaleFactor : 1
healthStatusPriority : low

- metricName : LowerMemoryThresholdExceeded
  timePeriod: 60
  averageLowerThreshold: 10
  triggerWorkflowID : scaleInVNF
  coolDownPeriod: 10000
  workflowInputTags :
    scaleFactor : 1
  healthStatusPriority : low

```

In the example above, breaches are reported on the following policies when the described criteria is met:

- **UpperCPUThresholdExceeded**: breach reported when the averageUpperThreshold of the `cpu_usage` metric is above 80% over 360 seconds
- **UpperMemoryThresholdExceeded**: breach reported when the averageUpperThreshold of `memory_usage` is above 70% over 60 seconds
- **LowerCPUThresholdExceeded**: breach reported when the averageLowerThreshold of the `cpu_usage` metric is below 30% over 360 seconds
- **LowerMemoryThresholdExceeded**: breach reported when the averageLowerThreshold of `memory_usage` is below 10% over 60 seconds

Downloading a Performance Policy

To download a performance policy file use the command `ovlm-cm policy upload -y policy_type -t tenant_id -v vnf_id -c vnf_id -f performance.yml|performance.json`

The following is an example of running the `policy download` command.

```
ovlm-cm policy download -y performance -t anycom -v webserver -c httpd -f performance.yml
```

The following is an example of a successful response

```

Content of policy file in respective format yml|json based on the request:

impleScalingPolicy:
  - policyName : UpperCPUThresholdExceeded
    metricName : cpu_usage
    timePeriod: 360
    averageUpperThreshold: 1
    triggerWorkflowID : scaleOutVNF
    workflowInputTags :
      - scaleFactor : 2
    healthStatusPriority : high

```

Disabling a Policy on a VNF Instance

By default policies are enabled on microservice startup. You can disable policies for a VNF instance using the command `ovlm-pm policy disable -t tenant_id -v vnf_id -i vnf_instance_id`.

This setting is not persisted. Therefore all instances revert to the default setting of policies enabled if the microservice is restarted.

The following is an example of policies being disabled for the dc_1 instance of the microblog VNF.

```
ovlm-pm policy disable -t anycom -v microblog -i dc_1
```

The following is an example response from running the command successfully.

```
data:
method: PUT
request: http://localhost:8080/api/v1/tenants/anycom/vnfs/microblog/vnf_instances/dc_1/policy/disable
status: 200
```

Enabling a policy

To enable a disabled policy you use the command `ovlm-pm policy enable -t tenant_id -v vnf_id -i vnf_instance_id`.

The following is an example of policies being enabled for the dc_1 instance of the microblog VNF.

```
ovlm-pm policy enable -t anycom -v microblog -i dc_1
```

The following is an example response from running the command successfully.

```
data:
method: PUT
request: http://localhost:8080/api/v1/tenants/anycom/vnfs/microblog/vnf_instances/dc_1/policy/enable
status: 200
```

Getting Policy Status

When you are not sure whether policies are disabled on an instance you can investigate using the command `ovlm-pm policy status -t tenant_id -v vnf_id -i vnf_instance_id`.

The following queries the policy status of the dc_1 instance of the microblog VNF.

```
ovlm-pm policy status -t anycom -v microblog -i dc_1
```

The following response shows that policies are enabled for the instance.

```
data:
method: GET
request: http://localhost:8080/api/v1/tenants/anycom/vnfs/microblog/vnf_instances/dc_1/policy/status
enabled: true, status: 200
```

Deleting a Performance Policy

To delete all the performance policy files for a specific VNFC use the command `ovlm-cm policy delete -y policy_type -t tenant_id -v vnf_id -c vnfc_id`

The following is an example of running the `policy delete` command.

```
ovlm-cm policy delete -y performance -t anycom -v webserver -c httpd
```

The following is an example of a successful response

```
data:
method: DELETE
request: http://localhost:9090/api/v2/tenants/anycom/vnfs/webserver/vnfcsh
```

```
ttpd/policies/performance
status: 200
```

Checking VNF Health

You can check whether a VNF is healthy by running the CLI command `workflow submit -t tenant_id -v vnf_id -i instance_id -y is_vnf_healthy -p vnf_template_id`

Note: Be sure the base url environment variables are set before using CLI commands. See [setting environment Variables](#) for more information. See the [API reference](#) for the equivalent API command.

The following example checks whether the specified instance of the microblog VNF for the default tenant is up and is free from performance issues.

```
ovlm-fe workflow submit -t default -v microblog -i dc_1 -y is_vnf_healthy
-p '{"vnf_template_id": "microblog_v1"}'
```

The output of the command is a workflow_instance_id generated by the system that is similar to the following example:

```
data:
  method: POST
  request: http://ovlm-vm:8080/api/v1/tenants/default/vnfs/microblog/vnf_instances/dc_1/workflow_instances
  workflow_instance_id: '112600123'
  status: 200
```

The `is_vnf_healthy` workflow is shown below.

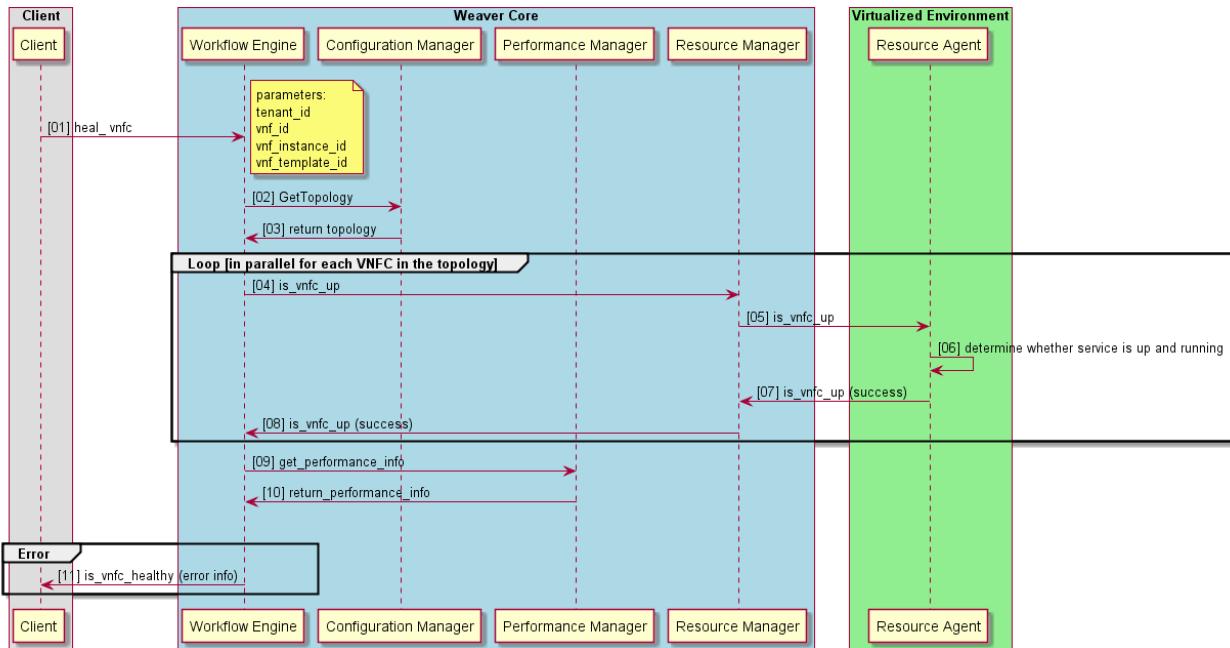


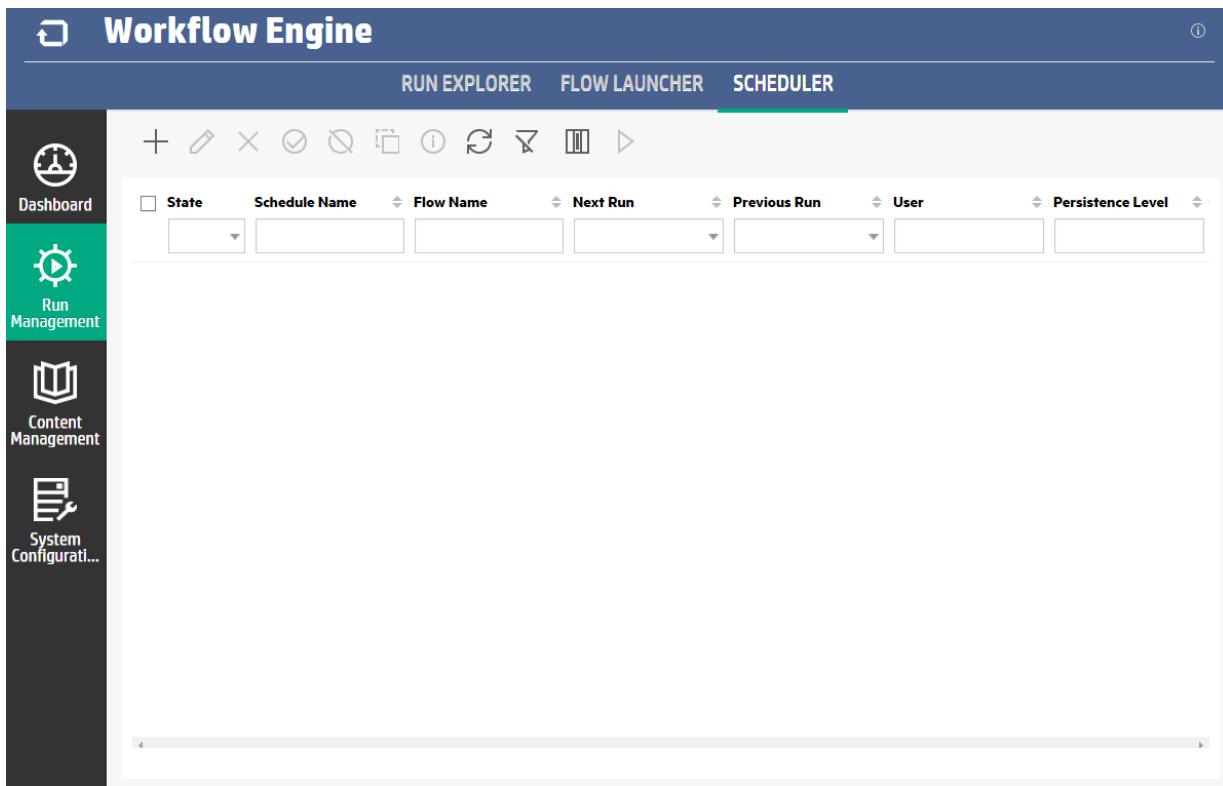
Figure 164: `is_vnf_healthy` Workflow

Setting Up `is_vnf_healthy` to Run Periodically

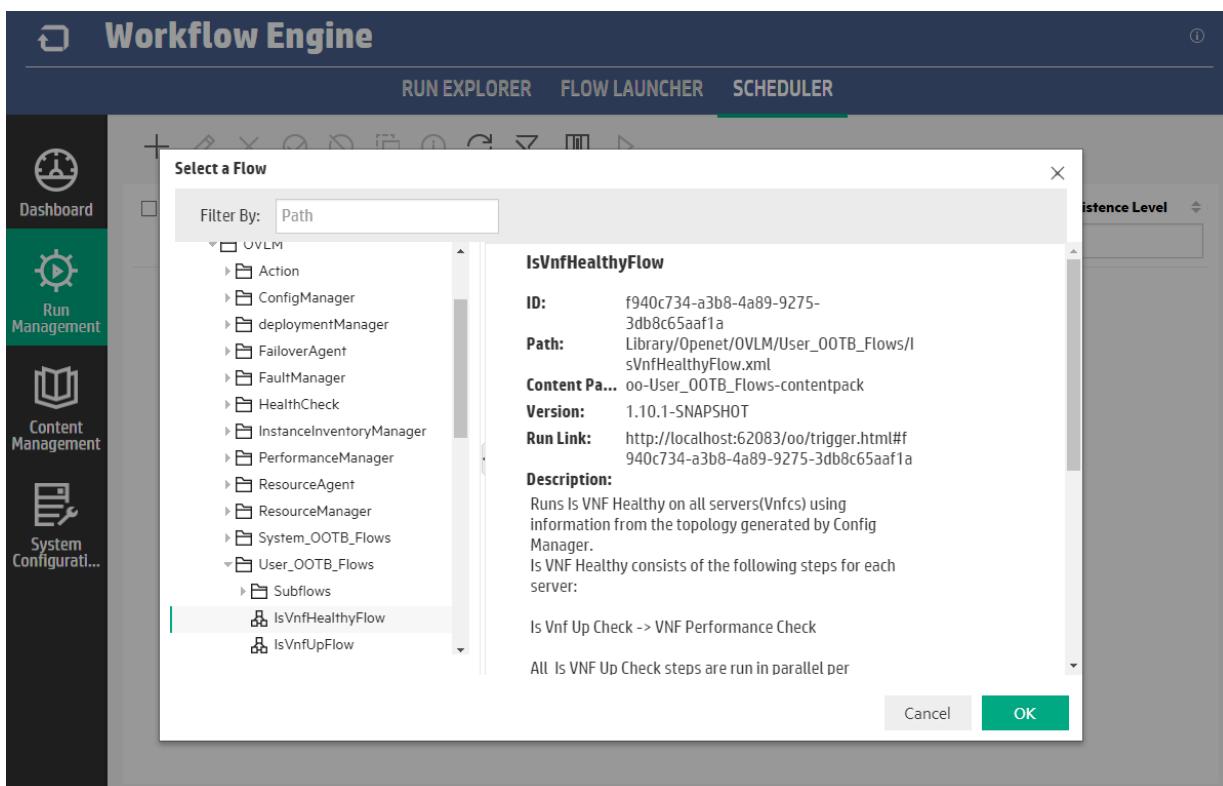
This task describes how to configure the orchestrator to run the `is_vnf_healthy` workflow automatically on a regular basis.

To set up the `is_vnf_healthy` workflow to run periodically follow these steps:

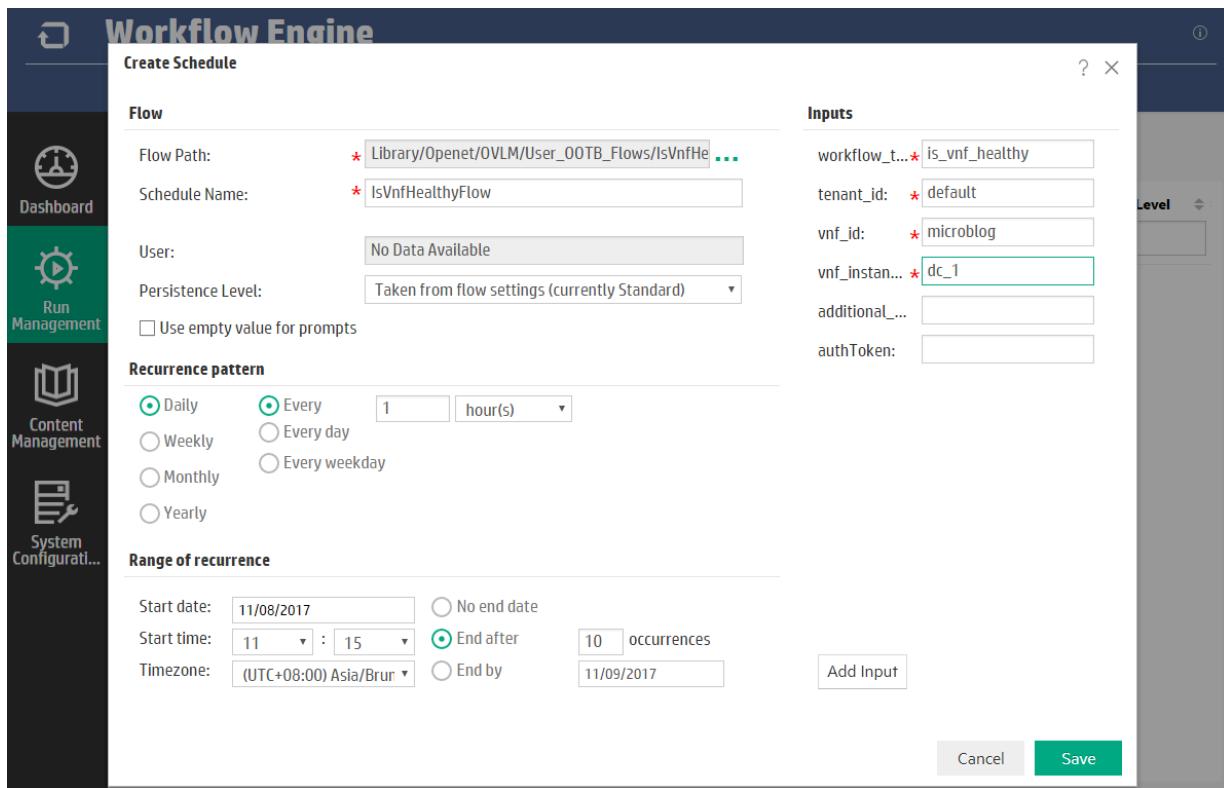
- In the Workflow Engine GUI, click **Run Management** in the navigation pane then click **Scheduler**.



- Click **Create**.
- In the Select a Flow dialog box, browse to the folder Library/Openet/OVLM/User_OOTB_Flows and click **IsVnfHealthyFlow**



4. In the Create Schedule dialog box, populate the Schedule Name field, the Recurrence pattern fields, and the Range of recurrence fields to specify when to run workflow.



5. In the Inputs section of the dialog box, add input parameters and values as described in the following table.

Table 46: Create Schedule Input Parameters

Parameter	Value
tenant_id	The ID of the tenant who owns the VNF on which you are performing the health check
vnf_id	The ID of the VNF on which you are performing the health check.
vnf_instance	The ID of the VNF instance on which you are performing the health check.
vnf_template	The ID of the package on which the VNF is based.

6. Click Save.

IsVNFFlow is now scheduled to be run automatically and periodically for the specified VNF instance.

Managing Metric Logs

Metrics collected by Performance Manager are stored in a rotating series of log files. Each current metric file is named for the metric, for example `cpu_usage`. Older metrics log files have the unixtime of file creation appended to the metric name. For example, a backup (older) `cpu_usage` log file could have the filename `cpu_usage.122132334`.

The metrics logs are stored under /usr/lib64/ovlm/pm/repository-root/metrics. Under that directory are subdirectories for each combination of tenant, VNF, VNFC, and server for which metrics are collected. For example, the metrics collected on server-1 for vnfc-1 of vnfs-1 for the anycom tenant would be stored as follows:

```
/usr/lib64/ovlm/pm/repository-root/
└── metrics
    └── tenants
        └── tenant-1
            └── vnfs
                └── vnfs-1
                    └── vnf_instances
                        └── dublin
                            └── vnfc
                                └── vnfc-1
                                    └── servers
                                        └── server-1.com
                                            └── metric_name
                                                ├── cpu_usage
                                                ├── cpu_usage.1467061299887
                                                ├── cpu_usage.1467061389971
                                                ├── memory_usage
                                                └── memory_usage.1467061299906
                                            └── memory_usage.1467061389973
```

The following is an example of a cpu_usage metrics log file. There is an entry for each time a metric is collected:

```
{"name":"cpu_usage","value":"15","timestamp":135246547,"tags":{}},
{"name":"cpu_usage","value":"15","timestamp":135246547,"tags":{}},
{"name":"cpu_usage","value":"15","timestamp":135246547,"tags":{}},
 {"name":"cpu_usage","value":"15","timestamp":135246547,"tags":{}},
 {"name":"cpu_usage","value":"15","timestamp":135246547,"tags":{}},
 {"name":"cpu_usage","value":"15","timestamp":135246547,"tags":{}},
 {"name":"cpu_usage","value":"15","timestamp":135246547,"tags":{}},
 {"name":"cpu_usage","value":"15","timestamp":135246547,"tags":{}},
 {"name":"cpu_usage","value":"15","timestamp":135246547,"tags":{}},
 {"name":"cpu_usage","value":"15","timestamp":135246547,"tags":{}},
 {"name":"cpu_usage","value":"15","timestamp":135246547,"tags":{}},
 {"name":"cpu_usage","value":"15","timestamp":135246547,"tags":{}},
 {"name":"cpu_usage","value":"15","timestamp":135246547,"tags":{}},
 {"name":"cpu_usage","value":"15","timestamp":135246547,"tags":{}},
 {"name":"cpu_usage","value":"15","timestamp":135246547,"tags":{}},
 {"name":"cpu_usage","value":"15","timestamp":135246547,"tags":{}},
 {"name":"cpu_usage","value":"15","timestamp":135246547,"tags":{}}
```

The metrics log file size and the number of backup logs to be held for each metric are user-configurable. When the file size is reached for one file, the next file is begun. When the maximum number of files is reached, the oldest log is deleted and a new log file begun.

You configure metrics logs collection settings in /etc/ovlm/pm/ovlm-pm.yml. The following is an example of metrics logs configuration within that file:

```
performance-manager:
  repository-root: /usr/lib64/ovlm/pm/repository-root
  metric-files-size: 2MB
  metric-files-history: 7
```

The configurable collection parameters are described in the table below.

Table 47: Configurable Metrics Log Collection Parameters

Parameter	Description
metric-files-size	<p>The size of each log file. The value applies to all metrics log files.</p> <p>The following file size units are available:</p> <ul style="list-style-type: none"> • Bytes • KB • MB • GB <p>Bytes is the default and requires no unit designated. For example metric-files-size: 500 means that the maximum size of each metrics log file is 500 bytes.</p> <p>The example in the code block above sets the file size to be 2MB for each file.</p>
metric-files-history	<p>The number of backup metrics logs collected for each metric. The value applies to each metric equally. For example, if the value is 7 as in the example in the code block above, up to seven backup logs can be collected for each metric in addition to the current one, so the total number of log files would eight multiplied by the number of available metrics.</p>

Managing Faults

The Fault Manager microservice is the Openet Weaver SNMP manager. SNMP enabled VNF components send SNMP alerts (trap messages) to Fault Manager. In some cases this can trigger the Openet Weaver out of the box auto-healing workflow, which attempts to correct the fault.

Openet Weaver microservices, which are managed by systemd, are configured to fire an SNMP trap upon entering a failed unit state. A failed unit state is defined as a service that was started more than twice in any sliding two minute window. Systemd restarts failed services until the threshold is met, after which the service is flagged as failed. An SNMP trap is fired upon a service entering a failed state.

Openet Weaver supports granular traps, each of which have a unique object identifier (OID) that allows the SNMP manager to distinguish the SNMP traps. Human-readable OID information is stored in a Management Information Base (MIB) file. Fault Manager accesses this file when handling traps sent by the SNMP agent.

When a VNF Instance is instantiated, Fault Manager must be updated by synchronizing the VNF instance configuration. When it's called the service requests the following information from Configuration Manager:

- VNF instance topology (mandatory)
- MIB (optional)
- Fault manager policy (optional)

Fault Manager is an optional microservice in an Openet Weaver deployment. When the microservice is not included in the installation file it is not installed and no errors are reported.

Fault Policy Structure

Fault policies allow you to bind alarms received by the Fault Manager microservice with the rules to be compared against them. Fault policies in a fault policy metadata file have the following structure:

```
<PolicyGroupName>
  - policyName : <policy_name_string>
  triggerWorkflowID : <workflow_id_string>
```

```

workflowInputTags :
  <custom_param> : <static_value> OR ${<tag_name>}
healthStatusPriority : "high|medium|low"
gracePeriod : <Number_of_seconds>
notificationType: "<alarm_oid>"
rules :
  - oid: '<oid_id_1>'
    value : "<value_1>"
  - oid: '<oid_id_2>'
    value : "<value_1>"
```

The following table lists the fault policy parameters. All parameters are mandatory.

Table 48: Fault Policy Parameters

Parameter	Type	Mand.	Unit of Measurement	Restrictions	Description
PolicyGroupName	String	Yes			The group to which the policy belongs. Usually multiple groups are specified.
policyName	String	Yes			The name of the policy. The value of this parameter must be unique.
triggerWorkflowID	String	Yes			Name of existing Openet Weaver workflow to be triggered, when the policy is breached.
workflowInputTags	section	No			<p>workflowInputTags contain one or more KEY:VALUE pairs. These are the parameters required by the workflow specified in triggerWorkflowID parameter.</p> <p>The VALUE can be initialized statically or dynamically. For dynamic initialization a substitution approach is used. The values to be replaced with runtime information must be template patterns.</p> <p>The following properties are supported:</p> <ul style="list-style-type: none"> • \${binding['OID']} • \${hostname} <p>\${binding['OID']} can be the the value of any existing OID from an alarm. variable_binding section</p> <p>\${hostname} - is a built-in template value that is substituted with a VNF instance topology "hostname" value taken that os taken from the component to which the received alarm is assigned.</p>
healthStatusPriority	String	Yes		high medium low	The priority of the rule.
gracePeriod	String	Yes	seconds		The time period over which the breach continues before a workflow is triggered.
notificationType	String	Yes			The object ID (OID) that represents the type of alarm, for example FILE_ERROR or script execution failed.

Parameter	Type	Mand.	Unit of Measurement	Restrictions	Description
rules	section	Yes			The objects to be tested against the alarm's variable binding mapping sequence.

The following is an example of a custom fault policy for an nginx upstream issue.

```
NginxFaultPolicy:
  - policyName : NginxUpStreamIssue
    triggerWorkflowID : heal_vnfc
    workflowInputTags :
      hostname: "${binding['1.3.6.1.62.31.18.1.35.0']}"
    healthStatusPriority : low
    gracePeriod : 30
    notificationType: "1.3.6.1.6.3.1.1.5.3"
    rules:
      - oid: '1.3.6.1.2.1.2.2.1.1'
        value : "1974"
```

The following is an example of an auto healing policy that is triggered by a Resource Agent event relating to a VNFC that is down.

```
SimpleFaultPolicies:
  - policyName : VNFCDown
    triggerWorkflowID : heal_vnfc
    workflowInputTags :
      hostname: "${hostname}"
    healthStatusPriority : high
    gracePeriod : 90
    notificationType: "1.3.6.1.4.1.7898.1.8.5"
    rules:
      - oid: '1.3.6.1.4.1.2021.7898.10.2.1'
        value : "is_vnfc_up"
      - oid: '1.3.6.1.4.1.2021.7898.10.4.1'
        value : "1"
```

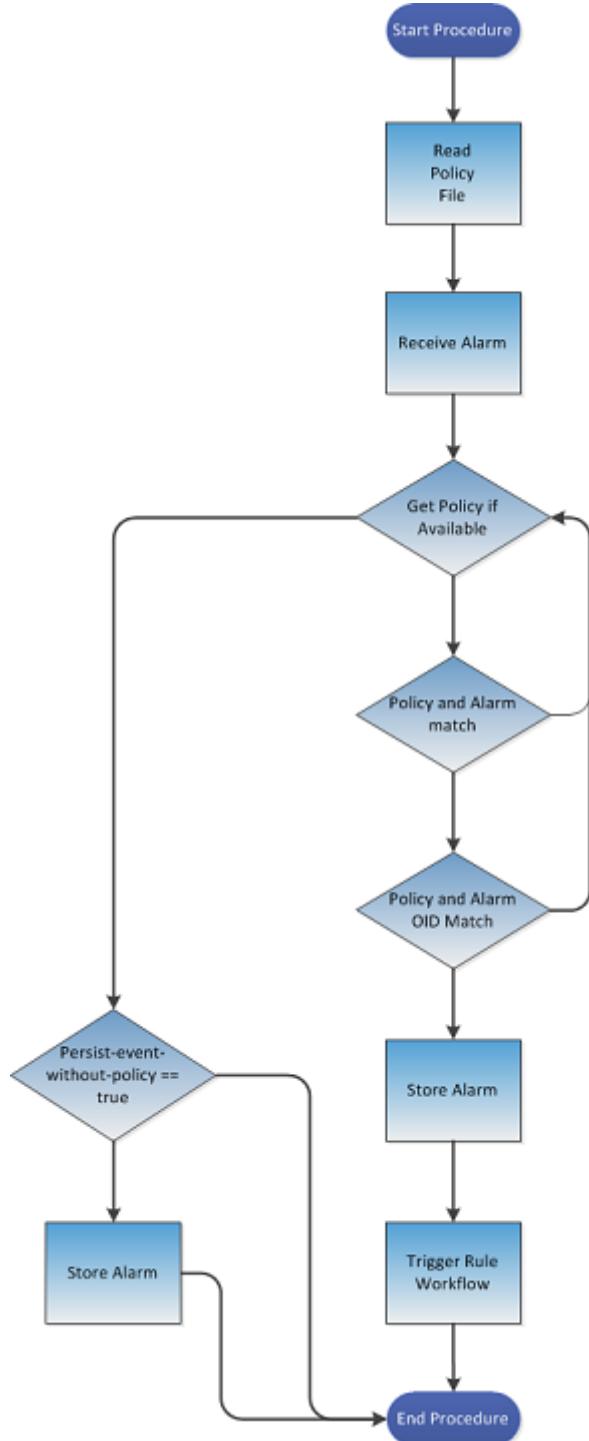
The following is an example of an alarm that could trigger the VNFCDown policy.

```
{
  "snmp_version" : "SNMPv2",
  "message_type" : "TRAP(v2)",
  "community" : "public",
  "notification_type" : "1.3.6.1.4.1.7898.1.8.5",
  "agent_address" : "10.0.3.81",
  "agent_timestamp" : 1472118811220,
  "variable_binding" : {
    "1.3.6.1.4.1.2021.7898.10.4.1" : 1,
    "1.3.6.1.4.1.2021.7898.10.3.1" : "is_vnfc_up_v2.py",
    "1.3.6.1.4.1.2021.7898.10.5.1" : "[Error] The following service(s) are not active: nginx",
    "1.3.6.1.4.1.2021.7898.10.2.1" : "is_vnfc_up",
    "1.3.6.1.4.1.2021.7898.10.1.1" : 1
  }
}
```

Policy Based Alarm Filtering

Fault Manager filters received alarms to determine whether they match a configured policy rule. By default, only alarms that match are stored. However, you can configure fault manager to store all received alarms by setting the property persist-event-without-policy to true in the Fault Manager configuration file.

The alarm filtering flow is shown below.



Managing Fault Policies

Fault policy files are stored and managed through Configuration Manager. Openet Weaver provides API and CLI commands that allow you to perform the following fault policy file management tasks:

Uploading a Fault Policy

You can upload a fault policy file to a VNF in Configuration Manager using the command `ovlm-cm policy upload -y policy_type -t tenant_id -v vnf_id -c vnfc_id -f policy_file` command, for example:

```
ovlm-cm policy upload -y fault -t anycom -v webserver -c httpd -f fault.yml
```

The following is an example of a successful response.

```
data:
  method: POST
  request: http://localhost:9090/api/v2/tenants/anycom/vnfs/webserver/vnfc/h
  ttpd/fault
  status: 200
```

Downloading a Fault Policy

To download a fault policy file use the command `ovlm-cm policy upload -y policy_type -t tenant_id -v vnf_id -c vnfc_id -f fault.yml \| fault.json`

The following is an example of running the `policy download` command.

```
ovlm-cm policy download -y fault -t anycom -v webserver -c httpd -f fault.yml
```

The following is an example of a successful response

```
SimpleFaultPolicies:
  - policyName : DBDown
    triggerWorkflowID : reboot_vnfc_instance
    healthStatusPriority : high
    gracePeriod : 360
    notificationType: "1.3.6.1.4.1.7898.1.8.5" # FW alarmNotification
    rules:
      - oid: '1.3.6.1.4.1.2021.7898.10.2.1'      # action type
        value : "is_vnfc_up"
      - oid: '1.3.6.1.4.1.2021.7898.10.4.1'      # severity
        value : "1"
```

Disabling a Policy on a VNF Instance

By default policies are enabled on microservice startup. You can disable policies for a VNF instance using the command `ovlm-pm policy disable -t tenant_id -v vnf_id -i vnf_instance_id`.

This setting is not persisted. Therefore all instances revert to the default setting of policies enabled if the microservice is restarted.

The following is an example of policies being disabled for the `dc_1` instance of the microblog VNF.

```
ovlm-pm policy disable -t anycom -v microblog -i dc_1
```

The following is an example response from running the command successfully.

```
data:
  method: PUT
  request: http://localhost:8080/api/v1/tenants/anycom/vnfs/microblog/vnf_instances/dc_1/policy/disable
  status: 200
```

Enabling a policy

To enable a disabled policy you use the command `ovlm-pm policy enable -t tenant_id -v vnf_id -i vnf_instance_id`.

The following is an example of policies being enabled for the dc_1 instance of the microblog VNF.

```
ovlm-pm policy enable -t anycom -v microblog -i dc_1
```

The following is an example response from running the command successfully.

```
data:
  method: PUT
  request: http://localhost:8080/api/v1/tenants/anycom/vnfs/microblog/vnf_instances/dc_1/policy/enable
  status: 200
```

Getting Policy Status

When you are not sure whether policies are disabled on an instance you can investigate using the command `ovlm-pm policy status -t tenant_id -v vnf_id -i vnf_instance_id`.

The following queries the policy status of the dc_1 instance of the microblog VNF.

```
ovlm-pm policy status -t anycom -v microblog -i dc_1
```

The following response shows that policies are enabled for the instance.

```
data:
  method: GET
  request: http://localhost:8080/api/v1/tenants/anycom/vnfs/microblog/vnf_instances/dc_1/policy/status
  enabled: true, status: 200
```

Deleting a Fault Policy

To delete the fault policy file for a specific VNFC use the command `ovlm-cm policy delete -y policy_type -t tenant_id -v vnfc_id -c vnfc_id`

The following is an example of running the `policy delete` command.

```
ovlm-cm policy delete -y fault -t anycom -v webserver -c httpd
```

The following is an example of a successful response

```
data:
  method: DELETE
  request: http://localhost:9090/api/v2/tenants/anycom/vnfs/webserver/vnfcs/httpd/policies/fault
  status: 200
```

Managing MIB Files

VNF components (VNFCs) are SNMP enabled when an MIB file is uploaded into the tenants/<tenant_id>/vnfs/<vnfc_id>/workspace/vnfc/<vnfc_id>/mib folder in the Configuration Manager repository.

Note: MIB files are not used in the current version of Openet Weaver.

Uploading an MIB

To upload an MIB file to a VNFC you use the command `ovlm-cm mib upload -t tenant_id -v vnf_id -cvnfc_Id -f file1.mib [file2.mib ...]`.

You can upload multiple MIB files in one command. If a file by the same name already exists it is overwritten. The following example uploads two mib files to the app VNF container of the Anycom webserver VNF.

```
ovlm-cm mib upload -t anycom -v webserver -c app -f traps_1.mib traps_2.mib
...]
```

A successful response for this command is as follows.

```
Uploading Files:
  traps_1.mib
  traps_2.mib
data:
  method: POST
  request: http://localhost:28010/api/v1/tenants/anycom/vnfs/webserver/vnfc/
app/mibs
status: 200
```

An error is generated when the `mib upload` command runs unsuccessfully. The following is an example failure when an attempt is made to upload MIB files when an incorrect `vnfc_id` is specified as a parameter.

```
Uploading Files:
  traps_1.mib
  traps_2.mib
Error Status: 404.

Error Code: RESOURCE_NOT_FOUND.

Detail: The specified vnfc with identifier 'vnf' does not exist. Please check
whether the request is correct.

Uploading Files:
  traps_1.mib
  traps_2.mib
Error Status: 500.
Error Code: CREATE_FILE_FAILED.
Detail: The specified file or directory could not be created at the server.
Please contact your system administrators.
```

You can use the `list resource` command to see what MIBs are listed in the VNF workspace or in the VNF package.

Downloading an MIB

You can use the `mib download` command to download a previously uploaded MIB file to make changes to it or back it up to another location. The parameters you use depend on whether you are downloading the MIB file from a workspace or from a package.

If you are downloading an MIB file from a workspace you use the command `ovlm-cm mib download -t tenant_id -v vnf_id -cvnfc_Id -f file.mib`.

The following is an example of downloading a file from a workspace.

```
ovlm-cm mib download -t anycom -v webserver -c app -f traps_1.mib
```

If you are downloading an MIB file from a package you add the package parameter, so the command is `ovlm-cm mib download -t tenant_id -v vnf_id -cvnfc_Id -p template_id -f file.mib`.

The following is an example of downloading a file from a package.

```
ovlm-cm mib download -t anycom -v webserver -c app -p app_package_1 -f traps_1.mib
```

Regardless of whether you are downloading from a workspace or package, the downloaded file is placed in the current directory. There is no response message.

Deleting an MIB

To remove an MIB file from a workspace you use the command `ovlm-cm mib delete -t tenant_id -v vnf_id -cvnfc_Id -f file.mib`.

The following is an example of downloading a file from a workspace.

```
ovlm-cm mib delete -t anycom -v webserver -c app -f traps_1.mib
```

Related concepts

[Openet Weaver VNF-M API Reference](#) on page 468

Openet Weaver REST API reference guides

[Openet Weaver VNF-M CLI Reference](#) on page 469

Openet Weaver CLI command reference guides

Related reference

[Listing VNF Resources](#) on page 209

[Configuration Manager Directory Structure](#) on page 28

Managing Alarms

Alarm management includes synchronizing VNF instance configurations and retrieving alarms.

VNF Instances are stored in Configuration Manager along with the MIB files for the alarms. Alarms are received, managed and stored on Fault Manager. Therefore, each time a VNF is instantiated, the VNF instance configuration must be synchronized with Fault Manager so that it can receive the alarms for the instance.

You can retrieve an alarm list for any synchronized VNF instance. You can also retrieve specific alarms.

You can also delete VNF instance configurations and alarms, for example when you remove a VNF instance from Configuration Manager you can also delete the associated alarms.

The following is an example of an alarm file.

```
{
  "snmp_version"      : "SNMPv2c",
  "message_type"       : "TRAP(v2)",
  "community"          : "public",
  "notification_type"  : "1.3.6.1.4.1.7898.1.8.5", # Trap OID
  "agent_timestamp"    : 362443,                      # SNMP agent timestamp
  "agent_address"      : "192.168.1.21",            # SNMP client IP
  "variable_binding": {
```

```

    "1.3.6.1.4.1.7898.1.10.1" : "simple_correlator",
    "1.3.6.1.4.1.7898.1.10.2" : "Correlator",
    "1.3.6.1.4.1.7898.1.10.3" : "SCRIPT_EXEC_ERROR",
    "1.3.6.1.4.1.7898.1.10.4" : "PROCESSING_ERROR_ALARM",
    "1.3.6.1.4.1.7898.1.10.5" : "FILE_ERROR",
    "1.3.6.1.4.1.7898.1.10.6" : "MAJOR",
    "1.3.6.1.4.1.7898.1.10.7" : "NON_CLEARABLE",
    "1.3.6.1.4.1.7898.1.10.8" : "0:00:00.00",
    "1.3.6.1.4.1.7898.1.10.9" : "Script execution failed."
    "1.3.6.1.4.1.7898.1.10.10" : "Line: 1 | File: rs1/r1 | Event: SHUTDOWN
| Error: putxs: no such function or object",
    "1.3.6.1.4.1.7898.1.10.11" : "1",
    "1.3.6.1.4.1.7898.1.10.12" : "9f:7a:02:04:d2",
    "1.3.6.1.4.1.7898.1.10.13" : "9f:7a:01:01",
}
}

```

Openet Weaver Resource Agent supports reporting health-check alarms to Fault Manager to trigger. These alarms can be used to trigger the healing flow. The following is an example of such an alarm.

```
{
  "snmp_version" : "SNMPv2",
  "message_type" : "TRAP(v2)",
  "community" : "public",
  "notification_type" : "1.3.6.1.4.1.7898.1.8.5",
  "agent_address" : "10.0.3.81",
  "agent_timestamp" : 1472148781143,
  "variable_binding" : {
    "1.3.6.1.4.1.2021.7898.10.4.1" : 0,
    "1.3.6.1.4.1.2021.7898.10.3.1" : "is_vnfc_up_v2.py",
    "1.3.6.1.4.1.2021.7898.10.2.1" : "is_vnfc_up",
    "1.3.6.1.4.1.2021.7898.10.1.1" : 1
  }
}
```

The following is an example of a Resource Agent coldStart alarm that is sent on service startup. The variable_binding section is unpopulated in such an alarm.

```
{
  "snmp_version" : "SNMPv2",
  "message_type" : "TRAP(v2)",
  "community" : "public",
  "notification_type" : "1.3.6.1.6.3.1.1.5.1",
  "agent_address" : "10.0.3.81",
  "agent_timestamp" : 1472121477024,
  "variable_binding" : { }
```

Synchronizing A VNF Instance Configuration

To synchronize a VNF topology and Fault Manager VNFC configuration for a specific instance us the command
`ovlm-fm configuration sync -t tenant_id -v vnf_id -i instance_id1 [,instance_id2, ...] -p template_id`

Note: There is an equivalent API command available.

You can synchronize multiple VNF instances in the same command. The same package is assigned to all VNF instances.

During synchronization Fault Manager requests the following from Configuration Manager:

- The VNF Instance topology
- The VNF component fault policy, which defines auto-healing behavior

- The MIB files

Note: Only the VNF instance topology is mandatory. If the other files are not found synchronization still succeeds.

In the event that there is an error for a VNF component, that component is excluded from the configuration and the status is reported in the response message. Alarms are received for the component but auto healing are disabled. Factors that can cause such an error during synchronization include the following:

- A VNF instance topology file is not received from Configuration Manager
- The IP address of a host defined in the topology file cannot be resolved

The following is an example of running a sync command.

```
ovlm-fm configuration sync -t anycom -v webapp -i dub_webserver, hk_webapp -p app_package1
```

The following is a successful response from running the command.

```
data:
  method: POST
  request: http://localhost:28100/api/v1/tenants/anycom/vnfs/webserver/sync
status: 200
```

Retrieving Alarms

You can retrieve a complete list of alarms for a VNF instance that are stored in using the command `ovlm-fm alarm list -t tenant_id -v vnf_id -ivnf_instance_id`

The following is an example that lists the all alarms for the `dub_webapp` instance of the Anycom webserver VNF.

```
ovlm-fm alarm list -t anycom -v webserver -i dub_webapp
```

The following is an example of a successful response.

```
data:
  list:
    - vnfcsvnfc/servers/192.168.0.13/alarms/20160726_225058_390_1.3.6.1
      .4.1.7898.1.8.5.alarm
    - vnfcsvnfc/servers/192.168.0.13/alarms/20160726_225449_709_1.3.6.1
      .4.1.7898.1.8.5.alarm
    - vnfcsvnfc/servers/192.168.0.13/alarms/20160726_225711_22_1.3.6.1
      .4.1.7898.1.8.5.alarm
  method: GET
  request: http://localhost:28100/api/v1/tenants/anycom/vnfs/webserver/vnf_instances/dub_webapp/alarms
  status: 200
```

You can retrieve and display the contents of a specific alarm by name, use the command `ovlm-fm alarm get -t tenant_id -v vnf_id -ivnf_instance_id -c vnf_id -sserver_id-a alarm_name`.

The following example displays the contents of one of the alarms listed when the `alarm list` command was run.

```
ovlm-fm alarm get -t anycom -v webserver -i dub_webapp -c httpd -s 192.0.2.13
-a 20160726_225711_22_1.3.6.1.4.1.7898.1.8.5.alarm
```

The following is an example of a successful response.

```
data:
  alarm:
    agent_address: 192.168.0.13
    agent_timestamp: 2715493
```

```

community: public
message_type: TRAP (v2)
notification_type: 1.3.6.1.4.1.7898.1.8.5
snmp_version: SNMPv2c
variable binding:
  1.3.6.1.4.1.7898.1.10.1: PCRF
  1.3.6.1.4.1.7898.1.10.10: 'Line: 1 | File: rs1/r1 | Event: SHUTDOWN | Error: putxs: no such function or object'
  1.3.6.1.4.1.7898.1.10.11: '1'
  1.3.6.1.4.1.7898.1.10.12: 9f:7a:02:04:d2
  1.3.6.1.4.1.7898.1.10.13: 9f:7a:01:01
  1.3.6.1.4.1.7898.1.10.2: Correlator
  1.3.6.1.4.1.7898.1.10.3: SCRIPT_EXEC_ERROR
  1.3.6.1.4.1.7898.1.10.4: PROCESSING_ERROR_ALARM
  1.3.6.1.4.1.7898.1.10.5: FILE_ERROR
  1.3.6.1.4.1.7898.1.10.6: MAJOR
  1.3.6.1.4.1.7898.1.10.7: NON_CLEARABLE
  1.3.6.1.4.1.7898.1.10.8: '0:00:00.00'
  1.3.6.1.4.1.7898.1.10.9: Script execution failed.
method: GET
request: http://localhost:28100/api/v1/anycom/tenant/vnfs/webserver/vnf_instances/dub_webapp/vnfcs/httpd/servers/192.0.2.13/alarms/20160726_225711_22_1.3.6.1.4.1.7898.1.8.5.alarm
status: 200

```

Deleting a VNF Instance Configuration

To delete a VNF instance configuration from Fault Manager use the command `ovlm-fm alarm delete -t tenant_id -v vnf_id -ivnf_instance_id`

Running the command deletes the instance and all associated alarms. The following is an example of running the `alarm delete` command for the `dub_app` instance of the Anycom webserver VNF.

```
ovlm-fm alarm delete -t anycom -v webserver -i dub_webapp
```

The following is a successful response.

```

data:
  method: DELETE
  request: http://localhost:28100/api/v1/tenants/anycom/vnfs/webserver/vnf_instances/dub_webapp
status: 200

```

SNMP Traps

Openet Weaver sends SNMP traps when a microservice enters a failed state.

Openet Weaver microservices are monitored by Systemd. If a microservice process dies, it will be restarted. However, a crash loop may occur if the restarted microservice simply dies again after being restarted.

Systemd is configured to allow two successful microservice startups in any rolling two minute window. That means that a microservice will be marked as failed if it is restarted more than twice in any two minute window.

When a microservice is flagged as being in a 'failed' state, Systemd will fire a componentFailedNotification SNMP Trap.

Notification	OID
componentFailedNotification	.1.3.6.1.4.1.7898.10.1.1

The componentFailedNotification NOTIFICATION contains the following OBJECT fields:

Field	OID	Comment
componentStatusNodeId	.1.3.6.1.4.1.7898.10.2.1	The node ID of the component that failed.
componentStatusComponentName	.1.3.6.1.4.1.7898.10.2.2	The name of the component (microservice).
componentStatusProductName	.1.3.6.1.4.1.7898.10.2.3	Always set to "Weaver".
componentStatusReason	.1.3.6.1.4.1.7898.10.2.4	The reason for this trap to be raised (optional).
componentStatusAdditionalText	.1.3.6.1.4.1.7898.10.2.5	Any additional text relating to the issue (optional).

Configuring Auto Healing Support for a VNF

Openet Weaver provides support for SNMP enabled VNF component to report SNMP alerts (SNMP trap messages) to Fault Manager, which is the Openet Weaver SNMP manager. Certain events trigger the out of the box auto-healing workflow.

Openet Weaver supports granular traps, which have unique object identifier (OID) numbers that allow the Fault Manager to distinguish between them. The human-readable meaning of each OID is stored in a translation file called a Management Information Base (MIB). The MIB is notified by Fault Manager to handle the trap sent by the agent.

Note: Currently only SNMP v3c is supported by Openet Weaver.

Associating Alarms with a VNFC

This topic and its subtopics describe how to use Fault Manager to use a VNFC-specific event to trigger the auto healing flow for a VNF based on the microblog VNF package supplied with Openet Weaver. It also uses the deployment you set up if you followed the Quick Start Guide. Before performing the scenario described in this section, review that deployment, and review the [Creating VNF](#) section of the webhelp.

The service node and resource agents from the Quick Start deployment are illustrated below.

Note: You must install the Quick Start Guide deployment of Openet Weaver if you have not already done so.

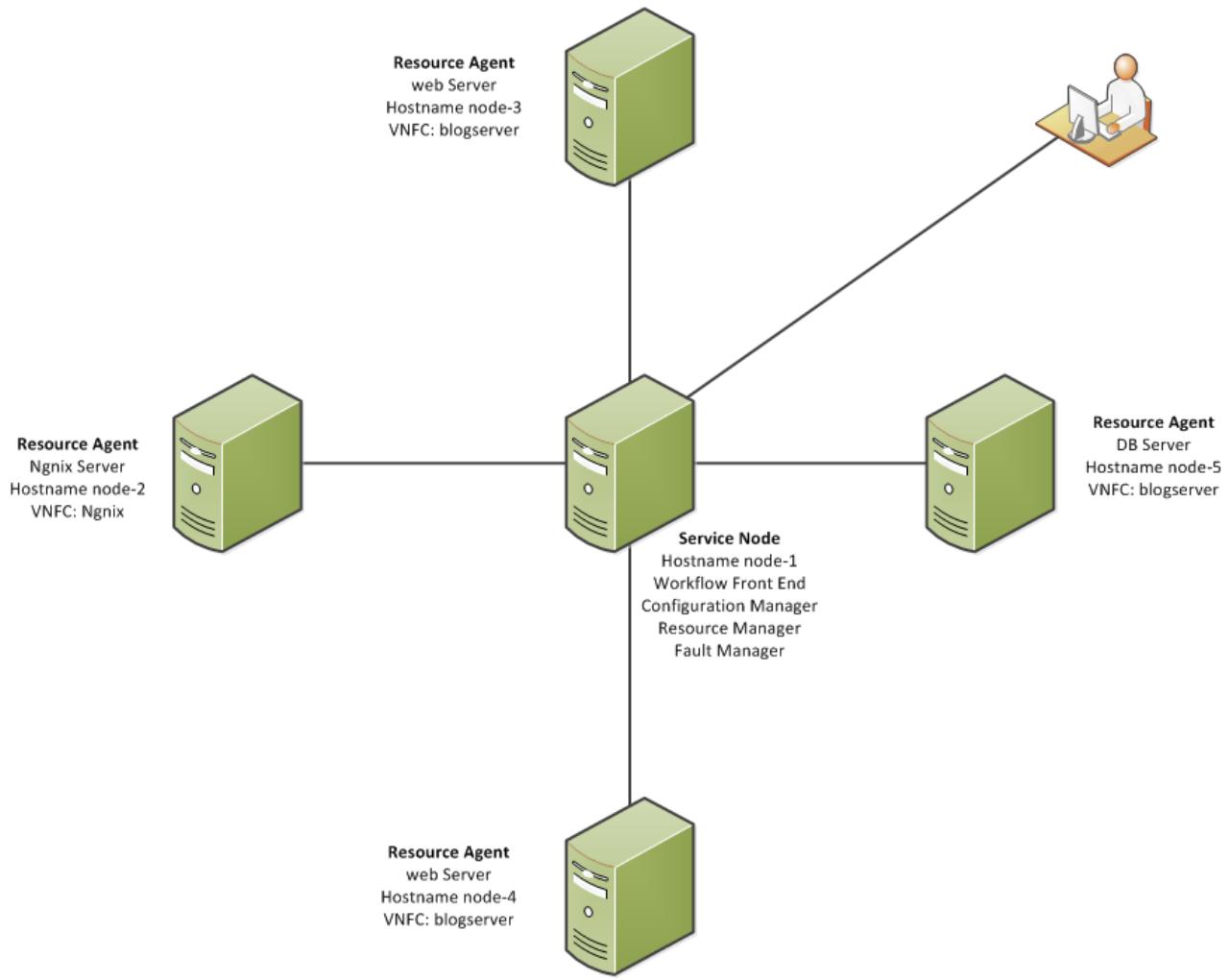


Figure 165: Quickstart Deployment

The out of the box package consists of three VNF components deployed across the resource agents:

- An Nginx server
- Two .js servers, which are the blog servers
- A postgresSQL database server

Note: The Resource Agent to which the Nginx component is going to be deployed must have net-snmp-utils.x86_64 and cronie-1.4.11-14.el7_2.1 (or similar) installed before deployment.

The other Openet Weaver microservices are installed on Node-1. Although there are others, this topic relies on the following microservices only:

- Workflow Front End
- Configuration Manager
- Resource Manager
- Fault Manager

Note: All the above microservices must be installed for the scenario to work.

The VNF package attributes used in this scenario are shown in the following table.

package Attribute	Value
tenant_id	default

package Attribute	Value
vnf_id	microblog
vnfc_id	nginx
vnf_template_id	microblog_v1
vnf_instance_id	dc_1

Associating Alarms with a VNFC

To associate SNMP alarms with a VNFC, configure SNMP enabled VNFCs to send the alarms to the correct Fault Manager host and port. You can find the port number in the snmp.trap_port property in the /etc/ovlm/fm/ovlm-fm.yml file.

To automate the configuration process, the properties can be localized on a runtime node using dynamic configuration information. The properties are in the following section.

```
weaver_services:fault_manager:snmp_host: <host>snmp_trap_port: <port>
```

There are examples in a subsequent topic of this section. Fault Manager uses a sophisticated lookup algorithm to address the VNFC.

It is expected that the VNFC snmp alarm contains a host IP in OID '1.3.6.1.6.3.18.1.3.0' as part of the "variable_binding" section. The host IP explicitly identifies the runtime node running the component.

When Fault Manager receives an alarm and recognizes the associated VNFC, it tries to find an associated healing workflow, trigger the workflow, and store the alarm. By default alarms are not stored when there is no policy associated with it.

Related concepts

[Understanding Dynamic Configuration](#) on page 213

Openet Weaver allows software to be localized on a runtime node using dynamic configuration information.

Updating a VNFC with SNMP Support

As part of this scenario, you monitor Nginx errors caused by Nginx upstreams access issues. In order to do this you must:

- Configure Nginx to log errors into log file
- Create monitoring script to parse log file for errors and send trap when have them found
- Add cron job to be installed on Agent node to run monitoring periodically

Configuring Nginx to Log Errors

- Download the existing nginx.conf.jinja configuration file from the Nginx VNF component in Configuration Manager workflow using the Configuration Manager CLI
- Update the configuration file with the following line.

```
error_log /var/log/nginx/error.log;
```

The resulting configuration file should look like the following.

```
events {
    worker_connections 1024;
}

error_log /var/log/nginx/error.log;

http {
```

```

upstream microblog {
    % for srv in topology.vnfc_ids.blogserver %
        server {{ srv.hostname }} ;
    % endfor %
}

server {
    listen 0.0.0.0:80;
    listen 0.0.0.0:8080;

    location / {
        proxy_pass http://microblog;
    }
}
}

```

Where {{ srv.hostname }} is a jinja template engine definition to be substituted with the blogserver upstream hostnames or IP addresses from the VNF topology file, which is known only at component installation time.

- Upload the configuration file back to the workspace using the Configuration Manager CLI.

Creating a Monitoring Script

The Nginx server community edition does not support monitoring when triggering SNMP traps. Therefore we'll add a simple script, make it configurable, and have it installed by the Openet Weaver instantiation workflow.

When Nginx fails to access the upstream server it reports the following error in log file.

```

2016/09/02 12:38:35 [error] 25513#0: *23 connect() failed (111: Connection refused)
while connecting to upstream, client: 127.0.0.1, server: , request: "GET /public HTTP/1.1",
upstream: "http://10.0.3.82:80/public", host: "node-1"

```

Note: Verify that the upstream IP address is the same as in "hostname" value in VNF instance topology. Otherwise Workflow Engine fails when starting the healing workflow.

The monitoring script checks the log file periodically for the error described above but reports the message only once. It does not report subsequent occurrences. does not report twice on the same message.

When an error condition is detected, the monitoring script triggers the LinkDown snmp trap. The monitoring script is as follows:

```

#!/bin/bash

#Sample monitoring script to check if there are issues with blogserver upstreams

LAST_MONITOR_STATUS_FILE=/etc/nginx/weaver_monitor/last_status.txt
NGINX_ERROR_LOG_FILE=/var/log/nginx/error.log

#get number of errors for blogserver[1,2..]/public upstreams
CUR_ERR_NUM=0
[[ -f $NGINX_ERROR_LOG_FILE ]] && CUR_ERR_NUM=`grep "Connection refused) while connecting to upstream" $NGINX_ERROR_LOG_FILE | grep "/public\\"" | wc -l` 

#load number of errors from previous iteration
OLD_ERR_NUM=0
[[ -f $LAST_MONITOR_STATUS_FILE ]] && OLD_ERR_NUM=`cat $LAST_MONITOR_STATUS_FILE` 

#Check if the number loaded. If not then reset counter to zero

```

```

re='^[0-9]+$'
if ! [[ $OLD_ERR_NUM =~ $re ]]; then
    OLD_ERR_NUM=$CUR_ERR_NUM
fi

#if logrotate occurs
[[ $OLD_ERR_NUM -gt $CUR_ERR_NUM ]] && OLD_ERR_NUM=0

#compare numbers and sent alarm if difference detected
if [[ $OLD_ERR_NUM -lt $CUR_ERR_NUM ]];
then
    #Set active log line to retrieve
    NUM=$((OLD_ERR_NUM+1))
    [[ -f $NGINX_ERROR_LOG_FILE ]] \
        && NEW_ERR_LINE=`grep "Connection refused" while connecting to upstream" \
$NGINX_ERROR_LOG_FILE | grep "/public\\"" | sed "${NUM}q;d"`

    if [[ -z "$NEW_ERR_LINE" ]]; then
        echo "nginx_vnfc_monitoring: unexpected log format: can not find
upstream error"
        exit 2
    fi

    UPSTREAM_IP=`echo $NEW_ERR_LINE | sed -E "s/^.*http[s]?:\/\/([[:digit:].]*:[[:digit:]]*)*/\.\*\$/\1/"` \
    if [[ -z UPSTREAM_IP ]]; then
        echo "nginx_vnfc_monitoring: unexpected log format, upstream ip not
found"
        exit 2
    fi
    snmptrap -m ALL -c public -v 2c {{weaver_services.fault_manager.snmp_host}}: \
{{weaver_services.fault_manager.snmp_trap_port}} "" \
        '1.3.6.1.6.3.1.1.5.3' \
        '1.3.6.1.2.1.2.2.1.1' i 1974 \
        '1.3.6.1.2.1.2.2.1.7' i 2 \
        '1.3.6.1.2.1.2.2.1.8' i 2 \
        '1.3.6.1.6.3.18.1.3.0' a "{{facts.ipaddress}}" \
        '1.3.6.1.2.1.1.5.0' s "$UPSTREAM_IP" || { echo "snmptrap failed
to send trap - code:$?" && exit 1; }
    #If success then store current number of errors
    echo $NUM > $LAST_MONITOR_STATUS_FILE
else
    #If success then store current number of errors
    echo $CUR_ERR_NUM > $LAST_MONITOR_STATUS_FILE
fi
exit 0

```

In the above monitoring script:

- snmptrap is command line tool from net-snmp-utils.x86_64 package to send SNMP traps
- {{weaver_services.fault_manager.snmp_host}}:{{weaver_services.fault_manager.snmp_trap_port}} is a jinja template engine definition to be substituted with the Fault Manager host and port, which is known only at component installation time.
- '1.3.6.1.6.3.1.1.5.3' is the OID of the linkDown trap (RFC 2863, The Interfaces Group MIB)
- '1.3.6.1.2.1.1.5.0' is the OID for SNMPv2-MIB::sysName. It is used to report the upstream failed.

This script sends SNMP trap on any occurrence of upstream error log message.

Configuring Script Installation

Now you add the monitoring script to the nginx component and configure it to be installed as a cron job.

In this example there are no other root cron jobs on the target Resource Agent node. So the job is configured by copying the following file to `/var/spool/cron/root` at update phase.

```
* * * * * /etc/nginx/weaver_healthcheck/nginx_vnfc_monitoring.sh
```

At this point, with all files in this section uploaded to the Nginx component, running the Configuratio Manager `resource list` command should return a response similar to the following.

```
$ovlm-cm resource list -y vnfc_configuration -p /tenants/default/vnfs/microblog/vnfcs/nginx
data:
method: GET
request: http://node-1:28010/api/v2/list/vnfc_configuration
resource_list:
- creation_time: '2016-09-01 16:32:03'
  resource_id: nginx_vnfc_monitoring.sh.jinja
- creation_time: '2016-09-01 16:32:03'
  resource_id: nginx.conf.jinja
- creation_time: '2016-09-01 16:32:03'
  resource_id: cron_job
resource_path: /tenants/default/vnfs/microblog/vnfcs/nginx
resource_type: vnfc_configuration
status: 200
```

Configuring Fault Manager with a Health Check Policy

To ensure that arbitrary workflows are not triggered, you configure Fault Manager so that specific predefined responses are associated with specific alarms received from a VNF component or an Openet Weaver Resource Agent. You make this association in the Fault Manager policy file.

After you create the policy file, you upload it and then use it to create a package with which to instantiate the VNF instance.

For the purpose of this example scenario the Fault Manager policy is configured to call `heal_vnfc`, passing the hostname parameter to the command to run the workflow. The `heal_vnfc` workflow is triggered for the following alarms when received from the node-hosted Nginx server:

- "nginx service is down"—triggered by the Resource Agent
- "nginx upstream is inaccessible"—triggered by the Nginx monitoring script.

The "nginx service is down" alarm is a built-in Resource Agent alarm with the following structure.

```
Alarm
{
  "snmp_version"      : "SNMPv2c",
  "message_type"       : "TRAP(v2)",
  "community"          : "public",                                # ovlm-ra.yml
'snmp.community' property
  "agent_timestamp"   : <timestamp>,                            # local timestamp
  "notification_type" : "1.3.6.1.4.1.7898.1.8.5",               # FW alarmNotification

  "variable_binding": {
    "of UCDEVIS (.1.3.6.1.4.1.2021)                                # Openet subtree (7898)
      "1.3.6.1.4.1.2021.7898.10.1.1" : 1                      # entry index
      "1.3.6.1.4.1.2021.7898.10.2.1" : "is_vnfc_up"           # action type
      "1.3.6.1.4.1.2021.7898.10.3.1" : "is_vnfc_up.py"        # script name
      "1.3.6.1.4.1.2021.7898.10.4.1" : 1                      # severity (exit
code): 1 (CRITICAL), 0 (CLEAR)
      "1.3.6.1.4.1.2021.7898.10.5.1" : "Error message"        # error output
joined in one line
```

```

    }
}
```

The following is the `fault_manager_policy.yml` policy file that you must create for this scenario.

```

SimpleFaultPolicies:
  - policyName : NginxDown
    triggerWorkflowID : heal_vnfc
    workflowInputTags :
      hostname: "${hostname}"
    healthStatusPriority : high
    gracePeriod : 60
    notificationType: "1.3.6.1.4.1.7898.1.8.5"
    rules:
      - oid: '1.3.6.1.4.1.2021.7898.10.2.1'
        value : "is_vnfc_up"
      - oid: '1.3.6.1.4.1.2021.7898.10.4.1'
        value : "1"

  - policyName : NginxUpStreamIssue
    triggerWorkflowID : heal_vnfc
    workflowInputTags :
      hostname: "${binding['1.3.6.1.6.3.18.1.3.0']}"
    healthStatusPriority : low
    gracePeriod : 30
    notificationType: "1.3.6.1.6.3.1.1.5.3"
    rules:
      - oid: '1.3.6.1.2.1.2.2.1.1'
        value : "1974"
```

Note: It's important to adjust the policy gracePeriod for the monitoring script polling interval. When an alarm for the second down upstream is sent from the same runtime node within the policy grace period it is discarded. You can test this by sending a second alarm before the grace period expires, after the first alarm completes.

30 seconds for the policy and 60 seconds for the cron job polling interval is sufficient.

After you create the policy file, upload it to the Configuration Manager workspace using the configuration Manager `policy upload` command. The following uploads the policy created for the scenario in the previous topic.

```
ovlm-cm policy upload -tdefault-v microblog -c nginx -y fault -f
fault_manager_policy.yml
```

When uploaded, the file is renamed `fault.yml`. Verify that the file uploaded successfully and was renamed by using the `resource list` command.

```
$ ovlm-cm --baseurl node-1:28010 resource list -y vnfc_policy \
-p /tenants/default/vnfs/microblog/vnfcs/nginx
```

The response should look similar to the following.

```

data:
  method: GET
  request: http://node-1:28010/api/v2/list/vnfc_policy
  resource_list:
    - creation_time: '2016-09-01 16:32:03'
      resource_id: fault.yml
      resource_path: /tenants/default/vnfs/microblog/vnfcs/nginx
      resource_type: vnfc_policy
  status: 200
```

Creating the package and Instantiating

At this point you must create a package and use it to instantiate a VNF. In this scenario you create a package called `microblog_v1` for the microblog VNF then associate it with a VNF instance that has a deployment-specific VNF topology.

To create the package and instantiate the microblog VNF from it, follow these steps:

Note: Steps 2 and 3 are not required when creating and instantiating a package configured for a Virtualized Infrastructure Manager (VIM). In such a case a VNFD file is used to automatically create a VNF instance and generate a topology file. Creating and populating a VNFD file is beyond the scope of this example scenario.

1. Create a package using the `template create` command as follows.

```
ovlm-cm template create -t default -v microblog -p microblog_v1
```

Running the command results in a response similar to the following.

```
data:
  method: POST
  request: http://node-1:28010/api/v2/tenants/default/vnfs/microblog/template
  status: 200
```

2. Create an instance of the microblog VNF using the `vnf_instance create` command as follows.

```
ovlm-cm vnf_instance create -t default -v microblog -i dc_1
```

Running the command results in a response similar to the following.

```
data:
  method: POST
  request: http://node-1:28010/api/v2/tenants/default/vnfs/microblog/vnf_instances
  status: 200
```

3. Upload the topology for the VNF instance using the `topology upload` command as follows.

```
ovlm-cm topology upload -t default -v microblog -i dc_1 -f
cluster_vnf_topology.yml -p microblog_v1 -d "demo"
```

Running the command results in a response similar to the following.

```
Uploading Files:
  cluster_vnf_topology.yml
data:
  method: POST
  request: http://node-1:28010/api/v2/tenants/default/vnfs/microblog/vnf_instances/dc_1/vnf_topology
  status: 200
```

4. Instantiate the VNF using the `workflow submit` command as follows.

```
ovlm-fe workflow submit -t default -v microblog -i dc_1 -y
instantiate_vnf_without_vim -p '{"vnf_template_id": "microblog_v1"}'
```

Running the command results in a response similar to the following.

```
data:
  method: POST
  request: http://ovlm-vm:8080/api/v1/tenants/default/vnfs/microblog/vnf_inst
```

```
ances/dc_1/workflow_instances
  workflow_instance_id: '112604592'
  status: 200
```

Synchronize the VNF Instance Policy with Fault Manager

So far you've created a fault policy, used it to create a package, and instantiated a VNF from that package. Now you must notify fault manager that there is a new policy by synchronizing the policy related to the VNF instance on Configuration Manager with Fault Manager, where the alarm are parsed and stored.

Note: By default, Fault Manager stores only those alarms that match policy rules.

You do this using the `configuration sync` command as follows.

```
ovlm-fm configuration sync -t default -v microblog -i dc_1 -p microblog_v1
```

When you run the command, Fault Manager requests the following VNF instance-related files from Configuration Manager:

- The VNF Instance topology
- The VNF component fault policy, which defines auto-healing behavior
- The MIB files

The VNF instance topology must be supplied for the command to be successful. However the other files are optional.

Assuming that the command runs successfully, you should see a response similar to the following:

```
data:
  method: POST
  request: http://node-1:28100/api/v1/tenants/default/vnfs/microblog-sync
  warning:
    blogserver:
      - error: There is no policy available for the vnfc
        error_code: NO_CHANGE_APPLIED
    postgres:
      - error: There is no policy available for the vnfc
        error_code: NO_CHANGE_APPLIED
status: 200
```

Note: The `configuration sync` command is triggered automatically as part of the instantiation workflow and the upgrade workflow.

Generate and Examine an SNMP Trap

In this topic you generate an SNMP trap against the example scenario fault policy configuration to verify that it triggers the `heal_vnfc` workflow as expected.

Before generating the SNMP trap, ensure that the service healthcheck is disabled on the Resource Agents that manage the blogserver VNFC. You must do this to avoid triggering `heal_vnfc` workflows with any other alarms that could be sent by the Nginx monitoring script or the blogserver healthcheck monitor in the Resource Agent.

Configure the Resource Agent to report alarms to Fault Manager. You can verify that this is the case by opening the `/etc/ovlm/ra/ovlm-ra.yml` on the Resource agent and finding the following lines.

```
ovlm:
  resource-agent:
    fm-enable: true
```

If `fm-enable` is set to false, change the setting to true.

Note: The preferred method of configuring microservice YML files is through the installation file.

If you change the configuration you must restart the Resource Agent using the restart command as follows.

```
systemctl restart ovlm-ra
```

As a final prerequisite to generating the SNMP trap, verify that the VNF works as expected by accessing Nginx at <http://localhost:8080/public>. on the host where the demonstration VM is running. You should see the microblog as shown below.



Figure 166: Microblog VNF

Now generate an alarm that is sent by the Nginx monitoring script by stopping both blogservers. The result should be that the monitoring script sends a notification that there is an Nginx upstream issue. You can stop the blog servers using the following commands.

```
$ curl -X PUT http://node-1:28020/api/v1/tenants/default/vnfs//microblog/vnf_instances/dc_1/vnfcs/blogserver/vnfc_instances/1/action -H "Content-Type: application/json" -d '{ "action_type" : "stop_vnfc", "server_id" : "node-3" }'

$ curl -X PUT http://node-1:28020/api/v1/tenants/default/vnfs//microblog/vnf_instances/dc_1/vnfcs/blogserver/vnfc_instances/1/action -H "Content-Type: application/json" -d '{ "action_type" : "stop_vnfc", "server_id" : "node-4" }'
```

If the commands worked as expected you should receive a 502 Bad Gateway response when trying to access Nginx. The issue creates an error log message Nginx log and is subsequently reported to Fault Manager by the monitoring script. Based on the configuration in this example scenario, the following occur:

- Alarms are triggered for both blog servers sequentially
- The received alarm triggers the heal_vnfc workflow against the failed upstream. The alarm can be seen in the Fault manager log and in the Workflow Engine Dashboard
- When the workflow finishes running, restart the upstream

The Fault Manager log is located in `/var/log/ovlm/fm/ovlm-fm.log`. When you open it you should see entries similar to those shown below.

```
14:30:01.713 [DispatcherPool.0] INFO c.o.m.o.f.service.TrapService - Received PDU: CommandResponderEvent[securityModel=2, securityLevel=1, maxSizeResponsePDU=65535, pduHandle=PduHandle[465651637], stateReference=StateReference[msgID=0, pduHandle=PduHandle[465651637], securityEngineID=null, securityModel=null, securityName=public, securityLevel=1, contextEngineID=null, contextName=null, retryMsgIDs=null], pdu=TRAP[requestID=465651637, errorStatus=Success(0), errorIndex=0, VBS[1.3.6.1.2.1.1.3.0 = 1 day, 21:08:08.31; 1.3.6.1.6.3.1.1.4.1.0 = 1.3.6.1.6.3.1.1.5.3; 1.3.6.1.2.1.2.2.1.1 = 1974; 1.3.6.1.2.1.2.2.1.7 = 2; 1.3.6.1.2.1.2.2.1.8 = 2; 1.3.6.1.6.3.18.1.3.0]
```

```

= 10.0.3.81; 1.3.6.1.2.1.1.5.0 = 10.0.3.82]], messageProcessingModel=1,
securityName=public, processed=false, peerAddress=10.0.3.81/43669,
transportMapping=org.snmp4j.transport.DefaultUdpTransportMapping@4edc02e,
tmStateReference=null]
14:30:01.713 [DispatcherPool.0] INFO c.o.m.o.f.service.TrapService - Peer
address: 10.0.3.81/43669
14:30:01.713 [DispatcherPool.0] INFO c.o.m.o.f.service.TrapService - Trap
OID: 1.3.6.1.6.3.1.1.5.3
14:30:01.713 [DispatcherPool.0] INFO c.o.m.o.f.service.TrapService -
VariableBinding = 1.3.6.1.2.1.2.2.1.1 = 1974 (ASN.1 syntax identifier = 2)
14:30:01.713 [DispatcherPool.0] INFO c.o.m.o.f.service.TrapService -
VariableBinding = 1.3.6.1.2.1.2.2.1.7 = 2 (ASN.1 syntax identifier = 2)
14:30:01.713 [DispatcherPool.0] INFO c.o.m.o.f.service.TrapService -
VariableBinding = 1.3.6.1.2.1.2.2.1.8 = 2 (ASN.1 syntax identifier = 2)
14:30:01.713 [DispatcherPool.0] INFO c.o.m.o.f.service.TrapService -
VariableBinding = 1.3.6.1.6.3.18.1.3.0 = 10.0.3.81 (ASN.1 syntax identifier =
64)
14:30:01.713 [DispatcherPool.0] INFO c.o.m.o.f.service.TrapService -
VariableBinding = 1.3.6.1.2.1.1.5.0 = 10.0.3.82 (ASN.1 syntax identifier = 4)
14:30:01.713 [DispatcherPool.0] INFO c.o.m.o.f.service.TrapService - Lookup
by 10.0.3.81
14:30:01.718 [DispatcherPool.0] INFO c.o.m.o.f.r.AlarmFileRepository - Alarm
file: /usr/lib64/ovlm/fm/repository-root/storage/tenants/default/vnfs/mi
croblog/vnf_instances/dc_1/vnfcs/nginx/vnfc_instances/10.0.3.81/al
arms/20160922_143001_718_1.3.6.1.6.3.1.1.5.3.alarm
14:30:01.718 [DispatcherPool.0] INFO c.o.m.o.f.r.AlarmFileRepository - Save
alarm file:
{
  "snmp_version" : "SNMPv2c",
  "message_type" : "TRAP(v2)",
  "community" : "public",
  "notification_type" : "1.3.6.1.6.3.1.1.5.3",
  "agent_address" : "10.0.3.81",
  "agent_timestamp" : 16248831,
  "variable_binding" : {
    "1.3.6.1.2.1.2.2.1.7" : "2",
    "1.3.6.1.2.1.2.2.1.8" : "2",
    "1.3.6.1.2.1.1.5.0" : "10.0.3.82",
    "1.3.6.1.2.1.2.2.1.1" : "1974",
    "1.3.6.1.6.3.18.1.3.0" : "10.0.3.81"
  }
}
14:30:01.719 [DispatcherPool.0] INFO c.o.m.o.f.service.PolicyService - Matched
policy: NginxUpStreamIssue
14:30:01.719 [DispatcherPool.0] INFO c.o.m.o.f.service.PolicyService - Set
Grace Period to 15s
14:30:01.719 [DispatcherPool.0] INFO c.o.m.o.f.service.PolicyService -
Triggering workflow: heal_vnfc
14:30:01.720 [DispatcherPool.0] INFO c.o.m.o.f.service.PolicyService -
WeRequestData: {workflow_type: heal_vnfc, vnfd:{"hostname":"10.0.3.82"}, hostname:null}
14:30:02.990 [DispatcherPool.0] INFO c.o.m.o.f.service.PolicyService -
Workflow: heal_vnfc triggered successfully

```

The called `heal_vnfc` workflow can be seen in the Workflow Engine GUI under **Run Management > Run Explorer** as shown.

Run Name	Run ID	Status	Start Time	User	Duration
VnfcHealingFlow	101000229	Completed - Resolved	1:43 PM	anonymousUser	44.530 seconds
VnfcHealingFlow	101000212	Completed - Resolved	1:42 PM	anonymousUser	29.647 seconds
VnfcHealingFlow	101000195	Completed - Resolved	1:42 PM	anonymousUser	27.166 seconds
VnfcHealingFlow	101000178	Completed - Resolved	1:41 PM	anonymousUser	37.035 seconds
VnfcHealingFlow	101000161	Completed - Resolved	1:38 PM	anonymousUser	33.245 seconds

Figure 167: VnfcHealingFlow in the Workflow Engine GUI

With the system restored, generate a health-check alarm for the Nginx VNFC. To generate this alarm stop the Nginx server on node-2 using the command shown below.

```
$ curl -X PUT http://node-1:28020/api/v1/tenants/default/vnfs//microblog/vnf_instances/dc_1/vnfc_instances/1/action -H "Content-Type: application/json" -d '{ "action_type" : "stop_vnfc", "server_id" : "node-2" }'
```

Running the command should result in a response similar to the following.

```
{
  "data" : {
    "status" : "COMPLETED",
    "result_code" : "0",
    "output_msg" : [ "The following service(s) were stopped successfully: nginx" ],
    "error_msg" : [ ],
    "request" : "http://node-1:28020/api/v1/tenants/default/vnfs//microblog/vnf_instances/dc_1/vnfc_instances/1/action",
    "method" : "PUT"
  },
  "status" : 200
}
```

Some time after the node stops, a health-check event is sent from the Resource Agent. The Fault Manager log file contains lines similar to the ones shown below, which indicates that Nginx is down.

```
14:48:28.789 [qtp1716511704-15] INFO c.o.m.o.f.aop.LoggingAdvice - ENTER com.openet.modules.ovlm.faultmanager.rest.controller.impl.FaultManagerControllerImpl#storeAlarm
Method parameters:
[request = Request(POST /api/v1/tenants/default/vnfs/microblog/vnf_instances/dc_1/vnfc_instances/1/action@719a1809]
[tenantId = default]
[vnfcId = microblog]
[vnfcInstanceId = dc_1]
[vnfcId = nginx]
[serverId = node-2]
[alarm = Alarm{snmpVersion='SNMPv2', messageType='TRAP(v2)', community='public', notificationType='1.3.6.1.4.1.7898.1.8.5', ipAddress='null', timeStamp=1474555708764, variableBinding={1.3.6.1.4.1.2021.7898.10.4.1=1, 1.3.6.1.4.1.2021.7898.10.3.1=is_vnfc_up_v2.py, 1.3.6.1.4.1.2021.7898.10.5.1=ERROR The following service(s) are inactive: nginx, 1.3.6.1.4.1.2021.7898.10.2.1=is_vnfc_up, 1.3.6.1.4.1.2021.7898.10.1.1=1}}]
14:48:28.794 [qtp1716511704-15] INFO c.o.m.o.f.r.AlarmFileRepository - Alarm
```

```

file: /usr/lib64/ovlm/fm/repository-root/storage/tenants/default/vnfs/microblog/vnf_instances/dc_1/vnfcs/nginx/vnfc_instances/10.0.3.81/alarms/20160922_144828_794_1.3.6.1.4.1.7898.1.8.5.alarm
14:48:28.794 [qtp1716511704-15] INFO c.o.m.o.f.r.AlarmFileRepository - Save alarm file:
{
  "snmp_version" : "SNMPv2",
  "message_type" : "TRAP(v2)",
  "community" : "public",
  "notification_type" : "1.3.6.1.4.1.7898.1.8.5",
  "agent_address" : "10.0.3.81",
  "agent_timestamp" : 1474555708764,
  "variable_binding" : {
    "1.3.6.1.4.1.2021.7898.10.4.1" : "1",
    "1.3.6.1.4.1.2021.7898.10.3.1" : "is_vnfc_up_v2.py",
    "1.3.6.1.4.1.2021.7898.10.5.1" : "ERROR The following service(s) are inactive: nginx",
    "1.3.6.1.4.1.2021.7898.10.2.1" : "is_vnfc_up",
    "1.3.6.1.4.1.2021.7898.10.1.1" : "1"
  }
}
14:48:28.799 [qtp1716511704-15] INFO c.o.m.o.f.service.PolicyService - Matched policy: NginxDown
14:48:28.799 [qtp1716511704-15] INFO c.o.m.o.f.service.PolicyService - Set Grace Period to 15s
14:48:28.799 [qtp1716511704-15] INFO c.o.m.o.f.service.PolicyService - Triggering workflow: heal_vnfc
14:48:28.799 [qtp1716511704-15] INFO c.o.m.o.f.service.PolicyService - WeRequestData: {workflow_type: heal_vnfc, vnfd:{"hostname":"10.0.3.81"}, hostname:null}
14:48:30.657 [qtp1716511704-15] INFO c.o.m.o.f.service.PolicyService - Workflow: heal_vnfc triggered successfully

```

The `heal_vnfc` workflow is triggered as indicated in the last line of the log file. When the workflow completes successfully the Resource Agent detects that the service restarted and the following event is reported to Fault Manager.

```

14:48:35.779 [qtp1716511704-19] INFO c.o.m.o.f.aop.LoggingAdvice - ENTER com.openet.modules.ovlm.faultmanager.rest.controller.impl.FaultManagerControllerImpl#storeAlarm
Method parameters:
[request = Request(POST /api/v1/tenants/default/vnfs/microblog/vnf_instances/dc_1/vnfcs/nginx/servers/node-2)@7828a5a0]
[tenantId = default]
[vnfId = microblog]
[vnfInstanceId = dc_1]
[vnfcId = nginx]
[serverId = node-2]
[alarm = Alarm{snmpVersion='SNMPv2', messageType='TRAP(v2)', community='public', notificationType='1.3.6.1.4.1.7898.1.8.5', ipAddress='null', timeStamp=1474555715757, variableBinding={1.3.6.1.4.1.2021.7898.10.4.1=0, 1.3.6.1.4.1.2021.7898.10.3.1=is_vnfc_up_v2.py, 1.3.6.1.4.1.2021.7898.10.2.1=is_vnfc_up, 1.3.6.1.4.1.2021.7898.10.1.1=1}}]
14:48:35.786 [qtp1716511704-19] INFO c.o.m.o.f.r.AlarmFileRepository - Alarm file: /usr/lib64/ovlm/fm/repository-root/storage/tenants/default/vnfs/microblog/vnf_instances/dc_1/vnfcs/nginx/vnfc_instances/10.0.3.81/alarms/20160922_144835_786_1.3.6.1.4.1.7898.1.8.5.alarm
14:48:35.786 [qtp1716511704-19] INFO c.o.m.o.f.r.AlarmFileRepository - Save alarm file:
{
  "snmp_version" : "SNMPv2",
  "message_type" : "TRAP(v2)",

```

```

"community" : "public",
"notification_type" : "1.3.6.1.4.1.7898.1.8.5",
"agent_address" : "10.0.3.81",
"agent_timestamp" : 1474555715757,
"variable_binding" : {
  "1.3.6.1.4.1.2021.7898.10.4.1" : "0",
  "1.3.6.1.4.1.2021.7898.10.3.1" : "is_vnfc_up_v2.py",
  "1.3.6.1.4.1.2021.7898.10.2.1" : "is_vnfc_up",
  "1.3.6.1.4.1.2021.7898.10.1.1" : "1"
}
}
14:48:35.792 [qtp1716511704-19] INFO c.o.m.o.f.aop.LoggingAdvice - EXIT
com.openet.modules.ovlm.faultmanager.rest.controller.impl.FaultMan
agerControllerImpl#storeAlarm with RESPONSE:
  Status: 200
body: {
  "status" : 200,
  "data" : {
    "request" : "http://node-1:28100/api/v1/tenants/default/vnfs/microblog/vnf_ins
tances/dc_1/vnfc/instances/node-2",
    "method" : "POST"
  }
}

```

Openet Weaver Operational Commands

Openet Weaver uses systemctl to manage Openet Weaver microservices operations. Openet Weaver has operational commands for:

- Starting microservices
- Stopping microservices
- Getting the status of microservices

Note: Starting and stopping the services can be done only by a root user or a user with sudo access.

The commands for these are described below.

The install process installs and starts the microservices by default. Operators can also start, stop, and get the status of microservices manually using systemctl commands.

The following table describes the Openet Weaver Microservices. The service names are passed as arguments to the systemctl command to start and stop the microservices.

Table 49: Openet Weaver Services

Service Name	Description
ovlm-ra	The Openet Weaver resource agent microservice runs on each resource_agent node in the installation file.
ovlm-fe	The Openet Weaver Front End Rest microservice that runs on the front end node defined in the installation file.
ovlm-cm	The Openet Weaver Configuration Manager microservice runs on the configuration_manager node defined in the installation file.
ovlm-rm	The Openet Weaver Resource Manager service runs on the resource_manager node defined in the installation file.

Service Name	Description
ovlm-wf	The Openet Weaver Work flow engine service runs on the workflow_engine node defined in the installation file.
ovlm-val	The Openet Weaver VIM abstraction layer runs on the vim_abstraction_layer node defined in the installation file.
ovlm-pm	The Openet Weaver Performance Manager microservice runs on the performance_manager node defined in the installation file.
ovlm-fm	The Openet Weaver Fault Manager microservice runs on the fault_manager node defined in the installation file.
ovlm-iim	The Openet Weaver VNF Instance Inventory Manager microservice runs on the instance_inventory_manager node defined in the installation file.
ovlm-vnfm-gui	The Openet Weaver VNFM GUI microservice runs on the vnfm_gui node defined in the installation file.
ovlm-dm	The Openet Weaver Deployment Manager microservice runs on the deployment_manager node defined in the installation file.

Starting Microservices

Use `systemctl start ServiceName` to start any of the above services where ServiceName is one of the Openet Weaver microservices defined in the [Table 1](#).

The following is an example of starting the Resource Agent:

```
systemctl start ovlm-ra
```

SystemD monitors all microservices except for the Workflow Engine, VIM Abstraction Layer, and the VNF Instance Inventory Manager. In the event that a monitored microservice goes down SystemD attempts to restart it.

Stopping Microservices

Use `systemctl stop serviceName` to stop any of the microservices listed in [Table 1](#).

The following is an example of stopping the Resource Agent:

```
systemctl stop ovlm-ra
```

Restarting Microservices

Use `systemctl restart serviceName` to restart any running microservice listed in [Table 1](#).

Note: You must restart when you make configuration changes to a microservice so that the changes are picked up by the system.

The following is an example of stopping the Resource Agent:

```
systemctl restart ovlm-ra
```

Getting the Status of a Microservice

Use `systemctl status serviceName` to get the status of microservices listed in [Table 1](#). The status is active (running) when the service is up or inactive (dead) when the service is stopped.

The following is an example of getting the status of a Resource Agent.

```
sudo systemctl status ovlm-ra
```

The resulting feedback shows the microservice is running:

```
● ovlm-ra.service - OVLM Resource Manager Agent
   Loaded: loaded (/usr/lib/systemd/system/ovlm-ra.service; enabled; vendor
   preset: disabled)
     Active: active (running) since Thu 2016-03-24 13:51:07 GMT; 52s ago
       Process: 15033 ExecStop=/etc/init.d/ovlm-ra stop (code=exited,
   status=0/SUCCESS)
       Process: 15050 ExecStart=/etc/init.d/ovlm-ra start (code=exited,
   status=0/SUCCESS)
     Main PID: 15053 (java)
        CGroup: /system.slice/ovlm-ra.service
                  └─15053 /usr/lib64/ovlm/java/jdk1.8.0_60/bin/java -jar
                     /usr/lib64/ovlm/ra/resource-agent-rest-impl-1.0.0-SNAPSHOT-microservice.jar

Mar 24 13:51:07 ovlmruntime systemd[1]: Starting OVLM Resource Manager Agent...
Mar 24 13:51:07 ovlmruntime systemd[1]: Started OVLM Resource Manager Agent.
Mar 24 13:51:07 ovlmruntime ovlm-ra[15050]: Starting ovlm-ra... [ OK ]
]
```

Troubleshooting Openet Weaver

This section provides information that can help you resolve some issues with Openet Weaver if they arise.

The log files are the best source of information when issues occur. Openet Weaver logs are stored in the following location:

/var/log/ovlm/

There are also logs for each Openet Weaver microservice or component located in the component subdirectory as shown below:

/var/log/ovlm/<component_name>

The paths to the microservice logs are listed in the following table.

Table 50: Microservice Log Locations

Microservice	Path
Config Manager	/var/log/ovlm/cm
Front End	/var/log/ovlm/fe
Resource Agent	/var/log/ovlm/ra
Resource Manager	/var/log/ovlm/rm
Workflow Engine	/var/log/ovlm/wf
Inventory Instance Manager	/var/log/ovlm/iim
Fault Manager	/var/log/ovlm/fm
Performance Manager	/var/log/ovlm/pm
VIM abstraction Layer	/var/log/ovlm/val

Microservice	Path
VNF-M GUI	var/log/ovlm/vnfm-gui
Deployment Manager	/var/log/ovlm/dm

The troubleshooting issues, causes, and resolutions are in the following topics:

Troubleshooting Installation Issues

This topic describes Installation issues you might encounter with Openet Weaver and how to resolve them. It covers the following issues:

- *Openet Weaver Installer failed*
- *Product RPM can not be installed*
- *Resource Agent RPM can not be installed*
- *Openet Weaver service can not be started*
- *Openet Weaver service starts then stops after several seconds*
- *Can not deploy Openet Weaver topology*
- *Can not reach an Openet Weaver service on specified port*
- *Computer stops responding while installing Openet Weaver on VM*
- *Failure when encrypting password*
- *Installation of Resource Agent Failed in Deployment Manager*

Problem: Openet Weaver Installer Failed

You might encounter an issue where installation fails with the following traceback:

```
./ovlm-deploy.sh -u root -t ovlm-install-sample.yml

Traceback (most recent call last): File "./installer/ovlm-deploy.py", line
662, in <module>
    execute(deploy_workflow_engine_hpoo, hosts=workflow_engine_host_list)
    raise ValueError(err)ValueError: 'rpms/ovlm-workflow-en
gine-10.70-1.x86_64.rpm'
is not a valid local path or
glob.
```

Cause

The third party packages were not extracted from the TAR file into the installation folder.

Solution

Locate the third-party TAR package on the virtual machine and extract it to the installation folder on the file system using the following command:

```
tar -xvf ovlm-thirdparty-dependencies-<release_version>.tar
```

For more information about downloading the TAR package see [Extracting the TAR File](#)

Problem: Product RPM Can Not Be Installed

You might encounter an issue where the product rpm can not be installed and you see the following message:

```
$ yum install ovlm-resource-agent-1.1.0-dev.x86_64.rpm
Loaded plugins: langpacks, product-id, subscription-manager
You must be root to perform this command.
```

Causes and Solutions

The causes and solutions that would prevent the product RPM from installing are described in the following table.

Table 51: Product RPM Can Not Be Installed Causes and Solutions

Cause	Solution
You do not have sufficient privileges to install Openet Weaver.	install the RPM using root permissions as follows: <code>sudo yum install package_name</code>
The default repos are not enabled.	Verify that the default Centos repos are included to the repolist and enabled using the following command: <code>yum repolist all</code>
A more recent version of Openet Weaver is already installed.	It is recommended that you do not install an older version of an RPM over a more recent version.

Problem: Resource Agent RPM Can Not Be Installed

You might encounter an issue where the Resource Agent RPM cannot be installed.

Cause

The facter dependency can not be resolved because the puppet repo is not enabled.

Solution

To enable puppet repo run the following command:

```
sudo rpm -ivh https://yum.puppetlabs.com/puppetlabs-release-el-7.noarch.rpm
```

Problem: Openet Weaver Service Can Not Be Started

You might encounter an issue where an Openet Weaver service cannot be started.

Cause

Java is not installed.

Solution

install java by running the following command:

```
sudo yum install java-1.8.0
```

Problem: Service Starts Then Stops After Several Seconds

You might encounter an issue where an Openet Weaver service starts but then stops after several seconds.

Cause

The SpringBootApplication failed to start.

Solution

Verify that config /etc/ovlm/ra/ovlm-ra.yml is valid.

Problem: Can Not Deploy Openet Weaver Topology

You might encounter an issue where the Openet Weaver topology file cannot be deployed, and you see the following message:

Causes and Solutions

You can determine what caused the issue by looking at the error message that is generated. The following table describes the error messages, their causes, and the solutions.

Table 52: Can Not Deploy Topology File Causes and Solutions

Message	Cause	Solution
Fatal error: Failed to parse topology file	The ovlm-install.yml file is invalid.	Check the ovlm-install.yml file and fix any errors.
Fatal error: Topology Error	The ovlm-install.yml file has an invalid structure.	Check the ovlm-install.yml file structure and fix any errors. The following is an example of the structure of a partial topology file: <pre># Frontend Configuration frontend: hosts: - node-1 properties: server: port: 28050</pre>

Problem: Can Not Reach an Openet Weaver Service on Specified Port

Causes and Solutions

The following table describes the causes and solutions for this issue.

Table 53: Can Not Reach Openet Weaver on a Specified Port Causes and Solutions

Cause	Solution
An incorrect hostname, port, or both was specified.	Check the host and port parameters in the ovlm-install.yml file and compare to the port set in the configuration file /etc/ovlm/ra/ovlm-ra.yml.
The service is not started.	Check whether the service started by running the following command: <pre>systemctl status service_name</pre>
A firewall is enabled.	Check iptables on the remote host by running the following command: <pre>sudo iptables -L</pre>

Problem: Computer Stops Responding While Installing Openet Weaver on VM

You might encounter an issue where the computer hangs while you are installing Openet Weaver on a VM.

Cause

The computer you are using does not meet the system requirements.

Solution

Review the [hardware and software requirements](#) and make sure the machine you are using is up to specifications.

Problem: Failure When Encrypting Password

You might encounter an issue when running the password encryption utility where you see the following error message:

```
./utilities/cipher/encrypt_scripts/encrypt-password.sh -f ./ovlm-installer-properties.yml
java.security.InvalidKeyException: Illegal key size
    at javax.crypto.Cipher.checkCryptoPerm(Cipher.java:1039)
    at javax.crypto.Cipher.implInit(Cipher.java:805)
    at javax.crypto.Cipher.chooseProvider(Cipher.java:864)
    at javax.crypto.Cipher.init(Cipher.java:1396)
    at javax.crypto.Cipher.init(Cipher.java:1327)
    at com.openet.modules.ovlm.encryption.Encryptor.encrypt(Encryptor.java:61)
    at com.openet.modules.ovlm.encryption.Encryptor.encrypt(Encryptor.java:48)
    at com.openet.modules.ovlm.encryption.Encryptor.encryptPasswords(Encryptor.java:175)
    at com.openet.modules.ovlm.encryption.Encryptor.encryptHttpsConfiguration(Encryptor.java:184)
    at com.openet.modules.ovlm.encryption.Encryptor.encryptConfig(Encryptor.java:231)
    at com.openet.modules.ovlm.encryption.Encryptor.main(Encryptor.java:267)
```

Cause

The Java™ Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy files in the JAVA environment that are required to execute password encryption utility script are missing.

Solution

Install (or re-install) the encryption software as described in the topic [Installing Encryption Software](#).

Problem: Installation of Resource Agent Failed in Deployment Manager

You might encounter an issue when Deployment Manager is installing a Resource Agent as indicated by the example error response from Deployment Manager shown below:

```
data:
request: http://localhost:28130/api/v1/install
method: POST
error_msg: The service installation has failed.
error_code: SERVICE_INSTALL_FAILED
services:
- version: 1.8.0-SNAPSHOT
service_type: resource_agent
service_version: 1.8.2-2128
hosts:
- host_identifier: ra06.openet-telecom.lan
status: FAILED
exit_code: '1'
response:
error_msg:
- {'returncode': 255, 'stdout': '', 'command': 'sudo -E echo Hello
World', 'stderr': '', 'message': 'LOCAL ssh client exited with error',
'local_command': ['/usr/bin/ssh', '-p 22', '-o', 'BatchMode=yes',
'-o', 'StrictHostKeyChecking=no', '-t', '-q', '-o',
'UserKnownHostsFile=/dev/null', '-i', '/usr/lib64/ovlm/dm/r
```

```

repository-root/artifacts/configurations/installer_ssh_key/id_rsa',
' 'Ovlm_install@kl-005114-vm06.openet-telecom.lan' ', ''export
OVLM_SSH_USER=ovlmrsync && export OVLM_USER=ovlm && export OVLM_USER_GROUP=ovlm
&& sudo -E echo Hello World']] }
status: 400

```

Cause

The installer ssh key file might not valid for Resource Agent node.

Solution

Check the Deployment Manager log file in log directory <path_log>/dm/service-install/ for more details. The log file name contains the host identifier.

The following is an example of a log filename:

```
service-install-[ra06.openet-telecom.lan]-20171010-174401.432957.log
```

Troubleshooting Workflow Runtime Errors

This topic describes issues you might encounter when running workflows with Openet Weaver and how to resolve them. It covers the following issues:

- [*No Connection to the Front End When Running workflow submit Command*](#)
- [*instantiate_vnf failed*](#)
- [*is_vnf_up flow failed after VNF was instantiated*](#)
- [*Locked Files*](#)
- [*Locked VNF Instance*](#)
- [*Locked VNFC Instance*](#)
- [*VNF Instantiation Error Caused by Conflict with Already Instantiated Template*](#)
- [*Missing Action Block in the VNFC Metadata*](#)
- [*Java Command Not Found*](#)
- [*Openet Framework Installation Already Exists on Target Node*](#)
- [*WIM Procedure Execution Timeout*](#)
- [*Incorrect User or Group Parameter*](#)
- [*Missing installation_id Attribute in the VNFC Metadata File*](#)

Problem: No Connection to the Front End When Running workflow submit Command

You might see an error similar to the following when you attempt to run the workflow submit command from the Front End microservice:

```

$ ovlm-fe workflow submit -t default -v policy_vnf -i dublin -p fw_v1 -y
instantiate_vnf_without_vim
Error Status: 0. Detail: <urllib3.connection.VerifiedHTTPSConnection object
at 0x2391a50>: Failed to establish a new connection: [Errno 111] Connection
refused

```

The error indicates that you cannot connect to the Front End

Cause

The OVLMFE_BASEURL environment variable was not set or was set with an incorrect port number for the microservice.

Solution

Find the port number of the Openet Weaver Front End microservice and set the OVLMFE_BASEURL.

You can find the port number in the Front End YML configuration file, which has a default location of . The default location is /etc/ovlm/fe/ovlm-fe.yml.

Find the section frontend:url as shown in the example below:

```
...
frontend:
  url: http://192.168.122.52:28050
...
```

Set the environment variable as shown in the following example.

```
export OVLMFE_BASEURL=http://192.168.122.52:2805
```

You can verify that the variable was set by using the grep command:

```
$ env | grep OVLMFE
OVLMFE_BASEURL=http://192.168.122.52:28050
```

For more information about setting BASEURL values see [Setting Environment Variables](#).

Problem: instantiate_vnf Failed

You might encounter an issue where running the instantiate_vnf workflow seems to have failed when you get the status. In such a case you see the following message:

```
"instantiate_vnf" failed:

Error Status: 500.

Error Code: EXECUTE_WORKFLOW_FAILED.

Detail: The execution of the specified operation or workflow failed. Please
contact
your system administrators.
```

Cause

There was an attempt made to verify the workflow status immediately after instantiate_vnf was run.

Solution

Wait a few moments and try getting the workflow status again.

Problem: is_vnf_up Flow Failed After VNF Was Instantiated

You might encounter an issue where the is_vnf_up workflow failed after the VNF was instantiated. You see the following message:

```
ERROR: 'Is Vnfc Up' failed.
```

Cause

There was an attempt made to verify the workflow status immediately after instantiate_vnf was called.

The workflow is_vnf_up was started while the instantiate_vnf status was RUNNING.

Solution

Wait until the instantiate_vnf status is COMPLETED then run the is_vnf_up workflow.

Problem: Locked Files

When you run a Configuration Manager command and it is interrupted, stops running, or you break out of the operation, you might see the following error message when trying to run subsequent commands:

```
c.o.m.o.c.r.c.impl.VnfcRestController      : The requested resource cannot be
locked.
The resource is already being locked by another process.
```

Cause

Required files are still locked due to breaking out of an operation before completion.

Solution

Locate the following directory in Configuration Manager: /usr/lib64/ovlm/cm/repository-root. In that folder, delete any files named *lock*.

Problem: Locked VNF Instance

A VNF instance can become locked due to a workflow not updating the VNF instance state in Instance Inventory Manager (IIM). This problem can occur when you cancel a workflow, for example. The following examples show when this can occur.

```
error: 'Fail to create VNF event for Tenant Id: default, VNF Id: microblog,
VNF Instance Id: dc_1. The corresponding VNF Instance is in action: ''start''
and action status: ''in_progress''. '
```

Cause Example 1

You trigger the start_vnf workflow on VNF instance dc_1. Execution completes successfully. The Workflow Engine attempts to create a VNF event in IIM with an action_status of: success, but the attempt fails in IIM. The Front End returns success 201 with a warning.

```
ovlm-fe workflow submit -t default -v microblog -i dc_1 -y start_vnf -sync 1
data:
  flow_result:
    success: '"VNF Start passed successfully on all servers."'
    warning: '"Update inventory failed with Connection error: Connect to
ovm1:28120 [ovm1/127.0.0.1, ovm1/10.3.10.20] failed: Connection refused"'
    method: POST
    request: http://ovm1:28085/api/v2/tenants/default/vnfs/microblog/vnf_instances/dc_1/workflow_instances
    workflow_instance_status: COMPLETED
status: 201
```

Cause Example 2

You trigger the start_vnf workflow on VNF instance dc_1. Execution fails because one of the VNFCs did not start. The Workflow Engine attempts to create a VNF event in IIM with an action_status of: failed, but the attempt fails in IIM. The Front End returns error 500.

```
ovlm-fe workflow submit -t default -v microblog -i dc_1 -y start_vnf -sync 1
Error Status: 500.
Error Code: EXECUTE_WORKFLOW_FAILED.
Detail:
  Result:
    data:
      error: 'Lane( 1): Server ''10.3.11.193'''blogserver'' ERROR: Start
failed.  Update inventory failed with Connection error: Connect to ovm1:28120
```

```
[ovm1/127.0.0.1, ovm1/10.3.10.20] failed: Connection refused'
    method: POST
    request: http://ovm1:28085/api/v1/tenants/default/vnf
s/microblog/vnf_instances/dc_1/workflow_instances
    status: 500
```

Cause Example 3

You submit a request to upgrade VNF instance dc_1 asynchronously as shown below.

```
ovlm-fe workflow submit -t default -v microblog -i dc_1 -y upgrade_vnf -p
microblog_v1
```

While the workflow is running, you cancel the upgrade_vnf workflow running on dc_1 using the following command.

```
ovlm-fe workflow cancel -t default -v microblog -i dc_1 -w 112600123
```

Canceling the workflow results in the following response.

```
data:
method: PUT
request: http://localhost:8080/api/v2/tenants/default/vnfs/microblog/vnf_in
stances/dc_1/workflow_instances/1722606007/cancel
status: 200
```

Next, you attempt to start dc_1.

```
ovlm-fe workflow submit -t default -v microblog -i dc_1 -y start_vnf -sync 1
```

Starting dc_1 results in the following error response.

```
error: 'Fail to create VNF event for Tenant Id: default, VNF Id: microblog,
VNF Instance Id: dc_1. The corresponding VNFC identifier: ''nginx'' is in
action: ''start'' and action status: ''in_progress''.'
```

Cause

Possible causes:

- The workflow that was run against the VNF Instance just prior to the current workflow created an IIM event at VNF level with an action_status of in_progress, but failed to update the action_status when the workflow completed it's run.
- The VNF is locked because of a cancelled workflow, which caused an error in Instance Inventory Manager (IIM).

Solution

Use the vnf_event unlock command to unlock VNFs asynchronously:

```
ovlm-fe vnf_event unlock -t <tenant_id> -v <vnf_id> -i <vnf_instance_id>
[-action_status <failed/success>] [-health_status <up/down/unknown>]
```

The parameters are described in the following table:

Table 54: vnf_event create parameters

Parameter	Mandatory	Description
-t	Yes	The tenant id.
-v	Yes	The VNF id.

Parameter	Mandatory	Description
-i	Yes	The VNF instance id.
-action_status	No	<p>The action status you want to insert into IIM for the VNF. Possible values are:</p> <ul style="list-style-type: none"> • success • failed <p>The default value is failed.</p>
-health_status	No	<p>The health status you want to insert into IIM for the VNF. Possible values are:</p> <ul style="list-style-type: none"> • up • down • unknown <p>The default value is unknown.</p>

Example 1 Using Default Values for action_status and health_status

The following is an example of command usage. The action_status and health_status parameters are omitted from the command so the default values are used:

```
ovlm-fe vnf_event unlock -t default -v microblog -i dc_1
```

The following is an example response to the command:

```
data:
  workflow_instance_id: '113401313'
  request: http://kl-005145-vm1:28050/api/v2/tenants/default/vnfs/microblog/vnfs_instances/dc_1/vnf_event/unlock
  method: POST
  status: 201
```

Based on the above example, the following is an example of running the workflow status command with a successful response after running the vnf_event create command:

```
ovlm-fe workflow status -t default -v microblog -i dc_1 -w 113401313
data:
  workflow_instance_status: COMPLETED
  request: http://kl-005145-vm1:28050/api/v2/tenants/default/vnfs/microblog/vnfs_instances/dc_1/workflow_instances/113401313
  method: GET
  status: 200
```

Example 2 Using Defined Values for action_status and health_status

The following is an example of vnf_event unlock command usage with values defined for action_status and health_status:

```
ovlm-fe vnf_event unlock -t default -v microblog -i dc_2 -action_status success
 -health_status up
```

The following is an example response to the command:

```
data:
  workflow_instance_id: '101000756'
  request: http://kl-005145-vm1:28050/api/v2/tenants/default/vnfs/microblog/vnfs_instances/dc_2/vnf_event/unlock
  method: POST
  status: 201
```

```
nf_instances/dc_2/vnf_event/unlock
  method: POST
  status: 201
```

Based on the above example, the following is an example of running the `workflow status` command with a successful response after running the `vnf_event create` command:

```
ovlm-fe workflow status -t default -v microblog -i dc_2 -w 101000756
data:
  workflow_instance_status: COMPLETED
  request: http://kl-005145-vm1:28050/api/v2/tenants/default/vnfs/microblog/v
nf_instances/dc_2 /workflow_instances/101000756
  method: GET
  status: 200
```

Problem: Locked VNFC Instance

A VNFC instance can become locked due to a workflow not updating the VNFC instance state in the Instance Inventory Manager (IIM). The following example shows when this can occur.

```
error: 'Fail to create VNF event for Tenant Id: default, VNF Id: microblog,
VNF Instance Id: dc_1. The corresponding VNFC identifier: ''nginx'' is in
action: ''start'' and action status: ''in_progress''.'
```

Example

You trigger the `start_vnfc` workflow on VNF instance "dc_1" and specified VNFC Type "Blogserver". The Workflow Engine attempts to create a VNFC event in IIM with an `action_status` of: success or failed (If execution failed), but the attempt fails in IIM. The Front End returns error 500.

The following shows the result of a created VNC event in IIM with an action status of Success:

```
ovlm-fe vnfc_workflow submit -t default -v microblog -i dc_1 -c blogserver -y
  start_vnfc -sync 1

Error Status: 500.
Error Code: EXECUTE_WORKFLOW_FAILED.
Detail:
  Result:
    data:
      error: 'Connection error: Connect to ovm1 [ovm1/10.3.10.20] failed:
Connection refused'
      method: POST
      request: http://ovm1:28050/api/v2/tenants/default/vnfs/microblog/vnf_instances/dc_1/workflow_instances
      status: 500
```

The following shows the result of a created VNC event in IIM with an action status of Failure:

```
ovlm-fe vnfc_workflow submit -t default -v microblog -i dc_1 -c blogserver -y
  start_vnfc -sync 1

Error Status: 500.
Error Code: EXECUTE_WORKFLOW_FAILED.
Detail:
  Result:
    data:
      error: 'Lane( 1): Server ''10.3.11.193'''-'blogserver'' ERROR: Start
failed. 'Connection error: Connect to ovm1 [ovm1/10.3.10.20] failed: Connection
refused'
```

```

method: POST
request: http://ovm1:28050/api/v2/tenants/default/vnf
s/microblog/vnf_instances/dc_1/workflow_instances
status: 500

```

Cause

The previous workflow invocation created an IIM event at VNFC level with action_status = 'in_progress', but fails to update the action_status after the workflow invocation is completed.

Solution

Update the action_status manually with IIM Create VNFC Event API.

```

POST http://<ovlm-iim-url>/api/v1/tenants/<tenant_id>/vnfs/<vnf_id>/vnf
_instances/<vnf_instance_id>/vnfc/<vnfc_id>/vnfc_instances/<vnfc_
instance_id>/event{
  "action": <action> // mandatory
  "action_status": <action_status> // mandatory
  "workflow_instance_id": <workflow_instance_id> // mandatory
  "vnf_template_id": <template_id> /optional
}

```

Problem: VNF Instantiation Error Caused by Conflict with Already Instantiated Template

You run the instantiate_vnf_without_vim workflow for a VNF and receive the following error message:

```

error: The VNFC Id "nginx" of VNF template "microblog_v1" has been staged for
VNF instance "dc_1". Multiple instantiation of the same VNFC Id in the same
target node from different VNF Instances is not supported.

```

Cause

A VNFC with the same vnfc_id, vnf_id and VNF template, but with a different vnf_instance_id is already instantiated in the same target node

Example

You run the instantiate_vnf_without_vim workflow for a VNF, which in this example has the following details:

- VNF—microblog
 - VNF Template—microblog_v1
 - VNF Instance—dc_2

In the VNF Instance dc_2, a VNFC with vnfc_id nginx is set for target node 192.168.10.1. However, there is already a VNFC with the vnfc_id nginx instantiated from a different VNF Instance, dc_1, on the same target node. This VNFC instance nginx is from the same VNF (microblog) and VNF Template (microblog_v1).

Openet Weaver generates the following error:

```

data:

workflow_instance_status: FAILED
error_code: EXECUTE_WORKFLOW_FAILED
error_msg: 'Instantiation of VNF has failed due to vnfc_id: nginx,
vnfc_instance_id: 1'
flow_result:
vnfc_instances:
- vnfc_id: nginx
  vnfc_instance_id: '1'

```

```

hostname: 192.168.10.1
exit_code: '1'
response:
  error_msg:
    - stage step has failed.
    - Abort instantiation workflow.
    - ERROR The VNFC Id "nginx" of VNF template "microblog_v1" has been
staged for VNF instance "dc_1". Multiple instantiation of the same VNFC Id in
the same target node from different VNF Instances is not supported.
  request: http://mgmt-1.openet-telecom.lan:28085/api/v2/tenants/default/vnfs
/microblog/vnf_instances/dc_2/workflow_instances
  method: POST
status: 500

```

Solution

Ensure your VNF topology does not conflict with an existing, instantiated VNF. Terminate the the conflicting VNF Instance, then re-run the instantiation flow with the instance you want to instantiate.

If you receive the error message when running the instantiate_reference_vnf workflow, remove the instantiated conflicting VNF instance manually from the target node. Then instantiate the reference VNF using the same VNF_instance_id, or instantiate using a new VNF Template.

For more information, see [Creating and Uploading the VNF Topology YML File](#).

Problem: Missing Action Block in the VNFC Metadata

When running the instantiate_reference_vnf workflow, you might see a FAILED error similar to the one below:

```

$ ovlm-fe workflow submit -t default -v jvolte_vnf -i dublin_ref -p jvolte_v1
-y instantiate_reference_vnf -run_as fworks -k /tmp/fworks_id_rsa/fworks_id_rsa
data:
  workflow_instance_id: '125800344'
  request: http://192.168.122.52:28050/api/v2/tenants/default/vnfs/jvolte_vnf
/vnf_instances/dublin_ref/workflow_instances
  method: POST
status: 201

$ ovlm-fe workflow status -t default -v jvolte_vnf -i dublin -w 125800344
data:
  workflow_instance_status: FAILED
  error_code: EXECUTE_WORKFLOW_FAILED
  error_msg: Instantiation of Reference VNF has failed. Dynamic config reference
step has failed.
  flow_result:
    vnfc_instances:
      - vnfc_id: SIGM
        vnfc_instance_id: '1'
        hostname: rt-1
        exit_code: 1
        response:
          error_msg:
            - Action 'dynamic_config_reference_vnfc' is not found in
/usr/lib64/ovlm/reference_vnf/repository-root/preactive/tenants/de
fault/vnfs/jvolte_vnf/templates/jvolte_v1/vnfcs/SIGM/metadata/meta.yml
        request: http://192.168.122.52:28050/api/v2/tenants/default/vnfs/jvolte_vnf
/vnf_instances/dublin/workflow_instances/125800344
        method: GET
status: 500

```

Cause

A section of the Metadata was not defined properly. In this example, the dynamic_config_reference_vnfc section cannot be found.

Solution

Update the VNFC metadata for the corresponding VNF so that the all sections are defined properly. For information about creating metadata files see [Creating and uploading Metadata for the VNFC](#). For metadata information relating to creating a reference VNF using Weaver Integration Module, see *Setting up a Reference Installation -> Setting up a Reference Installation without a VNF template -> Setting up the VNFC metadata YML File* in the *Weaver Integration Module User Guide*.

The FAILED error might also occur if the VNF Template you are attempting to use with the instantiate_reference_vnf workflow was created using a version of Weaver Integration Module prior to v1.5. If that is the case, you must re-create the VNF template by setting up a reference installation as described in the section *Setting up a Reference Installation -> Setting up a Reference Installation without a VNF template* in the *Weaver Integration Module User Guide*.

Problem: Java command not found

You might find that running the instantiate_reference_vnf workflow throws error indicating that a java command cannot be found in the system as in the following example:

```
$ ovlm-fe workflow submit -t default -v jvolte_vnf -i dublin_ref -p jvolte_v1
-y instantiate_reference_vnf -run_as fworks -k /tmp/fworks_id_rsa/fworks_id_rsa
-sync 1
data:
  workflow_instance_status: FAILED
  error_code: EXECUTE_WORKFLOW_FAILED
  error_msg: Instantiation of Reference VNF has failed. Install reference step
has failed.
  flow_result:
    vnfc_instances:
      - vnfc_id: SIGM
        vnfc_instance_id: '1'
        hostname: rt-1
        exit_code: 1
        response:
          error_msg:
            - Install reference vnfc has failed.
            - '[Error] [Framework] - Install action ''Install'' SignalingManager.jar
failed. Error: stdout: stderr: /bin/sh: java: command not found'
          request: http://192.168.122.52:28050/api/v2/tenants/default/vnfs/jvolte_vnf
/vnf_instances/dublin_ref/workflow_instances
          method: POST
        status: 500
```

Cause

The java_home setting in the VNFC metadata does not match the existing Java home directory in the target node.

The following is an example of dynamic_config_reference_vnfc for a Signaling Manager VNFC metadata file:

```
...
dynamic_config_reference_vnfc:
  parameters:
    packages:
    services:
    configurations:
      - configuration/fusionworks.env.jinja
      - configuration/install.properties.jinja
      - configuration/config_file.yml.jinja
  fw_prod: /u01/sigm/fwProd
  fw_home: /u01/sigm/fwHome
```

```

user: fworks
group: fworks
java_home: /usr/java/jdk1.8.0_121/
timezone: Europe/Dublin
timeout: 120
procedure_name: dynamic_config_reference_sigm.py
...

```

In the above example, java_home value /usr/java/jdk1.8.0_121 should match the Java installation directory in the target node rt-1.

Solution

Find the correct path to java_home and update the metadata file.

You can find the path to java_home in the dynamic_config_vnfc and dynamic_config_reference_vnfc action block in the VNFC metadata file.

Problem: Openet Framework Installation Already Exists on Target Node

When you run the instantiate_vnf_without_vim workflow you might receive the following error message:

```

"error_msg": ["Install reference vnfc has failed.", "[Error]: [Framework] - Framework installation directories already exist on the target Resource Agent. FUSIONWORKS_HOME = /home/fworks/fwhome, FUSIONWORKS_PROD = /home/fworks/fwprod. .... "]

```

Cause

An Openet Framework installation already exists on the target node, which is causing an installation conflict.

Solution

Stop the existing Openet Framework processes and delete the existing FUSIONWORKS_HOME and FUSIONWORKS_PROD directories.

Problem: WIM Procedure Execution Timeout

When running the instantiate_reference_vnf workflow, you might receive a message that the flow failed similar to the following:

```

"error_msg": ["Update config reference vnfc has timed out. ..."]

```

Cause

Execution of the procedure script for the workflow failed due to a timeout because the procedure script requires more time than the timeout duration specified in the VNFC metadata file. In the above example, the procedure is Update Config Reference VNFC.

Solution

Update the timeout duration in the VNFC metadata file for the VNF associated with the VNFC and action. Then create a new VNF template to use for running the workflow.

For more information about VNFC metadata files see [Creating and Uploading Metadata for the VNFC](#).

Problem: Incorrect User or Group Parameter

When running Weaver VNF Instantiation workflow command, you receive the following error message:

```
JPCRF|1|ashwinb-vm03.openet-telecom.lan||1|{"error_msg":["install_vnfc step has failed.", "Abort instantiation workflow.", "[Error] Command source /usr/lib64/ovlm/ra/repository/preactive/tenants/default/vnfs/jpcrf_vnf/templates/jpcrf_vnf_v1/vnfcs/JPCRF/configuration/fusionworks.env; chown -R $USER:$GROUP /home/fworks/fwhome/.. failed.", "stdout: ", "stderr: chown: invalid group: 'fworks:builders'"] }@*,*@
```

In addition to the above message, you might see a similar error to the following in the procedure log file `/var/log/ovlm/ra/procedures/install_policy-JPCRF.log`:

```
2018-04-03 08:37:02,839-125803167-default- jpcrf - JPCRF -1-
ERROR - Command source
/usr/lib64/ovlm/ra/repository/preactive/tenants/default/vnf
s/jpcrf/templates/jpcrf_v1/vnfcs/JPCRF/configuration/fusionworks.env;
chown -R $USER:$GROUP /home/fworks/fwhome/.. failed.stdout:stderr: chown:
invalid group:'fworks:builders'
```

Cause

The user or group parameter in the VNFC metadata file is not valid because it is not specified correctly.

Solution

Update the user or group parameter in the VNFC metadata file.

For more information see *Setting Up a Reference Installation -> Setting Up a Reference Installation without a VNF Template -> Setting up the VNFC Metadata YML File* in the *Weaver Integration Module* user guide.

Problem: Missing installation_id Attribute in the VNFC Metadata File

When running the `instantiate_reference_vnf` VNF workflow you might receive the following error message:

```
"error_msg": ["ERROR Can't render template file
\"configuration/install.properties.jinja\". 'dict object' has no
attribute 'installation_id'"],
```

Cause

The VNFC metadata file does not have an `installation_id` attribute, which is required for template rendering in the Dynamic Config VNFC and the Dynamic Config Reference VNFC action.

Solution

Enter the `installation_id` attribute in the VNFC metadata file that caused the error.

The following is an example of the `installation_id` attribute in an Openet Policy Manager VNFC:

```
dynamic_config_vnfc:parameters:packages:services:configurations:fw_prod:
/u01/jenkins-pm/installations/fwProdfw_home:
/u01/jenkins-pm/installations/fwHomeuser: fw-weavergroup: buildersjava_home:
/usr/java/jdk1.8.0_92/timezone: Europe/Dublin
db_user: fwweaver01db_pass:
fwweaver01db_sid: e12101S
installation_id: pcrfWeavervolt_node:10
.0.151.49
timeout:120
procedure_name:
dynamic_config_policy.py
```

For more information see *Setting Up a Reference Installation -> Setting Up a Reference Installation without a VNF Template -> Setting up the VNFC Metadata YML File* in the *Weaver Integration Module* user guide.

Troubleshooting OpenStack-Related Issues

This topic describes OpenStack-related issues upi might encounter with Openet Weaver and how to resolve them. It covers the following issues:

- *VM Hostnames are Changed*

Problem: VM Hostnames are Changed

Out-of-the-box OpenStack VM hostnames might change after a VM reboot. For example, a host is named myhost, might have the domain, for example novalocal, appended after a reboot. In this example the hostname changes to the following

```
# hostname myhost.novalocal.
```

Although this is not strictly an Openet Weaver issue, it can cause problems in your deployment.

Cause

The OpenStack cloud init causes hostnames to change as described above upon reboot.

Solution

Set the hostname manually to the original name as in the following example

```
# hostname myhost
```

Also, make sure the `/etc/hosts` file reflects the correct host name.

Troubleshooting Deployment Manager Issues

This topic describes Deployment Manager issues you might encounter with Openet Weaver and how to resolve them. It covers the following issues:

- Database is locked by Deployment Manager

Problem: Database is locked by Deployment Manager

You might encounter an issue where a database is locked by Deployment Manager due to Deployment Manager terminating prematurely.

Cause

The DB lock file remained in FS because Deployment Manager terminated prematurely.

Solution

Remove the DB lock file manually and restart Deployment Manager.

Openet Weaver VNF-M API Reference

Openet Weaver REST API reference guides

- *Front End REST API*
- *Config Manager REST API*
- *Performance Manager REST API*
- *Fault Manager REST API*
- *Deployment Manager REST API*

Openet Weaver VNF-M CLI Reference

Openet Weaver CLI command reference guides

Below are links to the Config Manager and Front CLI reference guides

- [*Front End CLI reference*](#)
- [*Config Manager CLI reference*](#)
- [*Performance Manager CLI Reference*](#)
- [*Fault Manager CLI Reference*](#)
- [*Deployment Manager CLI Reference*](#)

Openet Weaver Python API Reference

Below are links to the Python API reference guides

- [*OVLM Python API Constants*](#)
- [*OVLM Python API Ninja Template*](#)
- [*OVLM Python API Linux Shell*](#)
- [*OVLM Python API Logger*](#)
- [*OVLM Python API Parameters*](#)
- [*OVLM Python API Software Packages*](#)
- [*OVLM Python API systemd_Services*](#)

Glossary

A

API	Application Programming Interface is a particular set of rules and specifications that a software program can follow to access and make use of the services and resources provided by another particular software program that implements that API. It serves as an interface between different software programs and facilitates their interaction, similar to the way the user interface facilitates interaction between humans and computers.
Application server	Application server is also known as web application server. A server program on a computer in a distributed network that works with a web (HTTP) server to provide the logic for an application program.

C

[Back to Top](#)

CA	Certificate authority. An entity issuing certificates verifying identity. CAs must be trustworthy for their certificates to be meaningful.
----	--

Cache	<p>In the context of a credit cache, the cache is an amount of credit (money or other units) held.</p> <p>A cache can be identified uniquely within the system. The lifetime of a cache begins when it is created in the credit cache. It ends when the cache is evicted or removed from the credit cache. During its lifetime, the suppliers of the cache can increase, decrease or remove it. Consumers of the cache can make reservations against, increase or decrease the cache. The relationship between a cache and the money taken from an external purse is defined purely by the business logic of the system.</p>
CE	Community Edition
Certificate	A document that identifies a specific entity and at a minimum, consists of that entity's public key, the period for which the certificate is valid, the name of the CA that issued the certificate, and the digital signature of the CA.
CLI	Command-line interface. A mechanism for interacting with a computer operating system or software by typing commands to perform specific tasks.
Client installation	A client is the requesting program or user in a client server relationship. A client installation includes only those files necessary to run the user interface.
Content Packs	Content Packs are containers for workflows developed in the workflow studio.
CORBA	Common Object Request Broker Architecture. A middleware architecture and specification for creating, distributing, and managing distributed program objects in a network. It allows programs at different locations and developed by different vendors to communicate in a network through an object request broker.
CPU	Central Processing Unit.
CSV	Comma Separated Value.
CVS	Concurrent Versioning System. A free software revision control system used by software developers.

D[Back to Top](#)

DBA	Database Administrator.
DEB	Debian packages file format. Debian packages are standard Unix ar archives. they typically contain two TAR files. One file contains program data and the other contains control information for the data.
DHCP	Dynamic Host Configuration Protocol. A computer networking protocol used by hosts (DHCP clients) to retrieve IP address assignments and other configuration information.

DPI	Deep Packet Inspection.
DN	Distinguished Name (DN) attributes uniquely identify entries in LDAP directories.
DNS	Domain Name Server. A domain name identifies one or more IP addresses. Domain names are used in URLs to identify particular Web pages. Every domain name has a suffix that indicates the top level domain (TLD) to which it belongs.

E[Back to Top](#)

EMS	Element Management System supports the configuration aspects of FCAPS.
-----	--

F[Back to Top](#)

FC	Fiber Channel. A high-speed network technology used for storage networking.
FEX	Fabric Extender.
FQDN	Fully Qualified Domain Name.
FTAM	OSI application layer protocol for File Transfer Access and Management.
FTP	File Transfer Protocol is a method of transferring files between two computers on the Internet.

G[Back to Top](#)

GA	Generally Available
git	Widely used version control system for software development. It is a distributed revision control system with an emphasis on speed, data integrity, and support for distributed, non-linear workflows.
GUI	Graphical User Interface.

H[Back to Top](#)

HA	High Availability. A system design approach and associated service implementation that ensures that a prearranged level of operational performance is met during a contractual measurement period
HBA	Host Bus Adaptor. It connects a host system (the computer) to other networks and storage devices.
HLD	High Level Design. A design document describing the overall design of a system without going into low level detail.

[Back to Top](#)

ICMP	Internet Control Message Protocol. An extension to the IP and defined by RFC 792. ICMP supports packets containing error, control, and informational messages.
IDE	Integrated Development Environment.
IP	The Internet Protocol (IP) is the principal communications protocol used for relaying datagrams (also known as network packets) across an internetwork using the Internet Protocol Suite. Responsible for routing packets across network boundaries, it is the primary protocol that establishes the Internet.
IOT	Inter-operability testing.

J

[Back to Top](#)

JAR	Java Archive. An archived file containing Java classes.
JDK	The Java Development Kit (JDK) is used to develop applications by using Java.
JKS	Java Key Store (JKS). A Sun proprietary key store algorithm.
JRE	The Java Runtime Environment (JRE) is used to run applications developed by using Java.
JSON	JavaScript Object Notation (JSON). A lightweight data-interchange format that is easy for humans to read and machines to parse.

K

[Back to Top](#)

KVM	Kernel-based Virtual Machine. Virtualization infrastructure for the Linux kernel that turns it into a hypervisor. It was merged into the Linux kernel mainline in kernel version 2.6.20, which was released on February 5, 2007. KVM requires a processor with hardware virtualization extension
-----	--

L

[Back to Top](#)

LAN	Local Area Network. LAN is a computer network that interconnects computers within a limited area such as a residence or office building.
LDAP	Lightweight Directory Access Protocol. An application protocol for reading and editing directories over an IP network.
LVM	Logical volume manager (for the Linux kernel). It manages disk drives and similar mass-storage devices.

M[Back to Top](#)

Maven	Apache build manager for Java projects.
MBR	Maximum Bit Rate.
Metadata	Metadata is used to instruct OVLM components how to preform a VNF lifecycle event.
MIB	Management Information Base is a database of objects that can be monitored by a network management system.
Microservice	Larger functions are completed using smaller independant functions that communicate via API's

N[Back to Top](#)

Nagle-Algorithm	<p>Nagle's document, Congestion Control in IP/TCP Internetworks (RFC 896) describes what is called the small packet problem, where an application repeatedly emits data in small chunks, frequently only 1 byte in size. Since TCP packets have a 40 byte header (20 bytes for TCP, 20 bytes for IPv4), this results in a 41 byte packet for 1 byte of useful information, a huge overhead. This situation often occurs in Telnet sessions, where most keypresses generate a single byte of data that is transmitted immediately. Worse, over slow links, many such packets can be in transit at the same time, potentially leading to congestion collapse.</p> <p>Nagle's algorithm works by combining a number of small outgoing messages, and sending them all at once. Specifically, as long as there is a sent packet for which the sender has received no acknowledgment, the sender should keep buffering its output until it has a full packet's worth of output, so that output can be sent all at once.</p>
NAI	Network Access Identifier. NAI is a standard way of identifying users who request access to a network.
NAS	Network Attached Storage. File-level computer data storage server connected to a computer network providing data access to a heterogeneous group of clients.
NASREQ	Diameter Network Access Server Application. A Diameter server for SIP based IP multimedia services (RFC 4740). This application also provides the Diameter client with functions that go beyond. Provides AAA functionality to network devices. the typical authorization and authentication, such as the ability to.
NE	Network Element. A network element is the generic term given to a part of a telephony network, such as a switch, router, and so on.
NEBS	Network Equipment-Building System is the most common set of safety, spatial and environmental design guidelines applied to telecommunications equipment.

Network Controller	Functional block that centralizes some of all of the control and management functionality of a network domain and may provide an abstract view of its domain to other functional blocks via well-defined interfaces network forwarding path : ordered list of connection points forming a chain of NFs along with policies associated to the list.
Network Function (NF)	Functional block within a network infrastructure that has well-defined external interfaces and well-defined functional behaviour. Note: In practical terms, a Network Function is today often a network node or physical appliance.
Network Function Virtualisation (NFV)	Principle of separating network functions from the hardware they run on by using virtual hardware abstraction.
Network Function Virtualisation Infrastructure (NFVI)	Totality of all hardware and software components that build up the environment in which VNFs are deployed. Note: The NFV-Infrastructure can span across several locations e.g places where data centres are operated. The network providing connectivity between these locations is regarded to be part of the NFV Infrastructure. NFV-Infrastructure and VNF are the top-level conceptual entities in the scope of Network Function Virtualisation. All other component are sub-entities.
Network Function Virtualisation Infrastructure (NFVI) components	NFVI hardware resources that are not field replaceable, but are distinguishable as COTS components at manufacturing time.
Network Functions Virtualisation Infrastructure Node (NFVI-Node)	Physical device[s] deployed and managed as a single entity, providing the NFVI Functions required to support the execution environment for VNFs.
Network Function Virtualisation Infrastructure Point of Presence (NFVI-PoP)	N-PoP where a Network Function is or could be deployed as Virtual Network Function (VNF).
Network Function Virtualisation Management and Orchestration (NFV-MANO)	Functions collectively provided by NFVO, VNF-M and VIM.
Network Functions Virtualisation Management and Orchestration Architectural Framework (NFV-MANO)	Collection of all functional blocks (including those in NFV-MANO category as well as others that interwork with NFV-MANO category as well as others that interwork with NFV-MANO), data repositories used by these functional blocks and reference points and interfaces through which these functional blocks exchange information for the purpose of managing and orchestrating NFV.
Network Functions Virtualisation Orchestrator (NFVO)	Functional block that manages the Network Service (NS) lifecycle and coordinates the management of NS lifecycle, VNF lifecycle (supported by the VNF-M) and NFVI resources (supported by the VIM) to ensure an optimized allocation of the necessary resources and connectivity.
Network Interface Controller (NIC)	Device in a compute node that provides a physical interface with the infrastructure network.

Network Operator	Defined as an operator of an electronics communications network or part thereof. An association or organization of such network operators also falls within this category.
Network Point of Presence (N-PoP)	Location where a Network Function is implemented as either a Physical Network Function (PNF) or a Virtual Network Function (VNF).
Network Service	Composition of Network Functions and defined by its functional and behavioural specification. Note: The Network Service contributes to the behaviour of the higher layer service, which is characterised by at least performance, dependability, and security specification. The end-to-end network service behaviour is the result of the combination of the individual network function behaviours as well as the behaviours of the network infrastructure composition mechanism.
Network Service Descriptor	Template that describes the deployment of a Network Service including service topology (constituent VNFs and the relationships between them, Virtual Links, VNF Forwarding Graphs) as well as Network Service characteristics such as SLAs and any other artefacts necessary for the Network Service on-boarding and lifecycle management of its instances.
Network Service Orchestration	Subset of NFV Orchestrator function that are responsible for Network Service lifecycle management.
Network Service Provider	Type of Service Provider implementing the Network Service.
Network Stability	Ability of the NFV framework to maintain steadfastness while providing its function and resume its designated behaviour as soon as possible under difficult conditions, which can be excessive loads or other anomalies not exceeding the design limits.
NF Forwarding Graph	Graph of logical links connecting NF nodes for the purpose of describing traffic flow between these network functions.
NF Set	Collection of NFs with unspecified connectivity between them.
NFT	Network Flow Tool. Set of Openet-developed plug-ins for JMeter delivered by UTF.
NFS	Network File System. A distributed file system protocol that allows a user on a client computer to access files over a network, much like local storage is accessed.
NFV Framework	Totality of all entities, reference points, information models and other constructs define by the specifications published by the ETSI ISG NFV.

NFV Infrastructure (NFVI)	Totality of all hardware and software components which build up the environment in which VNFs are deployed. Note: The NFV-Infrastructure can span across several locations, i.e multiple N-PoPs. The network providing connectivity between these locations is regarded to be part of the NFV-Infrastructure.
NFV-Resource (NFV-Res)	NFV-Resources do exist inside the NFV-Infra and can be used by the VNF/VNSF to allow for their proper execution.
NFVI Component	NFVI hardware resource that is not field replaceable, but is distinguishable as a COTS component at manufacturing time.

O[Back to Top](#)

OAM	Operations, Administration, and Maintenance. OAM is the processes, activities, tools, standards, and so on, involved with operating, administering, managing and maintaining any system. This commonly applies to computer networks or computer hardware.
OAM Center	A web-based user interface that provides OAM features, such as viewing alarms, KPIs, and log messages, and searching for subscriber transactions.
OCF	Open Cluster Framework. Used within the context of cluster resource management.
OCLI	Openet Command Line Interface. This CLI allows you to perform many of the functions offered by the Management Center and OAM Center, by using the command line.
OSI	Open Systems Interconnection-An effort to standardize networking. The OSI model is a way of sub-dividing a communications system into smaller parts called layers. A layer is a collection of similar functions that provide services to the layer above it and receives services from the layer below it. On each layer, an instance provides services to the instances at the layer above and requests service from the layer below.
OSS	Operational Support System. A collection of software applications supporting network processes.
OVA	Open Virtualization.
OVLM	Openet Virtual Lifecycle Manager. This is now known as Weaver.

P[Back to Top](#)

Physical Network Function (PNF)	Implementation of a NF via a tightly coupled software and hardware system.
---------------------------------	--

Q[Back to Top](#)

QoS	Quality of Service. To quantitatively measure quality of service, several related aspects of the network service are often considered, such as error rates, bit rate, throughput, transmission delay, availability, and so on. Quality of service is particularly important for the transport of traffic with special requirements.
-----	---

R[Back to Top](#)

Red Hat Enterprise Advanced Platform	Advanced Platform Enterprise Linux distribution offered by Red Hat.
Redirect Agent	An agent with an understanding of the Diameter application (so it must be a proxy for a specific application(s)) that responds with redirects to request messages. A redirect agent must advertise support for these applications in the capabilities exchange.
Relay Agent	An agent with no understanding of Diameter applications. They advertise support for a special application id to indicate that they can forward messages for any application.
Reservation	A number of currency units that can be spent by a subscriber on a particular service type, where the amount has been reserved for use for this service type.
Resiliency	Ability of the NFV framework to limit disruption and return to normal or at a minimum acceptable service delivery level in the face of a fault, failure, or an event that disrupts the normal operation.
REST	The architectural style of the World Wide Web. It consists of a coordinated set of architectural constraints applied to components, connectors, and data elements within a distributed system. REST focuses on component roles, the constraints upon their interaction with other components, and their interpretation of significant data elements. It ignores the details of component implementation and protocol syntax. REST facilitates performance, Scalability, Simplicity, Modifiability, Visibility, Portability, and Reliability.
RFC	Request for Comments. A memorandum published by the Internet Engineering Task Force (IETF) describing methods, behaviors, research, or innovations applicable to the working of the Internet and Internet-connected systems. The IETF adopts some of the proposals published as RFCs as Internet standards.
RHEL	Red Hat Enterprise Linux. RHEL is a Linux distribution developed by Red Hat and targeted toward the commercial market.
RPM	Package management system.

RR

Round Robin as a scheduling policy routes requests to a set of peers in a domain.

S[Back to Top](#)

SASL	Simple Authentication and Security Layer (SASL) is a framework for authentication and data security in internet protocols. It decouples authentication mechanisms from application protocols.
Scaling	Ability to dynamically extend/reduce resources granted to the Virtual Network Function (VNF) as needed. Note: This includes scaling up/down and scaling out/in.
Scaling Out/In	Ability to scale by adding/removing resource instances (e.g VM)
Scaling Up/Down	Ability to scale by changing allocated resource, e.g increase/decrease memory, CPU capacity or storage size.
Service	Component of the portfolio of choices offered by service providers to a user, a functionality offered to a user. Note: A user may be an end-customer, a network or some intermediate entity.
Service Continuity	Continuous delivery of service in conformance with service's functional and behavioural specification and SLA requirements, both in the control and data planes, for any initiated transaction or session till its full completion even in the events of intervening exceptions or anomalies, whether scheduled or unscheduled, malicious, intentional or unintentional. Note: From an end-user perspective, service continuity implies continuation of ongoing communication sessions with multiple media traversing different network domains (access ,aggregation and core network) or different user equipment. Note: End to end service continuity requires that the service is delivered with service quality defined by an SLA. This is true regardless if the service is delivered via a non-virtual network, virtual network or a combination.
Service Level Agreement (SLA)	Negotiated agreement between two or more parties, recording a common understanding about the service and/or service behaviour(e.g availability, performance, service continuity, responsiveness to anomalies ,security ,serviceability ,operation)offered by one party to another ,and the measurable target values characterizing the level of services. Note: The scope of the above definition does not include business aspects of the SLA.

Service Provider	Defined as a company or organization, making use of an electronics communications network or part thereof to provide a service or services on a commercial basis to third parties.
Shell Script	A command script used to run the Management Center on UNIX platforms.
SNMP	Simple Network Management Protocol is a protocol governing network management and the monitoring of network devices and their functions.
SNMP Agent Port	An SNMP agent is needed to act as a server so that something which queries the MIB can send its queries and receive a response. The SNMP agent acts as the responder to SNMP requests for status information and sends traps to another application when state changes. The SNMP agent port is the port number for the SNMP agent. For the config port, event port and SNMP agent port, the port number range should be between 1024 and 65535. Any ports lower than 1024 are usually reserved.
SOAP	Simple Object Access Protocol is a means by which a program running on one operating system (OS) communicates with a program in the same or another OS by using HTTP or XML as the mechanisms for information exchange. SOAP specifies exactly how to encode an HTTP header and an XML file so that a program in one computer can call a program in another computer and pass required information.
SQL	Structured Query Language is a standard interactive and programming language for getting information from and updating a database. Queries take the form of a command language that lets you select, insert, update, find out the location of data, and so forth. There is also a programming interface.
SSL	Secure Sockets Layer, is a protocol used for transmitting private documents via the Internet. SSL uses a cryptographic system that avails of two keys to encrypt data—a public key known to everyone and a private or secret key known only to the recipient of the message.
SVN	Apache Subversion (SVN) is a software versioning and revision control system distributed as free software under the Apache License. Software developers use Subversion to maintain current and historical versions of files such as source code, web pages, and documentation.

T[Back to Top](#)

Tablespace Name	A tablespace is a logical division of a database. Tablespaces are used to group users or applications together for maintenance and performance benefits.
-----------------	--

Tenant	A tenant is a logical group of VNFs that may belong to a sample vendor, or managed by the same department of an operator, or even by data centre.
Tenant Domain	Domain that provides VNFs, and combinations of VNFs into Network Services, and is responsible for their management and orchestration, including their functional configuration and maintenance at application level.
TNS	TNS stands for Transparent Network Substrate. This is the software that Oracle uses to manage database connections.
Topology	Topology is the description of the layout of the network including nodes and how they are connected.
TPS	Transactions per second.
typedoc	A document type declaration, or DOCTYPE, is an instruction that associates a particular SGML or XML document (for example, a webpage) with a document type definition (DTD) (for example, the formal definition of a particular version of HTML).

U[Back to Top](#)

URI	Uniform Resource Identifier is a compact string of characters used for identifying an abstract or physical resource.
User Service	Component of the portfolio of choices offered by service providers to the end-users/customers/subscribers.
UTC	Coordinated Universal Time (abbreviated as UTC, and therefore often spelled out as Universal Time Coordinated, and sometimes as Universal Coordinated Time) is the standard time common to every place in the world. Formerly and still widely called Greenwich Mean Time (GMT) and also World Time, UTC nominally reflects the mean solar time along the Earth's prime meridian.

V[Back to Top](#)

VA	Virtual Application, a general term for a piece of software which can be loaded into a Virtual Machine. Note: A VNF is one type of VA.
VDSM	Virtual Desktop and Server Manager.

VDU	<p>Virtualisation Deployment Unit, a construct that can be used in an information model, supporting the description of the deployment and operational behaviour of a subset of a VNF, or the entire VNF if it was not componentized in subsets.</p> <p>Note: In the presence of a hypervisor, the main characteristics of a VDU is that a single VNF or VNF subset instance created based on the construct can be mapped to a single VM. A VNF may be modelled using one or multiple such constructs, as applicable.</p>
VIM	<p>Virtual Infrastructure Management, used to orchestrate the virtual machines (VMs) used in Network Function Virtualization (NFV).</p>
Virtual Link	<p>Set of connection points along with the connectivity relationships between them and any associated target performance metrics(e.g bandwidth, latency, QoS)</p> <p>Note: The Virtual Link can be interconnected two or more entities(VNF components, VNFs, or PNFs) and it is supported by a Virtual Network (VN) of the NFVI.</p>
Virtual Network	<p>Virtual Network routes information among the network interfaces of VM instances and physical network interfaces, providing the necessary connectivity.</p> <p>Note: The Virtual Network is bounded by its set of permissible network interfaces.</p>
Virtual Server Cluster	<p>Openet Dynamic Context Router supports the assignment of a virtual identity to a Domain. Clients will only see the virtual identity, not the real identities of the servers the Domain.</p>
Virtualisation Container	<p>Partition of a compute node that provides an isolated virtualized computation environment.</p> <p>Note: Examples of virtualization container includes virtual machine and OS container.</p>
Virtualised CPU (vCPU)	<p>Virtualised CPU created for a VM by a hypervisor.</p> <p>Note: In practice, a vCPU may be a time sharing of a real CPU and/or in the case if multi-core CPUs, it may be an allocation of one or more cores to a VM. It is also possible that the hypervisor may emulate a CPU instruction set such that the vCPU instruction set is different to the native CPU instruction set (emulation will significantly impact performance).</p>
Virtualised Storage (vStorage)	<p>Virtualised non-volatile storage allocated to a VM.</p>
Virtualised Switch (vSwitch)	<p>Ethernet switch implemented by the hypervisor that interconnects vNICs of VMs with each other and with the NIC of the compute node.</p>

VLAN	Virtual LAN. A group of hosts with a common set of requirements that communicate as if they were attached to the same broadcast domain regardless of their physical location. A VLAN has the same attributes as a physical LAN, but it allows end stations to be grouped together even if they are not located on the same network switch.
VM	Virtual Machine, a virtualized computation environment that behaves very much like a physical computer/server. Note: A VM has all its ingredients (processor, memory/storage, interfaces/ports) of a physical computer/server and is generated by a Hypervisor, which partitions the underlying physical resources and allocates them to Vms. Virtual Machines are capable of hosting a VNF Component (VNFC).
VM Image	A packaged virtual machine that is deployed to a hypervisor. The image is not bound to a particular virtual machine. Therefore, it does not have, for example, an IP address.
VM Instance	A deployed virtual machine image running on a hypervisor.
VM-FEX	Virtual Manager-Fabric Extension.
VN-Link	Virtual network link.
VNF	Virtualised Network Function , an implementation of an NF that can be deployed on a Network Function Virtualisation Infrastructure (NFVI)
VNF Forwarding Graph (VNF FG)	NF forwarding graph where at least one node is a VNF.
VNF Instance	A run-time instantiation of the VNF software, resulting from completing the instantiation of its components and of the connectivity between them, using the VNF deployment and operational information captured in the VNFD, as well as additional run-time instance-specific information and constraints.
VNF Package	Archive that includes a VNFD, the software image(s) associated with the VNF, as well as additional artefacts, e.g to check the integrity and to prove the validity of the archive.
VNF Set	Collection of VNFs with unspecified connectivity between them.
VNF package	A package is a snapshot of VNF configurations. They are software artefacts that be instantiated to a cluster of VMs. A VNF package is a concatenation of tenant-id, vnf-id and packaged label supplied by the VNF at development.
VNFC	Virtualised Network Function Component, an internal component of a VNF providing a VNF Provider a defined sub-set of that VNF's functionality, with the main characteristic that a single instance of this component maps 1:1 against a single Virtualisation Container.

VNFC Instance	An instance of a VNFC deployed in a specific Virtualisation Container instance. It has a lifecycle dependency with its parent VNF instance.
VNFD	Virtualised Network Function Descriptor, a configuration template that describes a VNF in terms of its deployment and operational behaviour, and is used in the process of VNF on-boarding and managing the lifecycle of a VNF instance.
VNF-M	Virtualised Network Function Manager, a functional block that is responsible for the lifecycle management of VNF.
VOIP	Voice over IP (VoIP, or voice over Internet Protocol) commonly refers to the communication protocols, technologies, methodologies, and transmission techniques involved in the delivery of voice communications and multimedia sessions over Internet Protocol (IP) networks, such as the Internet.
VRF	Virtual Routing and Forwarding is a technology that allows multiple instances of a routing table to coexist within the same router at the same time. Because the routing instances are independent, the same or overlapping IP addresses can be used without conflicting with each other.
VSAN	Virtual SAN

W[Back to Top](#)

WAR	Web ARchive. A WAR file is a standard JAR file, with a .war extension.
WF	Workflow.

X[Back to Top](#)

XML	Extensible Markup Language, which is a specification designed especially for Web documents enabling the definition, transmission, validation, and interpretation of data between applications.
-----	--

Y[Back to Top](#)

YAC	You are Always Connected.
-----	---------------------------

Z[Back to Top](#)

ZUD	Zero Unscheduled Downtime.
-----	----------------------------