Cisco Elastic Services Controller 4.2 User Guide

First Published: 2018-06-28

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

CONTENTS

Preface

Purpose

This guide will help you to perform tasks such as deploying, monitoring, scaling and healing of VNFs. To setup, provision, and configure Cisco Elastic Services Controller (ESC), see the Cisco Elastic Services Controller Install and Upgrade Guide.

Audience

This guide is designed for network administrators responsible for provisioning, configuring, and monitoring VNFs. Cisco Elastic Services Controller (ESC) and its VNFs are deployed in a Virtual Infrastructure Manager (VIM). Currently OpenStack, VMware vCenter, and Amazon Web Services (AWS) are the supported VIMs. The administrator must be familiar with the VIM layer, vCenter, OpenStack and AWS resources, 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
AWS	Amazon Web Services (AWS) is a secure cloud services platform, offering compute, database storage, content delivery and other functionalities.
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.

Terms	Definitions
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.
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.
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.
VNFM	Virtual Network Function Manager (VNFM) manages the life cycle of a VNF.
vMS	Cisco vMS is a Network Functions Virtualization (NFV) orchestration platform that enables fast deployment of cloud-based networking services.

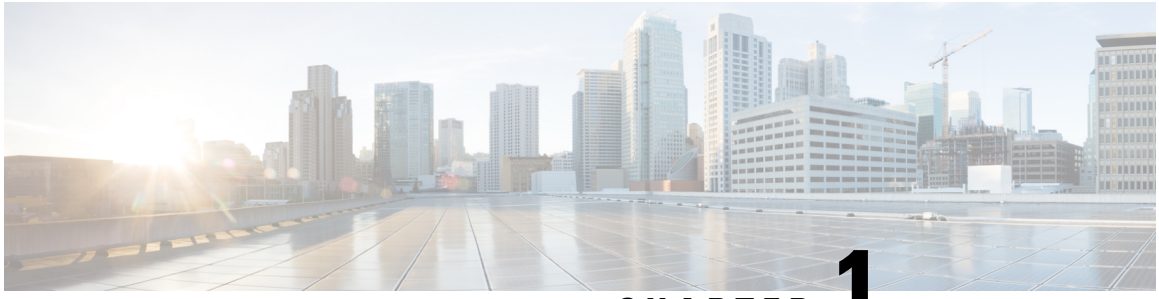
Obtaining Documentation Request

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 (OpenStack only).
- 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 to provide better hierarchical configuration and data modularity.
- Supports ETSI MANO interface for a subset of VNF lifecycle management operations.
- Supports deploying VMs on Single or multiple AWS VIMs.
- Supports deploying VMs on VMware vCloud Director VIM.

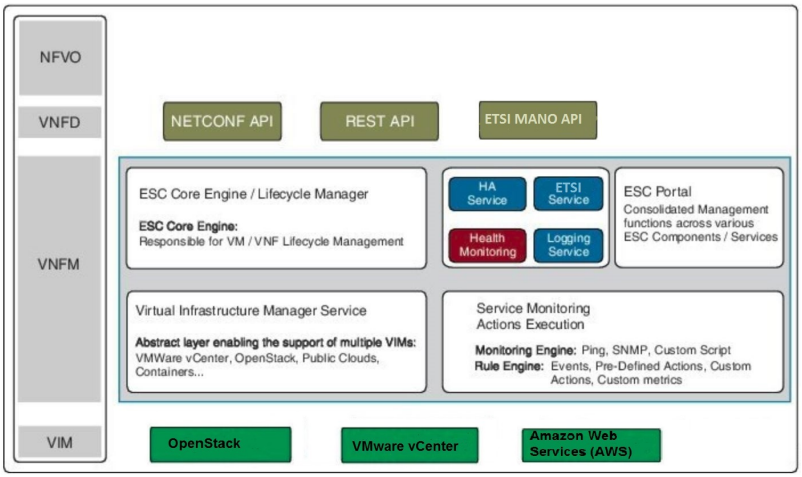
ESC Architecture

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 MANO NB 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.

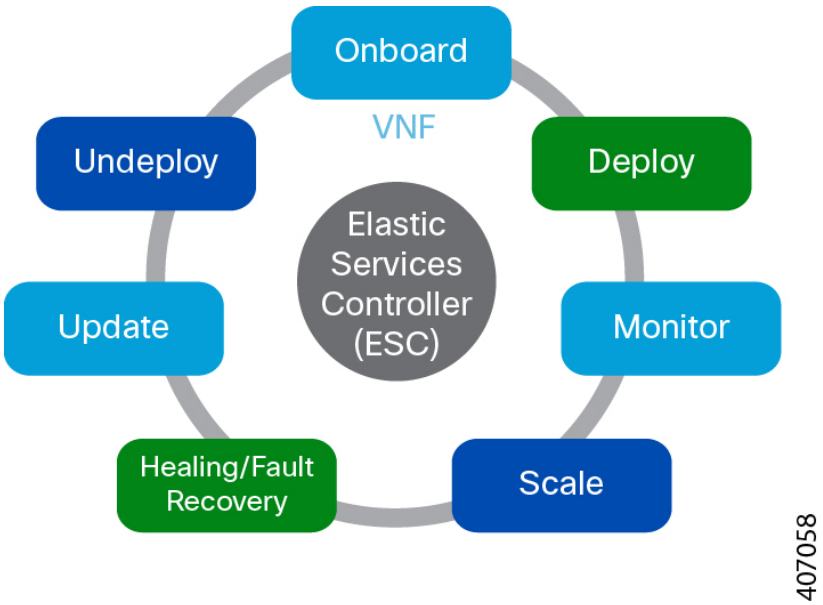
You can orchestrate VNFs within a virtual infrastructure domain—either on OpenStack or VMware vCenter. A VNF deployment is initiated as a service request. The service request comprises of templates that consist of XML payloads and configuration parameters.

Note You can deploy VNFs either on OpenStack or VMware vCenter. Hybrid VNF deployment is not supported.

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 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 MANO API, the VNF is onboarded to the NFVO. For more information, see prerequisites in VNF Lifecycle Operations Using ETSI API.

- **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 MANO API at this stage of the lifecycle. For more information, see Creating VNF Identifier in VNF Lifecycle Operations Using ETSI API.

- **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 a 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 ETSI API, see VNF Lifecycle Operations Using ETSI API.

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 15. 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 onOpenStack.

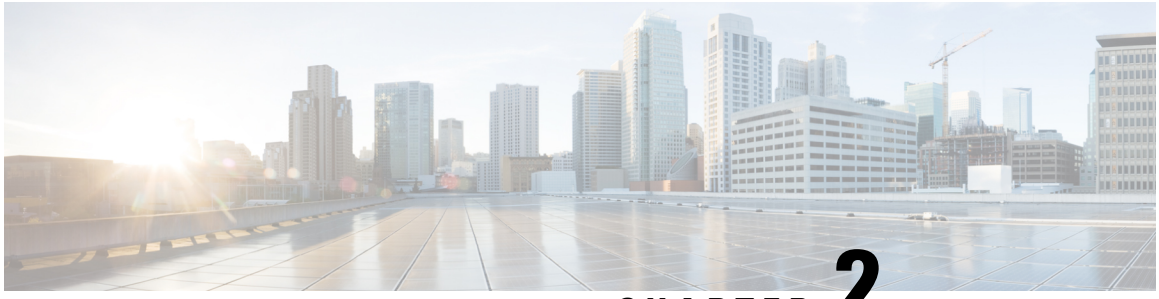
Deploying VNFs on VMware vCenter

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 15. The *deployment data model* refers to the resources to deploy VNFs on VMware vCenter.

In Cisco ESC Release 2.0 and later, 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 26. The *deployment data model* refers to these images to deploy VNFs.

Scenarios	Description	data model templates	Images	Advantages
Deploying VNFs on a single VIM 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 Deploying VNFs.



CHAPTER 2

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 Virtual Managed Services (vMS).
- 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 vMS, 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, on page 7
- Elastic Services Controller Portal, on page 13

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.

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 command <file name>* instead of the complete path for any CRUD operations using the netconf CLI.

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 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>
```

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 message-id="1" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
```

```
    <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>10.20.0.0</address>
            <no_gateway>true</no_gateway>
            <!-- DISABLE GATEWAY -->
```

```
            <gateway>10.20.0.1</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 MANO Northbound API

The ETSI MANO 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, ETSI MANO Northbound API Overview.

ETSI MANO API Documentation

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

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

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

Managing Resources Overview

Cisco Elastic Services Controller (ESC) resources comprises 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 26. The *deployment data model* refers to these images to deploy VNFs.

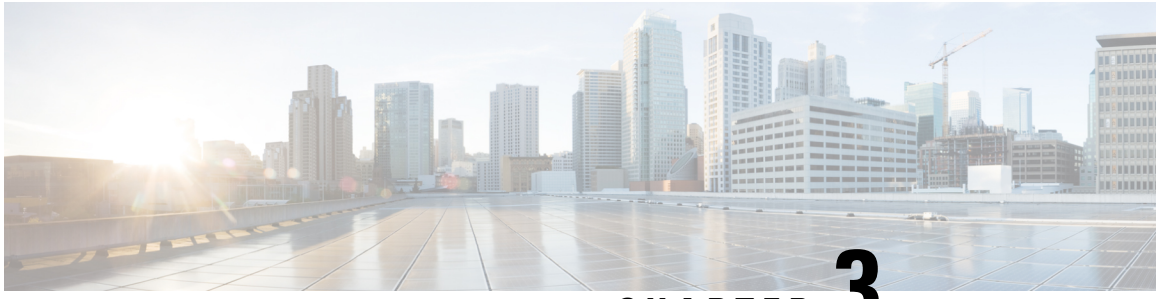
Note The procedure to create the resource definitions varies on OpenStack and VMware vCenter.

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 27.	Not applicable.



CHAPTER 3

Managing Resources on OpenStack

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>`

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>`

-	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 Multiple OpenStack VIMs".

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 11. For more information on adding and deleting networks using the ESC portal, see Managing Resources Using ESC Portal, on page 269.

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>10.60.2.0</address>
     <netmask>255.255.255.0</netmask>
     <gateway>10.60.2.1</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>10.20.0.0</address>
                <gateway>10.20.0.1</gateway>
                <netmask>255.255.255.0</netmask>
            </subnet>
        </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>10.25.90.0</address>
  <netmask>255.255.255.0</netmask>
  <gateway>10.25.90.1</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 11. For more information on adding and deleting networks using the ESC portal, see Managing Resources Using ESC Portal, on page 269.

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 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>10.20.0.0</address>
 <gateway>10.20.0.1</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 gateway.

```
<networks>
        <network>
            <name>mgmt-net</name>
            <subnet>
                <name>mgmt-net-subnet</name>
                <ipversion>ipv4</ipversion>
                <dhcp>false</dhcp>
                <address>10.20.0.0</address>
                <no_gateway>true</no_gateway><!--  DISABLE GATEWAY -->
                <gateway>10.20.0.1</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 11. For more information on adding and deleting networks using the ESC portal, see Managing Resources Using ESC Portal, on page 269.

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 11. For more information on adding and deleting flavors using the ESC portal, see Managing Resources Using ESC Portal, on page 269

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>nashrest-cirrosimage-indep</name>

<src>http://10.85.74.227:/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>nashrest-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 11. For more information on adding and deleting images using the ESC portal, see Managing Resources Using ESC Portal, on page 269.

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>
```

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.

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.

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:

```
<?xml version='1.0' encoding='ASCII'?>
<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.

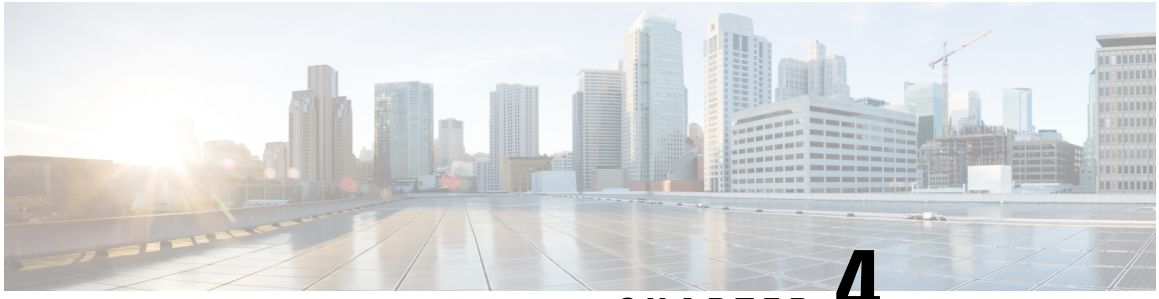
A sample deployment request with a volume created using the tenant-volume API is as follows:

```
<?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>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/>
</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 In ESC Release 2.0 and later, 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>nashrest-cirrosimage-indep</name>

<src>http://10.85.74.227:/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>nashrest-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 11. For more information on adding and deleting images using the ESC portal, see Managing VNFs Using the ESC Portal.

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:

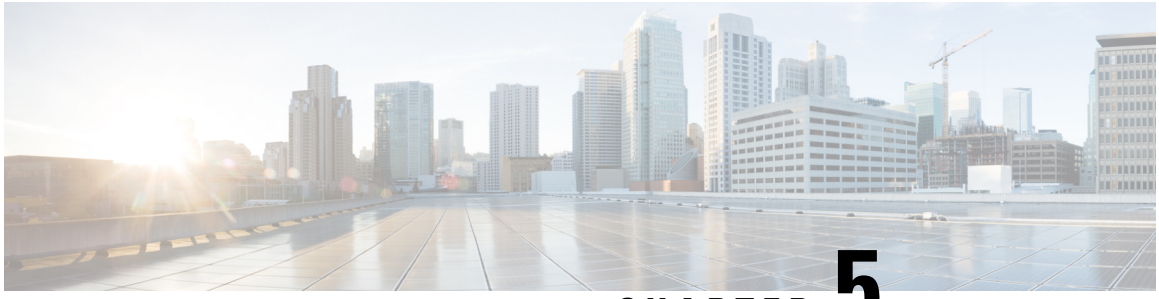
```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <name>network-portgroup-01</name>

    <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 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.

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".

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.

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 ". The VIM connectors allow multiple VIMs to connect to ESC. For more details on multi VIM deployment, see Deploying VNFs on Multiple OpenStack VIMs.

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-ucs-30</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 delete-vim-connector <vim connector id>
```

- Deleting VIM user using command:

```
$/opt/cisco/esc/esc-confd/esc-cli/esc_nc_cli 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-ucs-30</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://10.85.103.37:5000/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://10.85.103.37:5000/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-ucs-XY</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".

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-ucs-25-bad-user</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-ucs-25-bad-user</vim_connector_id>
    <event>
      <type>VIM_CONNECTION_STATE</type>
    </event>
  </escEvent>
</notification>
```

VIM Connector Configurations for OpenStack

You can configure the VIM connector for OpenStack specific operations.

Note To configure a VIM connector, see "Configuring VIM Connector".

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 2: Non-admin role permissions for ESC operations

ESC VIM Operation	Description	Permission	Note
Create Project	To create a n 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 a n 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.
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.

ESC VIM Operation	Description	Permission	Note
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.
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.

ESC VIM Operation	Description	Permission	Note
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>10.65.5.0</address> <netmask>255.255.255.0</netmask> <gateway>10.65.5.1</gateway> </subnet><shared_ip> <nicid>0</nicid> <static>false</static> </shared_ip> VM recovery also requires this privilege.
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>123.45.0.0</ip_address> <netmask>255.255.0.0</netmask> </address> <address> <ip_address>123.45.6.0</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.

ESC VIM Operation	Description	Permission	Note
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.
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/Antiaffinity	Not required	
	To create VMs in a servergroup Affinity/Antiaffinity	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 19
.

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".

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://10.85.103.153:35357/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>
      </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 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_SERCRET_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 delete-vimuser
AWS_EAST_2 AWS_ACCESS_ID

[admin@localhost ~]# esc_nc_cli 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, see Deploying VNFs on a Single or Multiple AWS Regions, on page 89.

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>
```

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-ucs-42</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-ucs-42</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>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>
          <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>
```

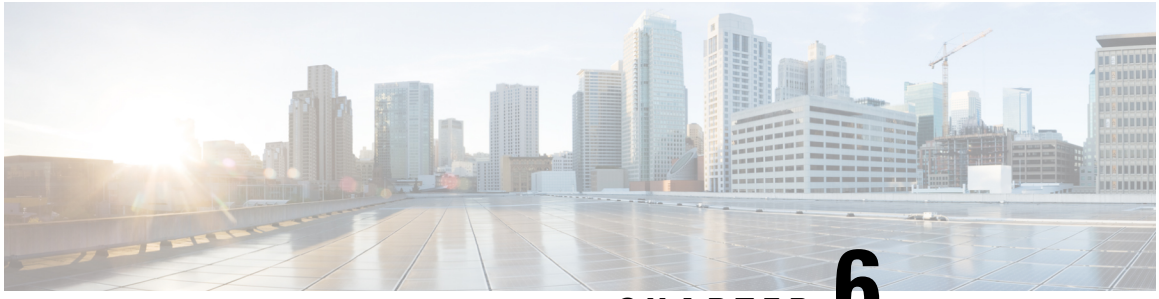
A default AES key is available in confD:

```
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.

PART III

Onboarding Virtual Network Functions



CHAPTER 6

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 15.

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 21.

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>10.101.1.0</address>
              <netmask>255.255.255.0</netmask>
              <gateway>10.101.1.1</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 97.

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/qatest/test-script.sh</file>
             </configuration>
             <configuration>
           <dst>/etc/configure-networking.sh</dst>
            <file>file://var/qatest/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 Configuring Deployment Parameters.

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 Deploying Virtual Network Functions on OpenStack.

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

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. 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 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>10.0.0.10</address>
        <netmask>255.255.255.0</netmask>
        <gateway>10.0.0.1</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>
```

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 97. 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

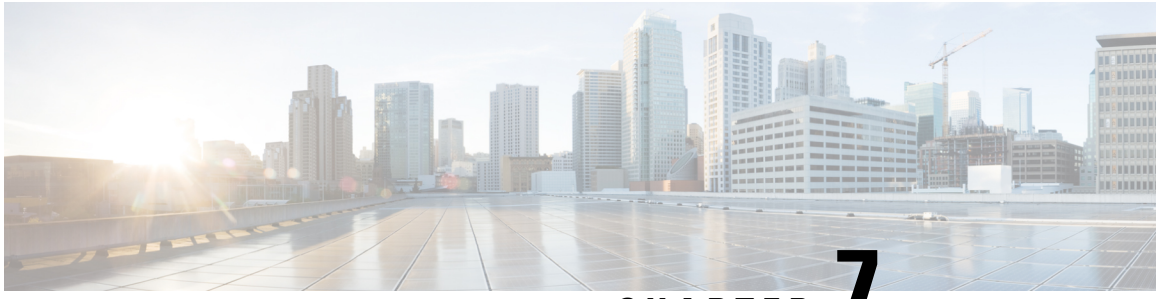
To onboard composite VNFs, you must configure some of the operational behaviors such as network connections, monitoring KPIs, placement policies, lifecycle stages, scaling rules and so on. These behaviors can be described in the deployment data model. For more information, see Configuring Deployment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

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 Deploying Virtual Network Functions on VMware vCenter.

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

PART IV

Deploying and Configuring Virtual Network Functions



CHAPTER 7

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.

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.

Scenarios	Description	Resources	Advantages
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.

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 15.

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, on page 132.

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>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_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-ucs-28
```

```
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-ucs-28
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, see "Deploying Virtual Network Functions on OpenStack".

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".

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".

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>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 />
               </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 122.

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 209. 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 151.

Deploying Virtual Network Functions on VMware vCenter

This section describes the deployment scenario for Elastic Services Controller (ESC) and the procedure to deploy VNFs on VMware. You can deploy VNFs using out-of-band image definitions. The following table lists the deployment scenarios:

Scenarios	Description	data model templates	Images	Advantages
Deploying VNFs on a single VIM 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.

Deploying VNFs on Single VMware vCenter 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 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 15. 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 Deploying VNFs on multiple VIMs is not supported on VMware vCenter.

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

- A vDS contains one or many ESXi hosts that are clustered.
- If the ESXi hosts are under one compute cluster, the VMware vCenter HA and DRS capabilities must be disabled.
- 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>OTT-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. For more information on deploying VNFs on a VDC, see .

The *default_locators* container in ESC operational data shows default locators configured in ESC.

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=127.0.0.1 --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-2_2_6_11</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>10.85.103.22</vcenter_ip>
                        <vcenter_port>80</vcenter_port>
                        <vcenter_username>root</vcenter_username>
                    </vmware_config>
                </system_config>
                <default_locators>
                    <datacenter>OTT-ESC-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-2_2_6_11 esc-cli]$
```

Deploying Virtual Network Functions on VMware vCloud Director (vCD)

This section describes the deployment scenario for Elastic Services Controller (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*.

The following resources must be created on vCD before deployment:

- **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 contains an organization, and an allocation mechanism for resources such as networks, storage, CPU, and memory.
- **Catalogs**—Catalogs contain references to vApp templates and media images.

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, the deployment is targeted to.

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>20.0.0.101</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>20.0.0.101</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>
         <!-- 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>20.0.0.102</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>20.0.0.102</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>
```

Deploying Virtual Network Functoins 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 89.

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.

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>
```

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.

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.

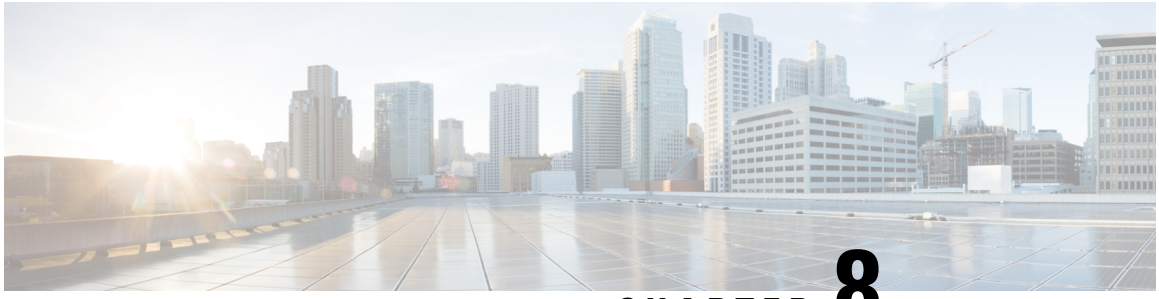
Sample undeploy request is as follows:

```
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 8

Configuring 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
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
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.)
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

Deployment Parameters	OpenStack	VMware vCenter
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.)
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.
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.
Multiple Virtual Datacenter (Multi VDC)	Not applicable	Multiple Virtual Datacenter selection is done in one of the following ways: • REST API • ESC Portal
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), on page 147.	Not applicable

Deployment Parameters	OpenStack	VMware vCenter
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

This chapter describes the procedures to configure the deployment customization. For more information on VNF deployment, see "Deploying Virtual Network Functions".

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_IP_ALLOCATION_TYPE	string containing FIXED | DHCP
NICID_n_NETWORK_ID	string containing neutron network uuid
NICID_n_IP_ADDRESS	ipv4 or ipv6 address
NICID_n_MAC_ADDRESS	string
NICID_n_GATEWAY	ipv4 or ipv6 gateway address
NICID_n_CIDR_ADDRESS	ipv4 or ipv6 cidr prefix address
NICID_n_CIDR_PREFIX	integer with prefix-length
NICID_n_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_ANYCAST_ADDRESS	string with ipv4 or ipv6
NICID_n_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_IP_ALLOCATION_TYPE: FIXED
NICID_0_NETWORK_ID: 9f8d9a97-d873-4a1c-8e95-1a123686f038
NICID_0_IP_ADDRESS: 2a00:c31:7fe2:1d:0:0:1:1000
NICID_0_MAC_ADDRESS: null
NICID_0_GATEWAY: 2a00:c31:7fe2:1d::1
NICID_0_CIDR_ADDRESS: 2a00:c31:7fe2:1d::
NICID_0_CIDR_PREFIX: 64
NICID_0_ANYCAST_ADDRESS: null
NICID_0_IPV4_OCTETS: 16.0
```

```
NICID_1_IP_ALLOCATION_TYPE: DHCP
NICID_1_NETWORK_ID: 0c468d8e-2385-4641-b1db-9080c170cb1a
NICID_1_IP_ADDRESS: 6.0.0.2
NICID_1_MAC_ADDRESS: null
NICID_1_GATEWAY: 6.0.0.1
NICID_1_CIDR_ADDRESS: 6.0.0.0
NICID_1_CIDR_PREFIX: 24
NICID_1_ANYCAST_ADDRESS: null
NICID_1_NETMASK: 255.255.255.0
```

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:

```
<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".

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 162.

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 3: 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 4: 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 5: 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

User Operation	Path	HTTP Operation	Payload	Response	Description
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 6: 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

Name	Type	Description
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, on page 103.

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,

```
<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 Master and Backup 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 `/opt/cisco/esc/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>/opt/cisco/esc/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

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"}...]}}}
```

The items are listed in the table below.

Table 7: 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.

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 209.

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 208.

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 208. For details on upgrading the VNF software upgrade policies, see Upgrading the Virtual Network Function Software Using Lifecycle Stages, on page 171.

Supported Lifecycle Stages (LCS)

Table 8:

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-ucs-42</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-ucs-42</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, "Encrypting Configuration Data".

Lifecycle Stage (LCS) Policy Conditions Defined at Different Stages

The tables below shows all policy conditions defined in the data model.

Table 9: 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
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	

Condition Name	Scope
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

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 10: 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.

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>
...
```

In ESC Release 2.0 and later, 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-ucs-4</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.

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 77.

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.

Note 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 ott-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>ott-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 are in the same deployment, but different VM groups can be explicitly deployed in different hosts. During deployment, ESC deploys the first VNF of the target VM group, and records its host to a list at the end. ESC then deploys the first VNF of each reference VM groups, ensure the VNFs are not deployed to the host in the list. Then, the second VNF of the target VM group, the second VNF of each reference VM group, and rest of the VNFs accordingly.

Example for Inter Group Anti-Affinity Policy:

```
<deployment>
<deployment>
<name>vm-groups</name>
<policies>
<placement>
<target_vm_group_ref>CSR-gp1</target_vm_group_ref>
<type>anti_affinity</type>
<vm_group_ref>CSR-gp2</vm_group_ref>
<vm_group_ref>ASA-gp1</vm_group_ref>
<enforcement>strict</enforcement>
</placement>
</policies>
```

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.

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.

```
curl  -k -X GET --header "Accept: application/xml"
"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-ucs-65</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>
```

Interface Configurations

The Interface configuration allows to choose various configuration for the interface including network, subnet, ip address, mac address, vim interface name, model, and so on. This section describes these basic and advance interface configurations for Elastic Services Controller (ESC) and procedures to configure these.

Basic Interface Configurations

In ESC Datamodel, Interface refers to the VNIC attached to the VM. We can add one or more Interface under a VM Group. The interface section will have details to configure the VNIC. This section describes basic interface configurations for Elastic Services Controller (ESC).

Configuring Basic Interface Settings

This section describes basic interface configurations, such as Network, Subnet, IP address, MAC address, VIM interface name, and so on for Elastic Services Controller (ESC).

Configuring an Interface Name

To configure VIM interface name, specify attribute `<vm_interface_name>` for an interface in the Deployment XML file. Use `<vm_interface_name>` to use a specific name when generating an interface name. If these

attribute is not specified, ESC will auto-generate an interface name, which is a combination of the deployment_name, group_name, and a random UUID string. For example: my-deployment-na_my-gro_0_8053d7gf-hyt33-4676-h9d4-9j4a5599472t.

Note This feature is currently supported only on OpenStack.

If the VM group is elastic and a `vim_interface_name` has been specified, a numeric index is added after the interface name for the second interface name onwards (the first one remains unchanged). For example, if the specified interface name is set as `<vim_interface_name>interface_1</vim_interface_name>` and scaling is set to 3, three VMs are created with three different interface name, interface_1, interface_1_1, and interface_1_2. If a VM group only has a single VM, then there is no "_<index>" appended to the custom interface name. A single deployment can contain multiple VM groups, and each individual VM group can specify a different `vim_interface_name` value, if required. For example, a deployment could have two VM groups: the first group specifies a `vim_interface_name` and all VMs have their names generated as described above. The second VM group does not specify a `vim_interface_name`, therefore all VM names created from this group are auto generated. The same interface name can be used in separate interface sections within the same VM group, or in separate VM groups within a deployment, or in different deployments if required.

If attributes `<vim_interface_name>` or `<port>` are used for the same interface, the `vim_interface_name` value will be ignored and the value in the `port` attribute will be used.

```
<esc_datamodel xmlns="http://www.cisco.com/esc/esc"> <tenants><tenant>
<name>Admin</name>
<deployments>
<deployment>
<deployment_name>NwDepModel_nosvc</deployment_name>
<interface>
<nicid>0</nicid>
<vim_interface_name>interface_1</vim_interface_name>
<network>esc-net</network>
</interface>
```

Note You can use a maximum of 61 characters for an interface name should not contain special characters and can only contain alphanumeric characters and "_" and "-". The following are some output samples with the custom port name. If the `vim_interface_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 port name.

- Below is an example of the output operational data fetched using the esc_nc_cli script after adding a custom interface name. A new element called `vim_interface_name` will be shown under the interface element.

```
[admin@esc-3-1-xxx]$ esc_nc_cli get 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">
. . .
    <interface>
        <nicid>0</nicid>
        <type>virtual</type>
        <port_id>e4111069-5d00-493b-8ea9-1a2ca134b5c8</port_id>
        <vim_interface_name>interface_1</vim_interface_name>      <!-- NEW IN OUTPUT
-->
        <network>c7fafeca-aa53-4349-9b60-1f4b92605420</network>
        <subnet>255.255.255.0</subnet>
        <ip_address>192.0.2.1</ip_address>
        <mac_address>fa:16:3e:d7:5e:da</mac_address>
        <netmask>255.255.240.0</netmask>
        <gateway>192.0.2.255</gateway>
    </interface>
```

- Below is an example output operational data fetched using a REST API.

```
curl  -k -X GET --header "Accept: application/xml"
"http://localhost:8080/ESCManager/v0/deployments/example-deployment-123"
| xmllint --format -
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<deployments>
. . .
    <interface>
        <network_uuid>c7fafeca-aa53-4349-9b60-1f4b92605420</network_uuid>
        <gateway>152.16.0.1</gateway>
        <ip_address>152.16.12.251</ip_address>
        <mac_address>fa:16:3e:30:0c:99</mac_address>
        <netmask>255.255.240.0</netmask>
        <nic_id>0</nic_id>
        <port_forwarding/>
        <port_uuid>1773cdbf-fe5f-4af1-adff-3a9c1dd1c47d</port_uuid>
       <vim_interface_name>interface_1</vim_interface_name>        <!-- NEW IN OUTPUT
 -->
        <security_groups/>
        <subnet_uuid>7b2ce63b-eb20-4ff8-8d49-e46ee8dde0f5</subnet_uuid>
        <type>virtual</type>
    </interface>
```

In all the above scenarios, if `vim_interface_name` is not specified in the deployment.xml, the output will still contain this element, however with an internally generated interface name. For example:

```
<vim_interface_name>vm-name-deployme_Grp1_1_0f24cd7e-cae7-402e-819a-5c84087103ba</vim_interface_name>
```

Assigning the MAC Address

ESC deployment on VMware vCenter supports assigning MAC address using the MAC address range, or MAC address list from the MAC address pool to deploy VMs to the network.

You can assign MAC address in the following ways:

Using the Interface

```
<interfaces>
    <interface>
 <nicid>1</nicid>
 <network>MANAGEMENT_NETWORK</network>
 <ip_address>10.88.0.11</ip_address>
 <mac_address>fa:16:3e:73:19:a0</mac_address>
    </interface>
</interfaces>
```

During scaling, you can assign the MAC address list or MAC address range from the MAC address pool.

```
<scaling>
 <min_active>2</min_active>
 <max_active>2</max_active>
 <elastic>true</elastic>
 <static_ip_address_pool>
  <network>MANAGEMENT_NETWORK</network>
  <ip_address>10.88.0.11</ip_address>
  <ip_address>10.88.0.12</ip_address>
  <ip_address>10.88.0.13</ip_address>
 </static_ip_address_pool>
 <static_mac_address_pool>
  <network>MANAGEMENT_NETWORK</network>
  <mac_address>fa:16:3e:73:19:a0</mac_address>
  <mac_address>fa:16:3e:73:19:a1</mac_address>
  <mac_address>fa:16:3e:73:19:a2</mac_address>
 </static_mac_address_pool>
</scaling>
```

Assign MAC address using MAC address range.

```
<scaling>
 <min_active>2</min_active>
 <max_active>2</max_active>
 <elastic>true</elastic>
 <static_ip_address_pool>
  <network>MANAGEMENT_NETWORK</network>
  <ip_address_range>
   <start>10.88.0.25</start>
   <end>10.88.0.27</end>
  </ip_address_range>
 </static_ip_address_pool>
 <static_mac_address_pool>
  <network>MANAGEMENT_NETWORK</network>
  <mac_address_range>
   <start>fa:16:3e:73:19:b0</start>
   <end>fa:16:3e:73:19:b2</end>
  </mac_address_range>
 </static_mac_address_pool>
</scaling>
```

Note You cannot change the MAC or IP pool in an existing deployment, or during scaling (when min and max value are greater than 1) of VM instances in a service update.

In VMware vCenter, while assigning the MAC address, the server might override the specified value for "Generated" or "Assigned" if it does not fall in the right ranges or is determined to be a duplicate. Because of this, if ESC is unable to assign the MAC address the deployment fails.

Configuring Subnet for an Interface

Subnets can be passed through the datamodel. Subnet within interfaces can be specified in the Interface section of the Deployment XML file. If there is no subnet specified in the datamodel, ESC will let OpenStack select the subnet for interface creation and will use the subnet from the port created by OpenStack.

```
<interface>
   <nicid>0</nicid>
   <network>esc-net</network>
  <subnet>esc-subnet</subnet>
    </interface>
```

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>10.20.0.0</address>
<no_gateway>true</no_gateway><!-- DISABLE GATEWAY -->
<gateway>10.20.0.1</gateway>
<netmask>255.255.255.0</netmask>
</subnet>
</network>
</networks>
```

Configuring an Out-of-Band Port

ESC also allows you to attach an out-of-band port to a VNF. To do this, pass the UUID or the name of the port in the deployment request file while initiating a service request.

Note While undeploying or restoring a VNF, the ports attached to that VNF will only be detached and not deleted.

ESC does not allow scaling while using OOB port for a VM group. You can configure only one instance of VM for the VM group.

Updating the scaling value for a VM group, while using the out of band port is not allowd during a deployment update.

```
<esc_datamodel xmlns="http://www.cisco.com/esc/esc">
    <name>tenant</name>
    <deployments>
        <deployment>
```

```
            <name>depz</name>
            <vm_group>
                <name>g1</name>
                <image>Automation-Cirros-Image</image>
                <flavor>Automation-Cirros-Flavor</flavor>
                <bootup_time>100</bootup_time>
                <reboot_time>30</reboot_time>
                <recovery_wait_time>10</recovery_wait_time>
                <interfaces>
                    <interface>
                        <nicid>0</nicid>
                        <port>057a1c22-722e-44da-845b-a193e02807f7</port>
                        <network>esc-net</network>
                    </interface>
                </interfaces>
            </vm_group>
        </deployment>
    </deployments>
</esc_datamodel>
```

IPv6 Support

ESC supports end-to-end IPv6 support for OpenStack deployments.

Dual Stack Support

A dual stack network allows you to assign multiple IP addresses. These multiple IP addresses can be assigned on different subnets to a given interface within a VNF deployment using ESC.

ESC supports the following for dual stack:

- Configuring the network and list of subnet
- Configuring the network and list of subnet and ip address
- Configuring the network and list of ip address (no subnet)
- Specifying the network and list of subnet/ip (same subnet but different ip)

Note Currently, ESC supports dual stack only on OpenStack.

A new container element named addresses is added to the Interface. This container holds a list of address elements. An address element must have an address_id (key). The subnet and fixed-ip address fields are optional, but you must specify either one.

The container address is as follows:

```
container addresses {
  list address {
    key "address_id";
    leaf address_id {
      description "Id for the address in address list.";
      type uint16;
      mandatory true;
    }
    leaf subnet {
      description "Subnet name or uuid for allocating IP to this port";
      type types:escnetname;
```

```
    }
    leaf ip_address {
      description "Static IP address for this specific subnet";
      type types:escipaddr;
      must "../../../../../scaling/max_active = 1"
      {
         error-message "Only single VM per group supported with multiple address option.";
      }
    }
  }
}
```

Dual stack now supports KPI monitoring. A new child element address_id has been added to the metric_collector element. This accepts a value which points to an address within the specified nicid to be used for KPI monitoring. That is, it allows one of the addresses defined beneath an interface to be used for KPI monitoring.

```
…
    <interface>
        <nicid>1</nicid>
        <network>demo-net</network>
        <addresses>
            <address>
                <address_id>0</address_id>
                <subnet>demo-subnet</subnet>
            </address>
        </addresses>
    </interface>
 <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>5</metric_occurrences_true>
               <metric_occurrences_false>5</metric_occurrences_false>
              <metric_collector>
                  <type>ICMPPing</type>
                  <nicid>1</nicid>
                  <address_id>0</address_id>
                  <poll_frequency>10</poll_frequency>
                  <polling_unit>seconds</polling_unit>
                 <continuous_alarm>false</continuous_alarm>
                </metric_collector>
              </kpi>
            </kpi_data>
...
```

Note The address_id under the metric_collector element must be the same as one of the address_id beneath the interface.

Dual stack interfaces can now be used in day-0 variable substitution. This means the ability to substitute the values from the multiple addresses defined under a single interface. Day 0 configuration is defined in the datamodel under the config_data tag.

In case of dual stack with multiple IP addresses, the variables are in the form NICID_<n>_<a>_<PROPERTY> where:

- <n> is the nicid for the interface.

- <a> is the address_id of an address within that interface.

The list of possible day-0 substitution variables from dual stack is:

NICID_n_a_IP_ALLOCATION_TYPE	string containing FIXED | DHCP	ipv4 or ipv6
NICID_n_a_IP_ADDRESS	IP address	ipv4 or ipv6
NICID_n_a_GATEWAY	Gateway address	ipv4 or ipv6
NICID_n_a_CIDR_ADDRESS	CIDR prefix address	ipv4 or ipv6
NICID_n_a_CIDR_PREFIX	Integer with CIDR prefix-length	ipv4 or ipv6
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.	ipv4 only

For information on day-0 confirguration for single IP address, see Day Zero Configuration, on page 97.

The template file defined in the config_data with day-0 configurations is as follows:

```
NICID_0_NETWORK_ID=${NICID_0_NETWORK_ID}
NICID_0_MAC_ADDRESS=${NICID_0_MAC_ADDRESS}

NICID_0_0_IP_ALLOCATION_TYPE=${NICID_0_0_IP_ALLOCATION_TYPE}
NICID_0_0_IP_ADDRESS=${NICID_0_0_IP_ADDRESS}
NICID_0_0_GATEWAY=${NICID_0_0_GATEWAY}
NICID_0_0_CIDR_ADDRESS=${NICID_0_0_CIDR_ADDRESS}
NICID_0_0_CIDR_PREFIX=${NICID_0_0_CIDR_PREFIX}
NICID_0_0_NETMASK=${NICID_0_0_NETMASK}

NICID_0_1_IP_ALLOCATION_TYPE=${NICID_0_1_IP_ALLOCATION_TYPE}
NICID_0_1_IP_ADDRESS=${NICID_0_1_IP_ADDRESS}
NICID_0_1_GATEWAY=${NICID_0_1_GATEWAY}
NICID_0_1_CIDR_ADDRESS=${NICID_0_1_CIDR_ADDRESS}
NICID_0_1_CIDR_PREFIX=${NICID_0_1_CIDR_PREFIX}
```

The datamodel is as follows:

```
<?xml version="1.0" encoding="ASCII"?>
<esc_datamodel xmlns="http://www.cisco.com/esc/esc">
  <tenants>
    <tenant>
      <name>dep-tenant</name>
      <deployments>
        <deployment>
          <name>cirros-dep</name>
          <vm_group>
            <name>Grp1</name>
            <bootup_time>600</bootup_time>
            <recovery_wait_time>30</recovery_wait_time>
            <flavor>Automation-Cirros-Flavor</flavor>
            <image>Automation-Cirros-Image</image>
           <interfaces>
              <interface>
                <!-- No dual stack support on mgmt interface in ESC 4.1 -->
```

```
  <nicid>0</nicid>
  <network>esc-net</network>
</interface>
<interface>
  <nicid>1</nicid>
  <network>ent-network1</network>
  <addresses>
    <address>
      <!-- IPv4 Dynamic -->
      <address_id>0</address_id>
      <subnet>v4-subnet_A</subnet>
    </address>
    <address>
      <!-- IPv6 Dynamic -->
      <address_id>1</address_id>
      <subnet>v6-subnet_B</subnet>
    </address>
  </addresses>
</interface>
<interface>
  <nicid>2</nicid>
  <network>ent-network2</network>
  <addresses>
    <address>
      <!-- IPv4 Static -->
      <address_id>0</address_id>
      <subnet>v4-subnet_C</subnet>
      <ip_address>10.87.87.8</ip_address>
    </address>
    <address>
      <!-- IPv6 Static -->
      <address_id>1</address_id>
      <subnet>v6-subnet_D</subnet>
      <ip_address>fd07::110</ip_address>
    </address>
  </addresses>
</interface>
<interface>
  <nicid>3</nicid>
  <network>ent-network3</network>
  <addresses>
    <address>
      <!--  Only ip config -  ipv6 but no subnet  -->
      <address_id>0</address_id>
      <ip_address>fd07::110</ip_address>
    </address>
    <address>
      <!--  Only ip config - ipv4 but no subnet  -->
      <address_id>1</address_id>
      <ip_address>10.87.88.9</ip_address>
    </address>
  </addresses>
</interface>
<interface>
  <nicid>4</nicid>
  <network>ent-network4</network>
  <addresses>
    <address>
      <!-- ipv4 same subnet as address_id 6 -->
      <address_id>0</address_id>      //
      <subnet>v4-subnet_F</subnet>
      <ip_address>10.87.86.10</ip_address>
    </address>
    <address>
```

```
                <!-- ipv4 same subnet as id 5 -->
                <address_id>1</address_id>
                <subnet>v4-subnet_F</subnet>
                <ip_address>10.87.86.11</ip_address>
              </address>
            </addresses>
          </interface>
        </interfaces>
        <kpi_data>
...
```

After successful deployment using multiple IPs, ESC provides a list of addresses as notification, or opdata.

A list of multiple <address> elements under the parent <interface> element containing the following:

- **address_id**—the address id specified in the input XML
- **subnet element**—subnet name or uuid
- **ip_address element**—the port's assigned IP on that subnet
- **prefix**—the subnet CIDR prefix
- **gateway**—the subnet gateway address
- ESC Static IP support

Notification:

```
              <vm_id>1834124d-b70b-41b9-9e53-fb55d7c901f0</vm_id>
              <name>jenkins-gr_g1_0_e8bc9a81-4b9a-437a-807a-f1a9bbc2ea3e</name>

<generated_name>jenkins-gr_g1_0_e8bc9a81-4b9a-437a-807a-f1a9bbc2ea3e</generated_name>
            <host_id>dc380f1721255e2a7ea15932c1a7abc681816642f75276c166b4fe50</host_id>

              <hostname>my-ucs-50</hostname>
              <interfaces>
                <interface>
                  <nicid>0</nicid>
                  <type>virtual</type>

<vim_interface_name>jenkins-gr_g1_0_e8bc9a81-4b9a-437a-807a-f1a9bbc2ea3e</vim_interface_name>

                  <port_id>4d57d4a5-3150-455a-ad39-c32fffbb10b1</port_id>
                  <mac_address>fa:16:3e:d2:50:a5</mac_address>
                  <network>45638651-2e92-45fb-96ce-9efdd9ea343e</network>
                  <address>
                    <address_id>0<address_id>
                    <subnet>6ac36430-4f58-454b-9dc1-82f7a796e2ff</subnet>
                    <ip_address>142.18.0.22</ip_address>
                    <prefix>24</prefix>
                    <gateway>142.18.0.1</gateway>
                  </address>
                  <address>
                    <address_id>1<address_id>
                    <subnet>8dd9f501-19d4-4782-8335-9aa9fbd4dab9</subnet>
                    <ip_address>2002:dc7::4</ip_address>
                    <prefix>48</prefix>
                    <gateway>2002:dc7::1</gateway>
                  </address>
                  <address>
                    <address_id>2<address_id>
                    <subnet>a234501-19d4-4782-8335-9aa9fbd4caf6</subnet>
```

```
                        <ip_address>10.87.87.8</ip_address>
                        <prefix>20</prefix>
                        <gateway>10.87.87.1</gateway>
                      </address>
                    </interface>
```

Sample opdata:

```
  <interfaces>
   <interface>
      <nicid>0</nicid>
      <type>virtual</type>

<vim_interface_name>jenkins-gr_g1_0_e8bc9a81-4b9a-437a-807a-f1a9bbc2ea3e</vim_interface_name>

      <port_id>4d57d4a5-3150-455a-ad39-c32fffbb10b1</port_id>
      <mac_address>fa:16:3e:d2:50:a5</mac_address>
      <network>45638651-2e92-45fb-96ce-9efdd9ea343e</network>
      <address>
         <address_id>0</address_id>
         <subnet>6ac36430-4f58-454b-9dc1-82f7a796e2ff</subnet>
         <ip_address>142.18.0.22</ip_address>
         <prefix>24</prefix>
         <gateway>142.18.0.1</gateway>
      </address>
      <address>
         <address_id>1</address_id>
         <subnet>8dd9f501-19d4-4782-8335-9aa9fbd4dab9</subnet>
         <ip_address>2002:dc7::4</ip_address>
         <prefix>48</prefix>
         <gateway>2002:dc7::1</gateway>
      </address>
   </interface>
</interfaces>
```

You can also see that the day-0 substitution values are replaced in the output data. Sample output data with the values populated in the day-0 configuration is as follows:

```
NICID_0_NETWORK_ID=45638651-2e92-45fb-96ce-9efdd9ea343e
NICID_0_MAC_ADDRESS=fa:16:3e:d2:50:a5

NICID_0_0_IP_ALLOCATION_TYPE=DHCP
NICID_0_0_IP_ADDRESS=142.18.0.22
NICID_0_0_GATEWAY=142.18.0.1
NICID_0_0_CIDR_ADDRESS=142.18.0.0
NICID_0_0_CIDR_PREFIX=24
NICID_0_0_NETMASK=255.255.255.0

NICID_0_1_IP_ALLOCATION_TYPE=DHCP
NICID_0_1_IP_ADDRESS=2002:dc7::4
NICID_0_1_GATEWAY=2002:dc7::1
NICID_0_1_CIDR_ADDRESS=2002:dc7::/48
NICID_0_1_CIDR_PREFIX=48
```

Dual Stack with Static IP Support

ESC supports dual stack with static IP support. As part of the initial configuration the user can provide the subnet and IP to be configured.

Note ESC supports static IP only when the scaling is false or minimum /maximum =1.

When you create a VM with out-of-band network, and specify a list of subnets with static IP (the network has multiple subnets), then ESC applies both subnet and the corresponding static IP.

In the example below, two subnets (ipv4 and ipv6) are added to a single interface.

```
  <?xml version="1.0" encoding="ASCII"?>
<esc_datamodel xmlns="http://www.cisco.com/esc/esc">
  <tenants>
    <tenant>
      <name>dep-tenant</name>
      <deployments>
        <deployment>
          <name>cirros-dep</name>
          <vm_group>
            <name>Grp1</name>
            <bootup_time>600</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>ent-network2</network>
                <addresses>
                  <address>
                    <!-- IPv4 Static -->
                    <address_id>0</address_id>
                    <subnet>v4-subnet_C</subnet>
                    <ip_address>10.87.87.8</ip_address>
                  </address>
                  <address>
                    <!-- IPv6 Static -->
                    <address_id>1</address_id>
                    <subnet>v6-subnet_D</subnet>
                    <ip_address>fd07::110</ip_address>
                  </address>
                </addresses>
              </interface>
            </interfaces>
          <kpi_data>
```

For information on deploying VNFs, see Deploying Virtual Network Functions on OpenStack.

Advanced Interface Configurations

This section describes several interface configurations for Elastic Services Controller (ESC) and the procedure to configure the hardware interfaces.

Configuring Advance Interface Settings

Configuring SR-IOV in ESC

Single Root I/O Virtualization (SR-IOV) allows multiple VMs running a variety of guest operating systems to share a single PCIe network adapter within a host server. It also allows a VM to move data directly to and

from the network adapter, bypassing the hypervisor for increased network throughput and lower server CPU burden.

Configuring SR-IOV in ESC for OpenStack

Before you configure SR-IOV in ESC for OpenStack, configure the hardware and OpenStack with the correct parameters.

To enable SR-IOV in ESC for OpenStack, specify the interface `type` as `direct`. The following snippet shows a sample datamodel:

```
<interfaces>
 <interface>
  <nicid>0</nicid>
  <network>esc-net</network>
  <type>direct</type>
  </interface>
</interfaces>
...
```

Configuring SR-IOV in ESC for VMware

Before you configure SR-IOV in ESC for VMware, consider the following:

- Enable SR-IOV Physical Functions on desired ESXi hosts. For more information, see VMware documentation.
- Consider the following important points before enabling SR-IOV:
  - Review the list of physical network adaptors that VMware supports for SR-IOV. See VMware documentation.
  - Review the list of VM features that are not supported on a VM with SR-IOV configured. See VMware documentation.
  - In a cluster deployment (defined by "zone" in the datamodel) with SR-IOV, make sure that each ESXi host has identical Physical Functions enabled for SR-IOV selection. For example, if a VM is going to use vmnic7 as the Physical Function, make sure that each host has vmnic7 and SR-IOV status for each vmnic7 is enabled.

    To enable SR-IOV in ESC for VMware, specify interface`<type>` as `direct` and also extension `<name>` as `sriov_pf_selection` in the deployment datamodel. Interface Type `direct`indicates an SR-IOV device and extension name `sriov_pf_selection` indicates the physical function. The following snippet shows a sample datamodel:

```
<vm_group>
...
<interface>
  <nicid>2</nicid>
  <network>MgtNetwork</network>
  <type>direct</type>
</interface>
<interface>
  <nicid>3</nicid>
  <network>MgtNetwork</network>
  <type>direct</type>
</interface>
...
<extensions>
 <extension>
 <name>sriov_pf_selection</name>
```

```
<properties>
<property>
<name>nicid-2</name>
<value>vmnic1,vmnic2</value>
</property>
<property>
<name>nicid-3</name>
<value>vmnic3,vmnic4</value>
</property>
</properties>
</extension>
</extensions>
</vm_group>
```

Configuring Allowed Address Pair

Cisco Elastic Services Controller allows you to specify the address pairs in the deployment datamodel to pass through a specified port regardless of the subnet associated with the network.

The address pair is configured in the following ways:

- List of Network—When a list of network is provided on a particular interface, ESC will get the subnet details from the OpenStack for these networks and add them to the corresponding port or interface. The following example explains how to configure address pairs as a list of network:

```
<interface>
  <nicid>1</nicid>
  <network>network1</network>
  <allowed_address_pairs>
  <network>
  <name>bb8c5cfb-921c-46ea-a95d-59feda61cac1</name>
  </network>
  <network>
  <name>6ae017d0-50c3-4225-be10-30e4e5c5e8e3</name>
  </network>
  </allowed_address_pairs>
  </interface>
</interfaces>
```

- List of Address— When a list of address is provided, ESC will add these addresses to the corresponding interface. The following example explains how to configure address pairs as a list of address:

```
<interface>
  <nicid>0</nicid>
  <network>esc-net</network>
  <allowed_address_pairs>
  <address>
  <ip_address>10.10.10.10</ip_address>
  <netmask>255.255.255.0</netmask>
  </address>
  <address>
  <ip_address>10.10.20.10</ip_address>
  <netmask>255.255.255.0</netmask>
  </address>
  </allowed_address_pairs>
</interface>
```

Configuring Port Security

The attribute `port_security_enabled` in the deployment datamodel allows you to add or modify the port or interface security status. You can specify this parameter during an initial VNF deployment or also while modifying an existing VNF deployment. Setting this parameter as 'true' will enable the port security. If the value is set to 'false', ESC will not allow you to configure Security Groups and the Allowed Address Pairs for the VNF instances. You can also configure the port security parameter through both REST and NETCONF .

Note Enable the port security on the OpenStack before enabling it in Cisco Elastic Services Controller.

```
<esc_datamodel xmlns="http://www.cisco.com/esc/esc"> <tenants><tenant>
  <vm_group>
<interfaces>
<interface>
<nicid>1</nicid>
<network>Net-1</network>
<port_security_enabled>false</port_security_enabled>
</interface>
</interfaces>
</vm_group>
```

Configuring Security Group Rules

Cisco Elastic Services Controller (ESC) allows you to associate security group rules to the deployed instances on OpenStack. These security group rules are configured by specifying the necessary parameters in the deployment datamodel. In addition to configuring security group rules, if any VNF instance fails, ESC recovers the instance and applies the security group rules for the redeployed VNF.

To configure security group rules, do the following:

Before you begin

- Make sure you have created a tenant through ESC.
- Make sure you have security groups created.
- Make sure you have the security group name or UUID.

Procedure

Step 1 Log in to the ESC VM as a root user.

Step 2 Run the following command to check the UUIDs of a given security group:

```
nova --os-tenant-name <NameOfTheTenant> secgroup-list
```

Step 3 Pass the following arguments in the deployment data model:

```
<interfaces>
 <interface>
  <nicid>0</nicid>
  <network>esc-net</network><!-- depends on network name -->
  <security_groups>
```

```
  <security_group>0c703474-2692-4e84-94b9-c29e439848b8</security_group>
  <security_group>bbcdbc62-a0de-4475-b258-740bfd33861b</security_group>
  </security_groups>
</interface>
<interface>
 <nicid>1</nicid>
 <network>sample_VmGrpNet</network><!--depends on network name  -->
 <security_groups>
 <security_group>sample_test_SQL</security_group>
 </security_groups>
</interface>
```

Step 4 Run the following command to verify whether the security groups are associated with the VM instance:

```
nova --os-tenant-name <NameOfTenant> show <NameOfVMinstance>
```

Hardware Acceleration Support (OpenStack Only)

You can configure hardware acceleration features on OpenStack using the *flavor data model*. The following hardware acceleration features can be configured:

- **vCPU Pinning**—enables binding and unbinding of a process to a vCPU (Virtual Central Processing Unit) or a range of CPUs, so that the process executes only on the designated CPU or CPUs rather than any CPU.
- **OpenStack performance optimization for large pages and non-uniform memory access (NUMA)**—enables improvement of system performance for large pages and NUMA i.e., system's ability to accept higher load and modify the system to handle a higher load.
- **OpenStack support for PCIe Passthrough interface**—enables assigning a PCI device to an instance on OpenStack.

The following example explains how to configure hardware acceleration features using *flavor data model*:

```
$ cat fl.xml
<?xml version='1.0' encoding='ASCII'?>
<esc_datamodel xmlns="http://www.cisco.com/esc/esc">
  <flavors>
    <flavor>
      <name>testfl6</name>
      <vcpus>1</vcpus>
      <memory_mb>2048</memory_mb>
      <root_disk_mb>10240</root_disk_mb>
      <ephemeral_disk_mb>0</ephemeral_disk_mb>
      <swap_disk_mb>0</swap_disk_mb>
      <properties>
        <property>
          <name>pci_passthrough:alias</name>
          <value>nic1g:1</value>
        </property>
      </properties>
    </flavor>
  </flavors>
</esc_datamodel>
$ /opt/cisco/esc/esc-confd/esc-cli/esc_nc_cli edit-config ./fl.xml
```

Configuring PCI or PCIe Device Passthrough on VMware vCenter

ESC supports VMware vCenter PCI or PCIe device passthrough (VMDirectPath I/O). This enables VM access to physical PCI functions on platforms with an I/O memory management unit.

Before You Begin

For the PCI / PCIe devices of a host VM to enable passthrough, the vSphere administrator must mark these devices in the vCenter.

Note You must reboot the host after PCI settings. Put the host to maintenance mode, power off or migrate all VMs to other hosts.

To specify PCI device passthrough request in ESC deployments include the <type> attribute with value set to ***passthru***. The data model is as follows:

```
<tenants>
  <tenant>
   <name>admin</name>
   <deployments>
    <deployment>
     <name>test</name>

     <vm_group>
      <name>test-g1</name>
      <image>uLinux</image>
      <bootup_time>300</bootup_time>
      <recovery_wait_time>10</recovery_wait_time>
      <interfaces>
       <interface>
        <nicid>1</nicid>
        <network>MgtNetwork</network>
        <ip_address>10.79.0.102</ip_address>
       </interface>
       <interface>
        <nicid>2</nicid>
        <network>VM Network</network>
        <type>passthru</type>
        <ip_address>172.16.0.0</ip_address>
       </interface>
       <interface>
        <nicid>3</nicid>
        <network>VM Network</network>
        <type>passthru</type>
        <ip_address>10.84.46.117</ip_address>
       </interface>
      </interfaces>
```

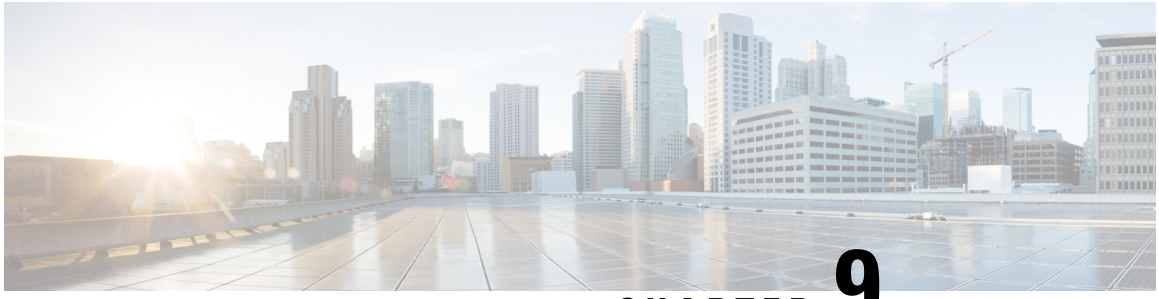
After successful deployment, the ***passthru*** value is set in the interface section of the notification as well as in the operational data.

Auto Selecting PCI or PCIe PassThrough Device

ESC needs one or more PCI or PCIe passthrough devices to be attached to each deployment without a particular PCI ID. ESC first selects a host. ESC selects the next available PCI or PCIe passthrough enabled device and attaches it during the deployment. If there is no PCI or PCIe passthrough enabled device available, ESC fails the deployment. The vSphere administrator has to ensure all computing-host within the target computing-cluster have enough number of PCI or PCIe passthrough enabled devices.

Note

- PCI or PCIe passthrough is not considered by ESC placement algorithm. For example, ESC does not select a host because it has available resources to complete the PCI or PCIe passthrough requests.
- ESC selects the PCI or PCIe passthrough device randomly. ESC does not consider the type or specification of the device. It selects the next available PCI or PCIe device from the list.
- Recovery fails if the VNF is recovered to a computing-host that ESC has selected based on the ESC placement algorithm, and if that computing-host does not have any PCI or PCIe passthrough enabled devices available.
- DRS must be turned off for the passthrough to work.



CHAPTER 9

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, on page 151
- Upgrading the Virtual Network Function Software Using Lifecycle Stages, on page 171

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.

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 11: Updating an Existing Deployment on OpenStack and VMware vCenter

Update	OpenStack	VMware vCenter
Adding a VM group	Supported	Supported
Deleting a VM group	Supported	Supported
Deleting VM groups when the service is in error state	Supported	Supported
Adding an ephemeral network	Supported	Not supported
Deleting an ephemeral network	Supported	Not supported

Update	OpenStack	VMware vCenter
Adding an interface	Supported	Not supported
Deleting an interface	Supported	Not supported
Updating an interface	Supported	Supported
Adding a Static IP pool	Supported	Not supported
Deleting a Static IP pool	Supported	Not supported
Updating the day-0 config in a VM group	Supported	Supported
Updating the KPIs and rules	Supported	Supported
Updating the number of VMs (Scale In or Scale Out) in a VM group	Supported	Supported
Updating the recovery wait time	Supported	Supported
Updating the recovery policy	Supported	Not supported
Updating an image	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>esc-net</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>esc-net</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>esc-net</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-ucs-3</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.
- Adding and deleting interfaces is not supported using ESC Portal.
- Adding and deleting interfaces is supported using NETCONF only.

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>10.85.103.14</hostname>
      <interfaces>
        <interface>
          <nicid>1</nicid>
          <type>virtual</type>
          <port_id/>
          <network>VNFNetwork</network>
          <subnet/>
          <ip_address>16.0.0.254</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>
```

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>10.66.5.13</start>
<end>10.66.5.13</end>
</ip_address_range>
</static_ip_address_pool>
<static_ip_address_pool>
<network>IP-pool-network-B</network>
<ip_address_range>
<start>10.66.7.13</start>
<end>10.66.7.13</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>10.66.5.13</start>
<end>10.66.5.13</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>10.66.7.13</start>
```

```
<end>10.66.7.13</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
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

Request	Supported or Rejected
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://10.60.73.167:4343/day0/cfg/vWSA/node/001-wsa/provider/Symphony_VNF_P-1B/file>

  </configuration>
  <configuration>
    <dst>license.txt</dst>

<file>https://10.60.73.167:4343/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://10.60.73.167:4343/day0/cfg/vWSA/node/001-wsa/provider/Symphony_VNF_P-1B/file>

  </configuration>
  <configuration>
    <dst>license.txt</dst>

<file>https://10.60.73.167:4343/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 97.

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>23.23.23.23</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>23.23.23.23</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>23.23.23.23</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 12: 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.
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.

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 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.
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>23.23.23.23</ip_address>
             <ip_address>23.23.23.24</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 23.23.23.24.

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	

Rollback condition	Expected behavior	Notification
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	
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>

Rollback condition	Expected behavior	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.	

Upgrading the Virtual Network Function Software Using Lifecycle Stages

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.

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:

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<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

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 116. 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>
```

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-ucs-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-ucs-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-ucs-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-ucs-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
        }
    }
}
```

CHAPTER 10

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 179
- Event Notifications or Callback Events, on page 181

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 13: 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 14: 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 15: 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.
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.
VM_MANUAL_RECOVERY_NEEDED	Manual Recovery	The manual recovery needed notification is sent when a recovery is triggered but manual recovery policy is configured.

Event State	Workflow	Description
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 16: 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.
SERVICE_STOPPED	Deployment Action	The service stopped event is sent when a request to stop all the VM\VNF in a service is completed.

Event State	Workflow	Description
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 17: 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
DELETE_FLAVOR	Flavor	Flavor deleted

CHAPTER 11

Virtual Network Function Operations

VNF Operations

You can start, stop and reboot VNFs. Start, stop and reboot operations are performed using the RESTful interface.

A payload is required 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 the KPI details. The VMs start running and move to VM_ALIVE_STATE. The service will be in service_active_state. Only undeploy can interrupt the start VNF workflow.
- Stop VNF: Once the service is stopped, monitoring is disabled and all the VM services are stopped. The VMs are no longer available. The service will be in service_stopped_state. VM will be in shutoff_state. You cannot perform any recovery, scale out, scale in. You can only undeploy the VNFs.
- Reboot VNF: Disables monitoring, reboots all VMs, that is stop and then start in OpenStack, enables monitoring, and reassigns thresholds according to 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 cannot 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 individual VMs.

A payload is required 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 in the operation field.

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 12

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 18: 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 19: 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 20: 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 21: 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 22: 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 primary and secondary 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 102.

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-ucs-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-ucs-50
16:13:19,143 14-Mar-2018 INFO  =====  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 13

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.

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 primary 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.

```
<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 servicescaleout.sh</action>
</rule>
<rule>
   <event_name>VM_UNDERLOADED</event_name>
   <action>ALWAYS log</action>
   <action>TRUE servicescalein.sh</action>
</rule>
```

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 14

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 datamodel. 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 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.

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 recovery policy.

If ESC does not receive a VM Alive after deployment, ESC recovers the VM with the recovery policy when timeout happens. All the recovery procedures depend on the recovery policy configuration. For example, if the user configured either one of the recovery policy such as Reboot Only, Redeploy Only, or Reboot and Redeploy then ESC will follow the same configured policy.

ESC provides 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:

- <kpi> section defines the type of monitoring, events, polling interval and other parameters.
- <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 102.

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 above examples, we define a KPI to monitor the ICMP Ping on the nicid 0. It defines the attributes metric condition 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 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 un-deployed and a new VM with same day-0 configuration is deployed. ESC tries to reuse the same network configuration like 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 for the duration of 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 taken to recover the VM. VM recovery resumes when the gateway becomes reachable.

In case of a double fault condition, that is, when the network gateway and the VM failure occur at the same time, ESC automatically performs VM monitoring after the gateway is reachable again.

Recovery Policy

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 recovery policy.

If ESC does not receive a VM Alive after deployment, ESC recovers VM with the recovery policy when timeout happens. All the recovery procedures depend on the recovery policy configuration.

ESC has the following VM recovery policies that you can specify when you deploy a VNF:

- **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:

- **REBOOT_THEN_REDEPLOY (default)**—When a VM down event is received or 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 timer expires, the healing workflow only attempts to reboot the VM.
- **REDEPLOY_ONLY**—When a VM down event is received or timer expires, the healing workflow only attempts to redeploy the VM.

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.

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. Further, the recovery action is based on the action-on-recovery parameter in the recovery policy. For manual recovery of a complete deployment, see Manual Recovery of a Deployment, on page 207

The manual recovery policy datamodel 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 datamodel, 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-ucs-27
 WARN      [DEBUG-ONLY] VM IP: 192.168.0.3;
 WARN  =====  SEND NOTIFICATION ENDS  =====
```

APIs for Manual Recovery of a VM

You can perform manual recovery using the Confd and Rest APIs.

- **Netconf API** `recovery-vm-action <DO> <VM-NAME>`

  To perform recovery using the API, login to esc_nc_cli and run the following command:

  ```
  $ esc_nc_cli recovery-vm-action DO
  SystemAdminTena_zz-g1_0_94fc1ab6-16a8-4121-9b04-c15b3d73cb39
  ```

  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.

- **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"
  }
  ```

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 205.

APIs for Manual Recovery of a Deployment

You can perform manual recovery using the NETCONF and REST APIs.

Note There is no service active notification after the deployment recovery. You must run a query, for example, *esc_nc_cli get esc_datamodel* to see if the service state of the deployment is active or not.

- **NETCONF API** `svc-action RECOVER <tenant-name> <deployment-name>`

  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://127.0.0.1:9010/
Callback-ESC-Events: http://127.0.0.1:9010/
<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 VM level recovery request when the service is in 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	ACTIVE	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.

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 116.

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.

Note You can perform recovery without using the policy framework. See Recovery Policy, on page 205

.

- **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)

Later than Cisco ESC Release 2.2, 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 205.

The table below describes the predefined actions performed at different lifecycle stages. For a detailed predefined action and LCS combination, see .

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.

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 116.

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 209.

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 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 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 198 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 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-ucs-12</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-ucs-12</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-ucs-28</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-ucs-12</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-ucs-28</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

These notifications are generated based on the workflow. Each notification will have 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 workflow reboots the vm and sends 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

ETSI MANO Compliant ESC Lifecycle Operations

CHAPTER 15

ETSI MANO Northbound API Overview

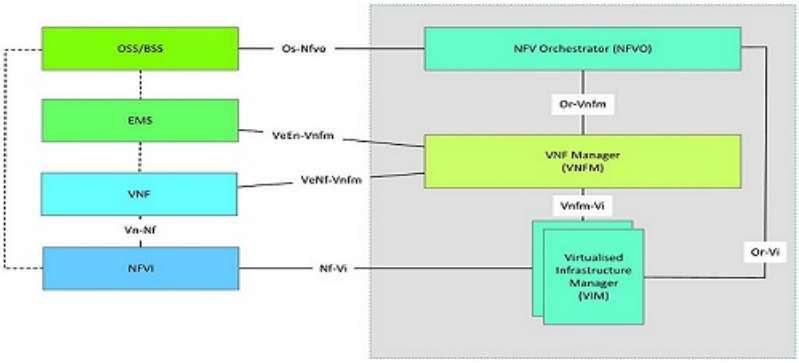
The ETSI MANO API is another programmatic interface to ESC that uses the REST architecture. The ETSI MANO adheres to the standards defined by the European Telecommunications Standards Institute (ETSI), specifically around Management and Orchestration (MANO). The API accepts and returns HTTP messages that contain JavaScript Object Notation (JSON). The API contains its own datamodel designed around the ETSI MANO specifications (*ETSI GS NFV-SOL 003 V2.3.1*) that abstract away from the ESC core datamodel.

The initial implementation of ETSI standard supports ETSI MANO API over Or-Vnfm reference point, which is the interface between ESC and NFV MANO. The Or-Vnfm reference point details the interactions to onboard ETSI compliant VNF packages, manage resources, and VNF lifecycle management (LCM) operations.

Note The terminology used in the ETSI-specific sections of the user guide align to the ETSI MANO standards defined in the ETSI documentation. For more information, see the ETSI website.

For more information on Or-Vnfm reference point, see the *ETSI Group Specification document* on the ETSI website. The figure below represents the NFV MANO architecture with the Or-Vnfm reference point.

Figure 3: NFV MANO Architecture with Reference Points



• Resource Definitions for ETSI API, on page 225

Resource Definitions for ETSI API

Cisco Elastic Services Controller (ESC) resources comprises of images, flavours, tenants, volumes, networks, and subnetworks. These resources are the ones that ESC requests to provision a Virtual Network Function.

For ETSI MANO, these resource definitions are created by NFVO either at the time of onboarding the VNF package or onboarding the tenant, and represented by the VIM identifiers in the request to ESC.

For information on managing resources using NETCONF or REST APIs, see Managing Resources Overview.

To access ETSI MANO API documentation directly from the ESC VM, see ETSI MANO Northbound API.

The following table lists the resource definitions on the VIM that must be made available before VNF instantiation.

Note Virtual Network Function lifecycle operations using ETSI MANO API is supported on OpenStack only.

Table 23: Resource Definitions on VIM

Resource Definitions	OpenStack
Tenants	Out of band tenants You can create a tenant using NETCONF API, REST API, or the ESC portal. You can also create a tenant directly on the VIM. The tenant is then referred to as the resourceGroupId.
Images	Out of band images The NFVO onboards a VNF package and then onboards the image contained within the VNF package on to the VIM. This can then be referenced in the sw_image attribute.
Flavours	Out of band flavours During onboarding of the VNF Package, the NFVO looks at the tosca.nodes.nfv.VDU.Compute node in the VNFD to determine the flavour to be created. This is available later at the time of instantiation. **Note** ETSI deployment flavour is a different concept than OpenStack compute flavor. For more information, see Terms and Conditions.
Volumes	Out of band volumes The out of band volume requirements are detailed in the tosca.nodes.nfv.VDU.Compute node in the VNFD.
External Networks (Virtual Link)	Out of band networks
Externally Managed Internal Virtual Links	The externally managed internal virtual links are defined in the VNFD, or referred to if the resource is out of band.
Subnetworks	Out of band subnets

For information on onboarding VNF packages and lifecycle operations using the ETSI MANO API, see VNF Lifecycle Operations Using ETSI API.

CHAPTER 16

Managing VNF Lifecycle Using ETSI API

The NFVO communicates with ESC using the ETSI MANO API for lifecycle management of a VNF. A configuration template, the Virtual Network Function Descriptor (VNFD) file describes the deployment parameters and operational behaviors of a VNF type. The VNFD is used in the process of deploying a VNF and managing the lifecycle of a VNF instance.

The lifecycle operations of a VNF instance is as follows:

1. **Create a VNF Identifier**—ESC generates a new VNF Instance Id (a universally unique identifier) that is subsequently used as a handle to reference the instance upon which to execute further operations.
2. **Instantiate / Deploy VNF**—As part of VNF instantiation, ESC instantiates a new VNF instance in the VIM. ESC receives a request to instantiate a VNF instance from NFVO. The instantiate request contains resource requirements, networking and other service operational behaviors. All these requirements along with the VNFD and the grant information provides all the necessary information to instantiate the VNF.
3. **Operate VNF**—ESC allows you to start and stop a VNF instance. The resources are not released or changed, but the VNF instance in the VIM is toggled between these two states.
4. **Query VNF**—To query one or more VNF instances known to ESC. To query one or more VNF instances known to ESC. This is a specific REST end point that can be filtered to find specific instances. The instances can be filtered using the VNF Instance Id.

   Also, a separate REST end point allows the NFVO to query the status of one or more lifecycle operation occurrences associated with a VNF. The lifecycle operations can be filtered using a specific occurrence identifier.
5. **Modify VNF**—ESC allows you to modify the properties of a single VNF instance. The instantiated VNF is updated, and the lifecycle management operation occurrence sends notification to the NFVO about the status of the VNF.
6. **Scale and Scale to Leve VNF**—ESC allows you to scale VNFs in two ways. You can scale a VNF incrementally, or to a specific level.
7. **Heal VNF**—ESC heals the VNF when there is a failure.
8. **Terminate / Undeploy VNF**—To terminate the VNF instance in the VIM. The resources themselves remain reserved for the VNF instance, however the VNF itself is undeployed.
9. **Delete VNF Identifier**—The resources are fully released in the VIM and in ESC and the associated VNF instance identifer is also released.

For VNF lifecycle operations using REST and NETCONF APIs, see Deploying and Configuring Virtual Network Functions.

• VNF Lifecycle Operations Using ETSI API, on page 230

VNF Lifecycle Operations Using ETSI API

Prerequisites

The following prerequisites must be met for VNF lifecycle operations:

- The resource definitions must be created out of band and must be available before VNF instantiation.
- The VIM connector specifies the VIM used. The VIM connector information must be available before instantiating the VNF. ESC must contain a valid, and authenticated default OpenStack VIM connector to deploy the VNFs.
- The VNF to be instantiated has to be onboarded to the NFVO within an ETSI compliant VNF Package.
  - The NFVO must provide ETSI compliant VNF Packages to ESC.
  - The VNF package must contain a VNF Descriptor (VNFD) file.

  The ETSI MANO NFV specifications supports /vnf_packages API to allow access to the package artifacts.See chapter 10 in the *ETSI GS NFV-SOL 003 V2.4.1* specification on the ETSI website.

  The properties file provides details about the NFVO to the ETSI VNFM service.

  The properties file, *etsi-production.properties* exists under: `/opt/cisco/esc/esc_database/`

  The single property *nfvo.apiRoot* allows specification of the NFVO host and port. For example, `nfvo.apiRoot=localhost:8280`.

Note The initial implementation of the ETSI MANO API supports only a single VIM. The tenant/project is currently specified using the resourceGroupId.

For notes on ESC in HA mode, enabled with ETSI service, see the Cisco Elastic Services Controller Install and Upgrade Guide.

Creating the VNF Identifier

Creating the VNF Identifier is the first request for any VNF instance. This identifier is used for all further LCM operations executed by the ETSI API. Resources are neither created nor reserved at this stage.

ESC sends a POST request to create VNF instances:

Method Type:

```
POST
```

VNFM Endpoint:

```
/vnf_instances/
```

HTTP Request Headers:

```
Content-Type:application/json
```

Request Payload (ETSI data structure: CreateVnfRequest):

```
{
    "vnfInstanceName": "Test-VNf-Instance",
    "vnfdId": "vnfd-88c6a03e-019f-4525-ae63-de58ee89db74"
}
```

Response Headers:

```
HTTP/1.1 201
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Frame-Options: DENY
Strict-Transport-Security: max-age=31536000 ; includeSubDomains
X-Application-Context: application:8250
Accept-Ranges: none
Location: http://localhost:8250/vnflcm/v1/vnf_instances/14924fca-fb10-45da-bcf5-59c581d675d8
Content-Type: application/json;charset=UTF-8
Transfer-Encoding: chunked
Date: Thu, 04 Jan 2018 12:18:13 GMT
```

Response Body (ETSI Data structure:VnfInstance)

```
{
    "_links": {
        "instantiate": {
            "href":
"http://localhost:8250/vnflcm/v1/vnf_instances/14924fca-fb10-45da-bcf5-59c581d675d8/instantiate"

        },
        "self": {
            "href":
"http://localhost:8250/vnflcm/v1/vnf_instances/14924fca-fb10-45da-bcf5-59c581d675d8"
        }
    },
    "id": "14924fca-fb10-45da-bcf5-59c581d675d8",
    "instantiationState": "NOT_INSTANTIATED",
    "onboardedVnfPkgInfoId": "vnfpkg-bb5601ef-cae8-4141-ba4f-e96b6cad0f74",
    "vnfInstanceName": "Test-VNf-Instance",
    "vnfProductName": "vnfd-1VDU",
    "vnfProvider": "",
    "vnfSoftwareVersion": "not-implemented",
    "vnfdId": "vnfd-88c6a03e-019f-4525-ae63-de58ee89db74",
    "vnfdVersion": ""
}
```

Instantiating Virtual Network Functions

The instantiation request triggers a number of message exchanges, which allows the call flow to be completed in order to instantiate a VNF instance. The resources are allocated when the VNF instance is instantiated. It requires the VNF instance identifier, returned by the create VNF request, encoded into the URL to which the request is posted.

The instatiation request sub-tasks within the flow include:

1. Requesting permission from the NFVO (bi-directional Grant flow). For more information see, Requesting Grant Permission.
2. Retrieving the VNF Descriptor template from the NFVO.

3. Requesting the VNF resources to be allocated to the instance from the VIM.

Typically, the request permission, and retrieve messages are customized for the NFVO.

Method type:

```
POST
```

VNFM Endpoint:

```
/vnf_instances/{vnfInstanceId}/instantiate
```

HTTP Request Header:

```
Content-Type:application/json
```

Request Payload (ETSI data structure: InstantiateVNFRequest)

```
{
    "flavourId": "vdu_node1",
    "extManagedVirtualLinks": [
        {
           "id": "esc-net",
            "resourceId": "RES1",
            "virtualLinkDescId": "VLD1"
        }
    ],
    "vimConnectionInfo": [
        {
            "accessInfo": {
                "resourceGroupId": "tenantName"
            },
            "id": "vimConnId",
            "vimType": "Openstack"
        }
    ]
}
```

The flavourId value must be same as the flavour_id specified in the VNFD file under properties.

Response Headers:

```
HTTP/1.1 202
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Frame-Options: DENY
Strict-Transport-Security: max-age=31536000 ; includeSubDomains
X-Application-Context: application:8250
Accept-Ranges: none
Location: http://localhost:8250/vnflcm/v1/vnf_lcm_op_occs/457736f0-c877-4e07-8055-39dd406c616b
Content-Length: 0
Date: Thu, 04 Jan 2018 12:20:40 GMT
```

Response Body:

```
not applicable.
```

You can customize the VNF before instantiation by adding variables to the VNFD template. Specify the variables in the *additionalParams* field of the LCM request. The variables are name-value pairs, where the

value can be either string, numeric or boolean. In the example below, the *cpus*, and *mem_size additionalParams* are defined in the VNFD template.

Method type:

```
POST
```

VNFM Endpoint:

```
/vnf_instances/{vnfInstanceId}/instantiate
```

HTTP Request Header:

```
Content-Type:application/json
```

Request Payload (ETSI data structure: InstantiateVNFRequest)

```
{
    "flavourId": "default",
    "additionalParams": {
        "cpus": 2,
        "mem_size": "512 MB"
    }
}'
```

When this template is submitted to the VNFM, the variables are merged into the same VNF instance.

The *additionalParams* variables are merged with the VNF variables, and actual values for the variables are provided only during instantiation.

The *cpus*, and *mem_size* variables are merged with the VNF variables in the example below.

```
node_templates:
    my_server:
      type: tosca.nodes.Compute
      capabilities:
        host:
          properties:
            num_cpus: 2
            mem_size: 512 MB
            disk_size: 10 GB
```

If further LCM requests with *additionalParams* variables are submitted for the same VNF, then the new variables overwrite the existing variables. The VNFM uses the new variables for instantiation.

Note The additional variables must be initially defined in the VNFD template, so that it can be used during instantiation, else the submission will fail.

ESC supports internal virtual links while instantiating a VNF using the TOSCA VNFD template.

Example for Virtual Internal Links from a TOSCA VNFD template:

```
   esc-test-net:
      type: tosca.nodes.nfv.VnfVirtualLinkDesc
      properties:
        connectivity_type:
          layer_protocol: ipv4
        description: VDU Internal Network VL

    EtsiTestInternalVL-Net-1:
      type: tosca.nodes.nfv.VnfVirtualLinkDesc
```

```
      properties:
        connectivity_type:
          layer_protocol: ipv4
        description: VDU Internal Network VL
        externally_managed: false
```

Requesting Grant Permission

The ETSI API requests for permission from the NFVO for lifecycle management operations to complete the operations in progress for the VNF instance resources.

```
{
  "vnfInstanceId":"b9909dde-e21e-45ec-9cc0-9e9ae413eee0",
  "vnfLcmOpOccId":"d1409dde-f23e-61fh-9dc4-9e9eg667egb7",
  "vnfdId":"aaaaa-bbbb-1111-2222-cccc-1a3c5bb6543",
  "flavourId":"vanilla", /* not required? */
  "operation":"instantiate",
  "isAutomaticInvocation":"false", /* not required? */
  "instantiationLevelId":"admin", /* not required? */
  "addResources": {
      "":""
  }, /* not required? */
  "tempResources":"", /* not required? */
  "placementConstraints": {
      "affinityOrAntiAffinity":"ANTI_AFFINITY",
      "scope":"ZONE",
      "resource": {
          "idType":"GRANT",
          "resourceId":"myresourceid123"
      },
  }, /* not required? */
  "additionalParams": {
      "project":"tenant1"
  }, /* not required? */
  "links": {

"vnfLcmOpOcc":"http://my-esc-vm:8250/vnf_lcm_op_occs/b9909dde-e21e-45ec-9cc0-9e9ae413eee2",

      "vnfInstance":
"http://my-esc-vm:8250/vnflcm/v1/vnfinstances/b9909dde-e21e-45ec-9cc0-9e9ae413eee0"
  }
  "action": "instantiate"
}
```

Querying Virtual Network Functions

Querying VNFs does not affect the state of any VNF instance. This operation simply queries ESC for all the VNF instances it knows about, or a specific VNF isntance.

Method Type:

```
GET
```

VNFM Endpoint:

```
/vnf_instances
/vnf_instances/{vnfInstanceId}
```

Content-Type:application/merge-patch+json

ETag: "etag returned from Vnf Instance query"

```
Content-Type:application/json
```

Request Payload:

```
 not applicable.
```

Response Headers:

```
< HTTP/1.1 200
HTTP/1.1 200
< X-Content-Type-Options: nosniff
X-Content-Type-Options: nosniff
< X-XSS-Protection: 1; mode=block
X-XSS-Protection: 1; mode=block
< Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
< Pragma: no-cache
Pragma: no-cache
< Expires: 0
Expires: 0
< X-Frame-Options: DENY
X-Frame-Options: DENY
< Strict-Transport-Security: max-age=31536000 ; includeSubDomains
Strict-Transport-Security: max-age=31536000 ; includeSubDomains
< X-Application-Context: application:8250
X-Application-Context: application:8250
< Accept-Ranges: none
Accept-Ranges: none
< ETag: "2"
ETag: "2"
< Content-Type: application/json;charset=UTF-8
Content-Type: application/json;charset=UTF-8
< Transfer-Encoding: chunked
Transfer-Encoding: chunked
< Date: Thu, 04 Jan 2018 12:25:32 GMT
Date: Thu, 04 Jan 2018 12:25:32 GMT
```

Response Body for a single VNF Instance (ETSI Data structure:VnfInstance)


```
{
    "_links": {
        "instantiate": {
            "href":
"http://localhost:8250/vnflcm/v1/vnf_instances/14924fca-fb10-45da-bcf5-59c581d675d8/instantiate"

        },
        "self": {
            "href":
"http://localhost:8250/vnflcm/v1/vnf_instances/14924fca-fb10-45da-bcf5-59c581d675d8"
        }
    },
    "id": "14924fca-fb10-45da-bcf5-59c581d675d8",
    "instantiationState": "NOT_INSTANTIATED",
    "onboardedVnfPkgInfoId": "vnfpkg-bb5601ef-cae8-4141-ba4f-e96b6cad0f74",
    "vnfInstanceName": "Test-VNf-Instance",
    "vnfProductName": "vnfd-1VDU",
    "vnfProvider": "",
    "vnfSoftwareVersion": "not-implemented",
    "vnfdId": "vnfd-88c6a03e-019f-4525-ae63-de58ee89db74",
    "vnfdVersion": ""
}
```


Response Body for all VNF Instances (ETSI Data structure:VnfInstance[])


```
{
```

```
    "_embedded": {
        "vnfInstances": [
            {
                "_links": {
                    "instantiate": {
                        "href":
"http://localhost:8250/vnflcm/v1/vnf_instances/14924fca-fb10-45da-bcf5-59c581d675d8/instantiate"

                    },
                    "self": {
                        "href":
"http://localhost:8250/vnflcm/v1/vnf_instances/14924fca-fb10-45da-bcf5-59c581d675d8"
                    }
                },
                "id": "14924fca-fb10-45da-bcf5-59c581d675d8",
                "instantiationState": "NOT_INSTANTIATED",
                "onboardedVnfPkgInfoId": "vnfpkg-bb5601ef-cae8-4141-ba4f-e96b6cad0f74",
                "vnfInstanceName": "Test-VNf-Instance",
                "vnfProductName": "vnfd-1VDU",
                "vnfProvider": "",
                "vnfSoftwareVersion": "not-implemented",
                "vnfdId": "vnfd-88c6a03e-019f-4525-ae63-de58ee89db74",
                "vnfdVersion": ""
            },
            {
                "_links": {
                    "instantiate": {
                        "href":
"http://localhost:8250/vnflcm/v1/vnf_instances/9b82e224-2be5-44d8-821b-64ad7f4997fb/instantiate"

                    },
                    "self": {
                        "href":
"http://localhost:8250/vnflcm/v1/vnf_instances/9b82e224-2be5-44d8-821b-64ad7f4997fb"
                    }
                },
                "id": "9b82e224-2be5-44d8-821b-64ad7f4997fb",
                "instantiationState": "NOT_INSTANTIATED",
                "onboardedVnfPkgInfoId": "vnfpkg-bb5601ef-cae8-4141-ba4f-e96b6cad0f74",
                "vnfInstanceName": "Test1-VNf-Instance",
                "vnfProductName": "vnfd-1VDU",
                "vnfProvider": "",
                "vnfSoftwareVersion": "not-implemented",
                "vnfdId": "vnfd-88c6a03e-019f-4525-ae63-de58ee89db74",
                "vnfdVersion": ""
            }
        ]
    },
    "_links": {
        "self": {
            "href": "http://localhost:8250/vnflcm/v1/vnf_instances"
        }
    }
}
```


The query VNF operation ouput shows the instantiated state of the VNF. The *InstantiatedVnfInfo* element shows the VIM resource information for all the VNFs.

Example


```
"instantiatedVnfInfo": {
                    "extCpInfo": [
                        {
                            "addresses": [
```

```
                                    {
                                        "ipAddress": "172.16.253.27",
                                        "macAddress": "fa:16:3e:f3:68:6a"
                                    }
                                ],
                                "cpdId": "node1_ecp",
                                "id": "extCp-4d6d9ea9-c159-4c4a-9f6b-b8305491a6bc"
                            }
                        ],
```

Modifying Virtual Network Functions

You can modify or update the properties of a VNF instance using the modify VNF lifecycle operation. ESC receives a PATCH request from NFVO to modify a single VNF instance.

A JSON merge algorithm is applied from the input payload against the stored data to modify the VNF instance.

Note Modifying VNF operation updates only the properties, but not the functionality of the VNF.

The VNF is instantiated and deployed within ESC. The modify VNF operation does not affect the VIM.

The following properties of an existing VNF instance can be modified:

- vnfInstanceName
- vnfInstanceDescription
- onboardedVnfPkgInfoId (null value is not allowed)
- vnfConfigurableProperties
- metadata
- extensions
- vimConnectionInfo

Method Type

```
PATCH
```

VNFM Endpoint

```
/vnf_instances/{vnfInstanceId}
```

HTTP Request Header

```
Content-Type:application/json
```

Request Payload(ETSI data structure: VnfInfoModifications)

```
{
    "vnfInstanceName": "My NEW VNF Instance Name",
    "vnfInstanceDescription": "My NEW VNF Instance Description",
    "vnfPkgId": "pkg-xyzzy-123",
    "vnfConfigurableProperties": {
        "isAutoscaleEnabled": "true"
    },
    "metadata": {
        "serialRange": "ab123-cc331",
```

```
        "manufacturer": "Cisco"
    },
    "extensions": {
        "testAccess": "false",
        "ipv6Interface": "false"
    },
    "vimConnectionInfo": [
        {
            "id": "vci1",
            "vimType": "openstack",
            "interfaceInfo": {
                "uri": "http://10.51.14.27:35357/v3"
            },
            "accessInfo": {
                "domainName": "default",
                "projectName": "admin",
                "userName": "default"
            }
        }
    ]
}
```

Response Header: NA

Response Body: NA

When the PATCH operation is complete, the VNF instance is modified, and the details are sent to the NFVO through the notification.

Operating Virtual Network Functions

You can start or stop a VNF instance using the operate lifecycle management operation. The VNF instance can be stopped gracefully or forcefully.

Note The OpenStack API supports only forceful stop.

The *changeStateTo* field must have the value STARTED or STOPPED in the request payload, to start or stop a VNF instance.

Permission is also required from the NFVO (bi-directional Grant flow) for this operation. See Requesting Grant Permission for more informaiton.

Method Type:

```
POST
```

VNFM Endpoint:

```
/vnf_instances/{vnfInstanceId}/operate
```

HTTP Request Headers:

```
Content-Type:application/json
```

Request Payload:

```
{
```

```
    "additionalParams": {
        "username": "user1"
    },
    "changeStateTo": "STOPPED",
    "stopType": "FORCEFUL"
}
```

Response Headers:

```
HTTP/1.1 202
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Frame-Options: TEST
Strict-Transport-Security: max-age=31536000 ; includeSubDomains
X-Application-Context: application:8250
Accept-Ranges: none
Location: http://localhost:8250/vnflcm/v1/vnf_lcm_op_occs/e775aad5-8683-4450-b260-43656b6b13e9
Content-Length: 0
Date: Thu, 04 Jan 2018 12:40:27 GMT
```

Response Body:

```
not applicable.
```

Terminating Virtual Network Functions

The terminating VNF request terminates a VNF instance. The resources are deallocated but remain reserved for this instance until it is deleted. Permission is required from the NFVO (bi-directional Grant flow) for this operation. The VNF instance can be decommissioned gracefully or forcefully.

Note The OpenStack API supports only forceful termination.

As per the Instantiate VNF Request, the terminate VNF request requires the VNF instance identifier encoded into the URL to which the request is posted.

Method Type:

```
POST
```

VNFM Endpoint:

```
/vnf_instances/{vnfInstanceId}/terminate
```

HTTP Request Headers:

```
Content-Type:application/json
```

Request Payload (ETSI data structure: TerminateVnfRequest)

```
{
  "terminationType":"FORCEFUL",
  "additionalParams": {
    "password": "pass1234",
    "username": "admin"
```

```
  }
}
```

Response Headers:

```
HTTP/1.1 202
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Frame-Options: TEST
Strict-Transport-Security: max-age=31536000 ; includeSubDomains
X-Application-Context: application:8250
Accept-Ranges: none
Location: http://localhost:8250/vnflcm/v1/vnf_lcm_op_occs/dae25dbc-fcde-4ff9-8fd6-31797d19dbc1
Content-Length: 0
Date: Thu, 04 Jan 2018 12:45:59 GMT
```

Response Body:

```
not applicable.
```

Deleting Virtual Network Functions

Deleting VNF operation releases the VIM resources reserved for the VNF instance as well as deletes the VNF instance identifier. Upon deletion, the VNF instance identifier is no longer available. So, no further lifecycle management operations are possible using this identifier.

Method Type:

```
DELETE
```

VNFM Endpoint:

```
/vnf_instances/{vnfInstanceId}
```

HTTP Request Headers:

```
Content-Type:application/json
```

Request Payload:

```
not applicable.
```

Response Headers:

```
HTTP/1.1 204
HTTP/1.1 204
< X-Content-Type-Options: nosniff
X-Content-Type-Options: nosniff
< X-XSS-Protection: 1; mode=block
X-XSS-Protection: 1; mode=block
< Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
< Pragma: no-cache
Pragma: no-cache
< Expires: 0
Expires: 0
< X-Frame-Options: TEST
X-Frame-Options: TEST
< Strict-Transport-Security: max-age=31536000 ; includeSubDomains
Strict-Transport-Security: max-age=31536000 ; includeSubDomains
< X-Application-Context: application:8250
X-Application-Context: application:8250
```

```
< Accept-Ranges: none
Accept-Ranges: none
< Date: Thu, 04 Jan 2018 12:48:59 GMT
Date: Thu, 04 Jan 2018 12:48:59 GMT
```

Response Body:

```
not applicable.
```

CHAPTER 17

Understanding Virtual Network Function Descriptors

Virtual Network Function Descriptor Overview

In ESC, ETSI supports a TOSCA based VNF Descriptor (VNFD). The VNFD conforms to the GS NFV-SOL 001 specifications and standards specified by ETSI.

The Virtual Network Function Descriptor (VNFD) file describes the instantiation parameters and operational behaviors of the VNFs. It contains KPIs, and other key requirements that can be used in the process of onboarding and managing the lifecycle of a VNF.

For VNF Lifecycle operations, see VNF Lifecycle Operations Using ETSI API, on page 230.

Creating Extensions to the Virtual Network Function Descriptor

ESC creates extensions to the VNFD during instantiation. These extensions specified by ESC map to the ETSI VNFD template attributes.

The ESC extensions are passed to the VNFD template through the *cisco_esc_properties* extension.

ESC Interface Field

The ESC interface field maps to the Connection Point Descriptor (CPD) in the VNFD template.

The following interface fields are currently supported.

- type
- nicid
- network
- allowed_address_pairs
- static_ip_address_pool
- security_groups

Example:

```
node_2_nic0:
  type: tosca.nodes.nfv.Cpd.CiscoESC
  properties:
    layer_protocol: ipv4
    cisco_esc_properties:
      management: true
      nicid: 0
      allowed_address_pairs:
        - ip_address: 162.77.11.0/24
        - ip_address: 162.77.6.0/20
  requirements:
    - virtual_link: esc-net
    - virtual_binding: vdu_node_2

node_2_nic1:
  type: tosca.nodes.nfv.Cpd.CiscoESC
  properties:
    layer_protocol: ipv4
    cisco_esc_properties:
      type: virtual
      management: false
      nicid: 1
      static_ip_address_pool:
        - 182.18.1.10
        - 182.18.1.11
        - 182.18.1.13
        - 182.18.1.15
      allowed_address_pairs:
        - ip_address: 172.77.12.0/30
      security_groups:
        - default
        - restricted_ssh
  requirements:
    - virtual_link: automation-static-network
    - virtual_binding: vdu_node_2
```

Day 0 Configuration

The ESC day 0 configuration parameters are passed through the VNFD template.

Example:

```
 vdu_node_1:
      type: tosca.nodes.nfv.VDU.Compute.CiscoESC
      capabilities:
..
      properties:
        additional_vnfc_configurable_properties: {}
        cisco_esc_properties:
          vim_flavour: Some-Flavor-3_14
          config_data:
            some_config.txt:
              file: http://10.85.103.34/share/qatest/day0/specific_config.sh
               variables:
                 CF_DOMAIN_NAME: cisco.com
                 CF_NAME_SERVER: 171.70.168.183
```

```
            # staros_param.cfg:
            #   file: http://10.85.103.34/share/automation_scripts/day0/asa_config.sh
```

Deployment Level Lifecycle Stages (LCS) Policies

ESC deployment level Lifecycle Stages (LCS) are passed through the VNFD template.

Example:

```
capabilities:
        deployment_flavour:
          properties:
            flavour_id: default
            description: 'Default VNF Deployment Flavour'
            vdu_profile:
              vdu_node_1:
                min_number_of_instances: 1
                max_number_of_instances: 1
            instantiation_levels:
              default:
                description: 'Default Instantiation Level'
                vdu_levels:
                  vdu_node_1:
                    number_of_instances: 1
            default_instantiation_level_id: default
            vnf_lcm_operations_configuration: {}
            cisco_esc_properties:
             policies:
             # Policy name in this case is "1"
               1:
                 conditions:
                   - name: 'LCS::PRE_DEPLOY'
                 actions:
                   - name: 'NOTIFY'
                     type: 'pre-defined'
                     properties:
                         prop_one: value_of_prop
                         script_filename: /opt/cisco/esc/esc-scripts/esc_vpc_chassis_id.py
...
```

Volume Container

The ESC volume maps to the virtual storage in the VNFD template.

The following fields are currently supported:

- bus
- vol_id
- type
- boot_index

Example:

```
 vdu_node_1:
      type: tosca.nodes.nfv.VDU.Compute.CiscoESC
      capabilities:
        virtual_compute:
..
      requirements:
        - virtual_storage: test-volume-ets
```

```
...

    test-volume-ets:
      type: tosca.nodes.nfv.VDU.VirtualStorage.CiscoESC # CiscoESC - NOTE: NFVO can strip
suffix
      properties:
        type_of_storage: volume # SOL001
        size_of_storage: 5 # SOL001 - NOTE: in GB
        cisco_esc_properties:
          vol_id: 0
          bus: virtio
          type: LUKS
```

For VNF Lifecycle operations, see VNF Lifecycle Operations Using ETSI API, on page 230.

CHAPTER 18

Scaling and Healing VNFs Using ETSI API

Scaling VNFs Using ETSI API

ESC can scale VMs using the ETSI API. The scaling workflow begins when the VNF instance is in the instantiated state. The scaling details are defined in the VNFD by the NFVO.

Scaling of a VNF instance can be achieved in two ways:

- Scale
- Scale to level

As part of scaling,

- Add VM instances to the VNF to scale out.
- Remove VM instances from the VNF to scale in.

Note Currently, ETSI supports only manual scaling.

Scale

ESC can scale a VNF incrementally. The *numberOfSteps* attribute determines the increment by which you can scale the VNF instance. By default, the number of steps is set to 1.

Method type:

```
POST
```

VNFM Endpoint:

```
/vnf_instances/{vnfInstanceId}/scale
```

HTTP Request Header:

```
Content-Type:application/json
```

Request Payload (ETSI data structure: ScaleVnfRequest)

```
{
  "type":"SCALE_OUT",
  "aspectId":"processing",
  "numberOfSteps":"1",
}
```

Response Headers:

```
not applicable.
```

Response Body:

```
not applicable.
```

Scale to Level

Allows ESC to scale the VNF to a specified level.

Method type:

```
POST
```

VNFM Endpoint:

```
/vnf_instances/{vnfInstanceId}/scale_to_level
```

HTTP Request Header:

```
Content-Type:application/json
```

Request Payload (ETSI data structure: ScaleVnfRequest)

```
{
/* "instantiationLevelId":"id111", */
"scaleInfo": [
{ "aspectId":"processing", "scaleLevel":"3" },
{ "aspectId":"database", "scaleLevel":"2" }
]
"additionalParams": {
"password": "pass1234",
"username": "admin"
}
}
```

Response Headers:

```
not applicable.
```

Response Body:

```
not applicable.
```

Healing VNFs Using ETSI API

As part of the VNF lifecycle management, ESC heals the VNFs when there is a failure. ETSI API heals a VNF instance based on the instructions from the NFVO.

Note Currently, ETSI supports only manual healing.

Method Type:

```
POST
```

VNFM Endpoint:

```
/vnf_instances/{vnfInstanceId}/heal
```

HTTP Request Headers:

```
Content-Type:application/json
```

Request Payload (ETSI data structure: HealVnfRequest):

```
{
"cause":"b9909dde-e21e-45ec-9cc0-9e9ae413eee0",
"additionalParams": {
"password": "pass1234",
"username": "admin"
}
}
```

Response Headers:

```
not applicable.
```

Response Body:

```
not applicable.
```

Manual Recovery of a Deployment

Currently, ETSI supports recovery of a complete deployment.

REST API to heal a complete deployment is as follows:

```
{http_scheme}://{restapi_root}:8080/ESCAPI/#!/Recovery_VM_Operations/handleOperation

/v0/{internal_tenant_id}/deployments/service/{internal_deployment_id}
```

payload:

```
Content-Type: application/xml
Accept: application/json
Callback: http://{etsi-vnfm-callback-endpoint}:{etsi-vnfm-callback-port}/
Callback-ESC-Events: http://127.0.0.1:9010/
 <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.

Both ids are encoded into the URL.

Manual healing of the complete deployment is supported when the deployment (service) is in the following states:

Table 24: VM and Deployment States

VM state	Deployment or service state	recovery-vm-action
error	error	supported
error	active	supported

CHAPTER 19

Error Handling Procedures

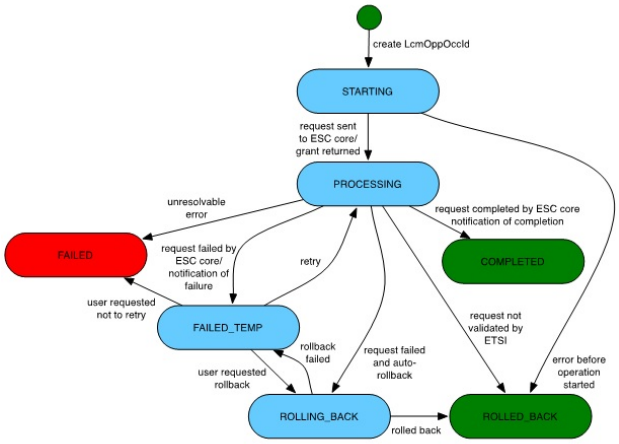
Error Handling Using the ETSI API

ETSI invokes the following error handling procedures for all its ETSI VNF lifecycle management (LCM) operations:

- Retry
- Rollback
- Fail
- Cancel

The image below represents the transitional states of the VNF lifecycle management operational occurrence.

Figure 4: VNF Lifecycle Management Transitional States



Note The *vnfLcmOpOccId* is encoded into the URI, which is the primary key to retrieve the request details.

The retry, rollback and fail requests are rejected if the LCM operation is in any other state other than the FAILED_TEMP state. This error returns HTTP code 409.

The retry, rollback, fail and cancel requests are not supported for the particular VNF LCM operation for the particular VNF. This error returns HTTP code 404.

An error occurs if the *vnfLcmOpOccId* does not exist in the ETSI database. This error returns HTTP code 404.

Retry

A retry request is applicable if there is a possibility of the LCM operation to succeed. The operation should be (pre-condition) in the FAILED_TEMP state for a retry request. You can send several retry requests, as long as the operation is in the FAILED_TEMP state.

Precondition	FAILED_TEMP state
Request	`POST {api_root}/vnf_lcm_op_occs/{vnfLcmOpOccId}/retry()`
Postcondition	PROCESSING state

Upon successful retry, ESC sends a START or PROCESSING notification. If the retry request fails, then ESC sends a notification to the NFVO with the details.

Rollback

A rollback request is made if it is not possible for the operation to succeed even after a retry request.

Set the *rollback_required* flag to true. If this is not set to true, then rollback is not performed.

Precondition	FAILED_TEMP state
Request	`POST {api_root}/vnf_lcm_op_occs/{vnfLcmOpOccId}/rollback()`
Postcondition	ROLLED_BACK

Upon successful rollback, the LCM operation is rolled back. If the rollback request fails, then the LCM operation is back to the failed_temp sate.

Fail

When an LCM operation does not require a retry request, or a clean up, a fail request allows you to free up resources for a subsequent request.

If the *rollback_required* flag is set to true, a fail request cannot be made.

Precondition	FAILED_TEMP state
Request	`POST {api_root}/vnf_lcm_op_occs/{vnfLcmOpOccId}/fail()`
Postcondition	FAILED state

Upon successful execution of this request, the LCM operation is in FAILED state.

Cancel

A cancel request is possible if the operation is in STARTING state.

Precondition	STARTING state
Request	`POST {api_root}/vnf_lcm_op_occs/{vnfLcmOpOccId}/cancel(CancelMode)`
Postcondition	ROLLED_BACK

The cancel request is Forceful.

Note ETSI supports canceling an LCM operation in starting state only. The cancel request for LCM operations in processing or rolling back states are currently not supported.

Example JSON payload (CancelMode):

```
{
  "cancelMode": "FORCEFUL",
  "action": "cancel"
}
```

Set the *IsCancelPending* attribute of the *VnfLcmOpOcc* to true. This will stop the processing request, and move the LCM operation to ROLLED_BACK state.

Error Handling Procedures for ETSI VNF Lifecycle Operations

If the LCM operation for a VNF instance fails, the operation moves to the FAILED_TEMP state according to the state machine. To complete the intended operation, you must either run the retry or rollback request.

- If creating a VNF identifier fails, then no further action is required. The rollback request is not supported.
- If instantiating the VNF fails, then ESC terminates the request, and sends a new instantiation request.
- If operating the VNF fails, then no further action is required.
- If terminating the VNF fails, you must retry the operation, as rollback is not supported.
- If deleting the VNF operation fails, then no further action is required. Th erollback request is not supported.

Note The error handling requests do not impact the operating VNF lifecycle operation.

For information on VNF lifecycle operations, see VNF Lifecycle Operations Using ETSI API, on page 230.

CHAPTER 20

Alarms and Notifications for ETSI LCM Operations

ETSI Alarms

ESC provides alarms and notifications to the NFVO. The NFVO has to subscribe to these alarms and notifications and send requests to ESC.

The NFVO can receive information about the alarms in the following ways:

Query All Alarms

The NFVO can get a list of all the alarms from the alarms resource.

Method Type:

```
GET
```

VNFM Endpoint:

```
/vnffm/v1/alarms
```

HTTP Request Headers:

```
Accept:application/json
```

For example, to query all alarms with the event type as ENVIRONMENTAL_ALARM

Method Type:

```
GET
```

VNFM Endpoint:

```
http://localhost:8250/vnffm/v1/alarms?eventType="ENVIRONMENTAL_ALARM"
```

HTTP Request Headers:

```
Accept:application/json
```

While querying for multiple alarms, the NFVO can use the URI query parameters to filter the results. The following attribute names are supported for the URI query of the alarms:

- id
- managedObjectId
- rootCauseFaultyResource.faultyResourceType
- eventType
- perceivedSeverity
- probableCause

Note The URI query parameters are for querying multiple alarms only.

Query an Individual Alarm

The NFVO can query a particular alarm from the *alarmId* resource.

Method Type:

```
GET
```

VNFM Endpoint

```
/vnffm/v1/alarms/{alarmId}
```

HTTP Request Header:

```
Accept:application/json
```

Modify an Individual Alarm

To modify an alarm, the NFVO must send a PATCH request to the *AlarmModifications* resource.

Method Type:

```
PATCH
```

VNFM Endpoint:

```
/vnffm/v1/alarms/{alarmId}
```

HTTP Request Header:

```
Content-Type:application/json
```

The supported attribute is ackState, and the supported attribute value is ACKNOWLEDGE. All other modification payloads are rejected.

Alarm Extensions

ETSI provides an extension for the alarms to interact with the third party tools. You must send a POST request to create the alarms.

Method Type

```
POST
```

VNFM Endpoint

```
/vnffm/v1/extension/alarms
```

HTTP Request Header

```
Content-Type:application/json
```

Request Payload

```
[admin@davwebst-esc-4-2-0-49-keep ETSI]$ cat CreateAlarm.json
{
    "id": "alm87032",
    "externalAlarmId": "ext-id-xx11214",
    "managedObjectId": "930fb087-c1b9-4660-bec8-2a8d97dc1df5",
    "rootCauseFaultyResource": {
        "id": "fres7629",
        "faultyResource": {
            "resourceId": "res7727"
        },
        "faultyResourceType": "NETWORK"
    },
    "alarmRaisedTime": "2018-05-30T13:55:15.645000+00",
    "ackState": "UNACKNOWLEDGED",
    "perceivedSeverity": "MAJOR",
    "eventTime": "2018-05-30T13:55:15.645000+00",
    "eventType": "ENVIRONMENTAL_ALARM",
    "probableCause": "Server room overheading",
    "isRootCause": "false"
}
```

Subscribing to Notifications

The NFVO can subscribe to the ETSI notifications related to fault management from ESC.

Create a Subscription

The NFVO sends a POST request to subscribe to the notifications.

Method Type

POST

VNFM Endpoint

```
/vnffm/v1/subscriptions
```

Response Payload

```
{
  "filter" : {
    "notificationTypes" : [
      "AlarmNotification",
      "AlarmClearedNotification",
      "AlarmListRebuiltNotification"
    ],
    "perceivedSeverities" : [
      "CRITICAL",
      "MAJOR"
    ]
  },
  "callbackUri" : "https://nfvo.endpoint.listener",
  "authentication" : {
    "authType" : "BASIC",
    "paramsBasic" : {
      "userName" : "admin",
```

```
      "password" : "pass123"
    }
  }
}
```

This creates a new subscription resource and a new identifier. The callbackUri is the only mandatory parameter. The others are all optional. You can verify if the callbackuri is valid and reachable by sending a GET request.

Query all Subscriptions

The NFVO can query information about its subscriptions by sending a GET request to the *subscriptions* resource.

Method Type

`GET`

VNFM Endpoint

`/vnffm/v1/subscriptions`

HTTP Request Headers

`Accept:application/json`

For example, to query all alert subscriptions, when the callbackUri is

`http://10.10.1.44:9202/alerts/subscriptions/callback`

`GET`

VNFM Endpoint

`http://localhost:8250/vnffm/v1/subscriptions?callbackUri="http://10.10.1.44:9202/alerts/subscriptions/callback"`

HTTP Request Headers

`Accept:application/json`

The NFVO can use the URI query parameters to filter the results. The following attribute names are supported for the URI query of the subscriptions:

- id
- filter
- callbackUri

Note The URI query parameters are for querying multiple subscriptions only.

Query an Individual Subscription

You must know the subscription ID to query an individual subscription.

Method Type

`GET`

VNFM Endpoint

`/vnffm/v1/subscriptions/{subscriptionId}`

HTTP Request Header

```
Accept:application/json
```

Delete a Subscription

You can delete a subscription if the NFVO does not need it. Send a delete request to the individual subscription.

Method Type

```
DELETE
```

VNFM Endpoint

```
/vnffm/v1/subscriptions/{subscriptionId}
```

HTTP Request Header

```
http://localhost:8250/vnffm/v1/subscriptions/682791f8-34ad-487e-811a-553036bf49b2
```

PART VII

ESC Portal

CHAPTER 21

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. The user session has an expiry of 1 hour so if the user is inactive on the portal for an hour then portal will expire the session and the user will have to re-login. 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.

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 25: 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.

CHAPTER 22

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 36.

Managing OpenStack Resources Using ESC Portal

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

Adding and Deleting Images in ESC Portal

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

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 23

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

Step 1 Deploying using a file—You can upload an existing datamodel file.

Step 2 Creating and deploying a datamodel using a form—A new deployment datamodel is created by filling all the appropriate fields in the ESC portal. After you create a datamodel using the ESC portal, you can either export a deployment datamodel from the ESC portal or deploy the datamodel.

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 Using a Form

To create a new deployment template, do the following:

Procedure

Step 1 Choose **Deployments**.

Step 2 Click + to deploy using a form.

Step 3 Enter a **Deployment name**.

Step 4 Select the tenant name.

Step 5 In the **General** tab, enter the appropriate values for the fields.

a) Click **Enable Smart Licensing** to enable smart licensing.

b) Click **Enable Intragroup Rules** to enable intragroup rules and select **Affinity** to enable affinity rules.

For more information on intragroup affinity rules, see Affinity and Anti-Affinity Rules, on page 121.

Step 6 (Optional): Click the **Add VNF Intergroup Rule** tab to select VNFs for which you the affinity rules to be applicable.

For more information on intergroup affinity rules, see Affinity and Anti-Affinity Rules, on page 121.

Step 7 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 203.

Step 8 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 9 To specify the number of instances of a particular type of VM that needs to be instantiated and to enable elastic scale in and scale out, click the **Scaling** tab.

Step 10 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 189.

Step 11 In the **Config Data** tab, enter the appropriate values for the fields.

Step 12 Do one of the following:

- To export the datamodel to a XML file, click **Export Template**.

- To deploy the datamodel, click **Deploy Template**.

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 121.

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 121.

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 203.

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 189.

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 24

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.

Step 3 Click the downloadable file to save it on your machine.

PART VIII

Administering ESC

CHAPTER 25

Monitoring ESC Health

You can monitor the health of ESC and its services, using one of the following:

- Monitoring Health API
- SNMP Trap

Monitoring the Health of ESC Using REST API

ESC provides REST API for any third party software to monitor the health of ESC and its services. Using the API, the third party software can query the health condition of ESC periodically to check whether ESC is in service. In response to the query, API provides status code and messages, see Table 26: ESC Health API Status Code and Messages , on page 283 for details. In an HA setup the virtual IP (VIP) must be used as the monitoring IP. The return value provides the overall condition of the ESC HA pairs. See the Table 27: Health API Status Messages for Standalone ESC and HA, on page 285 for details.

The REST API to monitor the health of ESC is as follows:

```
GET to https://<esc_vm_ip>:60000/esc/health
```

Note The monitoring health API is secured using the existing REST basic HTTP authentication. The user can retrieve the report by using the ESC REST API credentials.

The monitoring health API response is as follows:

```
<?xml version="1.0" encoding="UTF-8" ?>
<esc_health_report>
    <status_code>2000</status_code>
    <message>ESC services are running</message>
</esc_health_report>
```

XML and JSON responses are also supported for the monitoring health API.

The status code and messages below provide the health condition of ESC. The status codes with 2000 series imply that the ESC is operational. The status codes with 5000 series imply that at least one ESC component is not in service.

Note The ESC Health API does not check the VIM status because of multi VIM deployment introduced in ESC Release 3.0.

Table 26: ESC Health API Status Code and Messages

Status Code	Message
2000	ESC services are running.
2010	ESC services are running, but ESC High Availability node is not reachable.
2020	ESC services are running. One or more VIM services (for example, keystone and nova) not reachable. **Note** Not supported from ESC Release 3.0.
2030	ESC services are running, but VIM credentials are not provided. **Note** Not supported from ESC Release 3.0.
2040	ESC services running. VIM is configured, ESC initializing connection to VIM.
2100	ESC services are running, but ESC High-Availability node is not reachable. One or more VIM services (for example, nova) are not reachable. **Note** Not supported from ESC Release 3.0.
5010	ESC service, ESC_MANAGER is not running.
5020	ESC service, CONFD is not running.
5030	ESC service, MONA is not running.
5040	ESC service, VIM_MANAGER is not running.
5090	More than one ESC service (for example, confd and mona) are not running.

Note ESC HA mode refers to ESC HA in DRBD setup only. For more information on the ESC HA setup, see the Cisco Elastic Services Controller Install Guide.

The table below describes the status message for standalone ESC and HA with success and failure scenarios. For more information on ESC standalone and HA setup, see the Cisco Elastic Services Controller Install Guide.

Table 27: Health API Status Messages for Standalone ESC and HA

	Success	Partial Success	Failure
Standalone ESC	The response is collected from the monitoring health API and the status code is 2000.	NA	• Monitor cannot get the response from the monitoring health API. • The response is collected from the monitoring health API and the status code returned is in the 5000 series.

	Success	Partial Success	Failure
ESC in HA	The response is collected from the monitoring health API and the status code is 2000.	The response is collected from the monitoring health API and the status code is 2010. This indicates that the ESC standby node cannot connect to ESC master node in ESC HA. However, this does not impact the ESC service to northbound.	• The monitor cannot get the response from the monitoring health API for more than two minutes. **Note** ESC monitoring health API may not be available for a certain period during the HA switchover period. The monitoring software must set a proper threshold to report service failure in this scenario. • The response is collected from the monitoring health API and the status code returned is in the 5000 series.

Monitoring the Health of ESC Using SNMP Trap Notifications

You can also configure notifications on the health of various ESC components via SNMP traps using an SNMP Agent. This Agent is installed as part of the standard ESC installation and supports the SNMP version 2c protocol. The SNMP traps currently support only the state of the ESC product and not of the VNFs managed by ESC. This section describes the steps required to configure the ESC SNMP agent and also cover the events that will be triggered as part of the notifications.

Before you begin

- Ensure the **CISCO-ESC-MIB** and **CISCO-SMI MIB** files are available on your system. These are located in the /opt/cisco/esc/snmp/mibs directory. Download these to your SNMP Manager machine and place them in the $HOME/.snmp/mibs directory.
- Configure SNMP Agent. There are three methods to configure SNMP agent. These methods are discussed in detail in the section below.

Configuring SNMP Agent

In order to receive the SNMP traps, configure the SNMP Agent parameters. The agent can be configured using three different methods described in this section. The best or most applicable method to use depends on your use case.

Procedure

- **Configuring SNMP Agent during the installation of ESC via BootVM**: While installing ESC, use the following additional parameters to configure SNMP agent:

```
% bootvm.py <esc_vm_name> --image <image-name> --net <net-name> --enable-http-rest
--ignore-ssl-errors
--managers "udp:ipv4/port" or "udp:[ipv6]/port"
```

- **Configuring via ESCADM**: Using the escadm tool, you can modify the SNMP agent configuration parameters such as managers and ignoreSslErrors properties.

```
sudo escadm snmp set --ignore_ssl_errors=true --managers="udp:ipv4/port" or
"udp:[ipv6]/port"
```

- **Updating the configuration file**: The configuration is in the file /opt/cisco/esc/esc_database/snmp.conf. This file is in JSON format. Following is an example:

```
{"sysDescr": "ESC SNMP Agent",
    "listeningPort": "2001",
    "managers": [
        "udp:[ipv4]/port",
        "udp:[ipv6]/port"
    ],
    "ignoreSslErrors": "yes",
    "logLevel": "INFO",
    "sysLocation": "Unspecified",
    "sysName": "system name",
    "pollSeconds": "15",
    "listeningAddress": "0.0.0.0",
    "healthUrl": "https://<esc_vm_ip>:60000/esc/health",
    "sysContact": "root@localhost"}
```

The table below describes the properties that can be configured.

Note Using bootvm and escadm tool, you can only configure ignoreSslErrors and Managers parameters.

Table 28: Agent Configuration Parameters

Variable	Description
listeningAddress	Specify the network interface to listen on. 0.0.0.0 means all interfaces.
listeningPort	Specify the UDP port to listen on
ignoreSslErrors	Specify as no if you want the certificates to be validated for SSL connections. For untrusted self-signed certificates, set to yes.
healthUrl	Specify the URL of the Health Monitor API. **Note** The Agent may be used outside the ESC machine, such as a laptop with a Java-8 installed. In that case, the health URL should point to the ESC machine, and provide an ESC username/password (authUser/authPass) to authenticate with the ESC Monitor.
authUser	Specify the HTTP-BASIC username for the Health Monitor API. Use this parameter only if the agent is external to the ESC machine.
authPass	Specify the HTTP-BASIC password for the Health Monitor API. Use this parameter only if the agent is external to the ESC machine.
managers	Specify an array of strings in "udp:ipv4/port" or "udp:[ipv6]/port" format of where SNMP traps are to be delivered to. If these are changed when the Agent is running, the configuration is reloaded and the new managers used for future traps.
pollSeconds	This parameter is used by the scheduler. Use this paramter to specify the number of seconds between polls to the Health Monitor API.
logLevel	Specify the levels as INFO, ERROR, WARN, DEBUG, TRACE, ALL and OFF. This parameter can only be changed at runtime.
sysName	(SNMPv2-MIB) Optional value indicating the name of this node. The hostname is used by default.
sysDescr	(SNMPv2-MIB) Optional value describing this agent.
sysLocation	(SNMPv2-MIB) Optional value giving the physical location of this node.
sysContact	(SNMPv2-MIB) Optional value giving a contact name and/or email address for enquiries regarding this node

Defining ESC SNMP MIBs

The following table describes the content of ESC MIB. These values are configurable in snmp.conf file.

Variable	Simple IOD	Description
sysName	SNMPv2-MIB::sysName.0	Specify the name of the ESC machine. The host name is taken by default.
sysDescr	SNMPv2-MIB::sysDescr.0	Specify the name of the SNMP Agent.
sysLocation	SNMPv2-MIB::sysLocation.0	Specify where the ESC machine is located.
sysContact	SNMPv2-MIB::sysContact.0	Specify the Admin contact.

Enabling SNMP Trap Notifications

Use the escadm tool to start the SNMP services.

```
sudo escadm snmp start
```

You can also use esadm tool to stop, get the status, and modify the configurations of the SNMP agent.

```
sudo escadm snmp stop
sudo escadm snmp status
sudo escadm snmp restart
```

Managing SNMP Traps in ESC

This section covers:

- Understanding the SNMP Notification Types in ESC.
- Receiving SNMP Trap Message Directly From the Network
- Managing Trap Endpoints (SNMP Managers)
- Managing ESC SNMP in an HA Environment
- Managing Self-Signed Certificates in ESC

Procedure

- **Understanding the SNMP Notification Types in ESC:** The following table lists all the events supported by this version of the SNMP agent. These status codes and messages will be returned via a SNMP trap to a registered manager only when there is a change of state of ESC. The status codes with 2000 series imply that the ESC is operational. The status codes with 5000 series imply that at least one ESC component is not in service. For more details on status codes with 2000 series and 5000 series, see section, *Monitoring ESC Health Using REST API*.

Status Code	SNMP Agent-specific Message
5100	An HTTP error was received when using the ESC Monitor API

Status Code	SNMP Agent-specific Message
5101	The ESC Monitor replied, but the data could not be understood.
5102	The Agent could not create a network connection to the ESC Monitor API.
5199	An unhandled error occurred (details will be included in the message).
5200	In an HA environment when there are changes to the HA master node the agent will send this notification

- **Receiving SNMP trap messages directly from the network**: Directly receive SNMP trap messages from the network, by using basic SNMP UNIX tools such as, snmpget snmpwalk and snmptrapd. An example usage:

```
snmptrapd -m ALL -f -Lo -c snmptrapd.conf <port>
```

This will start an SNMP trap daemon on port 12113. Make sure the Cisco and ESC MIB's are present in ~/.snmp/mibs. The referenced snmptrapd.conf looks like this:

```
disableAuthorization yes
authCommunity   log,execute,net public
# traphandle default /Users/ahanniga/bin/notify.sh esc

createUser myuser MD5 mypassword DES myotherpassword

format2 %V\n% Agent Address: %A \n Agent Hostname: %B \n Enterprise OID: %N \n Trap
Sub-Type: %q \n Community/Infosec Context: %P \n Uptime: %T \n PDU Attribute/Value Pair
 Array:\n%v \n -------------- \n
```

The trap will contain two entries: statusCode and statusMessage. The trap will be sent when the status changes

```
DISMAN-EVENT-MIB::sysUpTimeInstance = Timeticks: (3971) 0:00:39.71
SNMPv2-MIB::snmpTrapOID.0 = OID: CISCO-ESC-MIB::statusNotif
SNMPv2-MIB::sysDescr.0 = STRING: ESC SNMP Server
CISCO-ESC-MIB::escStatusCode.0 = STRING: "2000"
CISCO-ESC-MIB::escStatusMessage.0 = STRING: "ESC services are running."
```

- **Managing Trap Endpoints (SNMP Managers)**: The SNMP Agent monitors its configuration file for changes and reloads when a change is made. Add or remove manager endpoints to the configuration file and the new configuration will be used in future traps.
- **Managing ESC SNMP Agent in an HA Environment**: Two or more ESC nodes can be deployed in a HA configuration and the SNMP agent does support this configuration. However, consider the following points in an HA deployment:
  - Only one ESC node (the master node) can send SNMP traps
  - The SNMP Agent must be up if the backup node becomes the master.
  - Any changes made to the master configuration must also be applied on backup nodes.
  - If a node becomes the master node due to failover, this will generate a trap.

- **Managing Self-Signed Certificates**: When ESC is deployed and the SNMP agent uses ESC Health APIs, it is recommended that a root trusted certificate is installed on the server. If the environment is a known and trusted one then it is possible to ignore these errors using the configuration parameter "ignoreSslErrors". However, if you did want to keep this setting to its more secure default it is possible to install a self-signed certificate by importing the ESC certificate into the JVM trust store. The following section describes the procedure to do so.
  a) Add esc as an alternative name for localhost. In the file "/etc/hosts:" add the following (or ensure that "esc" is added to the end):

     ```
     127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4 esc
     ```

  b) In the SNMP Agent configuration file "/opt/cisco/esc/esc_database/snmp.conf" the healthUrl must point to esc.

     ```
     "healthUrl": "https://esc:60000:/esc/health"
     ```

  c) Import the certificate into the truststore. Following is an example of importing the certificate, assuming $JAVA_HOME is/usr/lib/jvm/jre-1.8.0-openjdk.x86_64:

     ```
     cd /opt/cisco/esc/esc-config
     sudo openssl x509 -inform PEM -in server.pem -outform DER -out server.cer
     sudo keytool -importcert -alias esc -keystore $JAVA_HOME/lib/security/cacerts
     -storepass changeit -file server.cer
     ```

CHAPTER 26

ESC System Logs

ESC System Logs

Log messages are created for ESC events throughout the VNF lifecycle. These can be external messages, messages from ESC to other external systems, error messages, warnings, events, failures and so on. The log file can be found at `/var/log/esc/escmanager_tagged.log`.

The log message format is as follows:

```
date=<time-date>] [loglevel=<loglevel>] [tid=<transactionid>] [cl=<classifications>]
[tags=<tags>] [msg=<message>
```

Sample log is as follows:

```
date=15:43:58,46022-Nov-2016]
[loglevel=ERROR ] [tid=0793b5c9-8255-47f3-81e6-fbb59f6571f7] [cl=OS ]
[tags=wf:create_vm,eventType:VM_DEPLOY_EVENT,tenant:CSCvd94541,depName:test-dep,vmGrpName:test-VNF,
vmName:test-dep_test_0_dc3f406c-05ca-43b3-af21-0841e3b029a0]
[tags=wf:create_vm,eventType:VM_DEPLOY_EVENT,tenant:test,depName:test-dep,vmGrpName:test-VNF,
vmName:test-dep_test_0_dc3f406c-05ca-43b3-af21-0841e3b029a0] [msg=sleepingfor5seconds
to allow vm to become ACTIVE instance id:
162344f7-78f9-4e45-9f23-34cf87377fa7
name:test-dep_test_0_dc3f406c-05ca-43b3-af21-0841e3b029a0
```

When a request is received, a RequestDetails object is created which autogenerates a unique transaction id. This value is carried forward across all threads. Classifications and tags are optional. These are prefixes added to the log messages to enhance readability, and help in debugging. With classifications and tags, the log messages can be easily parsed and filtered by the log analysis tools.

The following classifications are supported:

NBI	"com.cisco.esc.rest""com.cisco.esc.filter"(North Bound Interface - Clientinterface)
SBI	"com.cisco.esc.rest"- source is a callback handler or"EventsResource"(South Bound Interface - i.e. between ESC and the VIM)
SM	"com.cisco.esc.statemachines". stands for StateMachine. This classification indicates logs in the StateMachine category.

MONITORING	"com.cisco.esc.monitoring""com.cisco.esc.paadaptor"(MONA related logs)
DYNAMIC_MAPPING	"com.cisco.esc.dynamicmapping""com.cisco.esc.db.dynamicmapping"(MONA related logs)
CONFD	"com.cisco.esc.confd"
CONFD_NOTIFICATION	"com.cisco.esc.confd.notif""com.cisco.esc.confd.ConfdNBIAdapter"
OS	"com.cisco.esc.vim.openstack"
LIBVIRT	"com.cisco.esc.vim.vagrant
VIM	"com.cisco.esc.vim"
REST_EVENT	"ESCManager_Event""com.cisco.esc.util.RestUtils". indicates REST notifications in logs.
WD	"com.cisco.esc.watchdog"
DM	"com.cisco.esc.datamodel""com.cisco.esc.jaxb.parameters"(Datamodel and resource objects)
DB	"com.cisco.esc.db"(Database related logs)
GW	"com.cisco.esc.gateway"
LC	"com.cisco.esc.ESCManager"(Start up related logs)
SEC	"com.cisco.esc.jaas"
MOCONFIG	"com.cisco.esc.moconfig"(MOCONFIG object related logs --thisis internalforESC developers)
POLICY	"com.cisco.esc.policy"(Service/VM Policy related logs)
TP	"com.cisco.esc.threadpool"
ESC	"com.cisco.esc" Any other packages not mentioned above

The following tags are supported:

- **Workflow [wf:]**—Generated using action and resource from RequestDetails object. Example "wf:create_network"
- **Event type [eventType:]**—Event that triggered the current action. Example: "eventType:VM_DEPLOY_EVENT"
- **Resource based**—These values are generated based on the type of parameter used by the event. The hierarchy, that is, the tenant, the vm group and so on is added to the log.

Tenant	[tenant:<tenant name>]
Network	[tenant:<tenant id>, network:<network name>] **Note** The tenant appears only if applicable.

Subnet	[tenant:<tenant name or id>, network:<network name or id>, subnet:<subnet name>] **Note** The tenant appears only if applicable.
User	[tenant:<tenant name>, user:<user name or id>] **Note** The tenant appears only if applicable.
Image	[image:<image name>]
Flavor	[flavor:<flavor name>]
Deployment	[tenant:<tenant name or id>, depName:<deployment name>]
DeploymentDetails	[tenant:<tenant name or id>, depName:<deployment name>, vmGroup:<vm group name>, vmName:<vm name>]
Switch	[tenant:<tenant name or id>, switch:<switch name>]
Volume	[volume:<volume name>]
Service	[svcName:<Service Registration name>]

Further, ESC logs can also be forwarded to an rsyslog server for further analysis and log management.

Filtering Logs Using Confd APIs

You can query and retrieve logs (for example, deployment logs, or error logs) in ESC using log filters introduced in the confd APIs. New filters for Tenant, Deployment Name, and VM Name are introduced. This enables you to query the ESC logs further for most recent error logs using the log filters in Confd APIs. You can also retrieve ESC logs related to the communication between ESC and the OS (by setting the classification tag to "OS").

The log format to retrieve confd API logs:

```
date=<time-date>] [loglevel=<loglevel>] [tid=<transactionid>] [cl=<classifications>]
[tags=<tags>] [msg=<message>
```

The sample log is as follows:

```
date=15:43:58,46022-Nov-2016] [loglevel=ERROR ] [tid=0793b5c9-8255-47f3-81e6-fbb59f6571f7]
 [cl=OS ]
[tags=wf:create_vm,eventType:VM_DEPLOY_EVENT,tenant:test,depName:test-dep,vmGrpName:test-VNF,
vmName:test-dep_test_0_dc3f406c-05ca-43b3-af21-0841e3b029a0]
[msg=sleepingfor5seconds to allow vm to become ACTIVE instance id:
162344f7-78f9-4e45-9f23-34cf87377fa7 name:test-dep_test_0_dc3f406c-05ca-43b3-af21-0841e3b029a0
```

The parameters for log level, classification and tags are dependent on each other to retrieve the logs. You can successfully retrieve the logs with the following combination.

- log_level=ERROR, classifications=OS, tags=(depName:test-dep)
- log_level=ERROR, classifications=OS, tags=(tenant: test)

The log filter returns a value when all the following conditions are met:

- Log level

- Classifications (if provided)
- Tags (if provided)

Note If there are more than one classification listed, it has to match at least one of the classifications. The same applies to the tags as well.

For example, the following log filter criteria does not return the log sample mentioned earlier:

```
log_level=ERROR, classifications=VIM, tags=(depName:test-dep)
```

It does not return any value though the log level and tags match, the classification VIM does not match.

The data model is as follows:

```
rpc filterLog {
    description "Query and filter escmanager logs using given parameters";
    tailf:actionpoint escrpc;
    input {
      leaf log_level {
        mandatory false;
        description "One of DEBUG / INFO / WARNING / ERROR / TRACE / FATAL. Results will
include all logs at and
                     above the level specified";
        type types:log_level_types;
        default ERROR;
      }
      leaf log_count {
        mandatory false;
        description "Number of logs to return";
        type uint32;
        default 10;
      }
      container classifications {
        leaf-list classification {
         description "Classification values to be used for the log filtering.  For example:
 'OS', 'SM'.
                      Logs containing any of the provided classification values will be
returned.";
          type types:log_classification_types;
        }
      }
      container tags {
        list tag {
          key "name";
          leaf name {
            mandatory true;
            description "Tag name to be used for the log filtering.  For example: 'tenant',
 'depName'.
                         Logs containing any of the provided tag name plus the tag values
will be returned.";
            type types:log_tag_types;
          }
          leaf value {
            mandatory true;
            description "Tag value pairs to be used for the log filtering.  For example:
'adminTenant', 'CSRDeployment'";
            type string;
          }
        }
      }
```

```
  }
  output {
    container filterLogResults {
      leaf log_level {
        description "Log level used to filter for the logs.";
        type types:log_level_types;
      }
      list logs {
        container classifications {
          leaf-list classification {
            description "Classifications used to filter for the logs.";
            type types:log_classification_types;
          }
        }
        container tags {
          list tag {
            key "name";
            leaf name {
              mandatory true;
              description "Tag name used to filter for the logs.";
              type types:log_tag_types;
            }
            leaf value {
              mandatory true;
              description "Tag value used to filter for the logs.";
              type string;
            }
          }
        }
        leaf log_date_time {
          description "Timestamp of the log.";
          type string;
        }
        leaf log_message {
          description "The log message.";
          type string;
        }
      }
    }
  }
}
```

You can query for the confd API logs through the netconf console or esc_nc_cli

- Through the netconf-console, run the following query:

  ```
  /opt/cisco/esc/confd/bin/netconf-console --port=830 --host=127.0.0.1 --user=admin
  --privKeyFile=/home/admin/.ssh/confd_id_dsa --privKeyType=dsa --rpc=log.xml
  ```

- Using the esc_nc_cli, run the following query:

  ```
  ./esc_nc_cli filter-log log.xml
  ```

  The sample log.xml is as follows:

  ```
  <filterLog xmlns="http://www.cisco.com/esc/esc">
    <log_level>INFO</log_level>
    <log_count>1</log_count>
    <classifications>
      <classification>OS</classification>
      <classification>SM</classification>
    </classifications>
    <tags>
      <tag>
        <name>depName</name>
```

```
      <value>CSR_ap1</value>
    </tag>
    <tag>
      <name>tenant</name>
      <value>admin</value>
    </tag>
  </tags>
</filterLog>
```

The response is as follows:

```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="1">
  <filterLogResults xmlns="http://www.cisco.com/esc/esc">
    <log_level>INFO</log_level>
    <logs>
      <classifications>
        <classification>OS</classification>
        <classification>SM</classification>
      </classifications>
      <tags>
        <tag>
          <name>depName</name>
          <value>CSR_ap1</value>
        </tag>
        <tag>
          <name>tenant</name>
          <value>admin</value>
        </tag>
      </tags>
      <log_date_time>13:06:07,575 31-Oct-2016</log_date_time>
      <log_message> No pending work flow to start.</log_message>
    </logs>
  </filterLogResults>
</rpc-reply>
```

Note The logging API responses are in XML format. If the log messages contain any XML characters, then the characters will be escaped so not to break the XML conformance.

Viewing ESC Log Files

You can find the logs of various ESC components here:

File	Component	Description	Rotation Size	Number of backup files
/var/log/esc/escmanager.log	ESCManager	This contains logs of the ESC Manager which includes workflow, request and persistence.	150 MB	10

File	Component	Description	Rotation Size	Number of backup files
/var/log/esc/error_escmanager.log	ESCManager	This is the same as escmanager.log but with a format that is easier to read for netconf logging API but can be used by other parsers if needed.	150 MB	10
/var/log/esc/event_escmanager.log	ESCManager		150 MB	10
/var/log/esc/yangesc.log	ESCManager	This contains logs related to our netconf request and notifications	150 MB	10
/var/log/esc/debug_yangesc.log	ESCManager		51MB	2
/var/log/esc/mona/mona.log	MONA		150 MB	10
/var/log/esc/mona/actions_mona.log	MONA		150 MB	10
/var/log/esc/mona/rules_mona.log	MONA		150 MB	10
/var/log/esc/vimmanager/vimmanager.log	VIM Manager Service	A detailed VIM manager log.	150 MB	10
/var/log/esc/vimmanager/operations_vimmanager.log	VIM Manager Service	A simplified log which only lists the VIM Manager operations been processed.	150 MB	10
/var/log/esc/vimmanager/vim_vimmanager.log	VIM Manager Service	Raw HTTP request/response (including header) between VIM Manager and VIM. Note, for OpenStrack to track this log, log level has to set to DEBUG for VIM. Manager.	150 MB	10
/var/log/esc/<timestamp>-esc-portal-be.log	ESC Portal		10 MB	4
/var/log/esc/confd/audit.log	confd		10 MB	4
/var/log/esc/confd/browser.log	confd		10 MB	4
/var/log/esc/confd/confd.log	confd		10 MB	4

File	Component	Description	Rotation Size	Number of backup files
/var/log/esc/confd/devel.log	confd		10 MB	4
/var/log/esc/confd/netconf.log	confd		10 MB	4
/var/log/esc/confd/netconf.trace	confd		10 MB	4
/var/log/esc/confd/cli-history/admin.hist				
/var/log/esc/confd/cli-history/global.data				
`/var/log/esc/esc_haagent.log`	ESC INFRA or HA		10 MB	4
/var/log/esc/ esc_monitor.log	ESC INFRA or HA		10 MB	4
/var/log/esc/ esc_monitor_output.log	ESC INFRA or HA		10 MB	4
/var/log/esc/esc_confd.log	ESC INFRA or HA		10 MB	4
/var/log/esc/ pgstartup.log	ESC INFRA or HA		10 MB	4
/var/log/esc/spy.log	ESC INFRA or HA		No logs (size 0)	No ESC generated logs.
/var/log/tomcat6/catalina.out	Tomcat		Not rated	No ESC generated logs. Only Error.
/var/log/esc/esc_dbtool.log				
/var/log/esc/snmp/snmp.log				
/var/log/esc/etsi-vnfm/etsi-vnfm.log	ETSI-Service	This is the main log file for ETSI processing, including requests, response, payloads and general logging information where appropriate.	150MB	10

File	Component	Description	Rotation Size	Number of backup files
/var/log/esc/etsi-vnfm/events-etsi-vnfm.log	ETSI-Service	Logs only API requests, both entering and leaving.	150MB	10
/var/log/esc/etsi-vnfm/event-details-etsi-vnfm.log	ETSI-Service	Logs both the API requests (entering and leaving) along with the actual JSON payloads.	150MB	10
/var/log/esc/escadm.log	escadm	Logs to capture both manual and automated messages and errors from escadm.py. This log is useful to track startup and configuration changes to ESC.		

APPENDIX A

ESC Error Conditions

Error Conditions for ESC Operations

Error Conditions for ESC Operations

If an operation fails in ESC, the user must cancel that operation. ESC will not rollback automatically to cancel any operations. The table below shows the error condition, and recovery details.

Notifications or Logging details for Error Conditions

Typically, for all error conditions, an error notification of the failed request will be sent to the NB client (ESC User) through callback if using REST interface, or through netconf notification if using NETCONF interface. An error log will be generated and sent to syslog, if syslog is configured.

Error Condition	Recovery
Failed create tenant request	NB client (ESC User) has to send in a delete tenant request before attempting to send in the same create tenant request.
Failed create network request	NB client (ESC User) has to send in a delete network request before attempting to send in the same create network request.
Failed create subnet request	NB client (ESC User) has to send in a delete subnet request before attempting to send in the same create subnet request.
Failed deployment request	NB client (ESC User) has to send in an undeploy request before attempting to send in the same deploy request If a deployment fails, ESC updates information in its database (with error state) until it receives an undeployment request. The undeployment will remove objects that are in error states.

Error Condition	Recovery
Failed Recovery	The existing deployment is not usable anymore. NB client (ESC User) has to send in an undeploy request then the same deploy request.
Failed Scale Out/In	No action required. The existing deployment is still functional. If at a later stage an undeploy was triggered, it will clean up any VMs that were affected part of the failed scale out and scale in.
Failed Service Update	No action required. The existing deployment is still functional. Any retries of that update will not be honored. If at a later stage an undeploy was triggered, it will clean up any created VMs part of the failed update.
Failed VM Operations (Start, Stop, Reboot, Enable Monitor, Disable Monitor)	No action required. The existing deployment is still functional. NB client (ESC User) can retry the failed operation.
Failed VNF/Service Operations (Start, Stop, Reboot, Enable Monitor, Disable Monitor)	No action required. The existing deployment is still functional. NB client (ESC User) can retry the failed operation.
Failed delete tenant request	Possibility of leaking resource in VIM. Manual intervention might be needed to clean up leaking resources on VIM.
Failed delete network request	Possibility of leaking resource in VIM. Manual intervention might be needed to clean up leaking resources on VIM.
Failed delete subnet request	Possibility of leaking resource in VIM. Manual intervention might be needed to clean up leaking resources on VIM.
Failed undeployment request	Possibility of leaking resource in VIM. Manual intervention might be needed to clean up leaking resources on VIM