Cisco Elastic Services Controller 5.10 User Guide

First Published: 2023-04-12

Last Modified: 2024-07-02

Americas Headquarters
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
http://www.cisco.com
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 527-0883

THE SPECIFICATIONS AND INFORMATION REGARDING THE PRODUCTS IN THIS MANUAL ARE SUBJECT TO CHANGE WITHOUT NOTICE. ALL STATEMENTS, INFORMATION, AND RECOMMENDATIONS IN THIS MANUAL ARE BELIEVED TO BE ACCURATE BUT ARE PRESENTED WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. USERS MUST TAKE FULL RESPONSIBILITY FOR THEIR APPLICATION OF ANY PRODUCTS.

THE SOFTWARE LICENSE AND LIMITED WARRANTY FOR THE ACCOMPANYING PRODUCT ARE SET FORTH IN THE INFORMATION PACKET THAT SHIPPED WITH THE PRODUCT AND ARE INCORPORATED HEREIN BY THIS REFERENCE. IF YOU ARE UNABLE TO LOCATE THE SOFTWARE LICENSE OR LIMITED WARRANTY, CONTACT YOUR CISCO REPRESENTATIVE FOR A COPY.

The Cisco implementation of TCP header compression is an adaptation of a program developed by the University of California, Berkeley (UCB) as part of UCB's public domain version of the UNIX operating system. All rights reserved. Copyright © 1981, Regents of the University of California.

NOTWITHSTANDING ANY OTHER WARRANTY HEREIN, ALL DOCUMENT FILES AND SOFTWARE OF THESE SUPPLIERS ARE PROVIDED “AS IS" WITH ALL FAULTS. CISCO AND THE ABOVE-NAMED SUPPLIERS DISCLAIM ALL WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, THOSE OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OR ARISING FROM A COURSE OF DEALING, USAGE, OR TRADE PRACTICE.

IN NO EVENT SHALL CISCO OR ITS SUPPLIERS BE LIABLE FOR ANY INDIRECT, SPECIAL, CONSEQUENTIAL, OR INCIDENTAL DAMAGES, INCLUDING, WITHOUT LIMITATION, LOST PROFITS OR LOSS OR DAMAGE TO DATA ARISING OUT OF THE USE OR INABILITY TO USE THIS MANUAL, EVEN IF CISCO OR ITS SUPPLIERS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Any Internet Protocol (IP) addresses and phone numbers used in this document are not intended to be actual addresses and phone numbers. Any examples, command display output, network topology diagrams, and other figures included in the document are shown for illustrative purposes only. Any use of actual IP addresses or phone numbers in illustrative content is unintentional and coincidental.

All printed copies and duplicate soft copies of this document are considered uncontrolled. See the current online version for the latest version.

Cisco has more than 200 offices worldwide. Addresses and phone numbers are listed on the Cisco website at www.cisco.com/go/offices.

Cisco and the Cisco logo are trademarks or registered trademarks of Cisco and/or its affiliates in the U.S. and other countries. To view a list of Cisco trademarks, go to this URL: https://www.cisco.com/c/en/us/about/legal/trademarks.html. Third-party trademarks mentioned are the property of their respective owners. The use of the word partner does not imply a partnership relationship between Cisco and any other company. (1721R)

© 2023 Cisco Systems, Inc. All rights reserved.

CONTENTS

About This Guide

This guide helps you to perform tasks such as lifecycle management operations, monitoring, healing and scaling of the VNFs.

- Audience, on page xiii

Audience

This guide is designed for network administrators responsible for provisioning, configuring, and monitoring VNFs. Cisco Elastic Services Controller (ESC) and the VNFs whose lifecycle it manages are deployed in a Virtual Infrastructure Manager (VIM). Currently OpenStack, VMware vCenter, VMware vCloud Director, and VMware NSX-T are the supported VIMs. The administrator must be familiar with the VIM layer, vCenter, OpenStack, and the commands used.

Cisco ESC is targeted for Service Providers (SPs) and Large Enterprises. ESC helps SPs reduce cost of operating the networks by providing effective and optimal resource usage. For Large Enterprises, ESC automates provisioning, configuring and monitoring of network functions.

Terms and Definitions

The below table defines the terms used in this guide.

Table 1: Terms and Definitions

Terms	Definitions
ESC	Elastic Services Controller (ESC) is a Virtual Network Function Manager (VNFM), performing lifecycle management of Virtual Network Functions.
ETSI	European Telecommunications Standards Institute (ETSI) is an independent standardization organization that has been instrumental in developing standards for information and communications technologies (ICT) within Europe.
ETSI Deployment Flavour	A deployment flavour definition contains information about affinity relationships, scaling, min/max VDU instances, and other policies and constraints to be applied to the VNF instance. The deployment flavour defined in the VNF Descriptor (VNFD) must be selected by passing the *flavour_id* attribute in the InstantiateVNFRequest payload during the instantiate VNF LCM operation.

Terms	Definitions
HA	ESC High Availability (HA) is a solution for preventing single points of ESC failure and achieving minimum ESC downtime.
KPI	Key Performance Indicator (KPI) measures performance management. KPIs specify what, how and when parameters are measured. KPI incorporates information about source, definitions, measures, calculations for specific parameters.
MSX	Cisco Managed Services Accelerator (MSX) is a service creation and delivery platform that enables fast deployment of cloud-based networking services for both Enterprises and Service Providers customers.
NFV	Network Function Virtualization (NFV) is the principle of separating network functions from the hardware they run on by using virtual hardware abstraction.
NFVO	NFV Orchestrator (NFVO) is a functional block that manages the Network Service (NS) lifecycle and coordinates the management of NS lifecycle, VNF lifecycle (supported by the VNFM) and NFVI resources (supported by the VIM) to ensure an optimized allocation of the necessary resources and connectivity.
NSO	Cisco Network Services Orchestrator (NSO) is an orchestrator for service activation which supports pure physical networks, hybrid networks (physical and virtual) and NFV use cases.
OpenStack Compute Flavor	Flavors define the compute, memory, and storage capacity of nova computing instances. A flavor is an available hardware configuration for a server. It defines the *size* of a virtual server that can be launched.
Service	A service consists of a single or multiple VNFs.
VDU	The Virtualisation Deployment Unit (VDU) is 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.
VIM	The Virtualized Infrastructure Manager (VIM) adds a management layer for the data center hardware. Its northbound APIs are consumed by other layers to manage the physical and virtual resources for instantiation, termination, scale in and out procedures, and fault & performance alarms.
VM	A Virtual Machine (VM) is an operating system OS or an application installed on a software, which imitates a dedicated hardware. The end user has the same experience on a virtual machine as they would have on dedicated hardware.
VNF	A Virtual Network Function (VNF) consists of a single or a group of VMs with different software and processes that can be deployed on a Network Function Virtualization (NFV) Infrastructure.
VNFC	A Virtual Network Function Component is (VNFC) a composite part of the VNF, synonymous with a VDU, which could be implemented as a VM or a container.
VNFM	Virtual Network Function Manager (VNFM) manages the life cycle of a VNF.

Related Documentation

The Cisco ESC doc set comprises of the following guides to help you perform installation, configuration; the lifecycle management operations, healing, scaling, monitoring and maintenance of the VNFs using different APIs.

Guide	**Information Provided in This Guide**
Cisco Elastic Services Controller Release Notes	Includes new features and bugs, known issues.
Cisco Elastic Services Controller Install and Upgrade Guide	Includes procedure for new installation and upgrade scenarios, pre and post installation tasks, and procedure for ESC High Availability (HA) deployment.
Cisco Elastic Services Controller User Guide	Includes lifecycle management operations, monitoring, healing and scaling of the VNFs.
Cisco Elastic Services Controller ETSI NFV MANO User Guide	Includes lifecycle management operations, monitoring, healing and scaling of the VNFs using the ETSI APIs.
Cisco Elastic Services Controller Administration Guide	Includes maintenance, monitoring the health of ESC, and information on system logs generated by ESC.
Cisco Elastic Services Controller NETCONF API Guide	Information on the Cisco Elastic Services Controller NETCONF northbound API, and how to use them.
Cisco Elastic Services Controller REST API Guide	Information on the Cisco Elastic Services Controller RESTful northbound API, and how to use them.
Cisco Elastic Services Controller ETSI REST API Guide	Includes information on the Cisco Elastic Services Controller ETSI APIs, and how to use them.
Cisco Elastic Services Controller Deployment Attributes	Includes information about deployment attributes used in a deployment datamodel.
Cisco Elastic Services Controller Open Source	Includes information on licenses and notices for open source software used in Cisco Elastic Services Controller.

Obtaining Documentation Request

For information on obtaining documentation, using the Cisco Bug Search Tool (BST), submitting a service request, and gathering additional information, see *What's New in Cisco Product Documentation*, at: http://www.cisco.com/c/en/us/td/docs/general/whatsnew/whatsnew.html.

Subscribe to *What's New in Cisco Product Documentation*, which lists all new and revised Cisco technical documentation, as an RSS feed and deliver content directly to your desktop using a reader application. The RSS feeds are a free service.

PART I

Introduction

CHAPTER 1

Elastic Services Controller Overview

Cisco Elastic Services Controller (ESC) is a Virtual Network Functions Manager (VNFM) managing the lifecycle of Virtual Network Functions (VNFs). ESC provides agentless and multi vendor VNF management by provisioning the virtual services. ESC monitors the health of VNFs , promotes agility, flexibility, and programmability in Network Function Virtualization (NFV) environments. It provides the flexibility to define rules for monitoring and associate actions that are triggered based on the outcome of these rules. Based on the monitoring results, ESC performs scale in or scale out operations on the VNFs. In the event of a VM failure ESC also supports automatic VM recovery.

ESC fully integrates with Cisco and other third party applications. As a standalone product, the ESC can be deployed as a VNF Manager. ESC integrates with Cisco Network Services Orchestrator (NSO) to provide VNF management along with orchestration. ESC as a VNF Manager targets the virtual managed services and all service provider NFV deployments such as virtual packet core, virtual load balancers, virtual security services and so on. Complex services include multiple VMs that are orchestrated as a single service with dependencies between them.

Key Features of Elastic Services Controller

- Provides open and modular architecture, which allows multi-vendor OSS, NFVO, VNF and VIM support.
- Provides end-to-end dynamic provisioning and monitoring of virtualized services using a single point of configuration.
- Provides customization across different phases of lifecycle management; while monitoring the VM, service advertisement, and custom actions.
- Provides agentless monitoring with an integrated Monitoring Actions (MONA) engine. The monitoring engine provides simple and complex rules, to decide scale in and scale out of VMs.
- Provides scale in and scale out options based on the load of the network.
- Deploys, reboots or redeploys VMs based on the monitoring errors and threshold conditions detected as part of healing (also known as recovery).
- Supports service agility by providing faster VNF deployment and life cycle management.
- Supports multi-tenant environments.

- Supports deploying VMs on multiple VIMs.
- Supports non admin roles for ESC users on OpenStack.
- Supports IPv6 on OpenStack.
- Supports dual stack network on OpenStack
- Supports REST and NETCONF / YANG interfaces providing hierarchical configuration and data modularity.
- Supports ETSI MANO interface for a subset of VNF lifecycle management operations.
- Supports ETSI performance reports.
- Supports deploying VMs on Single or multiple AWS VIMs.
- Supports deploying vApps on VMware vCloud Director VIM using both ESC REST and ETSI APIs.
- Supports deploying and monitoring D-MONA in an Active/Active setup. The Distributed Monitoring and Actions (D-MONA) is a standalone monitoring component for monitoring VNFs.
- Supports deploying brownfield VMs.
- Supports consistent ordering of resource values during scaling.

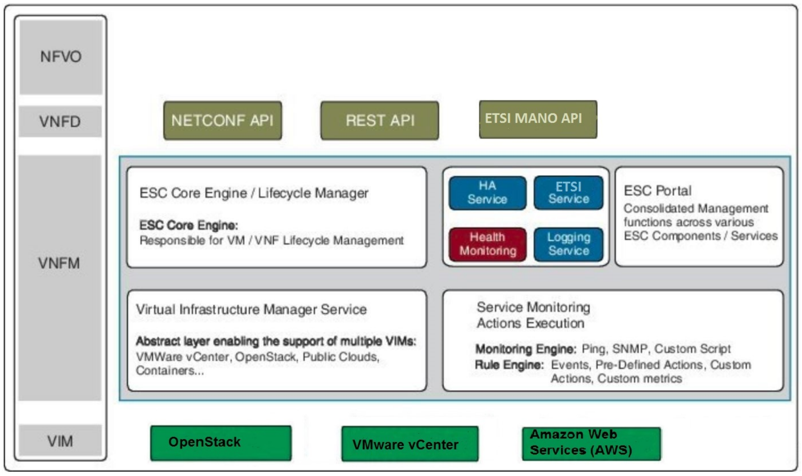
ESC Architecture

Cisco Elastic Services Controller (ESC) is built as an open and modular architecture, that allows multi-vendor support. It performs lifecycle management of the VNFs, that is, onboarding the VNFs, deploying, monitoring, and making VNF level lifecycle decisions such as healing and scaling based on the KPI requirements. ESC and its managed VNFs are deployed as VMs running within a Virtual Infrastructure Manager (VIM). Currently, OpenStack, VMware vCenter and AWS are the supported VIMs. The ESC core engine manages transactions, validations, policies, workflows, and VM state machines. The monitoring and actions service engine in ESC performs monitoring based on several monitoring methods. Events are triggered based on the monitoring actions. The monitoring engine also supports custom monitoring plugins.

ESC can be configured for High Availability. For details, see the Cisco Elastic Services Controller Install and Upgrade Guide.

ESC interacts with the top orchestration layer using the REST, NETCONF/YANG and ETSI NFV MANO NB APIs (ETSI APIs). The orchestration layer can be a Cisco NSO or any third party OSS or NFV Orchestrator. ESC integrates with NSO using NETCONF/YANG northbound interface support. A configuration template, Virtual Network Function Descriptor (VNFD) file is used to describe the deployment parameters and operational behaviors of the VNFs. The VNFD file is used in the process of onboarding a VNF and managing the lifecycle of a VNF instance. The figure below represents the Cisco Elastic Services Controller architecture.

Figure 1: Cisco Elastic Services Controller Architecture



Understanding the ESC Lifecycle

Cisco Elastic Services Controller (ESC) provides a single point of control to manage all aspects of VNF lifecycle for generic virtual network functions (VNFs) in a dynamic environment. It provides advanced VNF lifecycle management capabilities through an open, standards-based platform that conforms to the ETSI VNF management and orchestration (MANO) reference architecture.

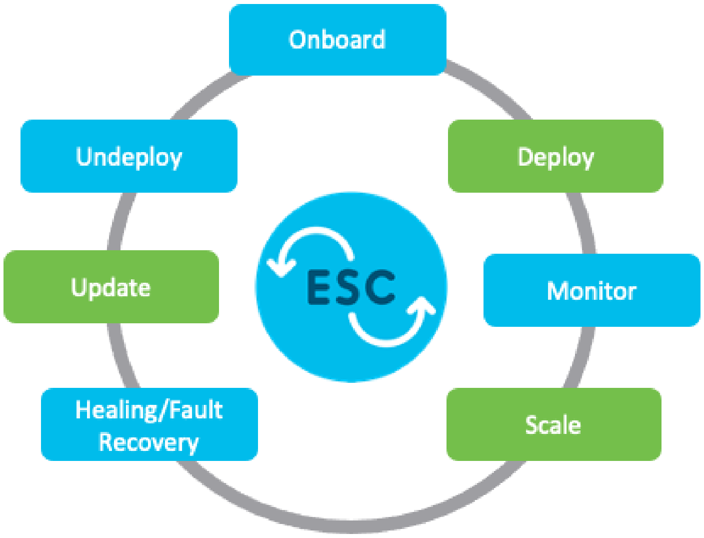
You can orchestrate VNFs within a virtual infrastructure domain—either on OpenStack or VMware vCenter. A VNF deployment is initiated as a service request. The service request comprises of templates that consist of XML payloads and configuration parameters.

Note Whilst hybrid deployments are supported, that is deploying VNFs across different VIMs and/or VIM types, the routing between these VMs is not the responsibility of ESC

ESC manages the complete lifecycle of a VNF. A VNF deployment is initiated as a service request through northbound interface or the ESC portal.

The figure explains the lifecycle management of ESC:

Figure 2: ESC's VNF Lifecycle



- **Onboarding**—In ESC, you can onboard any new VNF type as long as it meets the prerequisites for supporting it on OpenStack and VMware vCenter. For example on Openstack, Cisco ESC supports raw image, qcow2 and vmdk disk formats. ESC also supports config drive for the VNF bootstrap mechanism. You can define the XML template for the new VNF type to onboard the VNF with ESC.

  Using ETSI API, the VNF is onboarded to the NFVO. For more information, see prerequisites in the VNF Lifecycle Operations section in the *Cisco Elastic Services Controller ETSI NFV MANO User Guide*.

- **Deploying**—When a VNF is deployed, ESC applies Day Zero configuration for a new service. A typical configuration includes credentials, licensing, connectivity information (IP address, gateway), and other static parameters to make the new virtual resource available to the system. It also activates licenses for the new VNFs.

  An identifier is created using the ETSI API at this stage of the lifecycle. For more information, see the Creating VNF Identifier section in the *Cisco Elastic Services Controller ETSI NFV MANO User Guide*.

- **Monitoring**—ESC monitors the health of virtual machines using various methodologies including ICMP Ping, SNMP and so on. It tracks performance metrics such as CPU use, memory consumption, and other core parameters. The requester can specify all of the characteristics (for example, vCPU, memory, disk, monitoring KPIs, and more) typically associated with spinning up and managing a virtual machine in an XML template. It also provides an elaborate framework to monitor service performance-related metrics and other key parameters that you define.

- **Healing**—ESC heals the VNFs when there is a failure. The failure scenarios are configured in the KPI section of the data model. ESC uses KPI to monitor the VM and the events are triggered based on the KPI conditions. The actions to be taken for every event that is triggered is configured in the rules section during the deployment.

- **Updating**—ESC allows deployment updates after a successful deployment. You can either perform all the updates (that is, add or delete a vm_group, add or delete an ephemeral network in a vm_group, and add or delete an interface in a vm_group) in a single deployment or individually.

- **Undeploy**—ESC allows you to undeploy an already deployed VNF. This operation is either done using the northbound APIs or through the ESC portal.

  While deleting VNFs using the ETSI API, any associated identifier is also deleted.

Note For the complete VNF lifecycle operations using the ETSI API, see the *Cisco Elastic Services Controller ETSI NFV MANO User Guide*.

The following section explains how to deploy VNFs on OpenStack and VMware vCenter:

Deploying VNFs on OpenStack

In ESC, VNF deployment is initiated as a service request either originating from the ESC portal or the northbound interfaces. The service request comprises of templates that consist of XML payloads. These resources must either be available on OpenStack or can be created in ESC using the ESC portal or the northbound interfaces. For more information on managing resources in ESC, see Managing Resources Overview, on page 17. The *deployment data model* refers to the resources to deploy VNFs on OpenStack.

Based on how the resources are setup, you can deploy VNFs in one of the following ways:

Scenarios	Description	Resources	Advantages
Deploying VNFs on a single VIM by creating images and flavors through ESC	The *deployment data model* refers to the images and flavors created and then deploys VNFs.	Images and Flavors are created through ESC using NETCONF/REST APIs.	• The images and flavors can be used in multiple VNF deployments. • You can delete resources (images, flavors, and volumes) created by ESC.
Deploying VNFs on a single VIM using out-of-band images, flavors, volumes, and ports	The *deployment data model* refers to the out-of-band images, flavors, volumes, and ports in OpenStack and then deploys VNFs.	Images, Flavors, Volumes, and Ports are not created through ESC.	• The images, flavors, volumes, ports can be used in multiple VNF deployments. • You cannot delete resources that are not created by through ESC.
Deploying VNFs on multiple VIMs using out-of-band resources	The *deployment data model* refers to out-of-band images, flavors, networks and VIM projects and then deploys VNFs.	Images, Flavors, VIM projects (specified in the locators) and Networks are not created through ESC. They must exist out-of-band in the VIM.	You can specify the VIM (to deploy VMs) that needs to be configured in ESC within a deployment.

Note For more information on Deploying VNFs on OpenStack, see Deploying Virtual Network Functions on OpenStack, on page 107 .

Deploying VNFs on VMware vCenter

In ESC, VNF deployment is initiated as a service request either originating from the ESC portal or the Northbound interface. The service request comprises of templates that consist of XML payloads such as Networks, Images, and so on. These resources must be available on VMware vCenter. For more information on managing VM resources in ESC, see Managing Resources Overview, on page 17. The *deployment data model* refers to the resources to deploy VNFs on VMware vCenter.

When you deploy VNFs on VMware vCenter, you can either use the out-of-band images that are already available on VMware vCenter or create an image in the ESC portal or using REST APIs. For more information on creating images in the ESC portal, see Managing Images, on page 37. The *deployment data model* refers to these images to deploy VNFs.

Scenario	Description	Data model templates	Images	Advantages
Deploying VNFs by creating Images through ESC **Important** Images are also referred to as Templates on VMware vCenter.	The process of VNF deployment is as follows: **1.** VNF Deployment- The *deployment data model* refers to the images created and then deploys VNFs.	• *deployment data model* • *image data model*	Images are created through ESC using REST APIs.	• The images can be used in multiple VNF deployments. • You can add or delete image definitions through ESC.
Deploying VNFs on a single VIM using out-of-band images	**1.** VNF Deployment- The *deployment data model* refers to the out-of-band images on VMware vCenter and then deploys VNFs.	• *deployment data model* • Image on VMware vCenter	Images cannot be created or deleted through ESC.	• The images can be used in multiple VNF deployments. • You can view images through ESC portal. • During out-of -band deployment, you can choose images.

For more information on Deploying VNFs on VMware vCenter, see Images on VMware vCenter, on page 135.

CHAPTER 2

Elastic Services Controller Interfaces

Elastic Services Controller Interfaces

Cisco Elastic Services Controller (ESC) can be deployed in one of the following ways:

- As part of the Cisco Orchestration suite—ESC is packaged with Cisco Network Services Orchestrator (NSO), and available within Cisco Solutions such as Cisco Managed Services Accelerator (MSX).
- As a standalone product, ESC is available as a VNFM bundled with Cisco VNFs such as VPN, vRouter, vSecurity and many others.

When ESC is deployed as a part of the MSX, VPN, vRouter and so on, these applications interface with ESC through the Northbound APIs. ESC supports both REST and NETCONF northbound interfaces for operations and transactions. The ESC portal supports CRUD operations for some of the task for Virtual Network Function lifecycle management.

This chapter contains information about the Northbound APIs and the ESC portal.

Elastic Services Controller NB APIs

Elastic Services Controller (ESC) supports REST and NETCONF northbound interfaces for operations and transactions.

The northbound interfaces interact with the NB client, NSO or any OSS. For REST interface interactions, callbacks are triggered, and for NETCONF/YANG interface interactions, NETCONF notifications are triggered.

ESC also supports a REST API that conforms to the ETSI NFV MANO standards. Note that when using the ETSI API, the other ESC interfaces should be used for reading information only to preserve the integrity of the data models. The details of this API is outside the scope of this document. See the Cisco Elastic Services Controller ETSI NFV MANO User Guide for more information.

NETCONF/YANG Northbound API

ESC uses NETCONF to configure and manage the network and its devices. NETCONF is a network management protocol to install, manipulate, operate and delete the configuration of network devices. Cisco NSO communicates with ESC using the open NETCONF protocol and YANG based data models. ESC manages Virtual Network Functions at a device level, and NSO manages the entire network service lifecycle. Together, they make it a complete orchestration solution that spans across both physical and virtual infrastructure.

Note You can just type *esc_nc_cli --user <username> --password <password> command <file name>* instead of the complete path for any CRUD operations using the netconf CLI. For more information on CLI, see the *Cisco Elastic Services Controller Install and Upgrade Guide*.

Along with NETCONF notifications, the NETCONF/YANG model also provides operational data. You can run query to get details such as list of all tenants, networks, and deployments in ESC.

You can create a single NETCONF request to perform multiple actions. For more details, see Netconf Enhancement Request. The following is a NETCONF request to delete two tenants simultaneously:

```
<esc_datamodel xmlns="http://www.cisco.com/esc/esc">
<tenants>
  <tenant nc:operation="delete">
    <name>abc-mix-tenant1</name>
  </tenant>
  <tenant nc:operation="delete">
    <name>abc-mix-tenant2</name>
  </tenant>
</tenants>
</esc_datamodel>
```

Examples of NETCONF/YANG API are as follows:

NETCONF request to create a Tenant,

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="1">
  <edit-config>
    <source>
      <running />
    </source>
    <config>
      <esc_datamodel xmlns="http://www.cisco.com/esc/esc">
        <tenants>
          <tenant>
            <name>mytenant</name>
          </tenant>
        </tenants>
      </esc_datamodel>
    </config>
  </edit-config>
</rpc>
```

An escEvent of type CREATE_TENANT with a status of SUCCESS is sent to NETCONF subscribers once the configuration activation is completed. This indicates that the activation workflow is complete and the configuration resource is successfully created in the VIM.

NETCONF notification after a tenant is successfully created:

```
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2015-05-05T19:38:27.71+00:00</eventTime>
  <escEvent xmlns="http://www.cisco.com/esc/esc">
    <status>SUCCESS</status>
    <status_message>Tenant successfully created</status_message>
    <tenant>mytenant</tenant>
    <vm_source />
    <vm_target />
    <event>
      <type>CREATE_TENANT</type>
    </event>
  </escEvent>
</notification>
```

The operational data (Opdata) for the tenant shows the name and tenant_id. NETCONF request,

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="1">
  <get>
    <filter select="esc_datamodel/opdata/tenants/tenant[name='mytenant']" type="xpath" />
  </get>
</rpc>
```

NETCONF response,

```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="1">
  <data>
    <esc_datamodel xmlns="http://www.cisco.com/esc/esc">
      <opdata>
        <tenants>
          <tenant>
            <name>mytenant</name>
            <tenant_id>dccd22a13cc64e388a4b8d39e6a8fa7f</tenant_id>
        </tenant>
      </tenants>
    </esc_datamodel>
  </data>
</rpc-reply>
```

For more details on series of notifications, event failure notifications, and opdata, see the Cisco Elastic Services Controller API Guide.

The NETCONF API configuration and RPC calls are validated. If the request is not valid, it is rejected. The NETCONF API does not send any error code to NB, unlike REST (for example, REST sends 404 not found error).

A sample error message (rejected request) is as follows

```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="1">
    <rpc-error>
        <error-type>application</error-type>
        <error-tag>operation-failed</error-tag>
        <error-severity>error</error-severity>
        <error-path xmlns:esc="http://www.cisco.com/esc/esc"

xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">/nc:rpc/esc:filterLog</error-path>
        <error-message xml:lang="en">Exception from action callback: Error when handling
RPC
            calls: You can only query up to 30 logs.</error-message>
        <error-info>
            <bad-element>filterLog</bad-element>
        </error-info>
```

```
    </rpc-error>
</rpc-reply>
```

The no_gateway attribute allows ESC to create a subnet with the gateway disabled.

In the example below, the no_gateway attribute is set to true to create a subnet without gateway.

```
<networks>
    <network>
        <name>mgmt-net</name>
        <subnet>
            <name>mgmt-net-subnet</name>
            <ipversion>ipv4</ipversion>
            <dhcp>false</dhcp>
            <address>10.0.0.0</address>
            <no_gateway>true</no_gateway>
            <!-- DISABLE GATEWAY -->
            <gateway>10.0.0.1</gateway>
            <netmask>255.255.255.0</netmask>
        </subnet>
    </network>
</networks>
```

ESC shows OpenStack and VMware vCenter username in its Operational Data section.

The following configuration details are displayed in the Operational Data for,

OpenStack

- active_vim—displays the value as OpenStack
- os_auth_url—displays the OpenStack authentication URL
- admin_role—displays if the OpenStack user is an admin
- os_tenant_name—displays the tenant
- os_username—displays the Openstack user
- member_role—displays if the OpenStack user is a member

VMware vCenter

- active_vim—displays the value as VMware
- vcenter_ip—displays the vCentre IP address
- vcenter_port—displays if the vCentre port
- vcenter_username—displays the vCentre user

NETCONF Request to Configure Multiple Resources

A user can create a single NETCONF request to configure multiple resources.

Note A single request to configure multiple resources is supported using NETCONF only.

A single NETCONF request associates multiple resources based on the dependencies between the resources. For example, a subnet is dependent on a network, and a deployment is dependent on the image and flavor.

There are 2 types of dependencies in ESC.

1. Referential Dependency
2. Hierarchical Dependency

Referential Dependency

In referential dependency, one configuration has a reference to another configuration.

In the example below, deployment has referential dependency on image (test-mix-cirros) and flavor (test-mix-small). The image and flavor must be created before the deployment configuration.

```
  <images>
    <image>
        <name>test-mix-cirros</name>
...
</image>
</images>
<flavors>
    <flavor>
        <name>test-mix-small</name>
...
</flavor>
</flavors>
<tenants>
    <tenant>
        <name>test-mix-tenant</name>
        <deployments>
            <deployment>
                <name>dep</name>
                <vm_group>
                    <name>Group1</name>
                    <image>test-mix-cirros</image>
                    <flavor>test-mix-small</flavor>
...
</vm_group>
            </deployment>
        </deployments>
    </tenant>
</tenants>
```

Hierarchical Dependency

In hierarchical dependency, one configuration is within another configuration.

In the example below, the subnet (test-mix-shared-subnet1) is within the network (test-mix-shared-net1). The subnet has a hierarchical dependency on the network.

```
<esc_datamodel xmlns="http://www.cisco.com/esc/esc">
<networks>
    <network>
        <name>test-mix-shared-net1</name>
        <shared>true</shared>
        <admin_state>true</admin_state>
        <subnet>
            <name>test-mix-shared-subnet1</name>
            <ipversion>ipv4</ipversion>
            <dhcp>true</dhcp>
            <address>10.193.90.0</address>
```

```
                <netmask>255.255.255.0</netmask>
                <gateway>10.193.90.1</gateway>
            </subnet>
        </network>
    </networks>
    </esc_datamodel>
```

A hierarchical dependency is a subset of referential dependency. These configuration dependencies of the resources allow NETCONF to perform multiple configurations using a single request.

REST Northbound API

The REST API is a programmatic interface to ESC that uses a Representational State Transfer (REST) architecture. The API accepts and returns HTTP or HTTPS messages that contain JavaScript Object Notation (JSON) or Extensible Markup Language (XML) documents. You can use any programming language to generate the messages and the JSON or XML documents that contain the API methods or managed object (MO) descriptions.

The API model includes these programmatic entities:

- Classes—Templates that define the properties and states of objects in the management information tree (MIT).
- Methods—Actions that the API performs on one or more objects.
- Types—Object properties that map values to the object state (for example, equipmentPresence).

The ESC REST API contains headers, and other parameters. The header parameter contains a callback field with a URI. The client callback expects this value. A callback will not be performed if the URI field is not present.

REST API Documentation

You can access the REST API documentation directly from the ESC VM:

`http://[ESC VM IP]:8080/ESCAPI`

For detailed information, you can also see the Cisco Elastic Services Controller API Guide.

The REST API documentation provides details about all the various operations supported through the REST interface.

Example of REST APIs:

To create a tenant using REST:

```
POST /v0/tenants/123 HTTP/1.1
Host: client.host.com
Content-Type: application/xml
Accept: application/xml
Client-Transaction-Id: 123456
Callback:/createtenantcallback
<?xml version="1.0" encoding="UTF-8"?>
<tenant xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <name>tenant1</name>
   <enabled>true</enabled>
   <description>A description...</description>
</tenant>
```

REST response after a tenant is successfully created:

```
HTTP/1.1 201 OK
Content-Type: application/xml; charset=UTF-8
Content-Length: 200
Date: Sun, 1 Jan 2011 9:00:00 GMT
ESC-Transaction-Id: 123456
ESC-Status-Code: 200
ESC-Status-Message: Success …
<?xml version="1.0" encoding="UTF-8"?>
<tenant>
   <external_tenant_id>234243490854004</external_tenant_id>
   <internal_tenant_id>434344896854965</internal_tenant_id>
   <name>tenant1</name>
   <enabled>true</enabled>
   <description>A description...</description>
</tenant>
```

You cannot deploy VNFs with the same tenant name and deployment name using the REST API.

Note Further in this document, examples for scenarios will be provided either using REST or NETCONF/YANG, but not both.

ETSI NFV MANO Northbound API

The ETSI NFV MANO API (ETSI API) is another programmatic interface to ESC that uses the REST architecture. The ETSI MANO adheres to the standards defined by the European Telecommunications Standards Institute (ETSI), specifically around Management and Orchestration (MANO).

For more information see, the ETSI NFV MANO Northbound API Overview in the *Cisco Elastic Services Controller ETSI NFV MANO Guide*.

ETSI API Documentation

You can access the ETSI API documentation directly from the ESC VM:

`http://[ESC VM IP]:8250/API`

The ETSI API documentation provides details about all the various operations supported through the ESTI MANO interface.You can also see the Cisco ETSI API Guide for more information.

Elastic Services Controller Portal

The ESC portal is a simplified Web-based tool for an ESC administrator to create, read, update, or delete (CRUD) operations related to VNF lifecycle management. As an administrator you can create and view the real-time activities of ESC such as deploying, undeploying, healing and scaling.

The ESC portal is enabled by default while creating an ESC VM on OpenStack, VMware vCenter or KVM. For more information on enabling or disabling the ESC portal, see ESC Portal Dashboard.

To start, stop and restart the ESC Portal, do the following:

- To start the ESC portal, run *sudo escadm portal start*

- To stop the portal, run *sudo escadm portal stop*
- To restart the portal, run *sudo escadm portal restart*

Note The recommended browser screen size is 1920 pixels by 1080 pixels.

PART II

Managing Resources

CHAPTER 3

Managing Resources on OpenStack

Managing Resources Overview

Cisco Elastic Services Controller (ESC) resources comprise of images, flavors, tenants, volumes, networks, and subnetworks. These resources are the ones that ESC requests to provision a Virtual Network Function. These resources makeup the basic building blocks of a VNF service request, for example, Image is a bootable file system that can be used to launch VM instances. To manage these resources, you need to create the corresponding resources in ESC. These resource definitions exist or are created on OpenStack or VMware vCenter based on the provisioned infrastructure.

Depending upon the type of VNF deployment, you must ensure that the necessary resource definitions are available either on OpenStack or VMware vCenter. When you deploy VNFs on OpenStack you can either create these resource definitions in ESC or you have the option to use out-of-band image and flavor definitions that are already available on OpenStack. An out-of-band resource is a pre-existing resource. This resource is either created by ESC itself or by another source. For multiple VIM deployment, ESC uses out-of-band resources. ESC supports multiple VIM connectors for multi VIM deployments. The VIM connectors connect ESC to more than one VIM if configured.

ESC uses proxy server (if available) to reach OpenStack.

When you deploy VNFs on VMware vCenter, you can either use the out-of-band images that are already available on VMware vCenter, or create an image using the ESC portal, or using REST APIs. For more information on creating images using the ESC portal, see Managing Images, on page 37. The *deployment data model* refers to these images to deploy VNFs.

Note The procedure to create the resource definitions varies on OpenStack and VMware vCenter.

The resource (image, deployment and so on) names created from ESC must be globally unique.

The following table lists the different environments and the list of resource definitions that must be made available before VNF deployment:

Resource Definitions	OpenStack	VMware vCenter
Tenants	Creating and deleting tenant definitions is done in one of the following ways: • NETCONF API • REST API • ESC Portal	Not applicable.
Networks	Creating and deleting network definitions is done in one of the following ways: • NETCONF API • REST API • ESC Portal	Creating and deleting distributed port group definitions is done in one of the following ways: • NETCONF API • REST API • ESC Portal
Subnets	Creating and deleting subnet definitions is done in one of the following ways: • NETCONF API • REST API • ESC Portal	Not applicable.
Flavors	You can either use out-of-band flavor definitions that are already available in OpenStack or create flavor definitions in one of the following ways: • NETCONF API • REST API • ESC Portal	Not applicable.

Resource Definitions	OpenStack	VMware vCenter
Images	You can either use out-of-band image definitions that are already available on OpenStack or create image definitions in one of the following ways: • NETCONF API • REST API • ESC Portal	You can either use out-of-band image definitions that are already available on VMware vCenter or create image definitions in one of the following ways: • NETCONF API • REST API • ESC Portal
Volumes	You can use out-of-band volumes that are already available on OpenStack. For more information, see Managing Volumes, on page 38.	Not applicable.

The table below lists the OpenStack and VMware versions ESC supports.

Table 2: OpenStack and VMware supported versions

VIM	Version
OpenStack	• Newton • Ocata • Queens • Keystone v2 and v3 • Train
VMware	ESC supports VMware vCenter version 6.5, 6.7, 7.0.

For information on Installing ESC, see the Cisco Elastic Services Controller Install and Upgrade Guide.

Managing Resources on OpenStack

Managing Tenants

A tenant identifies a tenant organization or group that is associated with a set of administrators. When you create tenant definitions, the data stored on both regional and local clusters is segmented by tenant. A tenant cannot access the data of another tenant. You can use NETCONF/ REST interface, or the ESC portal to create a tenant definition through ESC.

Note Tenants are not supported on VMware vCenter.

Three types of tenants can be created in ESC:

1. Tenant on the VIM (ESC creates the tenant)—ESC creates and uses the tenant for deployments on default VIM. ESC can delete this tenant.
2. Pre-existing (out-of-band) tenant on the VIM—ESC does not create this tenant, but uses the tenant for deployments on default VIM only. The admin tenant, for example, is a pre-existing tenant, where the ESC itself is deployed. ESC supports deploying resources such as flavors, images and volumes on a pre-existing tenant that is identified by its name or UUID. ESC manages a pre-existing tenant for default VIM only. ESC cannot delete a pre-existing tenant.
3. Tenant within ESC—ESC creates a tenant within ESC, which is independent of any VIM. This tenant acts as the root tenant for deploying VMs on multiple VIMs.

Note that the tenant name must be unique.

Note ESC can create and manage resources such as tenants, networks, subnetworks, images and flavors on the default VIM only. Only deployments are supported on the non-default VIMs (other than the default VIM).

The following attributes manage the tenants in the data model.

- managed_resource attribute
- vim_mapping attribute

The table below further explains the tenant and the attribute mapping in the data model.

Tenant Type	managed_resource	vim_mapping	Description
Tenant on the VIM(created by ESC)	true	true	ESC creates a tenant on the VIM if the managed_resource attribute is set to true. By default, the managed_resource is true. The vim_mapping attribute is true.

```
<tenants>
    <tenant>
        <name>new-tenant</name>
       <managed_resource>true</managed_resource>

    </tenant>
</tenants>
```

Tenant Type	managed_resource	vim_mapping	Description
Pre-existing tenant on the VIM	false	true	For a pre-existing tenant, the managed_resource attribute is set to false. The vim_mapping attribute is true.

```
<tenants>
    <tenant>
        <name>pre-existing</name>
      <managed_resource>false</managed_resource>

    </tenant>
</tenants>
```

Sample data model using the tenant UUID

```
<tenants>
    <tenant>

<name>76eedcae-6067-44a7-b733-fc99a2e50bdf</name>

      <managed_resource>false</managed_resource>

    </tenant>
</tenants>
```

Tenant Type	managed_resource	vim_mapping	Description
Tenant within ESC	-	false	The vim_mapping attribute is set to false to create a tenant within ESC.

```
<tenants>
    <tenant>
        <name>esc-tenant-A</name>
        <vim_mapping>false</vim_mapping>
    </tenant>
</tenants>
```

Tenant Type	managed_resource	vim_mapping	Description
-	false	false	Tenant is not created. The request is rejected by ESC.

To deploy VMs on multiple VIMs of the same type (OpenStack VIMs), you must create a tenant with the vim_mapping attribute set to false. This tenant can be created independently or as part of the deployment. This creates a tenant within ESC, which acts as the root tenant for multi VIM deployments. A VIM locator attribute must be specified within the each vm group for multi VIM deployment. For more details, see Deploying VNFs on VMware vCenter VIM, on page 136.

Tenant Quotas

You can set the operational limit, known as quotas for the tenants created in ESC. The quotas can be set during the deployment using the deployment datamodel.

Note Tenant Quotas are not supported on pre-existing tenants and tenants within ESC.

The tenant supports the following quota settings for Compute (Nova) and Network (Neutron):

Compute settings:

- metadata_items
- floating_ips
- cores
- injected_file_path_bytes
- injected_files
- injected_file_content_bytes
- instances
- key_pairs
- ram
- security_groups
- security_group_rules

Compute settings:

- floatingip
- security_group_rule
- security_group
- network
- subnet
- port
- router

The deployment datamodel below shows the quota settings for the tenant.

```
<esc_datamodel xmlns="http://www.cisco.com/esc/esc">
 <tenants>
  <tenant>
   <name>tenant-quota-example</name>
    <extensions>
        <extension>
            <name>quota</name>
            <properties>
                <property>
                    <name>cores</name>
                    <value>26</value>
                </property>
                <property>
                    <name>metadata_items</name>
                    <value>260</value>
                </property>
                <property>
                    <name>floating_ips</name>
                    <value>26</value>
                </property>
                <property>
                    <name>injected_file_content_bytes</name>
```

```
                    <value>26000</value>
                </property>
                <property>
                    <name>injected_file_path_bytes</name>
                    <value>246</value>
                </property>
                <property>
                    <name>injected_files</name>
                    <value>26</value>
                </property>
                <property>
                    <name>instances</name>
                    <value>26</value>
                </property>
                <property>
                    <name>key_pairs</name>
                    <value>26</value>
                </property>
                <property>
                    <name>ram</name>
                    <value>26</value>
                </property>
                <property>
                    <name>security_groups</name>
                    <value>26</value>
                </property>
                <property>
                    <name>security_group_rules</name>
                    <value>26</value>
                </property>
                <property>
                    <name>floatingip</name>
                    <value>26</value>
                </property>
                <property>
                    <name>security_group_rule</name>
                    <value>26</value>
                </property>
                <property>
                    <name>security_group</name>
                    <value>26</value>
                </property>
                <property>
                    <name>network</name>
                    <value>26</value>
                </property>
                <property>
                    <name>subnet</name>
                    <value>26</value>
                </property>
                <property>
                    <name>port</name>
                    <value>26</value>
                </property>
                <property>
                    <name>router</name>
                    <value>26</value>
                </property>
            </properties>
        </extension>
    </extensions>
  </tenant>
 </tenants>
</esc_datamodel>
```

Note The property name in the deployment datamodel must match the compute and network setting names mentioned above. The tenant creation request is rejected.

Adding Tenants Using Northbound APIs

The following example explains how to create a tenant definition using NETCONF:

```
<rpc message-id="1" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <edit-config>
        <source>
            <running />
        </source>
        <config>
            <esc_datamodel xmlns="http://www.cisco.com/esc/esc">
                <tenants>
                    <tenant>
                        <name>mytenant</name>
                    </tenant>
                </tenants>
            </esc_datamodel>
        </config>
    </edit-config>
</rpc>
```

Note For more information about creating and deleting tenant definitions using NETCONF API, see Cisco Elastic Services Controller API Guide. To access the REST API documentation directly from the ESC VM, see REST Northbound API, on page 12. For more information on adding and deleting networks using the ESC portal, see Managing Resources Using ESC Portal, on page 367.

Updating Quotas for Tenants

You can update the quotas for the tenants created in ESC. The quota update is only allowed on the tenants for which managed_resource and vim_mapping attributes are set to true. However, updating other configurations, for example, name, vim_mapping, managed_resource, and description are not allowed.

The following deployment data model below shows the process of updating one or multiple properties of quota for the tenants.

```
<esc_datamodel xmlns="http://www.cisco.com/esc/esc">
 <tenants>
  <tenant>
   <name>ten-test-1</name>
    <managed_resource>true</managed_resource>
    <vim_mapping>true</vim_mapping>
    <extensions>
        <extension>
            <name>quota</name>
            <properties>
                <property>
                    <name>cores</name>
                    <value>15</value>
                </property>
                <property>
                    <name>ram</name>
```

```
                    <value>10000</value>
                </property>
            </properties>
        </extension>
    </extensions>
  </tenant>
 </tenants>
</esc_datamodel>
```

The following data model shows how to modify the core properties of the quota of a tenant.

```
<esc_datamodel xmlns="http://www.cisco.com/esc/esc">
 <tenants>
  <tenant>
   <name>ten-test-1</name>
    <managed_resource>true</managed_resource>
    <vim_mapping>true</vim_mapping>
    <extensions>
        <extension>
            <name>quota</name>
            <properties>
                <property>
                    <name>cores</name>
                    <value>20</value>
                </property>
                <property>
                    <name>ram</name>
                    <value>10000</value>
                </property>
            </properties>
        </extension>
    </extensions>
  </tenant>
 </tenants>
</esc_datamodel>
```

The following data model shows how to add a non-existing property to the quota of a tenant.

```
<esc_datamodel xmlns="http://www.cisco.com/esc/esc">
 <tenants>
  <tenant>
   <name>ten-test-1</name>
    <managed_resource>true</managed_resource>
    <vim_mapping>true</vim_mapping>
    <extensions>
        <extension>
            <name>quota</name>
            <properties>
                <property>
                    <name>cores</name>
                    <value>15</value>
                </property>
                <property>
                    <name>ram</name>
                    <value>10000</value>
                </property>
                <property>
                    <name>network</name>
                    <value>10</value>
                </property>
            </properties>
        </extension>
    </extensions>
  </tenant>
```

```
 </tenants>
</esc_datamodel>
```

The following example shows how to delete a property from the data model.

```
<esc_datamodel xmlns="http://www.cisco.com/esc/esc">
 <tenants>
  <tenant>
   <name>ten-test-1</name>
    <managed_resource>true</managed_resource>
    <vim_mapping>true</vim_mapping>
    <extensions>
        <extension>
            <name>quota</name>
            <properties>
                <property nc:operation="delete">
                    <name>cores</name>
                    <value>15</value>
                </property>
                <property>
                    <name>ram</name>
                    <value>10000</value>
                </property>
            </properties>
        </extension>
    </extensions>
  </tenant>
 </tenants>
</esc_datamodel>
```

Note The property gets deleted from the data model only. The quota values remain the same for that tenant in the OpenStack.

Updating Tenant Quotas with REST API

You can use REST API to create new tenants, or modify quotas of an existing tenant in ESC.

Method Type:

```
PUT
```

URL: */ESCManager/v0/tenants/[tenant_internal_id]*

HTTP Request Headers:

internal_tenant_id : is the tenant ID to be updated

callback : address and port to receive rest callback notifications

Content-Type : application/xml

Example of REST API while creating a tenant with quotas.

```
<tenant xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <name>tenant_internal_id</name>
    <managed_resource>true</managed_resource>
    <extensions>
        <extension>
            <name>quota</name>
            <properties>
                <property>
                    <name>port</name>
```

```
                    <value>17</value>
                </property>
                <property>
                    <name>ram</name>
                    <value>17021</value>
                </property>
                <property>
                    <name>cores</name>
                    <value>22</value>
                </property>
            </properties>
        </extension>
    </extensions>
</tenant>
```

Example of REST API while creating a tenant with modified or added quotas.

```
<tenant xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <name>tenant_internal_id</name>
    <managed_resource>true</managed_resource>
    <extensions>
        <extension>
            <name>quota</name>
            <properties>
                <property>
                    <name>port</name>
                    <value>20</value>
                </property>
                <property>
                    <name>ram</name>
                    <value>15000</value>
                </property>
                <property>
                    <name>network</name>
                    <value>5</value>
                </property>
            </properties>
        </extension>
    </extensions>
</tenant>
```

Managing Networks

In ESC, you can configure rich network topologies by creating and configuring networks and subnets, and then instructing either OpenStack or VMware vCenter services to attach virtual machines to ports on these networks.

OpenStack Network

In particular, OpenStack network supports each tenant to have multiple private networks, and allows tenants to choose their own IP addressing scheme, even if those IP addresses overlap with those used by other tenants. This enables very advanced cloud networking use cases, such as building multi-tiered web applications and allowing applications to be migrated to the cloud without changing IP addresses.

ESC supports the following networking functions:

- **Tenant Network**—A tenant network is created for a single network and all its instances. It is isolated from the other tenants.

- **Provider Network**—A provider network is created by the administrator. The attributes are mapped to the physical underlying network or a segment.

  The following attributes define a provider network:

  - network_type
  - physical_network
  - segmentation_id

- **External Network**—An external network typically provides Internet access for your instances. By default, this network only allows Internet access from instances using Network Address Translation (NAT). You can enable Internet access to individual instances using a floating IP address and suitable security group rules. The admin tenant owns this network because it provides external network access for multiple tenants.

ESC also supports ephemeral networks which are short-lived tenant networks purposely created during unified deployment and exists only during the lifetime of that deployment. For more details, see Unified Deployment Request.

Adding Networks Using Northbound APIs

The following example shows how to create a tenant network definition using NETCONF:

```
<?xml version="1.0" encoding="UTF-8"?>
<esc_datamodel xmlns:ns2="urn:ietf:params:xml:ns:netconf:notification:1.0"
xmlns:ns1="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:ns3="http://www.cisco.com/esc/esc_notifications"
xmlns:ns0="http://www.cisco.com/esc/esc" xmlns="http://www.cisco.com/esc/esc">
  <tenants>
   <tenant>
      <name>quicktest4</name>
  <networks>
   <network>
    <name>proto-tenant-network34</name>
    <shared>false</shared>
    <admin_state>true</admin_state>
   </network>
  </networks>
   </tenant>
  </tenants>
</esc_datamodel>
```

The following example shows how to create a subnet for tenant network definition using NETCONF:

```
<?xml version="1.0" encoding="UTF-8"?>
<esc_datamodel xmlns:ns2="urn:ietf:params:xml:ns:netconf:notification:1.0"
xmlns:ns1="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:ns3="http://www.cisco.com/esc/esc_notifications"
xmlns:ns0="http://www.cisco.com/esc/esc" xmlns="http://www.cisco.com/esc/esc">
  <tenants>
   <tenant>
      <name>quicktest4</name>
  <networks>
   <network>
    <name>proto-tenant-network27</name>
    <subnet>
     <name>proto-tenant-subnet4</name>
     <ipversion>ipv4</ipversion>
     <dhcp>true</dhcp>
```

```
      <address>172.16.0.0</address>
      <netmask>255.255.255.0</netmask>
      <gateway>172.16.0.1</gateway>
     </subnet>
    </network>
   </networks>
    </tenant>
   </tenants>
</esc_datamodel>
```

The following example shows how to create a simple provider network definition using NETCONF:

```
<?xml version="1.0"?>
<esc_datamodel xmlns:ns2="urn:ietf:params:xml:ns:netconf:notification:1.0"
 xmlns:ns1="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:ns3="http://www.cisco.com/esc/esc_notifications"
 xmlns:ns0="http://www.cisco.com/esc/esc" xmlns="http://www.cisco.com/esc/esc">
    <networks>
        <network>
          <name>test-net-12</name>
          <shared>true</shared>
          <admin_state>true</admin_state>
          <provider_physical_network>vm_physnet</provider_physical_network>
          <provider_network_type>vlan</provider_network_type>
          <provider_segmentation_id>200</provider_segmentation_id>
        </network>
    </networks>
</esc_datamodel>
```

The following example shows how to create a subnet for a provider network definition using NETCONF:

```
<?xml version="1.0" encoding="UTF-8"?>
<esc_datamodel xmlns:ns2="urn:ietf:params:xml:ns:netconf:notification:1.0"
 xmlns:ns1="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:ns3="http://www.cisco.com/esc/esc_notifications"
 xmlns:ns0="http://www.cisco.com/esc/esc" xmlns="http://www.cisco.com/esc/esc">
    <networks>
        <network>
            <name>test-net-12</name>
            <subnet>
                <name>test-net-12-subnet</name>
                <ipversion>ipv4</ipversion>
                <dhcp>false</dhcp>
                <address>172.16.0.0</address>
                <gateway>172.16.0.1</gateway>
                <netmask>255.255.255.0</netmask>
            </subnet>
        </network>
    </networks>
</esc_datamodel>
```

Starting ESC 5.9 release, users can update the shared property of an OpenStack network using ESC through REST API and NetConf interfaces.

To support this feature, a shared property is introduced at network level in ESC datamodel. Following is the sample deployment xml file :

Datamodel:

```
<?xml version="1.0" encoding="UTF-8"?>
<esc_datamodel xmlns="http://www.cisco.com/esc/esc">
   <tenants>
      <tenant>
         <name>admin</name>
         <networks>
            <network>
```

```
                <name>cirros-test-nw-1</name>
                <shared>false</shared>
                <admin_state>true</admin_state>
                <router_external>false</router_external>
                <provider_physical_network>physnet1</provider_physical_network>
                <provider_network_type>vlan</provider_network_type>
                <provider_segmentation_id>2000</provider_segmentation_id>
                <subnet>
                   <name>cirros-test-nw-1-subnet-1</name>
                   <ipversion>ipv4</ipversion>
                   <dhcp>false</dhcp>
                   <address>152.16.166.96</address>
                   <netmask>255.555.255.274</netmask>
                   <gateway>172.19.189.68</gateway>
                </subnet>
             </network>
          </networks>
       </tenant>
    </tenants>
</esc_datamodel>
```

Points to note while setting the shared property of a network through ESC:

- ESC does not allow the users to modify other properties of a network except shared property. If user tries to modify other properties of a network through service update request, ESC does not throw any error, instead ESC ignores the changes.
- ESC allows the users to set the shared property of a network to True only when the network is configured outside the deployment tag. If the user tries to create a network inside the ESC deployment tag with shared property as True, then ESC throws an error.
- Only an admin privileged tenant can update a network with the shared property as True or False.
- If a non-admin tenant tries to create or update the shared propert of the network, ESC throws one of the following errors:

```
-- Cannot create a 'shared' Tenant network.  Associated Tenant 'tenant-name' does not have
 admin privilege.
-- Cannot update the 'shared' flag of the Tenant network.  Associated Tenant 'tenant-name'
 does not have admin privilege.
```

The following example shows how to create vxlan-evpn for a provider network type in Cisco VIM:

```
<?xml version="1.0"?>
<esc_datamodel xmlns:ns2="urn:ietf:params:xml:ns:netconf:notification:1.0"
    xmlns:ns1="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:ns3="http://www.cisco.com/esc/esc_notifications"
    xmlns:ns0="http://www.cisco.com/esc/esc" xmlns="http://www.cisco.com/esc/esc">
    <networks>
        <network>
          <name>ProviderNetworkAttributes-vxlan-evpn</name>
          <shared>true</shared>
          <provider_network_type>vxlan-evpn</provider_network_type>
          <provider_segmentation_id>3010</provider_segmentation_id>
        </network>
    </networks>
</esc_datamodel>
```

The following example shows how to create an external network definition using NETCONF:

```
<network>
 <name>xyz-yesc-net-1</name>
 <shared>false</shared>
```

```
 <admin_state>true</admin_state>
 <router_external></router_external>
 <subnet>
  <name>xyz-yesc-subnet-1</name>
  <ipversion>ipv4</ipversion>
  <dhcp>true</dhcp>
  <address>172.16.0.0</address>
  <netmask>255.255.255.0</netmask>
  <gateway>172.16.0.1</gateway>
 </subnet>
</network>
```

Note For more information about creating and deleting network using NETCONF API, see Cisco Elastic Services Controller API Guide. To access the REST API documentation directly from the ESC VM, see REST Northbound API, on page 12. For more information on adding and deleting networks using the ESC portal, see Managing Resources Using ESC Portal, on page 367.

Managing Subnets

In ESC, a subnet is assigned to a virtual network. It specifies the IP address, the IP version for a network and so on. You can use the NETCONF/ REST interface to create subnet definitions.

Note Subnet is supported on OpenStack only.

Adding Subnet Definitions Using Northbound APIs

The following example shows how to create a subnet definition using NETCONF:

```
<rpc message-id="1" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <edit-config xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
 <target>
 <running/>
 </target>
 <config
 <esc_datamodel xmlns="http://www.cisco.com/esc/esc"
xmlns:ns0="http://www.cisco.com/esc/esc"
xmlns:ns3="http://www.cisco.com/esc/esc_notifications"
xmlns:ns1="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:ns2="urn:ietf:params:xml:ns:netconf:notification:1.0">
 <networks>
 <network>
 <name>mgmt-net</name>
 <subnet>
 <name>mgmt-net-subnet</name>
 <ipversion>ipv4</ipversion>
 <dhcp>false</dhcp>
 <address>172.16.0.0</address>
 <gateway>172.16.0.1</gateway>
 <netmask>255.255.255.0</netmask>
 </subnet>
 </network>
 </networks>
 </esc_datamodel>
```

```
 </config> </edit-config
</rpc>
```

The *no_gateway* attribute allows ESC to create a subnet with the gateway disabled.

In the example below, the *no_gateway* attribute is set to true to create a subnet without a gateway.

```
<networks>
        <network>
            <name>mgmt-net</name>
            <subnet>
                <name>mgmt-net-subnet</name>
                <ipversion>ipv4</ipversion>
                <dhcp>false</dhcp>
                <address>172.16.0.0</address>
                <no_gateway>true</no_gateway><!--  DISABLE GATEWAY -->
                <gateway>172.16.0.1</gateway>
                <netmask>255.255.255.0</netmask>
            </subnet>
        </network>
    </networks>
```

Note For more information about creating subnets using NETCONF API, see Cisco Elastic Services Controller API Guide. To access the REST API documentation directly from the ESC VM, see REST Northbound API, on page 12. For more information on adding and deleting networks using the ESC portal, see Managing Resources Using ESC Portal, on page 367.

Option To Assign DNS Name Server In Subnet

Tip Starting from ESC 5.9 release, users can configure the DNS name servers as a part of the subnet configuration through both Netconf and REST API interfaces

OpenStack allows the user to configure the subnets with one or more DNS name server IP addresses that maintain the order in which the DNS name server IP addresses are configured.

- A New DNS tag is added to the ESC Datamodel to configure DNS Name Servers for the subnet.
- NETCONF and REST API support to update only the DNS property of a subnet.
- Users cannot modify other properties of the subnet as the ESC ignores when a user tries to modify other properties of the subnet.

In New Data model, a list of DNS name servers under a subnet tag is introduced

New Datamodel:

```
<subnet>
   <name>test-subnet2</name>
   <ipversion>ipv4</ipversion>
   <dhcp>true</dhcp>
   <address>40.0.0.12</address>
   <netmask>255.255.255.0</netmask>
   <gateway>40.0.0.1</gateway>
   <dns>
      <server>24.0.0.23</server>
```

```
      <server>25.0.0.35</server>
   </dns>
   <no_gateway>false</no_gateway>
</subnet>
```

1: Configuring the DNS name servers in subnet outside the deployment.

- a) dep.xml file Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<esc_datamodel xmlns="http://www.cisco.com/esc/esc"
xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
               <networks>
                  <network>
                     <name>test-network2 </name>
                     <shared>false</shared>
                     <admin_state>true</admin_state>
                     <router_external>false</router_external>
                     <provider_physical_network>physnet_tenant</provider_physical_network>
                     <provider_network_type>vlan</provider_network_type>
                     <provider_segmentation_id>3112</provider_segmentation_id>
                    <subnet>
                        <name>test-subnet2</name>
                        <ipversion>ipv4</ipversion>
                        <dhcp>false</dhcp>
                        <address>40.10.11.22</address>
                        <netmask>255.255.255.0</netmask>
                        <gateway>40.10.11.1</gateway>
                        <dns>
                            <server>24.0.0.1</server>
                           <server>25.0.0.1</server>
                       </dns>
                     </subnet>
                  </network>
               </networks>
</esc_datamodel>
```

2: Configuring the subnet with DNS name servers inside a deployment

- a) dep.xml file Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<esc_datamodel xmlns="http://www.cisco.com/esc/esc"
xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
   <tenants>
      <tenant>
         <name>admin</name>
         <deployments>
            <deployment>
               <name>test-deployment1</name>
               <networks>
                  <network>
                     <name>test-network-dep-1</name>
                     <shared>false</shared>
                     <admin_state>true</admin_state>
                     <router_external>false</router_external>
                     <provider_physical_network>physnet_tenant</provider_physical_network>
                     <provider_network_type>vlan</provider_network_type>
                     <provider_segmentation_id>3012</provider_segmentation_id>
                     <subnet>
                        <name>test-subnet-dep-1</name>
                        <ipversion>ipv4</ipversion>
                        <dhcp>false</dhcp>
                        <address>40.10.11.2</address>
```

```
                        <netmask>255.255.255.0</netmask>
                        <gateway>40.10.11.1</gateway>
                        <dns>
                           <server>23.0.0.1</server>
                           <server>25.0.0.1</server>
                        </dns>
                     </subnet>
                  </network>
               </networks>
               <vm_group>
                  <name>test-dep-vm1</name>
                  <image>Automation-Cirros-Image</image>
                  <flavor>Automation-Cirros-Flavor</flavor>
                  <bootup_time>180</bootup_time>
                  <recovery_wait_time>180</recovery_wait_time>
                  <interfaces>
                     <interface>
                        <nicid>0</nicid>
                        <network>dnd-soltest-v4-private1</network>
                     </interface>
                  </interfaces>
                  <kpi_data>
                     <kpi>
                        <event_name>VM_ALIVE</event_name>
                        <metric_value>1</metric_value>
                        <metric_cond>GT</metric_cond>
                        <metric_type>UINT32</metric_type>
                        <metric_collector>
                           <nicid>0</nicid>
                           <type>ICMPPing</type>
                           <poll_frequency>3</poll_frequency>
                           <polling_unit>seconds</polling_unit>
                           <continuous_alarm>false</continuous_alarm>
                        </metric_collector>
                     </kpi>
                  </kpi_data>
                  <rules>
                     <admin_rules>
                        <rule>
                           <event_name>VM_ALIVE</event_name>
                           <action>ALWAYS log</action>
                           <action>FALSE recover autohealing</action>
                           <action>TRUE esc_vm_alive_notification</action>
                        </rule>
                     </admin_rules>
                  </rules>
                  <scaling>
                     <min_active>1</min_active>
                     <max_active>1</max_active>
                  </scaling>
               </vm_group>
            </deployment>
         </deployments>
      </tenant>
   </tenants>
</esc_datamodel>
```

3: Modifying the order of the DNS name server IP addresses in an existing subnet configuration with DNS name server entries

To modify the order of DNS entries, use nc:operation=”replace” as follows:

- a) dep.xml file Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<esc_datamodel xmlns="http://www.cisco.com/esc/esc"
xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
             <networks>
                <network>
                   <name>test-network2 </name>
                   <shared>false</shared>
                   <admin_state>true</admin_state>
                   <router_external>false</router_external>
                   <provider_physical_network>physnet_tenant</provider_physical_network>
                   <provider_network_type>vlan</provider_network_type>
                   <provider_segmentation_id>3112</provider_segmentation_id>
                  <subnet>
                      <name>test-subnet2</name>
                      <ipversion>ipv4</ipversion>
                      <dhcp>false</dhcp>
                      <address>40.10.11.22</address>
                      <netmask>255.255.255.0</netmask>
                      <gateway>40.10.11.1</gateway>
                      <dns nc:operation=”replace”>
                           <server>25.0.0.1</server>
                            <server>24.0.0.1</server>
                     </dns>
                   </subnet>
                </network>
             </networks>
</esc_datamodel>
```

4: Removing one or more DNS name server IP addresses from an existing subnet configuration

To delete any existing DNS Name Server entry, use nc:operation=”delete” as follows:

- a) dep.xml file Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<esc_datamodel xmlns="http://www.cisco.com/esc/esc"
xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
             <networks>
                <network>
                   <name>network-new-1</name>
                   <shared>false</shared>
                   <admin_state>true</admin_state>
                   <router_external>false</router_external>
                   <provider_physical_network>physnet_tenant</provider_physical_network>
                   <provider_network_type>vlan</provider_network_type>
                   <provider_segmentation_id>3112</provider_segmentation_id>
                   <subnet>
                      <name>subnet-new-1</name>
                      <ipversion>ipv4</ipversion>
                      <dhcp>false</dhcp>
                      <address>40.10.11.3</address>
                      <netmask>255.255.255.0</netmask>
                      <gateway>40.10.11.1</gateway>
                      <dns>
                           <server nc:operation=”delete”> 24.0.0.1</server>
                            <server nc:operation=”delete”>25.0.0.1</server>
                           <server> 26.0.0.1 </server>
                     </dns>
                   </subnet>
                </network>
             </networks>
</esc_datamodel>
```

Managing Flavors

A flavor defines sizes for RAM, disk, and number of cores.

When you deploy VNFs on OpenStack, you either have an option to use out-of-band flavors that are already available on OpenStack or create flavors in ESC. These flavors can be created using NETCONF or REST interface, or the ESC portal, and can be used for multiple deployments. For more information on deployment attributes see, Cisco Elastic Services Controller Deployment Attributes.

Note ESC Release 2.0 and later does not support creating or deleting flavor definitions on VMware vCenter.

Adding Flavors Using Northbound APIs

NETCONF request to create a flavor:

```
<?xml version="1.0" encoding="UTF-8"?>
<esc_datamodel xmlns:ns2="urn:ietf:params:xml:ns:netconf:notification:1.0"
xmlns:ns1="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:ns3="http://www.cisco.com/esc/esc_notifications"
xmlns:ns0="http://www.cisco.com/esc/esc" xmlns="http://www.cisco.com/esc/esc">
    <flavors>
        <flavor>
            <name>test-flavor-indep</name>
              <vcpus>1</vcpus>
              <memory_mb>512</memory_mb>
              <root_disk_mb>0</root_disk_mb>
              <ephemeral_disk_mb>0</ephemeral_disk_mb>
              <swap_disk_mb>0</swap_disk_mb>
        </flavor>
    </flavors>
</esc_datamodel>
```

NETCONF notification upon successful creation of a flavor:

```
<?xml version="1.0" encoding="UTF-8"?>
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2015-07-13T13:33:51.805+00:00</eventTime>
  <escEvent xmlns="http://www.cisco.com/esc/esc">
    <status>SUCCESS</status>
    <status_message>Flavor creation completed successfully.</status_message>
    <flavor>test-flavor-indep</flavor>
    <vm_source>
  </vm_source>
    <vm_target>
  </vm_target>
    <event>
      <type>CREATE_FLAVOR</type>
    </event>
  </escEvent>
</notification>
```

Note For more information about creating and deleting flavors using NETCONF API, see Cisco Elastic Services Controller API Guide. To access the REST API documentation directly from the ESC VM, see REST Northbound API, on page 12. For more information on adding and deleting flavors using the ESC portal, see Managing Resources Using ESC Portal, on page 367.

Managing Images

In ESC, an image is a bootable file system that can be used to launch VM instances.

When you deploy VNFs on OpenStack, you either have an option to use out-of-band images that are already available on OpenStack or create images in ESC. These images can be created using NETCONF or REST interface and can be used for multiple deployments.

An image can be made public or private on OpenStack. By default, the image is public. The visibility attribute is used to mark an image as public or private. A public image can only be created by an admin, whereas a private image does not require admin credentials.

Sample xml is as follows:

```
<images>
      <image>
        <name>mk-test-image</name>
        <src>file:///opt/cisco/esc/esc-confd/esc-cli/dumy.xml</src>
        <disk_format>qcow2</disk_format>
        <container_format>bare</container_format>
        <serial_console>true</serial_console>
        <disk_bus>virtio</disk_bus>
        <visibility>private</visibility>
      </image>
  </images>
```

Both out of band images, and images created by ESC can be public or private.

Adding Images Using Northbound APIs

NETCONF request to create an image:

```
<?xml version="1.0" encoding="UTF-8"?>
<esc_datamodel xmlns:ns2="urn:ietf:params:xml:ns:netconf:notification:1.0"
xmlns:ns1="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:ns3="http://www.cisco.com/esc/esc_notifications"
xmlns:ns0="http://www.cisco.com/esc/esc" xmlns="http://www.cisco.com/esc/esc">
    <images>
        <image>
            <name>example-cirrosimage-indep</name>

<src>http://172.16.0.0:/share/images/esc_automated_test_images/cirros-0.3.3-x86_64-disk.img</src>

              <disk_format>qcow2</disk_format>
              <container_format>bare</container_format>
              <serial_console>true</serial_console>
              <disk_bus>virtio</disk_bus>
        </image>
    </images>
</esc_datamodel>
```

NETCONF notification upon successful creation of an image:

```
<?xml version="1.0" encoding="UTF-8"?>
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2015-07-13T13:46:50.339+00:00</eventTime>
  <escEvent xmlns="http://www.cisco.com/esc/esc">
    <status>SUCCESS</status>
    <status_message>Image creation completed successfully.</status_message>
    <image>example-cirrosimage-indep</image>
    <vm_source>
  </vm_source>
    <vm_target>
  </vm_target>
    <event>
      <type>CREATE_IMAGE</type>
    </event>
  </escEvent>
</notification>
```

Note For more information about adding images using NETCONF API, see Cisco Elastic Services Controller API Guide. To access the REST API documentation directly from the ESC VM, see REST Northbound API, on page 12. For more information on adding and deleting images using the ESC portal, see Managing Resources Using ESC Portal, on page 367.

Managing Volumes

A volume is a storage device, similar to a block device in Nova. ESC supports both volumes created by ESC and out-of-band volumes. Further, ESC also supports bootable volumes created by ESC and out-of-band bootable volumes.

Note The maximum number of volumes that can be attached to a VM through the nova boot command is only two.

Volumes Created by ESC

To create volume as part of the VM group, the <size> and <sizeunits> parameters must be provided in the volumes section of the deployment request. The volume type is the default volume type in Cinder.

The following example shows how to create an ESC volume in the deployment request.

```
<volumes>
 <volume>
  <name>example</name>
  <volid>1</volid>
  <bus>ide</bus>
  <size>1</size>
  <sizeunit>GiB</sizeunit>
 </volume>
</volumes>
```

Note If a volume is added post-deployment, the Openstack API does not allow the supplied `bus` tag to be specified and uses the default defined in the OpenStack instance.

Bootable Volumes Created by ESC

A bootable volume is one which is used as a root disk. ESC creates bootable volumes using the image reference name or the UUID in the deployment request. To boot instances from the volume specify the boot_index , otherwise the instance will only be an attached volume.

For example,

```
<volumes>
    <volume>
        <name>cinder-vol1X</name>
        <volid>1</volid>
        <image>cirrosX1.75</image>
        <bus>ide</bus>
        <type>lvm</type>
        <size>1</size>
        <sizeunit>GiB</sizeunit>
        <boot_index>0</boot_index>
    </volume>
</volumes>
```

Out-of-band Volumes

The out-of-band (pre-existing) volume can be specified using the <type> attribute in the deployment request. If the <type> attribute is provided, ESC matches the volume with the type provided.

ESC differentiates an out-of-band volume and volume created by ESC based on the values set in the volumes section of deployment request. The volume (only if the volume is created by ESC) associated to a VM is deleted when a service is undeployed or the VM is scaled down.

Note The support for scale in/out when using out of band volumes is no longer available.

```
<volumes>
 <volume>
  <name>pre-existing</name>
  <volid>1</volid>
  <bus>ide</bus>
  <type>lvm</type>
 </volume>
</volumes>
```

If the <type> attribute is not provided, ESC matches a volume with no type.

ESC matches a volume with the same name. If more than one volume has the same name, ESC will fail the request.

```
<volumes>
 <volume>
  <name>pre-existing</name>
  <volid>1</volid>
  <bus>ide</bus>
 </volume>
</volumes>
```

Out-of-band Bootable Volumes

Out-of-band bootable volume (for OpenStack only) is a variation of out-of-band volume, where the specified volume is used as a root disk. The VM is booted from that volume, instead of the image. The <boot_index> attribute specifies the out-of-band bootable volumes in the deployment request.

For example,

```
<volumes>
 <volume>
  <name>pre-existing</name>
  <volid>0</volid>
  <bus>ide</bus>
  <type>lvm</type>
  <boot_index>0</boot_index>
 </volume>
</volumes>
```

The out of band bootable volume can be with or without <type> attribute, similar to out of band volumes.

Swapping Out-of-band Bootable Volume

To swap the out-of-band bootable volume, the update deployment request must remove the old volume and add new volume with the same volid and `boot_index` values. This will swap the bootable volume in OpenStack. Ensure that the VM must be rebooted following the update.

For example,

```
<volumes>
  <volume nc:operation="delete">
    <name>pre-existing</name>
    <volid>0</volid>
    <bus>ide</bus>
    <type>lvm</type>
    <boot_index>0</boot_index>
  </volume>
  <volume>
    <name>another-pre-existing</name>
    <volid>0</volid>
    <bus>ide</bus>
    <type>lvm</type>
    <boot_index>0</boot_index>
  </volume>
</volumes>
```

Parameter description

- Name—Specifies the display name of the pre-existing volume.
- Volid—Specifies the order in which volumes are attached. These are consecutive numbers starting from 0 or 1 for every VM group.
- Bus—Specifies the bus type of the volumes to be attached.
- Type—(Optional) If <type> is specified, then ESC matches the volume with the type provided.
- size and sizeunits—Defines a volume created by ESC
- boot_index—(Optional) specifies boot order. Set to 0 to boot from a given volume, similarly to how a VM would be booted from an image. The "bootable" property for that volume in OpenStack must be set to true for this to work.

Multi-attach Volumes

The ability to attach a volume to multiple hosts/servers simultaneously is a use case desired for active/active or active/standby scenarios (Openstack only). In order to attach a volume to multiple server instances you need to have the *multiattach* flag set to *True* in the volume details. Ensure that you have the right role and policy settings before performing the operation.

To create this special type that includes an extra-spec capability setting of `multiattach=<is> True`, use the following commands:

```
$ cinder type-create multiattach
$ cinder type-key multiattach set multiattach="<is> True"
```

The type-key name could be anything however, the property it references should be multiattach. This guide will reference the type as *multiattach*.

Once this type is created, create an out-of-band volume (bootable or otherwise) in Openstack by specifying the type. For example:

```
$ cinder create <volume_size> --name <volume_name> --volume-type multiattach
```

To use this volume, when creating a deployment, treat it the same as an out-of-band volume, except that you may specify the volume UUID or unique name against more than one VM. ESC only attempts to attach correctly typed volumes to more than one VM. For example:

```
<vm_group>
        <name>c1</name>
        ...
        <volumes>
         <volume>
            <name>cf-cdr0-volume</name>
            <volid>0</volid>
         </volume>
        </volumes>
        ...
  </vm_group>
  <vm_group>
       <name>c2</name>
       ...
       <volumes>
        <volume>
            <name>cf-cdr0-volume</name>
            <volid>0</volid>
       </volume>
      </volumes>
      ...
</vm_group>
```

The multiattach volume can be detached like a regular out-of-band volume and also be used to replace a regular out-of-band volume on a VM by using a service update. This action requires a reboot of the VM to recognize the newly attached volume (multiattach or otherwise).

Note **Openstack Requirements**

- The minimum required compute API micro-version for attaching a multiattach-capable volume to more than one server is 2.60.
- Cinder 12.0.0 (Queens) or latest is required (microversion ‘3.50’ or higher).
- The nova-compute service must be running at least Queens release level code (17.0.0) and the hypervisor driver must support attaching block storage devices to more than one guest. Refer to the feature support matrix for details on which compute drivers support volume multiattach.
- While using the libvirt compute driver, the following native package versions determine multiattach support:
- Libvirt must be greater than or equal to 3.10
- Qemu must be less than 2.10.
- Swapping an in-use multiattach volume is not supported (this is actually controlled via the block storage volume retype API).

Tenant-Volume API

The tenant-volume API allows you to create and delete volumes outside a deployment request. The tenant-volume API creates the volume directly under the tenant. You must provide the tenant details to create a volume.

A sample tenant-volume NETCONF API request is as follows:

```
<esc_datamodel xmlns="http://www.cisco.com/esc/esc">
    <tenants>
        <tenant>
            <name>admin</name>
            <volumes>
                <volume>
                    <name>some-volume</name>
                    <type>lvm</type>
                    <size>1</size>
                    <sizeunit>GiB</sizeunit>
                </volume>
            </volumes>
        </tenant>
    </tenants>
</esc_datamodel>
```

You can also use the tenant-volume API to create a volume using an existing tenant. For this, the volume name must be unique for that tenant.

Note

- The tenant-volume API is supported by both NETCONF and REST APIs.
- You cannot use the tenant-volume API to create or delete ephemeral or out-of-band volumes.
- The volumes that are managed by ESC only can be deleted.
- You cannot update an existing volume using the tenant-volume API.

Deploying with the Volumes Created by the Tenant-Volume API

ESC treats a volume created by the tenant-volume API as an out-of-band volume. To deploy a volume created by the tenant-volume API, you must provide the <size> and <sizeunit> parameters in the deployment data model. When the <size> and <sizeunit> parameters are not available, ESC looks for the volume created by the tenant-volume API. If this does not exist, then ESC looks for other out-of-band volumes created by other ESCs or other users. If out-of-band volumes are not available, then the deployment request is rejected.

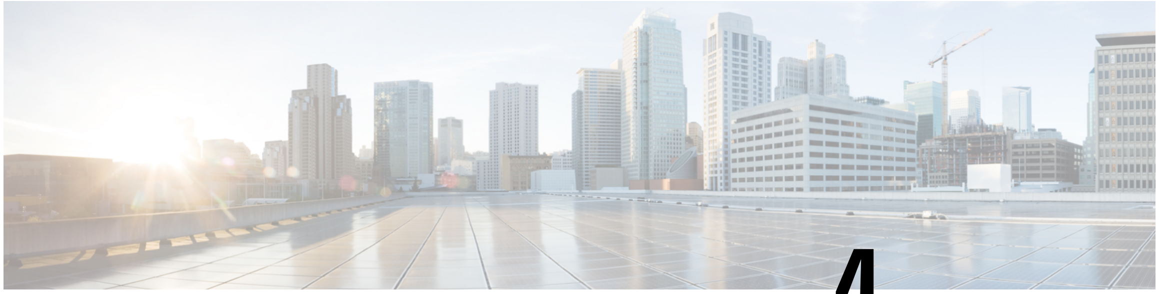
A sample deployment request with a volume created using the tenant-volume API is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<esc_datamodel xmlns:ns2="urn:ietf:params:xml:ns:netconf:notification:1.0"
xmlns:ns1="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:ns3="http://www.cisco.com/esc/esc_notifications"
xmlns:ns0="http://www.cisco.com/esc/esc" xmlns="http://www.cisco.com/esc/esc">
    <tenants>
        <tenant>
            <name>admin</name>
            <deployments>
                <deployment>
                    <name>admin-with-volume</name>
                    <vm_group>
                        <name>cirros</name>
                        <bootup_time>60</bootup_time>
                        <recovery_wait_time>0</recovery_wait_time>
                        <image>Automation-Cirros-Image</image>
                        <flavor>Automation-Cirros-Flavor</flavor>
                        <volumes>
                            <volume>
                                <name>some-volume</name>
                                <volid>1</volid>
                                <bus>ide</bus>
                            </volume>
                        </volumes>
                        <interfaces>
                            <interface>
                                <nicid>0</nicid>
                                <network>mynetwork</network>
                            </interface>
                        </interfaces>
                        <scaling>
                            <min_active>1</min_active>
                            <max_active>1</max_active>
                            <elastic>true</elastic>
                        </scaling>
                        <kpi_data>
                            <kpi>
                                <event_name>VM_ALIVE</event_name>
                                <metric_value>1</metric_value>
                                <metric_cond>GT</metric_cond>
```

```
                                    <metric_type>UINT32</metric_type>
                                    <metric_collector>
                                        <type>ICMPPing</type>
                                        <nicid>0</nicid>
                                        <poll_frequency>3</poll_frequency>
                                        <polling_unit>seconds</polling_unit>
                                        <continuous_alarm>false</continuous_alarm>
                                    </metric_collector>
                                </kpi>
                            </kpi_data>
                            <rules>
                                <admin_rules>
                                    <rule>
                                        <event_name>VM_ALIVE</event_name>
                                        <action>"ALWAYS log"</action>
                                        <action>"TRUE
                                          servicebooted.sh"</action>
                                        <action>"FALSE recover
                                          autohealing"</action>
                                    </rule>
                                </admin_rules>
                            </rules>
                            <config_data />
                        </vm_group>
                    </deployment>
                </deployments>
            </tenant>
        </tenants>
    </esc_datamodel>
```

If you provide the <size> and <sizeunit> parameters of a volume, then ESC creates a new volume using these values as part of the deployment. The new volume is treated as an ephemeral volume.

Note For ephemeral volumes, the minimum and maximum scaling value can be more than 1, but for tenants and out-of-band volumes the value can be 1 only.



CHAPTER 4

Managing Resources on VMware vCenter

This section contains the following topics:

Adding Images on VMware vCenter

When you deploy VNFs on VMware vCenter, you can either use the out-of-band images that are already available on VMware vCenter or create an image in the ESC portal, or using REST or NETCONF APIs. For more information on deployment attributes see, Cisco Elastic Services Controller Deployment Attributes.

Adding Images Using Northbound APIs

Note When you deploy VNFs on VMware vCenter, you can either use the out-of-band images that are already available on VMware vCenter or create an image in the ESC portal or using REST or NETCONF APIs.

NETCONF request to create an image:

```
<?xml version="1.0" encoding="UTF-8"?>
<esc_datamodel xmlns:ns2="urn:ietf:params:xml:ns:netconf:notification:1.0"
xmlns:ns1="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:ns3="http://www.cisco.com/esc/esc_notifications"
xmlns:ns0="http://www.cisco.com/esc/esc" xmlns="http://www.cisco.com/esc/esc">
    <images>
        <image>
            <name>cirrosimage-indep</name>

<src>http://172.16.0.0:/share/images/esc_automated_test_images/cirros-0.3.3-x86_64-disk.img</src>

              <disk_format>qcow2</disk_format>
              <container_format>bare</container_format>
              <serial_console>true</serial_console>
              <disk_bus>virtio</disk_bus>
        </image>
    </images>
</esc_datamodel>
```

NETCONF notification upon successful creation of an image:

```
<?xml version="1.0" encoding="UTF-8"?>
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
```

```
    <eventTime>2015-07-13T13:46:50.339+00:00</eventTime>
    <escEvent xmlns="http://www.cisco.com/esc/esc">
      <status>SUCCESS</status>
      <status_message>Image creation completed successfully.</status_message>
      <image>cirrosimage-indep</image>
      <vm_source>
    </vm_source>
      <vm_target>
    </vm_target>
      <event>
        <type>CREATE_IMAGE</type>
      </event>
    </escEvent>
  </notification>
```

Note For more information about adding images using NETCONF API, see Cisco Elastic Services Controller API Guide. To access the REST API documentation directly from the ESC VM, see REST Northbound API, on page 12. For more information on adding and deleting images using the ESC portal, see Managing VMware vCenter Resources Using ESC portal, on page 370.

Creating Distributed Port on VMware vCenter

On VMware vCenter, you configure a distributed port on a vSphere distributed switch that connects to the VM kernel or to a virtual machine's network adapter. It specifies port configuration options for each member port on a vSphere distributed switch. Distributed port groups define how a connection is made to a network. You can use REST interface to create distributed port groups.

The following example shows how to create a distributed port group (VMware vCenter only) using REST API:

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <name>network-portgroup-01</name>

    <switch_name>vdSwitch-01</switch_name>

    <vlan_id>0</vlan_id>

    <number_of_ports>8</number_of_ports>
</network>
```

Note On VMware vCenter, ESC only supports basic portGroup or network creation within a vSphere Distributed Switch (VDS). For advance vDS configuration, only out-of-band configuration is supported by ESC.

CHAPTER 5

Managing Resources on vCloud Director

Managing Resources on vCloud Director (vCD)

All vCD resources such as template, catalog, network and so on are managed by out of band (OOB). For information on deploying VMs on vCD, see Deploying Virtual Network Functions on VMware vCloud Director (vCD), on page 140.

Organizations

An organization is a group of users, groups, and computing resources. It contains the vApp templates that the organization creates, and the resources used to create the vApps. A cloud can contain one or more organizations.

Organization VDC

An organization virtual datacenter (organization VDC) is a deployment environment for virtual systems. It must be created before the deployment. It contains an organization, and an allocation mechanism for resources such as networks, storage, CPU, and memory. It must have enough memory and CPU capacity and storage spaces (storage profile).

Catalogs

Catalogs contain references to vApp templates and media images. The catalog where the vApp template is located must have read and write permissions for the organization user used for deployment. The write privilege is required if ESC needs to construct or upload an ISO file for day 0 configuration.

Network

For vApp, there are two levels of network.

- Network within vApp for communication among VMs within the vApp.
- Network within vDC for communication among VMs across vApp.

ESC is deployed to vCenter and is not part of the vCD. For ESC to monitor the VM status, each VM must have at least one network interface that connects to an Org VDC network or a vApp network which is connected to an external network directly or indirectly.

Deployment Storage Profile

The storage profile is specified in the deployment request.

Note The deployment storage profile is a way to specify the datastore from underneath VMware vSpere. It is different from the volume or disks of VM.

Example:

```
<volumes>
    <volume>
        <name>{Storage profile name}</name>
        <volid>1</volid>
    </volume>
</volumes>
```

CHAPTER 6

Managing ESC Resources

Managing VIM Connectors

A VIM connector contains details such as URL and authentication credentials, which enables ESC to connect and communicate with the VIM. ESC connects to more than one VIM if the VIM connectors are configured. You can configure the VIM connector and its credentials in two ways:

- At the time of installation using the bootvm.py parameters—Only a single VIM connector can be configured using bootvm.py, which becomes the default VIM connector.
- Using the VIM Connector APIs—The VIM connector API allows you to add multiple VIM connectors. You can configure a default VIM connector (if it is not already configured using the bootvm.py parameters), and additional VIM connectors.

The default VIM connector connects ESC to the default VIM. Each VIM in a multi VIM deployment is configured with a VIM connector. These VIMs are non-default VIMs. ESC creates and manages resources on a default VIM. Only deployments are supported on a non-default VIM.

For a single VIM deployment, a single configured VIM connector becomes the default VIM connector. For a multiple VIM deployment, you need to add multiple connectors, and specify one connector as default using the default VIM connector API. For more information, see Deploying VNFs on Multiple OpenStack VIMs, on page 111.

Note ESC accepts the northbound configuration request to create, update, or delete a resource, or a deployment only if the following conditions are met:

- ESC has the target VIM/VIMs and corresponding VIM user configured.
- ESC is able to reach the target VIM/VIMs.
- ESC is able to authenticate the VIM user.

Configuring the VIM Connector

You can configure the VIM Connector during or after installation.

Configuring the VIM Connector During Installation

To configure the VIM Connector during installation, the following parameter must be provided to bootvm.py:

Environment variables	bootvm.py arguments
OS_TENANT_NAME	--os_tenant_name
OS_USERNAME	--os_username
OS_PASSWORD	--os_password
OS_AUTH_URL	--os_auth_url

Configuring the VIM Connector After Installation

To configure the VIM Connector after installation, the following parameter must be provided to bootvm.py:

```
--no_vim_credentials
```

When the no_vim_credentials parameter is provided, the following bootvm.py arguments are ignored:

- os_tenant_name
- os_username
- os_password
- os_auth_url

For details on Installation, see the Cisco Elastic Services Controller Install and Upgrade Guide. You can configure the same using the VIM Connector APIs post installation, for more details, see Managing VIM Connector Using the VIM Connector APIs, on page 51.

Default VIM Connector

The default VIM connector API allows you to specify a default VIM connector when multiple connectors are available in a deployment.

For a Single VIM deployment, ESC supports a single VIM connector. This single VIM connector becomes the default VIM connector. ESC supports multiple VIM connectors for multi VIM deployments. You can configure the default VIM connector using the new locator attribute. If you are using the ESC Release 2.x datamodel for deployments and creating resources, then configure the default VIM connector explicitly in ESC.

The locator attribute is introduced in the data model for deploying VMs on non-default VIMs. For more details, see Deploying VNFs on Multiple OpenStack VIMs, on page 111.

While deploying, if the VIM connectors are available, but the default connector is not yet configured, then it is mandatory that you specify the locator attribute else the request is rejected.

The data model prior to ESC Release 3.0 cannot be used if the default VIM connector is not configured. While upgrading from ESC Release 2.x to ESC Release 3.0 and later, the existing VIM connector is provisioned as the default VIM connector.

Note You cannot change or delete the default VIM connector to a different one once configured.

You must specify the default connector at the top level (or beginning) of the data model. The data model is as follows:

```
<esc_system_config>
    <vim_connectors>
        <default_vim_connector>vim1</default_vim_connector>
        <vim_connector>
            <id>vim1</id>
...
</vim_connector>
        <vim_connector>
            <id>vim2</id>
...
</vim_connector>
    </vim_connectors>
</esc_system_config>
```

To add the default VIM connector using the REST API,

```
<?xml version="1.0"?>
<default_vim_connector xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <defaultVimConnectorId>tb3_v3</defaultVimConnectorId>
</default_vim_connector>
```

To add a VIM connector at the time of installation, see Configuring the VIM Connector During Installation in Configuring the VIM Connector, on page 50. The VIM connectors allow multiple VIMs to connect to ESC. For more details on multi VIM deployment, see Deploying VNFs on Multiple OpenStack VIMs, on page 111.

Deleting VIM Connector

ESC creates SystemAdminTenant automatically when the default VIM connector is created and configured. The SystemAdminTenant cannot be deleted. The VIM is connected and the VIM user is authenticated to the system admin tenant. Hence, the default VIM cannot be deleted or updated. However, the VIM user and its properties can be deleted or updated. You can update and delete the non-default VIM connectors if there are no resources created on the VIM from ESC. If there are resources created on the VIM through ESC, then you must first delete the resources, and then the VIM user to delete the VIM connector.

Managing VIM Connector Using the VIM Connector APIs

If ESC was deployed without passing VIM credentials, you can set the VIM credentials through ESC using the VIM connector and VIM User APIs (REST or Netconf API). Even if the default VIM connector is configured during installation, the additional VIM connectors can be configured using the VIM connector APIs.

Managing using Netconf API

- Passing VIM credential using Netconf:

```
<esc_system_config xmlns="http://www.cisco.com/esc/esc">
  <vim_connectors>
    <!--represents a vim-->
    <vim_connector>
      <!--unique id for each vim-->
      <id>my-server</id>
      <!--vim type [OPENSTACK|VMWARE_VSPHERE|LIBVIRT|AWS|CSP]-->
      <type>OPENSTACK</type>
      <properties>
        <property>
          <name>os_auth_url</name>
          <value>http://{os_ip:port}/v3</value>
        </property>
        <!-- The project name for openstack authentication and authorization -->
        <property>
          <name>os_project_name</name>
          <value>vimProject</value>
        </property>
        <!-- The project domain name is only needed for openstack v3 identity api -->
        <property>
          <name>os_project_domain_name</name>
          <value>default</value>
        </property>
        <property>
          <name>os_identity_api_version</name>
          <value>3</value>
        </property>
      </properties>
      <users>
        <user>
          <id>admin</id>
          <credentials>
            <properties>
              <property>
                <name>os_password</name>
                <value>********</value>
              </property>
              <!-- The user domain name is only needed for openstack v3 identity api
-->
              <property>
                <name>os_user_domain_name</name>
                <value>default</value>
              </property>
            </properties>
          </credentials>
        </user>
      </users>
    </vim_connector>
  </vim_connectors>
</esc_system_config>
```

- Updating VIM Connector using Netconf:

```
<esc_system_config xmlns="http://www.cisco.com/esc/esc">
  <vim_connectors>
    <vim_connector nc:operation="replace">
      <id>example_vim</id>
```

```
      <type>OPENSTACK</type>
      <properties>
        <property>
          <name>os_auth_url</name>
          <value>{auth_url}</value>
        </property>
        <property>
          <name>os_project_name</name>
          <value>vimProject</value>
        </property>
        <!-- The project domain name is only needed for openstack v3 identity api -->
        <property>
          <name>os_project_domain_name</name>
          <value>default</value>
        </property>
        <property>
          <name>os_identity_api_version</name>
          <value>3</value>
        </property>
      </properties>
    </vim_connector>
  </vim_connectors>
</esc_system_config>
```

- Updating VIM user using Netconf:

```
<esc_system_config xmlns="http://www.cisco.com/esc/esc">
  <vim_connectors>
    <vim_connector>
      <id>example_vim</id>
        <users>
          <user nc:operation="replace">
            <id>my_user</id>
            <credentials>
             <properties>
              <property>
                <name>os_password</name>
                <value>********</value>
              </property>
              <!-- The user domain name is only needed for openstack v3 identity api
-->
              <property>
                <name>os_user_domain_name</name>
                <value>default</value>
              </property>
             </properties>
           </credentials>
          </user>
        </users>
    </vim_connector>
  </vim_connectors>
</esc_system_config>
```

- Deleting VIM connector using Netconf:

```
<esc_system_config xmlns="http://www.cisco.com/esc/esc">  <vim_connectors>
    <vim_connector nc:operation="delete">
      <id>example_vim</id>
    </vim_connector>
  </vim_connectors>
</esc_system_config>
```

- Deleting VIM User using Netconf:

```
<esc_system_config xmlns="http://www.cisco.com/esc/esc">
  <vim_connectors>
    <vim_connector>
      <id>example_vim</id>
        <users>
          <user nc:operation="delete">
            <id>my_user</id>
          </user>
        </users>
    </vim_connector>
  </vim_connectors>
</esc_system_config>
```

- Deleting VIM Connector using command:

```
$/opt/cisco/esc/esc-confd/esc-cli/esc_nc_cli --user <username> --password <password>
delete-vim-connector <vim connector id>
```

- Deleting VIM user using command:

```
$/opt/cisco/esc/esc-confd/esc-cli/esc_nc_cli --user <username> --password <password>
delete-vim-user <vim connector id> <vim user id>
```

Managing using REST API

- Adding VIM using REST:

```
POST /ESCManager/v0/vims/
HEADER: content-type, callback

<?xml version="1.0"?>
<vim_connector xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <id>example_vim</id>
  <type>OPENSTACK</type>
  <properties>
    <property>
      <name>os_auth_url</name>
      <value>{auth_url}</value>
    </property>
    <property>
      <name>os_project_name</name>
      <value>vimProject</value>
    </property>
    <!-- The project domain name is only needed for openstack v3 identity api -->
    <property>
      <name>os_project_domain_name</name>
      <value>default</value>
    </property>
    <property>
      <name>os_identity_api_version</name>
      <value>3</value>
    </property>
  </properties>
</vim_connector>
```

- Adding VIM user using REST:

```
POST /ESCManager/v0/vims/{vim_id}/vim_users
HEADER: content-type, callback

<?xml version="1.0"?>
```

```
<user xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <id>my_user</id>
  <credentials>
    <properties>
      <property>
        <name>os_password</name>
        <value>********</value>
      </property>
      <!-- The user domain name is only needed for openstack v3 identity api -->
      <property>
        <name>os_user_domain_name</name>
        <value>default</value>
      </property>
    </properties>
  </credentials>
</user>
```

- Updating VIM using REST:

```
PUT /ESCManager/v0/vims/{vim_id}
HEADER: content-type, callback

<?xml version="1.0"?>
<vim_connector xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <!--unique id for each vim-->
  <id>example_vim</id>
  <type>OPENSTACK</type>
  <properties>
    <property>
      <name>os_auth_url</name>
      <value>{auth_url}</value>
    </property>
    <property>
      <name>os_project_name</name>
      <value>vimProject</value>
    </property>
    <!-- The project domain name is only needed for openstack v3 identity api -->
    <property>
      <name>os_project_domain_name</name>
      <value>default</value>
    </property>
    <property>
      <name>os_identity_api_version</name>
      <value>3</value>
    </property>
 </properties>
</vim_connector>
```

- Updating VIM user using REST:

```
PUT /ESCManager/v0/vims/{vim_id}/vim_users/{vim_user_id}
HEADER: content-type, callback

<?xml version="1.0"?>
<user xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <id>my_user</id>
  <credentials>
    <properties>
      <property>
        <name>os_password</name>
        <value>********</value>
      </property>
      <!-- The user domain name is only needed for openstack v3 identity api -->
```

```
      <property>
        <name>os_user_domain_name</name>
        <value>default</value>
      </property>
    </properties>
  </credentials>
</user>
```

- Deleting VIM using REST:

  ```
  DELETE /ESCManager/v0/vims/{vim_id}
  ```

- Deleting VIM user using REST:

  ```
  DELETE /ESCManager/v0/vims/{vim_id}/vim_users/{vim_user_id}
  ```

- Notification example after each VIM or VIM user configuration is done:

  ```
  <?xml version="1.0" encoding="UTF-8"?>
  <notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
    <eventTime>2016-10-06T16:24:05.856+00:00</eventTime>
    <escEvent xmlns="http://www.cisco.com/esc/esc">
      <status>SUCCESS</status>
      <status_code>200</status_code>
      <status_message>Created vim connector successfully</status_message>
      <vim_connector_id>my-server</vim_connector_id>
      <event>
        <type>CREATE_VIM_CONNECTOR</type>
      </event>
    </escEvent>
  </notification>
  ```

  For more information on the APIs, see Cisco Elastic Services Controller API Guides.

  Important Notes:

  - You can add more than one VIM connector, but all the VIM connectors must have the same VIM type. Multiple VIM connectors can be added for OpenStack VIM only. However, only one VIM user can be configured per VIM connector.
  - os_project_name and os_project_domain_name properties specify the OpenStack project details for authentication and authorization under the VIM connector properties. If the os_tenant_name property exists under the Vim User, it will be ignored.
  - The VIM connector properties os_auth_url and os_project_name and VIM User property os_password are mandatory properties for the OpenStack VIM. If these properties are not provided, then the request to create the VIM connector is rejected.
  - VIM username and password can be updated anytime. VIM endpoint cannot be updated while resources created through ESC exist.
  - The name of a VIM property or VIM user credentials property are not case sensitive, e.g. OS_AUTH_URL and os_auth_url is the same to ESC.

You can encrypt the VIM connector credentials by replacing the existing <value> field with <encrypted_value>.

For example,

```
<credentials>
  <properties>
    <property>
```

```
      <name>os_password</name>
      <encrypted_value>********</encrypted_value>
    </property>
    <property>
      <name>os_user_domain_name</name>
      <value>default</value>
    </property>
  </properties>
</credentials>
```

This stores the os_value password as an aes-cfb-128-encrypted-string in the CFB using the keys contained in /opt/cisco/esc/esc_database/esc_production_confd.conf.

Note The existing value must be replaced with encrypted value only within the credentials specified.

For more information, see Encrypting Configuration Data.

VIM Connector Status API

The table below shows the VIM connector status and a status message for each VIM connector. The status shows ESC connection and authentication status of the VIM.

VIM Reachability	User Authentication	Status (by ESC)	Status Message
NOT REACHABLE	-	CONNECTION_FAILED	Unable to establish VIM connection
REACHABLE	VIM user is not configured	NO_CREDENTIALS	No VIM user credentials found
REACHABLE	Authentication failed	AUTHENTICATION_FAILED	VIM authentication failed
REACHABLE	Authentication successful	CONNECTION_SUCCESSFUL	Successfully connected to VIM

Status using the REST API

HTTP Operation: GET

Path: `ESCManager/v0/vims, ESCManager/v0/vims/<specific_vim_id>`

Sample REST Response is as follows:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<vim_connector xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <properties>
        <property>
            <name>os_auth_url</name>
            <value>http://172.16.0.0:5000/v2.0/</value>
        </property>
    </properties>
    <id>default_openstack_vim</id>
    <status>CONNECTION_SUCCESSFUL</status>
    <status_message>Successfully connected to VIM</status_message>
    <type>OPENSTACK</type>
</vim_connector>
```

Status using the NETCONF API

The opdata shows the status. The VIM connector status is within the vim connector container.

Sample opdata is as follows:

```
<system_config>
    <active_vim>OPENSTACK</active_vim>
    <openstack_config>
        <os_auth_url>http://172.16.0.0:5000/v2.0/</os_auth_url>
        <admin_role>admin</admin_role>
        <os_tenant_name>admin</os_tenant_name>
        <os_username>admin</os_username>
        <member_role>_member_</member_role>
    </openstack_config>
    <vim_connectors>
        <vim_connector>
            <id>my-server</id>
            <status>CONNECTION_FAILED</status>
            <status_message>Unable to establish VIM connection</status_message>
        </vim_connector>
        <vim_connector>
            <id>Openstack-Liberty</id>
            <status>NO_CREDENTIALS</status>
            <status_message>No VIM user credentials found</status_message>
        </vim_connector>
    </vim_connectors>
</system_config>
```

VIM Connector Operation Status

The VIM_CONNECTION_STATE notification notifies the status of each VIM connector and user added to ESC through REST and NETCONF. For more details about the VIM connectors, see Managing VIM Connectors, on page 49.

The notification shows:

- Event Type: VIM_CONNECTION_STATE
- Status: Success or Failure
- Status message
- vim_connector_id

Notifications are sent for monitoring the VIM connector, adding or deleting the VIM user, and updating the VIM connector. The success and failure notification examples are as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2017-06-27T14:50:40.823+00:00</eventTime>
  <escEvent xmlns="http://www.cisco.com/esc/esc">
    <status>FAILURE</status>
    <status_code>0</status_code>
    <status_message>VIM Connection State Down</status_message>
    <vim_connector_id>my-server</vim_connector_id>
    <event>
      <type>VIM_CONNECTION_STATE</type>
    </event>
  </escEvent>
</notification>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
```

```
  <eventTime>2017-06-27T14:51:55.862+00:00</eventTime>
  <escEvent xmlns="http://www.cisco.com/esc/esc">
    <status>SUCCESS</status>
    <status_code>0</status_code>
    <status_message>VIM Connection State Up</status_message>
    <vim_connector_id>my-server</vim_connector_id>
    <event>
      <type>VIM_CONNECTION_STATE</type>
    </event>
  </escEvent>
</notification>
```

CHAPTER 7

VIM Connector Configurations

VIM Connector Configurations for OpenStack

You can configure the VIM connector for OpenStack specific operations.

Note To configure a VIM connector, see Configuring the VIM Connector, on page 50 .

Creating Non-admin Roles for ESC Users in OpenStack

By default, OpenStack assigns an admin role to the ESC user. Some policies may restrict using the default admin role for certain ESC operations. Starting from ESC Release 3.1, you can create non-admin roles with limited permissions for ESC users in OpenStack.

To create a non-admin role,

1. Create a non-admin role in OpenStack.
2. Assign the non-admin role to the ESC user.

   You must assign ESC user roles in OpenStack Horizon (Identity) or using the OpenStack command line interface. For more details see, OpenStack Documentation.

   The role name can be customized in OpenStack. By default, all non-admin roles in OpenStack have the same level of permissions.
3. Grant the required permissions to the non-admin role.

   You must modify the `policy.json` file to provide the necessary permissions.

Note You must grant permissions to the *create_port: fixed_ips* and *create_port: mac_address* parameters in the policy.json file for ESC user role to be operational.

The table below lists the ESC operations that can be performed by the non-admin role after receiving the necessary permissions.

Table 3: Non-admin role permissions for ESC operations

ESC VIM Operation	Description	Permission	Note
Create Project	To create an OpenStack project	`/etc/keystone/policy.json "identity:create_project" "identity:create_grant"`	For ESC managed OpenStack project, adding the user to the project with a role requires *identity:create_grant*.
Delete Project	To delete an OpenStack project	`/etc/keystone/policy.json "identity:delete_project"`	
Query Image	To get a list of all images	Not required	The owner (a user in the target project) can query. You can retrieve public or shared images.
Create Image	To create a public image	`/etc/glance/policy.json "publicize_image"`	By default an admin can create a public image. Publicizing an image is protected by the policy.
	To create a private image	Not required	You can use the following to create a private image `<image> <name>mk-test-image</name> ... <disk_bus>virtio</disk_bus> <visibility>private</visibility> </image>`
Delete Image	To delete an image	Not required	The owner can delete the image.
Query Flavor	To query a pre-existing flavor	Not required	The owner can query a flavor. You can query public flavors as well.
Create Flavor	To create a new flavor	`/etc/nova/policy.json "os_compute_api:os-flavor-manage"`	Managing a flavor is typically only available to administrators of a cloud.

ESC VIM Operation	Description	Permission	Note
Delete Flavor	To delete a flavor	`/etc/nova/policy.json` `"os_compute_api:os-flavor-manage"`	
Query Network	To get a list of networks	`/etc/neutron/policy.json` `"get_network"`	Owner can get the list of networks including shared networks.
Create Network	To create a normal network	Not required	

	To create network with special cases	`/etc/neutron/policy.json`
		`"create_network:provider:physical_network"`
		`"create_network:provider:network_type"`
		`"create_network:provider:segmentation_id"`
		`"create_network:shared"`
			You need these rules when you are creating network
			with physical_network (e.g., SR-IOV), or network_type (e.g., SR-IOV), or segmentation_id (e.g., 3008), or set the network for sharing.
			`<network> <name>provider-network</name> <!-- <shared>false</shared> //default is true --> <admin_state>true</admin_state> <provider_physical_network>VAR_PHYSICAL_NET </provider_physical_network> <provider_network_type>vlan </provider_network_type> <provider_segmentation_id>2330 </provider_segmentation_id> ... </network>`

Delete Network	To delete a network	Not required	The owner can delete the network.

ESC VIM Operation	Description	Permission	Note
Query Subnet	To get a list of subnets	`/etc/neutron/policy.json "get_subnet"`	The network owner can get a list of the subnets. You can get a list of subnets from a shared network as well. `<network> <name>esc-created-network</name> <!--network must be created by ESC--> <admin_state>false</admin_state> <subnet> <name>makulandyescextnet1-subnet1</name> <ipversion>ipv4</ipversion> <dhcp>true</dhcp> <address>10.6.0.0</address> <netmask>255.255.0.0</netmask> </subnet> </network>`
Create Subnet	To create a subnet	Not required	The network owner can create a subnet.
Delete Subnet	To delete a subnet	Not required	The network owner can delete a subnet.
Query Port	Get a pre-existing port	Not required	The owner can get a list of ports.
Create Port	To create a network interface with DHCP	Not required	
	Create a network interface with a mac address	`/etc/neutron/policy.json "create_port:mac_address"`	`<interfaces> <interface> <nicid>0</nicid> <mac_address>fa:16:3e:73:19:b5</mac_address> <network>esc-net</network> </interface> </interfaces>`VM recovery also requires this privilege.
	To create a network interface with a fixed IP or shared ips	`/etc/neutron/policy.json "create_port:fixed_ips"`	`<subnet> <name>IP-pool-subnet</name> <ipversion>ipv4</ipversion> <dhcp>false</dhcp> <address>172.16.0.0</address> <netmask>255.255.255.0</netmask> <gateway>172.16.0.1</gateway> </subnet><shared_ip> <nicid>0</nicid> <static>false</static> </shared_ip>` VM recovery also requires this privilege.

ESC VIM Operation	Description	Permission	Note
Update Port	Update port device owner	Not required	The owner can update the port.
	Update port to allow address pairs	`/etc/neutron/policy.json` `"update_port:allowed_address_pairs"`	`<interface> <nicid>0</nicid> <network>VAR_MANAGEMENT_NETWORK_ID</network> <allowed_address_pairs> <network> <name>VAR_MANAGEMENT_NETWORK_ID</name> </network> <address> <ip_address>172.16.0.0</ip_address> <netmask>255.255.0.0</netmask> </address> <address> <ip_address>172.16.6.1</ip_address> <ip_prefix>24</ip_prefix> </address> </allowed_address_pairs> </interface>`
Delete Port	To delete a port	Not required	The owner can delete the port.
Query Volume	To get a list of volumes	Not required	The owner can get the list of volumes.
Create Volume	To create a volume	Not required	
Delete Volume	To delete a volume	Not required	The owner can delete the volume.
Query VM	To get all the VMs in a project	Not required	The owner can get the list of all the VMs in a project.

ESC VIM Operation	Description	Permission	Note
Create VM	To create a VM	Not required	
	To create a VM in a host targeted deployment	`/etc/nova/policy.json` "os_compute_api:servers:create:forced_host"	`<placement> <type>zone_host</type> <enforcement>strict</enforcement> <host>anyHOST</host> </placement>`
	To create VMs in a zone targeted deployment	Not required	
	To create VMs in the same Host Affinity/Anti-affinity	Not required	
	To create VMs in a servergroup Affinity/Anti-affinity	Not required	This support is for intragroup anti-affinity only.
Delete VM	To delete a VM	Not required	The owner can delete the VM.

For more details on managing resources on OpenStack, see Managing Resources on OpenStack, on page 17.

Overwriting OpenStack Endpoints

By default, ESC uses endpoints catalog return option provided by OpenStack after a successful authentication. ESC uses these endpoints to communicate with different APIs in OpenStack. Sometimes the endpoints are not configured correctly, for example, the OpenStack instance is configured to use KeyStone V3 for authentication, but the endpoint returned from OpenStack is for KeyStone V2. You can overcome this by overwriting the OpenStack endpoints.

You can overwrite (configure) the OpenStack endpoints while configuring the VIM connector. This can be done at the time of installation using the bootvm.py parameters, and using the VIM connector APIs.

The following OpenStack endpoints can be configured using the VIM connector configuration:

- OS_IDENTITY_OVERWRITE_ENDPOINT
- OS_COMPUTE_OVERWRITE_ENDPOINT
- OS_NETWORK_OVERWRITE_ENDPOINT

- OS_IMAGE_OVERWRITE_ENDPOINT
- OS_VOLUME_OVERWRITE_ENDPOINT

To overwrite OpenStack endpoints at the time of installation, a user can create an esc configuration parameters file, and pass the file as an argument to bootvm.py while deploying an ESC VM.

Below is an example of the param.conf file:

```
openstack.os_identity_overwrite_endpoint=http://www.xxxxxxxxxxx.com
```

For more information on configuring the VIM connector at the time of Installation, see Configuring the VIM Connector, on page 50.

To overwrite (configure) the OpenStack endpoints for a non-default VIM connector using the VIM connector APIs (both REST and NETCONF), add the overwriting endpoints as the VIM connector properties either while creating a new VIM connector or updating an existing one.

Each VIM connector can have its own overwriting endpoints. There is no default overwriting endpoint.

In the example below, *os_identity_overwrite_endpoint* and *os_network_overwrite_endpoint* properties are added to overwrite the endpoints.

```
<esc_system_config xmlns="http://www.cisco.com/esc/esc">
  <vim_connectors>
    <!--represents a vim-->
    <vim_connector>
      <id>default_openstack_vim</id>
      <type>OPENSTACK</type>
      <properties>
        <property>
          <name>os_auth_url</name>
          <value>http://172.16.0.0:35357/v3</value>
        </property>
        <property>
          <name>os_project_domain_name</name>
          <value>default</value>
        </property>
        <property>
          <name>os_project_name</name>
          <value>admin</value>
        </property>
        <property>
          <name>os_identity_overwrite_endpoint</name>
          <value>http://some_server:some_port/</value>
        </property>
        <property>
          <name>os_network_overwrite_endpoint</name>
          <value>http://some_other_server:some_other_port/</value>
        </property>
      </properties>
    </vim_connector>
  </vim_connectors>
</esc_system_config>
```

VIM Connector Configurations for AWS

You can set the VIM credentials for an AWS deployment using the VIM connector and VIM User API.

Note AWS deployment does not support default VIM connector.

The VIM connector **aws_default_region** value provides authentication, and updates the VIM status. The default region cannot be changed after authentication.

Configuring the VIM Connector

To configure the VIM connector for AWS deployment, provide the AWS_ACCESS_ID, AWS_SECRET_KEY from your AWS credentials.

```
[admin@localhost ~]# esc_nc_cli --user <username> --password <password> edit-config
aws-vim-connector-example.xml
```

Note To edit the existing VIM connector configuration, use the same command after making the necessary changes.

The AWS VIM connector example is as follows:

```
<esc_system_config xmlns="http://www.cisco.com/esc/esc">
   <vim_connectors>
      <vim_connector>
         <id>AWS_EAST_2</id>
         <type>AWS_EC2</type>
         <properties>
            <property>
               <name>aws_default_region</name>
               <value>us-east-2</value>
            </property>
         </properties>
         <users>
            <user>
               <id>AWS_ACCESS_ID</id>
               <credentials>
                  <properties>
                     <property>
                        <name>aws_secret_key</name>
                        <encrypted_value>AWS_SECRET_KEY</encrypted_value>
                     </property>
                  </properties>
               </credentials>
            </user>
         </users>
      </vim_connector>
   </vim_connectors>
</esc_system_config>
```

Deleting VIM Connector

To delete the existing VIM connector, you must first delete the deployment, the VIM user, and then the VIM connector.

```
[admin@localhost ~]# esc_nc_cli --user <username> --password <password> delete-vimuser
AWS_EAST_2 AWS_ACCESS_ID
```

```
[admin@localhost ~]# esc_nc_cli --user <username> --password <password> delete-vimconnector
AWS_EAST_2
```

Note You can configure multiple VIM connectors, but for the same VIM type.

The VIM connectors for AWS deployment must be configured using the VIM connector API.

ESC supports one VIM user per VIM connector.

The VIM connector and its properties cannot be updated after deployment.

For information on deploying VNFs on AWS, seeDeploying VNFs on a Single or Multiple AWS Regions, on page 146 .

VIM Connector Configuration for VMware vCloud Director (vCD)

You must configure a VIM connector to connect to the vCD organization. The organization and the organization user must be preconfigured in the VMware vCD. For the deployment datamodel, see the Deploying Virtual Network Functions on VMware vCloud Director (vCD).

The VIM connector details are as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<esc_system_config xmlns="http://www.cisco.com/esc/esc">
   <vim_connectors>
      <vim_connector>
         <id>vcd_vim</id>
         <type>VMWARE_VCD</type>
         <properties>
            <property>
               <name>authUrl</name>
                <!-- vCD is the vCD server IP or host name -->
               <value>https://vCD</value>
            </property>
         </properties>
         <users>
            <user>
              <!-- the user id here represents {org username}@{org name} -->
               <id>user@organization</id>
               <credentials>
                  <properties>
                     <property>
                        <name>password</name>
                          <!—the organization user’s password-->
                        <value>put user’s password here</value>
                     </property>
                  </properties>
               </credentials>
            </user>
         </users>
      </vim_connector>
   </vim_connectors>
</esc_system_config>
```

VIM Connector Configuration for VMware vSphere

You must configure a VIM connector to connect to the vSphere organization. The organization and the organization user must be preconfigured in the VMware vSphere. For the deployment datamodel, see the Deploying Virtual Network Functions on VMware vSphere.

The VIM connector details are as follows:

```
<esc_system_config xmlns="http://www.cisco.com/esc/esc">
  <vim_connectors>
    <vim_connector>
      <id>vimc-vc-lab</id>
      <type>VMWARE_VSPHERE</type>
      <properties>
        <property>
          <name>vcenter_ip</name>
          <value>IP_ADDRESS</value>
        </property>
        <property>
          <name>vcenter_port</name>
          <value>PORT</value>
        </property>
      </properties>
      <users>
        <user>
          <id>esc@vsphere.local</id>
          <credentials>
            <properties>
              <property>
                <name>vcenter_password</name>
                <value>PASS</value>
              </property>
            </properties>
          </credentials>
        </user>
      </users>
    </vim_connector>
  </vim_connectors>
</esc_system_config>
```

Adding VIM Connector to CSP Cluster

ESC supports adding the VIM connector on CSP Cluster with cluster_name property in an existing VIM connector payload.

Creating a VIM Connector

When a VIM connector is added with cluster_name property, ESC validates and checks if csp_host_ip is a part of the Cluster.

The following example shows how to add a VIM connector to the cluster:

```
<esc_system_config xmlns="http://www.cisco.com/esc/esc">
  <vim_connectors>
   <vim_connector>
      <id>CSP-3</id>
      <type>CSP</type>
```

```
<properties>
  <property>
    <name>csp_host_ip</name>
    <value> 168.20.117.16</value>
  </property>
  <property>
    <name>csp_host_port</name>
    <value>2022</value>
  </property>
   <property>
    <name>cluster_name</name>
    <value>Cluster_Test</value>
  </property>
  </properties>
<users>
  <user>
    <id>admin</id>
    <credentials>
      <properties>
        <property>
          <name>csp_password</name>
          <value>password1</value>
        </property>
      </properties>
    </credentials>
  </user>
</users>
```

Run the following command on ESC to add the VIM Connector on the cluster:

```
esc_nc_cli --user <username> --password <password> edit-config  add_vim_connector.xml
```

If csp_host_ip is not a part of the cluster, ESC shows the following error:

```
Cluster [Cluster_Test] is not available or csp_host_ip is not valid.
```

For more information on the deploying VNFs using ESC on CSP cluster, see the Deploying VNFs Using ESC on CSP Cluster chapter.

CHAPTER 8

VIM Connector Properties for Different VIMs

VIM Connector Properties

The VIM connector configuration enables ESC to connect to the VIM. The properties within the configuration provides details specifc to the VIM and its credentials. The table below shows the VIM connector properties for different VIMs. For more information, see Managing VIM Connectors, on page 49.

Table 4: VIM Connector Properties

VIM	Property	Reference
OpenStack	<properties> <property> <name>os_auth_url</name> <value>http://172.16.103.153:35357/v3</value> </property> <property> <name>os_project_domain_name</name> <value>default</value> </property> <property> <name>os_project_name</name> <value>admin</value> </property> <property> <name>os_identity_overwrite_endpoint</name> <value>http://some_server:some_port/</value> </property> <property> <name>os_network_overwrite_endpoint</name> <value>http://some_other_server:some_other_port/</value> </property>	For more information, see VIM Connector Configurations for OpenStack, on page 61.

VIM	Property	Reference
AWS	<properties> <property> <name>aws_default_region</name> <value>us-east-2</value> </property> </properties> <users> <user> <id>AWS_ACCESS_ID</id> <credentials> <properties> <property> <name>aws_secret_key</name> <encrypted_value>AWS_SECRET_KEY</encrypted_value> </property> </properties>	For more information, see VIM Connector Configurations for AWS, on page 67.
VMware vCD	<properties> <property> <name>authUrl</name> <!-- vCD is the vCD server IP or host name --> <value>https://vCD</value> </property> </properties> <users> <user> <!-- the user id here represents {org username}@{org name} --> <id>user@organization</id> <credentials> <properties> <property> <name>password</name> <!—the organization user's password--> <value>put user's password here</value> </property> </properties> </credentials> </user>	For more information, see VIM Connector Configuration for VMware vCloud Director (vCD), on page 69.

VIM	Property	Reference
VMware vSphere		For more information, see VIM Connector Configuration for VMware vSphere, on page 70.
Cisco Cloud Services Provider (CSP) 2100		For CSP extenstions, see Cloud Services Provider Extensions, on page 385.

VMware vSphere

```
<esc_system_config
xmlns="http://www.cisco.com/esc/esc">

  <vim_connectors>
    <vim_connector>
      <id>vimc-vc-lab</id>

<type>VMWARE_VSPHERE</type>
      <properties>
        <property>

<name>vcenter_ip</name>

<value>IP_ADDRESS</value>
        </property>
        <property>

<name>vcenter_port</name>
          <value>PORT</value>
        </property>
      </properties>
      <users>
        <user>

<id>esc@vsphere.local</id>
          <credentials>
            <properties>
              <property>

<name>vcenter_password</name>

<value>PASS</value>
              </property>
            </properties>
          </credentials>
        </user>
      </users>
    </vim_connector>
  </vim_connectors>
</esc_system_config>
```

Cisco Cloud Services Provider (CSP) 2100

```
<properties>
        <property>
        <name>csp_host_ip</name>
       <value>172.16.89.100</value>
        </property>
        <property>
       <name>csp_host_port</name>
          <value>2022</value>
        </property>
      </properties>
      <users>
        <user>
          <id>admin</id>
          <credentials>
            <properties>
              <property>
           <name>csp_password</name>
            <value>*******</value>
              </property>
            </properties>
```

CHAPTER 9

Authenticating External Configuration Files

Authenticating External Configuration Files

Prior to Cisco ESC Release 4.0, ESC supports several external configuration files and scripts as part of day 0 configuration, monitoring, deployment and LCS actions. ESC supports getting these files from a remote server with or without authentication as part of the deployment.

Starting from ESC Release 4.0, the file locator attribute is defined at the deployment level, that is, directly under the deployment container. This allows multiple VM groups and their day 0 configuration and LCS actions to reference the same file locator wherever needed within the deployment.

Sample deployment data model is as follows:

```
<esc_datamodel xmlns="http://www.cisco.com/esc/esc">
  <tenants>
    <tenant>
      <name>sample-tenant</name>
      <deployments>
        <deployment>
          <name>sample-deployment</name>
          <file_locators>
            <file_locator>
              <name>post_deploy_alive_script</name>
              <remote_file>
                <file_server_id>http-my-server</file_server_id>
                <remote_path>/share/qatest/vnfupgrade/lcspostdeployalive.sh</remote_path>
                <local_target>vnfupgrade/lcspostdepalive.sh</local_target>
                <persistence>FETCH_ALWAYS</persistence>
                <properties/>
              </remote_file>
            </file_locator>
            <file_locator>
              <name>asa-day0-config</name>
              <remote_file>
                <file_server_id>http-my-server</file_server_id>
                <remote_path>/share/qatest/day0/asa_config.sh</remote_path>
                <local_target>day0.1/asa_config.sh</local_target>
                <persistence>FETCH_ALWAYS</persistence>
              </remote_file>
            </file_locator>
```

```
    <file_locator>
      <name>scriptlocator</name>
      <remote_file>
        <file_server_id>dev_test_server</file_server_id>
        <remote_path>/share/users/gomoore/actionScript.sh</remote_path>
        <local_target>action/actionScript.sh</local_target>
        <persistence>FETCH_MISSING</persistence>
        <properties/>
      </remote_file>
    </file_locator>
  </file_locators>
  <policies>
    <policy>
      <name>VNFUPGRADE_POST_DEPLOY_ALIVE</name>
      <conditions>
        <condition>
          <name>LCS::POST_DEPLOY_ALIVE</name>
        </condition>
      </conditions>
      <actions>
        <action>
          <name>post_deploy_alive_action</name>
          <type>SCRIPT</type>
          <properties>
            <property>
              <name>file_locator_name</name>
              <value>post_deploy_alive_script</value>
            </property>
          </properties>
        </action>
      </actions>
    </policy>
  </policies>
<vm_group>
    <name>ASA-group</name>
    <image>ASAImage</image>
    <flavor>m1.large</flavor>
    <recovery_policy>
      <max_retries>1</max_retries>
    </recovery_policy>
    <scaling>
      <min_active>1</min_active>
      <max_active>1</max_active>
      <elastic>true</elastic>
    </scaling>
    <placement>
      <type>affinity</type>
      <enforcement>strict</enforcement>
    </placement>
    <bootup_time>120</bootup_time>
    <recovery_wait_time>60</recovery_wait_time>
    <interfaces>
      <interface>
        <nicid>0</nicid>
        <network>my-net</network>
      </interface>
    </interfaces>
    <kpi_data>
      <kpi>
        <event_name>VM_ALIVE</event_name>
        <metric_value>1</metric_value>
        <metric_cond>GT</metric_cond>
        <metric_type>UINT32</metric_type>
        <metric_occurrences_true>1</metric_occurrences_true>
```

```
                <metric_occurrences_false>5</metric_occurrences_false>
                <metric_collector>
                  <nicid>0</nicid>
                  <type>ICMPPing</type>
                  <poll_frequency>5</poll_frequency>
                  <polling_unit>seconds</polling_unit>
                  <continuous_alarm>false</continuous_alarm>
                </metric_collector>
              </kpi>
            </kpi_data>
            <rules>
        <admin_rules>
                <rule>
                  <event_name>VM_ALIVE</event_name>
                  <action>ALWAYS log</action>
                  <action>TRUE servicebooted.sh</action>
                  <action>FALSE recover autohealing</action>
                </rule>
              </admin_rules>
            </rules>
            <config_data>
              <configuration>
                <dst>ASA.static.txt</dst>
                <file_locator_name>asa-day0-config</file_locator_name>
              </configuration>
            </config_data>
            <policies>
              <policy>
                <name>SVU1</name>
                <conditions>
               <condition><name>LCS::DEPLOY_UPDATE::PRE_VM_VOLUME_DETACH</name></condition>

                </conditions>
                <actions>
                  <action>
                    <name>LOG</name><type>pre_defined</type>
                  </action>
                  <action>
                    <name>pre_vol_detach</name>
                    <type>SCRIPT</type>
                    <properties>
                      <property>
                        <name>file_locator_name</name>
                        <value>scriptlocator</value>
                      </property>
                      <property>
                        <name>exit_val</name>
                        <value>0</value>
                      </property>
                    </properties>
                  </action>
                </actions>
              </policy>
            </policies>
          </vm_group>
        </deployment>
      </deployments>
    </tenant>
  </tenants>
</esc_datamodel>
```

You must configure a remote server (file server) separately using the APIs before performing any deployment. Both REST and NETCONF APIs are supported

- A remote server with URL, authentication details including username, and password. You can either use REST or NETCONF to configure.

  Note The username and password are optional. The password is encrypted within ESC.

  You must configure the remote file server before deployment. You can update the credentials anytime during the deployment.

- File locator is added to the deployment data model. It contains a reference to the file server, and the relative path to the file to be downloaded.

To get files remotely with authentication, you must

1. Add a remote server.
2. Refer the remote server in the file locator. The file locator is part of config data in day 0 and LCS action blocks.
3. The day 0 and lifecycle stage (LCS) scripts will then be retrieved based on the file locator as part of the deployment.

The file server parameters include:

- id—used as the key and identifier for a file server.
- base_url—the address of the server. (e.g. http://www.cisco.com or https://192.168.10.23)
- file_server_user—the username to use when authenticating to the server.
- file_server_password—string containing the password for authenticating to the server. Initially the user provides a cleartext string, which is encrypted internally.
- properties—name-value pair for extensibility in the future.

The file locator parameters include:

- name—used as the key and identifier for a file locator.
- local_file or remote_file—choice of file location. Local file is used to specify a file existing on the ESC VM file system already. The remote_file is used to specify a file to fetch from a remote server.
    - file_server_id—id of the File Server object to fetch the file from.
    - remote_path—path of the file from the base_url defined in the file server object.
    - local_target—optional local relative directory to save the file.
    - properties—name-value pairs of of information that may be required.
    - persistence—options for file storage. Values include CACHE, FETCH_ALWAYS and FETCH_MISSING (default).
- checksum—optional BSD style checksum value to use to validate the transferred file's validity.

The file server values such as server connectivity, file existence, checksum and so on will be verified for validity.

The encrypted_data values in the file_server_password and properties encrypted_data fields are encrypted using AES/128bits in CFB mode for transmission.The data remains encrypted until it is required for accessing the server. For more information on encrypted values, see Encrypting Configuration Data.

Example of file servers,

```
<esc_datamodel xmlns="http://www.cisco.com/esc/esc">
  <file_servers>
    <file_server>
      <id>server-1</id> <!-- unique name for server -->
      <base_url>https://www.some.server.com</base_url>
      <file_server_user>user1</file_server_user>
      <file_server_password>sample_password</file_server_password>
      <!-- encrypted value -->
      <!-- properties list containing additional items in the future -->
      <properties>
        <property>
          <name>server_timeout</name>
          <value>60</value>
      <!-- timeout value in seconds, can be over-ridden in a file_locator -->
        </property>
      </properties>
    </file_server>
    <file_server>
      <id>server-2</id>
      <base_url>https://www.some.other.server.com</base_url>
          <properties>
        <property>
          <name>option1</name>
          <encrypted_value>$8$EADFAQE</encrypted_value>
        </property>
    </file_server>
  </file_servers>
</esc_datamodel>
```

Example for day 0 configuration

```
<esc_datamodel xmlns="http://www.cisco.com/esc/esc">
  <tenants><tenant>
      <name>sample-tenant</name>
      <deployments><deployment>
          <name>sample-deployment</name>
          <vm_group>
            <name>sample-vm-group</name>
            <config_data>
              <!-- exisiting configuration example - remains valid -->
              <configuration>
                <file>file:///cisco/config.sh</file>
                <dst>config.sh</dst>
              </configuration>
              <!-- new configuration including use of file locators -->
              <configuration>
                <dst>something</dst>
                <file_locators>
                  <file_locator>
                    <name>configlocator-1</name> <!-- unique name -->
                    <remote_file>
                      <file_server_id>server-1</file_server_id>
                      <remote_path>/share/users/configureScript.sh</remote_path>
                      <!-- optional user specified local silo directory -->
                      <local_target>day0/configureScript.sh</local_target>
                      <!-- persistence is an optional parameter -->
                      <persistence>FETCH_ALWAYS</persistence>
                      <!-- properties in the file_locator are only used for
                           fetching the file not for running scripts -->
```

```
                      <properties>
                        <property>
                          <!-- the property name "configuration_file" with value "true"
indictates this is the
                               script to be used just as using the <file> member case of
the configuration -->
                          <name>configuration_file</name>
                          <value>true</value>
                        </property>
                        <property>
                          <name>server_timeout</name>
                          <value>120</value> <!-- timeout value in seconds, overrides the
file_server property -->
                      </property>
                      </properties>
                    </remote_file>
                    <!-- checksum is an optional parameter.
                         The following algorithms are supported: SHA-1, SHA-224, SHA-256,
SHA-384, SHA-512 -->
                    <checksum>SHA256 (configureScript.sh) =
dd526bb2c0711238ec2649c4b91598fb9a6cf1d2cb8559c337c5f3dd5ea1769e</checksum>
                  </file_locator>
                  <file_locator>
                    <name>configlocator-2</name>
                    <remote_file>
                      <file_server_id>server-2</file_server_id>
                      <remote_path>/secure/requiredData.txt</remote_path>
                      <local_target>day0/requiredData.txt</local_target>
                      <persistence>FETCH_ALWAYS</persistence>
                      <properties/>
                    </remote_file>
                  </file_locator>
                </file_locators>
              </configuration>
            </config_data>
          </vm_group>
        </deployment></deployments>
    </tenant></tenants>
</esc_datamodel>
```

For more details on day 0 configuration and LCS actions, see Day Zero Configuration, and Redeployment Policy sections.

Encrypting Configuration Data

You can encrypt configuration data with secret keys and private information. In ESC, the day 0 configuration, day 0 configuration variables, VIM connector and VIM user, and LCS actions contain secret keys.

ConfD provides encrypted string types. Using the built-in string types, the encrypted values are stored in ConfD. The keys used to encrypt the values are stored in confd.conf.

Encrypting data is optional. You can use the encrypt_data value to store data if necessary.

In the example below, the day 0 configuration data has encrypted values. The encrypted_data uses the built in string type *tailf:aes-cfb-128-encrypted-string*.

```
choice input_method {
  case file {
    leaf file {
      type ietf-inet-types:uri;
    }
```

```
  }
  case data {
    leaf data {
      type types:escbigdata;
    }
  }
  case encrypted_data {
    leaf encrypted_data {
      type tailf:aes-cfb-128-encrypted-string;
    }
  }
}
```

Generating Advanced Encryption Standard (AES) Key

The AES key is 16 bytes in length, and contains a 32 character hexadecimal string.

You must configure the AES key in confd.conf for the encryption to work.

```
/opt/cisco/esc/esc-confd/esc_production_confd.conf
```

```
<encryptedStrings>
    <AESCFB128>
        <key>0123456789abcdef0123456789abcdef</key>
        <initVector>0123456789abcdef0123456789abcdef</initVector>
    </AESCFB128>
</encryptedStrings>
```

A default AES key is available in confD:

```
0123456789abcdef0123456789abcdef
```

The confD key is hard-coded. The escadm.py generates a random AES key and replaces the default confD AES key before confD starts.

Encrypting Variables

You can encrypt variables such as passwords and chassis ids within the day 0 configuration using *encrypted_val*. ESC allows you to choose either *val* or *encrypted_val* for variables within the deployment datamodel.

The text within the *encrypted_val* is encrypted into the confD Database (CDB) and PostgreSQL DB. The text is decrypted only at the point of use (not when the data is at rest). In the ESC logs, the text in *encrypted_val* is masked.

In the example below, the northbound client (Netconf or REST) populates the *encrypted_val* with plain text. When the deployment request is processed by ESC ConfD, the plain text is encrypted into the ESC databases.

```
<config_data>
  <configuration>
    <dst>vnf_day0.cfg</dst>
    <data>file://opt/cisco/esc/esc_database/vnf_day0.cfg</file>
    <variable>
      <name>user</name>
      <val>admin</val>
    </variable>
    <variable>
      <name>password</name>
      <encrypted_val>ADMIN-PASSWORD</encrypted_val>
    </variable>
```

When the *encrypted_val* is retrieved from ConfD configuration via netconf or CLI, it displays the plain text in the encrypted form.

```
<config_data>
  <configuration>
    <dst>vnf_day0.cfg</dst>
    <data>file://opt/cisco/esc/esc_database/vnf_day0.cfg</file>
    <variable>
      <name>user</name>
      <val>admin</val>
    </variable>
    <variable>
      <name>password</name>

<encrypted_val>$8$cVl6r9aR7W3wmHLYUrAOQHnjJGH0XltJjiCBTXANJFV0sJfb/NF+lEJiUA0j/JxA</encrypted_val>

    </variable>
```

Note A single value is stored in *encrypted_val*. The same variable value is substituted into the day 0 configuration template for all VMs in the scale group.

You can use *encrypted_val* in the day 0 configuration to secure the chassis id. The value of the chassis id is provided by the northbound client or operator performing the VNF upgrade (chassis id generated through the script during VNF deployment is not supported).

```
<config_data>
  <configuration>
    <dst>staros_param.cfg</dst>
    <file>file://opt/cisco/esc/images/staros_param_upf.cfg</file>
    <variable>
      <name>CHASSIS_ID</name>
      <encrypted_val>VALUE-PROVIDED-BY-NORTHBOUND-OPERATOR</encrypted_val>
    </variable>
```

For information on day 0 configuration, see Day Zero Configuration, on page 161.

Cisco Elastic Controller Services Script for Encoding ConfD AES Encrypted Strings

This feature provides scripts to encode the AES encrypted strings compatible for use in config requests, for example, dep.xml. Following are the two scripts (alternatives) that provide the same function:

- `esc_nc_cli encrypt`
- `esc_confd_encrypt` - It is a standalone script to use on the ESC VM or a remote Linux server with connectivity to the ESC VM.

The following commands help you to encrypt the plain text into AES encrypted string.

```
esc_nc_cli encrypt
```

For example:

```
admin@esc-01$ esc_nc_cli encrypt
Enter plain text (input is not echoed to terminal) >  *********
admin@127.0.0.1's password:
$8$aaCBcnVmZ+6lEV1FvhhitzQMLisLc3pxk1uUh+7DL4A=
```

```
admin@esc-01$ esc_nc_cli encrypt input.txt
admin@127.0.0.1's password:
$8$SLwFZuA0m0Rgf69fPNOeiq4ispm5H1SZIVGzzDd5R2g=
```

The following command is equivalent to `esc_nc_cli`, implemented as a separate, standalone script.

For example:

```
admin@esc-01$ esc_confd_encrypt
Enter plain text (input is not echoed to terminal) > ********
admin@localhost's password:
$8$QL5vFU1vt3KEs3kKIrC0+Faq8cF83WdptPO45GTIBGA=

admin@esc-01$ esc_confd_encrypt --file input.txt
admin@localhost's password:
$8$uzN7+kMgCf4RLxB5R0qMnLIbixO6EUpliUuHJRwR944=
```

The following command connects to ConfD CLI ssh (port 2024).

For example:

```
admin@esc-01$ esc_nc_cli cli
ssh -o StrictHostKeyChecking=no -p 2024 admin@127.0.0.1
admin@127.0.0.1's password: *****

admin connected from 127.0.0.1 using ssh on esc-01
admin@esc-01>
```

Using the Scripts from a Remote Host

You can use both the scripts to perform the encryption at a remote ESC. For example, a linux server with connectivity to the ESC VM, northbound client or administrative 'jump host'.

For example:

```
abc@my-server-39:~$ esc_confd_encrypt --host 172.25.0.89 --user admin
Enter plain text (input is not echoed to terminal) >
admin@172.25.0.89's password:
$8$VUnQkT30fKqAWWCiyDPkqUjS+jDd0/sNIyGNd4bVppE=

abc@my-server-39:~$ esc_nc_cli encrypt --host 172.25.0.89 --user admin
Enter plain text (input is not echoed to terminal) >
admin@172.25.0.89's password:
$8$uRBKqpZZ9rcUIrfBam0WfCXq3tirTD+FRcafBqAArRs=

abc@my-server-39:~$ esc_nc_cli encrypt --host 172.25.0.89 --user admin --password 'REDACTED'
Enter plain text (input is not echoed to terminal) >
$8$iG9vvLAqk69wUSMVMVf5XDpwkdDi/P1V9ucJlXKn2NQ=
```

Enabling Password-less Access to the Scripts with Public Key Authentication

There are two methods to enable password-less (public key authentication) to ConfD (netconf and ssh cli) for direct use of through wrapper utilities, for examples `esc_nc_cli` and `esc_confd_encrypt`.

Following is an example for creating a private key pair and config public key auth in ConfD (preferred):

```
admin@esc-01$ ssh-keygen -t rsa -b 2048 -C "admin" -N "" -f ~/.ssh/test_confd_rsa
Generating public/private rsa key pair.
Your identification has been saved in /home/admin/.ssh/test_confd_rsa.
Your public key has been saved in /home/admin/.ssh/test_confd_rsa.pub.
The key fingerprint is:
```

```
SHA256:u3/dpc4iY6/60fiGjGeJjMcigUKlSrxCptZWYo8JQ6o admin
The key's randomart image is:
+---[RSA 2048]----+
|                 |
| . .             |
|+ o              |
|.X o .           |
|O *.*   S        |
|Eo.=..   . o    .|
|o..  . +.+oo.. o.|
|    . o *.Xo+.o .|
|     . ooB+Booo  |
+----[SHA256]-----+
admin@esc-01$ sudo mkdir --mode=700 -p /var/confd/homes/admin/.ssh
admin@esc-01$ sudo cp ~/.ssh/test_confd_rsa.pub /var/confd/homes/admin/.ssh/authorized_keys
admin@esc-01$ sudo chown -R esc-user:esc-user /var/confd/homes/admin/.ssh

admin@esc-01$ printf "value-of-encrypted_val" | esc_nc_cli encrypt --privKeyFile
~/.ssh/test_confd_rsa
$8$VmDBKYupSGUCaILw8g2VYykVD9D16jA44sQNglFUUAv+uQtO0BmEtSC85vfuRJu0

admin@esc-01$ printf "value-of-encrypted_val" | esc_confd_encrypt --privKeyFile
~/.ssh/test_confd_rsa
$8$oFXwX1jeIHVxmBuMdPe6Vz6usaSahPVh0gZEGHm0uoAvK+twC0kUK5w7/QY0goUM

admin@esc-01$ cat .ssh/config
Host localhost 127.0.0.1
   Port 2024
   IdentityFile ~/.ssh/test_confd_rsa

admin@esc-01$ printf "value-of-encrypted_val" | esc_nc_cli encrypt
$8$GZ4+2nSo/YklKVk8RTdNR9oDJjWe89VsUiUR2FnIwtW4WPSXLivOXbmZnHR2YpfP

admin@esc-01$ printf "value-of-encrypted_val" | esc_confd_encrypt
$8$ggQaMq3QEIhS+1P8gmtr47LwdPyrCFoHHC2jzv2vKnxBFvIPNQapHurj+bcHfpEe
```

Following is the example for enabling ConfD keys to access ConfD with built-in esc-nc-admin account (offered for backwards compatibility):

```
admin@esc-01$ sudo escadm confd keygen --user admin
Generated SSH key pair for user admin and authorized them for user esc-nc-admin

admin@esc-01$ printf "value-of-encrypted_val" | esc_nc_cli encrypt
$8$4c5m8cqK21VNyb1gCfc77p41LKxA9Ar8n6CApQwNst8yk/ilDphiDXetmHPmKuvP

admin@esc-01$ printf "value-of-encrypted_val" | esc_confd_encrypt
$8$yY8sG6leUkrnY+fBUrYVmnwPSBY9aIrUKXmpaHVGfvNWggLuSPkqZcRCjejPej+y
```

PART III

Onboarding Virtual Network Functions

CHAPTER 10

Onboarding Virtual Network Functions

You can onboard any new VNF on OpenStack and VMware vCenter. To onboard the VNF, you must fulfill the prerequisites, and prepare the deployment data model. This chapter describes the prerequisites and the procedure to prepare the deployment data model on OpenStack and VMware vCenter.

Onboarding Virtual Network Functions on OpenStack

You must fulfill the following prerequisites before onboarding VNFs on OpenStack:

- The VNF image formats must be compatible with OpenStack, for example qcow2 format. The image can be onboarded on OpenStack either by the OpenStack glance client, or by ESC using the NETCONF or REST APIs.
- The day 0 configuration file passed into the VM must be compatible with either the OpenStack config-drive or the user-data, so that the VMs can use the day 0 configuration details for bootstrap mechanism.
- The day 0 variables must be in plain text format and use the predefined day 0 variables, so that the VMs can use the static IP information available in the day 0 file.

Preparing the Deployment Data Model

You must prepare the deployment data model as part of VNF onboarding. The deployment data model is an XML file (template) that describes the operational behavior such as resource requirements, networking, monitoring KPI, placement policies, lifecycle stages (LCS), scaling rules and so on.

Preparing the Data Model for OpenStack Deployment

The VNF deployment data model is an XML file or template describing the resource requirements, networking, day zero configuration, and other service operational behaviors such as monitoring KPI, placement policies, lifecycle stages, scaling rules and so on.

To onboard a VNF and define the VNF services in the deployment data model, you must:

1. Prepare the VM Resources
2. Describe the VNF Networking

3. Prepare the Day Zero Configuration
4. Define the operational behaviors such as metrics and KPIs, in the deployment data model

Preparing the VM Resources

The deployment data model refers to resources such as tenants, images, flavors, volumes and so on to deploy the VNFs. You can either create these resources using ESC, or use the preexisting resources already available on OpenStack. For more information, see Managing Resources Overview, on page 17.

A sample data model with the resources is as follows:

```
<?xml version="1.0" encoding="ASCII"?>
<esc_datamodel xmlns="http://www.cisco.com/esc/esc">
  <tenants>
    <tenant>
     <name>vnf_tenant</name>
     <deployments>
      <deployment>
      ...
    <name>vnf-dep</name>
      <vm_group>
       <name>Grp1</name>
       <flavor>vnf_flavor</flavor>
       <image>vnf_image</image>
      ...
     </vm_group>
     </deployment>
     </deployments>
    </tenant>
  </tenants>
</esc_datamodel>
```

Describing the VNF Network

The deployed VMs in the VNF must connect to specific networks for different purposes. These networks could be the management network, the internal networks within VMs, and so on. Make sure these networks are either available on OpenStack, or created by ESC. You must define these networks in the deployment data model to create them during deployment. For more information, see Managing Networks, on page 27.

A sample deployment data model showing how to create networks and subnetworks, and specify the network connection for the VM interfaces is as follows:

```
  <deployment>
           <name>vnf-dep</name>
           ...
           <networks>
             <network>
              <name>vnf_net</name>
              <shared>false</shared>
              <admin_state>true</admin_state>
              <subnet>
               <name>vnf_subnet</name>
               <ipversion>ipv4</ipversion>
               <dhcp>true</dhcp>
               <address>172.16.0.0</address>
               <netmask>255.255.255.0</netmask>
               <gateway>172.16.0.1</gateway>
              </subnet>
             </network>
           </networks>
          ...
```

```
          </deployment>

        </deployments>

            <vm_group>
             <name>Grp1</name>
             ...
              <interfaces>
              <interface>
               <nicid>0</nicid>
                <network>vnf_management</network>
               </interface>
                <interface>
                 <nicid>1</nicid>
                 <network>vnf_net</network>
                </interface>
               </interfaces>
              ...
             </vm_group>
```

Preparing the Day Zero Configuration

As part of the Day Zero configuration, the day zero file is passed into the VNF at the time of installation for bootstrapping. The day zero file is described in the deployment data model. For more information, see Day Zero Configuration, on page 161.

A sample describing the day zero file as config drive and user data is as follows:

```
<config_data>
          <configuration>
           <dst>--user-data</dst>
            <file>file://var/test/test-script.sh</file>
             </configuration>
             <configuration>
           <dst>/etc/configure-networking.sh</dst>
            <file>file://var/test/configure-networking.sh</file>
          </configuration>
         </config_data>
```

Defining the Operational Behavior

To onboard composite VNFs, you must configure some of the operational behaviors such as network connections, monitoring KPIs, placement policies, lifecycle stages, scaling rules and so on. These behaviors can be described in the deployment data model. For more information, see Deployment Parameters, on page 157.

Once you have prepared the deployment data model with these details, you have onboarded the VNF and instantiated the VNF service on OpenStack. Now you can deploy the VNF. When the VNF is deployed, ESC applies the day zero configuration for the new service. For more information, seeDeploying Virtual Network Functions on OpenStack, on page 107.

For information on preparing the VNFs on VMware vCenter, see Preparing the Data Model for VMware vCenter Deployment, on page 92.

Onboarding Virtual Network Functions on VMware vCenter

The following prerequisites must be fulfilled before onboarding VNFs on VMware vCenter:

- The VNF image format must be compatible with VMware vCenter, for example ova.
- The day 0 configuration file passed into the VM must be compatible with either the ovf properties or reading configurations from the CDROM drive.
- The day 0 variables must be in plain text format on the CDROM drive.

Preparing the Data Model for VMware vCenter Deployment

The VNF deployment data model is an XML file or template describing the resource requirements, networking, day zero configuration, and other operational behaviors such as monitoring KPIs, placement policies, lifecycle stages, scaling rules and so on.

To onboard a VNF and define the VNF services in the deployment data model, you must:

1. Prepare the VM Resources
2. Describe the VNF Networking
3. Supporting Resource pool and Folder Specification
4. Prepare the Day Zero Configuration
5. Define the operational behaviors such as metrics and KPIs, in the deployment data model

Preparing the VM Resources

The deployment data model refers to resources to deploy the VNFs. An image (template) is the only resource referred in a VMware deployment. The image can be a pre-existing image, or created by ESC.

Note Tenants do not exist in a VMware vCenter deployment, but the default admin tenant is still required in the deployment data model.

A sample data model with image details are as follows:

```
<?xml version="1.0" encoding="ASCII"?>
<esc_datamodel xmlns="http://www.cisco.com/esc/esc">
  <tenants>
    <tenant>
     <name>admin</name>
     <deployments>
     <deployment>
     ...
     <name>vnf-dep</name>
      <vm_group>
        <image>vnf_image</image>
         ...
     </vm_group>
      </deployment>
     </deployments>
    </tenant>
  </tenants>
 </esc_datamodel>
```

On VMware vCenter, the placement policies and volume details are necessary for each vm_group. A zone_host type placement defines the target computing host or the cluster for a deployment. The volume defines the target data store for the deployment. The following deployment data model defines a deployment target to the computing-cluster cluster1 and allows ESC to choose a data store automatically.

```
<?xml version="1.0" encoding="ASCII"?>
<esc_datamodel xmlns="http://www.cisco.com/esc/esc">
  <tenants>
    <tenant>
      <name>admin</name>
      <deployments>
        <deployment>
  ...
        <name>vnf-dep</name>
          <vm_group>
            ...
 <placement>
   <type>zone_host</type>
  <zone>cluster1</zone>
   </placement>
     <volumes>
  <volume>
  <name>auto-select</name>
  <volid>1</volid>
  </volume>
  </volumes>
    </vm_group>
    </deployment>
     </deployments>
    </tenant>
  </tenants>
</esc_datamodel>
```

The following deployment data model defines a deployment target to the computing-host host1 and data store datastore1.

```
<?xml version="1.0" encoding="ASCII"?>
<esc_datamodel xmlns="http://www.cisco.com/esc/esc">
  <tenants>
    <tenant>
      <name>admin</name>
      <deployments>
        <deployment>
  ...
        <name>vnf-dep</name>
          <vm_group>
            ...
   <placement>
  <type>zone_host</type>
  <host>host1</host>
  </placement>
   <volumes>
     <volume>
      <name>datastore1</name>
      <volid>1</volid>
  </volume>
  </volumes>
   </vm_group>
    </deployment>
     </deployments>
    </tenant>
  </tenants>
</esc_datamodel>
```

Describing the VNF Network

The deployed VMs in the VNF must connect to specific networks for different purposes. Those networks could be the management network, the internal networks among VMs and other networks for different purposes. On VMware, a network refers to vDS port group, and a subnet refers to the IP pool under vCenter. ESC supports only static IP for VMware deployment. Make sure those networks are available on VMware vCenter, or created by ESC. To create a network during deployment, you can define the network in the deployment data model. The deployment data model is as follows:

```
    <deployment>
      <name>vnf-dep</name>
      ...
     <networks>
        <network>
          <name>vnf_management</name>
          <admin_state>true</admin_state>
          <number_of_ports>8</number_of_ports>
          <shared>false</shared>
          <switch_name>vdswitch1</switch_name>
          <vlan_id>0</vlan_id>
          <subnet>
            <name>vnf_management-subnet</name>
            <ipversion>ipv4</ipversion>
            <dhcp>false</dhcp>
            <address>172.16.0.0</address>
            <netmask>255.255.255.0</netmask>
            <gateway>172.16.0.1</gateway>
          </subnet>
        </network>
      </networks>
      ...

    </deployment>
  </deployments>
```

Note On VMware Vcenter, the nicid value starts from 1. On OpenStack the nicid value starts from 0.

```
<vm_group>
          <name>Grp1</name>
          ...
          <interfaces>
            <interface>
             <nicid>1</nicid>
             <network>vnf_management</network>
            </interface>
            <interface>
             <nicid>2</nicid>
             <network>vnf_net</network>
            </interface>
          </interfaces>
          ...
         </vm_group>
```

ESC to Support Resource Pool and Folders Other Than Default One (for vCenter)

As part of a vCenter or vSphere deployment, a user can optionally annotate the deployment XML descriptor to describe a resource pool or a folder for the deployment to be placed against.

Resource pools and folders are VM group level attributes and are specified using name or value pairs under an optional extensions section as shown in the following:

```
<deployment>
          <name>Deployment-Name</name>
          <vm_group>
            <name>VM-Group-Name</name>
            <bootup_time>300</bootup_time>
            . . .
            <interfaces>
              <interface>
                . . .
              </interface>
            </interfaces>
            <extensions>
              <extension>
                <name>vmware_vsphere_placement</name>
                <properties>
                  <property>
                    <name>folder</name>
                    <value>My_Folder</value>
                  </property>
                  <property>
                    <name>resource_pool</name>
                    <value>My_Resource_Pool</value>
                  </property>
                </properties>
              </extension>
            </extensions>
```

Note

1. The extension name must be *vmware_vsphere_placement*, otherwise, ESC ignores the properties underneath.
2. The folder "My_Folder" is created if the folder does not exist, otherwise the folder is used as it is.
3. The resource pool "My_Resource_Pool" should exist, otherwise an error is returned. An error returns because creating a resource pool requires many parameter values related to CPU and Memory usage that are not supported by the deployment attributes.
4. In the existing behaviour, if the extensions are not specified, then the default cluster resource pool and folders are used.
5. Specify a single resource pool or a single folder in the deployment XML. Both a single resource pool and a single folder are independent and do not rely on one another to be used in the deployment XML.

Preparing the Day Zero Configuration

As part of the day 0 configuration, the day 0 file is passed into the VNF at the time of installation for bootstrapping. The day 0 files have to be described in the deployment data model. For more information, see Day Zero Configuration, on page 161. The sample day zero file shows the day zero configurations passed in as files in CDROM content attached to the deployed VM.

```
<config_data>
           <configuration>
             <dst>day0-config</dst>
             <file>http://somehost:80/day0.txt</file>
           </configuration>
           <configuration>
```

```
        <dst>idtoken</dst>
        <file>http://somehost:80/idtoken.txt</file>
      </configuration>
    </config_data>
```

The sample below shows day 0 configurations passed through the ofv settings.

```
<config_data>
          <configuration>
           <dst>ovfProperty:mgmt-ipv4-addr</dst>
           <data>$NICID_1_IP_ADDRESS/16</data>
           </configuration>
          <configuration>
           <dst>ovfProperty:com.cisco.csr1000v:hostname</dst>
           <data>$HOSTNAME</data>
           <variable>
            <name>HOSTNAME</name>
            <val>csrhost1</val>
            <val>csrhost2</val>
           </variable>
         </configuration>
        </config_data>
```

Defining the Operational Behaviors

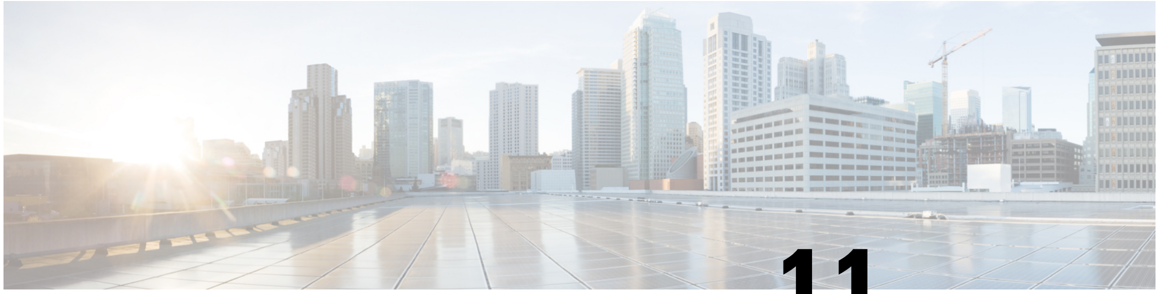
To onboard composite VNFs, you must configure some of the operational behaviors such as network connections, monitoring KPIs, placement policies, lifecycle stages, scaling rules and so on. These behaviors can be described in the deployment data model. For more information, see Deployment Parameters, on page 157.

Once you have prepared the deployment data model with these details, you have onboarded the VNF and instantiated the VNF service on OpenStack. Now you can deploy the VNF. When the VNF is deployed, ESC applies the day zero configuration for the new service. For more information, see Images on VMware vCenter, on page 135.

For information on preparing the VNFs on OpenStack, see Preparing the Data Model for OpenStack Deployment, on page 89.

PART IV

Deploying and Configuring Virtual Network Functions



CHAPTER 11

ESC Trunks and VLAN Functionality

ESC Trunks and VLAN Functionality

A VM may have one or more interfaces configured to access networks in a domain. For example eth0 is for data network and eth1 is for management networks.

If a VM needs to connect with multiple networks, then simplify the configuration using OpenStack Trunking.

Note Support for trunks and VLANs has been implemented in ESC 5.8.

In an ESC deployment, a VM group defines a trunk for example:

```
<vm_group>
    <name>...</name>
    <image>...</image>
    <flavor>...</flavor>
    <interfaces>
        <interface>
            <nicid>0</nicid>
            <network>parent-net</network>
        </interface>
    </interfaces>
    <trunks>
        <trunk>
            <name>trunk-name</name>
            <parent_nicid>0</parent_nicid>
            <subports>
                <subport>
                    <name>child-port</name>
                    <network>child-net</network>
                    <segmentation_type>vlan</segmentation_type>
                    <segmentation_id>500</segmentation_id>
                </subport>
            </subports>
        </trunk>
    </trunks>
</vm_group>
```

A description of elements under `<trunks>`

Element	Mandatory	Description
trunks	y	More than one trunk may be defined under trunks
trunk	y	Wraps the trunk definition
> name	y	Name of the trunk. This will be the name of the trunk in Openstack
> parent_nicid	y	This specifies the NIC ID as defined under <interfaces>, in this example, 0. The port created for this NIC will be used as the parent port for the trunk.
> subports	y	Wrapper for one or more subport elements
>> subport	y	Wraps the subport definition. At least one subport must be defined
>>> name	y	Name of the subport. This will be the name of the subport in Openstack
>>> network	y	Name or ID of the Openstack network to associate with this port. May be an external or ephemeral network defined for the deployment
>>> segmentation_type	n	Defaults to vlan. See Openstack API documentation for other possible values
>>> segmentation_id	y	The VLAN ID to assign to this subport

Creating trunks:

Submit the deployment XML using either REST or Netconf interface. The trunk is created before ESC deploys the VM.

```
<?xml version="1.0" encoding="UTF-8"?>
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2022-06-08T13:39:14.609+00:00</eventTime>
  <escEvent xmlns="http://www.cisco.com/esc/esc">
    <status>SUCCESS</status>
    <status_code>200</status_code>
    <status_message>Trunk trunk-D120-vm1: CREATE_TRUNK completed successfully</status_message>

    <event>
      <type>CREATE_TRUNK</type>
    </event>
  </escEvent>
</notification>
```

A HTTP callback message:

```
::ffff:127.0.0.1 - - [08/Jun/2022 13:39:09] "POST / HTTP/1.1" 200 2
------------------------------
REQUEST_METHOD:          POST
SERVER_PORT:             9009
PATH_INFO:               /
CONTENT_TYPE:            application/json
HTTP_ESC_TRANSACTION_ID  b9cce743-afd8-4eda-9303-5aaeccf4d400
HTTP_ESC_STATUS_MESSAGE *  Trunk trunk-D120-vm1: CREATE_TRUNK completed successfully
HTTP_ESC_STATUS_CODE     200
DATA:
------------------------------
* <JSON config data>
---------------END DATA---------------
```

Day-zero configuration:

A VM requires network configuration to use the trunk, such as the creation of sub-interfaces in the `/etc/network/interfaces` file or running IP link commands for testing. When ESC deploys the VM, the trunk sub-port information is available to day-zero scripts. The following variables are available in addition to the regular template variables. The index is the zero-based on the subport as defined under the trunk.

- SUBPORT_<index>_VLAN_ID
- SUBPORT_<index>_MAC_ADDRESS
- SUBPORT_<index>_ID
- SUBPORT_<index>_NETWORK
- SUBPORT_<index>_NAME
- SUBPORT_<index>_NETWORK_ID
- SUBPORT_<index>_SEGMENTATION_TYPE

Example config data:

```
<config_data>
    <configuration>
        <dst>--user-data</dst>
        <data>
#cloud-config
hostname: D120-vm2
password: secret
chpasswd: { expire: False }
ssh_pwauth: True
runcmd:
  - [ sh, -xc, "ip link add link ens3 name ens3.$SUBPORT_0_VLAN_ID address
$SUBPORT_0_MAC_ADDRESS type vlan id $SUBPORT_0_VLAN_ID" ]
  - [ sh, -xc, "ip link set dev ens3.$SUBPORT_0_VLAN_ID up" ]
  - [ sh, -xc, "dhclient -v ens3.$SUBPORT_0_VLAN_ID" ]
        </data>
    </configuration>
</config_data>
```

Querying trunks:

Using the REST or netconf API the deployment data of the trunk and subports is available.

```
<trunks>
    <trunk>
        <port_id>2f1dc1e6-a90a-4568-8c9e-965fda6c0cfb</port_id>
        <parent_nicid>0</parent_nicid>
        <id>13485eec-c0e0-41d0-b32e-b95ecd23ecef</id>
        <name>trunk-D120-vm1</name>
        <subports>
            <name>child-port-D120-vm1</name>
            <network>child-D120-net</network>
            <mac_address>fa:16:3e:72:cd:8d</mac_address>
            <segmentation_type>vlan</segmentation_type>
            <segmentation_id>120</segmentation_id>
            <id>1b6a0601-3281-4950-baca-f485470f74e3</id>
        </subports>
    </trunk>
</trunks>
```

Deleting the trunk:

Once the VM undeploys, the trunks and subports are deleted. A netconf message is posted for each trunk.

```
<?xml version="1.0" encoding="UTF-8"?>
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2022-06-08T13:24:47.834+00:00</eventTime>
  <escEvent xmlns="http://www.cisco.com/esc/esc">
    <status>SUCCESS</status>
    <status_code>200</status_code>
   <status_message>Trunk trunk-D120-vm2: DELETE_TRUNK completed successfully</status_message>

    <event>
      <type>DELETE_TRUNK</type>
    </event>
  </escEvent>
</notification>
```

A HTTP callback event:

```
::ffff:127.0.0.1 - - [08/Jun/2022 13:24:47] "POST / HTTP/1.1" 200 2
------------------------------
REQUEST_METHOD:          POST
SERVER_PORT:             9009
PATH_INFO:               /
CONTENT_TYPE:            application/xml
HTTP_ESC_TRANSACTION_ID  be42f1d2-a792-4516-91dc-583bdcd28c55
HTTP_ESC_STATUS_MESSAGE *  Trunk trunk-D120-vm2: DELETE_TRUNK completed successfully
HTTP_ESC_STATUS_CODE     200
DATA:
------------------------------
* <?xml version="1.0" encoding="UTF-8"?><!-- trunk details -->
---------------END DATA---------------
```

Modifying the trunk:

The trunk itself is not modified, but the subports can be added or removed and modified.

Submitting an HTTP PUT REST changes the trunk subports as required. Replace the Subports with those in the PUT request:

- Existing subports NOT in the requests are removed from the VM
- Existing subports in the request remain and keep their IDs, MAC address
- Adds new subports in the request.

```
curl -X PUT "http://localhost:8080/ESCManager/v0/deployments/D120" \
    -H "Callback: http://localhost:9009" \
    -H "Callback-ESC-Events: http://localhost:9009" \
    -H "Content-Type: application/xml" \
    -d "<esc_datamodel xmlns=\"http://www.cisco.com/esc/esc\"> ..."
```

Using Netconf `edit-config`

In a Netconf request, the rules are slightly different.

- Ignores the Existing Subports NOT in the request and continues unchanged.
- Adds new subports in the request.
- Removes the subports and annotates with nc:operation='delete' For example:

```
<subport nc:operation='delete'>
    <name>child-port-D120-vm1</name>
    <network>child-D120-net</network>
    <segmentation_type>vlan</segmentation_type>
    <segmentation_id>120</segmentation_id>
</subport>
```

Using `edit-config` (REST API)

Using the internal REST API, submit the Netconf payload

```
curl -X POST --location "http://localhost:8080/ESCManager/internal/conf/edit-config" \
    -H "Callback: http://localhost:9009" \
    -H "Callback-ESC-Events: http://localhost:9009" \
    -H "Content-Type: application/xml" \
    -d "<esc_datamodel
    xmlns=\"http://www.cisco.com/esc/esc\"
    xmlns:nc=\"urn:ietf:params:xml:ns:netconf:base:1.0\">
    ..."
```

Limitations

- Openstack does not allow adding a trunk to a deployed VM. It is possible to attach a secondary interface and use the port as the parent for the trunk
- Does not support scaling of a VM group.
- Does not support specifying a MAC address for the subport.

CHAPTER 12

Deploying Virtual Network Functions

Deploying Virtual Network Functions

You can orchestrate VNFs within a virtual infrastructure domain—either on OpenStack, VMware vCenter or AWS. A VNF deployment is initiated as a service request through northbound interface or the ESC portal. The service request comprises of templates that consist of XML payloads and deployment parameters. This chapter describes the procedures to deploy VNFs (OpenStack or VMware vCenter), and the operations that you can perform during a deployment. For more information on deployment parameters, see Configuring Deployment Parameters.

Important You can assign a static IP address to connect the network to the VNF. The deployment datamodel introduces a new *ip_address* attribute to specify the static IP address. See the Cisco Elastic Services Controller Deployment Attributes for more details.

For details on basic interface settings, see the *Cisco Elastic Services Controller Administration Guide*.

CHAPTER 13

Deploying Virtual Network Functions on OpenStack

Deploying Virtual Network Functions on OpenStack

This section describes several deployment scenarios for Elastic Services Controller (ESC) and the procedure to deploy VNFs. The following table lists the different deployment scenarios:

Scenarios	**Description**	**Resources**	**Advantages**
Deploying VNFs on a single VIM by creating images and flavors through ESC	The *deployment data model* refers to the images and flavors created and then deploys VNFs.	Images and Flavors are created through ESC using NETCONF/REST APIs.	• The images and flavors can be used in multiple VNF deployments. • You can delete resources (images, flavors, and volumes) created by ESC.
Deploying VNFs on a single VIM using out-of-band images, flavors, volumes, and ports	The *deployment data model* refers to the out-of-band images, flavors, volumes, and ports in OpenStack and then deploys VNFs.	Images, Flavors, Volumes, and Ports are not created through ESC.	• The images, flavors, volumes, ports can be used in multiple VNF deployments. • You cannot delete resources that are not created by through ESC.

Scenarios	Description	Resources	Advantages
Deploying VNFs on multiple VIMs using out-of-band resources	The *deployment data model* refers to out-of-band images, flavors, networks and VIM projects and then deploys VNFs.	Images, Flavors, VIM projects (specified in the locators) and Networks are not created through ESC. They must exist out-of-band in the VIM.	You can specify the VIM (to deploy VMs) that needs to be configured in ESC within a deployment.

To deploy VNFs on multiple OpenStack VIMs, see Deploying VNFs on Multiple OpenStack VIMs.

Deploying VNFs on a Single OpenStack VIM

The VNF deployment is initiated as a service request either originating from the ESC portal or the northbound interfaces. The service request comprises of XML payloads. ESC supports the following deployment scenarios:

- Deploying the VNFs by creating images, and flavors through ESC
- Deploying the VNFs using out-of-band images, flavors, volumes, and ports

Before you deploy the VNFs, you must ensure that the images, flavors, volumes, and ports are available on OpenStack, or you must create these resources. For more details on creating images, flavors, and volumes see Managing Resources Overview, on page 17.

In a deployment, the out-of-band port must be created by the same tenant as the deployment. For more details on configuring ports, see Interface Configurations in the *Cisco Elastic Services Controller Administration Guide*.

To deploy VMs on multiple VIMs, see Deploying VNFs on Multiple OpenStack VIMs.

During a deployment, ESC looks for the deployment details in the deployment data model. For more information on the deployment data model, see Cisco Elastic Services Controller Deployment Attributes. If ESC is unable to find the deployment details for a particular service, it uses the existing flavors and images under the *vm_group* to continue the deployment. If ESC is unable to find the image and flavor details, the deployment fails.

Important You can also specify the subnet that is used for a network. The deployment data model introduces a new `subnet` attribute to specify the subnet. See the Cisco Elastic Services Controller Deployment Attributes for more details.

Note When a SERVICE_UPDATE configuration fails, the minimum and maximum number of VMs change causing a scale in or scale out. ESC cannot rollback the minimum or maximum number of VMs in the configuration because of errors caused on OpenStack. The CDB (an ESC DB) would be out of synchronization. In this case, another SERVICE_UPDATE configuration must be performed to do a manual rollback.

For deployments on OpenStack, the UUID or name can be used to refer to the image and flavor. The name has to be unique on the VIM. If there are multiple images with the same name, the deployment cannot identify the right image and the deployment fails.

All deployment and ESC event notifications show tenant UUID. For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2016-01-22T15:14:52.484+00:00</eventTime>
  <escEvent xmlns="http://www.cisco.com/esc/esc">
    <status>SUCCESS</status>
    <status_code>200</status_code>
    <status_message>VIM Driver: VM successfully created,
      VM Name:
[SystemAdminxyz_abc_NwDepMod1_0_5e6b7957-20e7-4df9-9113-e5fc8c047e91]</status_message>
    <depname>test_NwDepModVmGrp1</depname>
    <tenant>admin</tenant>
    <tenant_id>62cd11f560b44bf5815eaad41fc94c80</tenant_id>
 </event>
```

Reboot Time Parameter

A reboot time parameter is introduced in the deployment request. This provides more granular control to the reboot wait time of recovery in a deployment. In a deployment, when the VM reboots, the monitor is set with the reboot time. If the reboot time expires before receiving the VM ALIVE event, the next action such as VM_RECOVERY_COMPLETE, or undeploy is performed.

Note The bootup time is used, if the reboot time is not provided.

The data model change is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<esc_datamodel xmlns="http://www.cisco.com/esc/esc">
   <tenants>
      <tenant>
         <name>tenant</name>
         <deployments>
            <deployment>
               <name>depz</name>
               <vm_group>
                  <name>g1</name>
                  <image>Automation-Cirros-Image</image>
                  <flavor>Automation-Cirros-Flavor</flavor>
                  <reboot_time>30</reboot_time>
                  <recovery_wait_time>10</recovery_wait_time>
                  <interfaces>
                     <interface>
                        <nicid>0</nicid>
                        <port>pre-assigned_IPV4_1</port>
                        <network>my-network</network>
                     </interface>
                  </interfaces>
                  <kpi_data>
                     <kpi>
                        <event_name>VM_ALIVE</event_name>
                        <metric_value>1</metric_value>
                        <metric_cond>GT</metric_cond>
                        <metric_type>UINT32</metric_type>
                        <metric_collector>
                           <nicid>0</nicid>
                           <type>ICMPPing</type>
                           <poll_frequency>3</poll_frequency>
                           <polling_unit>seconds</polling_unit>
                           <continuous_alarm>false</continuous_alarm>
```

```
                              </metric_collector>
                           </kpi>
                        </kpi_data>
                        <rules>
                           <admin_rules>
                              <rule>
                                 <event_name>VM_ALIVE</event_name>
                                 <action>ALWAYS log</action>
                                 <action>TRUE servicebooted.sh</action>
                                 <action>FALSE recover autohealing</action>
                              </rule>
                           </admin_rules>
                        </rules>
                        <config_data />
                        <scaling>
                           <min_active>1</min_active>
                           <max_active>2</max_active>
                           <elastic>true</elastic>
                        </scaling>
                        <recovery_policy>
                           <recovery_type>AUTO</recovery_type>
                           <action_on_recovery>REBOOT_ONLY</action_on_recovery>
                           <max_retries>1</max_retries>
                        </recovery_policy>
                     </vm_group>
                  </deployment>
               </deployments>
            </tenant>
         </tenants>
      </esc_datamodel>
```

Sample notification is as follows:

```
20:43:48,133 11-Oct-2016 WARN  ===== SEND NOTIFICATION STARTS =====
20:43:48,133 11-Oct-2016 WARN  Type: VM_RECOVERY_INIT
20:43:48,134 11-Oct-2016 WARN  Status: SUCCESS
20:43:48,134 11-Oct-2016 WARN  Status Code: 200
20:43:48,134 11-Oct-2016 WARN  Status Msg: Recovery event for
VM [dep-12_CSR1_c_0_37827511-be08-4702-b0bd-1918cb995118] triggered.
20:43:48,134 11-Oct-2016 WARN  Tenant: gilan-test-5
20:43:48,134 11-Oct-2016 WARN  Service ID: NULL
20:43:48,134 11-Oct-2016 WARN  Deployment ID: f6ff8164-fe6d-4589-84fa-f39d676e9231
20:43:48,134 11-Oct-2016 WARN  Deployment name: dep-12
20:43:48,134 11-Oct-2016 WARN  VM group name: CSR1_cirros
20:43:48,134 11-Oct-2016 WARN  VM Source:
20:43:48,134 11-Oct-2016 WARN  VM ID: 90d2066c-9a07-485b-8f72-b51026a62922
20:43:48,134 11-Oct-2016 WARN  Host ID:
69c3fba0a5b5ffff211bd05b9da7e2130d98d005a9bef71ace7d09ff
20:43:48,134 11-Oct-2016 WARN  Host Name: my-server
20:43:48,134 11-Oct-2016 WARN  [DEBUG-ONLY] VM IP: 192.168.0.75;
20:43:48,135 11-Oct-2016 WARN  =====  SEND NOTIFICATION ENDS  =====
20:43:56,149 11-Oct-2016 WARN
20:43:56,149 11-Oct-2016 WARN  ===== SEND NOTIFICATION STARTS =====
20:43:56,149 11-Oct-2016 WARN  Type: VM_RECOVERY_REBOOT
20:43:56,149 11-Oct-2016 WARN  Status: SUCCESS
20:43:56,149 11-Oct-2016 WARN  Status Code: 200
20:43:56,150 11-Oct-2016 WARN  Status Msg: VM
[dep-12_CSR1_c_0_37827511-be08-4702-b0bd-1918cb995118] is rebooted.
20:43:56,150 11-Oct-2016 WARN  Tenant: gilan-test-5
20:43:56,150 11-Oct-2016 WARN  Service ID: NULL
20:43:56,150 11-Oct-2016 WARN  Deployment ID: f6ff8164-fe6d-4589-84fa-f39d676e9231
20:43:56,150 11-Oct-2016 WARN  Deployment name: dep-12
20:43:56,150 11-Oct-2016 WARN  VM group name: CSR1_cirros
20:43:56,150 11-Oct-2016 WARN  VM Source:
20:43:56,151 11-Oct-2016 WARN    VM ID: 90d2066c-9a07-485b-8f72-b51026a62922
```

```
20:43:56,151 11-Oct-2016 WARN    Host ID:
69c3fba0a5b5ffff211bd05b9da7e2130d98d005a9bef71ace7d09ff
20:43:56,151 11-Oct-2016 WARN    Host Name: my-server
20:43:56,152 11-Oct-2016 WARN    [DEBUG-ONLY] VM IP: 192.168.0.75;
20:43:56,152 11-Oct-2016 WARN  =====  SEND NOTIFICATION ENDS  =====
20:44:26,481 11-Oct-2016 WARN
20:44:26,481 11-Oct-2016 WARN  ===== SEND NOTIFICATION STARTS =====
20:44:26,481 11-Oct-2016 WARN  Type: VM_RECOVERY_COMPLETE
20:44:26,481 11-Oct-2016 WARN  Status: FAILURE
20:44:26,481 11-Oct-2016 WARN  Status Code: 500
20:44:26,481 11-Oct-2016 WARN  Status Msg: Recovery: Recovery completed with errors
```

Deploying VNFs on Multiple OpenStack VIMs

You can deploy VNFs on multiple VIMs of the same type using ESC. ESC supports deploying VNFs on multiple OpenStack VIMs. To deploy VMs on a single instance of OpenStack, seeDeploying Virtual Network Functions on OpenStack, on page 107.

To deploy VNFs on multiple VIMs, you must:

- Configure the VIM connector and its credentials
- Create a tenant within ESC

A VIM connector registers the VIM to ESC. To deploy VNFs on multiple VIMs, you must configure the VIM connector and its credentials for each instance of the VIM. You can configure a VIM connector either at the time of installation using the bootvm.py parameters, or using the VIM connector APIs. A default VIM connector is used for a single VIM deployment. For multi VIM deployment, the locator attribute is used to specify the VIM connector.

Typically an ESC, which supports multi VIM deployment has,

- a default VIM on which ESC creates and manages resources,
- and a non-default VIM on which only deployments are supported.

For more details, see Managing VIM Connectors, on page 49.

A root tenant in the data model hierarchy, which is a tenant within ESC (with the vim_mapping attribute set to false), and an out-of-band VIM tenant placed within the locator attribute must be available for deploying VNFs on multiple VIMs. If the root tenant does not exist, ESC can create a tenant during the multiple VIM deployment itself. You can create more than one ESC tenant. A user can use more than one tenant for multiple VIMs. For more information, see Managing Tenants, on page 19.

In a multiple VIM deployment, you can specify the target VIM for each VM group. You can deploy each VM group on a different VIM, but the VMs within the VM group are deployed on the same VIM.

You must add a locator attribute to the VM group in the data model to enable multiple VIM deployment. The locator node consists of the following attributes:

Note If the locator attribute is present in the deployment, then the VMs are deployed on the VIM specified in the locator. If the locator attribute is not present in the deployment, then the VMs are deployed on the default VIM. If the default VIM is also not present, then the request is rejected.

- vim_id—the vim id of the target VIM. ESC defines the vim_id and maps it to the vim_connector id. The vim connector must exist before deploying to the VIM specified by the vim_id.
- vim_project—the tenant name created in target VIM. This is an out-of-band tenant or project existing in OpenStack.

Note ESC supports only out-of-band resources (pre-existing resources) such as ports, images, flavors and volumes in a multi VIM deployment. The out of band port must be created by the same tenant as the deployment.

However, multi VIM deployment supports creating only ephemeral volumes using the locator attribute on a non-default VIM. Other resources cannot be created on a non-default VIM.

Recovery of VMs, scale in and scale out of VMs are supported within the same VIM on which the VMs are deployed. The VMs cannot scale or recover on different VIMs.

In the example below, the esc-tenant is a tenant within ESC. There is no mapping to the VIM tenant, and the VMs are not deployed on this esc-tenant. The vim_project, project-test-tenant (within the locator attribute), which is created out-of-band is the tenant on which the VMs are deployed.

```
<tenants>
   <tenant>
      <name>esc-tenant</name>
      <deployments>
         <deployment>
            <name>dep-1</name>
            <vm_group>
               <name>group-1</name>
               <locator>
                  <vim_id>vim-1</vim_id>
                  <vim_project>project-test-tenant</vim_project>
               </locator>
            </vm_group>
         </deployment>
      </deployments>
   </tenant>
</tenants>
```

You can deploy VNFs on a single VIM as well with the locator attribute. That is, the datamodel with the locator attribute can also be used for deploying VMs on a single OpenStack VIM. To deploy without the locator attribute (ESC Release 2.x data model), see Deploying VNFs on a Single OpenStack VIM, on page 108.

The deployment data model is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<esc_datamodel xmlns="http://www.cisco.com/esc/esc" xmlns:ns0="http://www.cisco.com/esc/esc"
 xmlns:ns1="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:ns2="urn:ietf:params:xml:ns:netconf:notification:1.0"
xmlns:ns3="http://www.cisco.com/esc/esc_notifications">
   <tenants>
      <tenant>
         <name>test-esc-tenant1</name>
         <deployments>
            <deployment>
               <name>dep-1</name>
               <vm_group>
                  <name>g1</name>
                  <locator>
```

```
                     <vim_id>vim1</vim_id>
                     <vim_project>project-test</vim_project>
                  </locator>
                  <bootup_time>150</bootup_time>
                  <recovery_wait_time>30</recovery_wait_time>
                  <flavor>Automation-Cirros-Flavor</flavor>
                  <image>Automation-Cirros-Image</image>
                  <interfaces>
                     <interface>
                        <nicid>0</nicid>
                        <network>my-network</network>
                     </interface>
                  </interfaces>
                  <scaling>
                     <min_active>1</min_active>
                     <max_active>1</max_active>
                     <elastic>true</elastic>
                  </scaling>
                  <kpi_data>
                     <kpi>
                        <event_name>VM_ALIVE</event_name>
                        <metric_value>1</metric_value>
                        <metric_cond>GT</metric_cond>
                        <metric_type>UINT32</metric_type>
                        <metric_collector>
                           <type>ICMPPing</type>
                           <nicid>0</nicid>
                           <poll_frequency>3</poll_frequency>
                           <polling_unit>seconds</polling_unit>
                           <continuous_alarm>false</continuous_alarm>
                        </metric_collector>
                     </kpi>
                  </kpi_data>
                  <rules>
                     <admin_rules>
                        <rule>
                           <event_name>VM_ALIVE</event_name>
                           <action>ALWAYS log</action>
                           <action>TRUE servicebooted.sh</action>
                           <action>FALSE recover autohealing</action>
                        </rule>
                     </admin_rules>
                  </rules>
                  <config_data />
               </vm_group>
            </deployment>
         </deployments>
      </tenant>
   </tenants>
</esc_datamodel>
```

A sample multiple VIM deployment data model using out-of-band resources, and creating a root tenant as part of the deployment:

```
<esc_datamodel>
    <tenants>
        <tenant>
           <!-- This root level tenant is an ESC tenant either previously created or created
 here marked by vim_mapping atrribute. -->
            <name>esc-tenant-A</name>
            <vim_mapping>false</vim_mapping>
            <deployments>
                <deployment>
                    <name>dep-1</name>
                    <vm_group>
```

```
                        <name>Grp-1</name>
                        <locator>
                            <vim_id>SiteA</vim_id>
                            <!-- vim_project: OOB project/tenant that should already exist
 in the target VIM -->
                            <vim_project>Project-X</vim_project>
                        </locator>
                        <!-- All other details in vm group remain the same. -->
                        <flavor>Flavor-1</flavor>
                        <image>Image-1</image>
            ...
            ...
          </vm_group>
                </deployment>
            </deployments>
        </tenant>
    </tenants>
</esc_datamodel>
```

All the VIMs specified in a multi VIM deployment must be configured and in CONNECTION_SUCCESSFUL status for the request to be accepted by ESC. If a VIM specified in the deployment is unreachable or in any other status, the request is rejected.

You can apply the affinity and anti-affinity rules for VMs in a multiple VIM deployment. For more information, see Affinity and Anti-Affinity Rules on OpenStack, on page 193.

Multi VIM deployment supports recovery using the Lifecycle Stages (LCS). For more information on supported LCS, see Recovery Policy (Using the Policy Framework), on page 340. You can update an existing multi VIM deployment. However, the locator attribute within the VM group cannot be updated. For more information on updating an existing deployment, see Updating an Existing Deployment, on page 211.

CHAPTER 14

Deploying Virtual Network Functions on Multiple VIMs

Deploying Virtual Network Functions on Multiple VIMs

This section describes deployment scenarios for Elastic Services Controller (ESC) and the procedure to deploy different types of VIMs, like OpenStack, Cisco Cloud Services Platform (CSP), and vCloud Director (vCD).

Note

- ESC tenant is required for multi VIM type deployment.
- Deployments from same inter-deployment anti-affinity must be deployed in the same VIM in all the vm_groups.

The following table shows ESC VM and VNF deployment VIM type supported matrix:

Table 5: ESC VM and VNF deployment VIM type Support Matrix

ESC VM Installed On	OpenStack	vCloud Director	Cisco Cloud Services Platform
OpenStack	Supported	Supported	Supported
VMware vCenter	Supported	Supported	Supported

Sample Deployment Model

```
<?xml version="1.0"?>
<esc_datamodel xmlns="http://www.cisco.com/esc/esc">
  <tenants>
    <tenant>
      <name>VCDNCTestMVTypeDeployment-Tenant</name>
      <vim_mapping>false</vim_mapping>
      <deployments>
```

```
          <deployment>
            <name>VCDNCTestMVTypeDeployment-Dep</name>
            <vm_group>
              <name>VCDNCTestMVTypeDeployment-VCD-Group</name>
              <vim_vm_name>jenkins-VCDNCTestMVTypeDeployment-VCD-VM</vim_vm_name>
              <locator>
                <!-- vCD vim connector id -->
                <vim_id>VCD1</vim_id>
                <!-- vCD orgnization -->
                <vim_project>VAR{CFG{TARGET_LAB_0}:VCD_ORG1}</vim_project>
                <!-- vDC name -->
                <vim_vdc>VAR{CFG{TARGET_LAB_0}:VCD_ORG1_VDC1}</vim_vdc>
              </locator>
              <interfaces>
                <interface>
                  <nicid>0</nicid>
                  <network>VAR{CFG{TARGET_LAB_0}:VCD_MGT_NETWORK}</network>

<ip_address>VAR{CFG{TARGET_LAB_0}:VCD_MGT_NETWORK_IP_BASE}.VAR{CFG{TARGET_LAB_0}:STATIC_IP_RANGE}.0.2</ip_address>

                </interface>
              </interfaces>
            </vm_group>
            <vm_group>
              <name>VCDNCTestMVTypeDeployment-OS-Group</name>
              <vim_vm_name>jenkins-VCDNCTestMVTypeDeployment-OS-VM</vim_vm_name>
              <locator>
                <vim_id>Openstack1</vim_id>
                <!-- VIM Project = OOB Tenant -->

<vim_project>REPLACE_WITH_GENERATED_OOB_PROJECT_NAME_FOR_CFG{TARGET_LAB_1}</vim_project>
              </locator>
              <interfaces>
                <interface>
                  <nicid>0</nicid>
                  <network>VAR{CFG{TARGET_LAB_1}:MANAGEMENT_NETWORK}</network>
                </interface>
                <interface>
                  <nicid>1</nicid>
                  <network>VCDNCTestMVTypeDeployment-Net-2</network>
                </interface>
              </interfaces>
            </vm_group>
          </deployment>
        </deployments>
      </tenant>
    </tenants>
</esc_datamodel>
```

Supported Features in Multi VIM deployment

The following tables lists all the supported features in multi VIM deployment enviroment:

Table 6: Supported Features in Multi VIM deployment

Features	OpenStack	Cisco Cloud Services Platform	vCloud Director
Deployment with multiple VM group	Supported	Supported	Supported

Features	OpenStack	Cisco Cloud Services Platform	vCloud Director
Multiple Deployment with single VM group	Supported	Supported	Supported
Deployment scaling	Supported	Supported	Supported
Deployment Update	Supported	Supported	Supported
VIM Locators for ephemeral networks	Supported	Supported	Supported
Recovery	Supported	Supported	Supported
LCS Notification	Supported	Supported	Supported
VM Operations start/stop/reboot/disable_monitor/enable_monitor	Supported	Supported	Supported
ESC on OpenStack with/without default VIM	Supported	Supported	Supported

CHAPTER 15

Brownfield Deployments

Brownfield Enhacements to Support Openstack and ESC data reconciliation

Brownfield Deployment:

Brownfield deployments are ESC VNF deployments that allow the *target* ESC VM to manage a live VNF on the VIM.

Brownfield Deployments help to migrate the live VNF management from a *source*ESC VM to a *target*ESC VM without any disruption to the actual live VNF. The brownfield deployment process uses new and existing ESC APIs to create deployment data within the ESC datastores on the target ESC VM without actually creating resources on the VIM - simply validating existing VIM resources if and when required.

Quick starting Brownfield Deployment:

If the brownfield XML files that are my-brownfield-import.xml and my-brownfield-deployment.xml exist, then create a deployment in the target ESC VM using the brownfield APIs as shown:

Create a Brownfield Data:

Create the data and fix the errors if any returned

Example Payload

```
admin@esc_vm]$ esc_nc_cli import-deployment-data CREATE my-tenant my-deployment
/tmp/my-brownfield-import.xml
Import Deployment Data
/opt/cisco/esc/confd/bin/netconf-console --port=830 --host=127.0.0.1 --user=esc-nc-admin
--privKeyFile=/home/admin/.ssh/confd_id_rsa --privKeyType=rsa
--rpc=/tmp/tmp_esc_nc_cli.jtQHTuOubE
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="1">
  <imported_data xmlns="http://www.cisco.com/esc/esc">
    <import>
      <deployment_name>dep-complete</deployment_name>
      <project_name>dave-2000</project_name>
      <vms>
        <vm_details>

<generated_name>my-deployment_vm-gro_0_a4e82d3e-a3a5-403e-b321-cc0d7b1a779e</generated_name>
"<!-- Output removed for brevity --> "
```

```
        </vm_details>
      </vms>
    </import>
  </imported_data>
</rpc-reply>
```

Deploy VNF

After deploying the VNF, wait for SERVICE_ALIVE notification. If there is an error, undeploy the VNF, fix the errors and re deploy the VNF.

Example payload:

```
[admin@esc_vm]$ esc_nc_cli edit-config /tmp/my-brownfield-deployment.xml
Configure
/opt/cisco/esc/confd/bin/netconf-console --port=830 --host=127.0.0.1 --user=esc-nc-admin
--privKeyFile=/home/admin/.ssh/confd_id_rsa --privKeyType=rsa
--edit-config=/tmp/tmp_esc_nc_cli.53L6syLBhl
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="1">
  <ok/>
</rpc-reply>
```

Finalize the deployment:

In this state, the ESC VM manages the VNF

Example payload:

```
[admin@esc_vm]$ esc_nc_cli import-deployment-data FINALIZE my-tenant my-deployment
Import Deployment Data
/opt/cisco/esc/confd/bin/netconf-console --port=830 --host=127.0.0.1 --user=esc-nc-admin
--privKeyFile=/home/admin/.ssh/confd_id_rsa --privKeyType=rsa
--rpc=/tmp/tmp_esc_nc_cli.LY8Ai01yuz
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="1">
  <imported_data xmlns="http://www.cisco.com/esc/esc">
    <import>
      <deployment_name>dep-complete</deployment_name>
      <project_name>dave-2000</project_name>
      <vms>
        <vm_details>

<generated_name>my-deployment_vm-gro_0_a4e82d3e-a3a5-403e-b321-cc0d7b1a779e</generated_name>
"<!-- Output removed for brevity --> "
        </vm_details>
      </vms>
    </import>
  </imported_data>
</rpc-reply>
```

Delete the import data:

Note The following step is optional

```
[admin@esc_vm]$ esc_nc_cli import-deployment-data DELETE my-tenant my-deployment
Import Deployment Data
/opt/cisco/esc/confd/bin/netconf-console --port=830 --host=127.0.0.1 --user=esc-nc-admin
--privKeyFile=/home/admin/.ssh/confd_id_rsa --privKeyType=rsa
--rpc=/tmp/tmp_esc_nc_cli.LY8Ai01yuz
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="1">
```

```
  <imported_data xmlns="http://www.cisco.com/esc/esc">
    <import>
      <deployment_name>dep-complete</deployment_name>
      <project_name>dave-2000</project_name>
      <vms>
        <vm_details>

<generated_name>my-deployment_vm-gro_0_a4e82d3e-a3a5-403e-b321-cc0d7b1a779e</generated_name>
"<!-- Output removed for brevity --> "
        </vm_details>
      </vms>
    </import>
  </imported_data>
</rpc-reply>
```

Required files:

A Brownfield deployment requires at least two files to specify the VNF which is a **deployment XML** and an **import XML.** If the VNF uses VPC life cycle management (LCM) scripts that are GEN_VPC_CHASSIS_ID and GEN_VPC_SSH_KEYS during the original deployment, then further files specify the generated values. The same is discussed in the section "chassisID and user_key".

- Deployment XML:

Deployment XML details the XML data used during the original deployment. Ideally, XML is identical to the original *dep.xml* that is with the tenant's *vim_mapping* attribute set to true, but the ConfD API can be used to accurately generate the deployment XML on the source ESC VM if required.

- Import XML:

Import XML details the XML data that describes the VNF VMs and their ephemeral resources, sometimes known as *in-band* resources. The resource data exists on OpenStack, and ESC selects for the resource data during Brownfield deployment. All the in-band resources are supported including volumes, networks, subnets, and ports.

Note Both the import XML and the deployment XML specify all in-band and VIM resources required for the target deployment and therefore it is very important that both the XML are in sync with each other.

chassisID and user_key:

During a Brownfield deployment, if the original VNF deployment specifies LCM actions of type GEN_VPC_CHASSIS_ID, GEN_VPC_SSH_KEYS, or both, then the values that the LCM action executions created need to be specified in files and referenced appropriately.

Before the Brownfield deployment, the GEN_VPC_CHASSIS_ID LCM action generates a single string called a *chassisID,* and the generated value is saved to a local file on the *target* ESC VM.

The GEN_VPC_SSH_KEYS action generates two sets of SSH-related data that are SSH public and private key pairs called the *user_key*, and the generated values are saved to a local file on the *target* ESC VM.

Supported APIs:

Brownfield deployment requires two main APIs, they are, **ConfD API and ConfD or ESCManager REST APIs.**

- The ConfD API uses CREATE and FINALIZE keywords, and the most common access to the ConfD API is the esc_nc_cli script

- The ConfD or ESCManager REST API is to process a deployment XML.

ConfD: Brownfield CREATE:

The ConfD Brownfield CREATE API is used to load the import XML data that is the VNF resource data for VMs and their ephemeral data into ESC to be referenced during a VNF deployment.

The ConfD Brownfield CREATE API requires three mandatory arguments namely the tenant name, the deployment name, and a file specifying the import XML.

Example usage:

```
esc_nc_cli import-deployment-data CREATE my-tenant my-deployment /tmp/my-brownfield-import.xml
```

Upon invocation, the import XML contents are validated for structure and XML syntax and are stored in the ESC database. The actual data within the import XML is not validated for correctness as this can only be done at the deployment as shown in the following

Validation:

Upon invocation, perform the following validation steps and if there is an error, then fix and re-submit the data.

- XML is validated for syntax and to ensure mandatory values are present
- Import XML validates both the tenant and deployment names to ensure the values match
- If an index is specified, it must start at 0 for the VM group.

ConfD or ESCManager REST API: DEPLOY

Once import XML data loads and validates, the ConfD or ESCManager REST APIs are used to deploy the deployment XML data, in the same manner, that non- Brownfield deployment data is specified.

Example usage:

```
esc_nc_cli edit-config /tmp/my-brownfield-deployment.xml
```

Validation:

Upon invocation, ESC checks if the Brownfield CREATE operation previously exists for the same tenant and deployment, that is created during the ConfD Brownfield CREATE and if the deployment exists then ESC first validates all resource data to ensure the following:

- If the previously loaded import XML contains all the ephemeral resources specified in the deployment XML,
- If all resources exist on the VIM.

If either of the following fails, then the deployment itself will fail using familiar messaging.

Once the early validation passes, the deployment goes through the identical notification cycle as a non-Brownfield deployment, eventually generating a SERVICE_ALIVE notification or error notifications at the appropriate points in the workflow.

Note If there are any errors during deployment, for example, missing resources or VIM connectivity problems then the Brownfield deployment undeploys using the standard ESC undeployment APIs, once the underlying issue fixes, a re-deployment is attempted. This cycle of:

`deploy --> ERROR --> undeploy --> fix issue --> deploy` can be executed indefinitely until an error-free SERVICE_ALIVE notification is received.

ConfD: Brownfield FINALIZE

The ConfD Brownfield FINALIZE API is used to signal the *target* ESC VM which is ready to manage the VNF on the VIM as per any non-Brownfield VNF.

It requires two mandatory arguments that are the tenant name and the deployment name.

During the period where the Brownfield CREATE and DEPLOY APIs are used, but prior to this FINALIZE API called, the target ESC VM monitors the VNF, but recovery scenarios are not triggered if monitoring fails.

For example, if the VNF suddenly becomes un-pingable, then the target ESC VM does not invoke recovery policy nor it does not generate any notifications, therefore it is important that this final step occurs when the VNF on the VIM is active.

Furthermore, the target ESC VM does not implement VNF actions such as START, STOP, RECOVER, REBOOT, ENABLE/DISABLE MONITOR. If the actions are attempted, an error returns.

Validation

Brownfield deployments that have a SERVICE_ALIVE status is "finalized".

Example usage:

```
esc_nc_cli import-deployment-data FINALIZE my-tenant my-deployment
```

ConfD: Brownfield DELETE import data

The ConfD Brownfield DELETE API is used to remove all Brownfield data from the import tables.

It requires two mandatory arguments that are the tenant name and the deployment name.

Example usage

```
esc_nc_cli import-deployment-data DELETE my-tenant my-deployment
```

Validation

Only Brownfield deployments that are "finalized" can have their data deleted.

Example of import XML:

The following are some of the example import XML files or data snippets.

Basic VM plus an ephemeral volume and ports:

The following shows VNF with a single VM, one ephemeral volume, and two ephemeral ports. One port is specified using a single stack syntax, and the other port is specified using a dual-stack syntax.

Basic VM

```
<?xml version="1.0"?>
<import>
  <deployment_name>my-deployment</deployment_name>
  <project_name>my-tenant</project_name>
  <vms>
```

```
    <vm_details>
      <name>my-vm</name>
      <uuid>f4cad63c-a1c1-48ef-a3cd-8dd20abd2118</uuid>
      <vm_group>my-vm-group</vm_group>
      <attached_volume>
        <volume_id>df49a4a5-7def-450c-9881-b886b1abbe7f</volume_id>
        <volume_name>my-inband-volume</volume_name>
      </attached_volume>

<generated_name>my-deployment_vm-gro_0_2b92d247-7c08-48b1-a9f4-ff0849a82153</generated_name>

      <port>
        <port_id>4c2aa65d-e3fa-4529-a3f5-099031ffe6c3</port_id>
        <nicid>0</nicid>
      </port>
      <port>
        <port_id>2c627b23-ce8d-482a-9ea2-21d77df611b9</port_id>
        <fixed_ips>
          <address_id>0</address_id>
          <ip_address>192.26.13.442</ip_address>
        </fixed_ips>
        <nicid>1</nicid>
      </port>
    </vm_details>
  </vms>
</import>
```

The associated deployment XML excluding the non-relevant constructs is shown as follows:

Original deployment XML

```
<?xml version="1.0" encoding="UTF-8"?>
<esc_datamodel xmlns="http://www.cisco.com/esc/esc">
  <tenants>
    <tenant>
      <name>my-tenant</name>
      <vim_mapping>false</vim_mapping>  <!-- NOTE: Must always be "false" so that ESC does
 not try to create a new tenant -->
      <deployments>
        <deployment>
          <name>my-deployment</name>
          <vm_group>
            <name>my-vm-group</name>
            <image>Automation-Cirros-Image</image>
            <flavor>Automation-Cirros-Flavor</flavor>
            <vim_vm_name>my-vm</vim_vm_name>
            <bootup_time>180</bootup_time>
            <recovery_wait_time>180</recovery_wait_time>
            <volumes>
              <volume>
                <name>my-inband-volume</name>
                <volid>1</volid>
                <bus>virtio</bus>
                <size>2</size>
                <sizeunit>GiB</sizeunit>
                <type>LVM</type>
              </volume>
              <volume> <!-- NOTE: out of band resources do not need to be detailed in the
import XML -->
                <name>my-out-of-band-volume</name>
                <volid>0</volid>
                <bus>virtio</bus>
                <type>LVM</type>
              </volume>
            </volumes>
```

```
            <interfaces>
              <interface>
                <nicid>0</nicid>
                <network>esc-net</network>
              </interface>
              <interface>
                <nicid>1</nicid>
                <network>esc-net</network>
                <addresses>
                  <address>
                    <address_id>0</address_id>
                    <subnet>esc-subnet</subnet>
                  </address>
                </addresses>
              </interface>
              <interface> <!-- NOTE: out of band resources do not need to be detailed in
the import XML -->
                <nicid>2</nicid>
                <port>my-out-of-band-port</port>
              </interface>
            </interfaces>
 "<!-- Output removed for brevity --> "
          </vm_group>
        </deployment>
      </deployments>
    </tenant>
  </tenants>
</esc_datamodel>
```

Basic VM plus ephemeral network:

The following shows VNF with a single VM, a single ephemeral network, and a single ephemeral port specified using dual-stack syntax.

The difference between Basic VM plus ephemeral network and Basic VM plus and ephemeral volume is that the ephemeral network needs to be detailed in the import XML and the deployment XML.

Basic VM

```
<?xml version="1.0"?>
<import>
  <deployment_name>my-deployment</deployment_name>
  <project_name>my-tenant</project_name>
  <vms>
    <vm_details>
      <name>my-vm</name>
      <uuid>f4cad63c-a1c1-48ef-a3cd-8dd20abd2118</uuid>
      <vm_group>my-vm-group</vm_group>

<generated_name>my-deployment_vm-gro_0_2b92d247-7c08-48b1-a9f4-ff0849a82153</generated_name>

      <port>
        <port_id>2c627b23-ce8d-482a-9ea2-21d77df611b9</port_id>
        <fixed_ips>
          <address_id>0</address_id>
          <ip_address>10.120.04.198</ip_address>
        </fixed_ips>
        <nicid>0</nicid>
      </port>
    </vm_details>
  </vms>
  <network>
    <network_id>8abd5cb3-5107-4a63-bfd4-117a6d7e3824</network_id>
    <subnet>
      <subnet_id>a74bc215-172e-41f8-9f83-f578b4fb88d2</subnet_id>
```

```
      <name>my-suvnet</name>
    </subnet>
    <name>my-network</name>
  </network>
  <network>
    <network_id>xxxxx</network_id>
      </network>
</import>
```

The associated deployment XML excluding non-relevant constructs is shown as follows:

Original deployment XML

```
<?xml version="1.0" encoding="UTF-8"?>
<esc_datamodel xmlns="http://www.cisco.com/esc/esc">
  <tenants>
    <tenant>
      <name>my-tenant</name>
      <vim_mapping>false</vim_mapping>  <!-- NOTE: Must always be "false" so that ESC does
 not try to create a new tenant -->
      <deployments>
        <deployment>
          <name>my-deployment</name>
          <networks>
            <network>
              <name>my-network</name>
              <shared>false</shared>
              <locator>
                <vim_id>default_openstack_vim</vim_id>
                <vim_project>davwebst</vim_project>
              </locator>
              <subnet>
                <name>my-subnet</name>
                <ipversion>ipv4</ipversion>
                <dhcp>true</dhcp>
                <address>192.168.1.150</address>
                <netmask>255.255.255.0</netmask>
                <gateway>192.168.1.1</gateway>
              </subnet>
            </network>
          <vm_group>
            <name>my-vm-group</name>
            <image>Automation-Cirros-Image</image>
            <flavor>Automation-Cirros-Flavor</flavor>
            <vim_vm_name>my-vm</vim_vm_name>
            <bootup_time>180</bootup_time>
            <recovery_wait_time>180</recovery_wait_time>
            <interfaces>
              <interface>
                <nicid>0</nicid>
                <network>my-network</network>
                <addresses>
                  <address>
                    <address_id>0</address_id>
                    <subnet>my-subnet</subnet>
                  </address>
                </addresses>
              </interface>
            </interfaces>
 "<!-- Output removed for brevity --> "
          </vm_group>
        </deployment>
      </deployments>
    </tenant>
```

```
  </tenants>
</esc_datamodel>
```

Scaled out deployment:

The following shows VNF with a single VM and ephemeral port, one which is scaled by the other, either through the initial deployment, or a scaling KPI trigger.

Example payload:

```
<?xml version="1.0"?>
<import>
  <deployment_name>my-deployment</deployment_name>
  <project_name>my-tenant</project_name>
  <vms>
    <vm_details>
      <index>0</index> <!-- NOTE: index must start at zero, and is mandatory if multiple
vm details against the same vm group are specified -->
      <name>my-vm</name>
      <vm_group>my-vm-group</vm_group>

<generated_name>my-deployment_vm-gro_0_a8738b6c-1e0e-47b0-a469-db7cc63e6f92</generated_name>

      <port>
        <port_id>e974e20c-2e88-4321-beb9-5dd66e103865</port_id>
        <nicid>0</nicid>
      </port>
      <uuid>c06ab441-f895-4bd9-ab5a-9f731d42a3c1</uuid>
    </vm_details>
    <vm_details>
      <index>1</index> <!-- NOTE: index must be incremented by one if specifying vm details
 for a vm in the same vm group -->
      <name>my-vm_1</name>
      <vm_group>my-vm-group</vm_group>

<generated_name>deployment-brown_vm-gro_1_a5b5bb8b-e90e-47d4-95f0-e7481abce12e</generated_name>

      <port>
        <port_id>94290268-569d-46a2-bf73-0e81d8b18019</port_id>
        <nicid>0</nicid>
      </port>
      <uuid>317369cf-f722-4052-976b-035436fee303</uuid>
    </vm_details>
  </vms>
</import>
```

Note The scaled-out deployment is done by setting the scaling minimum and maximum values to "2" in the original deployment XML as shown:

```
<scaling>
  <min_active>2</min_active>
  <max_active>2</max_active>
</scaling>
```

ChassisID and User Key specification:

During a Brownfield deployment, if the original VNF deployment specifies LCM actions of type GEN_VPC_CHASSIS_ID, GEN_VPC_SSH_KEYS, or both, then the values that the LCM action executions created need to specify in files and referenced appropriately.

The values are specified using metadata configuration entries, with the following *type* key names supported:

```
• chassisID     - the chassis id
• user_key      - the private SSH user generated key
• user_kep.pub  - the pubic SSH user generated key
```

The following shows VNF with a single VM and ephemeral port and one which used both GEN_VPC_CHASSIS_ID and GEN_VPC_SSH_KEYS actions, therefore it requires three values that are *chassisID, user_key* and *user_key.pub* to exist within files and referenced in the import XML.

Note The actual data in the files must exist and be their unencrypted values.

Example payload:

```
<?xml version="1.0"?>
<import>
  <deployment_name>my-deployment</deployment_name>
  <project_name>my-tenant</project_name>
  <vms>
    <vm_details>
      <name>my-vm</name>
      <uuid>5d892d0d-c127-40f7-8ecd-d7660461b96b</uuid>
      <vm_group>my-vm-group</vm_group>

<generated_name>deployment-brown_vm-gro_0_c0083a56-bcd9-46c9-93f2-3c0405915963</generated_name>

      <metadata>
        <configuration>
          <entry>
            <type>chassisID</type>
            <file>file:///tmp/vpc_data/chassisID</file>
          </entry>
          <entry>
            <type>user_key</type>
            <file>file:///tmp/vpc_data/user_key</file>
          </entry>
          <entry>
            <type>user_key.pub</type>
            <file>file:///tmp/vpc_data/user_key.pub</file>
          </entry>
        </configuration>
      </metadata>
      <port>
        <port_id>5c0293f9-27cf-4054-98ad-20fd8e7ed5fd</port_id>
        <nicid>0</nicid>
      </port>
    </vm_details>
  </vms>
</import>
```

Note The file name path is the only possible manner to specify the values, and the <file> value **must**start with "file:///".

Note If there are multiple VMs to be specified, then the block repeats for each VM. This is the case as the script actions are at a deployment level and are therefore executed per VM during a deployment, and needs specifying per VM in the import XML.

The associated deployment XML excluding non-relevant constructs is shown as follows:

Original deployment XML

```
<?xml version="1.0" encoding="UTF-8"?>
<esc_datamodel xmlns="http://www.cisco.com/esc/esc">
  <tenants>
    <tenant>
      <name>my-tenant</name>
      <vim_mapping>false</vim_mapping>
      <deployments>
        <deployment>
          <name>my-deployment</name>
          <policies>
            <policy>
              <name>instantiate</name>
              <conditions>
                <condition>
                  <name>LCS::PRE_DEPLOY</name>
                </condition>
              </conditions>
              <actions>
                <action>
                  <name>GEN_VPC_CHASSIS_ID</name>
                  <type>SCRIPT</type>
                  <properties>
                    <property>
                      <name>CHASSIS_KEY</name>
                      <value>164c03a0-eebb-44a8-87fa-20c791c0aa6d</value>
                    </property>
                    <property>
                      <name>script_filename</name>
                    <value>file:///opt/cisco/esc/esc-scripts/esc_vpc_chassis_id.py</value>

                    </property>
                  </properties>
                </action>
              </actions>
            </policy>
            <policy>
              <name>instantiate_start</name>
              <conditions>
                <condition>
                  <name>LCS::PRE_DEPLOY</name>
                </condition>
              </conditions>
              <actions>
                <action>
                  <name>GEN_VPC_SSH_KEYS</name>
                  <type>SCRIPT</type>
                  <properties>
                    <property>
                      <name>script_filename</name>

<value>file:///opt/cisco/esc/esc-scripts/esc-vpc-di-internal-keys.sh</value>
                    </property>
                  </properties>
```

```
                </action>
              </actions>
            </policy>
          </policies>
          <vm_group>
            <name>my-vm-group</name>
            <image>Automation-Cirros-Image</image>
            <flavor>Automation-Cirros-Flavor</flavor>
            <vim_vm_name>my-vm</vim_vm_name>
            <bootup_time>180</bootup_time>
            <recovery_wait_time>180</recovery_wait_time>
            <interfaces>
              <interface>
                <nicid>0</nicid>
                <network>esc-net</network>
              </interface>
            </interfaces>
 "<!-- Output removed for brevity --> "
          </vm_group>
        </deployment>
      </deployments>
    </tenant>
  </tenants>
</esc_datamodel>
```

Lifecycle Management Script execution:

LCM scripts are not run during a Brownfield deployment as the executed scripts create resources or perform other action on the VM which executes once. Running the LCM scripts twice during a Brownfield deployment results in resource duplication errors or leaked resources.

Within the import XML, the default *to not run* all scripts can be toggled to ensure all LCM scripts for all policies are run by default. The following shows the new parent policies element which is top level import element:

```
<import>
  <deployment_name>my-deployment</deployment_name>
  <project_name>my-tenant</project_name>
  <vms>
   "<!-- Output removed for brevity --> "
  </vms>
  <policies>
    <execute>true</execute> <!-- run all LCM scripts for all policies -->
  </policies>
</import>
```

Script actions are specified per vm group and or per vm group including the policy name. For example, the following snippets show changing the default script execution policy to true for all scripts, except for those in the *my-vm-group-2* vm group for the instantiate policy and script which is a part of a terminate policy.

```
<import>
  <deployment_name>my-deployment</deployment_name>
  <project_name>my-tenant</project_name>
  <vms>
"<!-- Output removed for brevity --> "
  </vms>
  <policies>
    <execute>true</execute> <!-- run all LCM scripts for all policies -->
    <policy>
      <execute>false</execute>
      <name>instantiate</name>
      <vm_group>my-vm-group-2</vm_group>  <!-- this is optional -->
    </policy>
    <policy>
```

```
      <execute>false</execute>
      <name>terminate</name>
    </policy>
  </policies>
</import>
```

Limitations:

When deploying in *brownfield mode*, these are the following limitations to be aware of:

- ETSI deployments are currently not supported.
- Only ESC ConfD supports the CREATE and FINALIZE actions that is any Brownfield deployment uses either the confd_cli or esc_nc_cli APIs if deploying locally or invoke the ConfD API directly if deploying remotely. The ESCManager REST API does not support the CREATE nor FINALIZE actions, but can be used to support the deployment step.
- Tenant specification has vim_mapping attribute set to false to avoid tenant creation which results in a VIM error as the tenant already exists.
- If a VNF contains multiple VMs, but not all the VM data is specified during the Brownfield import, then un deploy and re deploy the VNF with the adjusted data.
- Only Brownfield deployments on an OpenStack VIM are supported, CVIM and VMWare Brownfield deployments fails with an appropriate error.
- The Brownfield APIs allow a single deployment to be specified in a single invocation.

Generation of Brownfield data:

Brownfield data uses both the import and the deployment XML but does not have a specific API for generation as the APIs are created manually.

There are two main ways to generate the data from a source ESC VM, given the deployment and tenant name.

- **Generate Import and deployment XML through ConfD API:**

ESC VM uses the ConfD API to generate both the original deployment XML and provides the information to manually generate the import XML.

- **Deployment XML**

Execute the following command to extract and save the original deployment XML for the tenant *my-tenant* and deployment *my-deployment*:

```
[admin@esc_vm]$ echo "show running-config esc_datamodel tenants tenant my-tenant deployments
 deployment my-deployment | display xml" | sudo /opt/cisco/esc/confd/bin/confd_cli -u admin
 -C > /tmp/my-tenant.my-deployment.config.xml
```

The resultant file is the replication of the original deployment XML but *set vim_mapping to false under the tenant.*

- **Import XML**

Execute the following to extract and save all the operational data for the tenant *my-tenant* and deployment *my-deployment*:

```
[admin@esc_vm]$ echo "show esc_datamodel opdata tenants tenant my-tenant deployments
my-deployment" | sudo /opt/cisco/esc/confd/bin/confd_cli -u admin -C >
/tmp/my-tenant.my-deployment.opdata.xml
```

This command produces a structured XML document outlining the operational data for the deployment, which is used to manually create the final XML import file.

- **Generate import and deployment XML using a script:**

Brownfield import uses the script `/opt/cisco/esc/escscripts/export_brownfield_data.py` to create both the deployment XML and import XML files to be used without modification.

The script runs on the source ESC VM therefore the script needs to be copied to the source ESC VM first if it is an older ESC which does not have this script, and extract using the ConfD API all the relevant data given a tenant and deployment name.

Therefore, the deployment itself must exist within the ConfD data store, but it does not necessarily have to be in a SERVICE_ALIVE state as the data is extracted irrespective of the deployment's status.

The script requires two mandatory arguments namely the tenant name and the deployment name. ConfD access defaults to RSA key-based access, but if this is not enabled, authentication can be achieved using a name and password.

Furthermore, additional arguments exist to adjust the output of the final import XML based on metadata which cannot be gathered from the ConfD data alone and therefore requires operator actions.

The following shows the output from the Usage page:

export_brownfield_data.py – Usage

```
[admin@esc_vm] > /opt/cisco/esc/esc-scripts/export_brownfield_data.py.local -h

Usage: export_brownfield_data.py -t <tenant> -d <deployment>
                                 [-u <confd_user>] [-p <confd_password>] [-l
<local_data_directory>]
                                 [-e <[True|False]>]
[--policy:<name>:<[True|False]>:[<vm_group>]]...
                                 [-c <config_entry_path>]

Mandatory Arguments:
-t <tenant>       The deployment tenant name.
-d <deployment>   The deployment name.

Optional Arguments:
-u <confd_user>            When connecting to the ConfD API, use <confd_user>.  Defaults
to 'admin'.
-p <confd_password>        When connecting to the ConfD API, use <confd_password> for
username/password
                           access with the <confd_user>.

NOTE: If <confd_password> is **not** set, then RPC authentication is assumed to have been
enabled and is
                           used instead when accessing the ConfD API (and <confd_user> is
ignored if set).

-l <local_data_directory>   The full path to a local directory that contains two ESC data
files specifying
                          what the files generated by the ConfD API - the ConfD API is not
 used.
                           The two file names need to be in the format:
                              <tenant>.<deployment>.config.xml - configuration data from
ConfD
                            <tenant>.<deployment>.opdata.xml - operational data from ConfD


-e <[True|False]>         Execute all scripts in every policy for every VM group.  Defaults
 to 'False'
```

```
--policy [--policy <name>,<[True|False]>,[<vm_group>]],...[--policy
<name>,<[True|False]>,[<vm_group>]]

                           1 or more policies to execute all scripts for.  The policy name
 and a boolean
                           indicating if scripts should be run for that policy.
                           If the scripts are at a VM group level, then the VM group must
be specified,
                           otherwise it is optional.

                           Incorrectly specified policies will be ignored.  Examples:

                           --policy instantiate,False --policy
VM_PRE_DEPLOY,True,vm_group_xyzzy

-c <config_entry_path>     Generates a <configuration> block under <metadata> for every VM
 in the
                           import XML with three <entry> children for chassidID, user_key
and user_key.pub
                          The path/to/target is mandatory and used as the <file> value for
 each <entry>.

                           Specify the path using an absolute path.  Example:

                           -c /tmp/target/data
```

Example usage

export_brownfield_data.py - example usage

```
[admin@esc-vm]$ ./export_brownfield_data.py -t my-tenant -d my-deployment

*** Parse and validate arguments ...
`--> Tenant = my-tenant
`--> Deployment = my-deployment
`--> Execute all scripts for all policies for all VM groups = None

*** Python version 2 ***
*** Current working directory is /home/admin/BROWNFIELD ***
*** Writing temporary files to /var/tmp/tmp4fk4Sq ***

*** Generate configuration and opdata tempfiles ...
`--> config data ...
`--> Adding <vim_mapping>false</vim_mapping> to config data XML as <locator> tag found in
the body.
`--> operational data ...

*** Generate configuration data for import - i.e. dep.xml ...
`--> Writing to /home/admin/BROWNFIELD/my-tenant.my-deployment.config.xml
`--> Creating vm_group: inband port map (if any exist) ...
`--> Found 3 interfaces.
`--> Found IN BAND port for vm group vm-group-complete with nicid 0
`--> Found IN BAND port for vm group vm-group-complete with nicid 1
`--> Creating vm_group: inband volume map (if any exist) ...
`--> Looking at vm_group name vm-group-complete
`--> Found 2 volumes.
`--> Found IN BAND volume for vm group vm-group-complete with volid 1
`--> Adding ephermeral networks if they exist ...
`--> In band networks found
`--> Adding policies if they were specified ...
`--> Skipping policies as none were specified ...
`--> Writing to /home/admin/BROWNFIELD/my-tenant.my-deployment.import.xml
*** Done ***
```

```
[admin@esc-vm]$ ls -l *.xml
total 2
-rw-r--r--.  1 admin admin  4383 May 17 11:21 my-tenant.my-deployment.config.xml
-rw-r--r--.  1 admin admin  1250 May 17 11:21 my-tenant.my-deployment.import.xml

[admin@esc-vm]$ head -20 my-tenant.my-deployment.import.xml
<?xml version="1.0"?>
<import>
  <deployment_name>my-deployment</deployment_name>
  <project_name>my-tenant</project_name>
  <vms>
    <vm_details>
      <index>0</index>
      <name>my-vm</name>
      <vm_group>my-vm-group</vm_group>
      <attached_volume>
        <volume_id>828051ba-fda8-4526-910a-caf36562cc26</volume_id>
        <volume_name>my-in-band-volume</volume_name>
      </attached_volume>

<generated_name>my-deployment-vm-gro_0_a4e82d3e-a3a5-403e-b321-cc0d7b1a779e</generated_name>

      <port>
        <port_id>5e6b0e08-8639-4965-b82f-237ad6c9fe26</port_id>
        <nicid>0</nicid>
      </port>
      <port>
        <port_id>69a69157-f47f-4514-af0c-23395185b5e8</port_id>
```

The two resultant files are copied directly to the *target* ESC VM and used as inputs to the Brownfield APIs without modification.

Removal of management from *source* ESC VM:

Once a VNF migrates successfully from a source ESC VM to a target ESC VM, the VNF is removed from the source ESC VM. This is necessary because, at this point, there are two ESC VMs managing a single VNF on OpenStack, and both ESC VMs responds if the VNF became unreachable.

Following are the four techniques which can be considered:

- Depending on the circumstances, the entire source ESC VM is shutdown if it is not managing other VNFs.
- The ESC VM is wiped clean of data, if it is not managing other VNFs.
- If the ESC VM needs to stay functional, and if the VM is a standalone configuration, then the VimManager service shuts down during a maintenance window and un deployment submits through any supported ESC API for the tenant and deployment combination. This results in an internal ESC warning as the VimManager is not reachable and therefore, the VIM cannot confirm the VNFs deletion. But this is only a warning and all ESC and ConfD data is removed for the deployment.
- If stopping VimManager service is not an option in H/A or A/A environments where a maintenance window is not acceptable for such operations, then the specific VimManager connection that the deployment uses can be invalidated by changing the password or URL to incorrect values. Then an un deployment submitted through any supported ESC API for the tenant and deployment combination can be attempted. Again, this generates an internal warning but the ESC and ConfD data is removed for the deployment.

CHAPTER 16

Deploying Virtual Network Functions on VMware

Images on VMware vCenter

You can deploy VNFs using out-of-band image definitions. The following table lists the deployment scenarios:

Scenario	Description	Data model templates	Images	Advantages
Deploying VNFs by creating Images through ESC **Important** Images are also referred to as Templates on VMware vCenter.	The process of VNF deployment is as follows: **1.** VNF Deployment- The *deployment data model* refers to the images created and then deploys VNFs.	- *deployment data model* - *image data model*	Images are created through ESC using REST APIs.	- The images can be used in multiple VNF deployments. - You can add or delete image definitions through ESC.

Scenario	Description	Data model templates	Images	Advantages
Deploying VNFs on a single VIM using out-of-band images	1. VNF Deployment- The *deployment data model* refers to the out-of-band images on VMware vCenter and then deploys VNFs.	- *deployment data model* - Image on VMware vCenter	Images cannot be created or deleted through ESC.	- The images can be used in multiple VNF deployments. - You can view images through ESC portal. - During out-of -band deployment, you can choose images.

Note ESC supports IPv6 deployment for VIM type VMware vSphere starting ESC 5.8 release, with a limiation that dual stack network creation is not supported. Meaning, either create an ipv4 or ipv6 subnet but **not** both.

Deploying VNFs on VMware vCenter VIM

This section describes the deployment scenario for Cisco Elastic Services Controller and the procedure to deploy VNFs on VMware.

The VNF deployment is initiated as a service request either originating from the ESC portal or the northbound interfaces. The service request comprises of XML payloads. ESC supports the following deployment scenarios:

- Deploying the VNFs by creating resources through ESC
- Deploying the VNFs using out-of-band resources

Before you deploy the VNFs, you must ensure that the resources are available on VMware vCenter, or you must create these resources. See Managing Resources Overview, on page 17. During a deployment, ESC looks for the deployment details in the deployment data model. For more information on the deployment data model, see Cisco Elastic Services Controller Deployment Attributes.

Note A single ESC instance only supports one vCenter Distributed Switch (vDS) per VIM locator:

- A vDS contains one or many ESXi hosts that are clustered.
- If the ESXi hosts are under one compute cluster, "Automation Level" must be set to "Manual" if DRS is ON.
- Clustered Data stores are not supported.
- If the hosts are clustered, only flat data stores under the cluster or under the datacenter are supported.

ESC only supports a default resource pool. You cannot add or create resource pools. When you see the error message "Networking Configuration Operation Is Rolled Back and a Host Is Disconnected from vCenter Server", it is due to a vCenter's limitation. The auto-select for datastore works as follows:

- ESC selects a host first. If deployment is cluster targeted, host will be selected based on the ratio of number of VMs against computing-host's capacity. Otherwise, host is selected as requested for host targeted deployment.
- From the host, datastore is picked based on its free space.

After every redeploy as part of recovery on VMware vCenter, the VM's interface(s) will have different MAC addresses.

Passing OVF Properties to a VM

As a part of deploying a VNF on VMware vCenter, you can pass the name value pair as OVF property to the VM. To pass these configurations while deploying a VNF, you must include additional arguments in the *deployment data model* template.

A sample configuration is as follows:

```
   <esc_datamodel ...>
    ...
    <config_data>
   <configuration>
      <dst>ovfProperty:mgmt-ipv4-addr</dst>
      <data>$NICID_1_IP_ADDRESS/24</data>
   </configuration>
   <configuration>
      <dst>ovfProperty:com.cisco.csr1000v:hostname</dst>
      <data>$HOSTNAME</data>
      <variable>
         <name>HOSTNAME</name>
         <val>csrhost1</val>
         <val>csrhost2</val>
      </variable>
   </configuration>
</config_data>
     ...
   </esc_datamodel>
```

Deploying VNFs on Multiple Virtual Data Centers (Multi-VDCs)

A Virtual Data Center (VDC) combines virtual resources, operational details, rules, and policies to manage specific group requirements. A group can manage multiple VDCs, images, templates, and policies. This group can allocate quotas and assign resource limits for individual groups at the VDC level.

To view the list of VDCs that are available and on the ESC portal, choose **Datacenters**.

Before you Begin

Before you deploy VNFs on multiple VDCs, ensure that the following conditions are met:

- Verify that a standard external network spanning both VDCs is available for the ESC to ping the deployed VMs.
- Verify that at least one management interface on the VMs is connected to the external network.
- Verify that the VDC is present in the vCenter.

Note

- ESC assumes all required resources to be created in VDC are out of band and present in the VDC.
- Currently, ESC can deploy in any VDC present in a vCenter. There is no scoping or restriction of VDCs that ESC can deploy in.

When you deploy a VNF, you must specify the virtual datacenter locator name on which the VNF needs to be provisioned.

A locator element is introduced in deployment request to create and delete resources.

The locator element contains:

- a datacenter name tag—to specify the target VDC for the resource (Deployment, Image, Network and Subnets).
- switch_name—to specify the target VDS to associate the network with.

Using the locator element,

- An image or a template can be created on another VDC by providing the datacenter attribute within the locator. For example,

```
<esc_datamodel xmlns="http://www.cisco.com/esc/esc">
    <images>
        <image>
            <name>automated-uLinux</name>
            <src>http://VAR_FILE_SERVER_IP/share/images/uLinux/uLinux.ovf</src>
            <locators>
                <datacenter>VAR_VDC2</datacenter>
            </locators>
        </image>
    </images>
</esc_datamodel>
```

- A network can be created and deleted from a VDC.

Note If the network is part of unified deployment, then the datacenter attribute is taken from the deployment attribute in deployment request.

```
<network>
    <locators>
        <datacenter>DC-03</datacenter>
        <switch_name>dvSwitch</switch_name>
    </locators>
    <name>test-yesc-net-u</name>
    <shared>false</shared>
    <admin_state>true</admin_state>
</network>
```

Cisco Elastic Services Controller Portal allows you to choose the VDC on which the VM is provisioned. When you are creating a service request, you can choose the VDC on which this VM is provisioned.

The *default_locators* container in ESC operational data shows default locators configured in ESC, however multiple vCenter VIMs can be configured.

Note The *default_locators* container is not displayed if there are no locators configured.

Sample operational data is as follows:

```
Operational Data
/opt/cisco/esc/confd/bin/netconf-console --port=830 --host=172.16.0.1 --user=admin
--privKeyFile=/var/confd/homes/admin/.ssh/confd_id_dsa --privKeyType=dsa --get -x
"esc_datamodel/opdata"
<?xml version="1.0" encoding="UTF-8"?><rpc-reply
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="1">
    <data>
        <esc_datamodel xmlns="http://www.cisco.com/esc/esc">
            <opdata>
                <status>OPER_UP</status>
                <stats>
                    <hostname>test-ESC-host</hostname>
                    <os_name>Linux</os_name>
                    <os_release>2.6.32-573.22.1.el6.x86_64</os_release>
                    <arch>amd64</arch>
                    <uptime>9481</uptime>
                    <cpu>
                        <cpu_num>4</cpu_num>
                    </cpu>
                </stats>
                <system_config>
                    <active_vim>VMWARE</active_vim>
                    <vmware_config>
                        <vcenter_ip>172.16.1.0</vcenter_ip>
                        <vcenter_port>80</vcenter_port>
                        <vcenter_username>root</vcenter_username>
                    </vmware_config>
                </system_config>
                <default_locators>
                    <datacenter>DC-4</datacenter>
                </default_locators>
                <tenants>
```

```
                    <tenant>
                        <name>admin</name>
                        <tenant_id>SystemAdminTenantId</tenant_id>
                    </tenant>
                </tenants>
            </opdata>
        </esc_datamodel>
    </data>
</rpc-reply>
[admin@test-ESC-host esc-cli]$
```

Deploying Virtual Network Functions on VMware vCloud Director (vCD)

This section describes the deployment scenario for ESC and the procedure to deploy VNFs on VMware vCloud Director (vCD). To install ESC on vCD, see the *Cisco Elastic Services Controller Install and Upgrade Guide*.

Resources such as organization, and organization VDC and so on must be created on vCD before deployment. For more information, see Managing Resources on vCloud Director (vCD), on page 47.

To deploy the VNF, you must:

1. Add a VIM connector, with the organization and organization user details preconfigured in the VMware vCD. See VIM Connector Configuration for VMware vCloud Director (vCD).

   The vim_vdc leaf under the locator refers to the vDC, for which the deployment is targeted.

2. Deploy the VNF with organization VDC, catalog and vApp template parameters preconfigured in the VMware vCD.

See the VMware vCloud Director Documentation to create these resources.

You must set the following key parameters, before deploying the VNFs on vCD:

- VMWARE_VCD_PARAMS—Specify the *VMWARE_VCD_PARAMS* parameter in the extensions section of the datamodel under each deployment section. The *VMWARE_VCD_PARAMS* parameter includes *CATALOG_NAME* and *VAPP_TEMPLATE_NAME*.
- CATALOG_NAME—Specify the name of the preconfigured catalog that contains references to vApp templates and the media images.
- VAPP_TEMPLATE_NAME—Specify the name of the preconfigured vApp template that contains virtual machine image that is loaded with an operating system, application, and data, it ensure that virtual machines are consistently configured across an entire organization.

A sample deployment is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<esc_datamodel xmlns="http://www.cisco.com/esc/esc" xmlns:ns0="http://www.cisco.com/esc/esc"
 xmlns:ns1="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:ns2="urn:ietf:params:xml:ns:netconf:notification:1.0"
xmlns:ns3="http://www.cisco.com/esc/esc_notifications">
   <tenants>
      <tenant>
         <!-- ESC scope tenant -->
         <name>esc-tenant</name>
         <vim_mapping>false</vim_mapping>
```

```
<deployments>
   <deployment>
      <!-- vApp instance name -->
      <name>vapp-inst1</name>
      <policies>
         <placement_group>
            <name>placement-anti-affinity</name>
            <type>anti_affinity</type>
            <enforcement>strict</enforcement>
            <vm_group>g1</vm_group>
            <vm_group>g2</vm_group>
         </placement_group>
      </policies>
      <extensions>
         <extension>
            <name>VMWARE_VCD_PARAMS</name>
            <properties>
               <property>
                  <name>CATALOG_NAME</name>
                  <value>catalog-1</value>
               </property>
               <property>
                  <name>VAPP_TEMPLATE_NAME</name>
                  <value>uLinux_vApp_Template</value>
               </property>
            </properties>
         </extension>
      </extensions>
      <vm_group>
         <name>g1</name>
         <locator>
            <!-- vCD vim connector id -->
            <vim_id>vcd_vim</vim_id>
            <!-- vCD organization corresponding to the vim connector -->
            <vim_project>organization</vim_project>
            <!-- vDC pre-preconfigured in organization -->
            <vim_vdc>VDC-1</vim_vdc>
         </locator>
         <!-- VM name in vAppTemplate -->
         <image>vm-001</image>
         <bootup_time>150</bootup_time>
         <recovery_wait_time>30</recovery_wait_time>
         <interfaces>
            <interface>
               <nicid>0</nicid>
               <network>MgtNetwork</network>
               <ip_address>172.16.0.0</ip_address>
            </interface>
         </interfaces>
         <scaling>
            <min_active>1</min_active>
            <max_active>1</max_active>
            <elastic>true</elastic>
            <static_ip_address_pool>
               <network>MgtNetwork</network>
               <ip_address>172.16.0.0</ip_address>
            </static_ip_address_pool>
         </scaling>
         <kpi_data>
            <kpi>
               <event_name>VM_ALIVE</event_name>
               <metric_value>1</metric_value>
               <metric_cond>GT</metric_cond>
               <metric_type>UINT32</metric_type>
```

```
         <metric_collector>
            <type>ICMPPing</type>
            <nicid>0</nicid>
            <poll_frequency>3</poll_frequency>
            <polling_unit>seconds</polling_unit>
            <continuous_alarm>false</continuous_alarm>
         </metric_collector>
      </kpi>
   </kpi_data>
   <rules>
      <admin_rules>
         <rule>
            <event_name>VM_ALIVE</event_name>
            <action>"ALWAYS log"</action>
            <action>"TRUE servicebooted.sh"</action>
            <action>"FALSE recover autohealing"</action>
         </rule>
      </admin_rules>
   </rules>
   <config_data>
      <configuration>
         <dst>ovfProperty:mgmt-ipv4-addr</dst>
         <data>$NICID_0_IP_ADDRESS/24</data>
      </configuration>
   </config_data>
</vm_group>
<vm_group>
   <name>g2</name>
   <locator>
      <!-- vCD vim connector id -->
      <vim_id>vcd_vim</vim_id>
      <!-- vCD organization corresponding to the vim connector -->
      <vim_project>organization</vim_project>
      <!-- vDC pre-preconfigured in organization -->
      <vim_vdc>VDC-1</vim_vdc>
 <locator>
    <vim_id>vcenter-22</vim_id>
    <vim_vdc>OTT-ESC-10</vim_vdc>
    </locator>
   </locator>
   <!-- VM name in vAppTemplate -->
   <image>vm-002</image>
   <bootup_time>150</bootup_time>
   <recovery_wait_time>30</recovery_wait_time>
   <interfaces>
      <interface>
         <nicid>0</nicid>
         <network>MgtNetwork</network>
         <ip_address>172.16.0.1</ip_address>
      </interface>
   </interfaces>
   <scaling>
      <min_active>1</min_active>
      <max_active>1</max_active>
      <elastic>true</elastic>
      <static_ip_address_pool>
         <network>MgtNetwork</network>
         <ip_address>172.16.0.1</ip_address>
      </static_ip_address_pool>
   </scaling>
   <kpi_data>
      <kpi>
         <event_name>VM_ALIVE</event_name>
         <metric_value>1</metric_value>
```

```
                        <metric_cond>GT</metric_cond>
                        <metric_type>UINT32</metric_type>
                        <metric_collector>
                           <type>ICMPPing</type>
                           <nicid>0</nicid>
                           <poll_frequency>3</poll_frequency>
                           <polling_unit>seconds</polling_unit>
                           <continuous_alarm>false</continuous_alarm>
                        </metric_collector>
                     </kpi>
                  </kpi_data>
                  <rules>
                     <admin_rules>
                        <rule>
                           <event_name>VM_ALIVE</event_name>
                           <action>"ALWAYS log"</action>
                           <action>"TRUE servicebooted.sh"</action>
                           <action>"FALSE recover autohealing"</action>
                        </rule>
                     </admin_rules>
                  </rules>
                  <config_data>
                     <configuration>
                        <dst>ovfProperty:mgmt-ipv4-addr</dst>
                        <data>$NICID_0_IP_ADDRESS/24</data>
                     </configuration>
                  </config_data>
               </vm_group>
            </deployment>
         </deployments>
      </tenant>
   </tenants>
</esc_datamodel>
```

To leverage any VM placement policies configured in vCD, the policy must be set to *Modifiable* in the vAppTemplate, use the following placement data model:

```
<vm_group>
  <name>vm_grp1</name>
  …
  <placement>
    <type>vm_policy</type>
    <enforcement>strict</enforcement>
    <policy>Test-VM-Placement-Policy-2</policy>
  </placement>
  …
</vm_group>
```

Once you supply the unique policy name, it notifies vCD to use that policy to determine the target host for the VM.

CHAPTER 17

Deploying Virtual Network Functions on Amazon Web Services

Deploying Virtual Network Functions on Amazon Web Services

This section describes the deployment scenario for Elastic Services Controller (ESC) and the procedure to deploy VNFs on Amazon Web Services (AWS). To install ESC on AWS, see the *Cisco Elastic Services Controller Install and Upgrade Guide*.

The following AWS resources must be created on AWS before deployment:

- Amazon Machine Images (AMI)
- Key Pairs
- Elastic IPs
- Security Groups
- Network Elements (such as VPCs, subnets, ACLs, gateways, routes and so on)

See the AWS documentation to create these resources.

For information on VIM connector configuration prior to AWS deployment, see "VIM Connector Configurations for AWS".

Scenarios	Description	Resources	Advantages
Deploying VNFs on a single VIM by creating Amazon Machine Image (AMI) and regions through ESC	The *deployment data model* refers to Amazon Machine Images (AMI), flavors, AWS regions, key pairs, security groups, network interfaces and VIM projects created, and then deploys VNFs.	Amazon Machine Images (AMI), flavors, AWS regions, key pairs, security groups, network interfaces, VIM projects (specified in the locators) and Networks created through ESC.	• You can specify the VIM (to deploy VMs) that needs to be configured in ESC within a deployment. • The images and flavors can be used in multiple VNF deployments. • You can delete resources created by ESC.
Deploying VNFs on multiple VIMs by creating AMIs and regions through ESC	The *deployment data model* refers to Amazon Machine Images (AMI), flavors, AWS regions, key pairs, security groups, network interfaces and VIM projects created and then deploys VNFs.	Images, Flavors, VIM projects (specified in the locators) and Networks created through ESC.	You can specify the VIM (to deploy VMs) that needs to be configured in ESC within a deployment.

For more details, see Deploying VNFs on a Single or Multiple AWS Regions, on page 146.

Deploying VNFs on a Single or Multiple AWS Regions

You can deploy VNFs on a single or multiple AWS regions or VIMs of the same type using ESC.

Note AWS is a Virtual Infrastructure Manager (VIM) for ESC. Further in this document, the terms AWS region and AWS VIM are used interchangeably.

To deploy VNFs on a single or multiple VIMs, you must:

- Configure the VIM connector and its credentials using the VIM connector API
- Create a tenant within ESC

A VIM connector registers the VIM to ESC. To deploy VNFs on a single or multiple AWS VIMs, you must configure the VIM connector and its credentials for each region of the VIM. You can configure a VIM connector using the VIM connector APIs. For more information, see VIM Connector Configurations for AWS, on page 67.

Note A default VIM connector is not supported for AWS deployment.

ESC creates a tenant within ESC with the *vim_mapping* attribute set to false. This tenant is independent of any VIM.

```
<esc_datamodel xmlns="http://www.cisco.com/esc/esc">
   <tenants>
      <tenant>
         <name>aws-sample-tenant</name>
         <vim_mapping>false</vim_mapping>
      </tenant>
   </tenants>
</esc_datamodel>
```

For a single or multiple AWS VIM deployment, you must specify the target region for each VM group.

You must add a locator attribute to the VM group in the datamodel to enable AWS VIM deployment. The locator node consists of the following attributes:

- vim_id—the vim id of the target VIM. ESC defines the vim_id and maps it to the vim_connector id. The vim connector must exist before deploying to the VIM specified by the vim_id.
- vim_project—the tenant name created in the target VIM. This is an out-of-band tenant or project existing in OpenStack.
- vim_region—the AWS region in which the VM groups are deployed. This is optional. If the vim region is not specified, then the VMs are deployed in the aws_default_region specified in the VIM connector.

  ```
  <locator>
     <vim_id>AWS_EAST_2</vim_id>
     <vim_region>us-east-1</vim_region>
     <!-- the deployment is going into
  North Virginia -->
  </locator>
  ```

  If the vim region is not specified,

  ```
  <locator>
     <vim_id>AWS_EAST_2</vim_id>
     <!-- the deployment is going into the default region Ohio (us-east-2)
  as defined in the VIM Connector example above -->
  </locator>
  ```

After configuring the VIM connectors and locators, you must pass certain resources as extensions to the deployment. In the example below, the elastic IP, key pair and source destination are passed as extensions to the AWS deployment.

```
<extensions>
   <extension>
      <name>AWS_PARAMS</name>
      <properties>
         <property>
            <name>elastic_ip</name>
            <value>13.56.148.25</value>
         </property>
         <property>
            <name>source_dest_check</name>
            <value>true</value>
         </property>
```

```
         <property>
            <name>key_pair_name</name>
            <value>esc-us-east-1</value>
         </property>
      </properties>
   </extension>
</extensions>
```

A sample AWS deployment is as follows:

```
<esc_datamodel xmlns="http://www.cisco.com/esc/esc">
   <tenants>
      <tenant>
         <name>aws-east-1-tenant</name>
         <vim_mapping>false</vim_mapping>
         <deployments>
            <deployment>
               <name>aws-east-1-dep</name>
               <vm_group>
                  <name>aws-vm-east-1</name>
                  <locator>
                     <vim_id>AWS_US_EAST_1</vim_id>
                  </locator>
                  <bootup_time>600</bootup_time>
                  <recovery_wait_time>33</recovery_wait_time>
                  <flavor>t2.micro</flavor>
                  <image>ami-c7bfa6bd</image>
                  <extensions>
                     <extension>
                        <name>AWS_PARAMS</name>
                        <properties>
                           <property>
                              <name>key_pair_name</name>
                              <value>esc-us-east-1</value>
                           </property>
                        </properties>
                     </extension>
                  </extensions>
                  <interfaces>
                     <interface>
                        <nicid>0</nicid>
                        <network>vpc-d7ee1bac</network>
                        <security_groups>
                           <security_group>esc-sg-us-east-1</security_group>
                        </security_groups>
                     </interface>
                  </interfaces>
                  <kpi_data>
                     <kpi>
                        <event_name>VM_ALIVE</event_name>
                        <metric_value>1</metric_value>
                        <metric_cond>GT</metric_cond>
                        <metric_type>UINT32</metric_type>
                        <metric_collector>
                           <type>ICMPPing</type>
                           <nicid>0</nicid>
                           <poll_frequency>3</poll_frequency>
                           <polling_unit>seconds</polling_unit>
                           <continuous_alarm>false</continuous_alarm>
                           <monitoring_public_ip>true</monitoring_public_ip>
                        </metric_collector>
                     </kpi>
                  </kpi_data>
```

```
                  <rules>
                     <admin_rules>
                        <rule>
                           <event_name>VM_ALIVE</event_name>
                           <action>ALWAYS log</action>
                           <action>FALSE recover autohealing</action>
                           <action>TRUE servicebooted.sh</action>
                        </rule>
                     </admin_rules>
                  </rules>
                  <config_data />
                  <scaling>
                     <min_active>1</min_active>
                     <max_active>1</max_active>
                     <elastic>true</elastic>
                  </scaling>
               </vm_group>
            </deployment>
         </deployments>
      </tenant>
   </tenants>
</esc_datamodel>
```

CHAPTER 18

Deploying VNFs Using ESC on CSP Cluster

Deploying VNFs Using ESC on CSP Cluster

The VNF deployment is initiated as a service request either originating from the ESC portal or the northbound interfaces. The service request comprises of XML payloads.

You can deploy VNF on CSP with the disk storage name as (glusterFS). By default, the disk storage is local.

You need disk_storage_name as gluster under image extension properties. Use the cluster VIM connector to do the initial deployment.

Following example shows how to add gluster as disk_storage_name under image extension properties:

```
deploy_csp_1.xml
<esc_datamodel xmlns="http://www.cisco.com/esc/esc">
              <extension>
                <name>image</name>
                <properties>
                  <property>
                    <name>disk-resize</name>
                    <value>true</value>
                  </property>
                  <property>
                    <name>disk_type</name>
                    <value>virtio</value>
                  </property>
                  <property>
                    <name>disk_storage_name</name>
                    <value>gluster</value>
                  </property>
                </properties>
              </extension>
```

CHAPTER 19

Unified Deployment

Unified Deployment

ESC creates OpenStack resources such as tenants, networks, and subnetworks before deploying a VNF.

During unified deployment, you send a single combined request to create or delete the OpenStack resources, and deploy a VNF. You can create multiple networks and subnetworks, but can create only a single VNF and a single tenant using unified deployment.

A unified deployment request is defined as a new deployment request, and any number of networks and subnetworks located directly inside the deployment definition. Networks and subnets located directly inside the tenant are not considered part of a unified deployment request, and will not be removed during a subsequent undeploy request.

Update the `deployment data model` and the files with the necessary information such as the service and deployment ID, tenant, network and subnetwork ids and so on. You can either use NETCONF or REST APIs. For example, send POST REST and DELETE REST calls.

Note A single NETCONF request can be used to perform multiple actions, such as creating networks and subnetworks; creating images, flavors and deploying VNFs.

See the Elastic Services Controller Deployment Attributes for a list of deployment attributes.

- To create a deployment datamodel with a single deployment request, send POST REST call to:

  `http://[ESC_IP]:8080/v0/deployments/[internal_dep_id]`

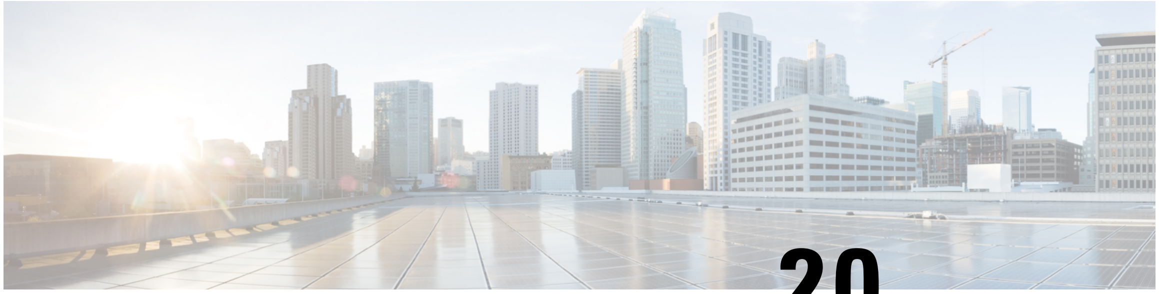
- To delete a single deployment request, send DELETE REST call to:

  `http://[ESC_IP]:8080/v0/deployments/[internal_dep_id]`

The VNF will be undeployed, and the network and subnet will be deleted in the specified order.

Note If tenant creation fails as part of a unified deployment request, a manual rollback is needed to clean up ESC. As part of manual rollback, first an undeploy request is required to clean up the deployment, followed by a delete tenant request to clean up the failed tenant creation.

During an undeploy request, any network and subnetwork created as part of the unified deployment request will be deleted along with the VNF. However, the tenant created through unified deployment request will not be deleted.



CHAPTER 20

Undeploying Virtual Network Functions

Undeploying Virtual Network Functions

You can undeploy an already deployed VNF. Use the REST or NETCONF / YANG APIs to undeploy the VNF.

Important You can also undeploy VNFs using the ESC portal. For more information, see ESC Portal Dashboard.

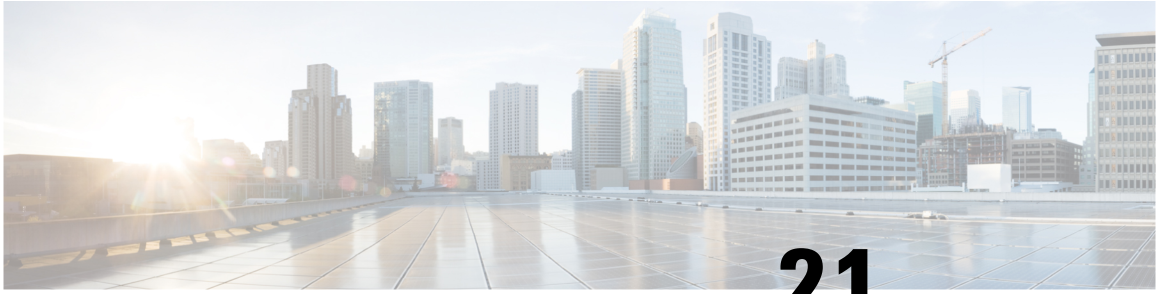
Sample undeploy request is as follows:

```
DELETE /v0/deployments/567 HTTP/1.1
Host: client.host.com
Content-Type: application/xml
Accept: application/xml
Client-Transaction-Id: 123456
Callback:/undeployservicecallback
```

For more details, see Cisco Elastic Services Controller API Guides.

Reboot Parameter

A reboot time parameter is introduced in the deployment request. This provides more flexibility to the operation time of the deployment. In a deployment, when the VM reboots, the monitor is set with the reboot time. If the reboot time expires before the VM alive event, the next action such as vm_recovery_complete, or undeploy is performed.



CHAPTER 21

Configuring Deployment Parameters

Deployment Parameters

A VNF deployment is initiated as a service request through the northbound interface or the ESC portal. The service request comprises of templates that consist of XML payloads and deployment parameters. Deployment parameters are rules, policies or day 0 configuration that determine properties of the VNF and its lifecycle. The table below lists the complete list of deployment parameters and how they interoperate on OpenStack or VMware vCenter:

Deployment Parameters	OpenStack	VMware vCenter	VMware vCloud Director
Day 0 Configuration	Day 0 configuration is done in one of the following ways: • NETCONF API • REST API • ESC Portal	Day 0 configuration is done in one of the following ways: • NETCONF API • REST API • ESC Portal	• NETCONF API • REST API • ETSI API
Deploying VNFs	Configuration of Individual and Composite VNFs is done in one of the following ways: • NETCONF API • REST API • ESC Portal (You can deploy using the Deployment Template.)	Configuration of Individual and Composite VNFs is done in one of the following ways: • NETCONF API • REST API • ESC Portal (You can configure the VNF settings through the Deployment Form, or the Deployment Template.)	Configuration of Individual and Composite VNFs is done in one of the following ways: • NETCONF API • REST API • ETSI API

Deployment Parameters	OpenStack	VMware vCenter	VMware vCloud Director
Undeploy Virtual Network Functions	Undeploying is done in one of the following ways: • NETCONF API • REST API • ESC Portal	Undeploying VNFs is done in one of the following ways: • NETCONF API • REST API • ESC Portal	Undeploying VNFs is done in one of the following ways: • NETCONF API • REST API • ETSI API
Affinity and anti-affinity Rule	Creating and deleting affinity and anti-affinity rule definitions is done in one of the following ways: • NETCONF API • REST API	Creating and deleting affinity rule definition in one of the following ways: • NETCONF API • REST API • ESC Portal (You can set up affinity and anti-affinity using the Deployment Form.)	Creating and deleting affinity and anti-affinity rule definitions is done in one of the following ways: • NETCONF API • REST API • ETSI API
VNF Operations	VNF Operations are done in one of the following ways: • REST API • NETCONF API • ESC Portal	VNF Operations are done in one of the following ways: • REST API • NETCONF API • ESC Portal For more information, see the Elastic Services Controller Portal, on page 13.	VNF Operations are done in one of the following ways: • REST API • NETCONF API • ETSI API
Multi Cluster	Not applicable	Multi Cluster configuration is done in one of the following ways: • REST API • ESC Portal For more information, see the Deploying VNFs on VMware vCenter using ESC Portal.	Not applicable
Multiple Virtual Datacenter (Multi VDC)	Not applicable	Multiple Virtual Datacenter selection is done in one of the following ways: • REST API • ESC Portal	Not applicable

Deployment Parameters	OpenStack	VMware vCenter	VMware vCloud Director
Hardware Acceleration	Hardware Acceleration is supported in one of the following ways: • NETCONF API • REST API For more information, see the Hardware Acceleration Support (OpenStack Only) in the Cisco Elastic Services Controller Administration Guide.	Not applicable	Not applicable
Single Root I/O Virtualization	Configuration of Single Root I/O Virtualization is done in one of the following ways: • NETCONF API • REST API	Configuration of Single Root I/O Virtualization is done in one of the following ways: • NETCONF API • REST API	Not applicable

This chapter describes the procedures to configure the deployment customization. For more information on VNF deployment, see Deploying Virtual Network Functions on OpenStack, on page 107.

CHAPTER 22

Day Zero Configuration

Day Zero Configuration

The initial or day 0 configuration of a VNF is based on the VM type. A VNF administrator configures the initial template for each VM type at the time of VNF deployment. The same configuration template is applied to all deployed and new VMs of that VM type. The template is processed at the time of individual VM deployment. The day 0 configuration continues to persists, so that all initial deployment, healing and scaling of VMs have the same day 0 template.

Some of the day 0 configuration tasks include bringing up the interface, managing the network, support for static or dynamic IP (DHCP, IPAM), SSH keys, and NetConf enabled configuration support on VNF.

Note ESC does not support day 0 configuration of interfaces added during service update. In case of recovery for day 0 configuration, all the interfaces with Network Interface Card IDs will be configured.

Day Zero in the Configuration Data Model

The day 0 configuration file can be specified in different ways in the data model, but you can use only one of the options at a time.

- <file> url </file>—The url specifies a file on the ESC VM file system or file hosted on report http server. ESC downloads the file specified by the URL. This file is used as a template to replace the tokens specified in this template with the values specified in the variables section. This template is used to generate the day 0 configuration.
- <data> inline config content </data>—Specifies URL for the template. This allows the use of inline text as the template.
- <encrypted_data> inline config content</encrypted_data>—The inline configuration content will be encrypted based on the data.

- <file_locators> list of file locators </file_locators>—Similar to file, a file_locator defines file to download from a remote server with basic authentication (if required).

Note The <file_locators> is deprecated in ESC Release 4.0.

- <file_locator_name> deployment defined file_locator </file_locator_name>—Similar to file, the file_locator_name is used to download the file from a remote server with basic authentication (if required).

Day 0 configuration is defined in the datamodel under the config_data tag. Each user data and the configuration drive file is defined under the configuration tag. The contents are in the form of a template. ESC processes the template through the Apache Velocity Template Engine before passing to the VM.

The config_data tag is defined for each vm_group. The same configuration template is applied to all VMs in the vm_group. The template file is retrieved and stored at deployment initialization. Template processing is applied at time of VM deployment. The content of the config file can be retrieved from the file or data.

```
<file> url </file>
<data> inline config content </data>
```

A destination name is assigned to the config by <dst>. User Data is a treated as a special case with <dst>--user-data</dst>.

A sample config data model,

```
<config_data>
    <configuration>
        <file>file://cisco/userdata_file.txt</file>
        <dst>--user-data</dst>
        <variable>
            <name>CUSTOM_VARIABLE_FOR_USERDATA</name>
            <val>SOME_VALUE_XXX</val>
        </variable>
    </configuration>
    <configuration>
        <file>file://cisco/config.sh</file>
        <dst>config.sh</dst>
        <variable>
            <name>CUSTOM_VARIABLE_FOR_CONFIG</name>
            <val>SOME_VALUE_XXX</val>
        </variable>
    </configuration>
</config_data>
```

Custom variable can be specified in the variables tag within the configuration. Zero or more variables can be included in each configuration. Each variable can have multiple values. Multiple values are only useful when creating more than one VM per vm_group. Also, when performing scale in and scale out, additional VMs can be added and removed from the VM group.

Note Note the following while providing multiple values for the variable tag.

- The variable values assigned to the initially deployed VMs are unique and from the pool. There is no order followed for assigning the values from the pool. That is, the first VM can use the second value from the pool.
- A scaled out VM should have a unique variable value and from the pool.

- A recovered VM (after undeploy or redeploy) must retain the same value it had before.

The contents of <file> are a template that is processed by the Velocity Template Engine. ESC populates a set of variables for each interface before processing the configuration template:

NICID_n_a_IP_ALLOCATION_TYPE	string containing FIXED | DHCP
NICID_n_a_NETWORK_ID	string containing neutron network uuid
NICID_n_a_IP_ADDRESS	ipv4 or ipv6 address
NICID_n_a_MAC_ADDRESS	string
NICID_n_a_GATEWAY	ipv4 or ipv6 gateway address
NICID_n_a_CIDR_ADDRESS	ipv4 or ipv6 cidr prefix address
NICID_n_a_CIDR_PREFIX	integer with prefix-length
NICID_n_a_NETMASK	If an ipv4 CIDR address and prefix are present, ESC will automatically calculate and populate the netmask variable. This is not substituted in the case of an IPv6 address and should not be used.
NICID_n_a_ANYCAST_ADDRESS	string with ipv4 or ipv6
NICID_n_a_IPV4_OCTETS	string with last 2 octets of ip address, such as 16.66, specific to CloudVPN

Where n is the interface number from the data model, for example, 0, 1, 2, 3

Note The interface number, n starts with 0 for OpenStack, and 1 for VMware.

Example

```
NICID_0_NETWORK_ID=0affdc19-60fd-4a4f-a02b-f062d7a66c27
NICID_0_MAC_ADDRESS=fa:16:3e:4d:c5:f8

NICID_0_0_IP_ALLOCATION_TYPE=DHCP
NICID_0_0_IP_ADDRESS=1.1.22.133
NICID_0_0_GATEWAY=1.1.0.1
NICID_0_0_CIDR_ADDRESS=1.1.0.0
NICID_0_0_CIDR_PREFIX=16
NICID_0_0_NETMASK=255.255.0.0

NICID_0_1_IP_ALLOCATION_TYPE=DHCP
NICID_0_1_IP_ADDRESS=fd04:1::a03
NICID_0_1_GATEWAY=fd04:1::1
NICID_0_1_CIDR_ADDRESS=fd04:1::/64
NICID_0_1_CIDR_PREFIX=64
```

By default, ESC substitutes the $ variable in the day 0 configuration file with the actual value during deployment. You can enable or disable the $ variable substitution for each configuration file.

Add the following field to the configuration data model:

```
<template_engine>VELOCITY | NONE</template_engine> field to configuration
```

where,

- VELOCITY enables variable substitution.
- NONE disables variable substitution.

If no value is set the default option is VELOCITY, and the $ variable substitution takes place. When set to NONE, the $ variable substitution does not take place.

You must follow these tips while processing the template through the velocity template engine.

- To escape dollar sign in the template insert,

  ```
  #set ( $DS = "$" )
  ```

  then replace the variable with

  ```
  passwd: ${DS}1${DS}h1VxC40U${DS}uf2qLUwGTjHgZplkP78xA
  ```

- To escape a block in the template, insert #[[and #]]. For example,

  ```
  #[[ passwd: $1$h1VxC40U$uf2qLUwGTjHgZplkP78xA ]]#
  ```

File Locator

To fetch external configuration files, a file locator is added to the day 0 configuration. The file locator contains a reference to the file server, and the relative path to the file to be downloaded.

Note The file locator attribute is defined at the deployment level, that is, directly under the deployment container instead of policy actions and day 0 configuration sections. For updated data model see Fetching Files From Remote Server.

Example of day 0 configuration with a file locator:

```
<?xml version="1.0" encoding="UTF-8"?>
<esc_datamodel xmlns="http://www.cisco.com/esc/esc">
  <tenants>
    <tenant>
      <name>sample-tenant</name>
      <deployments>
        <deployment>
          <name>sample-deployment</name>
          <vm_group>
            <name>sample-vm-group</name>
            <config_data>
              <!-- exisiting configuration example - remains valid -->
              <configuration>
                <file>file:///cisco/config.sh</file>
                <dst>config.sh</dst>
              </configuration>
              <!-- new configuration including use of file locators -->
              <configuration>
                <dst>ASA_config_0</dst>
                <file_locators>
                  <file_locator>
                    <name>configlocator-1</name>
                    <!-- unique name -->
                    <remote_file>
```

```
                      <file_server_id>server-1</file_server_id>
                      <remote_path>/share/users/configureScript.sh</remote_path>
                      <!-- optional user specified local silo directory -->
                      <local_target>day0/configureScript.sh</local_target>
                      <!-- persistence is an optional parameter -->
                      <persistence>FETCH_ALWAYS</persistence>
                      <!-- properties in the file_locator are only used for
                           fetching the file not for running scripts -->
                      <properties>
                        <property>
                          <!-- the property name "configuration_file" with value "true"
indictates this is the
                               script to be used just as using the <file> member case of
the configuration -->
                          <name>configuration_file</name>
                          <value>true</value>
                        </property>
                        <property>
                          <name>server_timeout</name>
                          <value>120</value>
                         <!-- timeout value in seconds, overrides the file_server property
 -->
                        </property>
                      </properties>
                    </remote_file>
                    <!-- checksum is an optional parameter.
                         The following algorithms are supported: SHA-1, SHA-224, SHA-256,
SHA-384, SHA-512 -->
                    <checksum>SHA256 (configureScript.sh) =
dd526bb2c0711238ec2649c4b91598fb9a6cf1d2cb8559c337c5f3dd5ea1769e</checksum>
                  </file_locator>
                  <file_locator>
                    <name>configlocator-2</name>
                    <remote_file>
                      <file_server_id>server-2</file_server_id>
                      <remote_path>/secure/requiredData.txt</remote_path>
                      <local_target>day0/requiredData.txt</local_target>
                      <persistence>FETCH_ALWAYS</persistence>
                      <properties />
                    </remote_file>
                  </file_locator>
                </file_locators>
              </configuration>
            </config_data>
          </vm_group>
        </deployment>
      </deployments>
    </tenant>
  </tenants>
</esc_datamodel>
```

The file locator parameters include:

- name—used as the key and identifier for a file locator.
- local_file or remote_file—choice of file location. Local file is used to specify a file existing on the ESC VM file system already. The remote_file is used to specify a file to fetch from a remote server.
  - file_server_id—id of the File Server object to fetch the file from.
  - remote_path—path of the file from the base_url defined in the file server object.
  - local_target—optional local relative directory to save the file.

  - properties—name-value pairs of of information that may be required.
  - persistence—options for file storage. Values include CACHE, FETCH_ALWAYS and FETCH_MISSING (default).
- checksum—optional BSD style checksum value to use to validate the transferred file's validity.

For more information, see Fetching Files From Remote Server.

To encrypt the files see, Encrypting Configuration Data.

Day 0 Configuration for vCD Deployment

The day 0 configuration for vCD deployment can be passed in different ways:

- Constructing an ISO file
- OVF properties
- Pre-existing ISO file in a catalog (OOB ISO file)

Note

- For initial deployment, the number of VM group(s) defined in the datamodel must be the same as the number of VM(s) in the vApp template. In a deployment, the image value of each VM group should be unique.
- The out of band (OOB) ISO file cannot be used along with constructing an ISO file method, as the VM can consider any one. The ovf property can be used with OOB ISO or constructing ISO together.

Day 0 configuration through constructing an ISO file:

```
</rules>
            <config_data>
              <!-- take content from the file path and save it as config.sh into the ISO
file -->
              <configuration>
                <dst>config.sh</dst>
                <file>file:///cisco/config.sh</file>
              </configuration>
             <!-- take content from the file path, replace variables with values, and save
 it as data/config.sh into the ISO file -->
              <configuration>
                <dst>data/params.cfg</dst>
                <file>file:///cisco/template.cfg</file>
                <variable>
                  <name>CF_VIP_ADDR</name>
                  <val>10.0.0.9</val>
                </variable>
                <variable>
                  <name>CF_DOMAIN_NAME</name>
                  <val>cisco.com</val>
                </variable>
                <variable>
                  <name>CF_NAME_SERVER</name>
                  <val>172.16.180.7</val>
```

```
                </variable>
              </configuration>
             <!-- take the data section as the content of the file, replace variables with
 values, and save it as user-data.txt into the ISO file-->
              <configuration>
                <dst>user-data.txt</dst>
                <data>#cloud-config
manage_etc_hosts: true
hostname: $HOST_NAME
local-hostname: $HOST_NAME
</data>
                <variable>
                  <name>$HOST_NAME</name>
                  <val>something.cisco.com</val>
                </variable>
              </configuration>
            </config_data>
```

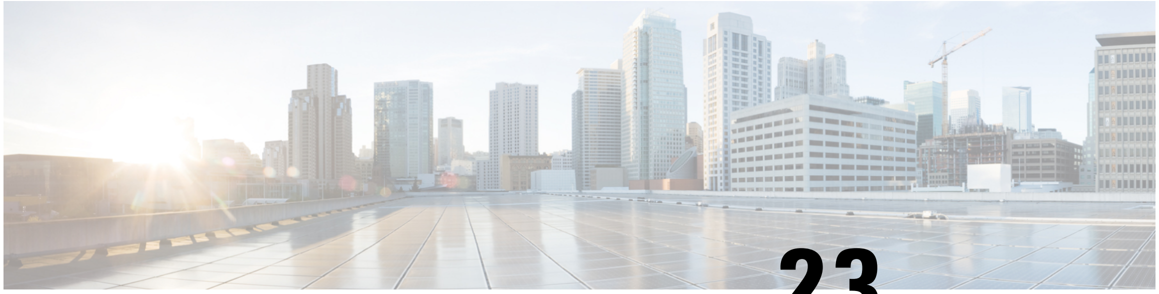
Day 0 configuration through OOB ISO file:

```
         </rules>
            <config_data>
              <configuration>
                <!-- ISO file stored in catalog-1 -->
                <dst>vcdCatalog:catalog-1</dst>
                <data>h2.iso</data>
              </configuration>
            </config_data>
```

Day 0 configuration through OVF properties:

```
           <config_data>
              <configuration>
                <!-- ovf properties as day0 -->
                <dst>ovfProperty:mgmt-ipv4-addr</dst>
                <data>$NICID_0_IP_ADDRESS/24</data>
              </configuration>
            </config_data>
```

For information on deploying VNFs on vCD, see Deploying Virtual Network Functions on VMware vCloud Director (vCD), on page 140.



CHAPTER 23

KPIs, Rules and Metrics

KPIs, Rules and Metrics

Cisco Elastic Services Controller VNF monitoring is done through the definition of Key Performance Indicators (KPIs) metrics. Core metrics are preloaded with ESC, a programmable interface gives to the end-user the ability to add and remove metrics, but also to define the actions to be triggered on specified conditions.These metrics and actions are defined at the time of deployment.

The ESC metrics and actions datamodel is divided into 2 sections:

1. KPI—Defines the type of monitoring, events, polling interval and other parameters. This includes the event_name, threshold and metric values. The event_name is user defined. The metric_values specify threshold conditions and other details. An event is triggered when the threshold condition is reached.
2. Rule—Defines the actions when the KPI monitoring events are triggered. The action element defines the actions to be performed when an event corresponding to the event_name is triggered.

Rules

The ESC object model defines for each vm_group a section where the end-user can specify the administrative rules to be applied based on the outcome of the KPIs selected metric collector.

```
<rules>
    <admin_rules>
        <rule>
            <event_name>VM_ALIVE</event_name>
            <action>TRUE esc_vm_alive_notification</action>
            <action>FALSE recover autohealing</action>
        </rule>
        : : : : : : : : : : : : : : : :
    </admin_rules>
</rules>
```

As mentioned within the KPIs section, correlation between KPIs and Rules is done based on the value of the <event_name> tag.

In the Rules section above, if the outcome of the KPIs defining event_name is VM_ALIVE, and the selected metric collector is TRUE, then the action identified by the key, TRUE esc_vm_alive_notification is selected for execution.

If the outcome of the KPIs defining event_name is VM_ALIVE, and the selected metric collector is FALSE, then the action identified by the key, FALSE recover autohealing is selected for execution.

For information on updating KPIs and Rules, see Updating the KPIs and Rules, on page 229.

Metrics and Actions

ESC Metrics and Actions (Dynamic Mapping) framework is the foundation of the kpis and rules sections. As described in the KPIs section the metric type uniquely identifies a metric and its metadata.

The metrics and actions is as follows:

```
<metrics>
    <metric>
        <name>ICMPPING</name>
        <userLabel>ICMP Ping</userLabel>
        <type>MONITOR_SUCCESS_FAILURE</type>
        <metaData>
            <type>icmp_ping</type>
            <properties>
                <property>
                    <name>ip_address</name>
                    <value />
                </property>
                <property>
                    <name>enable_events_after_success</name>
                    <value>true</value>
                </property>
                <property>
                    <name>vm_gateway_ip_address</name>
                    <value />
                </property>
                <property>
                    <name>enable_check_interface</name>
                    <value>true</value>
                </property>
            </properties>
        </metaData>
    </metric>
    : : : : : : : :
</metrics>
```

The above metric is identified by its unique name ICMPPING. The <type> tag identifies the metric type.

Currently ESC supports two types of metrics:

- MONITOR_SUCCESS_FAILURE
- MONITOR_THRESHOLD

The <metadata> section defines the attributes and properties that is processed by the monitoring engine.

The metric_collector type in the KPI show the following behavior:

At regular intervals of 3 seconds the behavior associated with the ICMPPING identifier is triggered. The ICMPPING metric is of type MONITOR_SUCCESS_FAILURE, that is the outcome of the monitoring action is either a success or a failure. In the sample above, an icmp_ping is performed using the <ip_address> field defined in the <metadata> section. In case of SUCCESS the rule action(s) with the TRUE prefix will be selected for execution. In case of FAILURE the rule action(s) with the FALSE prefix is selected for execution.

```
<actions>
    <action>
```

```
            <name>TRUE servicebooted.sh esc_vm_alive_notification</name>
            <type>ESC_POST_EVENT</type>
            <metaData>
                <type>esc_post_event</type>
                <properties>
                    <property>
                        <name>esc_url</name>
                        <value />
                    </property>
                    <property>
                        <name>vm_external_id</name>
                        <value />
                    </property>
                    <property>
                        <name>vm_name</name>
                        <value />
                    </property>
                    <property>
                        <name>event_name</name>
                        <value />
                    </property>
                    <property>
                        <name>esc_event</name>
                        <value>SERVICE_BOOTED</value>
                    </property>
                </properties>
            </metaData>
        </action>
        : : : : : : : :
    </actions>
```

The action sample above describes the behavior associated with the SUCCESS value. The ESC rule action name TRUE servicebooted.sh esc_vm_alive_notification specifies the action to be selected. Once selected the action <type> ESC_POST_EVENT identifies the action that the monitoring engine selects.

Metrics and Actions APIs

In Cisco ESC Release 2.1 and earlier, mapping the actions and metrics defined in the datamodel to the valid actions and metrics available in the monitoring agent was enabled using the *dynamic_mappings.xml* file. The file was stored in the ESC VM and was modified using a text editor. ESC 2.2 and later do not have an *esc-dynamic-mapping* directory and *dynamic_mappings.xml* file. However, if you have an existing dynamic_mapping xml file that you want to add to the ESC VM, do the following:

1. Backup this file to a location outside of ESC, such as, your home directory.

2. Create *esc-dynamic-mapping* directory on your ESC VM. Ensure that the read permissions are set.

3. Install on your ESC VM using the following bootvm argument:

```
 --file
root:root:/opt/cisco/esc/esc-dynamic-mapping/dynamic_mappings.xml:<path-to-local-copy-of-dynamic-mapping.xml>
```

The CRUD operations for mapping the actions and the metrics are available through REST API. Refer to the API tables below for mapped metrics and actions definition.

To update an existing mapping, delete and add a new mapping through the REST API.

Note While upgrading any earlier version of ESC to ESC 2.2 and later, to maintain the VNF monitoring rules, you must back up the *dynamic_mappings.xml* file and then restore the file in the upgraded ESC VM. For more information upgrading monitoring rules, see Upgrading VNF Monitoring Rules section in the *Cisco Elastic Services Controller Install and Upgrade Guide*. Cisco ESC Release 2.3.2 and later, the dynamic mapping API is accessible locally only on the ESC VM.

Table 7: Mapped Actions

User Operation	Path	HTTP Operation	Payload	Response	Description
Read	internal/dynamic_mapping/actions/ <action_name>	GET	N/A	Action XML	Get action by name
Read All	internal/dynamic_mapping/actions	GET	N/A	Action XML	Get all actions defined
Write	internal/dynamic_mapping/actions	POST	Actions XML	Expected Action XML	Create one or multiple actions
Delete	internal/dynamic_mapping/actions/ <action_name>	DELETE	N/A	N/A	Delete action by name
Clear All	internal/dynamic_mapping/actions	DELETE	N/A	N/A	Delete all non-core actions

The response for the actions APIs is as follows:

```
<actions>
    <action>
        <name>{action name}</name>
        <type>{action type}</type>
        <metaData>
            <type>{monitoring engine action type}</type>
            <properties>
                <property>
                    <name />
                    <value />
                </property>
                : : : : : : :
            </properties>
        </metaData>
    </action>
    : : : : : : : :
</actions>
```

Where,

{action name}: Unique identifier for the action. Note that in order to be compliant with the ESC object model, for success or failure actions, the name must start with either TRUE or FALSE.

{action type}: Action type in this current release can be either ESC_POST_EVENT, SCRIPT or CUSTOM_SCRIPT.

{monitoring engine action type}: The monitoring engine type are the following: icmp_ping, icmp4_ping, icmp6_ping, esc_post_event, script, custom_script, snmp_get. See Monitoring the VNFs for more details.

Core and Default Actions List

Table 8: Core and Default Actions List

Name	Type	Description
TRUE esc_vm_alive_notification	Core	Start Service
TRUE servicebooted.sh	Core/Legacy	Start Service
FALSE recover autohealing	Core	Recover Service
TRUE servicescaleup.sh	Core/Legacy	Scale Out
TRUE esc_vm_scale_out_notification	Core	Scale Out
TRUE servicescaledown.sh	Core/Legacy	Scale In
TRUE esc_vm_scale_in_notification	Core	Scale In
TRUE apply_netscaler_license.py	Default	Apply Netscaler License

The core actions and metrics are defined by ESC and cannot be removed or updated.

The default actions or metrics are defined by ESC and exist to supplement core actions or metrics for more complex monitoring capabilities. These can be deleted and modified by the user. The default actions or metrics are reloaded on ESC startup every time an action or a metric with the same name cannot be found in the database.

Metric APIs

Table 9: Mapped Metrics

User Operation	Path	HTTP Operation	Payload	Response	Description
Read	internal/dynamic_mapping/actions/ <metric_name>	GET	N/A	Metric XML	Get metrics by name
Read All	internal/dynamic_mapping/metrics/	GET	N/A	Metric XML	Get all metrics defined
Write	internal/dynamic_mapping/metrics/	POST	Metrics XML	Expected Metrics XML	Create one or multiple metrics
Delete	internal/dynamic_mapping/actions/ <metric_name>	DELETE	N/A	N/A	Delete metric by name
Clear All	internal/dynamic_mapping/metrics	DELETE	N/A	N/A	Delete all non-core metrics

The response for the Metric APIs is as follows:

```
<metrics>
    <metric>
        <name>{metric name}</name>
        <type>{metric type}</type>
        <metaData>
            <type>{monitoring engine action type}</type>
            <properties>
                <property>
                    <name />
                    <value />
                </property>
                : : : : : : :
            </properties>
        </metaData>
    </metric>
    : : : : : : : :
</metrics>
```

Where,

{metric name}: Unique identifier for the metric.

{metric type}: Metric type can be either MONITOR_SUCCESS_FAILURE, MONITOR_THRESHOLD or MONITOR_THRESHOLD_COMPUTE.

{monitoring engine action type}: The monitoring engine type are the following: icmp_ping, icmp4_ping, icmp6_ping, esc_post_event, script, custom_script, snmp_get. See Monitoring for more details.

Core and Default Metrics List

Table 10: Core and Default Metrics List

Name	Type	Description
ICMPPING	Core	ICMP Ping
MEMORY	Default	Memory compute percent usage
CPU	Default	CPU compute percent usage
CPU_LOAD_1	Default	CPU 1 Minute Average Load
CPU_LOAD_5	Default	CPU 5 Minutes Average Load
CPU_LOAD_15	Default	CPU 15 Minutes Average Load
PROCESSING_LOAD	Default	CSR Processing Load
OUTPUT_TOTAL_BIT_RATE	Default	CSR Total Bit Rate
SUBSCRIBER_SESSION	Default	CSR Subscriber Session

ESC Service Deployment

The KPI section defines the new KPI using the monitoring metrics.

```
<kpi>
    <event_name>DEMO_SCRIPT_SCALE_OUT</event_name>
```

```
    <metric_value>20</metric_value>
    <metric_cond>GT</metric_cond>
    <metric_type>UINT32</metric_type>
    <metric_collector>
        <type>custom_script_count_sessions</type>
        <nicid>0</nicid>
        <poll_frequency>15</poll_frequency>
        <polling_unit>seconds</polling_unit>
        <continuous_alarm>false</continuous_alarm>
    </metric_collector>
</kpi>
<kpi>
    <event_name>DEMO_SCRIPT_SCALE_IN</event_name>
    <metric_value>1</metric_value>
    <metric_cond>LT</metric_cond>
    <metric_type>UINT32</metric_type>
    <metric_occurrences_true>1</metric_occurrences_true>
    <metric_occurrences_false>1</metric_occurrences_false>
    <metric_collector>
        <type>custom_script_count_sessions</type>
        <nicid>0</nicid>
        <poll_frequency>15</poll_frequency>
        <polling_unit>seconds</polling_unit>
        <continuous_alarm>false</continuous_alarm>
    </metric_collector>
</kpi>
```

In the above sample, in the first KPI section, the metric identified by *custom_script_count_sessions* is executed at regular interval of 15 seconds. If the value returned by the metric is greater than 20, then the event name DEMO_SCRIPT_SCALE_OUT is triggered to be processed by the rules section.

In the above sample, in the second KPI section,The metric identified by custom_script_count_sessions is executed at regular interval of 15 seconds. If the value returned by the metric is less than 1, then the event name DEMO_SCRIPT_SCALE_IN is triggered to be processed by the rules section.

The rules section defines rules using the event_name that have been used by kpis. The action tag will define an action that will be executed when the event_name is triggered. In the example below, the action identified by the TRUE ScaleOut identifier is executed when the event DEMO_SCRIPT_SCALE_OUT is triggered.

```
<rule>
    <event_name>DEMO_SCRIPT_SCALE_OUT</event_name>
    <action>ALWAYS log</action>
    <action>TRUE ScaleOut</action>
</rule>
<rule>
    <event_name>DEMO_SCRIPT_SCALE_IN</event_name>
    <action>ALWAYS log</action>
    <action>TRUE ScaleIn</action>
</rule>
```

Script Actions

There are two types of actions supported:

1. Pre-Defined actions
2. Script actions

You can specify script execution as part of the Policy-driven data model. The *script_filename* property is mandatory to script actions, which specifies the absolute path to the script on the ESC VM. The following XML snippet shows a working example of a script action:

```
<action>
    <name>GEN_VPC_CHASSIS_ID</name>
    <type>SCRIPT</type>
    <properties>
        <property>
            <name>script_filename</name>
            <value>/opt/cisco/esc/esc-scripts/esc_vpc_chassis_id.py</value>
        </property>
        <property>
            <name>CHASSIS_KEY</name>
            <value>164c03a0-eebb-44a8-87fa-20c791c0aa6d</value>
        </property>
    </properties>
</action>
```

The script timeout is 15 minutes by default. However, you can specify a different timeout value for each script by adding a *wait_max_timeout* property to the properties section. The following example shows how to set the timeout to 5 minutes only for this script:

```
<action>
    <name>GEN_VPC_CHASSIS_ID</name>
    <type>SCRIPT</type>
    <properties>
        <property>
            <name>script_filename</name>
            <value>/opt/cisco/esc/esc-scripts/esc_vpc_chassis_id.py</value>
        </property>
        <property>
            <name>CHASSIS_KEY</name>
            <value>164c03a0-eebb-44a8-87fa-20c791c0aa6d</value>
        </property>
        <property>
            <name>wait_max_timeout</name>
            <value>300</value>
        </property>
    </properties>
</action>
```

In the above example, GEN_VPC_CHASSIS_ID will have a timeout value of 300 seconds, i.e. 5 mins. ESC also has a global parameter specifying the default timeout time for all the scripts that are being executed, called SCRIPT_TIMEOUT_SEC in the MONA category. This serves as the default value unless a wait_max_timeout property is defined in the script.

Triggering Pre-defined Actions

ESC introduces a new REST API to trigger the existing (pre-defined) actions defined through the Dynamic Mapping API, when required. For more information on the Metrics and Actions APIs, see Metrics and Actions APIs, on page 171.

A sample predefined action is as follows:

```
<actions>
    <action>
        <name>SaidDoIt</name>
        <userlabel>My Friendly Action</userlabel>
        <type>SCRIPT</type>
        <metaData>
            <type>script</type>
            <properties>
```

```
                <property>
                    <name>script_filename</name>
                    <value>/opt/cisco/esc/esc-scripts/do_somethin.py</value>
                </property>
                <property>
                    <name>arg1</name>
                    <value>some_val</value>
                </property>
                <property>
                    <name>notification</name>
                    <value>true</value>
                </property>
            </properties>
        </metaData>
    </action>
</actions>
```

Note A script file located on a remote server is also supported. You must provide the details in the <value> tag, for example,

`http://myremoteserverIP:80/file_store/do_somethin.py</value>http://myremoteserverIP:80/file_store/do_somethin.py</value>`

The pre-defined action mentioned above is triggered using the trigger API.

Execute the following HTTP or HTTPS POST operation:

POST `http://<IP_ADDRESS>:8080/ESCManager/v0/trigger/action/`

POST `https://<IP_ADDRESS>:8443/ESCManager/v0/trigger/action/`

The following payload shows the actions triggered by the API, and the response received:

```
<triggerTarget>
  <action>SaidDoIt</action>
  <properties>
   <property>
     <name>arg1</name>
     <value>real_value</value>
   </property>
  </properties>
</triggerTarget>
```

The response,

```
<triggerResponse>
    <handle>c11be5b6-f0cc-47ff-97b4-a73cce3363a5</handle>
    <message>Action : 'SAIDDOIT' triggered</message>
</triggerResponse>
```

ESC accepts the request, and returns a response payload and status code.

An http status code of 200 indicates that the action triggered exists, and is triggered successfully. An http status codes of 400 or 404 indicate that the action to be triggered is not found.

You can determine the status using the custom script notifications sent to NB at various lifecycle stages.

ESC sends the MANUAL_TRIGGERED_ACTION_UPDATE callback event to NB with a status message that describes the success or failure of the action execution.

The notification is as follows:

```
<esc_event xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
```

```
        <event_type>MANUAL_TRIGGERED_ACTION_UPDATE</event_type>
        <properties>
            <property>
                <name>handle</name>
                <value>c11be5b6-f0cc-47ff-97b4-a73cce3363a5</value>
            </property>
            <property>
                <name>message</name>
                <value>Action execution success</value>
            </property>
            <property>
                <name>exit_code</name>
                <value>0</value>
            </property>
            <property>
                <name>action_name</name>
                <value>SAIDDOIT</value>
            </property>
        </properties>
    </esc_event>
```

Note The script_filename property cannot be overwritten by the trigger API request. The trigger API must not contain any additional properties that do not exist in the predefined action.

The new API allows to overrides some of the special properties (of the actions) listed below:

- Notification—Set this if your script generates progress notifications at run time. The default value is false. This value can be set to true in the action or trigger payload.
- wait_max_timeout—Wait for the script to complete the execution before terminating. The default wait timeout is 900 seconds.

Note

- The trigger API supports only script type actions.
- Ensure that the script action located on the ESC VM is copied to the same path on both the active and standby HA instances. For more information, see the High Availability chapter in the *Cisco Elastic Services Controller Install and Upgrade Guide*.
- The script execution terminates if there is a failover, shutdown, or reboot of the ESC services.

Configuring Custom Script Metric Monitoring KPIs and Rules

Custom Script Metric Monitoring can be performed as follows:

1. Create Script
2. Add Metric
3. Add Action
4. Define Deployment
5. Update KPI data or Rules

6. Authenticating Remote Server Using KPIs and Rules

The script to be executed has to be compliant with the rules specified for a MONITOR_THRESHOLD action. Threshold crossing evaluation will be based on the exit value from the script execution. In the sample script below, the return value is the number of IP sessions.

```
#!/usr/bin/env python
import pexpect
import re
import sys
ssh_newkey = 'Are you sure you want to continue connecting'
# Functions
def get_value(key):
    i = 0
    for arg in sys.argv:
        i = i + 1
        if arg == key:
            return sys.argv[i]
    return None
def get_ip_addr():
    device_ip = get_value("vm_ip_address")
    return device_ip
# Main
CSR_IP = get_ip_addr()

p=pexpect.spawn('ssh admin@' + CSR_IP + ' show ip nat translations total')
i=p.expect([ssh_newkey,'assword:',pexpect.EOF])
if i==0:
    p.sendline('yes')
    i=p.expect([ssh_newkey,'assword:',pexpect.EOF])
if i==1:
    p.sendline("admin")
    p.expect(pexpect.EOF)
elif i==2:
    pass
n = p.before
result = re.findall(r'\d+', n)[0]
sys.exit(int(result))
```

The ESC monitoring and action engine processes the script exit value.

The script has to be installed into the following ESC VM directory: `/opt/cisco/esc/esc-scripts/`

The following payload describes a metric using a custom_script defined in the script

```
<!-- Demo Metric Counting Sessions -->
<metrics>
    <metric>
        <name>custom_script_count_sessions</name>
        <type>MONITOR_THRESHOLD</type>
        <metaData>
            <properties>
                <property>
                    <name>script_filename</name>
                    <value>/cisco/esc-scripts/countSessions.py</value>
                </property>
                <property>
                    <name>for_threshold</name>
                    <value>true</value>
                </property>
            </properties>
```

```
            <type>custom_script_threshold</type>
        </metaData>
    </metric>
</metrics>
<!-- -->
```

The metric payload has to be added to the list of supported ESC metrics by using the Mapping APIs.

Execute a HTTP POST operation on the following URI:

http://<my_esc_ip>:8080/ESCManager/internal/dynamic_mapping/metrics

The following payload describes custom actions that can be added to the list of supported ESC actions by using the Mapping APIs.

```
<actions>
    <action>
        <name>TRUE ScaleOut</name>
        <type>ESC_POST_EVENT</type>
        <metaData>
            <type>esc_post_event</type>
            <properties>
                <property>
                    <name>esc_url</name>
                    <value />
                </property>
                <property>
                    <name>vm_external_id</name>
                    <value />
                </property>
                <property>
                    <name>vm_name</name>
                    <value />
                </property>
                <property>
                    <name>event_name</name>
                    <value />
                </property>
                <property>
                    <name>esc_event</name>
                    <value>VM_SCALE_Out</value>
                </property>
                <property>
                    <name>esc_config_data</name>
                    <value />
                </property>
                <properties />
            </properties>
        </metaData>
    </action>
    <action>
        <name>TRUE ScaleIn</name>
        <type>ESC_POST_EVENT</type>
        <metaData>
            <type>esc_post_event</type>
            <properties>
                <property>
                    <name>esc_url</name>
                    <value />
                </property>
                <property>
                    <name>vm_external_id</name>
                    <value />
                </property>
```

```
                <property>
                    <name>vm_name</name>
                    <value />
                </property>
                <property>
                    <name>event_name</name>
                    <value />
                </property>
                <property>
                    <name>esc_event</name>
                    <value>VM_SCALE_IN</value>
                </property>
                <properties />
            </properties>
        </metaData>
    </action>
</actions>
```

Execute a HTTP POST operation on the following URI:

http://<IP_ADDRESS>:8080/ESCManager/internal/dynamic_mapping/actions

Custom Script Notification

ESC now supports sending notification to northbound about customized scripts run as part of the deployment at a certain lifecycle stage. You can also determine the progress of the script executed through this notification. To execute a custom script with notification, define action type attribute as *SCRIPT*, and property attribute name as *notification*, and set the value to true.

For example, in the datamodel below, the action is to run a customized script located at `/var/tmp/esc-scripts/senotification.py` with notification, when the deployment reaches POST_DEPLOY_ALIVE stage.

```
<policies>
    <policy>
        <name>PCRF_POST_DEPLOYMENT</name>
        <conditions>
            <condition>
                <name>LCS::POST_DEPLOY_ALIVE</name>
            </condition>
        </conditions>
        <actions>
            <action>
                <name>ANY_NAME</name>
                <type>SCRIPT</type>
                <properties>
                    <property>
                        <name>script_filename</name>
                        <value>/var/tmp/esc-scripts/senotification.py</value>
                    </property>
                    <property>
                        <name>notification</name>
                        <value>true</value>
                    </property>
                </properties>
            </action>
        </actions>
    </policy>
</policies>
```

You can notify northbound about the script execution progress using the following outputs:

- Standard JSON output

- REST API call
- NETCONF Notification

Standard JSON Output

The standard JSON output follows the MONA notification convention. MONA captures entries in this to generate notification.

```
{"esc-notification":{"items":{"properties":
[{"name":"name1","value":"value1"},{"name":"name2","value":"value2"}...]}}}
```

Table 11: Item list

Name	Description
type	Describes the type of notification. `progress_steps | progress_percentage | log | alert | error`
progress **Note** Progress item is required only when the type is progress-steps or progress-percentage.	For progress-steps type, `{current_step}|{total_steps}`
	For progress-percentage type, `{percentage}`
msg	Notification message.

Example JSON output is as follows:

```
{"esc-notification":{"items":{"properties": [{"name":"type",
"value":"progress_percentage"},{"name":"progress","value":"25"},{"name":"msg","value":"Installation
in progress."}]}}}
```

Note If the custom script is written in Python, because standard output is buffered by default, after each notification print statement, the script is required to call sys.stdout.flush() to flush the buffer (for pre Python 3.0). Otherwise MONA cannot process the script stdout in a real-time.

```
print
'{"esc-notification":{"items":{"properties": [{"name":"type",
"value":"progress_percentage"},{"name":"progress","value":"25"},{"name":"msg","value":"Installation
in progress."}]}}}'sys.stdout.flush()
```

REST API Call

```
http://localhost:8090/mona/v1/actions/notification
```

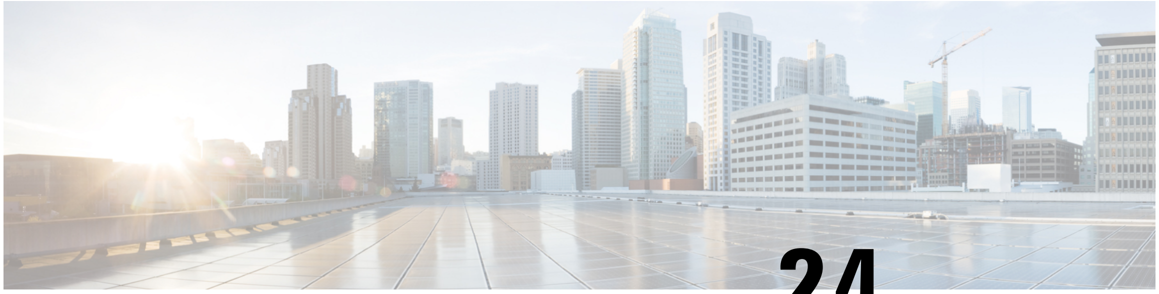
For REST API, the script must accept a script handle as the last parameter. The script handle can be UUID, MONA action or execution job Id. For example, if the script originally accepts 3 command line parameters, to support MONA notification, the script considers an additional parameter for the handle UUID. This helps MONA to identify the notification source. For every notification, the script is responsible for constructing a POST REST call to MONA's endpoint inside the script:

The payload is as follows:

```
{
 "esc-notification" : {
```

```
  "items" : {
   "properties" : [{
     "name" : "type",
     "value" : "log",
     "hidden" : false
    }, {
     "name" : "msg",
     "value" : "Log info",
     "hidden" : false
    }
   ]
  },
  "source" : {
   "action_handle" : "f82fe86d-6625-4b13-99f7-89d169e427ad"
  }
 }
}
```

Note The action_handle value is the handle UUID MONA passes into the script.



CHAPTER 24

Policy-Driven Data Model

Policy-Driven Data model

ESC supports a new policy-driven datamodel. A new <policy> section is introduced under <policies> at both deployment and VM group level.

Using the Policy Data model, a user can perform actions based on conditions. ESC supports predefined actions, or customized scripts during a deployment based on certain Lifecycle Stage (LCS). For example, the redeployment policy uses predefined actions based on lifecycle stages (LCS) to redeploy VMs. For more information, see Redeployment Policy, on page 342.

Policy Data model

The policy data model consists of conditions and actions. The condition is a Lifecycle Stage (LCS) in a deployment. The action is predefined or custom script.

- Predefined action—The action is predefined and executed when the condition is met.

  In the datamodel below, when condition2 is met, Action2 is performed. The action <type> is predefined.

- Custom Script—The action is a custom script, and executed when the condition is met.

  In the datamodel below, when condition1 is met, Action1-1 and Action 1-2 are executed. The action <type> is script.

```
<policies>
    <policy>
        <name>Name1</name>
        <conditions>
            <condition>
                <name>Condition1</name>
            </condition>
        </conditions>
        <actions>
            <action>
                <name>Action1-1</name>
                <type>SCRIPT</type>
            </action>
            <action>
                <name>Action1-2</name>
                <type>SCRIPT</type>
            </action>
```

```
            </actions>
        </policy>
        <policy>
            <name>Name2</name>
            <conditions>
                <condition>
                    <name>Condition2</name>
                </condition>
            </conditions>
            <actions>
                <action>
                    <name>Action2</name>
                    <type>PRE-DEFINED</type>
                </action>
            </actions>
        </policy>
    </policies>
```

For more information on Predefined actions, and scripts, see Recovery and Redeployment Policies, on page 339.

The table below shows the LCS in a deployment, and its description. The recovery and redeployment policies, and VNF software upgrade policies use the policy-driven data model. These policies are supported on both single deployment and multi VIM deployment. For more information, see "Deploying Virtual Network Functions". For details on configuring the recovery and redeployment policies using the policy framework, see Recovery and Redeployment Policies, on page 339. For details on upgrading the VNF software upgrade policies, see Upgrading VNF Software with Volume, on page 266.

CHAPTER 25

Supported Lifecycle Stages (LCS)

Supported Lifecycle Stages (LCS)

Table 12: Conditions and their Scope

Condition Name	Scope	Description
LCS::PRE_DEPLOY	Deployment	Occurs just before deploying VMs of the deployment.
LCS::POST_DEPLOY_ALIVE	Deployment	Occurs immediately after the deployment is active.
LCS::DEPLOY_ERR	Deployment	Occurs immediately after the deployment fails.
LCS::POST_DEPLOY:: VM_RECOVERY_ERR	Deployment	Occurs immediately after the recovery of one VM fails. (This is specified at deployment level and applies to all VM groups)
LCS::POST_DEPLOY:: VM_RECOVERY _REDEPLOY_ERR	Deployment	Occurs immediately after the redeployment of one VM fails. (This is specified at deployment level and applies to all VM groups)
LCS::DEPLOY_UPDATE::VM_ PRE_VOLUME_DETACH	Deployment	Triggered just before the ESC detaches a volume. (This is specified for a group of individual VMs and specified under <vm_group> rather than the entire deployment.)

LCS::DEPLOY_UPDATE:: VM_VOLUME_ATTACHED	Deployment	Triggered immediately after ESC has attached a new volume (This is specified for a group of individual VMs and specified under <vm_group> rather than the entire deployment.)
LCS::DEPLOY_UPDATE:: VM_SOFTWARE_VERSION_UPDATED	Deployment	Triggered immediately after ESC has updated the software version of the VM (This is specified for a group of individual VMs and specified under <vm_group> rather than the entire deployment.)

Fetching Files From Remote Server Using LCS Actions

Prior to ESc Release 4.0, a file locator is added to the LCS action scripts to fetch external configuration files. The file locator contains a reference to the file server, and the relative path to the file to be downloaded. Starting from ESC Release 4.0, the file locator attribute is defined at the deployment level, that is, directly under the deployment container instead of policy actions and day 0 configuration sections.

```
<esc_datamodel xmlns="http://www.cisco.com/esc/esc">
  <tenants>
    <tenant>
      <name>test-tenant</name>
      <deployments>
        <deployment>
          <name>test-deployment</name>
          <file_locators>
            <file_locator>
              <name>custom_bool_action</name>
              <remote_file>
                <file_server_id>http-my-server</file_server_id>
                <remote_path>share/qatest/custom_bool_action.sh</remote_path>
              </remote_file>
            </file_locator>
            <file_locator>
              <name>custom_bool_metric</name>
              <remote_file>
                <file_server_id>http-my-server</file_server_id>
                <remote_path>/share/qatest/custom_bool_metric.sh</remote_path>
              </remote_file>
            </file_locator>
          </file_locators>
          <!-- truncated for brevity -->
          <vm_group>
            <name>ASA-group</name>
            <!-- truncated for brevity -->
            <kpi_data>
              <kpi>
                <event_name>MY_CUSTOM_BOOL_ACTION</event_name>
                <metric_value>5</metric_value>
                <metric_cond>LT</metric_cond>
                <metric_type>UINT32</metric_type>
                <metric_occurrences_true>1</metric_occurrences_true>
                <metric_occurrences_false>1</metric_occurrences_false>
                <metric_collector>
                  <type>MY_CUSTOM_BOOL_METRIC</type>
                  <nicid>0</nicid>
                  <poll_frequency>3</poll_frequency>
                  <polling_unit>seconds</polling_unit>
```

```
                    <continuous_alarm>false</continuous_alarm>
                    <properties>
                      <!-- Add file locator reference here -->
                      <property>
                        <name>file_locator_name</name>
                        <value>custom_bool_action</value>
                      </property>
                    </properties>
                  </metric_collector>
                </kpi>
              </kpi_data>
              <rules>
                <admin_rules>
                  <rule>
                    <event_name>MY_CUSTOM_BOOL_ACTION</event_name>
                    <action>ALWAYS log</action>
                    <action>TRUE my_custom_bool_action</action>
                    <properties>
                      <!-- Add file locator reference here -->
                      <property>
                        <name>file_locator_name</name>
                        <value>custom_bool_action</value>
                      </property>
                    </properties>
                  </rule>
                </admin_rules>
              </rules>
            </vm_group>
          </deployment>
        </deployments>
      </tenant>
    </tenants>
</esc_datamodel>
```

See Fetching Files From Remote Server for more information.

To encrypt the files see, Authenticating External Configuration Files.

Lifecycle Stage (LCS) Policy Conditions Defined at Different Stages

The tables below shows all policy conditions defined in the data model.

Table 13: LifeCycle Stages

Condition Name	Scope
LCS::VM::PRE_VM_DEPLOY	VM
LCS::VM::POST_VM_DEPLOYED	VM
LCS::VM::POST_VM_ALIVE	VM
Lifecycle Stages in Deployment	
LCS::PRE_DEPLOY	VM / Deployment
LCS::DEPLOY:: POST_VM_DEPLOYED	VM

Condition Name	**Scope**
LCS::POST_DEPLOY_ALIVE	Deployment
LCS::DEPLOY_ERR	Deployment
Lifecycle Stages in Deployment Update	
LCS::DEPLOY_UPDATE::POST_VM_ALIVE	VM
LCS::DEPLOY_UPDATE::	VM
LCS::DEPLOY_UPDATE:: POST_VM_VOLUME_DETACHED	VM
LCS::DEPLOY_UPDATE:: POST_VM_VOLUME_ATTACHED	VM
LCS::DEPLOY_UPDATE:: PRE_VM_SOFTWARE_VERSION_UPDATED	VM
Lifecycle Stages in Recovery	
LCS::POST_DEPLOY:: POST_VM_RECOVERY_COMPLETE	VM
LCS::POST_DEPLOY:: VM_RECOVERY_ERR	VM
Lifecycle Stages in Recovery and Redeploy	
LCS::POST_DEPLOY:: VM_RECOVERY_REDEPLOY_ERR	VM

CHAPTER 26

Affinity and Anti-Affinity Rules

Affinity and Anti-Affinity Rules

Affinity and anti-affinity rules create relationship between virtual machines (VMs) and hosts. The rule can be applied to VMs, or a VM and a host. The rule either keeps the VMs and hosts together (affinity) or separated (anti-affinity).

Policies are applied during individual VM deployment. You can deploy a single VNF or multiple VNFs together through ESC portal by uploading an existing deployment datamodel or by creating a new deployment datamodel. For more information, see ESC Portal Dashboard.

Affinity and anti-affinity policy streamlines the deployment process.

Affinity and anti-affinity rules are created and applied on VMs at the time of deployment. VM receives the placement policies when the deploy workflow is initialized.

During a composite VNF deployment, if a couple of VMs need to communicate with each other constantly, they can be grouped together (affinity rule) and placed on the same host.

If two VMs are over-loading a network, they can be separated (anti-affinity rule) and placed on different hosts to balance the network.

Grouping or separating VMs and hosts at the time of deployment helps ESC to manage load across the VMs and hosts in the network. Recovery and scale out of these VMs do not impact the affinity and anti-affinity rules.

The anti-affinity rule can also be applied between VMs within the same group and on a different host. These VMs perform similar functions and support each other. When one host is down, the VM on the other host continues to run preventing any loss of service.

The table shows the types of affinity and anti-affinity policies in a deployment.

Table 14: Intra and Inter group affinity and anti-affinity policies

Policy	Policy	VM group	Host	Zone
affinity	Intra group affinity	same VM group	same host	same zone
	Inter group affinity	different VM group	same host	same zone

Policy	Policy	VM group	Host	Zone
anti-affinity	Intra group anti-affinity	same VM group	different host	same zone
	Inter group anti-affinity	different VM group	different host	same zone

Note If the zone is not specified on OpenStack, VMs will be placed on different hosts and different zones for inter and intra group anti-affinity rules.

CHAPTER 27

Affinity and Anti-Affinity Rules on OpenStack

Affinity and Anti-Affinity Rules on OpenStack

The following sections describe affinity and anti-affinity policies with examples.

Intra Group Affinity Policy

The VNFs within the same VM group can either be deployed on the same host, or into the same availability zone.

Example for Intra Group Affinity Policy:

```
<vm_group>
     <name>affinity-test-gp</name>
     <placement>
 <type>affinity</type>
 <enforcement>strict</enforcement>
     </placement>
...
```

The type *zone-host* is used to deploy VNFs in the same host, or into the same availability zone.

Zone or Host Based Placement

The VNFs are within the same VM group and deployed on the same host or the same available zone. The *host* tag is used to deploy VMs on the same host and the *zone* tag is used to deploy VMs in the same available zone. Before deploying, you need to make sure that the host exists in OpenStack. ESC validates the specified host on OpenStack. The *zone-host* tag specifies the type of placement. Hence, if a host or a zone is not specified during a deployment, the deployment fails.

Important You cannot specify both the host and zone tags to deploy VM on the same host or the same available zone.

Example for host placement:

```
<vm_group>
     <name>zone-host-test-gp1</name>
     <placement>
 <type>zone_host</type>
 <enforcement>strict</enforcement>
```

```
 <host>my-server</host>
     </placement>
...
```

Example for zone placement:

```
<vm_group>
     <name>zone-host-test-gp2</name>
     <placement>
 <type>zone_host</type>
 <enforcement>strict</enforcement>
        <zone>dt-zone</zone>
     </placement>
...
```

Intra Group Anti-Affinity Policy

The VNFs within the same VM group are explicitly deployed on different hosts. For example, back-up VNFs.

Example for Intra Group anti-affinity Policy:

```
<vm_group>
        <name>anti-affinity-test-gp</name>
        <placement>
            <type>anti_affinity</type>
            <enforcement>strict</enforcement>
        </placement>
...
```

Inter Group Affinity Policy

The VNFs in the same deployment but different VM groups can be explicitly deployed in the same host. For example VNF bundles. Multiple VM groups can follow this policy by adding the vm_group_ref tag and providing the VM group name as the value.

Note You can use one or more vm_group_ref tag, type tag and enforcement tag under the placement tag. The host or zone cannot be specified.

Example for Inter Group Affinity Policy:

```
<deployments>
            <deployment>
            <name>intergroup-affinity-dep</name>
             <policies>
              <placement>
                <target_vm_group_ref>affinity-test-gp1</target_vm_group_ref>
                <type>affinity</type>
                <vm_group_ref>affinity-test-gp2</vm_group_ref>
                <enforcement>strict</enforcement>
               </placement>
          </policies>
…
```

Inter Group Anti-Affinity Policy

The VNFs in the same deployment but different VM Groups can be explicitly deployed in different hosts. For example back-up VNFs or High-availability VNFs. Multiple VM groups can follow this policy by adding the vm_group_ref tag, and providing the VM group name as the value.

Note You can only use one <target_vm_group_ref> tag, type tag and enforcement tag under the placement tag. The host or zone cannot be specified.You can use multiple <vm_group_ref> tags, however the anti-affinity policy only applies between each <vm_group_ref> and their <target_vm_group_ref>, which means that 2 or more <vm_group_ref> can be deployed on the same host, as long as each of them are deployed on a different host from their <target_vm_group_ref> that is acceptable.

Example for Inter Group anti-affinity Policy:

```
<deployments>
           <deployment>
            <name>intergroup-anti_affinity-dep</name>
             <policies>
              <placement>
                  <target_vm_group_ref>affinity-test-gp1</target_vm_group_ref>
                  <type>anti_affinity</type>
                  <vm_group_ref>affinity-test-gp2</vm_group_ref>
                  <enforcement>strict</enforcement>
               </placement>
          </policies>
…
```

In a multiple VIM deployment, the VM groups of a placement policy must belong to the same VIM. That is, the VIM connector must be the same for the VM groups (specified in the vim_id attribute in the locator tag of the VM group). ESC rejects a deployment if the affinity and anti-affinity policies between VM groups are on different VIMs. For more details on deploying VMs on multiple deployments, see "Deploying VNFs onMultiple OpenStack VIMs".

A placement group tag is added under policies. Each <placement_group> contains the following:

- name—name unique per deployment.
- type—affinity or anti_affinity
- enforcement—strict
- vm_group—the content of each vm_group must be a vm group name listed under the same deployment.

The placement group tag is placed within the placement policy. The placement policy describes the relationship between the target vm group and the vm group members. The placement_group policy describes mutual relationship among all vm group members. The placement group policy is not applicable for target vm group.

The datamodel is as follows:

```
<policies>
 <placement_group>
  <name>placement-affinity-1</name>
  <type>affinity</type>
  <enforcement>strict</enforcement>
  <vm_group>t1g1</vm_group>
  <vm_group>t1g2</vm_group>
```

```
  <vm_group>t1g7</vm_group>
 </placement_group>
 <placement_group>
  <name>placement-affinity-2</name>
  <type>affinity</type>
  <enforcement>strict</enforcement>
  <vm_group>t1g3</vm_group>
  <vm_group>t1g4</vm_group>
 </placement_group>
 <placement_group>
  <name>placement-affinity-3</name>
  <type>affinity</type>
  <enforcement>strict</enforcement>
  <vm_group>t1g5</vm_group>
  <vm_group>t1g6</vm_group>
 </placement_group>
 <placement_group>
  <name>placement-anti-affinity-1</name>
  <type>anti_affinity</type>
  <enforcement>strict</enforcement>
  <vm_group>t1g1</vm_group>
  <vm_group>t1g3</vm_group>
  <vm_group>t1g5</vm_group>
 </placement_group>
</policies>
```

Note In the new placement group tag under policies, the <target_vm_group_ref> and <vm_group_ref> are replaced with <vm_group>. The ref based affinity and antiaffinity tags are deprecated.

The placement group policy is applicable for inter group affinity and anti-affinity policies only.

You cannot use both placement and placement group tags together in the inter group affinity and anti-affinity policies.

The placement group name tag must be unique for each placement group policy.

Limitations

Single VM can only be used on one server group for Affinity and Anti-Affinity policies.

Inter Deployment Anti-Affinity Policy

Inter Deployment anti-affinity rules define relationships between different deployments with respect to the host placement. Anti-affinity between deployments is defined such that any VM from one deployment is not co-located on the same host as any other VM from the other deployment.

Note Inter Deployment anti-affinity is supported on OpenStack only. Inter Deployment anti-affinity does not work with host-placement (affinity or anti-affinity) as the latter takes precedence over inter deployment anti-affinity rules.

In the ESC datamodel, inter deployment anti-affinity is defined using anti-affinity groups. All member deployments of an anti-affinity group have an anti-affinity relationship between them. For example, in an anti-affinity group called default-anti with 3 deployments dep-1, dep-2 and dep-3, dep-1 is anti-affinity to dep-2 and dep-3 deployments, dep-2 is anti-affinity to dep-1 and dep-3 deployments, dep-3 is anti-affinity dep-1 and dep-2. A deployment specifies its membership in an anti-affinity group by referencing to all group names it pertains to as shown below.

```
<deployment>
<name>VPC-dep</name>
 <deployment_groups>
  <anti_affinity_group>VPC-ANTI-AFFINITY</anti_affinity_group>
  <anti_affinity_group>VPNAAS-ANTI-AFFINITY</anti_affinity_group>
 </deployment_groups>
….
</deployment>
```

In the above example, VPC-dep is in 2 anti-affinity groups; any other deployment that references one of these 2 groups will have an anti-affinity relationship with VPC-dep.

Inter-deployment Placement Groups

Anti-affinity group is an example of placement group. Anti-affinity group has the following properties in ESC:

- The placement group need not be created or deleted.
- Placement groups can be referenced for the first time by one deployment as well as multiple deployments in parallel.
- Placement rules are applicable during any deployment phase of a service including:
  - Initial deployment
  - Scale Out
  - VM group update addition
  - VM group minimum scaling update (increasing minimum scaling to add VMs)
  - Recovery

A multiple VIM deployment, supports Inter-deployment anti-affinity. However, ESC rejects a deployment

- If the inter-deployment anti-affinity policy is defined between a multiple VIM deployment (with locators within VM groups) and a default VIM deployment (without locators).
- If all the deployments of an inter-deployment anti-affinity group are not deployed on the same VIM (with same vim_id). For more details on a multiple VIM deployment, see Deploying VNFs on Multiple OpenStack VIMs, on page 111.

CHAPTER 28

Affinity and Anti-Affinity Rules on VMware vCenter

Affinity and Anti-Affinity Rules on VMware vCenter

The affinity and anti-affinity rules for VMware vCenter is explained with examples. These rules are created for a cluster and a targeted host.

All VMware vCenter deployments must always be accompanied with zone-host placement policy. The zone-host defines the target VM group which is either the cluster or the host.

Intra Group Affinity Policy

The VNFs with the same VM group can be deployed on the same host.

During deployment, ESC deploys the first VM as an anchor VM for affinity. All the other VMs that follow the same affinity rule will be deployed to the same host as the anchor VM. The anchor VM deployment helps to optimize the resource usage.

Example for Intra Group Affinity Policy:

```
…
<vm_group>
<name>vm-gp</name>
…
<placement>
<type>zone_host</type>
<enforcement>strict</enforcement>
<zone>cluster1</zone>
</placement>
<placement>
<type>affinity</type>
<enforcement>strict</enforcement>
</placement>
…
```

Note Only *strict* attribute is supported for enforcement.

Note Affinity and anti-affinity policy with a host placement policy is incorrect and may cause deployment failure. Host placement alone (without affinity and anti-affinity placement policy within a VM group) can be used to achieve intra group affinity.

Intra Group Anti-Affinity

The VNFs with the same VM group can be deployed in different hosts. During deployment ESC deploys VNFs with the same VM group one after the other. At the end of each VNF deployment, ESC records its host to a list. At the beginning of each VNF's deployment, ESC deploys the VNF to a computing-host that is not in the list. If all the available computing-host(s) are in the list, ESC fails the whole deployment.

Example for Intra Group Anti-Affinity Policy:

```
…
<vm_group>
<name>vm-gp</name>
…
<placement>
<type>zone_host</type>
<enforcement>strict</enforcement>
<zone>cluster1</zone>
</placement>
<placement>
<type>anti_affinity</type>
<enforcement>strict</enforcement>
</placement>
```

Cluster Placement

All VMs in a VM group can be deployed to a cluster. For example, all VMs in a vm group CSR-gp1 can be deployed to cluster dc-cluster2.

Note The VMware vCenter cluster must be created by the administrator.

Example for cluster placement:

```
<name>CSR-gp1</name>
          <placement>
               <type>zone_host</type>
               <enforcement>strict</enforcement>
               <zone>dc-cluster2</zone>
           </placement>
```

Host Placement

All VMS in a VM group can be deployed to a host. For example, all VMs in the vm group CSR-gp1 will be deployed to host 10.2.0.2.

```
<name>CSR-gp1</name>
           <placement>
             <type>zone_host</type>
             <enforcement>strict</enforcement>
              <host>10.2.0.2</host>
            </placement>
```

Inter Group Affinity Policy

The VMs in different VM groups can be deployed to the same host. For example, all VMs in the VM group ASA-gp1 can be deployed to the same host as the VMs in the VM group CSR-gp1.

During deployment ESC deploys the first VM as an anchor VM. All other VMs that follow the same affinity rule will be deployed to the same host as the anchor VM.

Note To ensure that the inter-group affinity rules are applied within a single cluster, verify that all VM groups in a deployment are specified to the same cluster (<zone> in esc data_model).

Example for Inter Group Affinity Policy:

```
<deployment>
<deployment>
<name>test-affinity-2groups</name>
<policies>
<placement>
<target_vm_group_ref>CSR-gp1</target_vm_group_ref>
<type>affinity</type>
<vm_group_ref>CSR-gp2</vm_group_ref>
<vm_group_ref>ASA-gp1</vm_group_ref>
<enforcement>strict</enforcement>
</placement>
</policies>
```

Inter Group Anti-Affinity Policy

The VNFs in the same deployment but different VM Groups can be explicitly deployed in different hosts. For example back-up VNFs or High-availability VNFs. Multiple VM groups can follow this policy by adding the vm_group_ref tag, and providing the VM group name as the value.

Note You can only use one <target_vm_group_ref> tag, type tag and enforcement tag under the placement tag. The host or zone cannot be specified.You can use multiple <vm_group_ref> tags, however the anti-affinity policy only applies between each <vm_group_ref> and their <target_vm_group_ref>, which means that 2 or more <vm_group_ref> can be deployed on the same host, as long as each of them are deployed on a different host from their <target_vm_group_ref> that is acceptable.

Example for Inter Group anti-affinity Policy:

```
<deployments>
           <deployment>
            <name>intergroup-anti_affinity-dep</name>
             <policies>
              <placement>
                  <target_vm_group_ref>affinity-test-gp1</target_vm_group_ref>
                  <type>anti_affinity</type>
                  <vm_group_ref>affinity-test-gp2</vm_group_ref>
                  <enforcement>strict</enforcement>
               </placement>
          </policies>
…
```

In a multiple VIM deployment, the VM groups of a placement policy must belong to the same VIM. That is, the VIM connector must be the same for the VM groups (specified in the vim_id attribute in the locator tag of the VM group). ESC rejects a deployment if the affinity and anti-affinity policies between VM groups are on different VIMs. For more details on deploying VMs on multiple deployments, see "Deploying VNFs onMultiple OpenStack VIMs".

A placement group tag is added under policies. Each <placement_group> contains the following:

- name—name unique per deployment.
- type—affinity or anti_affinity
- enforcement—strict
- vm_group—the content of each vm_group must be a vm group name listed under the same deployment.

The placement group tag is placed within the placement policy. The placement policy describes the relationship between the target vm group and the vm group members. The placement_group policy describes mutual relationship among all vm group members. The placement group policy is not applicable for target vm group.

The datamodel is as follows:

```
<policies>
 <placement_group>
  <name>placement-affinity-1</name>
  <type>affinity</type>
  <enforcement>strict</enforcement>
  <vm_group>t1g1</vm_group>
  <vm_group>t1g2</vm_group>
  <vm_group>t1g7</vm_group>
 </placement_group>
 <placement_group>
  <name>placement-affinity-2</name>
  <type>affinity</type>
  <enforcement>strict</enforcement>
  <vm_group>t1g3</vm_group>
  <vm_group>t1g4</vm_group>
 </placement_group>
 <placement_group>
  <name>placement-affinity-3</name>
  <type>affinity</type>
  <enforcement>strict</enforcement>
  <vm_group>t1g5</vm_group>
  <vm_group>t1g6</vm_group>
 </placement_group>
 <placement_group>
```

```
  <name>placement-anti-affinity-1</name>
  <type>anti_affinity</type>
  <enforcement>strict</enforcement>
  <vm_group>t1g1</vm_group>
  <vm_group>t1g3</vm_group>
  <vm_group>t1g5</vm_group>
 </placement_group>
</policies>
```

Note In the new placement group tag under policies, the <target_vm_group_ref> and <vm_group_ref> are replaced with <vm_group>. The ref based affinity and antiaffinity tags are deprecated.

The placement group policy is applicable for inter group affinity and anti-affinity policies only.

You cannot use both placement and placement group tags together in the inter group affinity and anti-affinity policies.

The placement group name tag must be unique for each placement group policy.

Limitations

Following are the limitations when affinity and anti-affinity rules are applied on VMware vCenter:

- All Affinity rules defined on VMware vCenter are implemented in a cluster.
- DPM, HA and vMotion must be turned off.
- VM deployment and recovery are managed by ESC.
- DRS must be set to manual mode if it is turned on.
- To leverage DRS deployment, shared storage is required.
- Supported value for <enforcement> tag should be 'strict'.
- <zone_host> must be used for any VM group.

CHAPTER 29

Affinity and Anti-Affinity Rules on VMware vCloud Director

Affinity and Anti-Affinity Rules on VMware vCloud Director

ESC supports affinity and anti-affinity placement policy for vCD. However, it does not support zone-host placement policy.

The affinity and anti-affinity implementation in ESC depends on the affinity rule (VM-VM affinity rule in vSphere) in the vCloud Director. The example below shows affinity and anti-affinity rules in the vCD VNF deployment datamodel.

```
<deployments>
        <deployment>
          <!-- vApp instance name -->
          <name>d1</name>
          <policies>
            <placement_group>
              <name>d1-placement-affinity-1</name>
              <type>affinity</type>
              <enforcement>strict</enforcement>
              <vm_group>g1</vm_group>
              <vm_group>g2</vm_group>
            </placement_group>
          </policies>
         ….….….…...
         ….….….….….
       </deployment>
    </deployments>
```

For vCD deployment, see Deploying Virtual Network Functions on VMware vCloud Director (vCD), on page 140.

CHAPTER 30

Configuring Custom VM Name

Configuring Custom VM Name

You can customize VM names if you do not want ESC to auto-generate VM names. To customize VM names, specify the `vim_vm_name` in the VM group section of the deployment datamodel. If `vim_vm_name` is not specified, ESC will auto-generate the VM names.

While specifying a custom name, if a VM group has more than one VM, an "_<index>" is appended to the custom VM name in the output. For example, the first VM in the group is named as specified in the `vim_vm_name`, and second VM onwards an index "_1", "_2" is appended to the custom name. For a custom name specified as ABC, the output will display the VM names as VMname, VMname_1, VMname_2, and so on. If a VM group only has a single VM, then there is no "_<index>" appended to the custom VM name.

A single deployment can contain multiple VM groups, and each individual VM group can specify a different `vim_vm_name` value, if required. For example, a deployment could have two VM groups: the first group specifies a `vim_vm_name` and all VMs have their names generated as described above. The second VM group does not specify a `vim_vm_name`, therefore all VM names created from this group are auto generated.

Custom VM names only have to be unique within the deployment and tenant for an OpenStack deployment. In other words, custom VM names can be duplicated across different tenants - or even duplicated within the same tenant as long as it is for a different deployment. For a VMware deployment, the custom VM name must be unique throughout the entire vCenter server. In other words, no duplicate VM names are permitted.

Note You can use a maximum of 63 characters for the custom name. A VM name should not contain special characters and can only contain alphanumeric characters and "_" and "-".

```
<esc_datamodel xmlns="http://www.cisco.com/esc/esc"> <tenants><tenant>
  <name>Admin</name>
   <deployments>
    <deployment>
     <deployment_name>NwDepModel_nosvc</deployment_name>

    <vm_group>
    <name>CIRROS</name>
    <image>Automation-Cirros-Image</image>
    <flavor>Automation-Cirros-Flavor</flavor>
    <vim_vm_name>VMname</vim_vm_name>
```

```
    <scaling>
        <min_active>1</min_active>
        <max_active>2</max_active>
        <elastic>true</elastic>
    </scaling>
</vm_group>
```

Note

- The ESC Portal does not display the VM Name that was configured during the deployment time.
- Duplicate VM Names are not supported on VMWare.
- VM names cannot be modified after a deployment is complete.

The following are some output samples with the custom VM name. If the vim_vm_name was set during the deployment, the same value will be shown in the output. If this value was not set during the deployment, ESC will auto-generate the VM name.

- Below is an example of the output operational data fetched using the esc_nc_cli script after adding a custom VM name. A new element called <vmname> will be shown under the vm_group element. The value in the <status_message> field is also updated to reflect the custom VM name.

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="1">
  <data>
    <esc_datamodel xmlns="http://www.cisco.com/esc/esc">
      <opdata>
        <tenants>
          <tenant>
            <name>xyzzy</name>
            <deployments>
              <deployment_name>my-deployment-123</deployment_name>
              <deployment_id>78d48bf8-5f67-45fc-8d92-5ad4676yf57</deployment_id>
              <vm_group>
                <name>Grp1</name>
                <vm_instance>
                  <vm_id>df108144-ec4f-4d66-a62f-98096ecddef0</vm_id>
                  <name>VMname</name>
              </vm_group>
```

- Below is an example output operational data fetched using a REST API.

```
GET http://localhost:8080/ESCManager/v0/deployments/example-deployment-123
| xmllint --format -
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<deployment xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <datacenter>
    <default>false</default>
  </datacenter>
  <deployment_details>
    <host_uuid>8623f1476302a5815608dbd4c2f836c570e8c74cbfbaff41c78564b1</host_uuid>
    <host_name>my-server</host_name>
    <vm_uuid>e7e5a905-e0c7-4652-ae1f-23a409a58219</vm_uuid>
    <interfaces>
      <interface>

      </interface>
    </interfaces>
    <vm_group_name>Grp1</vm_group_name>
```

```
    <vm_name>VMname_1</vm_name><-- ##### custom vm name, single VM in the VM group, so
 no appended "_<index>" -->
    <vm_state_machine_state>VM_ALIVE_STATE</vm_state_machine_state>
  </deployment_details>
</deployment>
```

CHAPTER 31

Managing Existing Deployments

After a deployment is created successfully, the resources within a deployment can be updated. As part of deployment management, you can add or delete resources, or update the configuration of the existing resources. These updates can be made in a running deployment. This chapter describes managing these resources in detail.

Updating an Existing Deployment

You can update an existing deployment by adding new VM groups, interfaces, networks, and so on. You can also update the day-0 configuration, KPIs and Rules for the VM groups. You can add or delete a vm_group, add or delete an ephemeral network in a vm_group, and add or delete an interface in a VM group after successful deployment.

On OpenStack, you can perform all the updates such as add or delete a vm_group, ephemeral network vm_group, and an interface in a single deployment.

During a service update, auto-recovery actions may drive the service to an inconsistent state. To prevent triggering of auto-recovery actions, monitors are disabled before the service update workflow, and enabled after the update is complete.

Note During VM recovery in the middle of a service update request, the Northbound client may receive a SERVICE_UPDATED FAILURE notification even before receiving the VM recovery notifications. It is recommended to check the service until it moves to the SUCCESS or ERROR state before sending manual recovery or other service level requests.

Updating an existing deployment is supported both on OpenStack and VMware vCenter. The table below lists the components that can be updated in an existing deployment.

Table 15: Updating an Existing Deployment on OpenStack, VMware vCenter and vCloud Director

Update	OpenStack	VMware vCenter	vCloud Director
Adding a VM group	Supported	Supported	Supported
Deleting a VM group	Supported	Supported	Supported

Update	OpenStack	VMware vCenter	vCloud Director
Deleting VM groups when the service is in error state	Supported	Supported	Not supported
Adding an ephemeral network	Supported	Not supported	Not supported
Deleting an ephemeral network	Supported	Not supported	Not supported
Adding an interface	Supported	Not supported	Not supported
Deleting an interface	Supported	Not supported	Not supported
Updating an interface	Supported	Supported	Not supported
Adding a Static IP pool	Supported	Not supported	Not supported
Deleting a Static IP pool	Supported	Not supported	Not supported
Updating the day-0 config in a VM group	Supported	Supported	Not supported
Updating the KPIs and rules	Supported	Supported	Not supported
Updating the number of VMs (Scale In or Scale Out) in a VM group	Supported	Supported	Not supported
Updating the recovery wait time	Supported	Supported	Not supported
Updating the recovery policy	Supported	Not supported	Not supported
Updating an image	Supported	Not supported	Not supported

Note Updating an existing deployment on multiple OpenStack VIMs is also supported. However, the locator attribute within the vm group cannot be updated. For more information on Deploying VMs on Multiple VIMs, see Deploying VNFs on Multiple OpenStack VIMs.

Adding a VM Group

You can add or delete a vm_group from a running deployment using the existing images and flavors.

NETCONF request to add a vm_group:

```
<esc_datamodel xmlns="http://www.cisco.com/esc/esc"> <tenants><tenant>
  <name>Admin</name>
   <deployments>
```

```
    <deployment>
     <deployment_name>NwDepModel_nosvc</deployment_name>

     <vm_group>
        <image></image>
        <Flavor></Flavor>
       .........
    </vm_group>
    <vm_group>
  <image></image>
 <Flavor></Flavor>
 .........
   </vm_group>
   <vm_group>
 <image></image>
 <Flavor></Flavor>
 .........
   </vm_group>
  </deployment>
</deployments>
    </tenant></tenants>
</esc_datamodel>
```

NETCONF notification upon successful addition of a VM Group:

```
UPDATE SERVICE REQUEST RECEIVED (UNDER TENANT)
           VM_DEPLOYED
             VM_ALIVE
          SERVICE_UPDATED
UPDATE SERVICE REQUEST RECEIVED (UNDER TENANT)
```

Deleting a VM Group

NETCONF request to delete a vm_group:

```
<esc_datamodel xmlns="http://www.cisco.com/esc/esc">
 <tenants><tenant>
  <name>Admin</name>
   <deployments>
    <deployment>
     <deployment_name>NwDepModel_NoSvc</deployment_name>

     <vm_group>
        <image></image>
        <Flavor></Flavor>
       .........
    </vm_group>
    <vm_group nc:operation="delete">
 <image></image>
 <Flavor></Flavor>
 .........
   </vm_group>
   <vm_group nc:operation="delete">
 <image></image>
 <Flavor></Flavor>
  .........
   </vm_group>
  </deployment>
</deployments>
    </tenant></tenants>
</esc_datamodel>
```

NETCONF notification upon successful deletion of vm_group:

```
UPDATE SERVICE REQUEST RECEIVED (UNDER TENANT)
             VM_UNDEPLOYED
             SERVICE_UPDATED
UPDATE SERVICE REQUEST RECEIVED (UNDER TENANT)
```

Deleting VM Groups in Error State

You can now delete vm groups when the deployment is in error state by performing a deployment update. However, additional configurations to the vm groups such as adding one or more vm groups, or changing the attribute value of a different vm group while deleting a particular vm group are not allowed.

Adding an Ephemeral Network in a VM Group

You can add an ephemeral network in a vm_group using the existing images and flavors.

NETCONF request to add an ephemeral in a vm_group:

```
<esc_datamodel xmlns="http://www.cisco.com/esc/esc"> <tenants><tenant>
  <name>Admin</name>
   <deployments>
    <deployment>
     <deployment_name>NwDepModel_nosvc</deployment_name>
      <networks>
      <network>
 .........
 </network>
 <network>
 .........
 </network>
 <network>
 .........
 </network>
     </networks>
     <vm_group>
        <image></image>
        <Flavor></Flavor>
       .........
    </vm_group>
  </deployment>
</deployments>
    </tenant></tenants>
</esc_datamodel>
```

NETCONF notification upon successful addition of an ephemeral network in a vm_group:

```
UPDATE SERVICE REQUEST RECEIVED (UNDER TENANT)
           CREATE_NETWORK
           CREATE_SUBNET
          SERVICE_UPDATED
UPDATE SERVICE REQUEST RECEIVED (UNDER TENANT)
```

Deleting an Ephemeral Network in a VM Group

NETCONF request to delete an ephemeral network in a vm_group

```
<esc_datamodel xmlns="http://www.cisco.com/esc/esc"> <tenants><tenant>
  <name>Admin</name>
   <deployments>
    <deployment>
     <deployment_name>NwDepModel</deployment_name>
      <networks>
      <network nc:operation="delete">
 .........
```

```
 </network>
 <network>
 .........
 </network>
 <network nc:operation="delete">
 .........
 </network>
     </networks>
     <vm_group>
        <image></image>
        <Flavor></Flavor>
       .........
    </vm_group>
  </deployment>
</deployments>
    </tenant></tenants>
</esc_datamodel>
```

NETCONF notification upon successful deletion of an ephemeral network in a vm_group:

```
UPDATE SERVICE REQUEST RECEIVED (UNDER TENANT)
          DELETE_SUBNET
          DELETE_NETWORK
          SERVICE_UPDATED
UPDATE SERVICE REQUEST RECEIVED (UNDER TENANT)
```

Adding an Interface in a VM Group (OpenStack)

You can add an interface in a vm_group from a running deployment using the existing images and flavors.

NETCONF request to add an interface in a vm_group:

```
<interfaces>
  <interface>
    <nicid>0</nicid>
    <network>my-network</network>
  </interface>
  <interface>
    <nicid>1</nicid>
    <network>utr-net</network>
  </interface>
  <interface>
    <nicid>2</nicid>
    <network>utr-net-1</network>
  </interface>
</interfaces>
```

Note ESC Release 2.3 and later supports adding and deleting interfaces using the ESC Portal for OpenStack.

ESC supports adding and deleting interfaces from a vm_group using both REST and NETCONF APIs.

Deleting an Interface in a VM Group (OpenStack)

NETCONF request to delete an interface in a vm_group:

```
<interfaces>
  <interface>
    <nicid>0</nicid>
    <network>my-network</network>
```

```
  </interface>
  <interface>
    <nicid>1</nicid>
    <network>utr-net</network>
  </interface>
  <interface nc:operation="delete">
    <nicid>2</nicid>
    <network>utr-net-1</network>
  </interface>
</interfaces>
```

You can simultaneously add and delete interfaces in a VM group (OpenStack only) in the same deployment request.

Note ESC does not support the following:

- Updating the properties of an existing vm_group, network or subnet.
- Updating the image and flavor of a vm_group.
- Blank names for resource names (that is, vm_group, network, subnet or Interface).

In Cisco ESC Release 2.0 or earlier, the ephemeral networks or subnets can only be added or deleted.

ESC does not support the day 0 configuration of new interfaces added during a deployment update. You must perform additional configuration separately in the VNF as part of the day-n configuration. If you delete an interface with token replacement, you must update the day 0 configuration to remove that interface. In future, ESC will use the new day 0 configuration for recovery.

A new interface without the nic ids is not configured during a deployment update.

New interfaces with existing day 0 configuration are configured.

Updating an Interface (OpenStack)

Updating an interface on OpenStack deletes the previous interface and creates a new one with the existing nic id.

The datamodel is as follows:

```
<interfaces>
                <interface>
                  <nicid>0</nicid>
                  <network>my-network</network>
                </interface>
                 <interface>
                  <nicid>1</nicid>
                  <network>utr-net-2</network>
                </interface>
        </interfaces>
```

A VM_UPDATED notification is sent with the details of all the interfaces in a VM, followed by a SERVICE_UPDATED notification after the workflow is updated.

```
<?xml version="1.0" encoding="UTF-8"?>
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2015-07-25T00:45:27.64+00:00</eventTime>
  <escEvent xmlns="http://www.cisco.com/esc/esc">
    <status>SUCCESS</status>
    <status_code>200</status_code>
```

```
    <status_message>VM has been updated successfully. vm:
utr-80__7515__utr-80__utr-80utr-80utr-801.2__0__utr-80__0</status_message>
    <svcname>utr-80</svcname>
    <svcversion>1.2</svcversion>
    <depname>utr-80</depname>
    <tenant>utr-80</tenant>
    <svcid>c1294ad1-fd7b-4a73-8567-335160dce90f</svcid>
    <depid>ecedf755-502c-473a-82f2-db3a5485fdf5</depid>
    <vm_group>utr-80</vm_group>
    <vm_source>
      <vmid>4b20024f-d8c8-4b1a-8dbe-3bf1011a0bcb</vmid>
      <hostid>71c7f3afb281485067d8b28f1734ec6b63f9e3225045c581168cc39d</hostid>
      <hostname>my-server</hostname>
      <interfaces>
        <interface>
          <nicid>0</nicid>
          <port_id>6bbafbf5-51a1-48c0-a4a5-cd6092657e5c</port_id>
          <network>7af5c7df-6246-4d53-91bd-aa12a1607656</network>
          <subnet>7cb6815e-3023-4420-87d8-2b10efcbe14e</subnet>
          <ip_address>192.168.0.10</ip_address>
          <mac_address>fa:16:3e:bc:07:d5</mac_address>
          <netmask>255.255.255.0</netmask>
          <gateway>192.168.0.1</gateway>
        </interface>
        <interface>
          <nicid>1</nicid>
          <port_id>6d54d3a8-b793-40b8-9a32-c7e2f08e0917</port_id>
          <network>4f85613a-d3fc-4b49-9cb0-b91d4360918b</network>
          <subnet>c3724a64-ffed-43b6-aba8-63287c5344ea</subnet>
          <ip_address>10.91.90.2</ip_address>
          <mac_address>fa:16:3e:49:d0:00</mac_address>
          <netmask>255.255.255.0</netmask>
          <gateway>10.91.90.1</gateway>
        </interface>
        <interface>
          <nicid>3</nicid>
          <port_id>04189123-fc7a-4418-877b-61c24a5e8508</port_id>
          <network>f9c7978f-800e-4bfc-bc20-1c29acef87d9</network>
          <subnet>63ae5e39-c41a-4b28-9ac7-ed94b5e477b0</subnet>
          <ip_address>172.16.0.97</ip_address>
          <mac_address>fa:16:3e:5e:2e:e3</mac_address>
          <netmask>255.240.0.0</netmask>
          <gateway>172.16.0.1</gateway>
        </interface>
      </interfaces>
    </vm_source>
    <vm_target>
  </vm_target>
    <event>
      <type>VM_UPDATED</type>
    </event>
  </escEvent>
</notification>
```

Note

- Interfaces are unique based on nic ids. If new interfaces are added, they should have different nic ids. If an interface is edited, and has the same nic id, it is considered as an update to the existing interface.

Updating an Interface (VMware vCenter)

You can update a network associated with an interface, while updating an existing deployment. Replace the old network name with a new name in the deployment request to update the network. The port group on the interfaces is updated for all VMs in the VM group during the network update.

Note IP update is not supported during an interface update on VMware vCenter.

Static IP and mac pool updates are not supported during an interface update on VMware vCenter when min > 1 in a vm group.

The datamodel update is as follows:

Existing datamodel:

```
<interface>
  <nicid>1</nicid>
  <network>MgtNetwork</network>
</interface>
```

New datamodel:

```
<interface>
  <nicid>1</nicid>
  <network>VNFNetwork</network>
</interface>
```

The following notification is received after successful update:

```
<?xml version="1.0" encoding="UTF-8"?>
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2016-08-17T12:03:12.518+00:00</eventTime>
  <escEvent xmlns="http://www.cisco.com/esc/esc">
    <status>SUCCESS</status>
    <status_code>200</status_code>
    <status_message>Updated 1 interface: [net=VNFNetwork,nicid=1]</status_message>
    <depname>u1-asa</depname>
    <tenant>admin</tenant>
    <tenant_id>SystemAdminTenantId</tenant_id>
    <depid>90139aa1-9705-4b07-9963-d60691d3b0ad</depid>
    <vm_group>utr-asa-1</vm_group>
    <vm_source>
      <vmid>50261fbc-88a0-8601-71a9-069460720d4f</vmid>
      <hostid>host-10</hostid>
      <hostname>172.16.103.14</hostname>
      <interfaces>
        <interface>
          <nicid>1</nicid>
          <type>virtual</type>
          <port_id/>
          <network>VNFNetwork</network>
          <subnet/>
          <ip_address>192.168.0.254</ip_address>
          <mac_address>00:50:56:a6:d8:1d</mac_address>
        </interface>
      </interfaces>
    </vm_source>
    <vm_target>
  </vm_target>
    <event>
      <type>VM_UPDATED</type>
    </event>
```

```
  </escEvent>
</notification>
<?xml version="1.0" encoding="UTF-8"?>
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2016-08-17T12:03:12.553+00:00</eventTime>
  <escEvent xmlns="http://www.cisco.com/esc/esc">
    <status>SUCCESS</status>
    <status_code>200</status_code>
    <status_message>Service group update completed successfully</status_message>
    <depname>u1-asa</depname>
    <tenant>admin</tenant>
    <tenant_id>SystemAdminTenantId</tenant_id>
    <depid>90139aa1-9705-4b07-9963-d60691d3b0ad</depid>
    <vm_source>
  </vm_source>
    <vm_target>
  </vm_target>
    <event>
      <type>SERVICE_UPDATED</type>
    </event>
  </escEvent>
</notification>
```

Updating an Interface (Cloud Services Platform)

You can now configure and update the Vlan, type and bandwidth properties of an interface using the interface extensions for a CSP deployment. The admin stuatus (admin_state_up) and network attributes can be configured and updated using interfaces.

The **container name** attribute must match the **nicid** value. For example, if the **container name** is 1, the **nicid** value must also be 1 for configuring and updating the interface properties.

Vlan

To configure and update the vlan property, execute the following command from ESC:

esc_nc_cli --user <username> --password <password> edit-config interfaceVlan.xml

The sample interfaceVlan.xml is as follows:

```
<interfaces>
   <interface>
    <nicid>0</nicid>
    <type>virtual</type>
    <model>virtio</model>
    <network>Eth0-2</network>
    <ip_address>192.168.24.45</ip_address>
    <admin_state_up>true</admin_state_up>
  </interface>
 <interface>
    <nicid>1</nicid>
    <type>virtual</type>
    <model>virtio</model>
    <network>Eth0-2</network>
    <admin_state_up>true</admin_state_up>
  </interface>
</interfaces>
......
  .....
<extensions>
  <extension>
    <name>interfaces</name>
    <containers>
```

```
     <container>
      <name>0</name>
      <properties>
        <property>
          <name>passthroughMode</name>
          <value>none</value>
        </property>
        <property>
          <name>tagged</name>
          <value>false</value>
        </property>
        <property>
          <name>type</name>
          <value>access</value>
        </property>
        <property>
          <name>vlan</name>
          <value>1</value>
        </property>
      </properties>
    </container>
   <container>
      <name>1</name>
      <properties>
        <property>
          <name>passthroughMode</name>
          <value>none</value>
        </property>
        <property>
          <name>tagged</name>
          <value>false</value>
        </property>
        <property>
          <name>type</name>
          <value>access</value>
        </property>
        <property>
          <name>bandwidth</name>
          <value>750</value>
        </property>
        <property>
          <name>vlan</name>
          <value>11</value>
        </property>
      </properties>
    </container>
  </containers>
</extension>
<extension>
  <name>serial_ports</name>
  <containers>
    <container>
      <name>0</name>
      <properties>
        <property>
          <name>serial_type</name>
          <value>console</value>
        </property>
      </properties>
    </container>
  </containers>
</extension>
<extension>
  <name>image</name>
```

```
        <properties>
          <property>
            <name>disk-resize</name>
            <value>true</value>
          </property>
          <property>
            <name>disk_type</name>
            <value>virtio</value>
          </property>
        </properties>
      </extension>
    </extensions>
```

Bandwidth

You can configure and update the bandwidth for an interface. The value of bandwidth is in megabits per second. It must be a positive integer.

To configure and update the bandwidth, execute the following command from ESC:

esc_nc_cli --user <username> --password <password> edit-config bandwidth.xml

The sample bandwidth.xml is as follows:

```
        <properties>
              <property>
                <name>passthroughMode</name>
                <value>none</value>
              </property>
              <property>
                <name>tagged</name>
                <value>false</value>
              </property>
              <property>
                <name>type</name>
                <value>access</value>
              </property>
              <property>
                <name>bandwidth</name>
                <value>750</value>
              </property>
              <property>
                <name>vlan</name>
                <value>11</value>
              </property>
            </properties>
          </container>
        </containers>
      </extension>
      <extension>
        <name>serial_ports</name>
        <containers>
          <container>
            <name>0</name>
            <properties>
              <property>
                <name>serial_type</name>
                <value>console</value>
              </property>
            </properties>
          </container>
        </containers>
      </extension>
      <extension>
```

```
            <name>image</name>
            <properties>
              <property>
                <name>disk-resize</name>
                <value>true</value>
              </property>
              <property>
                <name>disk_type</name>
                <value>virtio</value>
              </property>
            </properties>
          </extension>
        </extensions>
```

Type

The valid values for the property *type* are access and trunk only. To configure and update the property *type*, execute the following command from ESC:

esc_nc_cli --user <username> --password <password> edit-config interfaceType.xml

The sample interfaceType.xml is as follows:

```
<extensions>
      <extension>
        <name>interfaces</name>
        <containers>
           <container>
            <name>0</name>
            <properties>
              <property>
                <name>passthroughMode</name>
                <value>none</value>
              </property>
              <property>
                <name>tagged</name>
                <value>false</value>
              </property>
              <property>
                <name>type</name>
                <value>access</value>
              </property>
              <property>
                <name>vlan</name>
                <value>1</value>
              </property>
            </properties>
          </container>
         <container>
            <name>1</name>
            <properties>
              <property>
                <name>passthroughMode</name>
                <value>none</value>
              </property>
              <property>
                <name>tagged</name>
                <value>false</value>
              </property>
              <property>
                <name>type</name>
                <value>access</value>
              </property>
              <property>
```

```
                <name>bandwidth</name>
                <value>750</value>
              </property>
              <property>
                <name>vlan</name>
                <value>11</value>
              </property>
            </properties>
          </container>
        </containers>
      </extension>
      <extension>
        <name>serial_ports</name>
        <containers>
          <container>
            <name>0</name>
            <properties>
              <property>
                <name>serial_type</name>
                <value>console</value>
              </property>
            </properties>
          </container>
        </containers>
      </extension>
      <extension>
        <name>image</name>
        <properties>
          <property>
            <name>disk-resize</name>
            <value>true</value>
          </property>
          <property>
            <name>disk_type</name>
            <value>virtio</value>
          </property>
        </properties>
      </extension>
    </extensions>
```

Admin Status

The `admin_state_up` attribute in the interface allows you to enable or disable the VNIC. The `admin_state_up` value can be set to true or false. If true, then vNIC is enabled. If the `admin_state_up` value is not configured through ESC, then the status is **UP** on CSP. To configure and update the admin_state_up attribute, execute the following command from ESC:

esc_nc_cli --user <username> --password <password> edit-config adminStateUp.xml

The sample adminStateUp.xml is as follows:

```
<interfaces>
               <interface>
                <nicid>0</nicid>
                <type>virtual</type>
                <model>virtio</model>
                <network>Eth0-2</network>
                <ip_address>192.168.24.45</ip_address>
                <admin_state_up>true</admin_state_up>
              </interface>
             <interface>
                <nicid>1</nicid>
                <type>virtual</type>
```

```
    <model>virtio</model>
    <network>Eth0-2</network>
    <admin_state_up>false</admin_state_up>
  </interface>
</interfaces>
........
  .......
<extensions>
  <extension>
    <name>interfaces</name>
    <containers>
       <container>
        <name>0</name>
        <properties>
          <property>
            <name>passthroughMode</name>
            <value>none</value>
          </property>
          <property>
            <name>tagged</name>
            <value>false</value>
          </property>
          <property>
            <name>type</name>
            <value>access</value>
          </property>
          <property>
            <name>vlan</name>
            <value>1</value>
          </property>
        </properties>
      </container>
     <container>
        <name>1</name>
        <properties>
          <property>
            <name>passthroughMode</name>
            <value>none</value>
          </property>
          <property>
            <name>tagged</name>
            <value>false</value>
          </property>
          <property>
            <name>type</name>
            <value>access</value>
          </property>
          <property>
            <name>vlan</name>
            <value>11</value>
          </property>
        </properties>
      </container>
    </containers>
  </extension>
  <extension>
    <name>serial_ports</name>
    <containers>
      <container>
        <name>0</name>
        <properties>
          <property>
            <name>serial_type</name>
            <value>console</value>
```

```
                  </property>
                </properties>
              </container>
            </containers>
          </extension>
          <extension>
            <name>image</name>
            <properties>
              <property>
                <name>disk-resize</name>
                <value>true</value>
              </property>
              <property>
                <name>disk_type</name>
                <value>virtio</value>
              </property>
            </properties>
          </extension>
        </extensions>
      .......................
```

Network

You can configure and update the network attribute through the interface. To configure and update the network, execute the following command from ESC:

esc_nc_cli --user <username> --password <password> edit-config NetworkNameChange.xml

The sample NetworkNameChange.xml is as follows:

```
<interfaces>
               <interface>
                <nicid>0</nicid>
                <type>virtual</type>
                <model>virtio</model>
                <network>Eth0-2</network>
                <ip_address>192.168.24.45</ip_address>
                <admin_state_up>true</admin_state_up>
              </interface>
             <interface>
                <nicid>1</nicid>
                <type>virtual</type>
                <model>virtio</model>
                <network>Eth0-2</network>
                <admin_state_up>false</admin_state_up>
              </interface>
            </interfaces>
            .........
            ............
            <extensions>
              <extension>
                <name>interfaces</name>
                <containers>
                   <container>
                    <name>0</name>
                    <properties>
                      <property>
                        <name>passthroughMode</name>
                        <value>none</value>
                      </property>
                      <property>
                        <name>tagged</name>
                        <value>false</value>
                      </property>
```

```
                <property>
                  <name>type</name>
                  <value>access</value>
                </property>
                <property>
                  <name>vlan</name>
                  <value>1</value>
                </property>
              </properties>
            </container>
           <container>
              <name>1</name>
              <properties>
                <property>
                  <name>passthroughMode</name>
                  <value>none</value>
                </property>
                <property>
                  <name>tagged</name>
                  <value>false</value>
                </property>
                <property>
                  <name>type</name>
                  <value>access</value>
                </property>
                <property>
                  <name>vlan</name>
                  <value>11</value>
                </property>
              </properties>
            </container>
          </containers>
        </extension>
        <extension>
          <name>serial_ports</name>
          <containers>
            <container>
              <name>0</name>
              <properties>
                <property>
                  <name>serial_type</name>
                  <value>console</value>
                </property>
              </properties>
            </container>
          </containers>
        </extension>
        <extension>
```

Adding a Static IP Pool

You can add a new static IP pool to the existing deployment.

NETCONF request to add a static IP pool:

```
<scaling>
<min_active>2</min_active>
<max_active>5</max_active>
<elastic>true</elastic>
<static_ip_address_pool>
<network>IP-pool-network-A</network>
<ip_address_range>
<start>172.16.5.13</start>
<end>172.16.5.13</end>
```

```
</ip_address_range>
</static_ip_address_pool>
<static_ip_address_pool>
<network>IP-pool-network-B</network>
<ip_address_range>
<start>172.16.7.13</start>
<end>172.16.7.13</end>
</ip_address_range>
</static_ip_address_pool>
</scaling>
```

Deleting a Static IP Pool

You can delete the existing IP pools in a running deployment.

NETCONF request to delete a static IP pool:

```
<scaling>
<min_active>2</min_active>
<max_active>5</max_active>
<elastic>true</elastic>
<static_ip_address_pool>
<network>IP-pool-network-A</network>
<ip_address_range>
<start>172.16.5.13</start>
<end>172.16.5.13</end>
</ip_address_range>
</static_ip_address_pool>
<static_ip_address_pool nc:operation="delete">
<network>IP-pool-network-B</network>
<ip_address_range>
<start>172.16.7.13</start>
<end>172.16.7.13</end>
</ip_address_range>
</static_ip_address_pool>
</scaling>
```

Note

- You cannot update an already existing static IP pool in an existing deployment. You can only add a new static IP pool, or delete if the static IP pool is not in use.
- You cannot update the IP address of an interface. That is, you cannot deploy with one IP address, and then add a new IP in the same nic id.

The following scenarios are supported or rejected because of the dependencies within the static IP pools, interfaces, and networks.

Request	Supported or Rejected
Add or delete new static IP pools in single or different requests.	Supported
Add interfaces with static IP.	Supported
Add an interface and the corresponding IP pool in the same request.	Supported

Request	Supported or Rejected
Delete an interface, retaining the corresponding IP pool.	Supported
Delete an interface and its corresponding IP pool in the same request.	Supported
Delete an IP pool, when one of its IPs are being used in an interface in a VM.	Rejected
Add a network, and a static IP pool having different network in a single request.	Supported
To an existing network, add a corresponding interface and an IP pool in the same update.	Supported
Add a new network in an update, and a new corresponding IP pool in the next update.	Supported
Add an IP pool without corresponding network.	Rejected
Delete a network and the referencing IP pool in the same request, when none of the IPs are being used in any interfaces.	Supported
Delete a network which is being used in an IP pool and interface.	Rejected
To an existing network, add an interface and an IP pool in the same update.	Supported
Delete an IP pool that does not have any IPs used in interface, though the network with subnet is present.	Supported
Add an IP pool which already exists.	Request is accepted by NETCONF but no action taken
Update the IP addresses of an existing IP pool.	Rejected

Updating the Day 0 Configuration in a VM Group

To update (add, delete or change) the day-0 configuration of a VM group in an existing deployment, edit-config the deployment and update the configuration under config_data. The new day-0 config file is only applied on future deployment, which is triggered by either VM recovery (that is undeploy/deploy) or scale-out.

Note To change the existing day-0 config file, the URL or path must be specified. This enables ESC to detect the change that has occurred in the configuration.

In the example below, if a VM ALIVE event is not received, you can change the action from triggering auto recovery to simply logging the event.

Existing configuration:

```
<config_data>
  <configuration>
    <dst>WSA_config.txt</dst>

<file>https://172.16.73.167:4343/day0/cfg/vWSA/node/001-wsa/provider/Symphony_VNF_P-1B/file>

  </configuration>
  <configuration>
    <dst>license.txt</dst>

<file>https://172.16.73.167:4343/day0/cfg/vWSA/node/001-wsa/provider/Symphony_VNF_P-1B/wsa-license.txt</file>

  </configuration>
</config_data>
```

New configuration:

```
<config_data>
  <configuration>
    <dst>WSA_config.txt</dst>

<file>https://172.16.73.167:4343/day0/cfg/vWSA/node/001-wsa/provider/Symphony_VNF_P-1B/file>

  </configuration>
  <configuration>
    <dst>license.txt</dst>

<file>https://172.16.73.167:4343/day0/cfg/vWSA/node/002-wsa/provider/Symphony_VNF_P-1B/wsa-license.txt</file>

  </configuration>
</config_data>
```

SERVICE_UPDATED notification is received after updating the configuration.

```
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2016-05-05T00:35:15.359+00:00</eventTime>
  <escEvent xmlns="http://www.cisco.com/esc/esc">
    <status>SUCCESS</status>
    <status_code>200</status_code>
    <status_message>Service group update completed successfully</status_message>
    <depname>900cd7554d31-5454000964474c1cbc07256792e63240-cloudvpn</depname>
    <tenant>Symphony_VNF_P-1B</tenant>
    <tenant_id>3098b55808e84484a4f8bab2160a41a7</tenant_id>
    <depid>b7d566ce-1ee6-4147-8c23-c8bcb5d05fd4</depid>
    <vm_source/>
    <vm_target/>
    <event>
      <type>SERVICE_UPDATED</type>
    </event>
  </escEvent>
</notification>
```

For more information on day-0 configuration, see Day Zero Configuration, on page 161.

Updating the KPIs and Rules

ESC allows updating KPIs and rules for a VM in the existing deployment. Edit the datamodel to update the KPIs and rules section.

For example, to change the *Polling Frequency* in an existing deployment, update the *<poll_frequency>* element in the KPI section of the datamodel.

Change *<poll_frequency>3</poll_frequency>* to *<poll_frequency>20</poll_frequency>* in the sample below.

```
<kpi>
            <event_name>VM_ALIVE</event_name>
            <metric_value>1</metric_value>
            <metric_cond>GT</metric_cond>
            <metric_type>UINT32</metric_type>
            <metric_collector>
              <type>ICMPPing</type>
              <nicid>0</nicid>
              <poll_frequency>3</poll_frequency>
              <polling_unit>seconds</polling_unit>
              <continuous_alarm>false</continuous_alarm>
            </metric_collector>
          </kpi>
```

Similarly, the existing rules can be updated for a VM. For example, to switch off the auto- recovery on a boot failure and to log the action, update *<action>FALSE recover autohealing</action>* to *<action>FALSE log</action>* in the sample below.

```
<rules>
             <admin_rules>
               <rule>
                 <event_name>VM_ALIVE</event_name>
                 <action>ALWAYS log</action>
                 <action>FALSE recover autohealing</action>
                 <action>TRUE servicebooted.sh</action>
               </rule>
              ...
  ...
           </rules>
```

Note

- During the KPIs or rules update, auto-recovery does not happen as the monitors are unset. Auto-recovery happens when the monitors are reset in the deployment.
- The *event_name* cannot be modified during an update. It can only be added or deleted.

For more information on KPIs and Rules, see the KPIs and Rules Section.

Updating the Number of VMs in a Deployment (Updating Manual Scale In/ Scale Out)

You can add and remove VMs from an existing deployment by changing the min_active and max_active values in the scaling section of the datamodel. This alters the size of the initial deployment.

In the example below, the deployment has an initial count of 2 VMs, which can scale out to 5 VMs.

```
 <esc_datamodel xmlns:ns2="urn:ietf:params:xml:ns:netconf:notification:1.0"
xmlns:ns1="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:ns3="http://www.cisco.com/esc/esc_notifications"
xmlns:ns0="http://www.cisco.com/esc/esc" xmlns="http://www.cisco.com/esc/esc">
  <version>1.0.0</version>
      . . .
     <vm_group>
        </interfaces>
          <interface>
            <network>1fbf9fc2-3074-4ae6-bb0a-09d526fbada6</network>
            <nicid>1</nicid>
            <ip_address>10.0.0.0</ip_address>
          </interface>
        </interfaces>
       <scaling>
          <min_active>2</min_active>
```

```
            <max_active>5</max_active>
            <elastic>true</elastic>

              . . .
```

The example below creates an additional 8 VMs bringing the number of active VMs up to a minimum of 10. See the table below for more scenarios.

```
<esc_datamodel xmlns:ns2="urn:ietf:params:xml:ns:netconf:notification:1.0"
xmlns:ns1="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:ns3="http://www.cisco.com/esc/esc_notifications"
xmlns:ns0="http://www.cisco.com/esc/esc" xmlns="http://www.cisco.com/esc/esc">
  <version>1.0.0</version>
      . . .
     <vm_group>
        </interfaces>
          <interface>
            <network>1fbf9fc2-3074-4ae6-bb0a-09d526fbada6</network>
            <nicid>1</nicid>
            <ip_address>10.0.0.0</ip_address>
          </interface>
        </interfaces>
       <scaling>
          <min_active>10</min_active>
          <max_active>15</max_active>
          <elastic>true</elastic>
          <static_ip_address_pool>
            <network>1fbf9fc2-3074-4ae6-bb0a-09d526fbada6</network>
            <gateway>192.168.0.1</gateway> <!-- not used -->
            <netmask>255.255.255.0</netmask> <!-- not used -->
            <ip_address>10.0.0.0</ip_address>
          </static_ip_address_pool>
        </scaling>
```

The table below shows some more scenarios on updating the minimum and maximum values in the scaling section.

Table 16: Updating the Number of VMs in a Deployment

Scenario	Old Value	New Value	Active Value
If the initial number of VMs are a minimum of 2 and maximum of 5 in the scaling section, updating the minimum number of VMs to 3 would create one additional VM. This assumes that the active number of VMs remains at 2.	The old minimum number of VMs is 2.	The new minimum number of VMs is 3.	The active number of VMs is 2.

Scenario	Old Value	New Value	Active Value
If the initial number of VMs is a minimum value of 2 and maximum value of 5, then updating the minimum value to 3 would update the database but will not impact the deployment. This scenario will occur if the original deployment has scaled creating one additional VM.	The old minimum value is 2.	The new minimum value is 3.	The active count is 3.
If the initial number of VMs is a minimum of 2 and maximum of 5, then updating the minimum value to 1 will update the database but will not impact the deployment. Having an active number of VMs greater than the minimum value is a valid deployment as the number of active VMs falls within the minimum or maximum range.	The old minimum value is 2.	The new minimum value is 1.	The active number of VMs is 2.
If the initial number of VMs is a minimum of 2 and maximum of 5, then updating the maximum to 6 will update the database but will not impact the deployment. Having an active number of VMs lesser than the maximum value is a valid deployment as the number of active VMs falls within the minimum or maximum range.	The old maximum value is 5.	The new maximum value is 6.	The active number of VMs is 2.

Scenario	Old Value	New Value	Active Value
If the initial number of VMs is a minimum of 2 and maximum of 5, then updating the maximum value to 4 will update the database but will not have any impact on the deployment. Having an active VM count lesser than the maximum value is a valid deployment as the number of active VMs falls within the minimum or maximum range.	The old maximum value is 5.	The new maximum value is 4.	The active number of VMs is 2.
If the initial number of VMs is a minimum of 2 and maximum of 5, then updating the maximum number of VMs to 4 will update the database and remove one VM from the deployment. The last VM created will be removed bringing the active and maximum count down to 4.	The old maximum value is 5.	The new maximum value is 4.	The active number of VMs is 4.

If static IPs are used, adding more VMs to a deployment needs update to the scaling pool section.

The deployment datamodel is as follows:

```
<esc_datamodel xmlns:ns2="urn:ietf:params:xml:ns:netconf:notification:1.0"
xmlns:ns1="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:ns3="http://www.cisco.com/esc/esc_notifications"
xmlns:ns0="http://www.cisco.com/esc/esc" xmlns="http://www.cisco.com/esc/esc">
  <version>1.0.0</version>
      . . .
     <vm_group>
        </interfaces>
          <interface>
            <network>1fbf9fc2-3074-4ae6-bb0a-09d526fbada6</network>
            <nicid>1</nicid>
            <ip_address>23.23.23.23</ip_address>
          </interface>
        </interfaces>
       <scaling>
          <min_active>1</min_active>
          <max_active>1</max_active>
          <elastic>true</elastic>
          <static_ip_address_pool>
            <network>1fbf9fc2-3074-4ae6-bb0a-09d526fbada6</network>
            <gateway>192.168.0.1</gateway> <!- not used ->
            <netmask>255.255.255.0</netmask> <!- not used ->
            <ip_address>23.23.23.23</ip_address>
```

```
          </static_ip_address_pool>
        </scaling>
```

Pools are linked to interfaces through network id. The updated datamodel is as follows:

```
Update payload
<esc_datamodel xmlns:ns2="urn:ietf:params:xml:ns:netconf:notification:1.0"
xmlns:ns1="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:ns3="http://www.cisco.com/esc/esc_notifications"
xmlns:ns0="http://www.cisco.com/esc/esc" xmlns="http://www.cisco.com/esc/esc">
  <version>1.0.0</version>
      . . .
     <vm_group>
        <interfaces>
          <interface>
            <network>1fbf9fc2-3074-4ae6-bb0a-09d526fbada6</network>
            <nicid>1</nicid>
            <ip_address>23.23.23.23</ip_address>
          </interface>
        </interfaces>
        <scaling>
          <min_active>2</min_active>
          <max_active>2</max_active>
          <elastic>true</elastic>
          <static_ip_address_pool>
              <network>1fbf9fc2-3074-4ae6-bb0a-09d526fbada6</network>
              <gateway>192.168.0.1</gateway>
              <netmask>255.255.255.0</netmask>
              <ip_address>10.0.0.0</ip_address>
              <ip_address>10.0.0.24</ip_address>
          </static_ip_address_pool>
        </scaling>
```

The first IP is also included in the update datamodel. If a value is not present in the update list it will be removed from the pool. This results in creating a single VM using the IP address 10.0.0.24.

Note You cannot remove a specific VM from the deployment.

Updating the Recovery Wait Time

You can now update the recovery wait time in an existing deployment. In the example below, the <recovery_wait_time> parameter is set to 60 seconds during the initial deployment.

```
<vm_group>
 <name>CSR</name>
 <recovery_wait_time>60</recovery_wait_time>
```

The recovery wait time is updated to 100 seconds in the existing deployment.

```
<vm_group>
 <name>CSR</name>
 <recovery_wait_time>100</recovery_wait_time>
```

Updating the recovery wait time impacts the VMs created in the existing deployment.

After receiving a VM_DOWN event, recovery wait time allows ESC to wait for a certain amount of time before proceeding with the VM recovery workflow. The time allocated for recovery wait time allows the VM to restore network connectivity or heal itself. If a VM_ALIVE is triggered within this time, VM recovery is canceled.

Updating the Recovery Policy

You can add the recovery policy, or update the existing recovery policy parameters while updating a deployment.

Auto recovery is triggered automatically without notification. For manual recovery, the VM_MANUAL_RECOVERY_NEEDED notification is sent, and the recovery starts only if the user sends command.

When the recovery type is set to auto, the recovery starts automatically without notification. When the recovery type is set to manual, the VM_MANUAL_RECOVERY_NEEDED notification is sent, and the recovery starts only if the user sends command.

In the example below, the recovery action is set to REBOOT_THEN_REDEPLOY during initial deployment. It is updated to REBOOT_ONLY during the deployment update. If the recovery is not successful, the maximum number of retries is 1 in the initial deployment. You can update the maximum retries as well in an existing deployment. In the example below, the maximum number of retries is updated to 3.

Initial Deployment

```
<recovery_policy>
           <action_on_recovery>REBOOT_THEN_REDEPLOY</action_on_recovery>
              <max_retries>1</max_retries>
         </recovery_policy>
```

Deployment Update

```
<recovery_policy>
           <action_on_recovery>REBOOT_ONLY</action_on_recovery>
            <max_retries>3</max_retries>
        </recovery_policy>
```

The recovery policy notification is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2017-06-21T12:35:12.354+00:00</eventTime>
  <escEvent xmlns="http://www.cisco.com/esc/esc">
    <status>SUCCESS</status>
    <status_code>200</status_code>
    <status_message>Service group update completed successfully</status_message>
    <depname>jenkins-update-recovery-success-dep-201102</depname>
    <tenant>jenkins-update-recovery-success-tenant-201102</tenant>
    <tenant_id>11ade63bac8a4010a969df0d0b91b9bf</tenant_id>
    <depid>574b2e11-61a9-4d9b-83b1-e95a3aa56fdd</depid>
    <event>
      <type>SERVICE_UPDATED</type>
    </event>
  </escEvent>
</notification>
```

During the deployment update, a recovery policy cannot be overwritten with LCS. For example, a recovery policy with REBOOT_ONLY cannot be overwritten with lifecycle stage (LCS).

Updating an Image

You can update the image reference of VMs in an existing deployment.

The datamodel update is as follows:

Existing datamodel:

```
<recovery_wait_time>30</recovery_wait_time>
```

```
<flavor>Automation-Cirros-Flavor</flavor>
<image>Automation-Cirros-Image</image>
```

New datamodel:

```
<recovery_wait_time>30</recovery_wait_time>
<flavor>Automation-Cirros-Flavor</flavor>
<image>Automation-CSR-Image-3_14</image>
```

You receive a service update notification after the image is updated.

```
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
<eventTime>2018-05-10T17:34:00.605+00:00</eventTime>
<escEvent xmlns="http://www.cisco.com/esc/esc">
<status>SUCCESS</status>
<status_code>200</status_code>
<status_message>Service group update completed successfully</status_message>
<depname>ud-A</depname>
<tenant>ut-AM</tenant>
<tenant_id>24e21e581ad441ebbb3bd22e69c36322</tenant_id>
<depid>e009b1cc-0aa9-4abd-8aac-265be7f9a80d</depid>
<event>
<type>SERVICE_UPDATED</type>
</event>
</escEvent>
</notification>
```

The new image reference appears in the opdata:

```
<vm_group>
<name>ug-1</name>
<flavor>m1.large</flavor>
<image>cirror</image>
<vm_instance>
<vm_id>9a63afed-c70f-4827-91e2-72bdd86c5e39</vm_id>
```

If an incorrect image name is provided, then the following error appears:

```
<?xml version="1.0" encoding="UTF-8"?>
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
<eventTime>2018-05-08T19:28:12.321+00:00</eventTime>
<escEvent xmlns="http://www.cisco.com/esc/esc">
<status>FAILURE</status>
<status_code>500</status_code>
<status_message>Error during service update: Failed to [Update] deployment: The image
Automation-1-Cirros-Image cannot be found on the virtual infrastructure
manager.</status_message>
<depname>ud-A</depname>
<tenant>ut-AL</tenant>
<tenant_id>4fb19d82c5b34b33aa6162c0b33f07d7</tenant_id>
<depid>6eed6eba-4f3f-401d-83be-91d703ee4946</depid>
<event>
<type>SERVICE_UPDATED</type>
</event>
</escEvent>
</notification>
```

Rollback scenarios for Image Update

You must update the image reference even when the service is in error state so that the image reference gets updated in the subsequent update. The table below lists the image update rollback conditions, the expected behavior and notifications.

Rollback condition	**Expected behavior**	**Notification**
The service is in the ERROR state, and the request has image update only	The image is updated but the service remains in the ERROR state	?xml version="1.0" encoding="UTF-8"?> <notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0"> <eventTime>2018-06-06T13:59:04.331+00:00</eventTime> <escEvent xmlns="http://www.cisco.com/esc/esc"> <status>SUCCESS</status> <status_code>200</status_code> <status_message>Deployment update successful. But one or more VMs are still in ERROR state.</status_message> <depname>ud-A</depname> <tenant>ut-JJ</tenant> <tenant_id>0dbb67d6457642f68520565ce785976a</tenant_id> <depid>0feea6bc-310c-49c8-8416-94f89a324bfb</depid> <event> <type>SERVICE_UPDATED</type> </event> </escEvent> </notification>
Service is in ERROR state and the request is sent to remove the VM group (in error)	The VM group is removed and the service is in ACTIVE state	
The service is in ERROR state. A request to remove the VM group (in error) is sent along with an image update request in the same VM group	The VM group should be removed. There is no impact due to the image update. The service is back to ACTIVE state	

Rollback condition	Expected behavior	Notification
The service is in ERROR state. A request to remove the VM groups (in active) is sent along with the image update in a different VM group (in error)	The VM group (in active) is removed. The image updated in the vm group (in error). the service remains in the ERROR state.	?xml version="1.0" encoding="UTF-8"?> <notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0"> <eventTime>2018-06-06T13:59:04.331+00:00</eventTime> <escEvent xmlns="http://www.cisco.com/esc/esc"> <status>SUCCESS</status> <status_code>200</status_code> <status_message>Deployment update successful. But one or more VMs are still in ERROR state.</status_message> <depname>ud-A</depname> <tenant>ut-JJ</tenant> <tenant_id>0dbb67d6457642f68520565ce785976a</tenant_id> <depid>0feea6bc-310c-49c8-8416-94f89a324bfb</depid> <event> <type>SERVICE_UPDATED</type> </event> </escEvent> </notification>
The service is in the ERROR state. A single VM group is present (in error). The image update request is sent.	The image is updated but the service remains in the ERROR state. The VM group (in error) cannot be removed, as it is the only one in the service. User must undeploy and redeploy.	

Adding a VM Group (vCloud Director)

ESC supports only addition and deletion of VM group(s) in vCD. One or multiple VM group(s) can be added or deleted in a service update.

```
<?xml version="1.0" encoding="UTF-8"?>
<esc_datamodel xmlns="http://www.cisco.com/esc/esc" xmlns:ns0="http://www.cisco.com/esc/esc"
 xmlns:ns1="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:ns2="urn:ietf:params:xml:ns:netconf:notification:1.0"
xmlns:ns3="http://www.cisco.com/esc/esc_notifications">
  <tenants>
    <tenant>
```

```
<!-- ESC scope tenant -->
<name>vnf-dep</name>
<vim_mapping>false</vim_mapping>
<deployments>
  <deployment>
    <!-- vApp instance name -->
    <name>dep</name>
    <policies>
      <placement_group>
        <name>placement-affinity-1</name>
        <type>affinity</type>
        <enforcement>strict</enforcement>
        <vm_group>g1</vm_group>
        <vm_group>g2</vm_group>
        <vm_group>g3</vm_group>
      </placement_group>
    </policies>
    <extensions>
      <extension>
        <name>VMWARE_VCD_PARAMS</name>
        <properties>
          <property>
            <name>CATALOG_NAME</name>
            <value>catalog-1</value>
          </property>
          <property>
            <name>VAPP_TEMPLATE_NAME</name>
            <value>uLinux_vApp_Template</value>
          </property>
        </properties>
      </extension>
    </extensions>
    <vm_group>
      <name>g1</name>
      <locator>
        <!-- vCD vim connector id -->
        <vim_id>vcd</vim_id>
        <!-- vCD orgnization -->
        <vim_project>esc</vim_project>
        <!-- vDC name -->
        <vim_vdc>VDC-1</vim_vdc>
      </locator>
      <!-- VM name in vAppTemplate -->
      <image>vm-001</image>
      <bootup_time>120</bootup_time>
      <recovery_wait_time>5</recovery_wait_time>
      <interfaces>
        <interface>
          <nicid>0</nicid>
          <network>MgtNetwork</network>
          <ip_address>10.0.0.155</ip_address>
          <mac_address>00:1C:B3:09:85:15</mac_address>
        </interface>
      </interfaces>
      <scaling>
        <min_active>1</min_active>
        <max_active>1</max_active>
        <elastic>true</elastic>
        <static_ip_address_pool>
          <network>MgtNetwork</network>
          <ip_address>10.0.0.155</ip_address>
        </static_ip_address_pool>
        <static_mac_address_pool>
          <network>MgtNetwork</network>
```

```
            <mac_address>00:1C:B3:09:85:15</mac_address>
          </static_mac_address_pool>
        </scaling>
        <kpi_data>
          <kpi>
            <event_name>VM_ALIVE</event_name>
            <metric_value>1</metric_value>
            <metric_cond>GT</metric_cond>
            <metric_type>UINT32</metric_type>
            <metric_collector>
              <type>ICMPPing</type>
              <nicid>0</nicid>
              <poll_frequency>30</poll_frequency>
              <polling_unit>seconds</polling_unit>
              <continuous_alarm>false</continuous_alarm>
            </metric_collector>
          </kpi>
        </kpi_data>
        <rules>
          <admin_rules>
            <rule>
              <event_name>VM_ALIVE</event_name>
              <action>"ALWAYS log"</action>
              <action>"TRUE servicebooted.sh"</action>
              <action>"FALSE recover autohealing"</action>
            </rule>
          </admin_rules>
        </rules>
        <config_data>
          <configuration>
            <dst>ovfProperty:mgmt-ipv4-addr</dst>
            <data>$NICID_0_IP_ADDRESS/24</data>
          </configuration>
        </config_data>
        <recovery_policy>
          <action_on_recovery>REBOOT_ONLY</action_on_recovery>
        </recovery_policy>
      </vm_group>
      <vm_group>
        <name>g2</name>
        <locator>
          <!-- vCD vim connector id -->
          <vim_id>vcd</vim_id>
          <!-- vCD orgnization -->
          <vim_project>esc</vim_project>
          <!-- vDC name -->
          <vim_vdc>VDC-1</vim_vdc>
        </locator>
        <!-- VM name in vAppTemplate -->
        <image>vm-002</image>
        <bootup_time>120</bootup_time>
        <recovery_wait_time>5</recovery_wait_time>
        <interfaces>
          <interface>
            <nicid>0</nicid>
            <network>MgtNetwork</network>
            <ip_address>10.0.0.156</ip_address>
            <mac_address>00:1C:B3:09:85:16</mac_address>
          </interface>
        </interfaces>
        <scaling>
          <min_active>1</min_active>
          <max_active>1</max_active>
          <elastic>true</elastic>
```

```
      <static_ip_address_pool>
        <network>MgtNetwork</network>
        <ip_address>10.0.0.156</ip_address>
      </static_ip_address_pool>
      <static_mac_address_pool>
        <network>MgtNetwork</network>
        <mac_address>00:1C:B3:09:85:16</mac_address>
      </static_mac_address_pool>
    </scaling>
    <kpi_data>
      <kpi>
        <event_name>VM_ALIVE</event_name>
        <metric_value>1</metric_value>
        <metric_cond>GT</metric_cond>
        <metric_type>UINT32</metric_type>
        <metric_collector>
          <type>ICMPPing</type>
          <nicid>0</nicid>
          <poll_frequency>30</poll_frequency>
          <polling_unit>seconds</polling_unit>
          <continuous_alarm>false</continuous_alarm>
        </metric_collector>
      </kpi>
    </kpi_data>
    <rules>
      <admin_rules>
        <rule>
          <event_name>VM_ALIVE</event_name>
          <action>"ALWAYS log"</action>
          <action>"TRUE servicebooted.sh"</action>
          <action>"FALSE recover autohealing"</action>
        </rule>
      </admin_rules>
    </rules>
    <config_data>
      <configuration>
        <dst>ovfProperty:mgmt-ipv4-addr</dst>
        <data>$NICID_0_IP_ADDRESS/24</data>
      </configuration>
    </config_data>
    <recovery_policy>
      <action_on_recovery>REBOOT_ONLY</action_on_recovery>
    </recovery_policy>
  </vm_group>
  <vm_group>
    <name>g3</name>
    <locator>
      <!-- vCD vim connector id -->
      <vim_id>vcd</vim_id>
      <!-- vCD orgnization -->
      <vim_project>esc</vim_project>
      <!-- vDC name -->
      <vim_vdc>VDC-1</vim_vdc>
    </locator>
    <!-- VM name in vAppTemplate -->
    <image>vm-002</image>
    <bootup_time>120</bootup_time>
    <recovery_wait_time>5</recovery_wait_time>
    <interfaces>
      <interface>
        <nicid>0</nicid>
        <network>MgtNetwork</network>
        <ip_address>20.0.0.157</ip_address>
        <mac_address>00:1C:B3:09:85:17</mac_address>
```

```
              </interface>
            </interfaces>
            <scaling>
              <min_active>1</min_active>
              <max_active>1</max_active>
              <elastic>true</elastic>
              <static_ip_address_pool>
                <network>MgtNetwork</network>
                <ip_address>10.0.0.157</ip_address>
              </static_ip_address_pool>
              <static_mac_address_pool>
                <network>MgtNetwork</network>
                <mac_address>00:1C:B3:09:85:17</mac_address>
              </static_mac_address_pool>
            </scaling>
            <kpi_data>
              <kpi>
                <event_name>VM_ALIVE</event_name>
                <metric_value>1</metric_value>
                <metric_cond>GT</metric_cond>
                <metric_type>UINT32</metric_type>
                <metric_collector>
                  <type>ICMPPing</type>
                  <nicid>0</nicid>
                  <poll_frequency>30</poll_frequency>
                  <polling_unit>seconds</polling_unit>
                  <continuous_alarm>false</continuous_alarm>
                </metric_collector>
              </kpi>
            </kpi_data>
            <rules>
              <admin_rules>
                <rule>
                  <event_name>VM_ALIVE</event_name>
                  <action>"ALWAYS log"</action>
                  <action>"TRUE servicebooted.sh"</action>
                  <action>"FALSE recover autohealing"</action>
                </rule>
              </admin_rules>
            </rules>
            <config_data>
              <configuration>
                <dst>ovfProperty:mgmt-ipv4-addr</dst>
                <data>$NICID_0_IP_ADDRESS/24</data>
              </configuration>
            </config_data>
            <recovery_policy>
              <action_on_recovery>REBOOT_ONLY</action_on_recovery>
            </recovery_policy>
          </vm_group>
        </deployment>
      </deployments>
    </tenant>
  </tenants>
</esc_datamodel>
```

Deleting a VM Group (vCloud Director)

ESC allows deleting a VM group in vCloud Director:

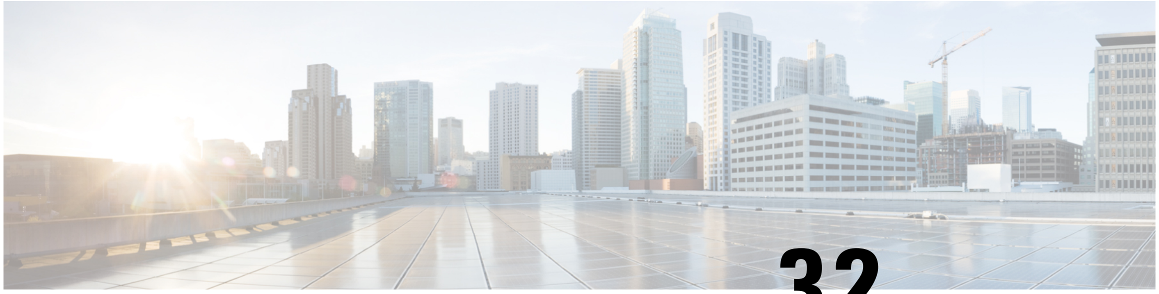
```
<?xml version="1.0" encoding="UTF-8"?>
<esc_datamodel xmlns="http://www.cisco.com/esc/esc" xmlns:nc="http://www.cisco.com/esc/esc"
 xmlns:ns0="http://www.cisco.com/esc/esc" xmlns:ns1="urn:ietf:params:xml:ns:netconf:base:1.0"
 xmlns:ns2="urn:ietf:params:xml:ns:netconf:notification:1.0"
```

```
xmlns:ns3="http://www.cisco.com/esc/esc_notifications">
  <tenants>
    <tenant>
      <!-- ESC scope tenant -->
      <name>vnf-dep</name>
      <vim_mapping>false</vim_mapping>
      <deployments>
        <deployment>
          <!-- vApp instance name -->
          <name>dep</name>
          <policies>
            <placement_group>
              <name>placement-affinity-1</name>
              <type>affinity</type>
              <enforcement>strict</enforcement>
              <vm_group>g1</vm_group>
              <vm_group>g2</vm_group>
              <vm_group nc:operation="delete">g3</vm_group>
            </placement_group>
          </policies>
          <extensions>
            <extension>
              <name>VMWARE_VCD_PARAMS</name>
              <properties>
                <property>
                  <name>CATALOG_NAME</name>
                  <value>catalog-1</value>
                </property>
                <property>
                  <name>VAPP_TEMPLATE_NAME</name>
                  <value>uLinux_vApp_Template</value>
                </property>
              </properties>
            </extension>
          </extensions>
          <vm_group>
            <name>g1</name>
            <locator>
              <!-- vCD vim connector id -->
              <vim_id>vcd</vim_id>
              <!-- vCD orgnization -->
              <vim_project>esc</vim_project>
              <!-- vDC name -->
              <vim_vdc>VDC-1</vim_vdc>
            </locator>
            <!-- VM name in vAppTemplate -->
            <image>vm-001</image>
            <bootup_time>120</bootup_time>
            <recovery_wait_time>5</recovery_wait_time>
            <interfaces>
              <interface>
                <nicid>0</nicid>
                <network>MgtNetwork</network>
                <ip_address>10.0.0.155</ip_address>
                <mac_address>00:1C:B3:09:85:15</mac_address>
              </interface>
            </interfaces>
            <scaling>
              <min_active>1</min_active>
              <max_active>1</max_active>
              <elastic>true</elastic>
              <static_ip_address_pool>
                <network>MgtNetwork</network>
                <ip_address>10.0.0.155</ip_address>
```

```
      </static_ip_address_pool>
      <static_mac_address_pool>
        <network>MgtNetwork</network>
        <mac_address>00:1C:B3:09:85:15</mac_address>
      </static_mac_address_pool>
    </scaling>
    <kpi_data>
      <kpi>
        <event_name>VM_ALIVE</event_name>
        <metric_value>1</metric_value>
        <metric_cond>GT</metric_cond>
        <metric_type>UINT32</metric_type>
        <metric_collector>
          <type>ICMPPing</type>
          <nicid>0</nicid>
          <poll_frequency>30</poll_frequency>
          <polling_unit>seconds</polling_unit>
          <continuous_alarm>false</continuous_alarm>
        </metric_collector>
      </kpi>
    </kpi_data>
    <rules>
      <admin_rules>
        <rule>
          <event_name>VM_ALIVE</event_name>
          <action>"ALWAYS log"</action>
          <action>"TRUE servicebooted.sh"</action>
          <action>"FALSE recover autohealing"</action>
        </rule>
      </admin_rules>
    </rules>
    <config_data>
      <configuration>
        <dst>ovfProperty:mgmt-ipv4-addr</dst>
        <data>$NICID_0_IP_ADDRESS/24</data>
      </configuration>
    </config_data>
    <recovery_policy>
      <action_on_recovery>REBOOT_ONLY</action_on_recovery>
    </recovery_policy>
  </vm_group>
  <vm_group>
    <name>g2</name>
    <locator>
      <!-- vCD vim connector id -->
      <vim_id>vcd</vim_id>
      <!-- vCD orgnization -->
      <vim_project>esc</vim_project>
      <!-- vDC name -->
      <vim_vdc>VDC-1</vim_vdc>
    </locator>
    <!-- VM name in vAppTemplate -->
    <image>vm-002</image>
    <bootup_time>120</bootup_time>
    <recovery_wait_time>5</recovery_wait_time>
    <interfaces>
      <interface>
        <nicid>0</nicid>
        <network>MgtNetwork</network>
        <ip_address>10.0.0.156</ip_address>
        <mac_address>00:1C:B3:09:85:16</mac_address>
      </interface>
    </interfaces>
    <scaling>
```

```
      <min_active>1</min_active>
      <max_active>1</max_active>
      <elastic>true</elastic>
      <static_ip_address_pool>
        <network>MgtNetwork</network>
        <ip_address>10.0.0.156</ip_address>
      </static_ip_address_pool>
      <static_mac_address_pool>
        <network>MgtNetwork</network>
        <mac_address>00:1C:B3:09:85:16</mac_address>
      </static_mac_address_pool>
    </scaling>
    <kpi_data>
      <kpi>
        <event_name>VM_ALIVE</event_name>
        <metric_value>1</metric_value>
        <metric_cond>GT</metric_cond>
        <metric_type>UINT32</metric_type>
        <metric_collector>
          <type>ICMPPing</type>
          <nicid>0</nicid>
          <poll_frequency>30</poll_frequency>
          <polling_unit>seconds</polling_unit>
          <continuous_alarm>false</continuous_alarm>
        </metric_collector>
      </kpi>
    </kpi_data>
    <rules>
      <admin_rules>
        <rule>
          <event_name>VM_ALIVE</event_name>
          <action>"ALWAYS log"</action>
          <action>"TRUE servicebooted.sh"</action>
          <action>"FALSE recover autohealing"</action>
        </rule>
      </admin_rules>
    </rules>
    <config_data>
      <configuration>
        <dst>ovfProperty:mgmt-ipv4-addr</dst>
        <data>$NICID_0_IP_ADDRESS/24</data>
      </configuration>
    </config_data>
    <recovery_policy>
      <action_on_recovery>REBOOT_ONLY</action_on_recovery>
    </recovery_policy>
  </vm_group>
  <vm_group nc:operation="delete">
    <name>g3</name>
    <locator>
      <!-- vCD vim connector id -->
      <vim_id>vcd</vim_id>
      <!-- vCD orgnization -->
      <vim_project>esc</vim_project>
      <!-- vDC name -->
      <vim_vdc>VDC-1</vim_vdc>
    </locator>
    <!-- VM name in vAppTemplate -->
    <image>vm-002</image>
    <bootup_time>120</bootup_time>
    <recovery_wait_time>5</recovery_wait_time>
    <interfaces>
      <interface>
        <nicid>0</nicid>
```

```
                <network>MgtNetwork</network>
                <ip_address>10.0.0.157</ip_address>
                <mac_address>00:1C:B3:09:85:17</mac_address>
              </interface>
            </interfaces>
            <scaling>
              <min_active>1</min_active>
              <max_active>1</max_active>
              <elastic>true</elastic>
              <static_ip_address_pool>
                <network>MgtNetwork</network>
                <ip_address>10.0.0.157</ip_address>
              </static_ip_address_pool>
              <static_mac_address_pool>
                <network>MgtNetwork</network>
                <mac_address>00:1C:B3:09:85:17</mac_address>
              </static_mac_address_pool>
            </scaling>
            <kpi_data>
              <kpi>
                <event_name>VM_ALIVE</event_name>
                <metric_value>1</metric_value>
                <metric_cond>GT</metric_cond>
                <metric_type>UINT32</metric_type>
                <metric_collector>
                  <type>ICMPPing</type>
                  <nicid>0</nicid>
                  <poll_frequency>30</poll_frequency>
                  <polling_unit>seconds</polling_unit>
                  <continuous_alarm>false</continuous_alarm>
                </metric_collector>
              </kpi>
            </kpi_data>
            <rules>
              <admin_rules>
                <rule>
                  <event_name>VM_ALIVE</event_name>
                  <action>"ALWAYS log"</action>
                  <action>"TRUE servicebooted.sh"</action>
                  <action>"FALSE recover autohealing"</action>
                </rule>
              </admin_rules>
            </rules>
            <config_data>
              <configuration>
                <dst>ovfProperty:mgmt-ipv4-addr</dst>
                <data>$NICID_0_IP_ADDRESS/24</data>
              </configuration>
            </config_data>
            <recovery_policy>
              <action_on_recovery>REBOOT_ONLY</action_on_recovery>
            </recovery_policy>
          </vm_group>
        </deployment>
      </deployments>
    </tenant>
  </tenants>
</esc_datamodel>
```



CHAPTER 32

Migrating VNF in CSP Cluster

You can deploy, update, or migrate VNFs/VMs on a CSP cluster. The VNFs within the CSP cluster can be migrated from one cluster to another within the CSP Cluster.

Migrating VNF in CSP Cluster

Scenario 1

Migrate the VM from CSP-1 to CSP-2, when CSP-1 is reachable.

To migrate the VM from CSP-1 to CSP-2, NB sends an update to ESC by changing the locator (vim_id, vim_project) when CSP-1 is reachable.

The following sample shows the VM group from the deployment payload/XML:

```
<vm_group>
            <name>Group1</name>
            <locator>
              <vim_id>CSP-1</vim_id>
              <vim_project>CSP-1</vim_project>
            </locator>
```

The following sample shows migration success notification of VM on CSP-2:

```
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2020-09-03T05:41:16.299+00:00</eventTime>
  <escEvent xmlns="http://www.cisco.com/esc/esc">
    <status>SUCCESS</status>
    <status_code>200</status_code>
    <status_message>VIM Locator Updated Successfully</status_message>
    <vm_update_type>LOCATOR_UPDATED</vm_update_type>
    <depname>dep</depname>
    <tenant>demo</tenant>
    <depid>06c94f58-b753-425b-b97c-f7adb9140ead</depid>
    <vm_group>group</vm_group>
    <vm_source>
      <vmid>6b0e7179-fd5e-487e-9570-e7ba98cce0ec</vmid>
      <vmname>dep_group_0_46e607a8-b797-4056-96f3-42a90a63b555</vmname>
     <generated_vmname>dep_group_0_46e607a8-b797-4056-96f3-42a90a63b555</generated_vmname>

      <vim_id>CSP-2</vim_id>
      <vim_project>CSP-2</vim_project>
      <interfaces>
        <interface>
```

```
          <nicid>0</nicid>
          <type>access</type>
          <port_id>539c6df4-4680-4bba-8a0d-d621947f2228</port_id>
          <admin_state_up>true</admin_state_up>
          <network>Eth0-2</network>
          <subnet/>
          <ip_address>192.168.23.62</ip_address>
          <netmask>255.255.255.0</netmask>
        </interface>
        <interface>
          <nicid>1</nicid>
          <type>trunk</type>
          <port_id>0adc3096-509c-49b7-9bd7-a25bbf2a9345</port_id>
          <admin_state_up>true</admin_state_up>
          <network>Eth0-2</network>
          <subnet/>
        </interface>
      </interfaces>
    </vm_source>
    <event>
      <type>VM_UPDATED</type>
    </event>
  </escEvent>
</notification>

<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2020-09-03T05:41:16.322+00:00</eventTime>
  <escEvent xmlns="http://www.cisco.com/esc/esc">
    <status>SUCCESS</status>
    <status_code>200</status_code>
    <status_message>Service group update completed successfully</status_message>
    <depname>dep</depname>
    <tenant>demo</tenant>
    <tenant_id>demo</tenant_id>
    <depid>06c94f58-b753-425b-b97c-f7adb9140ead</depid>
    <event>
      <type>SERVICE_UPDATED</type>
    </event>
  </escEvent>
```

Scenario 2

Migrate the VM from CSP-1 to CSP-2, when CSP-1 is not reachable.

Assuming the recovery mode is auto and recovery policy as REBOOT_ONLY during initial deployment. Consider CSP-1 host fails and ESC detects the VM failed due to CSP-1 failure. ESC tries to recover the VM but fails because CSP-1 is down. Now the NB sends an update to move the VM from CSP-1 to CSP-2.

The following sample shows recovery failure notification of VM on CSP-1:

```
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2020-09-03T04:30:18.642+00:00</eventTime>
  <escEvent xmlns="http://www.cisco.com/esc/esc">
    <status>SUCCESS</status>
    <status_code>200</status_code>
    <status_message>Recovery event for VM Generated ID
[dep_group_0_46e607a8-b797-4056-96f3-42a90a63b555] triggered.</status_message>
    <depname>dep</depname>
    <tenant>demo</tenant>
    <tenant_id>demo</tenant_id>
    <depid>06c94f58-b753-425b-b97c-f7adb9140ead</depid>
    <vm_group>group</vm_group>
    <vm_source>
      <vmid>6b0e7179-fd5e-487e-9570-e7ba98cce0ec</vmid>
      <vmname>dep_group_0_46e607a8-b797-4056-96f3-42a90a63b555</vmname>
```

```
      <generated_vmname>dep_group_0_46e607a8-b797-4056-96f3-42a90a63b555</generated_vmname>

       <vim_id>CSP-1</vim_id>
       <vim_project>CSP-1</vim_project>
     </vm_source>
     <event>
       <type>VM_RECOVERY_INIT</type>
     </event>
   </escEvent>
 </notification>

 <notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
   <eventTime>2020-09-03T04:31:20.449+00:00</eventTime>
   <escEvent xmlns="http://www.cisco.com/esc/esc">
     <status>FAILURE</status>
     <status_code>500</status_code>
     <status_message> VM [dep_group_0_46e607a8-b797-4056-96f3-42a90a63b555] failed to be
 rebooted.</status_message>
     <depname>dep</depname>
     <tenant>demo</tenant>
     <tenant_id>demo</tenant_id>
     <depid>06c94f58-b753-425b-b97c-f7adb9140ead</depid>
     <vm_group>group</vm_group>
     <vm_source>
       <vmid>6b0e7179-fd5e-487e-9570-e7ba98cce0ec</vmid>
       <vmname>dep_group_0_46e607a8-b797-4056-96f3-42a90a63b555</vmname>
      <generated_vmname>dep_group_0_46e607a8-b797-4056-96f3-42a90a63b555</generated_vmname>

       <vim_id>CSP-2</vim_id>
       <vim_project>CSP-2</vim_project>
     </vm_source>
     <event>
       <type>VM_RECOVERY_REBOOT</type>
     </event>
   </escEvent>
 </notification>


 <notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
   <eventTime>2020-09-03T04:41:20.844+00:00</eventTime>
   <escEvent xmlns="http://www.cisco.com/esc/esc">
     <status>FAILURE</status>
     <status_code>500</status_code>
     <status_message>Recovery: Recovery completed with errors for VM:
 [dep_group_0_46e607a8-b797-4056-96f3-42a90a63b555]</status_message>
     <depname>dep</depname>
     <tenant>demo</tenant>
     <tenant_id>demo</tenant_id>
     <depid>06c94f58-b753-425b-b97c-f7adb9140ead</depid>
     <vm_group>group</vm_group>
     <vm_source>
       <vmid>6b0e7179-fd5e-487e-9570-e7ba98cce0ec</vmid>
       <vmname>dep_group_0_46e607a8-b797-4056-96f3-42a90a63b555</vmname>
      <generated_vmname>dep_group_0_46e607a8-b797-4056-96f3-42a90a63b555</generated_vmname>

       <vim_id>CSP-1</vim_id>
       <vim_project>CSP-1</vim_project>
     </vm_source>
     <vm_target>
       <vmname>dep_group_0_46e607a8-b797-4056-96f3-42a90a63b555</vmname>
      <generated_vmname>dep_group_0_46e607a8-b797-4056-96f3-42a90a63b555</generated_vmname>

     </vm_target>
     <event>
```

```
      <type>VM_RECOVERY_COMPLETE</type>
    </event>
  </escEvent>
</notification>
```

Consider, you have a cluster of three CSPs (CSP-1,CSP-2, and CSP-3). VM is deployed on CSP-1

Before you begin

- A VIM Connector must be created. For more information, See Adding VIM Connector to CSP Cluster Chapter.
- VM is deployed with the same underlying storage. For more information, See Deploying VNFs Using ESC on CSP Cluster Chapter.

The following scenarios show the migration of the VMs:

Procedure

Step 1 Update locator details in the followingdeployment payload

```
(Update vim_id, vim_project to CSP-1 → CSP-2)
<locator>
<vim_id>CSP-2</vim_id>
<vim_project>CSP-2</vim_project>
</locator>
```

Step 2 Execute the following command for migrating the VM:

```
esc_nc_cli --user <username> --password <password> edit-config deploy_csp_2.xml
```

Example payload:

```
deploy_csp_1.xml
<esc_datamodel xmlns="http://www.cisco.com/esc/esc">
  <flavors>
    <flavor>
      <name>FLAVOR_2_4096_10000</name>
      <vcpus>2</vcpus>
      <memory_mb>4096</memory_mb>
      <root_disk_mb>10240</root_disk_mb>
    </flavor>
  </flavors>
  <tenants>
    <tenant>
      <name>name1</name>
      <vim_mapping>false</vim_mapping>
      <deployments>
        <deployment>
          <name>dep</name>
          <vm_group>
            <name>Group1</name>
            <locator>
              <vim_id>CSP-1</vim_id>
              <vim_project>CSP-1</vim_project>
            </locator>
            <image>csr1000v-universalk9.16.06.01.qcow2</image>
            <flavor>FLAVOR_2_4096_10000</flavor>
            <bootup_time>600</bootup_time>
            <recovery_wait_time>60</recovery_wait_time>
        <recovery_policy>
```

```
        <recovery_type>AUTO</recovery_type>
        <action_on_recovery>REBOOT_ONLY</action_on_recovery>
        <max_retries>1</max_retries>
</recovery_policy>
    <interfaces>
       <interface>
        <nicid>0</nicid>
        <type>virtual</type>
        <model>virtio</model>
        <network>Eth0-2</network>
        <ip_address>192.168.23.61</ip_address>
        </interface>
     <interface>
        <nicid>1</nicid>
        <type>virtual</type>
        <model>virtio</model>
        <network>Eth0-2</network>
          <ip_address>192.168.23.61</ip_address>
        <admin_state_up>false</admin_state_up>
      </interface>
    </interfaces>
    <kpi_data>
      <kpi>
        <event_name>VM_ALIVE</event_name>
        <metric_value>50</metric_value>
        <metric_cond>GT</metric_cond>
        <metric_type>UINT32</metric_type>
        <metric_occurrences_true>3</metric_occurrences_true>
        <metric_occurrences_false>3</metric_occurrences_false>
        <metric_collector>
          <type>ICMPPing</type>
          <nicid>0</nicid>
          <poll_frequency>15</poll_frequency>
          <polling_unit>seconds</polling_unit>
          <continuous_alarm>false</continuous_alarm>
        </metric_collector>
      </kpi>
    </kpi_data>
    <rules>
      <admin_rules>
        <rule>
          <event_name>VM_ALIVE</event_name>
          <action>ALWAYS log</action>
          <action>FALSE recover autohealing</action>
          <action>TRUE servicebooted.sh</action>
        </rule>
      </admin_rules>
    </rules>
    <config_data>
      <configuration>
        <dst>iosxe_config.txt</dst>
        <file>file:///var/tmp/csr_config.sh</file>
      </configuration>
    </config_data>
    <scaling>
      <min_active>1</min_active>
      <max_active>1</max_active>
      <elastic>true</elastic>
      <static_ip_address_pool>
        <network>Eth0-2</network>
        <netmask>255.255.255.0</netmask>
         <gateway>192.168.23.1</gateway>
         <ip_address>192.168.23.61</ip_address>
      </static_ip_address_pool>
```

```
</scaling>
<extensions>
  <extension>
    <name>interfaces</name>
    <containers>
       <container>
        <name>0</name>
        <properties>
          <property>
            <name>passthroughMode</name>
            <value>none</value>
          </property>
          <property>
            <name>tagged</name>
            <value>false</value>
          </property>
          <property>
            <name>type</name>
            <value>access</value>
          </property>
          <property>
            <name>vlan</name>
            <value>1</value>
          </property>
        </properties>
      </container>
     <container>
        <name>1</name>
        <properties>
          <property>
            <name>passthroughMode</name>
            <value>none</value>
          </property>
          <property>
            <name>tagged</name>
            <value>false</value>
          </property>
          <property>
            <name>type</name>
            <value>access</value>
          </property>
           <property>
            <name>bandwidth</name>
            <value>160</value>
          </property>
          <property>
            <name>vlan</name>
           <value>2</value>
          </property>
        </properties>
      </container>
    </containers>
  </extension>
  <extension>
    <name>serial_ports</name>
    <containers>
      <container>
        <name>0</name>
        <properties>
          <property>
            <name>serial_type</name>
            <value>console</value>
          </property>
        </properties>
```

```
                  </container>
                </containers>
              </extension>
              <extension>
                <name>image</name>
                <properties>
                  <property>
                    <name>disk-resize</name>
                    <value>true</value>
                  </property>
                  <property>
                    <name>disk_type</name>
                    <value>virtio</value>
                  </property>
                  <property>
                    <name>disk_storage_name</name>
                    <value>gluster</value>
                  </property>
                </properties>
              </extension>
            </extensions>
          </vm_group>
        </deployment>
      </deployments>
    </tenant>
  </tenants>
</esc_datamodel>
```

Step 3 Check the notification for the success or failure of migration.

```
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2020-09-03T05:41:16.299+00:00</eventTime>
  <escEvent xmlns="http://www.cisco.com/esc/esc">
    <status>SUCCESS</status>
    <status_code>200</status_code>
    <status_message>VIM Locator Updated Successfully</status_message>
    <vm_update_type>LOCATOR_UPDATED</vm_update_type>
    <depname>dep</depname>
    <tenant>demo</tenant>
    <depid>06c94f58-b753-425b-b97c-f7adb9140ead</depid>
    <vm_group>group</vm_group>
    <vm_source>
      <vmid>6b0e7179-fd5e-487e-9570-e7ba98cce0ec</vmid>
      <vmname>dep_group_0_46e607a8-b797-4056-96f3-42a90a63b555</vmname>
     <generated_vmname>dep_group_0_46e607a8-b797-4056-96f3-42a90a63b555</generated_vmname>

      <vim_id>CSP-2</vim_id>
      <vim_project>CSP-2</vim_project>
      <interfaces>
        <interface>
          <nicid>0</nicid>
          <type>access</type>
          <port_id>539c6df4-4680-4bba-8a0d-d621947f2228</port_id>
          <admin_state_up>true</admin_state_up>
          <network>Eth0-2</network>
          <subnet/>
          <ip_address>192.168.23.62</ip_address>
          <netmask>255.255.255.0</netmask>
        </interface>
        <interface>
          <nicid>1</nicid>
          <type>trunk</type>
          <port_id>0adc3096-509c-49b7-9bd7-a25bbf2a9345</port_id>
          <admin_state_up>true</admin_state_up>
```

```
          <network>Eth0-2</network>
          <subnet/>
        </interface>
      </interfaces>
    </vm_source>
    <event>
      <type>VM_UPDATED</type>
    </event>
  </escEvent>
</notification>

<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2020-09-03T05:41:16.322+00:00</eventTime>
  <escEvent xmlns="http://www.cisco.com/esc/esc">
    <status>SUCCESS</status>
    <status_code>200</status_code>
    <status_message>Service group update completed successfully</status_message>
    <depname>dep</depname>
    <tenant>demo</tenant>
    <tenant_id>demo</tenant_id>
    <depid>06c94f58-b753-425b-b97c-f7adb9140ead</depid>
    <event>
      <type>SERVICE_UPDATED</type>
    </event>
  </escEvent>
</notification>
```

Scenario 3

Migrate the VM from Local to Gluster storage on the same CSP.

To migrate the VM from local to gluster, the NB sends an update with the following properties:

a) Add a new VIM connector with cluster name.

```
<property>
           <name>cluster_name</name>
           <value>Cluster_Test</value>
           </property>
```

For more information on adding a new VIM connector, see Adding VIM Connector to CSP Cluster.

b) After adding the new connector, update locator and disk_storage_name to **gluster** in deployment payload to enable configuration change.

Following example shows how to add disk_storage_name as gluster under image extension properties and updating with the cluster VIM Connector:

```
<vm_group>
<name>Group1</name>
<locator>
<vim_id>CSP-1</vim_id>
<vim_project>CSP-1</vim_project>
</locator>
<extension>
                <name>image</name>
                <properties>
                  <property>
                    <name>disk-resize</name>
                    <value>true</value>
                  </property>
                  <property>
                    <name>disk_type</name>
                    <value>virtio</value>
                  </property>
```

```
                <property>
                  <name>disk_storage_name</name>
                  <value>gluster</value>
                </property>
              </properties>
            </extension>
```

c) Execute the following command to deploy the VIM:

```
esc_nc_cli --user <username> --password <password> edit-config deploy.xml
```

Check the following notification for success/failure migration.

```
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2020-09-03T05:41:16.299+00:00</eventTime>
  <escEvent xmlns="http://www.cisco.com/esc/esc">
    <status>SUCCESS</status>
    <status_code>200</status_code>
    <status_message>VIM Locator Updated Successfully</status_message>
    <vm_update_type>LOCATOR_UPDATED</vm_update_type>
    <depname>dep</depname>
    <tenant>demo</tenant>
    <depid>06c94f58-b753-425b-b97c-f7adb9140ead</depid>
    <vm_group>group</vm_group>
    <vm_source>
      <vmid>6b0e7179-fd5e-487e-9570-e7ba98cce0ec</vmid>
      <vmname>dep_group_0_46e607a8-b797-4056-96f3-42a90a63b555</vmname>

<generated_vmname>dep_group_0_46e607a8-b797-4056-96f3-42a90a63b555</generated_vmname>
      <vim_id>CSP-2</vim_id>
      <vim_project>CSP-2</vim_project>
      <interfaces>
        <interface>
          <nicid>0</nicid>
          <type>access</type>
          <port_id>539c6df4-4680-4bba-8a0d-d621947f2228</port_id>
          <admin_state_up>true</admin_state_up>
          <network>Eth0-2</network>
          <subnet/>
          <ip_address>192.168.23.62</ip_address>
          <netmask>255.255.255.0</netmask>
        </interface>
        <interface>
          <nicid>1</nicid>
          <type>trunk</type>
          <port_id>0adc3096-509c-49b7-9bd7-a25bbf2a9345</port_id>
          <admin_state_up>true</admin_state_up>
          <network>Eth0-2</network>
          <subnet/>
        </interface>
      </interfaces>
    </vm_source>
    <event>
      <type>VM_UPDATED</type>
    </event>
  </escEvent>
</notification>

<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2020-09-03T05:41:16.322+00:00</eventTime>
  <escEvent xmlns="http://www.cisco.com/esc/esc">
    <status>SUCCESS</status>
    <status_code>200</status_code>
    <status_message>Service group update completed successfully</status_message>
    <depname>dep</depname>
```

```
      <tenant>demo</tenant>
      <tenant_id>demo</tenant_id>
      <depid>06c94f58-b753-425b-b97c-f7adb9140ead</depid>
      <event>
        <type>SERVICE_UPDATED</type>
      </event>
    </escEvent>
  </notification>
```

CHAPTER 33

Deployment States and Events

ESC deployment lifecycle is represented using various states. The datamodel defines various states the service and VNF will be in during the deployment lifecycle. In general, the deployment or service life cycle is represented in two stages. The service contains one of more different type of vm groups. The vm group represents a group of same type of VM or VNF. After receiving a deployment or service request, ESC validates the request and accepts the request for processing. During processing, ESC deploys the VM or VNF in the underlying VIM using the resources defined in the data model. ESC monitors these VM\VNF based on the kpi and actions defined. As defined by configured policies and actions, ESC triggers auto healing, scale in, scale out and other workflows.

During deployment or any other workflow, the service or deployment's state and VM or VNF state changes and events are sent. The state and events play a key role in identifying the status of the deployment. The current state of the deployment is represented in the operational data. ESC sends the notifications or events when a deployment, or VM or VNF state change that needs to be notified. In the datamodel all the different states and events are defined.

- Deployment or Service States, on page 257
- Event Notifications or Callback Events, on page 259

Deployment or Service States

The service state represents the state of the full service or deployment. The state of the service also depends on the various VM or VNF states, state of the VM in the vm groups, and the current workflow that is running on the service or the VM or VNF. The service or deployment state is an aggregate summary of the whole deployment.

Table 17: Deployment or Service States

Service State	Description
SERVICE_UNDEF_STATE	The initial service state. Service will be in this state until ESC starts processing the deployment.
SERVICE_DEPLOYING_STATE	In this state, VMs are being deployed for this service or deployment.
SERVICE_INERT_STATE	In this state, VMs under this deployment are deployed but are still not active or booted up.

Service State	Description
SERVICE_ACTIVE_STATE	In this state, all the VMs under this deployment are deployed and alive.
SERVICE_ERROR_STATE	Service will be in this state if any error happened during the deployment, recovery, scale in or scale out, or any other workflow.
SERVICE_UNDEPLOYING_STATE	In this state, VM are being undeployed for this service or deployment.
SERVICE_STOPPING_STATE	In this state, the VM or VNF under the service are being stopped due to service action request.
SERVICE_STOPPED_STATE	In this state, the VM or VNF under the service are stopped due to service action request.
SERVICE_STARTING_STATE	In this state, the VM or VNF under the service are starting due to service action request.
SERVICE_REBOOTING_STATE	In this state, the VM or VNF under the service are being rebooted due to service action request.

VM or VNF States

The VM or VNF state represents the state of the particular VM or VNF in the service or deployment. The VM state is key to identify the current state of a particular VNF and the workflows that are running on this VM or VNF.

Table 18: VM or VNF States

VM State	Description
VM_UNDEF_STATE	The initial state of VM or VNF before deployment of this VM.
VM_DEPLOYING_STATE	VM or VNF is being deployed on to the VIM.
VM_MONITOR_UNSET_STATE	VM or VNF is deployed in the VIM but the monitoring rules are not applied.
VM_MONITOR_DISABLED_STATE	Due to a VM action request or recovery workflow, the monitoring or kpi rules applied on the VM or VNFs were not enabled.
VM_STOPPING_STATE	VM or VNF is being stopped.
VM_SHUTOFF_STATE	VM or VNF is in stopped or shutoff state.
VM_STARTING_STATE	VM or VNF is being started.
VM_REBOOTING_STATE	VM or VNF is being rebooted.

VM State	Description
VM_INERT_STATE	VM or VNF is deployed but not alive. The kpi monitor is applied and waiting for the VM to become alive.
VM_ALIVE_STATE	VM or VNF is deployed and successfully booted up or alive as per the monitor or kpi metric.
VM_UNDEPLOYING_STATE	VM or VNF is being undeployed or terminated.
VM_ERROR_STATE	VM or VNF will be in error state if deployment or any other operation is failed.

In ESC, the events play a key role in providing the current status of deployment or any other workflow. For more information, see the Event Notifications or Callback Events.

Event Notifications or Callback Events

In ESC, the events play a key role in providing the current status of deployment or any other workflow. In the Netconf Interface, ESC sends notifications and in the REST Interface, ESC sends the callback events. This section describes all the notifications or callback events sent by ESC.

Event Notification or Callback for a Deployment or a VNF

The notifications or callback event type defined below are the event that will be sent to Northbound during the life cycle of a deployment. These events are sent from ESC once the deployment request is received and processing is commenced. ESC sends notification about all stages with the status message that describes the success or failure of the stage.

Table 19: Event Notification or Callback for a Deployment or a VNF

Event State	Workflow	Description
VM_DEPLOYED	Deployment	When a VM or VNF is deployed. Success if VM or VNF deployment is successful or failure. It will be sent per VM or VNF
VM_ALIVE	Deployment	When a VM or VNF deployed successfully booted-up or alive as per the monitor\kpi metric. It will be sent per VM or VNF.
SERVICE_ALIVE	Deployment	When the deployment or service is complete and all VMs are alive or any of them failed.
VM_UNDEPLOYED	Undeployment	When a VM or VNF is undeployed. Success if VM or VNF is successfully undeployed, or Failure. It will be sent per VM or VNF.

Event State	Workflow	Description
SERVICE_UNDEPLOYED	Undeployment	When all the VMs or VNFs are undeployed. Success if all the VMs and resources under the deployment are successfully deleted, or Failure.
VM_UPDATED	Deployment Update	In any successful deployment, for each of the VM group details are updated. Success if the update is completed, or Failure. It will be sent per VM\VNF
SERVICE_UPDATED	Deployment Update	In any successful deployment, if all of the update is complete. Success if the update is completed, or Failure.
SERVICE_UPDATING_STATE	Deployment Update	In this state, some components like, VM, VNF, and data under this deployment gets updated. .
VM_RECOVERY_INIT	Recovery	The recovery init notification is sent when recovery workflow is triggered
VM_RECOVERY_DEPLOYED	Recovery	The recovery deployed notification is sent when the VM or VNF is deployed as part of the recovery workflow.
VM_RECOVERY_UNDEPLOYED	Recovery	The recovery undeployed notification is sent when the VM or VNF is undeployed as part of the recovery workflow.
VM_RECOVERY_COMPLETE	Recovery	The recovery complete notification is sent when the VM recovery is complete. Success if VM is recovered, else Failure.
VM_RECOVERY_REBOOT	Recovery	The recovery reboot notification is sent when the VM or VNF is rebooted as part of recovery. Success if reboot is successful, else Failure.
VM_RECOVERY_CANCELLED	Recovery	The recovery canceled notification is sent when a recovery was triggered but before the recovery wait time, VM went to active state.

Event State	Workflow	Description
VM_MANUAL_RECOVERY_NEEDED	Manual Recovery	The manual recovery needed notification is sent when a recovery is triggered but manual recovery policy is configured.
VM_MANUAL_RECOVERY_NO_NEED	Manual Recovery	The manual recovery not needed notification is sent when a recovery is triggered with manual recovery policy configured and the VM becomes active again.
VM_SCALE_OUT_INIT	Scale Out	The scale out init notification is sent when a scale out work flow is triggered
VM_SCALE_OUT_DEPLOYED	Scale Out	The scale out deployed notification is sent when a VM is deployed as part of scale out.
VM_SCALE_OUT_COMPLETE	Scale Out	The scale out completed notification is sent when the scale out workflow is complete.
VM_SCALE_IN_INIT	Scale In	The scale in init notification is sent when a scale in workflow is started.
VM_SCALE_IN_COMPLETE	Scale In	The scale in completed notification is sent when the scale in workflow is complete.

Event Notifications or Callback Event Types for Deployment or VNF Operation

The notifications or callback event type defined below are the event that will be sent to Northbound during various operation or action performed by the user. These events are sent from ESC once the action request is received and processing is commenced. ESC sends notification about all stages with the status message that describes the success or failure of the stage.

Table 20: Event Notifications or Callback Event Types for Deployment or VNF Operation

Event State	Workflow	Description
VM_REBOOTED	VM Action	The event is sent when a VM or VNF is rebooted.
VM_STOPPED	VM Action	The event is sent when a VM or VNF is stopped.
VM_STARTED	VM Action	The event is sent when a VM or VNF is started.

Event State	Workflow	Description
SERVICE_STOPPED	Deployment Action	The service stopped event is sent when a request to stop all the VM\VNF in a service is completed.
SERVICE_STARTED	Deployment Action	The service started event is sent when a request to start all the VM\VNF in a service is completed.
SERVICE_REBOOTED	Deployment Action	The service rebooted event is sent when a request to reboot all the VM or VNF in a service is completed.
HOST_DISABLE	Host Action / Redeploy	(OpenStack Only) The event is sent when the request to disable the host is completed.
HOST_ENABLE	Host Action / Redeploy	(OpenStack Only) The event is sent when the request to enable the host is completed.
VIM_OPERATIONAL_STATE	N/A	This event is sent when ESC detects the VIM operational state was changed.

Event Notifications or Callback Event Types for Resources

The notifications or callback event types defined below are the events that will be sent to northbound during resource creation or deletion. These events are sent from ESC once the request is received and processing is commenced. ESC sends notification about all stages with the status message that describes the success or failure of the stage.

Table 21: Event Notifications or Callback Event Types for Resources

Event State	Workflow	Description
CREATE_TENANT	Tenant	Tenant created
DELETE_TENANT	Tenant	Tenant deleted
CREATE_NETWORK	Network	Network created
DELETE_NETWORK	Network	Network deleted
CREATE_SUBNET	Subnet	Subnet created
DELETE_SUBNET	Subnet	Subnet deleted
CREATE_IMAGE	Image	Image created
DELETE_IMAGE	Image	Image deleted
CREATE_FLAVOR	Flavor	Flavor created

Event State	Workflow	Description
DELETE_FLAVOR	Flavor	Flavor deleted

CHAPTER 34

Upgrading the VNF Software Using LCS

ESC supports upgrading the VNF software application while updating a deployment. Using the policy datamodel, new Lifecycle Stages (conditions) are introduced to support the VNF upgrade. The VNF upgrade policies can be different for different VM groups. These policies are applicable for a group of VMs, and can be specified under <vm_group> rather than the entire deployment.

Upgrading VNF Software

ESC supports upgrading the initial or base image in a deployment. The ESC policy framework provides custom scripts to upgrade the software for new and existing VMs. Incremental updates are supported for the VMs, provided the ESC policy frameworks are up-to-date.

- **Upgrading Existing VMs**—The following ESC policy framework triggers script for upgrading existing VMs already deployed before the software version update.

  ```
  LCS::DEPLOY_UPDATE::POST_VM_SOFTWARE_VERSION_UPDATED
  ```

- **Upgrading New VMs**—The following ESC policy framework triggers script for upgrading new VMs when deployed, being recovered, or when scaling out.

  ```
  LCS::DEPLOY::POST_VM_ALIVE
  ```

For information on VNF Upgrade with Volume, see Upgrading VNF Software with Volume.

Updating VNF Software Version and triggering Software Upgrade

The scenario explains the procedure to trigger a software upgrade using the custom script. A CSR VM is upgraded in the example below. The service update using the `csr_dep2.xml` triggers the custom script action LCS::DEPLOY_UPDATE::POST_VM_SOFTWARE_VERSION_UPDATED. The LCS first disables monitoring of that VM, and then calls the `csr_upgrade.exp` script. The script connects to the CSR, scp's the specified upgrade .bin to the boot flash of the CSR, points the boot loader to that new bin file, and reboots the CSR VM. It then resets the bootup_time and enables monitoring. The bootup_time allows the CSR to finish rebooting without being redeployed by ESC.

Procedure

Step 1 Deploy the ESC VM.

Step 2 Upload the Day 0 configuration to the ESC VM as `/var/tmp/csp-csr-day0-config`.

Step 3 Upload the custom upgrade script to the ESC VM. For example, upload csr_upgrade.exp script to the ESC VM as `/var/tmp/csr_upgrade.exp`.

Step 4 Execute `chmod +x /var/tmp/csr_upgrade.exp`.

Step 5 Edit the initial deployment data model, for example dep.xml to include relevant IPs, username, password, and the upgrade version of the CSR.

Step 6 Edit the deployment data model's (dep.xml's) software version to reflect the upgraded CSR version.

Step 7 Upload the CSR upgrade to the home directory of the ESC user.

Step 8 Upgrade the deployed CSR VM. Run the command: `esc_nc_cli --user <username> --password <password> edit-config csr_dep2.xml`

Upgrading VNF Software with Volume

When a service is initially deployed, the data model has the policies configured for future software upgrade. When a deployment update request is received, VM upgrade is initiated as part of deployment update. LCS::DEPLOY_UPDATE::VM_PRE_VOLUME_DETACH is triggered before ESC detaches a volume. A script is supported at this lifecycle stage to unmount the volume before it is detached. ESC detaches and deletes the old volume which contains the old version of the software. After the volume is detached successfully, LCS::DEPLOY_UPDATE::VM_POST_VOLUME_DETACHED is triggered. A script is run at this LCS for further clean ups. When the new volume with a newer software version is attached, LCS::DEPLOY_UPDATE::VM_VOLUME_ATTACHED is triggered. ESC creates and attaches the new volume which contains the new version of the software. A script is run to mount the volume and trigger software installation. Once the volume is attached, LCS::DEPLOY_UPDATE::VM_SOFTWARE_VERSION_UPDATED is triggered after ESC has updated the software version of the VM. A script is run at this stage to complete the configuration for the software upgrade.

Data model for VNF Software Upgrade:

```
<esc_datamodel xmlns="http://www.cisco.com/esc/esc">
  <tenants>
    <tenant>
      <name>test</name>
      <deployments>
        <deployment>
          <name>dep</name>
          <vm_group>
            <name>Group1</name>
            <volumes>
              <volume nc:operation="delete">
                <name>v1.0</name>
                <volid>0</volid>
              </volume>
              <volume>
                <name>v2.0</name>
                <volid>1</volid>
```

```
                <sizeunit>GiB</sizeunit>
                <size>2</size>
                <bus>virtio</bus>
                <type>lvm</type>
                <image>Image-v2</image>
              </volume>
            </volumes>
            <software_version>2.0</software_version>
            <policies>
              <policy>
                <name>SVU1</name>
                <conditions>
                  <condition>
                    <name>LCS::DEPLOY_UPDATE::PRE_VM_VOLUME_DETACH</name>
                  </condition>
                </conditions>
                <actions>
                  <action>
                    <name>LOG</name>
                    <type>pre_defined</type>
                  </action>
                </actions>
              </policy>
              <policy>
                <name>SVU2</name>
                <conditions>
                  <condition>
                    <name>LCS::DEPLOY_UPDATE::POST_VM_VOLUME_ATTACHED</name>
                  </condition>
                </conditions>
                <actions>
                  <action>
                    <name>LOG</name>
                    <type>pre_defined</type>
                  </action>
                </actions>
              </policy>
              <policy>
                <name>SVU3</name>
                <conditions>
                  <condition>
                    <name>LCS::DEPLOY_UPDATE::POST_VM_SOFTWARE_VERSION_UPDATED</name>
                  </condition>
                </conditions>
                <actions>
                  <action>
                    <name>LOG</name>
                    <type>pre_defined</type>
                  </action>
                </actions>
              </policy>
            </policies>
          </vm_group>
        </deployment>
      </deployments>
    </tenant>
  </tenants>
</esc_datamodel>
```

In this data model, the existing volume v1.0 with volid of 0 is deleted. A new volume v2.0 with volid of 1 is added. The software version, <software_version> value is changed from 1.0 to 2.0. Three policies are added for the VNF software upgrade.

Note

- Instead of deleting and creating a new volume, you can update the volume properties. You can retain the name, vol_id, and image properties. If any of the above three properties change, then the volume will be deleted and created again.
- The volume size can be extended, and the bootable property can be changed. Other properties such as volume type, and image properties that are changed will trigger the volume to be created again.
- To update the volume id, you must remove the volume and add the volume again with a different volume id.
- The volume created by ESC cannot be updated by an out of band volume with same volume id, and vice versa.

Supported Lifecycle Stages (LCS) for VNF Software Upgrade with Volume

Each lifecycle stage has a condition and an action. Based on the condition, the action is executed. For information on policy driven data model, see Policy-Driven Data model, on page 185. The following three conditions are configured for the VNF software upgrade:

Condition Name	Scope	Description
LCS::DEPLOY_UPDATE::VM_PRE_VOLUME_DETACH	Deployment	Triggered just before the ESC detaches a volume
LCS::DEPLOY_UPDATE::POST_VM_VOLUME_DETACHED	Deployment	Triggered immediately after ESC has detached a volume
LCS::DEPLOY_UPDATE::POST_VM_VOLUME_ATTACHED	Deployment	Triggered immediately after ESC has attached a new volume
LCS::DEPLOY_UPDATE::POST_VM_SOFTWARE_VERSION_UPDATED	Deployment	Triggered immediately after ESC has updated the software version of the VM

LCS::DEPLOY_UPDATE::PRE_VM_VOLUME_DETACH

This LCS condition is triggered before ESC detaches the volume. A script is run to unmount the volume before it is detached.

```
<policy>
    <name>SVU1</name>
    <conditions>
        <condition>
            <name>LCS::DEPLOY_UPDATE::PRE_VM_VOLUME_DETACH</name>
        </condition>
    </conditions>
    <actions>
        <action>
            <name>LOG</name>
            <type>pre_defined</type>
        </action>
    </actions>
</policy>
```

LCS::DEPLOY_UPDATE::POST_VM_VOLUME_ATTACHED

This LCS is triggered after the ESC has attached a new volume. A script is run to mount the volume and install new applications on the new volume.

```
<policy>
    <name>SVU2</name>
    <conditions>
        <condition>
            <name>LCS::DEPLOY_UPDATE::POST_VM_VOLUME_ATTACHED</name>
        </condition>
    </conditions>
    <actions>
        <action>
            <name>LOG</name>
            <type>pre_defined</type>
        </action>
    </actions>
</policy>
```

LCS::DEPLOY_UPDATE::POST_VM_SOFTWARE_VERSION_UPDATED

This LCS is triggered after the ESC has updated the software version of the VM. A Script is run to perform final configurations to complete the software upgrade.

```
<policy>
   <name>SVU3</name>
   <conditions>
      <condition>
         <name>LCS::DEPLOY_UPDATE::POST_VM_SOFTWARE_VERSION_UPDATED</name>
      </condition>
   </conditions>
   <actions>
      <action>
         <name>LOG</name>
         <type>pre_defined</type>
      </action>
   </actions>
</policy>
```

Note All three policies above show LOG action as the predefined action in the data model sample. If a script execution is needed, then a SCRIPT action can be added. See the Script action section below for a sample script.

Script Action

In the above examples, all the actions are pre-defined logs. You can have custom scripts instead.

```
<action>
   <name>unmount_volume</name>
   <type>SCRIPT</type>
   <properties>
      <property>
         <name>script_filename</name>
         <value>/opt/cisco/esc/esc-scripts/unmount.sh</value>
      </property>
      <property>
         <name>user_param</name>
```

```
         <value>value</value>
      </property>
   </properties>
</action>
```

All the property name and value pairs are passed to the script as space separated parameters. In the above example, the `unmount.sh` value will be be called by the scripts as follows:

```
/opt/cisco/esc/esc-scripts/unmount.sh user_param value
```

Prebuilt property names can be set to pass the ESC internal ids to the specified script. The prebulit property names are as follows:

```
  <property>
    <name>internal_deployment_id</name>
  </property>
  <property>
    <name>external_deployment_id</name>
  </property>
  <property>
    <name>deployment_name</name>
  </property>
  <property>
    <name>internal_tenant_id</name>
  </property>
  <property>
    <name>external_tenant_id</name>
  </property>
```

Here is an example of a script with the prebuilt property names and values, which ESC generates.

```
script_name.sh deployment_name my-deployment-name external_deployment_id
18fbcfd5-8b63-44e0-97ec-68de25902917
external_tenant_id my-tenant-id internal_deployment_id my-tenant-idmy-deployment-name
internal_tenant_id my-tenant-id
```

By default, ESC allows 15 minutes for the script execution to complete. Some scripts may take longer time to complete. An optional property can be specified to extend the timeout value in seconds. In the example below, the timeout of the script is set to 3600 seconds.

```
<property>
    <name>wait_max_timeout</name>
    <value>3600</value>
</property>
```

Notifications for Virtual Network Function Software Upgrade

Notifications are triggered at each stage of the VNF Software upgrade.

Volume Detached

```
status SUCCESS
        status_code 200
        status_message Detached 1 volume: [Volume=test-esc-1,volid=1]
        depname dep
        tenant test
        tenant_id 9132cc90b8324a1c95a6c00975af6206
        depid eb4fe3b5-138d-41a3-b6ff-d6fa9035ca6c
        vm_group Group1
        vm_source {
            vmid cd4eeb61-61db-45a6-9da1-793be08c4de6
```

```
            hostid 8e96b8830d7bfbb337ce665586210fcca9644cbe238240e207350735
            hostname my-server-5
            software_version 1.0
            interfaces {
                interface {
                    nicid 0
                    type virtual
                    port_id 26412180-45cf-4f0b-ab45-d05bb7ca7091
                    network 943fda9e-79f8-400c-b442-3506f102721a
                    subnet e313b95c-ca1f-4c81-8d60-c9e721a85d0b
                    ip_address 192.168.0.56
                    mac_address fa:16:3e:18:90:1e
                    netmask 255.255.255.0
                    gateway 192.168.0.1
                }
            }
            volumes {
                volume {
                    display_name test-esc-1__v0_0_0_1
                    external_id 5d008a12-6fb1-492a-b648-4cf7fc8c68b1
                    bus virtio
                    type lvm
                    size 2
                }
            }
        }
        vm_target {
        }
        event {
            type VM_UPDATED
        }
    }
}
```

Volume Removed

```
notification {
    eventTime 2016-11-24T00:27:25.457+00:00
    escEvent {
        status SUCCESS
        status_code 200
        status_message Removed 1 volume: [Volume=test-esc-3,volid=1]
        depname dep
        tenant test
        tenant_id 9132cc90b8324a1c95a6c00975af6206
        depid f938ca24-d0c2-42b3-a757-66b0543fe0a6
        vm_group Group1
        vm_source {
            vmid 91379ad1-1cfc-4a10-abaf-068d01ae92b9
            hostid 101f55110748903af4844a2517e854f64843b9ac8d880ad68be8af59
            hostname my-server-4
            software_version 1.0
            interfaces {
                interface {
                    nicid 0
                    type virtual
                    port_id a8201c3e-2c6e-4313-94d0-1b4eee14f08a
                    network 943fda9e-79f8-400c-b442-3506f102721a
                    subnet e313b95c-ca1f-4c81-8d60-c9e721a85d0b
                    ip_address 192.168.0.220
                    mac_address fa:16:3e:eb:bd:77
                    netmask 255.255.255.0
                    gateway 192.168.0.1
                }
```

```
            }
        }
        vm_target {
        }
        event {
            type VM_UPDATED
        }
    }
}
```

Volume Attached

```
notification {
    eventTime 2016-11-23T19:54:48.105+00:00
        status_message Attached 1 volume: [Volume=test-esc-2,volid=0]
        depname dep
        tenant test
        tenant_id 9132cc90b8324a1c95a6c00975af6206
        depid eb4fe3b5-138d-41a3-b6ff-d6fa9035ca6c
        vm_group Group1
        vm_source {
            vmid cd4eeb61-61db-45a6-9da1-793be08c4de6
            hostid 8e96b8830d7bfbb337ce665586210fcca9644cbe238240e207350735
            hostname my-server-5
            software_version 1.1
            interfaces {
                interface {
                    nicid 0
                    type virtual
                    port_id 26412180-45cf-4f0b-ab45-d05bb7ca7091
                    network 943fda9e-79f8-400c-b442-3506f102721a
                    subnet e313b95c-ca1f-4c81-8d60-c9e721a85d0b
                    ip_address 192.168.0.56
                    mac_address fa:16:3e:18:90:1e
                    netmask 255.255.255.0
                    gateway 192.168.0.1
                }
            }
            volumes {
                volume {
                    display_name test-esc-2__v0_0_0_1
                    external_id bf5c9a01-e9fb-42fa-89ee-73699d6c519c
                    bus virtio
                    type lvm
                    size 2
                }
            }
        }
        vm_target {
        }
        event {
            type VM_UPDATED
        }
    }
}
```

Software Version Updated

```
notification {
    eventTime 2016-11-23T20:06:56.75+00:00
    escEvent {
        status SUCCESS
        status_code 200
```

```
        status_message VM Software Updated. VM name:
[dep_Group1_0_c9edef63-4d9d-43ea-af1b-16527ed2edae], previous version: [1.0], current
version: [1.1]
        depname dep
        tenant test
        tenant_id 9132cc90b8324a1c95a6c00975af6206
        depid eb4fe3b5-138d-41a3-b6ff-d6fa9035ca6c
        vm_group Group1
        vm_source {
            vmid cd4eeb61-61db-45a6-9da1-793be08c4de6
            hostid 8e96b8830d7bfbb337ce665586210fcca9644cbe238240e207350735
            hostname my-server-5
            software_version 1.1
            interfaces {
                interface {
                    nicid 0
                    type virtual
                    port_id 26412180-45cf-4f0b-ab45-d05bb7ca7091
                    network 943fda9e-79f8-400c-b442-3506f102721a
                    subnet e313b95c-ca1f-4c81-8d60-c9e721a85d0b
                    ip_address 192.168.0.56
                    mac_address fa:16:3e:18:90:1e
                    netmask 255.255.255.0
                    gateway 192.168.0.1
                }
            }
            volumes {
                volume {
                    display_name test-esc-2__v0_0_0_1
                    external_id bf5c9a01-e9fb-42fa-89ee-73699d6c519c
                    bus virtio
                    type lvm
                    size 2
                }
            }
        }
        vm_target {
        }
        event {
            type VM_SOFTWARE_VERSION_UPDATED
        }
    }
}
```

Service Updated

```
notification {
    eventTime 2016-11-23T20:06:56.768+00:00
    escEvent {
        status SUCCESS
        status_code 200
        status_message Service group update completed successfully
        depname dep
        tenant test
        tenant_id 9132cc90b8324a1c95a6c00975af6206
        depid eb4fe3b5-138d-41a3-b6ff-d6fa9035ca6c
        vm_source {
        }
        vm_target {
        }
        event {
            type SERVICE_UPDATED
        }
```

```
        }
}
```

Upgrading VNF in a Deployment

ESC allows upgrading the VNF software in an already existing deployment in any of the following lifecycle stages.

- LCS—PRE SOFTWARE UPGRADE-SCRIPT ACTION
- LCS—POST SOFTWARE UPGRADE-SCRIPT ACTION

The NB can choose to use PRE, POST or BOTH to execute the custom action script.

For details on custom scripts, see custom scripts in Script Actions, on page 175. For lifecycle stages, see Lifecycle Stage (LCS) Policy Conditions Defined at Different Stages, on page 189.

The LCS_NOTIFY notification can be turned on or off for each of the lifecycle stage. For any software_version change, the final notification for each VM is VM_SOFTWARE_VERSION_UPDATED. ESC receives the SERVICE_UPDATED notification for each of the deployment update.

ESC supports the following VNF software upgrade scenarios in an existing deployment.

- VNF upgrade after deployment
- VNF deployment and application upgrade in an existing deployment

For details on updating other resources in an existing deployment, see Updating an Existing Deployment, on page 211.

VNF Upgrade After Deployment

The VNF upgrade can happen in a single or staged transaction.

ESC adds the LCS policy and changes the software version in a single transaction.

In the two staged transaction, ESC adds the LCS policy in the first transaction, and triggers the software upgrade with the software version change in the second transaction.

Notifications

- LCS_NOTIFY—LCS::DEPLOY_UPDATE::PRE_VM_SOFTWARE_VERSION_UPDATE
- LCS_NOTIFY—LCS::DEPLOY_UPDATE::POST_VM_SOFTWARE_VERSION_UPDATED
- VM_SOFTWARE_VERSION_UPDATED
- SERVICE_UPDATED

Error

ESC performs an early validation on the VNF upgrade process. An error occurs if the custom script file does not exist. The transaction is rejected and no notifications are sent to the NFVO.

An error occurs if the custom script times out. The following notifications are sent to the NFVO.

- LCS::DEPLOY_UPDATE::PRE_VM_SOFTWARE_VERSION_UPDATE
- LCS::DEPLOY_UPDATE::PRE_VM_SOFTWARE_VERSION_UPDATE

- VM_SOFTWARE_VERSION_UPDATED
- SERVICE_UPDATED

VNF Deployment and Application Upgrade in an Existing Deployment

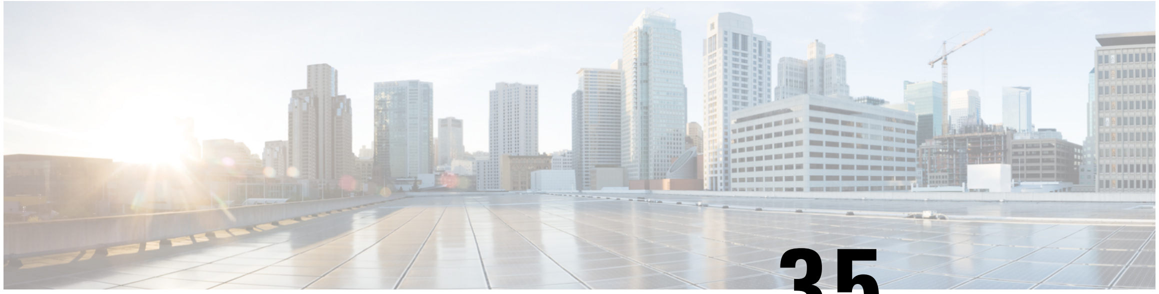
During the VNF deployment and application upgrade, ESC sends the following notifications to the NFVO.

- VM_DEPLOYED
- LCS_NOTIFY-LCS::DEPLOY::POST_VM_ALIVE
- VM_ALIVE
- SERVICE_ALIVE

Error

An error occurs when the custom script times out. The following notifications are sent to the NFVO.

- VM_DEPLOYED
- LCS::VM::POST_VM_ALIVE
- VM_DEPLOYED
- SERVICE_ALIVE



CHAPTER 35

Virtual Network Function Operations

VNF Operations

You can start, stop, and reboot VNFs. You can perform start, stop, and reboot operations using the RESTful interface.

You require a payload for VNF operations:

```
POST ESCManager/v0/{internal_tenant_id}/deployments/service/{internal_deployment_id}
```

Example,

```
<?xml version='1.0' encoding='UTF-8'?>
<service_operation xmlns='urn:ietf:params:xml:ns:netconf:base:1.0'>
   <operation>stop</operation>
</service_operation>
```

You must mention start, stop, or reboot in the operation field.

- Start VNF: Starts all VMs, enables monitoring, and reassigns thresholds according to the KPI details. The VMs start running and move to VM_ALIVE_STATE. The service is in service_active_state. Only undeploy can interrupt the start VNF workflow.
- Stop VNF: After the service is stopped, monitoring is disabled and all the VM services are stopped. The VMs are no longer available. The service is in service_stopped_state. VM is in shutoff_state. You can't perform any recovery, scale out, scale in. You can only undeploy the VNFs.
- Reboot VNF: Disables monitoring, and reboots all VMs. It stops and then starts in OpenStack, enables monitoring, and reassigns thresholds according to the KPI details. The VM is in VM_ALIVE_STATE and the service is in service_alive_state. Only undeploy can interrupt the reboot operation.

You can't start monitoring a VNF which is already running. After a reboot, logging back into the VM must indicate the reboot, update, and monitoring details. It must also indicate recovery.

VM Operations

Similar to VNF operations, you can start, stop, and reboot, resize and migrate individual VMs.

You require a payload for VM operations:

```
POST ESCManager/v0/{internal_tenant_id}/deployments/vm/{vm_name}
```

Example,

```
<?xml version='1.0' encoding='UTF-8'?>
<vm_operation xmlns='urn:ietf:params:xml:ns:netconf:base:1.0'>
   <operation>stop</operation>
   <force>true/false</force>
</vm_operation>
```

You must mention start, stop, or reboot, resize, or migrate in the operation field.

VM Resize

Note The VM Resize feature is available from the 5.7 release.

The VM resize feature in ESC provides flexibility in resizing the VMs once deployed by changing the flavor per VM group.

Changing the VM Flavor for Existing Deployment

To change the flavor of a VM, perform a service update request through the Netconf or REST API.

When a resize request is made from the ESC, the ESC stops monitoring the VM and makes the appropriate API request to OpenStack to resize the VM.

When OpenStack accepts the resize operation, the VM moves to the “VERIFY_RESIZE” state in OpenStack.

You can send a CONFIRM_RESIZE or REVERT_RESIZE request from ESC. Once the VM moves to the ACTIVE state, the ESC monitors the VM.

The REST API and RPC command in *esc_nc_cli supports* the confirm or revert requests.

Pre-requisite:

- The new flavor, for example, SolTest-Cirros-Flavor-v2 to which the VM resize needs to be done must be present in OpenStack.

Service Update Request for Change Flavor:

- For Netconf:

  esc_nc_cli edit-config sample_deployment_update.xml

- For REST API:

  PUT /v0/deployments/{internal_deployment_id}

ESC NETCONF:

Changing [*VM - ACTIVE* to *VERIFY_RESIZE* state]

The following example shows the *sample_deployment_update.xml* for VM resize service update request for NETCONF and REST API. In this xml file, you needs to modify the existing flavor name to a new flavor name for example `SolTest-Cirros-Flavor-v2` that is present in OpenStack.

```
<?xml version="1.0" encoding="UTF-8"?>
<esc_datamodel xmlns="http://www.cisco.com/esc/esc">
    <tenants>
        <tenant>
            <name>admin</name>
            <deployments>
                <deployment>
                    <name>Cirros-VNF</name>
                    <vm_group>
                        <name>SolTest-Cirros-VM-Group</name>
                        <bootup_time>60</bootup_time>
                        <recovery_wait_time>0</recovery_wait_time>
                        <image>SolTest-Cirros-Image</image>
                        <flavor>SolTest-Cirros-Flavor-v2</flavor>
                        <interfaces>
                            <interface>
                                <nicid>0</nicid>
                                <network>SolTest-Network</network>
                            </interface>
                        </interfaces>
                        <scaling>
                            <min_active>1</min_active>
                            <max_active>1</max_active>
                            <elastic>true</elastic>
                        </scaling>
                        <kpi_data>
                            <kpi>
                                <event_name>VM_ALIVE</event_name>
                                <metric_value>1</metric_value>
                                <metric_cond>GT</metric_cond>
                                <metric_type>UINT32</metric_type>
                                <metric_collector>
                                    <type>ICMPPing</type>
                                    <nicid>0</nicid>
                                    <poll_frequency>3</poll_frequency>
                                    <polling_unit>seconds</polling_unit>
                                    <continuous_alarm>false</continuous_alarm>
                                </metric_collector>
                            </kpi>
                        </kpi_data>
                        <rules>
                            <admin_rules>
                                <rule>
                                    <event_name>VM_ALIVE</event_name>
                                    <action>"ALWAYS log"</action>
                                    <action>"TRUE servicebooted.sh"</action>
                                    <action>"FALSE recover autohealing"</action>
                                </rule>
                            </admin_rules>
                        </rules>
                      <config_data/>
                    </vm_group>
                </deployment>
            </deployments>
        </tenant>
    </tenants>
</esc_datamodel>
```

Upon successful deployment, an XML payload returns with *<ok/>*.

Upon unsuccessful deployment, a validation error or OpenStack API error with an appropriate error message returns.

ESC REST API:

An HTTP PUT operation specifies the *ESCManager* API with a deployment XML payload passed with an updated flavor name inside the tag.

For Example: *<flavor>UpdateflavorName</flavor>.*

PUT:/ESCManager/v0/deployments/{internal_deployment_id}

Upon successful deployment, an HTTP 200 code returns.

Upon an unsuccessful deployment, a validation error or OpenStack API error, along with an appropriate HTTP error code, and error messages return.

Notifications:

- ESC sends the *SERVICE_UPDATED* notification, once the request in OpenStack VIM for resizing the operation is accepted.

  Sample Notification sent by ESC for VM RESIZE request:

  ```
  2022-04-11 05:45:47.232 INFO
  2022-04-11 05:45:47.232 INFO  ===== SEND NOTIFICATION STARTS =====
  2022-04-11 05:45:47.232 INFO  Type: SERVICE_UPDATED
  2022-04-11 05:45:47.232 INFO  Status: SUCCESS
  2022-04-11 05:45:47.232 INFO  Status Code: 200
  2022-04-11 05:45:47.232 INFO  Status Msg: Service group update completed successfully
  2022-04-11 05:45:47.233 INFO  Tenant: admin
  2022-04-11 05:45:47.233 INFO  Deployment ID: 84779a3b-01e0-4eeb-9b1c-fe30317b6bb5
  2022-04-11 05:45:47.233 INFO  Deployment name: Cirros-VNF
  2022-04-11 05:45:47.233 INFO  =====  SEND NOTIFICATION ENDS  =====
  ```

- ESC sends the *VM_VERIFY_RESIZE* notification, once the VM goes to VERIFY_RESIZE state.

  Sample Notification sent by ESC for VM RESIZE request:

  ```
  2022-04-11 05:46:09.425 INFO
  2022-04-11 05:46:09.425 INFO  ===== SEND NOTIFICATION STARTS =====
  2022-04-11 05:46:09.425 INFO  Type: VM_VERIFY_RESIZE
  2022-04-11 05:46:09.425 INFO  Status: SUCCESS
  2022-04-11 05:46:09.426 INFO  Status Code: 200
  2022-04-11 05:46:09.426 INFO  Status Msg: VM Resize request processed
  2022-04-11 05:46:09.426 INFO  VM State: VERIFY_RESIZE
  2022-04-11 05:46:09.426 INFO  Tenant ID: 31fb89f6647b4c109efda76479c07332
  2022-04-11 05:46:09.426 INFO  Deployment ID: 84779a3b-01e0-4eeb-9b1c-fe30317b6bb5
  2022-04-11 05:46:09.426 INFO  Deployment name: Cirros-VNF
  2022-04-11 05:46:09.426 INFO  VM group name:SolTest-Cirros-VM-Group
  2022-04-11 05:46:09.426 INFO  VM Source:
  2022-04-11 05:46:09.426 INFO      VM ID: 3b24cef1-5cf4-4e9c-b98e-682bfbdf8240
  2022-04-11 05:46:09.426 INFO     VM Name:cirros-vm_0_357b379e-bac5-4c6a-a214-0c0b1343ad4d
  2022-04-11 05:46:09.427 INFO      VM Name (Generated):
  cirros-vm_0_357b379e-bac5-4c6a-a214-0c0b1343ad4d
  2022-04-11 05:46:09.427 INFO      VIM ID: default_openstack_vim
  2022-04-11 05:46:09.427 INFO      VIM Project: admin
  2022-04-11 05:46:09.427 INFO      VIM Project ID: 31fb89f6647b4c109efda76479c07332
  2022-04-11 05:46:09.427 INFO      Host ID:
  549a38c4fd21432060486a69bf9f82839f126adbe7f02e9cd2f4f3b6
  2022-04-11 05:46:09.427 INFO      Host Name: esc-soltest-116
  2022-04-11 05:46:09.427 INFO  =====  SEND NOTIFICATION ENDS  =====
  2022-04-11 05:46:09.427 INFO
  ```

Confirm or Revert Request:

ESC NETCONF:

To confirm or revert the request, that is to change [VM - *VERIFY_RESIZE* to *ACTIVE* state], use the following RPC command.

esc_nc_cli Impaction <action: CONFIRM_RESIZE or *REVERT_RESIZE><Generated VM Name>*

For Example:

esc_nc_cli vm-action CONFIRM_RESIZE cirros-vm_0_357b379e-bac5-4c6a-a214-0c0b1343ad4d

Upon successful action, an XML payload returns with *<ok/>*

Upon unsuccessful action that is validation error or OpenStack API error, an appropriate error message returns.

ESC REST API:

An HTTP POST operation specifies the *ESCManager* API with an XML payload passed with the operation name inside the tag.

For Example,*<operation>confirm_resize</operation>* or <operation>*revert_resize</operation>*

POST:/ESCManager/v0/SystemAdminTenantId/deployments/vm/{Generated_vm_name}

The following shows the POST Sample payload:

```
[admin@dnd-admin-5-7-0-52-vtp-scan ~]$ cat operation.xml
<vm_operation xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <operation>confirm_resize</operation>
</vm_operation>
```

To revert VM resize, use the following payload:

```
<vm_operation xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <operation>revert_resize</operation>
</vm_operation>
```

Upon successful deployment, an HTTP 200 code returns.

Upon an unsuccessful deployment that is for a validation error or OpenStack API error, an appropriate HTTP error code, and error messages returns.

Notifications:

To confirm resize:

- ESC sends the VM_CONFIRM_RESIZE_INIT notification once the confirm request in OpenStack VIM is accepted.
- ESC sends the VM_CONFIRM_RESIZE_COMPLETE notification once it receives confirmation that the operation completed successfully in OpenStack VIM.
- ESC sends the VM_ALIVE notification once the monitor is set and the VM is in the ACTIVE state.

To revert resize:

- ESC sends the VM_REVERT_RESIZE_INIT notification once the confirm request in OpenStack VIM is accepted..
- ESC sends the VM_REVERT_RESIZE_COMPLETE notification once it confirms that the operation completes successfully in OpenStack VIM.
- ESC sends the VM_ALIVE notification once you set the monitor, and the VM is in the ACTIVE state.

Note In ESC, the flavor change request is at the VM Group level. But a VM Group can have more than one VM. This resize feature works only when the VM Group has a single VM in it.

VM Migrate

Note The VM Migrate feature is available from the 5.7 release.

ESC provides flexibility in migrating the VMs. Migration is of two types:

- Live Migration
- Cold Migration

Live migration transfers a live VM from one host to another while cold migration transfers a powered-off VM from one host to another.

VM Live Migration:

Use the following commands to perform VM live migration from one host to another:

Note The host parameter is optional. If the host parameter is not included, the NOVA scheduler selects the host.

Example for REST API:

```
POST /v0/{internal_tenant_id}/deployments/migrate-vm/{vm_name}
curl -X POST
'http://localhost:8080/ESCManager/v0/test-tenant/deployments/migrate-vm/test-dep_g1_0_6be93ee9-f7fb-473e-a3f6-eca281802008'
 \
-H 'Content-Type: application/xml' \
-H 'Callback: http://localhost:9009/callback' \
-H 'Callback-ESC-Events: http://localhost:9009/event' \
--data-raw '<vm_migrate_operation xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <operation>migrate</operation>
   <migrationType>live</migrationType>
   <host>Host-24</host>
</vm_migrate_operation>'
```

Example for RPC Payload:

```
<vmMigrate xmlns=http://www.cisco.com/esc/esc>
   <vmName>test-dep_g1_0_6be93ee9-f7fb-473e-a3f6-eca281802008</vmName>
   <migrationType>live</migrationType>
   <host>Host-24</host>
</vmMigrate>
```

Example for NETCONF:

```
esc_nc_cli supports VM migration as follows:
Migrate VM Action : migrate-vm-action <vm-name> <migration-type> [block-migration] [host]
                               migration-type : = live|cold
                               block-migration : = true|false (live migration only)
```

Example for esc_nc_cli live migrate:

```
esc_nc_cli migrate-vm-action test-dep_g1_0_6be93ee9-f7fb-473e-a3f6-eca281802008 live
```

Example for esc_nc_cli live migrate with host:

```
esc_nc_cli migrate-vm-action test-dep_g1_0_6be93ee9-f7fb-473e-a3f6-eca281802008 live Host-22
```

Example for esc_nc_cli live migrate with block-migration:

```
esc_nc_cli migrate-vm-action test-dep_g1_0_6be93ee9-f7fb-473e-a3f6-eca281802008 live true
```

Example for esc_nc_cli live migrate with host and block-migration:

```
esc_nc_cli migrate-vm-action test-dep_g1_0_6be93ee9-f7fb-473e-a3f6-eca281802008 live true
Host-22
```

Notifications:

ESC sends the following notifications once the live migration is successful:

```
VM_MIGRATE_INIT
VM_MIGRATED
VM_MIGRATE_COMPLETE
```

VM Cold Migration:

Use the following commands to perform VM cold migration from one host to another:

Note The host parameter is optional. If the host parameter is not included, the NOVA scheduler selects the host.

Example for REST API:

```
POST /v0/{internal_tenant_id}/deployments/migrate-vm/{vm_name}
curl -X POST
'http://localhost:8080/ESCManager/v0/test-tenant/deployments/migrate-vm/test-dep_g1_0_6be93ee9-f7fb-473e-a3f6-eca281802008'
 \
-H 'Content-Type: application/xml' \
-H 'Callback: http://localhost:9009/callback' \
-H 'Callback-ESC-Events: http://localhost:9009/event' \
--data-raw '<vm_migrate_operation xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <operation>migrate</operation>
   <migrationType>cold</migrationType>
   <host>Host-24</host>
</vm_migrate_operation>'
```

Example for RPC Payload:

```
<vmMigrate xmlns="http://www.cisco.com/esc/esc">
 <vmName>test-dep_g1_0_6be93ee9-f7fb-473e-a3f6-eca281802008</vmName>
 <migrationType>cold</migrationType>
 <host>Host-24</host>
</vmMigrate>
```

Example for NETCONF:

Example for esc_nc_cli cold migrate:

```
esc_nc_cli migrate-vm-action test-dep_g1_0_6be93ee9-f7fb-473e-a3f6-eca281802008 cold
```

Example for esc_nc_cli cold migrate with host:

```
esc_nc_cli migrate-vm-action test-dep_g1_0_6be93ee9-f7fb-473e-a3f6-eca281802008 cold Host-22
```

Confirm or revert the migration once ESC sends the VM_VERIFY_RESIZE notification.

To confirm or revert, use either of the following commands: esc_nc_cli (Netconf) command or REST API command

```
VM Action : vm-action <action> <generated vm name>
```

```
action:=STOP|START|REBOOT|DISABLE_MONITOR|ENABLE_MONITOR|CONFIRM_RESIZE|REVERT_RESIZE
```

Example for esc_nc_cli to confirm the migration or resize:

```
esc_nc_cli vm-action CONFIRM_RESIZE test-dep_g1_0_6be93ee9-f7fb-473e-a3f6-eca281802008
```

Once the action is triggered, the following notifications are received:

- VM_CONFIRM_RESIZE_INIT
- VM_CONFIRM_RESIZE_COMPLETE
- VM_ALIVE

REST API to confirm the migration or resize:

```
POST /v0/{internal_tenant_id}/deployments/vm/{vm_name}
```

Sample payload:

```
<vm_operation xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <operation>confirm_resize</operation>

</vm_operation>
```

Notifications:

ESC sends the following notifications once the cold migration is successful:

```
VM_MIGRATE_INIT
VM_MIGRATED
VM_VERIFY_RESIZE

VM_CONFIRM_RESIZE_INIT
VM_CONFIRM_RESIZE_COMPLETE
VM_ALIVE
```

or

```
VM_REVERT_RESIZE_INIT
VM_REVERT_RESIZE_COMPLETE
VM_ALIVE
```

VNF Backup and Restore Operations

This section describes VNF backup and restore operations using VM snapshots.

VNF Backup Operations

Manage VM Snapshots

ESC creates a snapshot, which is an image (and a volume in certain circumstances) on OpenStack VIM. ESC APIs manage the snapshots of VNFs managed by ESC. ESC supports the below main snapshot operations:

- Create VM snapshots
- List VM snapshots
- Delete VM snapshots

ESC executes the VM snapshot operations using ESC REST API with HTTP and HTTPS protocols. The create and delete VM snapshot operations are supported through `esc_nc_cli` script. Both Netconf and REST API notifications are generated during the various stages of the snapshot operations for create and delete operations.

Note The VM snapshot operations are supported on the OpenStack VIM only.

Create VM Snapshot

You can create a snapshot (using the REST API or the esc_nc_cli script) from any VNF that is managed by the ESC VM. A snapshot can only be created for active or stopped VNFs, which translate to the ESC VM status of VM_ALIVE or VM_STOPPED respectively. You can specify the snapshot name in the payload of the API invocations. A unique ID is generated along with the snapshot name, which is used as a reference when specifying the ESC snapshot operation payloads, if the snapshot name is not unique. A snapshot (an image) is created on OpenStack. In case of a VNF which uses a bootable volume, a *volume snapshot* is also created on OpenStack.

Create Snapshot Using REST API

To create a snapshot, specify an HTTP POST operation to the ESCManager API:

```
POST: /ESCManager/v0/<tenant-id>/deployments/snapshot-vm/<generated-vm-name>
```

The payload must contain *operation* and *name* values, and the operation value must be **snapshot**.

```
operation: snapshot
name: <snapshot-name>
```

If successful, an HTTP 200 code is returned, and there is no payload.

If unsuccessful (validation error or OpenStack API error), an appropriate HTTP error code and error message are returned.

The following shows the API invocation to create a snapshot:

```
[admin@localhost]$ cat snapshot.json
{
    "operation": "snapshot",
    "name": "my-snapshot-name"
}

[admin@localhost]$ curl -X POST -d @snapshot.json -H 'Content-Type: application/json' -H
'callback: http://localhost:9009' -H 'Callback-ESC-Events: http://localhost:9009'
"http://localhost:8080/ESCManager/v0/snapshot-tenant/deployments/snapshot-vm/new-deployment-n_new-gr_0_af0148e2-e74c-4be7-b8c1-49bd53def6ba"
```

Create Snapshot Using `esc_nc_cli script`

To create a snapshot using the `esc_nc_cli` script, fixed parameters can be passed specifying the generated VM name and operation:

```
VM Backup Action : vm-backup-action vm-name backup-name [<action-type>] [<xmlfile>]
    action-type := SNAPSHOT|EXPORT
```

The optional action-type parameter defaults to SNAPSHOT if not specified. The following shows the script invocation to create a snapshot:

```
[admin@localhost]$ esc_nc_cli vm-backup-action
new-deployment-n_new-gr_0_af0148e2-e74c-4be7-b8c1-49bd53def6ba my-snapshot-name SNAPSHOT
VM Backup Action
/opt/cisco/esc/confd/bin/netconf-console --port=830 --host=127.0.0.1 --user=esc-nc-admin
--privKeyFile=/home/admin/.ssh/confd_id_rsa --privKeyType=rsa
--rpc=/tmp/tmp_esc_nc_cli.c8d9kAjcGf
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="1">
    <ok/>
</rpc-reply>
```

If successful, an XML payload with a single <ok/> element is returned. If unsuccessful (validation error or OpenStack API error), an appropriate error message is returned.

Notes

Note the following for both ESC REST API and the `esc_nc_cli`:

- Snapshot names must be less than or equal to 255 characters in length.
- The generated VM name must be valid.
- action-type must be SNAPSHOT or EXPORT (`esc_nc_cli` only).
- xmlfile - if specified - must contain a valid XML document (`esc_nc_cli` only).

Notifications

Both Netconf notifications and ESC REST callback messages are sent during the create snapshot operation.

Table 22:

Notification (NETCONF or ESC callback)	When the notification is sent
VM_BACKUP_INIT	When the API is invoked and validation passed.
VM_BACKUP_CREATED	When OpenStack has successfully received and validated the snapshot create request.
VM_BACKUP_COMPLETE	When OpenStack has finished the snapshot create request operation, and it was either a success, or an error occurred.

The following shows an example VM_BACKUP_CREATED successful Netconf notification (other notifications are similar):

```
<?xml version="1.0" encoding="UTF-8"?>
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
    <eventTime>2021-09-14T12:18:39.836+00:00</eventTime>
    <escEvent xmlns="http://www.cisco.com/esc/esc">
        <status>SUCCESS</status>
        <status_code>202</status_code>
        <status_message>Snapshot is now active.</status_message>
        <depname>snapshot-deployment-name</depname>
        <tenant>snapshot-tenant</tenant>
```

```
        <tenant_id>7d61b5de73874f88a458d486759a9b83</tenant_id>
        <depid>ae0bea05-9630-4d17-a9e7-926f1f625dc7</depid>
        <vm_group>snapshot-group</vm_group>
        <vm_source>
            <vmid>1773914c-20cd-4f50-b337-1e46be2cf295</vmid>
           <vmname>new-deployment-n_new-gr_0_af0148e2-e74c-4be7-b8c1-49bd53def6ba</vmname>


<generated_vmname>new-deployment-n_new-gr_0_af0148e2-e74c-4be7-b8c1-49bd53def6ba</generated_vmname>

            <vim_id>default_openstack_vim</vim_id>
            <vim_project>snapshot-tenant</vim_project>
            <vim_project_id>7d61b5de73874f88a458d486759a9b83</vim_project_id>
            <hostid>95503baadeccce2d33e5d924322390aee9d30c6ed24043284bf46984</hostid>
            <hostname>pf-ucs-27</hostname>
        </vm_source>
        <event>
            <type>VM_BACKUP_CREATED</type>
        </event>
    </escEvent>
</notification>
```

For the failure cases, Netconf notifications and ESC REST callback messages are still generated, but:

- the <status> value will be *FAILURE*,
- the <status_code> will be *500*, and
- the <status_message> will be an appropriate message, either internally generated or sent back from OpenStack.

List Snapshot

You can list the snapshots using the ESC REST API. Only the snapshots managed by ESC can be listed. A subset of the snapshot data can be specified as query parameters to reduce the number of snapshots returned. The returned snapshot data can be in XML or JSON format, controlled by the HTTP *Accept* header. The Accept header value defaults to XML if not specified.

Only the ESC REST API supports listing snapshots. The `esc_nc_cli` does not supported listing ESC managed entities.

List Snapshot Usig ESC REST API

To list snapshots, an HTTP GET operation can be specified to the ESCManager API:

```
GET: /ESCManager/v0/snapshots
```

Optional query parameters can also be specified: *internalTenantId*, *generatedVMName*

An HTTP 200 code is always returned regardless of how many snapshots are returned.

The following shows the API invocation to list a snapshot for a specific internal tenant id and generated VM name:

```
[admin@localhost]$ curl -X GET --header "Accept: application/xml"
"http://localhost:8080/ESCManager/v0/snapshots?internalTenantId=snapshot-tenant&generatedVMName=new-deployment-n_new-gr_0_af0148e2-e74c-4be7-b8c1-49bd53def6ba"
 | xmllint --format -

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<snapshots>
    <snapshot xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
        <id>7813c20b-94b6-492b-ae74-0bd36c1168dc</id>
        <name>my-snapshot-name</name>
```

```
    <creation_start_date>2021-07-20T11:26:47.532Z</creation_start_date>
    <creation_end_date>2021-07-20T11:27:53.139Z</creation_end_date>
    <status>available</status>
    <status_message>Snapshot image for VM [gen_vm_name] is active.</status_message>

<gen_vm_name>new-deployment-n_new-gr_0_af0148e2-e74c-4be7-b8c1-49bd53def6ba</created_from_generated_vm_name>

    <vim_id>default_openstack_vim</vim_id>
    <tenant>snapshot-tenant</tenant>
    <bootable_volume_snapshot_id:c3cd5d13-63bf-49f0-b864-df3bc024d5e4/>
  </snapshot>
</snapshot>
```

Note There are no notifications generated when this API is invoked.

Delete Snapshots

Snapshot created by ESC can be deleted using REST API or the `esc_nc_cli` script. Only snapshots managed by the current ESC VM can be deleted, and only one snapshot can be deleted at a time. If successful, the snapshot is deleted from ESC, and also the related image and volume snapshot (if applicable) within OpenStack.

Delete Snapshots Using REST API

To delete a snapshot previously created via ESC, an HTTP DELETE operation can be specified to the ESCManager API:

```
DELETE: /ESCManager/v0/snapshots/<snapshot-id|snapshot-name>
```

Either a snapshot id can be passed, or a snapshot name. If successful, an HTTP 200 code is returned, and there is no payload. If unsuccessful (validation error or OpenStack API error), an appropriate HTTP error code and error message are returned. The following shows the API invocation to delete a snapshot:

```
[admin@localhost]$ curl -X DELETE -H 'callback: http://localhost:9009' -H
'Callback-ESC-Events: http://localhost:9009'
"http://localhost:8080/ESCManager/v0/snapshots/7813c20b-94b6-492b-ae74-0bd36c1168dc"
```

Delete Snapshot Using `esc_nc_cli`

To delete a snapshot using the `esc_nc_cli` script, only a snapshot id or snapshot name needs to be passed as the single parameter:

```
Snapshot Action : snapshot-action <snapshot-id|snapshot-name>
```

The following shows the script invocation to create a snapshot:

```
[admin@localhost]$ esc_nc_cli snapshot-action delete my-snapshot-name
```

or

```
[admin@localhost]$ esc_nc_cli snapshot-action delete my-snapshot-name-1
<?xml version="1.0" encoding="UTF-8"?>
<error xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <error_code>404</error_code>
  <error_message>Snapshot image [my-snapshot-name-1] not found.</error_message>
</error>
```

If successful, an XML payload with a single <ok/> element is returned. If unsuccessful (validation error or OpenStack API error), an appropriate error message is returned.

Note the following for both ESC REST API and `esc_nc_cli`.

- The snapshot id or snapshot name must be valid.

- If a snapshot name is specified, it must be unique.

Notifications

Both Netconf notifications and ESC REST callback messages are sent during the delete snapshot operation.

The notifications are:

Table 23:

Notification (Netconf and/or ESC Callback)	When the notification is sent
VM_SNAPSHOT_DELETING	Sent upon successful submission and validation.
VM_SNAPSHOT_DELETED	When OpenStack has finished the snapshot delete operation, and it was either a success, or an error occurred. If the snapshot has a volume snapshot along with the image snapshot, then the notification will not be sent until both are deleted.

The following shows an example VM_SNAPSHOT_DELETED successful Netconf notification (other notifications are similar):

```
<?xml version="1.0" encoding="UTF-8"?>
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
    <eventTime>2021-09-14T12:18:39.836+00:00</eventTime>
    <escEvent xmlns="http://www.cisco.com/esc/esc">
    <status>SUCCESS</status>
    <status_code>200</status_code>
    <status_message>Snapshot image [2ffadd36-3b41-4c13-a9d6-a48c07764d1a] has been
deleted.</status_message>
    <depname>snapshot-deployment-name</depname>
    <tenant>snapshot-tenant</tenant>
    <tenant_id>7d61b5de73874f88a458d486759a9b83</tenant_id>
    <depid>ae0bea05-9630-4d17-a9e7-926f1f625dc7</depid>
    <vm_group>snapshot-group</vm_group>
    <vm_source>
        <vmid>1773914c-20cd-4f50-b337-1e46be2cf295</vmid>
        <vmname>new-deployment-n_new-gr_0_af0148e2-e74c-4be7-b8c1-49bd53def6ba</vmname>

<gen_vm_name>new-deployment-n_new-gr_0_af0148e2-e74c-4be7-b8c1-49bd53def6ba</generated_vmname>

        <vim_id>default_openstack_vim</vim_id>
        <vim_project>snapshot-tenant</vim_project>
        <vim_project_id>7d61b5de73874f88a458d486759a9b83</vim_project_id>
        <hostid>95503baadeccce2d33e5d924322390aee9d30c6ed24043284bf46984</hostid>
        <hostname>pf-ucs-27</hostname>
    </vm_source>
    <event>
        <type>VM_SNAPSHOT_DELETED</type>
    </event>
    </escEvent>
</notification>
```

For the failure cases, Netconf notifications and ESC REST callback messages are still generated, but:

- the <status> value will be *FAILURE*,
- the <status_code> will be *500*, and

- the <status_message> will be an appropriate message, either internally generated or sent back from OpenStack.

VM Snapshot Polling Parameters

Vim Manager configuration properties can be used to control the interval between calls to OpenStack to check the status of the create and delete operations, along with the maximum number of minutes an operation is allowed before timing out with an error.

These properties are:

- vim.asyncpoller.snapshot.create.poll.secs # default 15, the number of seconds between polls
- vim.asyncpoller.snapshot.create.timeout.mins # default 20, maximum number of minutes for the create snapshot operation
- vim.asyncpoller.snapshot.delete.poll.secs # default 15, the number of seconds between polls
- vim.asyncpoller.snapshot.delete.timeout.mins # default 20, maximum number of minutes for the delete snapshot operation

The default values can be overridden by setting them in the application.properties file under */opt/cisco/esc/vimmanager/application.properties*, and restarting the vim manager service, as shown below:

```
[admin@localhost]$ sudo cat /opt/cisco/esc/vimmanager/application.properties
vim.asyncpoller.snapshot.create.poll.secs=5
vim.asyncpoller.snapshot.create.timeout.mins=10
vim.asyncpoller.snapshot.delete.poll.secs=10
vim.asyncpoller.snapshot.delete.timeout.mins=60

[admin@localhost]$ sudo escadm vimmanager restart
Stopping vimmanager service: [OK]
Starting vimmanager service: [OK]

[admin@localhost]$ sudo escadm vimmanager show
VimManager System Configurations.
{
    "ccp.pollRetries": "200",
    "ccp.pollRetryDelaySecs": "15",

    . . .

    "vim.asyncpoller.snapshot.create.poll.secs": "5",
    "vim.asyncpoller.snapshot.create.mins": "10",
    "vim.asyncpoller.snapshot.delete.poll.secs": "10",
    "vim.asyncpoller.snapshot.delete.timeout.mins": "60",

    . . .

    "vmware.ovftool.params": "--acceptAllEulas --disableVerification --noSSLVerify
--allowExtraConfig",
    "vmware.powerOnRetry": "8"
}
```

In an HA setup, the application properties file will need to be copied to both nodes.

Alternatively, the values can be set dynamically (although their values will not be persisted after a restart), using the escadm script:

```
[admin@loclhost] sudo escadm vimmanager set --config
vim.asyncpoller.snapshot.create.poll.secs=200
vim.asyncpoller.snapshot.create.timeout.mins=1
```

```
VimManager configuration [vim.asyncpoller.snapshot.create.poll.secs] has updated to [200].
VimManager configuration [vim.asyncpoller.snapshot.create.timeout.mins] has updated to [1].
```

Snapshots of VNFs with Bootable Volumes

If a snapshot is taken of an ESC managed VNF which has a boot volume, then both an image snapshot and a volume snapshot are created within OpenStack.

Note The image snapshot name will be the snapshot name specified in the snapshot payload. The *volume snapshot* name (if applicable) will be be prepended with *snapshot for* .

For example, if a snapshot is taken on an ESC VM of a VNF with a bootable volume, and the snapshot was named *my-snapshot-name*, then the following would hold true:

```
[admin@localhost]$ openstack volume snapshot list | grep my-snapshot-name
| 52a96891-f22d-4863-bb47-bd9442ca0cb1 | snapshot for my-snapshot-name | None | available
| 2 |
```

```
[admin@localhost]$ openstack image list | grep my-snapshot-name
| c8846c14-48e4-45db-88a0-f838fc3ac29d | my-snapshot-name | active |
```

Volume snapshots cannot be used directly either within ESC or natively on OpenStack in a restore operation: a bootable volume must be created from the snapshot first. ESC supports creating bootable volumes from volume snapshots. For more information, see VNF Restore Operations.

VNF Restore Operations

Snapshot of VNFs Without Bootable Volumes

If ESC takes a snapshot of a VNF with a non-bootable volume, then that snapshot is stored in OpenStack as a snapshot image, and that snapshot image can be used to restore from via ESC's service update functionality. A service update XML can be created identical to the original deployment XML but with the snapshot image name. Once this service update XML is deployed to ESC using the REST or `esc_nc_cli` interfaces, ESC will internally update its image name for the VNF, and upon the next redeployment (typically triggered manually using the REST or `esc_nc_cli` interfaces), a new deployment will be created with the new image.

Snapshots of VNFs With Bootable Volumes

If ESC takes a snapshot of a VNF with a bootable volume, then that snapshot is stored in OpenStack as both a snapshot image and a volume snapshot. The volume snapshot cannot be used directly within the restore process of a VNF, either by ESC or by OpenStack. A bootable volume must first be created from the volume snapshot which is considered out-of-band from ESC's perspective (i.e. not managed directly by ESC). Once the bootable volume is created and available, then it can be used to restore from via ESC's service update functionality. A service update XML can be created identical to the original deployment XML, but with a delete operation against the named original volume, and a create operation specifying the new, bootable volume. Once this service update XML is deployed to ESC using the REST or `esc_nc_cli` interfaces, ESC will automatically swap out the original volume for the new volume without the need for a VNF redeployment, thus preserving all OpenStack UUIDs and resources.

The original volume (which was detached from the VNF) will still remain in OpenStack and must be cleaned up manually.

Create Bootable Volume from Volume Snapshot

If ESC takes a snapshot of a VNF with a bootable volume, then a volume snapshot is created in OpenStack, and ESC can then be used to create a bootable volume from the volume snapshot. Creating a bootable volume from the volume snapshot is required for any restoration operations, either within ESC or the OpenStack APIs directly. The volume name can be specified in the payload of the ESC REST API invocation. The `esc_nc_cli` script does not support creating a bootable volume from a volume snapshot. Volume names do not have to be unique, as a unique ID is generated alongside the name which can be used as a reference when specifying ESC restore operation payloads. The end result of a successfully created volume from volume snapshot will be a new, bootable volume in OpenStack

Note The new, bootable volume on OpenStack is not managed by ESC. It is out-of-band and must be managed by an Orchestrator directly.

Create Bootable Volume from Volume Snapshot Using REST API

To create a volume from a volume snapshot, an HTTP POST operation can be specified to the snapshot endpoint of the ESCManager API:

```
POST: /ESCManager/v0/snapshots/<snapshot-id>/volumes
```

The payload must contain a *name* value which will be the new volume's name, but can *volume_type*, *multiattach* and *bootable* can be optionally specified.

```
operation: snapshot
name: <snapshot-name>
volume_type: <valid-volume-type>   # defaults to the OpenStack default volume type
multiattach: <true|false>          # defaults to false
bootable: <true|false>             # defaults to true
```

If successful, an HTTP 202 code is returned (indicating that the operation has bee successfully submitted to OpenStack), and there is no payload. If unsuccessful (validation error or OpenStack API error), an appropriate HTTP error code and error message are returned.

The following shows the API invocation to create a snapshot after listing it within ESC first to determine the snapshot id:

```
[admin@localhost]$ curl -s
"http://localhost:8080/ESCManager/v0/snapshots?internalTenantId=dave-2000" | xmllint --format
 -
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<snapshots>
    <snapshot xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
        <id>171ffa7d-8318-47d1-acab-b01db4501a39</id>
        <name>my-snapshot-name</name>
        <creation_start_date>2021-09-20T08:29:22.074+01:00</creation_start_date>
        <creation_end_date>2021-09-20T08:33:41.193+01:00</creation_end_date>
        <status>active</status>
        <status_message>Snapshot image for
[new-dep-4_new-gr_0_34b9da8a-af64-4452-a8a3-8972e23e4e98] is active.</status_message>

<created_from_generated_vm_name>new-dep-4_new-gr_0_34b9da8a-af64-4452-a8a3-8972e23e4e98</created_from_generated_vm_name>

        <vim_id>my-snapshot-vim</vim_id>
        <tenant>dave-2000</tenant>
        <volume_snapshot_id>c4548ba4-0480-4b42-8229-ad98de44b3ea</volume_snapshot_id>
    </snapshot>
</snapshots>
```

```
[admin@localhost]$ cat volume_from_volume_snapshot.json
{
    "name": "my-bootable-volume-from-snapshot-volume",
    "multiattach": true
}

[admin@localhost]$ curl -X POST -d @volume_from_volume_snapshot.json -H 'Content-Type:
application/json' -H 'callback: http://localhost:9009' -H 'Callback-ESC-Events:
http://localhost:9009'
"http://localhost:8080/ESCManager/v0/snapshots/171ffa7d-8318-47d1-acab-b01db4501a39/volumes"

[admin@localhost]$ openstack volume list | grep my-bootable-volume
| c4548ba4-0480-4b42-8229-ad98de44b3ea | my-bootable-volume-from-snapshot-volume | available
 | 2 | |
```

The ESC REST API applies the following validation:

- Volume names must be less than or equal to 255 characters in length.
- The snapshot id must be for a snapshot that ESC managed (i.e. it must be one of the snapshots that a *list snapshot* operation would return).
- *name* must be specified in the payload - all other attributes can optionally be set.
- Non-supported attribute names in the payload are ignored.
- The snapshot must have been taken for a VNF which had a bootable volume.

Notifications

Only ESC REST callback messages are generated for this operation.

Two callback messages are generated: VM_VOLUME_ACCEPTED_EVENT and VM_VOLUME_CREATED_EVENT.

Following is an example of the VM_VOLUME_CREATED_EVENT ESC REST callback message:

```
{
    "escTransactionId": "5acac790-9213-45c0-8fde-9dd7d3111fdb",
    "eventType": "VM_VOLUME_CREATED_EVENT",
    "eventSourceContext": null,
    "eventTargetContext": null,
    "message": "Create volume snaphot request completed",
    "stateMachineEventNBInfo": {
        "id": "de9440c0-1342-441d-a16e-5c8267231ae5",
        "message": {},
        "logNames": [],
        "keywords": {},
        "actionInfo": {},
        "stackTrace": ""
    },
    "escParameter": {
        "external_volume_id": "c3cd5d13-63bf-49f0-b864-df3bc024d5e4",
        "size": "2",
        "sizeunit": null,
        "bus": "virtio",
        "type": "LVM",
        "outOfBand": "false",
        "bootIndex": null,
        "name": "daves-ooband-bootable-volume-for-restore",
        "format": null,
        "deviceType": null,
        "storageLocation": null,
        "external_tenant_id": null,
```

```
        "internal_tenant_id": null,
        "internal_volume_id": null,
        "volid": null,
        "event_type": null,
        "image": null
    },
    "vmUpdateType": null,
    "requestDetails": null,
    "statusCode": "201",
    "notificationOnlyEvent": false
}
```

Polling Configuration Parameters

Vim Manager configuration properties can be used to control the interval between calls to OpenStack to check the status of the create volume operation, along with the maximum number of minutes an operation is allowed before timing out with an error.

These properties are:

- vim.asyncpoller.volume.create.poll.secs # default 15, the number of seconds between polls
- vim.asyncpoller.volume.create.timeout.mins # default 20, maximum number of minutes for the create volume operation

```
[admin@localhost]$ sudo cat /opt/cisco/esc/vimmanager/application.properties
vim.asyncpoller.volume.create.poll.secs=5
vim.asyncpoller.volume.create.timeout.mins=10

[admin@localhost]$ sudo escadm vimmanager restart
Stopping vimmanager service: [OK]
Starting vimmanager service: [OK]

[admin@localhost]$ sudo escadm vimmanager show
VimManager System Configurations.
{
    "ccp.pollRetries": "200",
    "ccp.pollRetryDelaySecs": "15",

    . . .

    "vim.asyncpoller.volume.create.poll.secs": "5",
    "vim.asyncpoller.volume.create.mins": "10",

    . . .

    "vmware.ovftool.params": "--acceptAllEulas --disableVerification --noSSLVerify
--allowExtraConfig",
    "vmware.powerOnRetry": "8"
}
```

Note In an HA setup, the application properties file will need to be copied to both nodes.

Managing Individual and Composite VNFs

An individual service consists of a single VNF. A coupled service or a composite VNF consists of several VMs of different types. The ESC interface receives VM interdependency information from the northbound system, and uses this information during VM and VNF creation, and life cycle management. Interdependency could include VM specific workflow in the group of VMs in a single VNF, VNF monitoring and scalability and so on.

Create, read, update and delete operations are allowed on the VMs. To add more VM instances to a deployed VNF using static IP, you must provide additional IP addresses into the static IP pool. If you are using an existing static IP deployment, the minimum number of VMs is altered.

If the new minimum value, which is the number of VMs is greater than the active VMs, a new VM is added to the service. If the value is greater than the max value, the update is rejected.

PART V

Monitoring, Scaling, and Healing

CHAPTER 36

Monitoring Virtual Network Functions

Monitoring the VNFs

After deploying VNFs, they are monitored periodically to check their health and workload. Monitoring is based on the definition of metrics within the KPI section of the deployment data model. As described in the KPIs section the metric type determined not only the variable to monitor, but also the collector action to be executed. ESC allows you to define the metrics to be monitored and the actions that needs to be executed when the conditions are met. These metrics and actions are defined in the *deployment datamodel*. Several monitoring methods are used to monitor the VNFs. You can monitor the following:

- VM aliveness
- VM variables for Disk usage, Memory, CPU, Network throughput
- ICMP message on the VM monitoring interface.

Pre-requisites for Monitoring

The following pre-requisites must be met for the VMs to be monitored by ESC:

- Monitoring is enabled for VMs that are successfully deployed. The deployed VMs must be alive.
- KPI must be configured in the data model with the monitoring parameters.

Monitoring and Action Execution Engine

Monitoring is based on the definition of metrics within the KPI section of the deployment datamodel. As described in the KPIs section the metric type determines not only the variable to monitor, but also the collector action to be executed. The monitoring engine comprises of metrics and actions.

1. Metrics
2. Actions

The metrics and actions <metadata> section describes the properties or entries controlling the programmable aspect of the engine.

Metrics Section

The metrics section is as follows:

```
<metrics>
     <metric>
     <name>{metric name}name>
           <type>{metric type}type>
           <metaData>
           <type>{monitoring engine action type}</type>
         <properties>
         <property>
         <name></name>
        <value></value>
       </property>
       : : : : : : :
      <properties/>
     </metaData>
    </metric or action>
     : : : : : : : :
<metrics>
```

Table 24: Metric Section Description

Tag name	Description	Values
name	A user defined metric name. The metric name must be unique.	
type	Dynamic mapping supported type.	MONITOR_SUCCESS_FAILURE MONITOR_THRESHOLD MONITOR_COMPUTE_THRESHOLD

Metric Metadata Section

The purpose of the metadata section is to provide information specific to the monitoring solution.

Table 25: Metric Metadata Section

Tag Name	Description	Values
type	The action type, values are a one to one mapping with MONA supported actions.	custom_script custom_script_threshold snmp_get_threshold
properties	A container for a list of properties (name/value) that will be passed to selected action. The properties are defined by the list of expected monitoring and actions attributes.	Properties are based on the selected action type.

Actions Section

The actions section is as follows:

```
<actions>
           <action>
           <name>{action name}name>
           <type>{action type}type>
           <metaData>
           <type>{monitoring engine action type}</type>
           <properties>
           <property>
           <name></name>
           <value></value>
          </property>
           : : : : : : :
           <properties/>
          </metaData>
         </action>
           : : : : : : : :
          <actions>
```

Table 26: Actions

Tag Name	Description	Values
name	A user defined action name. The action name must be unique.	One of the main requirements is also to have the chosen name prefixed with TRUE or FALSE to allow mapping between ESC data model rule and dynamic actions, just for MONITOR_SUCCESS_FAILURE.
type	Supported type.	ESC_POST_EVENT SCRIPT CUSTOM_SCRIPT

Actions Metadata Section

The purpose of the metadata section is to provide information specific to the monitoring solution.

Table 27: Action Metadata section

Tag Name	Description	Values
type	The action type, values are a one to one mapping with monitoring and actions engine supported actions.	icmp_ping icmp4_ping icmp6_ping esc_post_event script custom_script snmp_get snmp_get_threshold
properties	A container for a list of properties (name/value) that will be passed to selected action. The properties are defined by the list of expected monitoring and action attributes.	Properties are based on the selected action type.

For more details see the KPIs, Rules and Dynamic Mapping APIs section.

Table 28: Supported Action Types

Type	Properties and their description
icmp_ping	• ip_address • enable_events_after_success: Boolean controlling when MONA will start forwarding events notifications. If set to true, notification will be forwarded only after the first transition to success. • timeOut: Default set to 5 seconds
icmpv4_ping	• ip_address • enable_events_after_success: Boolean controlling when MONA will start forwarding events notifications. If set to true, notification will be forwarded only after the first transition to success. • timeOut: Default set to 5 seconds

Type	Properties and their description
icmpv6_ping	• ip_address • enable_events_after_success: Boolean controlling when MONA will start forwarding events notifications. If set to true, notification will be forwarded only after the first transition to success. • timeOut: Default set to 5 seconds
script	• script_filename: Full path to the script to be executed (The script has to be located on the ESC VM). • wait_for_script: Boolean controlling if the action is waiting for the completion of the script. (Not actually exercised)
custom_script	script_filename: Full path to the script to be executed (The script has to be located on the ESC Manager VM).
custom_script_threshold	• script_filename: Full path to the script to be executed (The script has to be located on the ESC Manager VM). • threshold
post_esc_event	• esc_url • vm_external_id • vm_name • esc_event • event_name
snmp_get	• target_oid , agent_address, IP Address of the SNMP agent (IPV4/IPV6 is supported) • agent_port: Port used by the SNMP Agent. • agent_protocol: Protocol used by the SNMP Agent (tcp/udp). • Community: SNMP v2c community string used by the SNMP agent

Type	Properties and their description
snmp_get_threshold	• target_oid: Object Identifier that will be used for the threshold comparison. • agent_address: IP Address of the SNMP agent (IPV4/IPV6 is supported). • agent_port: Port used by the SNMP Agent. • agent_protocol: Protocol used by the SNMP Agent (tcp/udp). • community: SNMP v2c community string used by the SNMP agent
snmp_get_threshold_ratio	• oid_total_value: Object Identifier that will be used to represent the current for the ratio/percentage computation. • oid_current_value: Object Identifier that will be used to represent the current for the ratio/percentage computation. Algorithm to be used for the computation of the percentage/ratio. We are currently supporting two algorithms: COMPUTE_TOTAL_CURRENT_BASED , COMPUTE_TOTAL_AVAILABILITY_BASED. • agent_address: IP Address of the SNMP agent (IPV4/IPV6 is supported). • agent_port: Port used by the SNMP Agent. • agent_protocol: Protocol used by the SNMP Agent (tcp/udp). • community: SNMP v2c community string used by the SNMP agent.

Properties and Runtime Parameter Injection

The properties list passed to the selected action type supports the capabilities to automatically inject runtime value for some selected parameters. For example, runtime value of the virtual machine ip_address or its name can be passed automatically as arguments to the selected action.

Following are some of the parameters that can be passed to the scripts at the time of execution. Parameter value is populated at runtime only if :

- the parameter is a supported one, and
- its value is empty within the dynamic-mappings.xml file.

Otherwise, the value defined within the script is passed as is.

Table below shows the parameters passed during runtime.

esc_url	The URL of the Elastic Services Controller.
vm_external_id	The external id of the managed VM.
vm_name	The name of the managed VM.
vm_mac_address	The mac address of the managed VM.
vm_external_host_id	The VM external host Identifier.
vm_external_host_name	The VM external host name.
vm_group_name	The VM group name.
ip_address	The VM IP Address.
event_name	The ESC event name.

Note The properties list passed to the selected action, is not bound by the parameters in the action type. A script designer can define its own parameters. However, the values have to be provided.

Monitoring Methods

ESC uses several monitoring methods to monitor the VNFs. You must configure the KPI data model for the monitoring methods.

ICMP Ping Monitoring

Ping monitoring assess the liveliness or reachability of a VNF.

If a VM is unreachable the healing of the VM is triggered. At every defined interval, ESC polls the metric value and sends alarms whenever needed. The number of polls, metric value, and other configuration are set in the KPI datamodel.

SNMP Monitoring

In SNMP Monitoring, load of the VM such as memory usage and CPU in a given period is monitored. The SNMP Get operation is used to assess the liveliness or reachability of a VNF. In this monitoring method, only the success or failure is monitored.

SNMP Threshold Monitoring

In SNMP threshold monitoring, you can set the upper and lower threshold levels in the kpi section of the data model. Actions are performed based on the upper and lower threshold levels.

Custom Monitoring

In ESC 2.1 or earlier, the Dynamic Mapping XML is required to map the actions and metrics defined in the datamodel to the valid actions and metrics available in the monitoring agent. The file is stored on the ESC VM and is modified using a text editor. This method is error prone and modification for an HA pair requires to take place on both the active and standby VMs. ESC 2.2 or later does not have an *esc-dynamic-mapping* directory and *dynamic_mappings.xml* file. The CRUD operations for mapping the actions and the metrics is now available through REST API in ESC. For more information, see KPIs, Rules and Metrics, on page 169.

Monitoring a VM

Cisco Elastic Services Controller monitors the VM to detect any erroneous condition. ESC uses one of its monitoring methods to detect actions on a VM, and passes this information to the rules service for processing. The monitoring request comes from the northbound client along with VNF deployment requests.

There are two sections in the datamodel xml file which define the events and rules: KPI and Rule.

Based on the monitors and actions, rules are triggered.

```
<kpi>
   <event_name>VM_ALIVE</event_name>
   <metric_value>50</metric_value>
   <metric_cond>GT</metric_cond>
   <metric_type>UINT32</metric_type>
   <metric_occurrences_true>3</metric_occurrences_true>
   <metric_occurrences_false>3</metric_occurrences_false>
   <metric_collector>
      <type>ICMPPing</type>
      <nicid>0</nicid>
      <poll_frequency>15</poll_frequency>
      <polling_unit>seconds</polling_unit>
      <continuous_alarm>false</continuous_alarm>
   </metric_collector>
</kpi>
```

In the example above, an event is sent to check whether the VM is alive. The VM is pinged at regular intervals, and based on the result VM_ALIVE event is sent to the rules engine along with the details of the VM.

The rules engine receives events from the monitoring engine. The rules engine can handle simple to complex events. Based on the event received an action is triggered.

If the VM is not alive, based on the event the actions defined in the <rule> section are triggered. This can be found in the dep.xml datamodel.

```
<rules>
   <admin_rules>
      <rule>
         <event_name>VM_ALIVE</event_name>
         <action>ALWAYS log</action>
         <action>FALSE recover autohealing</action>
         <action>TRUE servicebooted.sh</action>
      </rule>
   </admin_rules>
</rules>
```

The rules section describes the actions to be executed once a monitoring event has been detected. The dynamic mapping API drives the rules based on keywords.

In the above example, the following actions are performed based on the given condition:

- ALWAYS log: Whether the event is pingable or not, the details are logged.

- TRUE servicebooted.sh: The action identified by this keyword in the dynamic mapping API is triggered when the VM moves from a non-pingable to a pingable state. The serviceboot script informs ESC that the VM is alive allowing it to transition the VMs state.
- FALSE recover autohealing: The action identified by this keyword will be triggered and the VM will be recovered without the administrator's intervention.

Monitoring log files for troubleshooting are available at `/var/log/mona`.

Monitoring the VM Network Status

When using ICMP ping monitoring, if ESC receives a VM Down event, the healing workflow attempts to recover the VM with the recovery policy. If there is an issue with the network interface or IP route from ESC to the VNF. For example, if the gateway is down, it might trigger the VM Down event incorrectly, which leads to an unnecessary recovery.

The check interface function does further scan to the network route by checking the health status of all the network interfaces and the operation state of the gateway. If there is any problem in the network environment, it assumes the VNF is alive.

The VM_NETWORK_STATE event is sent to northbound if ESC detects a network issue or if any existing issue is fixed (autohealing).

The following failure notification is sent to northbound:

```
16:13:15,567 14-Mar-2018 WARN  ===== SEND NOTIFICATION STARTS =====
16:13:15,567 14-Mar-2018 WARN  Type: VM_NETWORK_STATE
16:13:15,567 14-Mar-2018 WARN  Status: FAILURE
16:13:15,567 14-Mar-2018 WARN  Status Code: 500
16:13:15,567 14-Mar-2018 WARN  Status Msg: Warning: VM
[NG_G1_0_46fdcf70-f4ea-4289-ae79-08674e7d6f42] has a network problem: Network interface not
 healthy, please check.
16:13:15,567 14-Mar-2018 WARN  Tenant: tenant2
16:13:15,567 14-Mar-2018 WARN  Deployment ID: 455d2407-9dda-4203-95b0-724c4a651720
16:13:15,567 14-Mar-2018 WARN  Deployment name: NG
16:13:15,567 14-Mar-2018 WARN  VM group name: G1
16:13:15,567 14-Mar-2018 WARN  VM Source:
16:13:15,567 14-Mar-2018 WARN      VM ID: 4bee016a-6b30-43ff-a249-157a07d9b4db
16:13:15,567 14-Mar-2018 WARN      VM Name: NG_G1_0_46fdcf70-f4ea-4289-ae79-08674e7d6f42
16:13:15,568 14-Mar-2018 WARN      VM Name (Generated):
NG_G1_0_46fdcf70-f4ea-4289-ae79-08674e7d6f42
16:13:15,568 14-Mar-2018 WARN      VIM ID: default_openstack_vim
16:13:15,568 14-Mar-2018 WARN      VIM Project: tenant2
16:13:15,568 14-Mar-2018 WARN      VIM Project ID: 62afb63cd28647a7b526123cac1ba605
16:13:15,568 14-Mar-2018 WARN      Host ID:
b83004159a46c20bc8383927c2231067bb0c1905b4b4c28475653190
16:13:15,568 14-Mar-2018 WARN      Host Name: my-server-50
16:13:15,568 14-Mar-2018 WARN  =====  SEND NOTIFICATION ENDS  =====
```

The following success notification is sent to northbound when the network problem is fixed.

```
16:13:19,141 14-Mar-2018 INFO  ===== SEND NOTIFICATION STARTS =====
16:13:19,141 14-Mar-2018 INFO  Type: VM_NETWORK_STATE
16:13:19,142 14-Mar-2018 INFO  Status: SUCCESS
16:13:19,142 14-Mar-2018 INFO  Status Code: 200
16:13:19,142 14-Mar-2018 INFO  Status Msg: Network of VM
[NG_G1_0_46fdcf70-f4ea-4289-ae79-08674e7d6f42] has been restored.
16:13:19,142 14-Mar-2018 INFO  Tenant: tenant2
16:13:19,142 14-Mar-2018 INFO  Deployment ID: 455d2407-9dda-4203-95b0-724c4a651720
16:13:19,142 14-Mar-2018 INFO  Deployment name: NG
```

```
16:13:19,142 14-Mar-2018 INFO  VM group name: G1
16:13:19,142 14-Mar-2018 INFO  VM Source:
16:13:19,142 14-Mar-2018 INFO      VM ID: 4bee016a-6b30-43ff-a249-157a07d9b4db
16:13:19,142 14-Mar-2018 INFO      VM Name: NG_G1_0_46fdcf70-f4ea-4289-ae79-08674e7d6f42
16:13:19,142 14-Mar-2018 INFO      VM Name (Generated):
NG_G1_0_46fdcf70-f4ea-4289-ae79-08674e7d6f42
16:13:19,142 14-Mar-2018 INFO      VIM ID: default_openstack_vim
16:13:19,143 14-Mar-2018 INFO      VIM Project: tenant2
16:13:19,143 14-Mar-2018 INFO      VIM Project ID: 62afb63cd28647a7b526123cac1ba605
16:13:19,143 14-Mar-2018 INFO      Host ID:
b83004159a46c20bc8383927c2231067bb0c1905b4b4c28475653190
16:13:19,143 14-Mar-2018 INFO      Host Name: my-server-50
16:13:19,143 14-Mar-2018 INFO  =====  SEND NOTIFICATION ENDS  =====
```

For information on monitoring VNFs using ETSI API, see the *Cisco Elastic Services Controller ETSI NFV MANO Guide*.

Notification for VM Monitoring Status

ESC sends the VM_MONITORING_STATUS under the following conditions:

Error occurs during a set/unset monitoring operations, if a monitoring script is missing or when a monitoring timer expires when monitors are at rest after ESC switchover.

VM_MONITOR_STATUS notification is sent to NB. ESC doesn't monitor the VM, it fails to enter the recovery process. To enable monitoring after the failure, you must disable, and then enable the monitoring.

Notifications

```
WARN ===== SEND NOTIFICATION STARTS =====
WARN Type: VM_MONITORING_STATUS
WARN Status: FAILURE
WARN Status Code: 500
WARN Status Msg: No response from the monitor
WARN Tenant: tenant
WARN Deployment ID: 02cc4018-e4e3-4974-884a-f9fee17d7040
WARN Deployment name: dep
WARN VM group name: g1
WARN VM Source:
WARN VM ID: 6aa98b79-9d35-442a-9abb-f611e6316083
WARN VM Name: dep_g1_0_7fdae2a6-5095-4071-9c50-fb80c0e6b80e
WARN VM Name (Generated): dep_g1_0_7fdae2a6-5095-4071-9c50-fb80c0e6b80e
WARN VIM ID: default_openstack_vim
WARN VIM Project: tenant
WARN VIM Project ID: 33bf6768e45445da87feed838b248849
WARN Host ID: 79e4104d1d33de80aab13205b1e3c61d64aa4b61230c8b7b064b2891
WARN Host Name: my-ucs-62
WARN ===== SEND NOTIFICATION ENDS =====
```

Monitoring Operations

You can set and unset monitoring of VMs using RESTful interface.

A payload is required to monitoring VMs:

```
POST ESCManager/v0/{internal_tenant_id}/deployments/vm/{vm_name}
```

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<vm_operation xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <operation>enable_monitoring</operation>
   <force>false</force>
</vm_operation>
```

You must mention enable_monitoring to set VM monitoring, and disable_monitoring to unset VM monitoring in the operation field.

Note When a user reboots the VM from the ESC portal, the monitoring is automatically enabled.

CHAPTER 37

Monitoring VNF Using D-MONA

The ESC Monitoring and Action (MONA) monitors VNFs that are deployed by ESC. To maintain accuracy, it executes actions, such as ping, custom_scripts, and so on at specific intervals.

Onboarding D-MONA

The following prerequisites must be fulfilled before deploying D-MONA:

Prerequisites

- Ensure Connectivity exists between ESC and the D-MONA.
- Ensure connectivity exists between the D-MONA and the deployed VNFs.

Upon successful deployment, D-MONA is monitored by the local MONA running on the ESC VM.

Note Monitoring of D-MONA by another D-MONA is not supported.

Deploying D-MONA

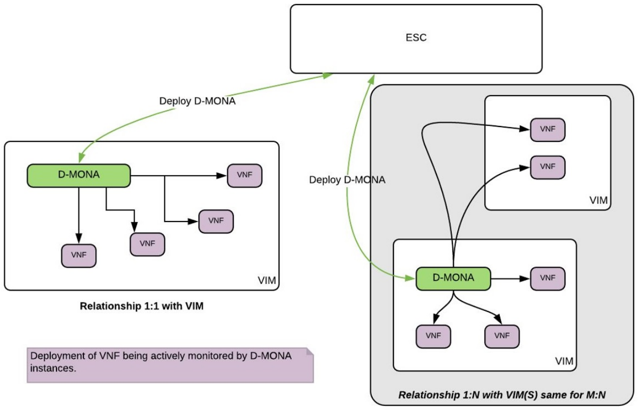
From ESC 5.3 or later, 1:1 mapping is not required. It supports explicit D-MONA deployment.

- In this scenarios, multiple D-MONA Instances can be deployed.
- VNFs can be deployed under, or migrated to specified monitoring agent.

For using D-MONA in your infrastructure, you must:

1. Deploy the D-MONA with the monitoring infrastructure.
2. Deploy the VNFs using the D-MONA for the monitoring.

Figure 3: D-MONA Deployment Types



If you are not using D-MONA for monitoring, see Monitoring Virtual Network Functions section.

The following table lists the D-MONA VM flavors for large scale deployments:

Deployment	Number of VMs	Virtual CPU per VM	Virtual Memory (GB) per VM	Virtual Hard Disk (GB)per VM	Number of total VMs Supported
D-MONA	1	4	8	40	1500

Configuring D-MONA

While configuring D-MONA, you can view 2 types of runtime behavior, one where you can view the full behavior expected from a typical ESC deployment, and the other one depicts the capabilities provided by D-MONA.

The D-MONA runtime behavior is controlled by the day-0 configuration that is provided to the VM at deployment time. For more information on day zero configuration, see the Day Zero Configuration section.

You must provide the notification URL for HA Active/Standby and Standalone. However, for the Active/Active HA, the URL is auto-generated or computed during the deployment.

D-MONA Day Zero Configuration

The following example shows D-MONA SSH VM access configuration:

```
<configuration>
 <dst>--user-data</dst>
 <file>file:///opt/cisco/esc/esc-config/dmona/user-data.template</file>
 <variable>
  <name>vm_credentials</name>
  <val>REPLACED_WITH_GENERATED_PWD</val>
```

```
 </variable>
</configuration>
```

The following example shows the notification URL for HA Active/Standby and Standalone:

```
<variable>
  <name>notification.url</name>
  <val>
    http(s)://xxx.xx.x.xx:xxxx/ESCManager/dmona/api/events/notif
  </val>
</variable>
```

The `vm_credentials` passes the encrypted password to admin for SSH access to the D-MONA.

The following example shows the D-MONA ESC certificate configuration:

```
<configuration>
 <dst>/opt/cisco/esc/moan/dmona.crt</dst>
 <data>$DMONA_CERT</data>
</configuration>
```

The following example shows the D-MONA application user data configuration:

```
<configuration>
<dst/opt/cisco/esc/mona/config/application-dmona.properties</dst>
 <file>file:///opt/cisco/esc/esc-config/dmona/application-dmona.template</file>
 <variable>
  <name>monitoring.agent</name>
  <val>true</val>
 </variable>
 <variable>
  <name>monitoring.agent.vim.mapping</name>
  <val>true</val>
 </variable>
 <!—Used to enable Basic Authentication for communication with the D-MONA Application.->
 <variable>
  <name>security_basic_enabled</name>
  <val>true</val>
 </variable>
 <variable>
  <name>security_user_name</name>
  <val>REPLACED_WITH_USER_NAME</val>
 </variable>
 <variable>
  <name>security_user_password</name>
  <val>REPLACED_WITH_USER_PASSWORD</val>
 </variable>

</configuration>
```

The following example shows the D-MONA day-0 template file for CSP:

Upload the D-MONA day-0 template to the `/var/tmp/` directory in all the ESC instances with proper access permission prior to deployment.

```
#cloud-config
users:
 - name: admin          # The user's login name
   gecos: admin         # The user name's real name
   groups: esc-user     # add admin to group esc-user
   passwd: $vm_credentials
                        # The hash -- not the password itself -- of the password you want
                       #           to use for this user. You can generate a safe hash via:
```

```
                        #                mkpasswd --method=SHA-512 --rounds=4096
   lock-passwd: false   # Defaults to true. Lock the password to disable password login
                        # Set to false if you want to password login
   homedir: /home/admin # Optional. Set to the local path you want to use. Defaults to
/home/<username>
   sudo: ALL=(ALL) ALL  # Defaults to none. Set to the sudo string you want to use

ssh_pwauth: True        # Defaults to False. Set to True if you want to enable password
authentication for sshd.
write_files:
# ESC Configuration
- path: /opt/cisco/esc/esc-config/esc-config.yaml
  content: |
    resources:
      mona:
        dmona: true
- path: /etc/sysconfig/network-scripts/ifcfg-eth0
  content: |
    DEVICE="eth0"
    BOOTPROTO="none"
    ONBOOT="yes"
    TYPE="Ethernet"
    USERCTL="yes"
    IPADDR="${NICID_0_IP_ADDRESS}"
    NETMASK="${NICID_0_NETMASK}"
    GATEWAY="${NICID_0_GATEWAY}"
    DEFROUTE="yes"
    NM_CONTROLLED="no"
    IPV6INIT="no"
    IPV4_FAILURE_FATAL="yes"
- path: /etc/sysconfig/network-scripts/ifcfg-eth1
  content: |
    DEVICE="eth1"
    BOOTPROTO="none"
    ONBOOT="yes"
    TYPE="Ethernet"
    USERCTL="yes"
    IPADDR="${NICID_1_IP_ADDRESS}"
    NETMASK="${NICID_1_NETMASK}"
    GATEWAY="${NICID_1_GATEWAY}"
    DEFROUTE="yes"
    NM_CONTROLLED="no"
    IPV6INIT="no"
    IPV4_FAILURE_FATAL="yes"
runcmd:
- [ cloud-init-per, once, apply_network_config, sh, -c, "systemctl restart network"]
- [ cloud-init-per, once, copy_dmona_config, sh, -c, "cp -RT /media/cdrom/opt/cisco/esc/mona/
 /opt/cisco/esc/mona/"]
- [ cloud-init-per, once, esc_service_start, sh, -c, "chkconfig esc_service on && service
esc_service start"] # You must include this line
```

Deploying VNFs with Explicit D-MONA Monitoring Agent

From ESC 5.3 onwards, ESC allows to explicitly specify the D-MONA identifier to monitor a VNF. Following are the steps to deploy VNFs with explicit VNF to D-MONA monitoring agent:

Procedure

Step 1 Deploy a D-MONA with monitoring.agent.vim.mapping property in the D-MONA day-0 config omitted or set to false.

The following example shows the day 0 config of a D-MONA datamodel where monitoring.agent.vim.mapping is set to false.

```
<configuration>
        <dst>/opt/cisco/esc/mona/config/application-dmona.properties</dst>
        <file>file:///opt/cisco/esc/esc-config/dmona/application-dmona.template</file>
        <variable>
            <name>monitoring.agent</name>
            <val>true</val>
        </variable>
        <!-- property for one to one mapping - omit or set to false for explicit VNF to
D-MONA mapping-->
        <variable>
            <name>monitoring.agent.vim.mapping</name>
            <val>false</val>
        </variable>
        <!-- property to enable basic auth in dmona. Not to be confused with basic auth for
 esc -->
        <variable>
            <name>security_basic_enabled</name>
            <val>true</val>
        </variable>
        <variable>
            <name>security_user_name</name>
            <val>REPLACE_WITH_USER_NAME</val>
        </variable>
        <variable>
            <name>security_user_password</name>
            <val>REPLACE_WITH_USER_PASSWORD</val>
        </variable>
</configuration>
```

Step 2 Deploy the VNF by specifying the monitoring_agent parameter in the KPI config of the deployment datamodel.

The tag *<monitoring_agent>* is used as an explicit identification of a distributed mona deployment which monitors the VNF. When the tag is present, ESC looks for a distributed mona deployment with that exact deployment name. The D-MONA identifier is specified in the URI by using a specific scheme to represent a previously deployed D-MONA VNF.

For example, `dmonaName://<D_MONA_DEP_NAME>` Replace `<D_MONA_DEP_NAME>` with deployment name of Distributed MONA instance.

The following example shows the KPI config of a VNF datamodel with monitoring agent specified:

```
<kpi>
    <event_name>VM_ALIVE</event_name>
    <!-- specify dmona deployment name using dmonaName:// URI format-->
    <monitoring_agent>dmonaName://D-MONA-OTTAWA</monitoring_agent>
    <metric_value>1</metric_value>
    <metric_cond>GT</metric_cond>
    <metric_type>UINT32</metric_type>
    <metric_collector>
    <type>ICMPPing</type>
    <nicid>0</nicid>
    <poll_frequency>3</poll_frequency>
    <polling_unit>seconds</polling_unit>
    <continuous_alarm>false</continuous_alarm>
```

```
    <monitoring_public_ip>true</monitoring_public_ip>
    </metric_collector>
</kpi>
```

Note ESC allows only one single monitoring agent per VNF.

Troubleshooting Monitoring Status

To determine if a VNF is monitored by a monitoring agent for D-MONA, run the following command:

```
curl -u username:pwd -H 'Accept:application/json'
http://localhost:8080/ESCManager/v0/api/monitoring/agents/config
```

The sample below shows the result:

```
{
    "a8345881-adc8-4d16-8741-9d105592c676": {
        "monitoringAgents": [
            {
                "name": "sample-dmona-10",
                "notificationUrl":
"https://172.16.235.73:8443/ESCManager/dmona/api/events/notif",
                "oneToOneMapping": false,
                "state": "ACTIVE",
                "uri": "https://172.16.235.81:8443/mona/v1/rules",
                "vimId": "OPENSTACK_VIMCONN_pf-ucs-20",
                "vnfData": [
                    {
                        "deploymentExternalId": "785e170c-55b5-4df7-929f-d34f052e4616",
                        "deploymentName": "dmona-10-vnf-121-f2b1df6d",
                        "state": "MONITORED",  <============ Monitoring state for DMONA
                        "vmGroupName": "vm1"
                    },
                    {
                        "deploymentExternalId": "2e42c8d9-51fa-4de8-a260-d3a3429be7d4",
                        "deploymentName": "dmona-10-vnf-442-faa43053",
                        "state": "MONITORED",  <============ Monitoring state for DMONA
                        "vmGroupName": "vm1"
                   }
                ]
            },
            {
                "name": "local_mona",
                "notificationUrl": "",
                "oneToOneMapping": false,
                "state": "ACTIVE",
                "uri": "http://localhost:8090/mona/v1/rules",
                "vimId": "N/A",
                "vnfData": [
                    {
                        "deploymentExternalId": "9501376e-e29e-4c99-b5fb-66ab66de45b7",
                        "deploymentName": "sample-dmona-2",
                        "state": "N/A",   <============ Local Mona monitoring state is not
 available
                        "vmGroupName": "g1"
                    }
                ]
            }
```

```
        ]
    }
}
```

Recovering the D-MONA from One VIM Instance to Another

If a D-MONA agent fails, ESC can recover it quickly, ensuring minimal downtime and reinstating monitoring of the deployed VNFs at the earliest; However, the monitored VNFs remain unmonitored while the agent is recovered. The VNFs monitored by D-MONA remain in the last known state until the D-MONA becomes active again.

The VNFs monitoring status restores only if D-MONA successfully recovers with ESC while reprogramming each VNF monitoring rule. Until then, the D-MONA agent's state remains *UNKNOWN*, and the VNFs' state remains *UNMONITORED.*

The following example shows the state of D-MONA and VNFs in the monitoring agent API when D-MONA is down:

```
{
    "name": "Test-dmona-dep-1",
    "notificationUrl": "",
    "oneToOneMapping": false,
    "state": "UNKNOWN",
    "uri": "https://172.29.0.15:8443/mona/v1/rules",
    "vimId": "default_openstack_vim",
    "vnfData": [
        {
            "deploymentExternalId": "70d7f1f0-362e-4d2b-a89b-4877d8bfabf4",
            "deploymentName": "Test-dep-2",
            "state": "UNMONITORED",
            "vmGroupName": "g1"
        }
    ]
}
```

Disaster Recovery of a Monitoring Agent

If the deployed distributed D-MONA is unreachable due to the unavailability of the VIM, then the D-MONA can be recovered on another VIM by sending a *HealVnfRequest* with the cause `VIM_FAILURE`

Use the following to recover D-MONA on another VIM instance:

- Initiate a manual *HealVnfRequest,* as per the following SOL003 example:

Method type :

`POST`

VNFM Endpoint:

`/vnf_instances/{vnfInstanceId}/heal`

HTTP Request Header:

`Content-Type:application/json`

Request Payload (ETSI data structure: HealVnfRequest):

```
{
  "cause":"VIM_FAILURE"
}
```

- The recovery request is rejected if the Grant from the NFVO does not include the new *vimConnectionInfo* which identifies the VIM to redeploy the D-MONA VNF.
- When the *HealVnfRequest* completes successfully, the D-MONA VNF recreates in the new VIM and continues to monitor all the VNFs it had previously

Note The original deployment is not removed from the old VIM. Manually remove the previous D-MONA once the old VIM is reachable.

Failover in Active/Active HA

A failover in an ESC Active/Active HA deployment transfers the VNFs owned by the failed ESC instance to other ESC instances in the cluster.

If a D-MONA deployment transfers from the failed ESC instance, then it updates to UNKNOWN state in the monitoring agent API. VNFs monitored by transferred D-MONA reconciles when the D-MONA monitoring agent state is ACTIVE.

As per any D-MONA deployment, the VNFs monitored by transferred D-MONA stay in the last known state until the D-MONA becomes active once again.

Retrieving D-MONA Logs

Access the D-MONA with the `vm_credentials` password that was provided as part of the D-MONA day-0 configuration.

To retrieve the D-MONA logs, use the following command:

```
<security_user_name>:<security_user_password>
```

Where ip-address is the IP Address of the targeted D-MONA and username, password are the username and password provided as day-0 configuration at deployment of the D-MONA.

For complete list of all ESC logs, see ESC Logs section in the ESC Administration Guide.

For ETSI-related information, see Monitoring VNF Using D-MONA chapter in the Cisco Elastic Services Controller ETSI NFV MANO User Guide.

Resetting the Monitoring Rules for D-MONA

The Monitoring and Action (MONA) monitors and executes actions, such as ping, custom_scripts, and so on at specific intervals to maintain accuracy.

The local MONA keeps a track of the last known startup time of the polled D-MONA process. The status code 200 indicates the successful request. In case of successful request, the local MONA compares the last known startup time with the returned startup time from the polled application. On DMONA restart, the recovery setup starts automatically.

To enable the start time check, you must set `application_startup_time` in the dep.xml.

However, if the `application_startup_time` is not present or set to false, then DMONA reboot check is disabled. You must set this property for DMONA deployment.

Note Backward compatibility is not supported. It must be set for version 5.3 and later only.

Following is a sample deployment model for D-MONA:

```
<?xml version="1.0"?>
<esc_datamodel xmlns="http://www.cisco.com/esc/esc">
  <tenants>
    <tenant>
      <name>A_tenant_name</name>
      <deployments>
        <deployment>
          <name>dmona_deployment</name>
          <vm_group>
            <name>g1</name>
            <image>ESC-5_3_0_31</image>
            <flavor>m1.large</flavor>
            <bootup_time>120</bootup_time>
            <recovery_wait_time>0</recovery_wait_time>
            <interfaces>
              <interface>
                <nicid>0</nicid>
                <network>esc-net</network>
              </interface>
            </interfaces>
            <kpi_data>
              <kpi>
                <event_name>VM_ALIVE</event_name>
                <metric_value>1</metric_value>
                <metric_cond>GT</metric_cond>
                <metric_type>UINT32</metric_type>
                <metric_occurrences_true>1</metric_occurrences_true>
                <metric_occurrences_false>5</metric_occurrences_false>
                <metric_collector>
                  <type>HTTPGET</type>
                  <nicid>0</nicid>
                  <poll_frequency>3</poll_frequency>
                  <polling_unit>seconds</polling_unit>
                  <continuous_alarm>false</continuous_alarm>
                  <properties>
                 <!-- Set to true to enable start time check --->
                <property>
                <name>application_startup_time</name>
                <value>true</value>
                </property>
                 <property>
                      <name>protocol</name>
                      <value>https</value>
                    </property>
                    <property>
                      <name>port</name>
                      <value>8443</value>
                    </property>
                    <property>
                      <name>path</name>
                      <value>mona/v1/health/status</value>
                    </property>
```

```
                    </properties>
                  </metric_collector>
                </kpi>

              </kpi_data>
[...]
           </vm_group>
          </deployment>
        </deployments>
      </tenant>
    </tenants>
  </esc_datamodel>
```

CHAPTER 38

Migrating the Monitoring Agent

Migrating the Monitoring Agent

Each ESC instance has an agent to monitor it to enable ESC to control recovery and scaling operations. Following are the various scenarios that need migration of the monitoring agent:

1. Migrating from **local** to **distributed**

   For example:

   When introducing a new D-MONA into a data center.

2. Migrating from **distributed** to **local**

   For example:

   When performing a software upgrade.

3. Migrating from **distributed** to **distributed**

   For example:

   When performing load balancing.

4. Migrating many instances in quick succession from **distributed** to **distributed**

   For example:

   Disaster recovery

Follow the following procedures for migrating the monitoring agent:

Procedure

Step 1 Adding/editing the <monitoring_agent> tag value in the KPI config section of the deployment datamodel:

a) To migrate to a D-MONA do the following:

   Set <monitoring_agent>dmonaName://dmona-dep-name</monitoring_agent> where dmona-dep-name is the deployment name of the D-MONA.

b) To migrate to local mona do the following:

Set <monitoring_agent>dmonaName://local_mona</monitoring_agent> where local_mona is a special identifier introduced in ESC 5.3 for local mona

Step 2 Performing a service update using the updated deployment datamodel:

When you perform a service update, it will unset the monitor on the current monitoring agent, update the VNF with new monitoring agent, and set monitor on the new monitoring agent.

For more information on the monitoring_agent parameter, see the Deploying VNFs with Explicit D-MONA Mapping chapter.

Post Migration Notifications

ESC sends three notifications to NorthBound after migration:

1. SERVICE_UPDATED notification:

   This notification is sent to indicate if the update was successful.

2. VM_SET_MONITOR_STATUS notification:

   This notification is sent to indicate the status of setting monitor on the new monitoring agent for each VM in the VNF.

3. SVC_SET_MONITOR_STATUS notification

   This notification is sent to indicate the service level status of setting monitor for a deployment.

Monitoring agent migration is considered successful when NorthBound receives a successful SERVICE_UPDATED and SVC_SET_MONITOR_STATUS notification.

The following example shows a VM_SET_MONITOR_STATUS notification:

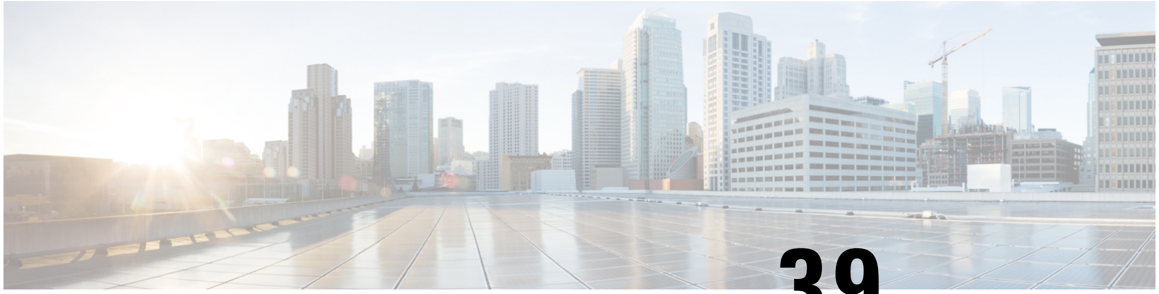
```
<?xml version="1.0" encoding="UTF-8"?>
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2020-08-06T14:04:47.124+00:00</eventTime>
  <escEvent xmlns="http://www.cisco.com/esc/esc">
    <status>SUCCESS</status>
    <status_code>200</status_code>
    <status_message>VM monitor setting completed successfully.</status_message>
    <depname>test-dep</depname>
    <tenant_id>563fba7044c847a6a370cc10d5ef7d57</tenant_id>
    <depid>995f6849-0599-4287-bc3b-fca6de7bfcb2</depid>
    <vm_group>g1</vm_group>
    <vm_source>
      <vmid>ca40ccb1-fe21-4846-a15f-79900e7e3baa</vmid>
      <vmname>test-dep_g1_0_88e9b2af-aef2-472c-84c1-1dbbf96df31f</vmname>
     <generated_vmname>test-dep_g1_0_88e9b2af-aef2-472c-84c1-1dbbf96df31f</generated_vmname>

      <hostid>16e897fa14b3d1ecee0f7489a7a9ac7902f66c1f017437f27474a4c5</hostid>
      <hostname>my-ucs-3</hostname>
      <interfaces>
        <interface>
          <nicid>0</nicid>
          <type>virtual</type>

<vim_interface_name>test-dep_g1_0_88e9b2af-aef2-472c-84c1-1dbbf96df31f</vim_interface_name>
```

```
          <port_id>f8cc9d5b-6bb0-4050-98bd-8aa25d71a68c</port_id>
          <network>3d8a4b3d-6ced-4733-8143-6cea6da85411</network>
          <subnet>e0f2da9e-0c8d-4351-847a-1bf36cc3ffdc</subnet>
          <ip_address>172.29.0.9</ip_address>
          <mac_address>fa:16:3e:f6:3b:b7</mac_address>
          <netmask>255.255.240.0</netmask>
          <gateway>172.29.0.1</gateway>
        </interface>
      </interfaces>
      <properties>
        <property>
          <name>monitoring_agent</name>
          <value>dmonaName://test-dmona-dep-1</value>
        </property>
      </properties>
    </vm_source>
    <event>
      <type>VM_SET_MONITOR_STATUS</type>
    </event>
  </escEvent>
</notification>
```

The following example shows a SVC_SET_MONITOR_STATUS notification:

```
<?xml version="1.0" encoding="UTF-8"?>
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2020-08-06T14:04:47.132+00:00</eventTime>
  <escEvent xmlns="http://www.cisco.com/esc/esc">
    <status>SUCCESS</status>
    <status_code>200</status_code>
    <status_message>Service monitor setting completed successfully.</status_message>
    <depname>test-dep</depname>
    <tenant>admin</tenant>
    <tenant_id>563fba7044c847a6a370cc10d5ef7d57</tenant_id>
    <depid>995f6849-0599-4287-bc3b-fca6de7bfcb2</depid>
    <monitoring>
      <vm_group>
        <name>g1</name>
        <monitoring_agent>dmonaName://test-dmona-dep-1</monitoring_agent>
        <status_message>VM group setting monitor completed successfully.</status_message>
      </vm_group>
    </monitoring>
    <event>
      <type>SVC_SET_MONITOR_STATUS</type>
    </event>
  </escEvent>
</notification>
```

For information on monitoring agent migration in VNFs using ETSI API, see the Migrating the Monitoring Agent chapter in the Cisco ElasticServices Controller ETSI NFV MANO User Guide.



CHAPTER 39

Scaling Virtual Network Functions

Scaling Overview

ESC is capable of elastically scaling the service. It can be configured to do both scale in and scale out automatically. The scaling is achieved using KPI, rules and actions. These are configured during deployment. The KPI define the event name and threshold. The rules define action to trigger scale out and scale in.

For information on KPIs, Rules and Metrics, see KPIs, Rules and Metrics, on page 169.

Scale In and Scale Out of VMs

Scaling workflow begins after successful deployment of a VNF. VMs are configured to monitor attributes such as CPU load, memory usage, and so on, which form the KPI data in the data model. If for any attributes, KPI reaches its threshold, based on the action defined, scale in and scale out is performed.

- During scale out, if the number of VMs is less than maximum active, a new VM deployment is triggered.
- During scale in, if the number of VMs is greater than the minimum active, the VM will be undeployed.

Note If the VM is deployed and did not receive the VM alive event, then recovery will be triggered. Any error during undeployment will be notified to the northbound user.

In the scaling section of the datamodel, the minimum and maximum values are configured. The min_active defines the number of VMs deployed. The max_active defines the number of maximum VMs that can be deployed. For example, if a VNF is deployed with a minimum 2 VMs and a maximum of 100 VMs, the below xml will define scaling under each VM group.

If the active VM was configured using a static IP address, the scaled out VMs must be assigned a static IP address. During deployment, a list of static IP addresses must be specified. The following example explains how to create a static IP pool:

```
 <scaling>
   <min_active>1</min_active>
   <max_active>2</max_active>
   <elastic>true</elastic>
   <static_ip_address_pool>
      <network>1234-5678-9123</network>
      <gateway>10.86.22.1</gateway>
      <netmask>255.255.255.0</netmask>
      <ip_address>10.86.22.227</ip_address>
      <ip_address>10.86.22.228</ip_address>
   </static_ip_address_pool>
</scaling>
```

The following example explains the method of detecting the CPU load in the KPI data section.

```
<?xml version="1.0" encoding="UTF-8"?>
<kpi>
  <event_name>VM_OVERLOADED</event_name>
  <metric_value>70</metric_value>
  <metric_cond>GT</metric_cond>
  <metric_type>UINT32</metric_type>
  <metric_occurrences_true>2</metric_occurrences_true>
  <metric_occurrences_false>4</metric_occurrences_false>
  <metric_collector>
    <type>CPU_LOAD_1</type>
    <nicid>0</nicid>
    <poll_frequency>3</poll_frequency>
    <polling_unit>seconds</polling_unit>
    <continuous_alarm>false</continuous_alarm>
  </metric_collector>
</kpi>
<kpi>
   <event_name>VM_UNDERLOADED</event_name>
   <metric_value>40</metric_value>
   <metric_cond>LT</metric_cond>
   <metric_type>UINT32</metric_type>
   <metric_occurrences_true>2</metric_occurrences_true>
   <metric_occurrences_false>4</metric_occurrences_false>
   <metric_collector>
      <type>CPU_LOAD_1</type>
      <nicid>0</nicid>
      <poll_frequency>3</poll_frequency>
      <polling_unit>seconds</polling_unit>
      <continuous_alarm>false</continuous_alarm>
   </metric_collector>
</kpi>
```

KPI rules are as follows:

```
<rule>
   <event_name>VM_OVERLOADED</event_name>
   <action>ALWAYS log</action>
   <action>TRUE servicescaleup.sh</action>
</rule>
<rule>
   <event_name>VM_UNDERLOADED</event_name>
   <action>ALWAYS log</action>
   <action>TRUE servicescaledown.sh</action>
</rule>
```

For information on scaling VNFs using ETSI API, see the *Cisco Elastic Services Controller NFV MANO Guide*.

Consistent Ordering of Resources for Scaling

ESC enables specifying resources such as ip address, mac address or day 0 configuration variables in a consistent manner in the deployment data model.

During manual and autoscaling, ESC allocates and deallocates the static IP address pool in the deployment data model in a consistent manner.

For example:

```
<scaling>
  <min_active>3</min_active>
  <max_active>6</max_active>
  <static_ip_address_pool>
    <network>jenkins-internal-vnf-net-1</network>
    <ip_address>192.168.15.3</ip_address>
    <ip_address>192.168.15.111</ip_address>
    <ip_address>192.168.15.22</ip_address>
    <ip_address>192.168.15.5</ip_address>
    <ip_address>192.168.15.4</ip_address>
    <ip_address>192.168.15.222</ip_address>
  </static_ip_address_pool>
</scaling>
```

- Manual Scaling—ESC allocates the IP addresses in the order available in the static IP pool during scale-out. During scale in, the IP addresses are released in the last in first out order.
- Autoscaling—Autoscaling uses SNMP events to indicate overload and underload of the VNFs. The overload event causes ESC to scale out, and allocates the first free IP address in the static IP pool from the order listed in the deployment data model. During scale-in, ESC deallocates the IP address, and the IP address is free for future scaling events.

For more information on day 0 configuration, ip address in the deployment data model, see Deployment Parameters, on page 157.

Scaling Notifications and Events

The scaling notifications are sent to the northbound users. The notification includes status message and other details to identify the service that is undergoing scaling. Below is the list of notifications:

```
VM_SCALE_OUT_INIT
VM_SCALE_OUT_DEPLOYED
VM_SCALE_OUT_COMPLETE
VM_SCALE_IN_INIT
VM_SCALE_IN_COMPLETE
```

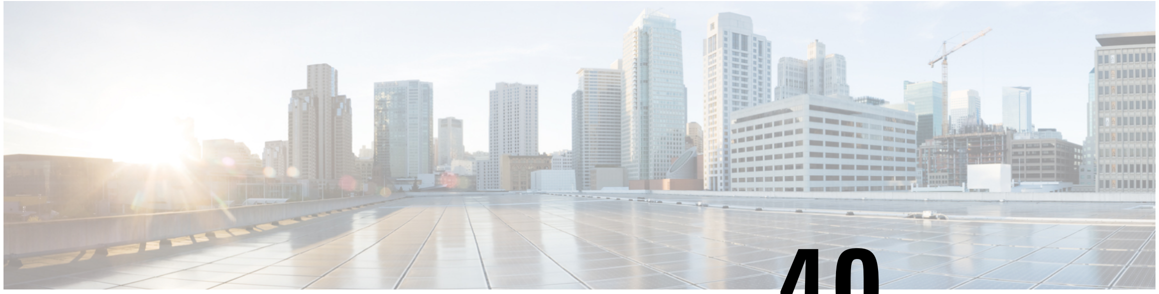
The following table lists the scaling scenarios and the notifications that are generated:

Scenarios	Notifications

Scale Out	ESC deploys VMs and sets KPI\Monitors and all VM Alives received. The following NETCONF notification is triggered.
	`<type>SERVICE_ALIVE</type>`
	`<status>SUCCESS</status>`
	When ESC receives a VM_OVERLOADED event, the following NetConf notification is triggered:
	`<type> VM_SCALE_OUT_INIT</type>`
	`<status>SUCCESS</status>`
	ESC checks if the max limit is reached, if not, it deploys a new VM.
	`<type> VM_SCALE_OUT_DEPLOYED</type>`
	`<status>SUCCESS</status>`
	Once the deployment is complete, the following Netconf Notification is sent,
	`<type>VM_SCALE_OUT_COMPLETE</type>`
	` <status>SUCCESS</status>`

Scale In	ESC deploys VMs and sets KPI\Monitors and all VM Alives received.
	Netconf Notification Sent
	`<type>SERVICE_ALIVE</type>`
	`<status>SUCCESS</status>`
	When ESC receives a VM_UNDERLOADED event, the following NetConf notification is triggered
	`<type> VM_SCALE_IN_INIT</type>`
	`<status>SUCCESS</status>`
	ESC check if number of VM is more than minimum active limit, if so, it undeploys one of the VM after undeployment is complete, Netconf Notification Sent.
	`<type>VM_SCALE_IN_COMPLETE</type>`
	`<status>SUCCESS</status>`

For all the error scenarios, the notification will be sent with FAILURE status. Also status message should have the corresponding failure details.



CHAPTER 40

Healing Virtual Network Functions

Healing Overview

As part of life cycle management, ESC heals the VNFs when there is a failure. The healing parameters are configured in the KPI section of the data model. The recovery policy specified during deployment controls the recovery. ESC supports recovery using the policy-driven framework.

ESC uses KPIs to monitor the VMs and events are triggered based on the KPI conditions. The actions to be taken for every event that is triggered are configured in the rules section during the deployment.

Healing a VM

Each VM group is configured to enable the healing. Healing is performed at two stages: Before the service is alive and after the service is alive with a recovery policy defined in the data model (at the VNF-level, and optionally overridden at the VM level).

The VMs are deployed and are being monitored. After ESC receives a VM Alive event, if it receives a VM Down event, the healing workflow attempts to recover the VM with the configured recovery policy.

If ESC doesn’t receive a VM Alive after deployment, ESC recovers the VM with the recovery policy when a timeout happens. All the recovery procedures depend on the recovery policy configuration and the policy names described in the following paragraph. Some additional policies can be used to override the behavior when manually recovering the VM.

ESC provides a YANG-based data model with comprehensive details of all the parameters and description that is needed to define the healing. ESC uses two sections in the data model XML file which define the events and rules:

- The <kpi> section defines the type of monitoring, events, polling interval, and other parameters.
- The <rule> section defines the actions when the KPI monitoring events are triggered.

For more information on KPI, rules, and data model, see KPIs, Rules and Metrics, on page 169.

The configuration involves the following steps:

1. Define kpi.
2. Define rules.

The following example shows how to configure the KPI in the data model:

```
<kpi>
<event_name>VM_ALIVE</event_name>
<metric_value>1</metric_value>
<metric_cond>GT</metric_cond>
<metric_type>UINT32</metric_type>
<metric_collector>
<type>ICMPPing</type>
<nicid>0</nicid>
<poll_frequency>3</poll_frequency>
<polling_unit>seconds</polling_unit>
<continuous_alarm>false</continuous_alarm>
</metric_collector>
</kpi>
```

The following example shows how to configure the rules for every event:

```
<rules>
<admin_rules>
<rule>
<event_name>VM_ALIVE</event_name>
<action>ALWAYS log</action>
<action>FALSE recover autohealing</action>
<action>TRUE servicebooted.sh</action>
</rule>
</admin_rules>
</rules>
```

In the previous examples, we define a KPI to monitor the ICMP Ping on the nicid 0. It defines the attributes of metric conditions and polling. Based on the KPI, the VM_ALIVE event is triggered with appropriate values. The action in the corresponding rule defines what the next steps are:

- FALSE—Triggers recovery of the VM.
- TRUE—Triggers the defined action.

If recovery is triggered on the VM with the reboot, then redeploy option configured in the recovery policy, ESC reboots the VM as the first step to recover the VM. If it fails, the VM is undeployed and a new VM with the same day-0 configuration is deployed. ESC tries to reuse the same network configuration as MAC and IP Address as the previous VM.

Typically, if the VM is unreachable, ESC starts VM recovery on all unreachable VMs. During a network outage, ESC suspends VM recovery during the network outage, thus delaying the VM recovery. ESC detects the unreachable VM and evaluates the reachability of the gateway first to detect the presence of a network failure.

If ESC cannot ping the gateway, no action is performed to recover the VM. VM recovery resumes when the gateway becomes reachable.

If there is a double fault condition, that is, when the network gateway and the VM failure occur at the same time, the ESC automatically performs VM monitoring after the gateway is reachable again.

For information on healing a VNF using the ETSI API, see the *Cisco Elastic Services Controller NFV MANO Guide*.

Recovery Policy

ESC has the following VM recovery types that you can specify when you deploy a VNF:

1. - **Auto Recovery**
   - **Manual Recovery**

ESC supports recovery using the policy-driven framework, see Recovery Policy (Using the Policy Framework) for details.

There are three types of actions for a VM recovery that can be specified in the deployment data model:

- **REBOOT_THEN_REDEPLOY (default)**—When a VM down event is received or the timer expires, the healing workflow first attempts to reboot the VM, if it fails to reboot, then it attempts to redeploy the VM on the same host.
- **REBOOT_ONLY**—When a VM down event is received or the timer expires, the healing workflow only attempts to reboot the VM.
- **REDEPLOY_ONLY**—When a VM down event is received or the timer expires, the healing workflow only attempts to redeploy the VM.
- **MIGRATE_ONLY**—When a VM down event is received or the timer expires, the healing workflow only attempts to migrate the VM..
- **REBOOT_THEN_MIGRATE**—When a VM down event is received or the timer expires, the healing workflow first attempts to reboot the VM, if it fails to reboot, then it attempts to migrate the VM.

  VM Migrate is supported from 5.7 version onwards. Please refer to the VM Operations section to know more about VM Migrate operation.

Note If the policy involves REBOOT_THEN_REDEPLOY and REDEPLOY_ONLY for redeploying the VMs, and if the placement policy is not enforced, then the VIM decides which host to redeploy the VM on.

Note ESC supports both manual and auto-recovery for vCloud Director. All three types of recovery actions are applicable for the vCloud Director. The REBOOT_THEN_REDEPLOY is the default recovery action. For vCD deployment, see Deploying Virtual Network Functions on VMware vCloud Director (vCD), on page 140.

Any recovery action that involves redeploying the VM will automatically recreate and attach ephemeral ports and volumes managed by ESC, that are faulty or deleted to ensure the recovery is successful.

Auto Recovery

In Auto recovery, the recovery type parameter is set to Auto. ESC automatically recovers the VM with the specified <action-on-recovery> value in the recovery policy. The recovery type is auto by default if the user does not choose a recovery type.

```
<recovery_policy>
 <recovery_type>AUTO</recovery_type>
 <action_on_recovery>REBOOT_THEN_REDEPLOY</action_on_recovery>
 <max_retries>3</max_retries>
</recovery_policy>
```

Manual Recovery

Manual Recovery of a VM

In manual recovery, ESC sends the VM_MANUAL_RECOVERY_NEEDED notification to northbound (NB) and waits for the instruction from NB for recovery. ESC performs recovery when it receives recovery instruction from NB. For manual recovery of a complete deployment, see Manual Recovery of a Deployment, on page 335

ESC also supports overriding of the actions on a single request basis, using the action-on-recovery parameter in the recovery policy. In addition to the 3 recovery actions listed before, there are 2 more recovery actions available:

1.
    - **RESET_STATE_THEN_REBOOT** – before rebooting the VM, the VM state is reset to allow the VIM to reboot the VM for recovery. This is only applicable to Openstack.
    - **DISASTER_RECOVERY** – when the VIM to which the VNF has been deployed, become unavailable and there is a need to move the VNF to a new VIM for service continuity, this action can be invoked to redeploy the VNF (entire service, not individual VMs) on to a new VIM.

      To use this action, it must be preceded by a model-only service update to update the VIM locator; failure to carry out this step will result in the recovery request failing. See below for details on how to perform this type of service update (via the REST API only).

      The original VNF is not attempted to be removed. Since it is assumed that use of this recovery action implies that the VNF is unreachable from the orchestration stack and when the VIM itself has been recovered, the old deployment **must** be manually cleaned.

The manual recovery policy data model is as follows

```
<vm_group>
        ...
        <recovery_policy>
         <recovery_type>MANUAL</recovery_type>
         <action_on_recovery>REBOOT_THEN_REDEPLOY</action_on_recovery>
         <max_retries>3</max_retries>
        </recovery_policy>
        ...
  </vm_group
```

For more information about recovery policy parameters in the data model, see Elastic Services Controller Deployment Attributes. For more information about configuring the recovery policy in the ESC Portal (VMware only), see the Deploying VNFs on VMware vCenter using ESC Portal.

The VM_MANUAL_RECOVERY_NEEDED notification is as follows:

```
===== SEND NOTIFICATION STARTS =====
 WARN  Type: VM_MANUAL_RECOVERY_NEEDED
 WARN  Status: SUCCESS
 WARN  Status Code: 200
 WARN  Status Msg: Recovery event for VM
[manual-recover_error-g1_0_7d96ad0b-4f27-4a5a-bdf7-ec830e93d07e] triggered.
 WARN  Tenant: manual-recovery-tenant
 WARN  Service ID: NULL
 WARN  Deployment ID: 08491863-846a-4294-b305-c0002b9e8daf
 WARN  Deployment name: dep-error
 WARN  VM group name: error-g1
 WARN  VM Source:
 WARN      VM ID: ffea079d-0ea2-4d47-ba31-26a08e6dff22
 WARN      Host ID: 3a5351dc4bb7df0ee25e238a8ebbd6c6fcdf225aebcb9dff6ba10249
 WARN      Host Name: my-server-27
 WARN      [DEBUG-ONLY] VM IP: 192.168.0.3;
 WARN  =====  SEND NOTIFICATION ENDS  =====
```

APIs for Manual Recovery of a VM

You can perform the manual recovery using the Confd and Rest APIs. The manual recovery request can be configured to override the predefined recovery action to any desired action.

Netconf API `recovery-vm-action DO generated vm name [xmlfile]`

To perform recovery using the API, login to esc_nc_cli and run the following command:

```
$ esc_nc_cli --user <username> --password <password> recovery-vm-action DO [xmlfile]
```

The recovery is performed and the recovery notification is sent to NB.

Note Recovery (recovery-vm-action DO <VM-NAME>) can be performed after the VM is alive and the service is active. If the deployment is incomplete, it must be completed before performing recovery.

If a failover happens during a configurable manual recovery, the manual recovery resumes with predefined recover action.

The migration of any deployment must always use default recovery policy. You must not provide recovery action for VM/VNF manual recovery in an LCS based recovery.You must not use enable monitor and configurable manual recovery options together.

REST API

```
http://ip:8080/ESCAPI/#!/Recovery_VM_Operations/handleOperation
```

```
POST /v0/{internal_tenant_id}/deployments/recovery-vm/{vm_name}
```

Recovery VM operation payload:

```
{
"operation":"recovery_do",
"properties":{
"property":[
{
"name":"action",
"value":"REDEPLOY_ONLY"
}
]
}
}
```

In order to perform a model-only service update, a new parameter can be supplied to the edit-config API to prevent any action to be taken on the VIM and limit the update to the ESC data model only. This allows the preparation of the data model to complete until such time that the deployment is ready to be updated on the VIM:

`http://ip:8080/ESCManager/v0/conf/edit-config?modelOnly=true`

The VIM locator update, for example, prior to invoking the recovery API with `DISASTER_RECOVERY` as the <action-on-recovery>:

```
<?xml version="1.0" encoding="UTF-8"?>
<esc_datamodel xmlns="http://www.cisco.com/esc/esc"
xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:ns1="urn:ietf:params:xml:ns:netconf:notification:1.0">
   <tenants>
      <tenant>
         <name>admin-tenant</name>
         <deployments>
            <deployment>
               <name>test-deploy</name>
               <networks>
                  <network>
                     <name>test-network</name>
                     <locator>
                        <vim_id>my-ucs-59</vim_id>
                        <vim_project>admin</vim_project>
                     </locator>
                  </network>
               </networks>
               <vm_group>
                  <name>g1</name>
                  <locator>
                     <vim_id>my-ucs-59</vim_id>
                     <vim_project>admin</vim_project>
                  </locator>
                  <bootup_time>120</bootup_time>
               </vm_group>
            </deployment>
         </deployments>
      </tenant>
   </tenants>
</esc_datamodel>
```

Note Do not forget that the old deployment has to be deleted in the disaster recovery scenario, once the VIM is available again.

A further use for this API is to update the persistent volume UUID prior to rebooting the VM via the recovery API documented above. This has the benefit of negating the need to remove the VM group and re-add it, as per earlier versions of ESC. Here is an example payload:

```
<esc_datamodel xmlns="http://www.cisco.com/esc/esc"
xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:ns1="urn:ietf:params:xml:ns:netconf:notification:1.0">
 <tenants>
    <tenant>
       <name>my-tenant</name>
       <deployments>
          <deployment>
             <name>my-dep</name>
             <vm_group>
```

```
                <name>my-vm</name>
                <bootup_time>1800</bootup_time>
                <volumes>
                   <volume>
                      <name>new-volume</name>
                      <volid>1</volid>
                      <bus>ide</bus>
                      <type>lvm</type>
                   </volume>
                   <volume nc:operation="delete">
                      <name>old-volume</name>
                      <volid>1</volid>
                   </volume>
                </volumes>
             </vm_group>
          </deployment>
       </deployments>
    </tenant>
 </tenants>
</esc_datamodel>
```

Supported VM States and Service Combinations for Manual Recovery of a VM

The API, recovery-vm-action, applies to both auto and manual recovery types, but only under certain VM states and services. The following table shows the details. In general, during deployment, service update, undeployment, and recovery, the manual recovery action is rejected by ESC.

VM State	Service State	recovery-vm-action
ALIVE	ACTIVE	supported
ALIVE	ERROR	supported
ERROR	ERROR	supported

Manual Recovery of a Deployment

Recovery Without Monitoring Parameters

ESC supports manual recovery of VMs at the service level, that is, recovery of a complete deployment. After the successful deployment of a service, the service may move into an error state because of failed VMs. ESC can manually recover these failed VMs, or the complete deployment through a deployment recovery request. For manual recovery of a single VM, see Manual Recovery, on page 332.

APIs for Manual Recovery of a Deployment

You can perform the manual recovery using the NETCONF and REST APIs.

The manual recovery request can be configured to override the predefined recovery action to any desired action.

Note There is no service active notification after the deployment recovery. You must run a query, for example, *esc_nc_cli --user <username> --password <password> get esc_datamodel* to see if the service state of the deployment is active or not.

If a failover happens during a configurable manual recovery, the manual recovery resumes with predefined recover action.

The migration of any deployment must always use a default recovery policy. You must not provide recovery action for VM/VNF manual recovery in an LCS-based recovery.You must not use enable monitor and configurable manual recovery options together.

NETCONF API

```
svc-action RECOVER tenant-name deployment-name [xmlfile]
```

To perform recovery using the API, login to `esc_nc_cli`.

REST API

```
POST /v0/{internal_tenant_id}/deployments/service/{internal_deployment_id}
Content-Type: application/xml
Accept: application/json
Callback: http://172.16.0.1:9010/
Callback-ESC-Events: http://172.16.0.1:9010/
<service_operation xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <operation>recover</operation>
</service_operation>
```

where,

internal_tenant_id—is the system admin tenant ID or the tenant name.

internal_deployment_id—is the deployment name.

Supported VM States and Service Combinations for Manual Recovery of a Deployment

The API, svc-action RECOVER, applies to both auto and manual recovery types, but only under certain VM states and services. The following table shows the details. In general, during deployment, service update, undeployment, and recovery, the manual recovery action is rejected by ESC.

Note ESC accepts a VM level recovery request when the service is in an active or error state.

Notifications are not sent to NB if all VMs are in the ALIVE state after a service recovery request.

VM State	Service State	svc-action RECOVER
ERROR	ERROR	supported
ERROR	ERROR	supported

Recovery Enabled with Monitoring Parameters

During manual recovery, you can recover a VM depending on its monitoring parameters. If the VM is in error state, set the monitoring parameters to bring back the VM in error state to live state. If the VM is recovered, then ESC sends a RECOVERY_CANCELLED notification. If the VM does not come back live, then the recovery process is triggered. See Manual Recovery for more details.

NETCONF API

```
svc-action SET_MONITOR_AND_RECOVER <tenant-name> <dep-name>
```

Recovery notification:

```
===== SEND NOTIFICATION STARTS =====
 WARN  Type: VM_RECOVERY_INIT
 WARN  Status: SUCCESS
 WARN  Status Code: 200
 WARN  Status Msg: Recovery with enabling monitor first event for VM Generated ID
[dep-resource_g1_0_74132737-d0a4-4ef0-bd9e-86465c1017bf] triggered.
```

Note Recovery enabled with monitoring parameters is for manual recovery at service level only.

The *monitor_on_error* parameter enables continuous monitoring of the VMs in error state.

```
<recovery_policy>
                         <recovery_type>AUTO</recovery_type>
                         <action_on_recovery>REBOOT_ONLY</action_on_recovery>
                         <max_retries>1</max_retries>
                         <monitor_on_error>true</monitor_on_error>
                     </recovery_policy>
```

The default value is false.

if false, monitoring is **unset** on the vm in error state.

If true, monitoring is **set** on the vm in error state. If any VM Alive event occurs later (after VM_RECOVERY_COMPLETE), the VM is moved back to alive state.

VM Recovery Using Migrate Option

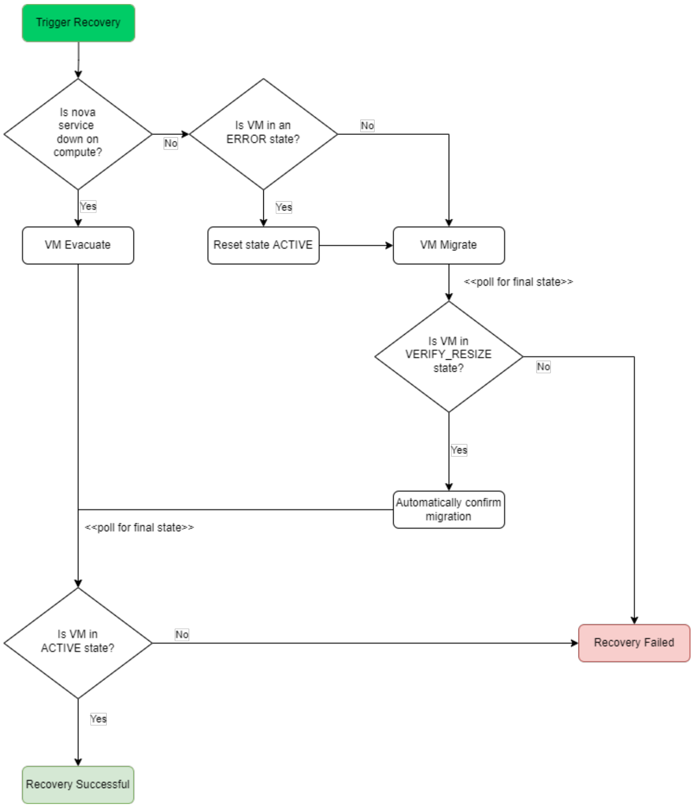
ESC Manager supports two new recovery policies that are REBOOT_THEN_MIGRATE and MIGRATE_ONLY. These recovery policies use OpenStack cold migration without host parameters.

You can perform the manual recovery with these new policies using the existing recovery Netconf or REST API.

Once the VM moves to the VERIFY_RESIZE state in the recovery flow, ESC auto confirms the migration.

ESC performs either of the two actions in the recovery process:

- If the *nova-compute* service is down on the host, then the VM evacutes and OpenStack rebuild the VM in another host.
- If the VM is in an error state, ESC resets the VM to ACTIVE state and migrates.



For the REBOOT_THEN_MIGRATE recovery policy, the VM reboots and if the reboot fails, then ESC triggers the VM migration.

For the MIGRATE_ONLY recovery policy, ESC triggers the VM migration.

Example Payload:

```
<?xml version="1.0" encoding="UTF-8"?>
<esc_datamodel xmlns="http://www.cisco.com/esc/esc">
    <tenants>
        <tenant>
            <name>admin</name>
            <deployments>
                <deployment>
                    <name>Test-Cirros-VNF</name>
                    <vm_group>
                        <name>Test-vm-group</name>
                        <bootup_time>60</bootup_time>
                        <recovery_wait_time>0</recovery_wait_time>
                        <image>Automation-Cirros-Image</image>
                        <flavor>Automation-Cirros-Flavor</flavor>
                        <interfaces>
                            <interface>
                                <nicid>0</nicid>
                                <network>esc-net</network>
                            </interface>
                        </interfaces>
                        <scaling>
                            <min_active>1</min_active>
                            <max_active>1</max_active>
                            <elastic>true</elastic>
                        </scaling>
```

```
                        <kpi_data>
                            <kpi>
                                <event_name>VM_ALIVE</event_name>
                                <metric_value>1</metric_value>
                                <metric_cond>GT</metric_cond>
                                <metric_type>UINT32</metric_type>
                                <metric_collector>
                                    <type>ICMPPing</type>
                                    <nicid>0</nicid>
                                    <poll_frequency>3</poll_frequency>
                                    <polling_unit>seconds</polling_unit>
                                    <continuous_alarm>false</continuous_alarm>
                                </metric_collector>
                            </kpi>
                        </kpi_data>
                        <rules>
                            <admin_rules>
                                <rule>
                                    <event_name>VM_ALIVE</event_name>
                                    <action>"ALWAYS log"</action>
                                    <action>"TRUE servicebooted.sh"</action>
                                    <action>"FALSE recover autohealing"</action>
                                </rule>
                            </admin_rules>
                        </rules>
                        <config_data/>
                        <recovery_policy>
                            <recovery_type>AUTO</recovery_type>
                            <action_on_recovery>REBOOT_THEN_MIGRATE</action_on_recovery>
                            <max_retries>3</max_retries>
                            <monitor_on_error>false</monitor_on_error>
                        </recovery_policy>
                    </vm_group>
                </deployment>
            </deployments>
        </tenant>
    </tenants>
</esc_datamodel>
```

Notifications:

ESC sends the following notifications once the recovery is successful.

```
VM_RECOVERY_INIT
VM_RECOVERY_MIGRATED
VM_RECOVERY_COMPLETE
```

Recovery and Redeployment Policies

ESC uses a policy driven framework to perform actions based on the lifecycle stages in a deployment. A deployment consists of several stages through its lifecycle. Each lifecycle stage (LCS) is associated with a condition. The condition in turn is associated with a predefined action or custom scripts. These conditions and actions are specified within the policies tag in the data model. For more information on Policy driven Framework, see Policy-Driven Data model, on page 185.

The recovery and redeployment workflows in ESC are policy driven. When VNFs are deployed, the recovery and redeployment policies are specified in the deployment data model. These policies are based on the lifecycle stages of VM or VNF and have actions associated with it.

When a deployment data model is created, you can specify the following policies:

- **Recovery Policy**—The recovery policy is for the VM lifecycle, that is for the recovery of a single VM. Based on the predefined actions, the VM is rebooted, or redeployed. You can perform recovery without using the policy framework. See Recovery Policy, on page 331.
- **Redeployment Policy**—The redeployment policy is for the entire deployment lifecycle, that is for all the VM groups within a deployment. Based on a set of predefined actions, the host is disabled, and VMs are recovered in the deployment.

If the VM recovery fails after the maximum attempts, ESC disables the host and triggers redeployment for all VMs within the deployment. All VMs are undeployed from the old host and redeployed to a new host.

ESC supports redeploying the failed VMs first. During a redeployment, the failed VMs are recovered first, and the VMs that have not failed are queued up for redeployment.

Recovery Policy (Using the Policy Framework)

ESC supports recovery of VMs using the policy-driven framework data model. The recovery is based on the lifecycle stages of VM deployment and predefined actions.

For auto and manual recovery, see Recovery Policy, on page 331.

The table below describes the predefined actions performed at different lifecycle stages.

Predefined Action Name	Scope	Description
SET_RECOVERY::REBOOT_ONLY	Deployment	Sets the recovery action for all VM groups (in a deployment), or for a VM (in a VM group) to REBOOT_ONLY.
SET_RECOVERY::REBOOT_THEN_REDEPLOY	Deployment	Sets the recovery action for all VM groups (in a deployment), or for a VM (in a VM group) to REBOOT_THEN_REDEPLOY.
SET_RECOVERY::REDEPLOY_ONLY	Deployment	Sets the recovery action for all VM groups (in a deployment), or for a VM (in a VM group) to REDEPLOY_ONLY.
SET_RECOVERY::RESET_STATE_THEN_REBOOT	Deployment	Sets the recovery action for all VM groups (in a deployment), or for a VM (in a VM group) to RESET_STATE_THEN_REBOOT (Openstack only).
SET_RECOVERY::MIGRATE_ONLY	Deployment	Sets the recovery action for all VM groups (in a deployment), or for a VM (in a VM group) to MIGRATE_ONLY.
SET_RECOVERY::REBOOT_THEN_MIGRATE	Deployment	Sets the recovery action for all VM groups (in a deployment), or for a VM (in a VM group) to REBOOT_THEN_MIGRATE.

Supported Conditions and Predefined Action Combinations

The following table describes the supported LCS conditions and its actions for recovery and redeployment policies using the policy framework. For more details on the policy driven framework, see Recovery and Redeployment Policies, on page 339.

Condition	Predefined action	Description
LCS::PRE_DEPLOY —Occurs just before deploying VMs in a deployment.	• SET_RECOVERY::REBOOT_ONLY —Sets the recovery action for all VM groups (in a deployment), or for a VM (in a VM group) to REBOOT_ONLY. • SET_RECOVERY:: REBOOT_THEN_REDEPLOY —Sets the recovery action for all VM groups (in a deployment), or for a VM (in a VM group) to REBOOT_THEN_REDEPLOY. • SET_RECOVERY:: REDEPLOY_ONLY —Sets the recovery action for all VM groups (in a deployment), or for a VM (in a VM group) to REDEPLOY_ONLY. • SET_RECOVERY_ REDEPLOY:: SERIALIZED —Queues up the recoveries in the deployment. That is, new recovery does not start until the current ongoing recovery completes.	Choose any one of the predefined action for recovery. Choose SET_RECOVERY_ REDEPLOY::SERIALIZED, if DROP_RECOVERIES action is used. This means the VMs need to be kept on the original host once the redeployment fails. If not chosen, then DROP_RECOVERIES action cannot be used.
LCS:: POST_DEPLOY_ALIVE —Occurs immediately after the deployment is active.		
LCS::DEPLOY_ERR —Occurs immediately after the deployment fails.	DISABLE_HOST —Disables the host(s) the deployment or the VM is using.	-
LCS::POST_ DEPLOY::VM_RECOVERY _ERR —Occurs immediately after the recovery of one VM fails.	DISABLE_HOST —Disables the host(s) the deployment or the VM is using. REDEPLOY_ALL:: DISABLE_HOST —Disables the host the VM is using then trigger redeploy for all VMs (within a deployment), or all VMs on that host.	Choose DISABLE_HOST if needed. REDEPLOY_ALL:: DISABLE_HOST Choose if redeploy is needed after disabling the host. DISABLE_HOST and REDEPLOY_ALL::DISABLE_HOST cannot be together as they overlap.

Condition	Predefined action	Description
LCS::POST_DEPLOY::VM_RECOVERY_REDEPLOY_ERR —Occurs immediately after the redeploy of one VM fails.	• DISABLE_HOST —Disables the host(s) the deployment or the VM is using. • DROP_RECOVERIES —Drops all pending recoveries in the deployment.	DISABLE_HOST Choose if DISABLE_HOST is needed. Choose DROP_RECOVERIES if VMs need to be kept on the original host once the redeploy fails. When choosing DROP_RECOVERIES, ensure SET_RECOVERY_REDEPLOY:: SERIALIZED action is complete.

Redeployment Policy

Redeployment policies are a part of the policy driven framework. Using this framework, you can specify predefined actions for specific lifecycle conditions. For more information on ESC policy driven framework, see Policy-Driven Data model, on page 185.

Redeployment policies are invoked when a VM recovery fails after the maximum number of attempts. ESC disables the host and triggers redeployment for all VMs within the deployment. All VMs are undeployed from the old host and redeployed to a new host. Based on the combination of lifecycle stages (LCS) and predefined actions, the VMs are redeployed. The redeployment policy is for the entire deployment.

You can use the following lifecycle condition and action combination in the policy datamodel.

Note ESC uses default recovery action, **REBOOT_THEN_REDEPLOY** if nothing is chosen.

A sample redeployment policy data model is as follows:

```
<tenants>
        <tenant>
            <name>xyz-redeploy-ten-0502</name>
            <deployments>
                <deployment>
                    <name>dep</name>
                    <policies>
                        <policy>
                            <name>1</name>
                            <conditions>
                                <condition>
                                    <name>LCS::PRE_DEPLOY</name>
                                </condition>
                            </conditions>
                            <actions>
                                <action>
                                    <name>SET_RECOVERY::REBOOT_THEN_REDEPLOY</name>
                                    <type>pre-defined</type>
                                </action>
```

```
        <action>
            <name>SET_RECOVERY_REDEPLOY::SERIALIZED</name>
            <type>pre-defined</type>
        </action>
    </actions>
</policy>
<policy>
    <name>2</name>
    <conditions>
        <condition>
            <name>LCS::POST_DEPLOY_ALIVE</name>
        </condition>
    </conditions>
    <actions>
        <action>
            <name>SET_RECOVERY::REBOOT_ONLY</name>
            <type>pre-defined</type>
        </action>
    </actions>
</policy>
<policy>
    <name>3</name>
    <conditions>
        <condition>
            <name>LCS::DEPLOY_ERR</name>
        </condition>
    </conditions>
    <actions>
        <action>
            <name>DISABLE_HOST</name>
            <type>pre-defined</type>
        </action>
    </actions>
</policy>
<policy>
    <name>4</name>
    <conditions>
        <condition>
            <name>LCS::POST_DEPLOY::VM_RECOVERY_ERR</name>
        </condition>
    </conditions>
    <actions>
        <action>
            <name>REDEPLOY_ALL::DISABLE_HOST</name>
            <type>pre-defined</type>
        </action>
    </actions>
</policy>
<policy>
    <name>5</name>
    <conditions>
        <condition>
           <name>LCS::POST_DEPLOY::VM_RECOVERY_REDEPLOY_ERR</name>

        </condition>
    </conditions>
    <actions>
        <action>
            <name>DISABLE_HOST</name>
            <type>pre-defined</type>
        </action>
        <action>
            <name>DROP_RECOVERIES</name>
            <type>pre-defined</type>
```

```
                            </action>
                        </actions>
                    </policy>
                </policies>
                <vm_group>
                    <name>Group1</name>
                    <image>xyz-redeploy-img-0502</image>
                    <flavor>xyz-redeploy-flv-0502</flavor>
                    <recovery_policy>
                        <max_retries>1</max_retries>
                    </recovery_policy>
           ......
           ......
         </deployment>
        </deployments>
    </tenant>
</tenants>
```

Supported Lifecycle Stages (LCS)

Condition Name	Scope	Description
LCS::PRE_DEPLOY	Deployment	Occurs just before deploying VMs of the deployment.
LCS::POST_DEPLOY_ALIVE	Deployment	Occurs immediately after the deployment is active.
LCS::DEPLOY_ERR	Deployment	Occurs immediately after the deployment fails.
LCS::POST_DEPLOY:: VM_RECOVERY_ERR	Deployment	Occurs immediately after the recovery of one VM fails. (This is specified at deployment level and applies to all VM groups)
LCS::POST_DEPLOY:: VM_RECOVERY_REDEPLOY_ERR	Deployment	Occurs immediately after the redeployment of one VM fails. (This is specified at deployment level and applies to all VM groups)

Supported Predefined actions

Predefined Action Name	Scope	Description
DISABLE_HOST	Deployment	Disables the host(s) the deployment or the VM is using.
REDEPLOY_ALL::DISABLE_HOST	Deployment	Disables the host the VM is using then trigger redeploy for all VMs (within a deployment), or all VMs on that host.
DROP_RECOVERIES	Deployment	Drops all pending recoveries in the deployment.
SET_RECOVERY_REDEPLOY::SERIALIZED	Deployment	Queues up the recoveries in the deployment. That is, new recovery does not start until the current ongoing recovery completes.

Limiting the Number of Redeployments

Cisco Elastic Services Controller (ESC) limits the number of redeployments using the following parameters:

- **max_redep**: limits the maximum number of redeployments. By default, the max_redep value is -1, which indicates that there is no limit on the maximum number of redeployments. You can change this value using the bootvm.py arguments or REST API.
- **redep_count**: consists of the current number of redeployments. The redep_count automatically increases by 1 after a redeployment, irrespective of the success or failure of the redeployment.

Note The redeployment limit is for,

- redeployments triggered by REDEPLOY_ALL::DISABLE_HOST policy.
- deployments with single VIM configuration only.

Cisco Elastic Services Controller (ESC) performs redeployment,

- if the maximum number of redeployments is set to the default value of -1, that is max_redep = -1.
- if the current number of redeployments is less than the maximum number of redeployments (redep_count < max_redep), then ESC performs redeployment, and increases the redeployment count by 1 after the redeployment is complete.

ESC does not perform any redeployment if the redeployment count is more than or equal to the maximum number of redeployments, (redep_count >= max_redep).

You can use the bootvm.py parameters and REST APIs to configure the values.

Using the bootvm.py parameters

Specify the max_redep value in the esc_params.conf file that contains the following line: default.max_redep = 3

Run the command, `bootvm.py ... --esc_params_file <path_to_file>/esc_params.conf ...`

Using the REST APIs

You can retrieve, and reset the redep_count parameter using the following APIs:

- To retrieve the current value of redep_count:

  ```
  GET http://<ESC IP>:8080/ESCManager/v0/systemstate/redep_count
  ```
- To reset redep_count:

  ```
  POST http://<ESC IP>:8080/ESCManager/v0/systemstate/redep_count/reset
  ```

You can also use the REST API to retrieve and change the max_redep value.

- To retrieve the current value of max_redep:

  ```
  GET http://<ESC IP>:8080/ESCManager/v0/config/default/max_redep
  ```
- To change the max_redep value:

  ```
  PUT http://<ESC IP>:8080/ESCManager/v0/config/default/max_redep/<value>
  ```

  where <value> can be,

-1, which is the default value with no limit

0, which does not allow any redeployment

more than zero (> 0), which specifies the maximum number of redeployments allowed.

You can also use the escadm tool to configure these values. For more information on the escadm tool, see the Elastics Services Controller Install and Upgrade Guide.

For more details on the redeployment policy, see Redeployment Policy, on page 342.

The VMs that are not redeployed because of the redeployment limit are moved to error state. ESC manually recovers these VMs in error state by enabling the monitoring operation on each VM.

To enable monitoring operation on a single VM in error state:

```
POST http://<ESC IP>:8080/ESCManager/v0/<internal-tenant-id>/deployments/vm/<vm-name> {
"operation" : "enable_monitoring" }
```

You can also enable monitoring using the esc_nc_cli command:

```
esc_nc_cli --user <username> --password <password> vm-action ENABLE_MONITOR <generated vm
name>
```

As part of the manual recovery process, the enable monitoring operation moves the VMs from error state to alive state. If manual recovery fails for these VMs, then auto recovery is triggered.

To enable the monitoring operation on VMs (in error state) in a deployment:

```
POST http://<ESC
IP>:8080/ESCManager/v0/<internal-tenant-id>/deployments/service/<internal-deployment-id> {
 "operation" : "enable_monitoring" }
```

You can also enable monitoring using the esc_nc_cli command:

```
esc_nc_cli --user <username> --password <password> svc-action ENABLE_MONITOR <tenant> <dep
name>
```

As part of the manual recovery process, the enable monitoring operation moves all the VMs in a deployment from error state to alive state. If manual recovery fails, then auto recovery is triggered on all the VMs in the deployment.

For more information, see Monitoring Operations, on page 308 and Recovery Policy.

Enabling and Disabling the Host

You can enable or disable the host on OpenStack using NETCONF and REST APIs. The host can also be disabled during a VNF recovery or redeployment scenario.

Note Enabling and disabling the host on VMware vCenter is not supported.

You cannot enable or disable a host on a non-default VIM using NETCONF and REST APIs in an ESC with multiple OpenStack VIMs.

Using NETCONF

```
/opt/cisco/esc/esc-confd/esc-cli/esc_nc_cli --user <username> --password <password>
host-action < ENABLE | DISABLE > <host-name>
```

The payload is as follows:

```
<hostAction xmlns="http://www.cisco.com/esc/esc">
  <actionType>ENABLE/DISABLE</actionType>
  <hostName>my-server</hostName>
</hostAction>
```

where,

- actionType is ENABLE or DISABLE
- hostName is the host name or UUID of the target host

Using REST

```
POST /v0/hosts/{hostName}/disable
POST /v0/hosts/{hostName}/enable
GET /v0/hosts/{hostName}/status
```

Enabling the Host

By enabling the host, you bring a disabled host back to OpenStack and deploy new VM instances on it.

Sample NETCONF notification is as follows:

```
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2016-03-30T15:04:05.95+00:00</eventTime>
  <escEvent xmlns="http://www.cisco.com/esc/esc">
    <status>SUCCESS</status>
    <status_code>200</status_code>
    <status_message>Host action successful</status_message>
    <vm_source>
      <hostname>my-server</hostname>
    </vm_source>
    <vm_target>
  </vm_target>
    <event>
      <type>HOST_ENABLE</type>
    </event>
  </escEvent>
</notification>
```

Sample REST notification is as follows:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
  <host_action_event xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <event_type>HOST_ENABLE</event_type>
    <host_name>my-server</host_name>
    <message>Host action successful</message>
</host_action_event>
```

Disabling a Host

During VNF redeployment, you disable the host, and trigger a host-based redeployment for all the VMs within that deployment. This ensures that the redeployed VMs are on a different host. You can also disable a host when it is not working properly. Once a host is disabled, it is removed from OpenStack, so that no new instances are deployed on it.

Sample NETCONF notification is as follows:

```
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2016-03-30T15:03:48.121+00:00</eventTime>
  <escEvent xmlns="http://www.cisco.com/esc/esc">
    <status>SUCCESS</status>
```

```
    <status_code>200</status_code>
    <status_message>Host action successful</status_message>
    <vm_source>
      <hostname>my-server</hostname>
    </vm_source>
    <vm_target>
  </vm_target>
    <event>
      <type>HOST_DISABLE</type>
    </event>
  </escEvent>
</notification>
```

Sample REST notification is as follows:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<host_action_event xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <event_type>HOST_DISABLE</event_type>
  <host_name>my-server</host_name>
  <message>Host action successful</message>
</host_action_event>
```

Notifications and Events

The following notifications are generated by the ESC during healing:

- VM_RECOVERY_INIT
- VM_RECOVERY_DEPLOYED
- VM_RECOVERY_UNDEPLOYED
- VM_RECOVERY_COMPLETE
- VM_RECOVERY_CANCELLED
- VM_RECOVERY_REBOOT
- VM_RECOVERY_MIGRATED

These notifications are generated based on the workflow. Each notification has details about the deployment for which the notification is triggered. All recovery starts with VM_RECOVERY_INIT and ends with VM_RECOVERY_COMPLETE.

During vm recovery, if the vm is back to normal within the recovery wait time, the VM_RECOVERY_CANCELLED notification is sent as there is no recovery action to be performed. If the recovery wait time expires, then the recovery action is triggered. After the recovery is complete, ESC sends the success or failure notification, for example, the VM_RECOVERY_REBOOT notification.

The following table lists the different scenarios and the notifications that are generated for every event:

Scenario	Notifications

ESC-NORTHBOUND Recovery Call Flow After VM Alive - Reboot	When Northbound places a deploy request to ESC, ESC deploys VMs and set KPI to monitor on all VM Alive received. The following NETCONF notification is triggered:
	`<type>SERVICE_ALIVE</type>`
	`<status>SUCCESS</status>`
	After ESC receives VM down event, the following NETCONF notification is triggered:
	`<type>VM_RECOVERY_INIT</type>`
	`<status>SUCCESS</status>`
	ESC performs hard reboot on the VM, and the VM alive event is received within the boot time.
	`<type>VM_RECOVERY_COMPLETE</type>`
	`<status>SUCCESS</status>`
	ESC receives an error while attempting to recover through Reboot. The following NETCONF notification is triggered:
	`<type>VM_RECOVERY_COMPLETE</type>`
	`<status>FAILURE</status>`

Scenario	Notifications

ESC-NORTHBOUND Recovery Call Flow After VM Alive - Undeploy/Redeploy	When Northbound places a deploy request to ESC, ESC deploys VMs and set KPI to monitor on all VM Alive received. The following NETCONF notification is triggered:
	`<type>SERVICE_ALIVE</type>`
	`<status>SUCCESS</status>`
	After ESC receives VM down event, the following NETCONF notification is triggered:
	`<type>VM_RECOVERY_INIT</type>`
	`<status>SUCCESS</status>`
	ESC fails to recover the VM by *Reboot* and proceeds with recovery by *Undeploy* and then *Redeploy*.
	It unsets monitoring and un-deploys the VM.
	The following NETCONF notification is triggered:
	`<type>VM_RECOVERY_UNDEPLOYED</type>`
	`<status>SUCCESS</status>`
	ESC deploys VM and sets KPI to monitor VM Alive event and triggers the following NETCONF notifications:
	`<type>VM_RECOVERY_DEPLOYED</type>`
	`<status>SUCCESS</status>`
	ESC receives a VM Alive event and triggers the following NETCONF notifications:
	`<type>VM_RECOVERY_COMPLETE</type>`
	`<status>SUCCESS</status>`

Scenario	Notifications

ESC-NORTHBOUND Recovery Call Flow Multiple Recovery Attempts	When Northbound places a deploy request to ESC, ESC deploys VMs and set KPI to monitor on all VM Alive received. The following NETCONF notification is triggered:
	`<type>SERVICE_ALIVE</type>`
	`<status>SUCCESS</status>`
	After ESC receives VM down event, the following NETCONF notification is triggered:
	`<type>VM_RECOVERY_INIT</type>`
	`<status>SUCCESS</status>`
	ESC fails to recover the VM by *Undeploy* and then *ReDeploy* until it receives a VM Alive event. It keeps attempting the recovery for a specified boot time until the maximum attempts of recovery is reached.
	It un-sets monitoring and un-deploys the VM.
	The following NETCONF notification is triggered:
	`<type>VM_RECOVERY_UNDEPLOYED</type>`
	`<status>SUCCESS</status>`
	ESC deploys VM and sets KPI to monitor VM Alive event.
	The following NETCONF notifications is triggered:
	`<type>VM_RECOVERY_DEPLOYED</type>`
	`<status>SUCCESS</status>`
	ESC receives a VM Alive event and triggers the following NETCONF notifications:
	`<type>VM_RECOVERY_COMPLETE</type>`
	`<status>SUCCESS</status>`

Scenario	Notifications

ESC-NORTHBOUND Recovery Call Flow Before VM Alive - Undeploy/Redeploy	When Northbound places a deploy request to ESC, ESC deploys VMs and sets KPI to monitor on all VM Alive received.
	ESC does not receive a VM Alive event after the deployment. Recovery is performed by *Undeploying* and *Redeploying* the VM.
	The following NETCONF notification is triggered:
	`<type>VM_RECOVERY_INIT</type>`
	`<status>SUCCESS</status>`
	ESC un-sets the monitoring and un-deploys the VM.
	The following NETCONF notification is triggered:
	`<type>VM_RECOVERY_UNDEPLOYED</type>`
	`<status>SUCCESS</status>`
	ESC deploys VM and sets KPI to monitor VM Alive event and triggers the following NETCONF notifications:
	`<type>VM_RECOVERY_DEPLOYED</type>`
	`<status>SUCCESS</status>`
	ESC receives a VM Alive event and triggers the following NETCONF notifications:
	`<type>VM_RECOVERY_COMPLETE</type>`
	`<status>SUCCESS</status>`

Scenario	Notifications

Error Path For ESC-NORTHBOUND Recovery Call Flow After VM Alive - Undeploy/ReDeploy	When Northbound places a deploy request to ESC, ESC deploys VMs and set KPI to monitor on all VM Alives received. The following NETCONF notification is triggered:
	`<type>SERVICE_ALIVE</type>`
	`<status>SUCCESS</status>`
	After ESC receives VM down event, the following NETCONF notification is triggered:
	`<type>VM_RECOVERY_INIT</type>`
	`<status>SUCCESS</status>`
	ESC fails to recover the VM by *Reboot* and proceeds with recovery by *Undeploy* and then *Redeploy*.
	It un-sets monitoring and un-deploys the VM.
	The following NETCONF notification is triggered:
	`<type>VM_RECOVERY_UNDEPLOYED</type>`
	`<status>SUCCESS</status>`
	If ESC receives an error or if the maximum attempts for recovery is reached.
	The following NETCONF notifications is triggered:
	`<type>VM_RECOVERY_COMPLETE</type>`
	`<status>FAILURE</status>`

Scenario	Notifications

Error Path For ESC-NORTHBOUND Recovery Call Flow Before VM Alive - Undeploy/Redeploy	When Northbound places a deploy request to ESC, ESC deploys VMs and set KPI to monitor on all VM Alives received. The following NETCONF notification is triggered:
	`<type>SERVICE_ALIVE</type>`
	`<status>SUCCESS</status>`
	After ESC receives VM down event, the following NETCONF notification is triggered:
	`<type>VM_RECOVERY_INIT</type>`
	`<status>SUCCESS</status>`
	ESC un-sets monitoring and un-deploys the VM. Recovery is performed by *Undeploy* and then *Redeploy*.
	The following NETCONF notification is triggered:
	`<type>VM_RECOVERY_UNDEPLOYED</type>`
	`<status>SUCCESS</status>`
	If ESC receives an error or if the maximum attempts for recovery is reached.
	The following NETCONF notifications is triggered:
	`<type>VM_RECOVERY_COMPLETE</type>`
	`<status>FAILURE</status>`
	`<type>SERVICE_ALIVE</type>`
	`<status>FAILURE</status>`

ESC-NORTHBOUND Recovery Call Flow After VM Alive -VM_RECOVERY_CANCELLED	When Northbound places a deploy request to ESC, ESC deploys VMs and sets KPI to monitor all VM Alive notifications received. The following NETCONF notification is triggered:
	`<type>SERVICE_ALIVE</type>`
	`<status>SUCCESS</status>`
	After ESC receives VM down event, the following NETCONF notification is triggered:
	`<type>VM_RECOVERY_INIT</type>`
	`<status>SUCCESS</status>`
	During the recovery wait time, if VM is back to normal, then the VM_RECOVERY_CANCELLED notification is sent. Recovery action is not performed.
	`<type>VM_RECOVERY_CANCELLED</type>`
	`<status>SUCCESS</status>`

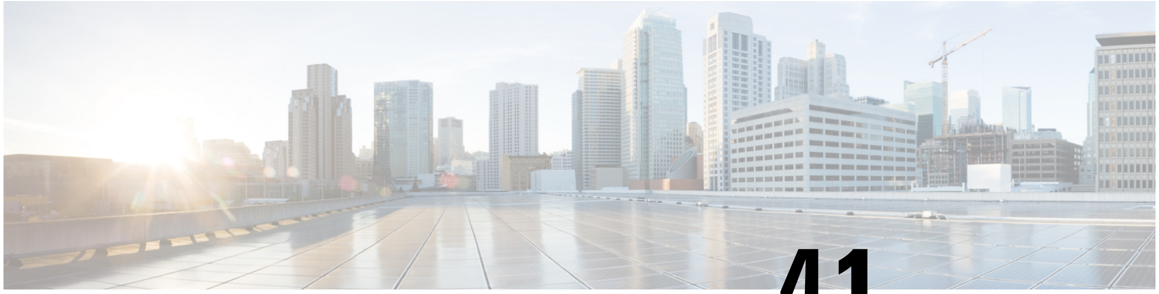
Scenario	Notifications

ESC-NORTHBOUND Recovery Call Flow After VM Alive - Reboot	When Northbound places a deploy request to ESC, ESC deploys VMs and sets KPI to monitor all VM Alive notifications received. The following NETCONF notification is triggered:
	`<type>SERVICE_ALIVE</type>`
	`<status>SUCCESS</status>`
	After ESC receives VM down event, the following NETCONF notification is triggered:
	`<type>VM_RECOVERY_INIT</type>`
	`<status>SUCCESS</status>`
	ESC performs hard reboot on the VM and sends reboot notification.
	`<type>VM_RECOVERY_REBOOT</type>`
	`<status>SUCCESS</status>`
	And the VM alive event is received within the boot time.
	`<type>VM_RECOVERY_COMPLETE</type>`
	`<status>SUCCESS</status>`

Error Path For ESC-NORTHBOUND Recovery Call Flow After VM Alive - Reboot	When Northbound places a deploy request to ESC, ESC deploys VMs and sets KPI to monitor on all VM Alive notifications received. The following NETCONF notification is triggered:
	`<type>SERVICE_ALIVE</type>`
	`<status>SUCCESS</status>`
	After ESC receives VM down event, the following NETCONF notification is triggered:
	`<type>VM_RECOVERY_INIT</type>`
	`<status>SUCCESS</status>`
	Then ESC sends reboot notification.
	`<type>VM_RECOVERY_REBOOT</type>`
	`<status>FAILURE</status>`
	ESC receives an error while attempting to recover through *Reboot*.
	The following NETCONF notification is triggered:
	`<type>VM_RECOVERY_COMPLETE</type>`
	`<status>FAILURE</status>`

PART VI

ESC Portal



CHAPTER 41

Getting Started

Logging In to the ESC Portal

Note

- The ESC portal is enabled by default. You must ensure that the ESC portal is not disabled during installation. For more information on enabling or disabling the ESC portal, see Installing ESC in the Cisco ESC Install and Upgrade Guide.
- When you log in to the ESC portal for the first time you are prompted to change the default password.

To log in to the ESC portal, do the following:

Before you begin

- Register an instance of ESC. For more information on registering the ESC instance, see the Cisco Elastic Services Controller Install and Upgrade Guide.
- Ensure that you have the username and password.

Procedure

Step 1 Using your web browser, enter the IP address of ESC.

Example:

For example, if the IP address of ESC is 192.0.2.254, enter:

https://192.0.2.254 [login via https]. The portal runs on default security port 443.

A Security Alert message is displayed.

Step 2 Click **Yes** to accept the security certificate. The Login page is displayed.

Step 3 Enter the username and password and click **Login** .

If you are logging in for the first time, the login page reappears, prompting you to change your password.

Step 4 Enter the old password in the Old Password field, then enter a new password in the New Password and Confirm Password fields.

Step 5 Click **Update Password** or press **Enter**.

Note
- If the portal becomes unresponsive, restart the portal by executing the **escadm portal restart** from the escadm tool.
- ESC portal only supports one user.
- Currently, a pre-installed self-signed certificate supports HTTPS. The user must confirm the self-signed certificate before proceeding with the ESC portal.
- In HTTPS communication mode, if the URL protocol type returned by OpenStack is not HTTPS, the access to the VNF Console may be disabled. For security reasons, while running in HTTPS more non-secure communication will be rejected.

Changing the ESC Password

You will be forced to change the default password on first time login. Portal will not let you bypass this step and will keep returning you to this page until you change the default password. After the first time password change, you can change your password using the procedures described in this section. Also, if the user has multiple browsers or tabs or the SAME user is logged on by 2 or more computers and one of the user changes the password then everyone will be logged off and asked to re-enter the new password. If the user is idle in the portal for more than 20 minutes, then the user is logged out. You can configure the user idle timeout in the portal environment file. If you forgot your password, you can also reset the password.

This section discusses how to change the portal password.

Changing the ESC Portal Password

To change an existing ESC portal password from the portal, do the following:

Procedure

Step 1 Log in to ESC portal using your username and password.

Step 2 Click the user icon on the upper right corner of the screen.

Step 3 Choose **Account Settings**. The page to update account information and password appears.

Step 4 Click **Update Password**.

Step 5 Enter the old password in the Old Password field, then enter a new password in the New Password and Confirm Password fields.

Step 6 Click **CREATE**.

What to do next

For information on how to change the password using the CLI and so on, see Cisco Elastic Services Controller Install and Upgrade Guide

ESC Portal Dashboard

The Cisco Elastic Services Controller dashboard provides a tabular representation of all the managed ESC resources such as tenants, flavors, and images, deployments, incoming requests, notifications, and visual indicators of system health. The following dashboard elements help you track, monitor and diagnose data and system health over time.

The dashboard is best used in a monitoring desk context, where the system displaying the dashboard is dedicated for that purpose and might be distinct from the systems running the portal servers. The dashboard system should point its browser to the system running the portal servers.

If you notice unusual spikes or drops in activity, there could be communication failures or power outages on the network that you need to investigate.

In case of HA Switchover, the user must log out and log in to view the portal resources.

Table below lists the details you can view in the portal:

Note These tasks can also be performed using the NB APIs. See the Elastic Services Controller NB APIs, on page 7 for more details.

Table 29: Portal Details

Task	Navigate	Description
To view Dashboard	Choose **Dashboard**	View the summary of all the managed ESC resources, notifications, system configuration and the system health.
To view notifications	Choose **Notifications** or Click the notification icon on the top right corner of the portal.	Displays notifications received on the Portal from ESC.

Task	Navigate	Description
To deploy a VNF	Choose **Deployments** **Important** To deploy a VNF in the VMware vCenter using a form, see Deploying Using a Form .	Deploys a VNF. The drag and drop feature allows you to grab an existing deployment data model and to re-use it by dragging the file to the deployment table. You can even use the upload xml on the table toolbar, which allows you to browse appropriate file from your file system. **Note** Only xml files are accepted. The drag and drop feature executes a REST call as of now and does not execute NETCONF calls.
To view existing deployments (for both OpenStack and VMware vCenter)	Choose **Deployments**, and select a deployment from the table. • Click **View VM Groups**. You can view further details such as monitoring, scaling, and other information on the corresponding tabs.	Displays a high level summary of deployments that are currently being deployed. You can view the name and status of the deployments, and the number of VMs that are deployed in the deployment.
To view VIMs	Choose **Resources** > **VIMs**	Provides a list of VIMs with its VIM id, the type of VIM, status of the VIM, properties and the VIM users.
To view tenants (OpenStack only)	Choose **Resources** > **Tenants**	Provides a list of tenants, along with its name, description and ID. **Important** ESC does not support multi-tenancy on VMware vCenter. Portal does automatic rollback of resources if the resources failed to be created on the VIM. In some cases (because of conflicting dependencies), the tenant has to be deleted manually after getting a rollback failure notification.
To view VNF Images	Choose **Resources** > **Images**	Provides a list of images for the selected resources.

Task	Navigate	Description
To view VNF deployment flavors (OpenStack only)	Choose **Resources** > **Flavors**	Provides a list of flavors for the selected resources.
To view networks	Choose **Resources** > **Networks**	Displays details of network, tenant name, network ID, network type and so on for each network. of sub-network and interfaces. You can find details such as name, network ID, tenant ID and so on for each of them.
To view subnetworks (OpenStack only)	Choose **Resources** > **Subnetworks**	Displays details of subnetwork, network ID, subnet ID and so on for each subnetwork. **Note** Subnetwork and interfaces tabs are available only on OpenStack. On initial booting of ESC VM, the network and sub-network creation forms may show an empty tenant combo box. Refresh the page to load the tenants correctly.
To view interfaces (OpenStack only)	Choose **Resources** > **Interfaces**	Displays details of interfaces, network ID, subnet ID, VM name and so on for each interface.
To view the switch details (VMware vCenter only)	Choose **Resources** > **Switches**	Provides a list of switches, its names, descriptions, UUIDs and hosts.
To deploy VNFs using a deployment template	Choose **System** > **Deployment Template**	Creates preconfigured deployment templates
To view incoming requests to ESC	Choose **System** > **Incoming Requests**	Lists all the incoming requests to ESC such as Transaction ID and request details.
To view configurations	Choose **System** > **Configuration**	Lists all the configuration parameters used for configuring VMs, monitoring rules, applying policies during VM deployments, and so on.
To view boot parameters (OpenStack only)	Choose **System** > **Boot Parameters**	Lists all the boot parameters used to boot ESC.

Task	Navigate	Description
To view host details (OpenStack only)	Choose **System** > **Host Details**	Lists the host details such as Operating System (OS), version of the OS, System uptime , RAM, Storage and other details.
To view the health of ESC (OpenStack only)	Choose **System** > **Health**	Show the health of ESC, Confd status, Operational status and other details.
To download logs	Choose **System** > **Logs**	Allows you to download log messages.
To view the infrastructure details (OpenStack only)	Choose **Infrasctructure** > **Instances**	All VMs running on the virtualization infrastructure.
To view the Hypervisors (OpenStack only)	Choose **Infrasctructure** > **Hypervisors**	All hypervisors running on the virtualization infrastructure.
To undeploy a VNF	• Choose **Deployments**. • Select a deployment from the table, and click **X** on the table toolbar to undeploy.	Undeploys VNF(s).
To view VDC (VMware vCenter only)	Choose **Resources** > **Datacenters**	View list of all Virtual Datacenters.

Note The ESC portal pages might have table formatting issue, if viewed in a small screen. The browser screen must be 15 inches or more for the tables to appear correctly.

The System Panel comprises of the following tabs:

- **Performance**—Displays the tabular and graphical representation of the performance data.
- **Storage**—Displays the disk usage information.
- **vCPU Utilization**— Displays the usage of vCPUs in the ESC VM.
- **Health**—Displays the health of various ESC processes such as network, database, and tomcat.
- **Host Details**—Displays the host details such as Operating System (OS), version of the OS, System uptime , RAM, and Storage details.

Notifications

The Notification page lists all the notifications for the ESC deployments:

- Error Events—

  Select the error event from the notifications page, and click **View More Info** to see a complete report of the error event.

Note Error events with explicit error messages do not have a detailed report.

A complete report can also be generated using the REST API. A *troubleshooting-Id* is included in the ESC-Status-Message to generate a report.

- Clear Notifications—

  You can sort the notifications by date and delete the notifications. Click **Clear Notifications** to delete all the notifications.

CHAPTER 42

Managing Resources Using ESC Portal

Managing VIM Connectors Using ESC Portal

ESC supports adding and updating VIM connectors and VIM users using the ESC portal. You can add or update multiple VIMs to manage the multi VIM deployment. For more information on multi VIM deployment, see Deploying VNFs on Multiple OpenStack VIMs.

The VIM connector table shows details such as the VIM id, the type of VIM, status of the VIM, properties and the VIM users.

Adding and Deleting VIM Connectors

To add or delete the VIM connectors, perform the following:

Procedure

Step 1 Choose **Resources** > **VIMs**.

Step 2 Click Upload XML and select a file. The Confirm VIMs dialog box appears.

Step 3 Click CONFIRM to upload the XML file.

Step 4 To delete a VIM from the list of VIMs, select the VIM and click **X**. A dialog box appears.

Step 5 Click OK to delete the VIM.

You cannot delete the default VIM connector, and the VIM connector with resource dependencies.

Managing VIM Users

The VIM user details are available under the view details tab. The ESC portal allows you to create, update and delete the VIM users.

Procedure

Step 1 Select the VIM connector from the **Resources** > **VIM** table, and click **View Details**.

The properties and the VIM user page appears.

Step 2 Click **OK** to confirm.

To update a VIM user, select the user, and then click upload XML to upload an updated XML.

To delete a VIM user, select the VIM user in the table, and click **X**. The VIM user is deleted.

For more information on VIM connectors and VIM users, see Configuring the VIM Connector, on page 50.

Managing OpenStack Resources Using ESC Portal

The following sections explain how you can manage OpenStack resources using the ESC Portal by:

- Adding and deleting tenants
- Adding and deleting images
- Adding and deleting flavors
- Adding and deleting networks
- Adding and deleting subnetworks

Adding and Deleting Tenants in ESC Portal

To add and delete tenants from the ESC portal, do the following:

Procedure

Step 1 Choose **Resources** > **Tenants**.

Step 2 Click **+** to add a tenant. The Add Tenant dialog box appears.

Step 3 Add a name and a description, and click **Create**.

Step 4 To delete a tenant, select the tenant from the list of tenants, and click **X**.

Step 5 Click OK to delete.

Adding and Deleting Images in ESC Portal (OpenStack)

To add and delete images from the ESC portal, do the following:

Procedure

Step 1 Choose **Resources** > **Images**.

Step 2 Drag and drop your images file to the Images table. The Confirm Image dialog box appears.

Step 3 Click **CONFIRM** to create an image from the dragged template.

Step 4 To delete an image from the list of images, select the image and click **X**. A dialog box appears.

Step 5 Click **OK** to delete the image.

Adding and Deleting Flavors in ESC Portal

To add and delete flavors from the ESC portal, do the following:

Procedure

Step 1 Choose **Resources** > **Flavors**.

Step 2 Drag and drop your file to the Flavor table. The Confirm Flavor dialog box appears.

Step 3 Click **CONFIRM** to create a flavor from the dragged template.

Step 4 To delete a flavor from the list of flavors, select the flavor and click **X**. A dialog box appears.

Step 5 Click **OK** to delete the flavor.

Adding and Deleting Networks in ESC Portal

To add and delete networks from the ESC portal, do the following:

Procedure

Step 1 Choose **Resources** > **Networks**.

Step 2 Drag and drop your file to the Networks table. The Confirm Network dialog box appears.

Step 3 To delete a network from the list of networks, select the network and click **X**. A dialog box appears.

Step 4 Click **OK** to delete the network.

Adding and Deleting Subnetworks in ESC Portal

To add and delete subnetworks from the ESC portal, do the following:

Procedure

Step 1 Choose **Resources** > **Subnetworks**.

Step 2 Drag and drop your file to the Subnetworks table.

Note The drag and drop feature executes a REST call as of now and does not execute NETCONF calls.

Step 3 To delete a subnet from the list of subnets, select the subnet and click **X**. A dialog box appears.

Step 4 Click **OK** to delete the subnetwork.

Managing VMware vCenter Resources Using ESC portal

The following sections explain how you can manage VMware vCenter resources using the ESC Portal by:

- Adding and deleting images
- Adding and deleting networks

Adding and Deleting Images in ESC Portal (VMware)

The ESC portal allows you to create an image by filling the appropriate fields in the form.

Creating Image from a Form

To create images from a form, do the following:

Procedure

Step 1 Choose **Resources** > **Images**.

Step 2 Click **+** to add a VNF Image. The add Image to datacenter windows appears.

Step 3 From the **Virtual Datacenter** drop-down list, select the datacenter where you want to create the image.

Step 4 In the **Image Name** field, enter the image name.

Step 5 In the **Image Path** field, enter the image path.

Step 6 Click **Create** to create an image.

Step 7 To delete the image, select the image from the list, and click **X**. A dialog box appears.

Step 8 Click **OK** to delete the image.

Adding and Deleting Networks in ESC Portal (VMware)

To add and delete networks from the ESC portal, do the following:

Procedure

Step 1 Choose **Resources** > **Networks**, to create networks from a form.

Step 2 Click **+** to add a networks. The add network to datacenter window appears.

Step 3 From the **Virtual Datacenter** drop-down list, select the datacenter where you want to add the network.

Step 4 From the **Switch** drop-down list, select a switch.

Step 5 In the **Network Name** field, enter the network name.

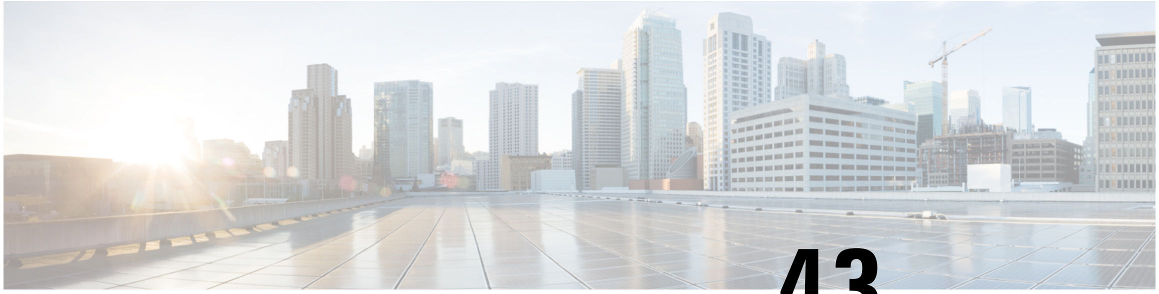
Step 6 In the **VLan** field, enter the number of VLANs.

Step 7 In the **Number of Ports** field, enter the number of ports.

Step 8 Click **Create** .

Step 9 To delete the network, select the network from the list, and click **X**. A dialog box appears.

Step 10 Click **OK** to delete the network.



CHAPTER 43

Deploying VNFs Using ESC Portal

Deploying Virtual Network Functions Using ESC Portal (OpenStack Only)

You can use the ESC portal to deploy a single VNF or multiple VNFs together by deploying a datamodel XML file. You can use the ESC portal to deploy a single VNF or multiple VNFs together either by:

Procedure

Deploying using a file—You can upload an existing datamodel file.

The following sections explain how to deploy VNFs using the ESC portal.

Deploy Using a File (Deployment Data model)

An existing deployment data model is used to deploy VNFs. The deployment data model is preconfigured with the number of VNFs and other specifications. It is either uploaded by locating the deployment data model or you can drag and drop the existing deployment data model. The drag and drop feature allows you to grab an existing deployment data model and to reuse it by dragging the file and dropping it off to the deployment table.

Note Only XML files are accepted.

Procedure

Step 1 Choose **Deployments**.

Step 2 Drag and drop your file to the Deployments table, or click Upload XML on the table toolbar to browse and select the file.

Note The drag and drop feature executes a REST call as of now and does not execute NETCONF calls.

Deploying VNFs on VMware vCenter using ESC Portal

The ESC portal allows you deploy a single VNF or multiple VNFs together. An existing deployment data model is either uploaded through the portal, or a new deployment data model is created. A new deployment data model is created by filling all the appropriate fields in the ESC portal. ESC also allows you to export a deployment data model from the portal. The following section explains multiple ways to deploy VNFs using the ESC portal.

The following sections explain how to deploy VNFs using the ESC portal.

Procedure

Step 1 Deploy using a file.

Step 2 Deploy using a form.

Deploy Using a File (Deployment Data model)

An existing deployment data model is used to deploy VNFs. The deployment data model is preconfigured with the number of VNFs and other specifications. It is either uploaded by locating the deployment data model or you can drag and drop the existing deployment data model. The drag and drop feature allows you to grab an existing deployment data model and to reuse it by dragging the file and dropping it off to the deployment table.

Note Only XML files are accepted.

Procedure

Step 1 Choose **Deployments**.

Step 2 Drag and drop your file to the Deployments table, or click Upload XML on the table toolbar to browse and select the file.

Note The drag and drop feature executes a REST call as of now and does not execute NETCONF calls.

Deploying Using a Form

To create a new deployment template, do the following:

Note Click **Export Template** to export a deployment data model.

Procedure

Step 1 Choose **Deployments**.

Step 2 Click + to deploy using a form.

Step 3 Enter a **Deployment name**.

Step 4 From the **Datacenter** drop-down list, choose a datacenter on which you want to deploy the VNF.

For more information on virtual datacenter, see Deploying Virtual Network Functions on VMware vCenter.

Step 5 In the **General** tab, enter the appropriate values for the fields.

a) In the **Placement** field, select the **Cluster** or **Host** radio button .

- **Cluster**—Choose the name of a cluster to deploy a VNF in the same cluster.
- **Host**— Choose a host to deploy a VNF in the same host.
- **Datastore**— Choose a datastore for the selected cluster.
- **Image** Choose an image.

Step 6 Click **Enable Smart Licensing** to enable smart licensing.

Step 7 Click **Enable Intragroup Rules** to enable intragroup rules.

a) From the **Type** drop-down list, choose **Affinity** or **Anti-Affinity** to enable affinity or anti-affinity rules.

For more information on intragroup affinity rules, Affinity and Anti-Affinity Rules, on page 191.

Step 8 (Optional) Click the **Add VNF Intergroup Rule** tab to select VNFs for which you want the affinity or anti-affinity rules to be applicable.

For more information on intergroup affinity rules, see Affinity and Anti-Affinity Rules, on page 191.

Step 9 To specify the parameters that ESC will utilize to heal the VNFs when there is a failure, click the **Recovery** tab.

For more information on recovery or healing, see Healing Virtual Network Functions, on page 329.

Step 10 To specify the number of interfaces and properties for each interface, click the **Interfaces** tab. The order of the interfaces specified here does not correspond to the order of the interfaces in the VM.

a) Click **Add Interface** to add interfaces.

Step 11 To specify the number of instances of a particular type of VM that needs to be instantiated and to elastic scale in and scale out, click the **Scaling** tab.

a) Click **Add Static IP Pool** to add a static IP pool.

Step 12 To specify the monitoring rules that will be used to configure the monitor module within ESC, click the **Monitoring** tab.

For more information on monitoring, see Monitoring Virtual Network Functions, on page 299.

Step 13 In the **Config Data** tab, enter the appropriate values for the fields.

Step 14 (Optional) In the **OVF Settings** tab, enter the appropriate values for the fields.

a) Click **Add OVF Property** to add a list of OVF properties.

Deploying Virtual Network Functions Using a Deployment Template

You can now deploy VNFs by uploading a preconfigured deployment template through the ESC portal.

1. Navigate to **System** > **Deployment Templates**
2. Click **Upload XML**.

   You can drag and drop, or choose a preconfigured deployment template (dep.xml) and click **Confirm**. The deployment template appears in the table.
3. Select the uploaded deployment template, and Click **Deploy from Template**.
4. The deployment name and tenant name are added from the uploaded template. Modify the fields if necessary, or click **Create** to create the template.
5. A success message appears on the screen. Click **Ok**.

The new deployment template appears in the Deployments view.

Preconfigured template

You can make changes to an existing dep.xml to use as a preconfigured template. You must make the following changes to the datamodel:

- Use esc_datamodel_template tag instead of esc_datamodel.
- The esc_datamodel_template name property is unique and must be specified to identify the template.
- *param_key* is used by the portal to identify customizable values. This is a required field. This key is unique, but can appear multiple times in the template.
- *prompt* shows the input value that needs to be added by the user. This is a required field. If the prompt is different for the same param_key specified elsewhere in the document, the first prompt is used.
- *core*, is the default value, which can be left blank.
- *required* specifies if the user must enter this value. This is an optional field. The default value is true.
- *range* validates the number field. This is an optional field.

Sample preconfigured template:

```
<?xml version="1.0" encoding="UTF-8"?>
<esc_datamodel_template xmlns="http://www.cisco.com/esc/esc" name="VPC Template 1">
   <tenants>
      <tenant>
         <name param_key="tenant_name" prompt="Tenant Name">core</name>
         <managed_resource>false</managed_resource>
         <deployments>
```

```
            <deployment>
               <name param_key="dep_name" prompt="Deployment
Name">vnfd3-deployment-1.0.0-1</name>
               <policies>
                  <placement>
                     <target_vm_group_ref>c2</target_vm_group_ref>
                     <type>anti_affinity</type>
                     <enforcement>strict</enforcement>
                     <vm_group_ref>c1</vm_group_ref>
                  </placement>
               </policies>
            </deployment>
         </deployments>
      </tenant>
   </tenants>
</esc_datamodel_template>
```

CHAPTER 44

VNF and VM Operations Using ESC Portal

You can perform VNF operations such as starting, stopping and rebooting using the portal. The VNF operations can be performed on deployed VNFs depending on the state of the deployment.

Performing VNF Operations

To perform VNF operations:

Procedure

Step 1 Choose **Deployments**.

Step 2 Select a VNF on the deployments page.

Note The operations are enabled depending on the state of the deployment.

Step 3 Click the necessary operation from the table toolbar. See the table below for the list of operations you can perform.

The VNFs must be in the following deployment states to perform the operations:

VNF operations	Deployment state
Enable Monitor	Inert or Error
Disable Monitor	Active
Start VNF	Stopped
Stop VNF	Active or Inert
Reboot VNF	Active or Inert
Recover VNF	Error

VNF operations	Deployment state
Monitor and Recover VNF (Manual Recovery)	Error

Performing VM Operations

To perform VM operations:

Procedure

Step 1 Choose **Deployments**.

Step 2 Select a VNF on the deployments page.

Note The operations are enabled depending on the state of the deployment.

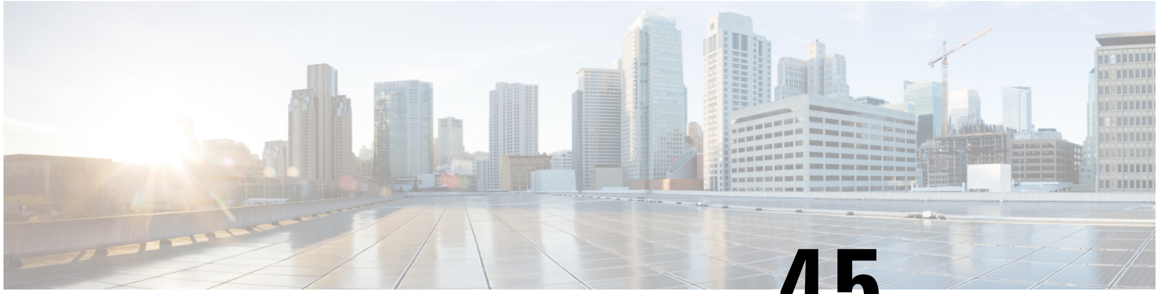
Step 3 Click View VM Groups.

Step 4 Under VM Group Instances, select an operation. See the table below for the list of operations you can perform.

Step 5 Click Confirm.

The VMs must be in the following deployment states to perform the operations:

VM operations	Deployment state
Enable Monitor	Inactive or Error
Disable Monitor	Active
Start VM	Shutoff
Stop VM	Active or Inactive
Reboot VM	Active or Inactive
Recover VM	Error



CHAPTER 45

VNF and VM Recovery Using the Portal

VNF and VM Recovery Using the Portal

You can now perform manual recovery of VNFs and VM using the ESC portal:

Procedure

Step 1 Choose **Deployments**.

Step 2 Select a deployment in error state.

For VM level recovery, select a VM in error state from the View VNFs page.

Step 3 Click **Recover VNF** or **Monitor + Recover VNF**.

Step 4 Click **OK** to confirm.

Step 5 Select a **Recovery Action** from the Recovery Action dropdown and click **OK**.

The following recovery actions are available:

- Default—Triggers the recovery action that was defined in the datamodel.
- REBOOT_ONLY
- REDEPLOY_ONLY
- REBOOT_THEN_REDEPLOY

For more information on the recovery options, see Recovery Policy, on page 331.

Important Points

1. Configurable manual recovery does not support the in-flight transaction behaviour. So if a failover happens during a configurable manual recovery, the manual recovery resumes with predefined recover action.
2. For deployment migration, use the default recovery policy. Cisco does not provide recovery action for VM/VNF manual recovery in LCS based recovery.

CHAPTER 46

ESC System Level Configuration

Downloading Logs from the ESC Portal

You can now download all log files from the ESC portal. There are two types of logs:

- Trace logs: This includes vimmanager log, esc_rest log, and esc_netconf log.
- System logs: This includes escmanager log, vimmanager log, and all other ESC related logs except for the trace logs.

Procedure

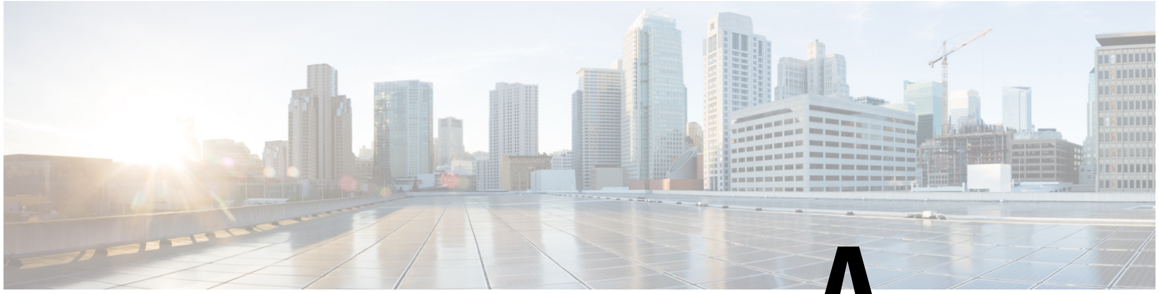
Step 1 Choose **System** > **Logs**.

Step 2 Click **Request message trace logs** for trace logs, or **Request system logs** for all ESC related logs.

The downloadable file appears (after it is created) in the table.

If the logs are huge, it might take more time to compile. You must wait for some time, before downloading the files.

Step 3 Click the downloadable file to save it on your machine.



APPENDIX A

Cisco Cloud Services Platform (CSP) Extensions

Cloud Services Provider Extensions

The table below lists all the additional extensions added to ESC to support CSP as a VIM. For more information on the VIM connectors, see Configuring the VIM Connector.

Table 30: CSP Extensions

Resource	**Extension** **Sample Deployment**
Deployment/VM Group	Extension: None. Sample deployment: See the Managing VIM Connectors section
Flavor	Extension: None. Sample deployment:

```
<flavor>
    <name>FLAVOR_2_4096_10000</name>
    <vcpus>2</vcpus>
    <memory_mb>4096</memory_mb>
    <root_disk_mb>10000</root_disk_mb>
</flavor>
```

Resource	Extension Sample Deployment
Storage disk Deployment/volume	Extension:

```
<extension>
    <name>volumes</name>
    <containers>
        <container>
            <name>1</name>
            <properties>
                <property>
                    <name>storage_disk_format</name>
                    <value>raw | qcow2</value>
                </property>
                <property>
                    <name>storage_disk_device</name>
                    <value>disk | cdrom</value>
                </property>
                <property>
                    <name>storage_disk_location</name>
                    <value>local | NFS mount </value>
                </property>
            </properties>
        </container>
    </containers>
</extension>
```

Sample deployment:

```
<volumes>
    <volume>
        <volid>1</volid>
        <sizeunit>GiB</sizeunit>
        <size>20</size>
        <bus>virtio</bus>
    </volume>
</volumes>
```

Resource	Extension Sample Deployment
Deployment/ vm group / extentions/ image	Extension:

```
<extension>
    <name>image</name>
    <properties>
        <property>
            <name>disk-resize</name>
            <value>true</value>
        </property>
        <property>
            <name>disk_type</name>
            <value>virtio</value>
        </property>
        <property>
            <name>disk_storage_name</name>
            <value>esc_nas_old</value>
        </property>
        <property>
            <name>image_storage_name</name>
            <value>esc_nas_old</value>
        </property>
    </properties>
</extension>
```

Resource	**Extension** **Sample Deployment**
Deployment/ vm group / extentions/ vnc	Extension:

```
<extension>
    <name>vnc</name>
    <properties>
        <property>
            <name>vnc_password</name>
            <value>**********</value>
        </property>
    </properties>
</extension>
```

Deployment/ vm group / extentions/ vnf_mgmt_ip	Extension:

```
<extension>
    <name>vnf_mgmt_ip</name>
    <properties>
        <property>
            <name>nicid</name>
            <value>0</value>
        </property>
    </properties>
</extension>
```

Deployment/ vm group / serial_ports	Extension:

```
<extension>
    <name>serial_ports</name>
    <containers>
        <container>
            <name>0</name>
            <properties>
                <property>
                    <name>serial_type</name>
                    <value>console</value>
                </property>
            </properties>
        </container>
    </containers>
</extension>
```

Resource	Extension Sample Deployment
Deployment/ vm group / interfaces /	Extension:

```
<extensions>
    <extension>
        <name>interfaces</name>
        <containers>
            <container>
                <name>1</name>
                <properties>
                    <property>
                        <name>passthroughMode</name>
                        <value>none</value>
                    </property>
                    <property>
                        <name>tagged</name>
                        <value>false</value>
                    </property>
                    <property>
                        <name>type</name>
                        <value>access</value>
                    </property>
                    <property>
                        <name>bandwidth</name>
                        <value>7500</value>
                    </property>
                    <property>
                        <name>vlan</name>
                        <value>1</value>
                    </property>
                </properties>
            </container>
        </containers>
    </extension>
</extensions>
```

Sample deployment:

```
<interface>
    <nicid>1</nicid>
    <type>virtual</type>
    <model>virtio</model>
    <network>Eth0-2</network>
    <admin_state_up>false</admin_state_up>
</interface>
```

Deployment/ vm group / <vim_vm_name>	Extension: None. Sample deployment: <vim_vm_name>my-custom-csr</vim_vm_name>

Resource	Extension Sample Deployment
Deployment/ vm group /day0-volume-id	Extension: None. Sample deployment:

```
<config_type>CONFIG_DATA_OPTIONS</config_type>
<config_options>
    <options>
        <option>
            <name>day0-volume-id</name>
            <value>cidata</value>
        </option>
    </options>
</config_options>
```