## CONTENTS

### CHAPTER 1

**Preface**  
- **Purpose** 1  
- **Audience** 1  
- **Obtaining Documentation Request** 1

### CHAPTER 2

**Elastic Services Controller Overview** 3  
- **Elastic Services Controller Overview** 3  
- **Key Features of Elastic Services Controller** 4  
- **ESC Architecture** 4  
- **Elastic Services Controller Usage Scenarios** 5

### CHAPTER 3

**Elastic Services Controller Interface** 7  
- **Elastic Services Controller NB APIs** 7  
  - **NETCONF/YANG Northbound API** 7  
  - **REST Northbound API** 8  
- **Elastic Services Controller Portal** 9

### CHAPTER 4

**Creating Resources in ESC** 13  
- **Creating an Image** 13  
- **Creating a Flavor** 14  
- **Creating a Network** 16  
  - **Creating Subnets** 17  
- **Creating Tenants** 18

### CHAPTER 5

**Registering Virtual Network Functions** 19  
- **Registering Virtual Network Functions** 19  
- **Unregistering Virtual Network Functions** 20
Preface

• Purpose, page 1

Purpose

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

Audience

Network administrators responsible for provisioning, configuring, and monitoring VNFs. Elastic Services Controller (ESC) and its VNFs are deployed in a Virtual Infrastructure Manager (VIM). Currently, OpenStack is the VIM used. The administrator must be familiar with the VIM layer, and OpenStack resources and commands used.

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:

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.
Elastic Services Controller Overview

Cisco Elastic Services Controller (ESC) is a Virtual Network Functions Manager (VNFM), performing life cycle management of Virtual Network Functions (VNFs). ESC provides agentless and multi vendor VNF management by provisioning the virtual services, and monitoring their health and load. ESC provides the flexibility to define rules for monitoring, and associate actions to be triggered based on the outcome of these rules. Based on the monitoring results, ESC performs scale in or scale out on the VNFs. It also supports automatic VM recovery when a VM fails.

ESC fully integrates with Cisco and other third party applications. As a standalone product, the Elastic Services Controller can be deployed as a VNF Manager. ESC integrates with Cisco Network Services Orchestrator (NSO) to provide VNF management along with orchestration. Elastic Services Controller (ESC) as a VNF Manager targets the virtually managed services and all service provider NFV deployments such as virtual video, WiFi, authentication and others.

Complex services include multiple VMs that are orchestrated as a single service with dependencies between them. These multiple VMs are managed as a single entity, for example, Virtually Managed Services (vMS) 1.0.

ESC Terminology Usage

The following terms are used throughout the document:

- **Virtual Machine (VM)**—A virtual machine (VM) is an operating system OS or application installed on a software which imitates dedicated hardware. The end user has the same experience on a virtual machine as they would have on dedicated hardware.

- **Virtual Network Function (VNF)**—A VNF consists of a single or a group of VMs with different softwares and processes that can be deployed on a Network Function Virtualization Infrastructure (NFVI).

- **Virtual Network Function Manager (VNFM)**—Manages the life cycle of a VNF.
• Network Function Virtualization Orchestrator (NFVO)—An ETSI NFV MANO standard term. The orchestrator communicates with ESC APIs to perform all the VNFM functions.

• Service—A Service consists of single or multiple VNFs.

• Virtual Infrastructure Manager (VIM)—The Virtualized Infrastructure Manager 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.

• 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.

Target Users

• Service Providers (SPs)—Helps SPs reduce cost of operating the networks by providing effective and optimal resource usage.

• Small-to-Medium Businesses (SMBs)—Helps meet the needs of single- and small-to-medium businesses to administer and control network functions. Cisco Elastic Services Controller automates provisioning, configuring and monitoring of network functions. Small-to-medium businesses can benefit by leveraging some of the basic services such as DNS, DHCP and NTP.

Key Features of Elastic Services Controller

• Provides open and modular architecture, which allows multi-vendor OSS, 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 life cycle 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 or removes VMs based on the monitoring errors and threshold conditions detected as part of healing (also called as recovery).

• Supports service agility by providing faster VNF deployment and life cycle management.

• Supports multi-tenant environments.

• Supports REST and NETCONF / YANG interfaces to provide better hierarchical configuration and data modularity.

ESC Architecture

Elastic Services Controller (ESC) provides open and modular architecture, which allows OSS, and multi-vendor support. ESC performs life cycle management of the VNFs, that is, VNF onboarding, configuring the VNFs,
monitoring them, and making VNF level life cycle decisions such as healing and scaling based on the KPI requirements. Elastic Services Controller (ESC) and its managed VNFs are deployed as VMs running within a Virtual Infrastructure Manager (VIM). Currently, OpenStack is supported. The ESC core engine manages transactions, validations, policies, workflows, VM state machines and rollbacks. 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 and NETCONF/YANG NB APIs. The orchestration layer can be a Cisco NSO or any third party OSS. Confd enables integration with NSO by adding NETCONF/YANG north bound interface support. A configuration template, Virtual Network Function Descriptor (VNFD) file is used to describe the deployment parameters and operational behaviors of the VNFs. This VNFD is used in the process of onboarding a VNF and managing the life cycle of a VNF instance. Figure 1: Cisco Elastic Services Controller Architecture represents the Elastic Services Controller architecture.

**Figure 1: Cisco Elastic Services Controller Architecture**

![Cisco Elastic Services Controller Architecture Diagram]

Elastic Services Controller Usage Scenarios

ESC can be deployed in the following ways:
• As part of the Cisco Orchestration Suite, ESC is packaged with Cisco Network Service Orchestrator (NSO), and available within Cisco Solutions such as Virtually Managed Services (vMS) and Cisco Cloud Services (CCS).

• As a standalone product, ESC is available as a VNFM bundled with Cisco VNFs such as VPN, vRouter, vSecurity and many others.
Elastic Services Controller Interface

- Elastic Services Controller NB APIs, page 7
- Elastic Services Controller Portal, page 9

Elastic Services Controller NB APIs

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

NETCONF is a network management protocol to install, manipulate, operate and delete the configuration of network devices. ESC supports NETCONF northbound interface with YANG based data models. ESC uses NETCONF to configure and manage the network and its devices. Confd enables integration with NSO by adding NETCONF/YANG northbound interface support.

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.

Examples of NETCONF/YANG API are as follows:

NETCONF request to create a Tenant,

```xml
<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 will be sent to Netconf subscribers once the configuration activation has completed. This indicates that the activation workflow is complete and the configuration resource has been successfully created in the VIM.

**NETCONF notification upon successful creation of a tenant**

```
<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 ESC NetConf API guide.

---

**REST Northbound API**

REST is a simple stateless architecture. When a web service uses this architecture, it is called REST API. REST runs over HTTP. The REST calls are asynchronous.

**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>Elie</name>
  <enabled>true</enabled>
  <description>A description...</description>
</tenant>
```
REST response upon successful creation of tenant:

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>Elie</name>
<enabled>true</enabled>
<description>A description...</description>
</tenant>

For more details on response callback, request parameters, see the REST API guide.

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

Elastic Services Controller Portal

ESC portal is a simplified tool for an ESC administrator to view, sort and filter on various features. As an administrator you can view the real-time activities of ESC such as deploying, undeploying, healing and scaling.

Logging into the Web Portal

The ESC portal provides default admin access. admin can create new users, and assign privileges to them. User profiles can also be updated in the UI.

1. Open your browser and type <ESC_VM_IP>:9000.
2. Enter the username and password, and login to the portal. Admin is the default user account.

For more details on port, and browser requirements see the Cisco Elastic Services Controller Install and Upgrade Guide.

Table below lists the details users can view in the portal:

<table>
<thead>
<tr>
<th>Task</th>
<th>Navigate</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>To view existing deployments</td>
<td>Go to Deployments</td>
<td>Shows all the deployed VMs, and VM groups and their status.</td>
</tr>
<tr>
<td></td>
<td>• To view further details, click View VNFs</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Select KPI data tab to view the KPI details.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Select Rules tab to view the admin rules and actions.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Select Scaling tab to view scaling details.</td>
<td></td>
</tr>
<tr>
<td>Task</td>
<td>Navigate</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>To view tenants</td>
<td>Go to Resources &gt; Tenants</td>
<td>Provides a list of tenants, name, description and tenant ID.</td>
</tr>
<tr>
<td>To view networks, sub-networks and interfaces</td>
<td>Go to Resources &gt; Networks</td>
<td>Details of Network, Sub-network and Interfaces are available on different tabs. You can find details such as name, network ID, tenant ID and so on for each of them.</td>
</tr>
<tr>
<td>To view registered VNFS, VNF Images, and VNF Deployment flavors</td>
<td>Go to Resources &gt; Registration</td>
<td>Details of registered VNFS, VNF images and VNF deployment flavors are all shown in three different tabs.</td>
</tr>
<tr>
<td>To view logs</td>
<td>Go to System &gt; Logs</td>
<td>You will find real-time logs for ESC events throughout VNF lifecycle. In the Setting page, you can filter error, trace, warning, thread activities, rest calls, database, transitions, OpenStack driver and timer messages. Displays real-time logs for ESC events, such as messages from the external systems, messages from ESC to external systems, and some key events such as spin up, spin down, failures and registered services.</td>
</tr>
<tr>
<td>To view incoming requests to ESC</td>
<td>Go to System &gt; Incoming Requests</td>
<td>Lists all the incoming requests to ESC such as Transaction ID and request details.</td>
</tr>
<tr>
<td>To view configurations</td>
<td>Go to System &gt; Configuration</td>
<td>Lists all the configuration parameters used for ESC, and configuration details of VMs, Day-0 configuration and monitoring rules. Shows dynamic information on the CPU usage, and memory utilization.</td>
</tr>
<tr>
<td>To view boot parameters</td>
<td>Go to System &gt; Boot Parameters</td>
<td>Lists all the boot parameters used to boot ESC.</td>
</tr>
<tr>
<td>To view host details</td>
<td>Go to System &gt; Host details</td>
<td>Lists the host details such as Operating System (OS), version of the OS, System up time, RAM, Storage and other details.</td>
</tr>
<tr>
<td>Task</td>
<td>Navigate</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>---------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>To view the health of ESC</td>
<td>Go to System &gt; Health</td>
<td>Show the health of ESC, Confd status, Operational status and other details.</td>
</tr>
<tr>
<td>To view the infrastructure</td>
<td>Go to Infrastructure &gt; Instances</td>
<td>All VMs running on the virtualization infrastructure.</td>
</tr>
<tr>
<td>details</td>
<td></td>
<td></td>
</tr>
<tr>
<td>To view the Hypervisors</td>
<td>Go to Infrastructure &gt; Hypervisors</td>
<td>All hypervisors running on the virtualization infrastructure.</td>
</tr>
<tr>
<td>To register a VNF</td>
<td></td>
<td>Registers a VNF.</td>
</tr>
<tr>
<td>• Go to Resources &gt; Registration</td>
<td></td>
<td>The drag and drop feature allows you to grab an existing registration datamodel and to re-use it by dragging the file to the drop off area.</td>
</tr>
<tr>
<td>• Click on Register VNF(s)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Click on Drop File Here and locate the file or,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Drag and drop your file to the Drop File Here area.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Note</td>
<td></td>
<td>• The Register button will be disabled until the xml file is identified.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Only xml files are accepted.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The drag and drop feature executes a REST call as of now and does not execute NETCONF calls. If you want to delete the registration created through the drag and drop feature then you must do it from either the ESC Portal or directly through REST call. If you mix REST and NETCONF calls then it will cause ESC to get out of sync with the registrations.</td>
</tr>
<tr>
<td>Task</td>
<td>Navigate</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>----------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| To deploy a VNF | • Go to Deployments  
• Click on New Deployment  
• Select Deploy from a file  
• Click on Drop File Here and locate the file or,  
Drag and drop your file to the Drop File Here area. | Deploys a VNF.  
The drag and drop feature allows you to grab an existing registration datamodel and to re-use it by dragging the file to the drop off area.  
**Note**  
• The Deploy button will be disabled until the xml file is identified.  
• Only xml files are accepted  
The drag and drop feature executes a REST call as of now and does not execute NETCONF calls. If you want to delete the registration created through the drag and drop feature then you must do it from either the ESC Portal or directly through REST call. If you mix REST and NETCONF calls then it will cause ESC to get out of sync with the registrations. |
| To un-register a VNF | • Go to Resources > Registration  
• Click on Un-register VNF | Un-registers VNF(s). |
| To un-deploy | • Go to Deployments  
• Click on Un-deploy VNF | Un-deploys VNF(s). |

You can start, stop and check the status of the ESC UI. ssh to the ESC VM, login as super user or sudo su, and execute the following using CLI:

- To start ESC UI, run `start esc_ui`
- To stop ESC UI, run `stop esc_ui`
- To check status of ESC UI, run `status esc_ui`

ESC UI limitations are listed under Limitations section in the Elastic Services Controller 1.0 Release Notes.
Creating Resources in ESC

Starting from Cisco Elastic Services Controller (ESC) Release 1.1, you can create an image or a flavor independent of the service registration process. This allows you to create images and flavors separately. These images and flavors can be used in multiple VNF deployments.

Images and flavors exist independently in OpenStack. You can either use an existing image or flavor during VNF deployment, or create new a image or a flavor in OpenStack to deploy a VNF without registering. For more details, see Deploying Virtual Network Functions with Service Registration, on page 24.

If you have created these resources, you can deploy a VNF without registering it. See Deploying Virtual Network Functions with Service Registration, on page 24.

- Creating an Image, page 13
- Creating a Flavor, page 14
- Creating a Network, page 16
- Creating Tenants, page 18

Creating an Image

In ESC, an image contains bootable file systems that can be used to launch instances. You can use NETCONF/REST interface to create an image. These images can be used to deploy multiple VNFs without registering.

See the Elastics Services Controller Registration and Deployment Attributes page for details.

NETCONF request to create an image:

```xml
<?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:
xmlns>
<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>
</image>
</images>
</esc_datamodel>
```
Creating a Flavor

In ESC, a flavor defines sizes for RAM, disk, and number of cores. You can use NETCONF/REST interface to create a flavor. These flavors can be used to deploy multiple VNFs without registering.

See the Elastics Services Controller Registration and Deployment Attributes page for details.
Creating Resources in ESC

Creating a Flavor

NETCONF request to create an flavor:

```xml
<?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"
<flavors>
  <flavor>
    <name>nashrest-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 an flavor:

```xml
<?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>nashrest-flavor-indep</flavor>
  </escEvent>
</notification>
```

Deleting a Flavor

You can delete an existing flavor using NETCONF or REST interface.

NETCONF request to delete a flavor:

```xml
<?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"
<flavors>
  <flavor nc:operation="delete">
    <name>nashrest-flavor-indep</name>
  </flavor>
</flavors>
</esc_datamodel>
```

NETCONF notification upon successful deletion of an image:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2015-07-13T13:49:08.635+00:00</eventTime>
  <escEvent xmlns="http://www.cisco.com/esc/esc">
    <status>SUCCESS</status>
    <status_message>Flavor deletion completed successfully.</status_message>
    <flavor>nashrest-flavor-indep</flavor>
    <vm_source/>
    <vm_target/>
    <event>
      <type>DELETE_FLAVOR</type>
    </event>
  </escEvent>
</notification>
```
Creating a Network

ESC supports both a tenant network and a provider network.

- **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

ESC also supports Ephemeral networks which are short-lived tenant networks purposely created during unified deployment and exists only till the lifetime of that deployment. For more details, see [Unified Deployment Request](#).

The following example shows how to create a tenant network:

```xml
<?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:

```xml
<?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:

```xml
<?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:ns0="http://www.cisco.com/esc/esc_notifications"
   xmlns:ns3="http://www.cisco.com/esc/esc"
   xmlns="http://www.cisco.com/esc/esc">
  <networks>
    <network>
      <name>leke-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:

```xml
<?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:ns0="http://www.cisco.com/esc/esc_notifications"
   xmlns:ns3="http://www.cisco.com/esc/esc"
   xmlns="http://www.cisco.com/esc/esc">
  <networks>
    <network>
      <name>leke-net-12</name>
      <subnet>
        <name>leke-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>
```

Creating 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 a subnet.

NETCONF request to create a subnet:

```xml
<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_notifications"
         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>
```
Creating Resources in ESC

Creating Tenants

In ESC, a tenant identifies a tenant organization or group that is associated with a set of administrators. When you create tenants, 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 to create a tenant.

NETCONF request to create a tenant:

```xml
<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>
```
Registering Virtual Network Functions

Before you deploy a VNF, you must register an VNF in ESC.

As part of registering or onboarding a VNF, you must add the VNF image and create a flavor for the VNF in VIM (OpenStack). Information related to monitoring or boot configuration is specified at the time of registration. ESC receives request from the NFVO or the user in the form of a descriptor file (VNFD) and registers the VNF. The template applies to both single and composite VNFs. It validates the request before scheduling the registration workflow. After the VNF is registered, ESC sends details back to the NFVO.

Important
Starting from Cisco ESC Release 1.1, you can also deploy a VNF without registering it. However, you must have the necessary resources before you deploy a VNF. For more details on creating resources, see Creating Resources in ESC, on page 13.

You can register once, and use it in other deployments. Depending on the size of VNF you are registering, registration may take time (in minutes).

When ESC is installed, it is not aware of the VMs, or VM types it is going to provision and manage. The XML template, which is the reg.xml file, contains VNF deployment details, day-0 configuration, scripts, monitoring and recovery details, scaling rules and customization parameters. ESC supports registration of Individual VMs, Individual VNFs (with multiple VMs of same type), Composite VNF (with multiple VMs of different type). The XML template must be updated with attributes for ESC to read and register the VNFs. For more details, see the Elastic Services Controller Registration and Deployment Attributes page.

Sample NETCONF registration request is as follows:

```xml
<edit-config xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
    <target>
        <running/>
    </target>
</config>
    <services>
        <service_definition>
            <name>csr-reg</name>
        </service_definition>
    </services>
</esc_datamodel>
```
Unregistering Virtual Network Functions

ESC allows you to unregister or delete an already registered VNF.

Sample NETCONF Request is as follows:

```xml
<rpc message-id="1" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <edit-config :nc="urn:ietf:params:xml:ns:netconf:base:1.0">
        <target>
            <running/>
        </target>
        <config>
            <esc_datamodel xmlns="http://www.cisco.com/esc/esc">
                <services>
                    <service_definition nc:operation="delete">
                        <name>csr-reg</name>
                        <version>1.2</version>
                    </service_definition>
                </services>
            </esc_datamodel>
        </config>
    </edit-config>
</rpc>
```

Upon success, the NETCONF notification is as follows:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
    <eventTime>2015-05-19T19:56:00.167+00:00</eventTime>
    <escEvent xmlns="http://www.cisco.com/esc/esc">
        <status>SUCCESS</status>
        <status_message>Unregister service completed successfully</status_message>
        <svccname>csr-reg</svccname>
        <svccversion>1.2</svccversion>
    </escEvent>
</notification>
```
For more details see the Cisco Elastic Services Controller NETCONF API Guide.

**Note**

- If the registration request has already reached OpenStack, then the unregister request cannot be processed until OpenStack returns the callback to ESC after creating the image.

- If an image or flavor is deleted out of band, and an unregister request is placed, then the image or flavor might leak on OpenStack, depending on which one was deleted out of band first. The ESC database must be cleaned if this happens.
Unregistering Virtual Network Functions
# Deploying Virtual Network Functions

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

<table>
<thead>
<tr>
<th>Scenarios</th>
<th>Description</th>
<th>XML templates</th>
<th>Images and Flavors</th>
<th>Advantages</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Deploying VNFs with Service Registration</strong></td>
<td>This is a two-step procedure:  &lt;br&gt; 1 Service Registration- The reg.xml is used to create the images and flavors  &lt;br&gt; 2 VNF Deployment- The dep.xml is used to deploy VNFs.</td>
<td>• reg.xml  &lt;br&gt; • dep.xml</td>
<td>Images and Flavors are created through ESC by referring to reg.xml.</td>
<td>The images and flavors limited to the service.</td>
</tr>
<tr>
<td><strong>Deploying VNFs without Registration (Creating Images and Flavors through ESC)</strong></td>
<td>Starting from ESC Release 1.1, you can deploy VNFs without service registration process.  &lt;br&gt; 1 VNF Deployment- The dep.xml refers to the images and flavors created and then deploys VNFs.</td>
<td>• dep.xml  &lt;br&gt; • image.xml  &lt;br&gt; • flavor.xml</td>
<td>Images and flavors can be created through ESC by referring to image.xml and flavor.xml</td>
<td>• The images and flavors can be used in multiple VNF deployments.  &lt;br&gt; • You can add or delete images through ESC.  &lt;br&gt; • Upgrade and Downgrade can be done easily.</td>
</tr>
</tbody>
</table>
Advantages
Images and Flavors
XML templates
Description
Scenarios

- The images and flavors can be used in multiple VNF deployments.
- You can add or delete images through ESC.
- Upgrade and Downgrade can be done easily.

Deploying VNFs without Registration (Using pre-existing images and flavors)

Starting from ESC Release 1.1, you can deploy VNFs without service registration process.

1. VNF Deployment:
The dep.xml refers to the pre-existing images and flavors in OpenStack and then deploys VNFs.

XML templates: • dep.xml
Images and Flavors: • Images and Flavors in Openstack
Images and Flavors are not created through ESC

After you have decided on the deployment scenario you want to implement, see the following sections:

- Deploying Virtual Network Functions with Service Registration, page 24
- Deploying Virtual Network Functions without Registration, page 28
- Unified Deployment Request, page 31
- Applying Policies, page 32
- Affinity and Anti-affinity Rule, page 32
- Single Root I/O Virtualization, page 34
- Managing Individual and Composite VNFs, page 35
- Undeploying Virtual Network Functions, page 35

Deploying Virtual Network Functions with Service Registration

When a service is ordered through the NFVO UI, NFVO sends a deploy request to ESC. The request is in the form of a template, consisting of xml payload with names, interfaces, day-0 configuration and so on. During VNF deployment, an instance of the registered VNF is created. During deployment, you can update dep.xml with any new policies, affinity and anti-affinity rules, startup order dependency and single deployment details. See the Elastics Services Controller Registration and Deployment Attributes page for details.

While deploying a VNF, you can either use the existing tenant, network or subnetwork, or create them as part of the VNF deployment request. You can place a single deployment request to create tenant, network and subnetwork along with the VNF. For more details, see Unified Deployment Request.

If you are using the NETCONF / YANG interface, then a success notification is triggered after the deployment is complete. You can also refer to the operational data details to query the list of successful deployments.
If you are using the REST APIs, you will receive callbacks for each VM that gets deployed. You will receive callback at different stages of deployment. A callback is triggered when the VM is being processed for deployment, when the VM is successfully deployed, and when the VM is alive. A final callback is triggered when the deployment of the VM group is completed successfully.

A sample REST Request is as follows:

```
POST /v0/deployments/123 HTTP/1.1
Host: client.host.com
Content-Type: application/xml
Accept: application/xml
Client-Transaction-Id: 123456
Callback:/deployservicecallback
Callback-ESC-Events:/escevents
<datamodel>
  <version>1.0</version>
  <service_definition>
    ...
  </service_definition>
</datamodel>
```

A VNF must be registered before it is deployed. For details, see Registering Virtual Network Functions, on page 19.

ESC reads the dep.xml file to deploy the VNFs.

### Note

A VNF must be registered before it is deployed. For details, see Registering Virtual Network Functions, on page 19.

ESC reads the dep.xml file to deploy the VNFs.

#### Using the Network Universal Unique Identifier (UUID)

You can either use network UUID or name to refer to a network created using ESC or OpenStack.
While using a network name in the interface, if there are networks with duplicate network names, and the tenant that is getting deployed already has a network by that name in its own network pool, then the tenant's own network will take precedence over others and will be used for the deployment. If the deploying tenant does not have a network by that name in its own pool then it will query for any shared networks for that name and if there is only one network by that name then it will use that one otherwise if there are duplicate shared networks then the deployment will be rejected altogether.

If the tenant really wants to deploy using a specific network and might already have a network with the same name as that desired network then the a Network UUID should be passed in the interface instead.

---

**Note**

Networks created through unified deployments are private networks only. ESC rejects creating shared and external networks.

---

**Day-0 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 registration or 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.

**Day-0 in the Configuration datamodel**

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>
```

The url specifies a file on the ESC VM file system or file hosted on report http server.

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 datamodel,

```
<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>
```
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.

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 datamodel, for example, 0, 1, 2, 3

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::1
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_IPV4_OCTETS: 16.0
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: $DS1$h1VxC40U$uf2qLUwGTjHgZplkP78xA
  ```
- To escape a block in the template, insert #[[ and #]]. For example,
  ```
  #[[ passwd: $1$h1VxC40U$uf2qLUwGTjHgZplkP78xA ]]#
  ```

## Deploying Virtual Network Functions without Registration

Starting from Cisco ESC Release 1.1, you can also deploy VNFs without registering. It refers to an image, or a flavor or any resource during deployment. You can also create an image or a flavor. For more details on creating images and flavors, see Creating Resources in ESC, on page 13.

During deployment, ESC looks for the registration details. If ESC is unable to find the registration details for the particular service, it uses the existing flavor and image under the vm_group to continue the deployment. If ESC is unable to find the service registration, image, and flavor details, the deployment fails.

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

## Deployment Update

Starting from Cisco ESC Release 1.1, you can 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 during deployment.

**Note**

- All the deployment updates are supported only when you deploy VNFs without service registration.
- You can 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.

## Adding a VM Group

You can add or delete a vm_group from a running deployment using existing images and flavors.

NETCONF request to add a vm_group:
```xml
<esc_datamodel xmlns="http://www.cisco.com/esc/esc">  <tenants><tenant>  <name>Admin</name>  </tenant>  </tenants>  </esc_datamodel>
```

```xml
<esc_datamodel xmlns="http://www.cisco.com/esc/esc">  <tenants><tenant>  <name>Admin</name>  </tenant>  </tenants>  <deployments>  <deployment>  <deployment_name>NwDepModel_nosvc</deployment_name>  </deployment>  </deployments>  <vm_group>  <image></image>  <Flavor></Flavor>  ............  </vm_group>  <vm_group>  <image></image>  <Flavor></Flavor>  ............  </vm_group>  ............
```
NETCONF notification upon successful addition of 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 nc:operation="delete">
          ..........<image>
          ..........<Flavor>
          ..........<image>
          ..........<Flavor>
          ..........<image>
        </vm_group>
        <vm_group nc:operation="delete">
          ..........<image>
          ..........<Flavor>
          ..........<image>
        </vm_group>
      </deployment>
    </deployments>
  </tenant>
</tenants>

NETCONF notification upon successful deletion of vm_group:
UPDATE SERVICE REQUEST RECEIVED (UNDER TENANT)
   VM_UNDEPLOYED
SERVICE_UPDATED
UPDATE SERVICE REQUEST RECEIVED (UNDER TENANT)

Adding an Ephemeral Network in a VM Group

You can add an ephemeral network in a vm_group using 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>
              ..........<image>
              ..........<Flavor>
              ..........<image>
            </network>
          </networks>
        </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>
        ...........
        ...........
        ...........
        ...........
        ...........
      </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
You can add an interface in a vm_group from a running deployment using existing images and flavors.
NETCONF request to add an interface in a vm_group:
<interfaces>
  <interface>
    <nicid>0</nicid>
    <network>esc-net</network>
  </interface>
</interfaces>
Deleting an Interface in a VM Group

NETCONF request to delete an interface in a vm_group:

```xml
<interfaces>
  <interface>
    <nicid>0</nicid>
    <network>esc-net</network>
  </interface>
  <interface>
    <nicid>1</nicid>
    <network>utroycho-net</network>
  </interface>
  <interface nc:operation="delete">
    <nicid>2</nicid>
    <network>utroycho-net-1</network>
  </interface>
</interfaces>
```

Unified Deployment Request

ESC creates OpenStack resources such as tenants, networks, and sub-networks before deploying a VNF. During a unified deployment request, you send a unified request to create or delete the OpenStack resources, and deploy a VNF. However, registration of the VNF is not part of this request. The VNF must be registered before sending a deployment request. You can create multiple networks and sub-networks, but can create only a single VNF and a single tenant using unified deployment request.

Update the `dep.xml` and `reg.xml` files with the necessary information such as the service and deployment ID, tenant, network and sub-network ids and so on. You can either use NETCONF or REST APIs. For example, send POST REST and DELETE REST calls.

See the Elastic Services Controller Registration and Deployment Attributes for a list of registration and deployment attributes.

- To create and update `reg.xml` and `dep.xml` 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 subnet, network and tenant will be deleted in the specified order.

Note

If a tenant is not created as part of unified deployment request, there is nothing to roll back. An error occurs and a manual un-deploy must be triggered to cleanup.

If both create and delete unified deployment requests fail causing two failed events, there is an attempt to roll back twice. But, there is nothing to rollback.

Limitations:
• You must register a VNF before a unified deployment request.
• A unified deployment request can reuse an existing tenant only. It cannot reuse any existing networks, sub-networks or deployments. If there is a duplicate network, sub-network, or deployment, then the unified deployment request is rejected.
• During an undeploy request, any network and subnetwork created as part of the unified deployment request will be deleted along with the VNF. However, tenant created through unified deployment request will not be deleted.

### Applying Policies

Policies are applied during individual VM deployment. Policies such as affinity and anti-affinity streamline the deployment process. During a composite VNF deployment, rules defined in these policies enable easy deployment of individual VMs reducing the deployment time and cost.

### Affinity and Anti-affinity Rule

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).

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-up 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, VM on the other host continues to run preventing any loss of service.

### Intra Group Affinity Policy

The VNFs within the same VM group can either be deployed on the same host, or into the availability zone.

Example for Intra Group Affinity Policy:

```xml
<vm_group>
  <name>affinity-test-gp</name>
  <placement>
    <type>affinity</type>
    <enforcement(strict</enforcement>
  </placement>
...
```
**Host Based Placement**

The VNFs are within the same VM group and deployed on the same host. For example, VNF bundles. For deployments specific to a host, only one host can be specified. Before deploying, you need to make sure that the host exists in OpenStack. ESC validates the specified host in OpenStack. Hence, if a host specified in ESC does not exist in OpenStack, the deployment fails.

Example for host placement:

```
<vm_group>
 <placement>
   <type>host_placement</type>
   <enforcement>strict</enforcement>
   <host>my-ucs-4</host>
 </placement>
 ...
```

**Zone Based Placement**

The VNFs within the same VM group can be deployed in the available zone. The availability zones is created out of band of ESC. Before deploying, you need to make sure that the zone exists in OpenStack. ESC validates the specified zone in OpenStack. Hence, if a zone specified in ESC does not exist in OpenStack, the deployment fails.

Example for zone placement:

```
<vm_group>
   <name>affinity-test-gp1</name>
   <placement>
     <type>zone_placement</type>
     <enforcement>strict</enforcement>
     <zone>nova</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 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 only use one vm_group_ref tag, type tag and enforcement tag under the placement tag. The host or zone cannot be specified.
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 vm_group_ref tag, type tag and enforcement tag under the placement tag. The host or zone cannot be specified.

Example for Inter Group Anti-Affinity Policy:

```
<service_definition>
    <name>test-strict-affinity-2groups</name>
    <version>1.4</version>
    <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>
</vm_group>
```

Single Root I/O Virtualization

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. SR-IOV 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. Recent x86 server processors include chipset enhancements, such as Intel VT-x technology, that facilitate direct memory transfers and other operations required by SR-IOV.

The SR-IOV specification defines two device types:

- Physical Function (PF)-Essentially a static vNIC, a PF is a full PCIe device that includes SR-IOV capabilities. PFs are discovered, managed, and configured as normal PCIe devices. A single PF can provide management and configuration for a set of virtual functions (VFs).
- Virtual Function (VF)-Similar to a dynamic vNIC, a VF is a full or lightweight virtual PCIe device that provides at least the necessary resources for data movements. A VF is not managed directly but is derived from and managed through a PF. One or more VFs can be assigned to a VM.
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 bootup sequence, 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 address 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.

Undeploying Virtual Network Functions

You can undeploy an already deployed VNF. Use the REST or NetConf / Yang APIs to un-deploy the VNF.

Sample undeploy request

DELETE /v0/deployments/567 HTTP/1.1
Host: client.host.com
Content-Type: application/xml
Accept: application/xml
Client-Transaction-Id: 123456
Callback:/undeployservicecallback

A sample callback (first stage) for undeployment is as follows:

A callback is received at several stages of undeployment. One callback for each VM that is undeployed. A final callback after all the VMs are undeployed.

HTTP/1.1 200 OK
Content-Type: application/xml; charset=UTF-8
Content-Length: 1488
Date: Sun, 1 Jan 2011 9:00:00 GMT
ESC-Transaction-Id: 123456
<?xml version="1.0" encoding="UTF-8"?>
<deployment>
<deployment_stage>VM_UNDEPLOYED</deployment_stage>
<deployment_details>
<vm_uuid>26ebf6a5-812e-4034-8a50-588c6579f70a</vm_uuid>
<host_uuid<rereea5-812e-4034-8a50-588c6579f70a</host_uuid>
<interfaces>
<interface>
<network_uuid>0e89b2ed-5e79-47b2-9ff1-07018f0417ef</network_uuid>
<ip_address>10.0.3.58</ip_address>
<port_uuid>c806c915-d5af-4269-bb6b-b766b68c20e1</port_uuid>
<subnet_uuid>fd96c915-d5af-4269-bb6b-b766b68c20e1</subnet_uuid>
<mac_address>fa:16:3e:01:ed:13</mac_address>
</interface>
</interfaces>
<management_ip_address>10.0.3.58</management_ip_address>
</deployment_details>
</deployment>

<external_deployment_id>9866330f-a4e0-4d29-9ce2-ef90b94a65e8</external_deployment_id>
<external_service_registration_id>2d5a7003-bdf9d-4a1c-b2eb-f2de1d3c18f7</external_service_registration_id>
<external_tenant_id>4f384dfc667a4776bc3169da27d09db5</external_tenant_id>
See the REST and NetConf API Guide for more details.
Monitoring Virtual Network Functions

- Monitoring the VNFs, page 37
- Monitoring Methods, page 39
- Monitoring a VM, page 41
- Monitoring Operations, page 42

Monitoring the VNFs

After deploying VNFs, they are monitored periodically to check their health and workload. 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.

KPIs, Rules and Dynamic Mapping Files in the Datamodel

The Elastic Service Controller allows user to define metrics to be monitored, and actions to be executed when certain conditions are met. These metrics and actions are defined either at the time of registration or 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.

The monitoring and actions service (MONA) engine performs monitoring activities, and executes actions. A dynamic mapping file supports mapping of KPI and rules to the MONA data model.

Dynamic mapping file structure:

```xml
<dynamic_mappings>
  <actions>
    <!-- service booted action for backward compatibility with previous script -->
    <action>
      <name>TRUE_servicebooted.sh</name>
      <type>ESC_POST_EVENT</type>
      <metaData>
        <properties>
          <property></property>
        </properties>
        <type>snmp_get_threshold</type>
      </metaData>
    </action>
  </actions>
  <metrics>
    <metric>
      <name>MEMORY</name>
      <type>MONITOR_COMPUTE_THRESHOLD</type>
      <metaData>
        <properties>
          <property></property>
        </properties>
        <type>snmp_get_threshold</type>
      </metaData>
    </metric>
  </metrics>
</dynamic_mappings>
```

Each of the sections in the dynamic mapping file allows the administrator to add and remove actions and metrics to the set of supported ESC KPIs and rules. 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.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>esc_url</td>
<td>The URL of the Elastic Services Controller.</td>
</tr>
<tr>
<td>vm_external_id</td>
<td>The external id of the managed VM.</td>
</tr>
<tr>
<td>vm_name</td>
<td>The name of the managed VM.</td>
</tr>
<tr>
<td>vm_mac_address</td>
<td>The mac address of the managed VM.</td>
</tr>
<tr>
<td>vm_external_host_id</td>
<td>The VM external host Identifier.</td>
</tr>
<tr>
<td>vm_external_host_name</td>
<td>The VM external host name.</td>
</tr>
<tr>
<td>vm_group_name</td>
<td>The VM group name.</td>
</tr>
<tr>
<td>ip_address</td>
<td>The VM IP Address.</td>
</tr>
<tr>
<td>agent_address</td>
<td>The VM IP Address.</td>
</tr>
<tr>
<td>event_name</td>
<td>The ESC event name.</td>
</tr>
</tbody>
</table>
A sample dynamic mapping file is as follows:

```
<action>
  <name>TRUE demo_script_service_booted</name>
  <type>SCRIPT</type>
  <metaData>
    <type>script</type>
    <properties>
      <property>
        <name>script_filename</name>
        <value>/cisco/esc-scripts/pipe.sh</value>
      </property>
      <property>
        <name>esc_url</name>
        <value></value>
      </property>
      <property>
        <name>vm_external_id</name>
        <value></value>
      </property>
      <property>
        <name>vm_name</name>
        <value></value>
      </property>
      <property>
        <name>event_name</name>
        <value>VM_ALIVE</value>
      </property>
      <property>
        <name>esc_event</name>
        <value>SERVICE_BOOTED</value>
      </property>
      <property>
        <name>ip_address</name>
        <value></value>
      </property>
    </properties>
  </metaData>
</action>
```

The name is set to TRUE demo_script_service_booted. The prefix TRUE is maintained to be compliant with the existing ESC datamodel. The action name here denotes a script that the user wants to run at service boot up time. /cisco/esc-scripts/pipe.sh is the script to be executed. Parameters will be added to the empty value field at the time of execution.

**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**

ICMP Ping monitoring, ICMP pings assess the liveliness or reachability of a VNF.

If the VM is dead then 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, 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 script based monitoring, scripts are executed to assess a condition on a VM. Custom scripts are pre-installed on ESC VM. These scripts are executed at defined intervals to assess the liveliness of the VNFs, and also support special operations. The execution of these scripts are controlled by the data model, kpi, metric_occurrences and so on. The script must return 0 for success, and 1 for failure.

For example, to add custom action scripts, include the details in the dynamic mapping file and update your KPI model to reference it.

You can update the dynamic mapping script with

```xml
<action>
   <name>TRUE Sample_Demo_Script</name>
   <type>SCRIPT</type>
   ......................
</action>
```

Update your KPI model. For example,

```xml
<kpi>
   <event_name>VM_SAMPLE_DEMO_EVENT</event_name>
   <metric_value>100</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>SUBSCRIBER_SESSION</type>
      <nicid>0</nicid>
      <poll_frequency>15</poll_frequency>
      <polling_unit>seconds</polling_unit>
      <continuous_alarm>false</continuous_alarm>
</metric_collector>
</kpi>
```

The rule section of the same deployment can have the following custom script.

```xml
<rule>
   <event_name>VM_UNDERLOADED_EMPTY_SAMPLE</event_name>
   <action>TRUE Sample_Demo_Script</action>
</rule>
```

Upon execution of the action identified by TRUE Sample_Demo_Script, the custom script defined in the property script_filename is run, and the event VM_UNDERLOADED_EMPTY_SAMPLE is triggered.
Monitoring a VM

ESC 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 servicerecovery.sh 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 file 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 file will be 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 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_maintenance</operation>
  <force>false</force>
</vm_operation>
```

You must mention enable maintenance to set VM monitoring, and disable maintenance to unset VM monitoring in the operation field.
Scaling Virtual Network Functions

- Scaling Overview, page 43
- Scale In and Scale Out of VMs, page 43
- Scaling Notifications and Events, page 44

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.

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 minimum 2 VMs and a maximum of 100 VMs, the below xml will define scaling under each VM group.

```xml
<scaling>
  <min_active>2</min_active>
  <max_active>100</max_active>
  <elastic>true</elastic>
</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 servicescaleup.sh</action>
</rule>
<rule>
    <event_name>VM_UNDERLOADED</event_name>
    <action>ALWAYS log</action>
    <action>TRUE servicescaledown.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.
**Notifications Scenarios**

ESC deploys VMs and sets KPI/Monitors and all VM Alives received. The following NETCONF notification is triggered.

```xml
<type>SERVICE_ALIVE</type>
<status>SUCCESS</status>
```

When ESC receives a VM_OVERLOADED event, the following NetConf notification is triggered:

```xml
<type>VM_SCALE_OUT_INIT</type>
<status>SUCCESS</status>
```

ESC checks if the max limit is reached, if not, it deploys a new VM.

```xml
<type>VM_SCALE_OUT_DEPLOYED</type>
<status>SUCCESS</status>
```

Once the deployment is complete, the following Netconf Notification is sent,

```xml
<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

```xml
<type>SERVICE_ALIVE</type>
<status>SUCCESS</status>
```

When ESC receives a VM_UNDERLOADED event, the following NetConf notification is triggered

```xml
<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.

```xml
<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.
Healing Virtual Network Functions

- Healing Overview, page 47
- Healing a VM, page 47
- Notifications and Events, page 48

Healing Overview

As part of life cycle management, ESC heals the VNFs when there is a failure. The failure scenarios 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 is configured in the rules section during the deployment.

Healing a VM

Each VM group is configured to enable the healing. Healing is performed at various stages and in various ways depending on the state of the VM.

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 reboot the VM. If the reboot does not recover the VM, the VM is undeployed and redeployed.

If ESC does not receive a VM Alive after deployment, it undeploys and redeploy the VM. The deploy and undeploy procedure is repeated for a maximum number of attempts.

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 Monitoring the VNFs, on page 37.

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:

```xml
<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:

```xml
<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 ICMPPing 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, 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.

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

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.

The following table lists the different scenarios and the notifications that are generated for every event:
<table>
<thead>
<tr>
<th>Scenario</th>
<th>Notifications</th>
</tr>
</thead>
</table>
| 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-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 or 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>``` |
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:

```xml
<type>SERVICE_ALIVE</type>
<status>SUCCESS</status>
```

After ESC receives VM down event, the following NetConf notification is triggered:

```xml
<type>VM_RECOVERY_INIT</type>
<status>SUCCESS</status>
```

ESC fails to recover the VM by Undeploy or 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:

```xml
<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:

```xml
<type>VM_RECOVERY_DEPLOYED</type>
<status>SUCCESS</status>
```

ESC receives a VM Alive event and triggers the following Netconf notifications:

```xml
<type>VM_RECOVERY_COMPLETE</type>
<status>SUCCESS</status>
```
### Notifications and Events

#### Scenario

**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:

```xml
<type>VM_RECOVERY_INIT</type>
<status>SUCCESS</status>
```

ESC un-sets the monitoring and un-deploys the VM.

The following NetConf notification is triggered:

```xml
<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:

```xml
<type>VM_RECOVERY_DEPLOYED</type>
<status>SUCCESS</status>
```

ESC receives a VM Alive event and triggers the following Netconf notifications:

```xml
<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 set KPI to monitor on all VM Alive received. The following NetConf notification is triggered:

```xml
<type>SERVICE_ALIVE</type>
<status>SUCCESS</status>
```

After ESC receives VM down event, the following NetConf notification is triggered:

```xml
<type>VM_RECOVERY_INIT</type>
<status>SUCCESS</status>
```

ESC receives an error while attempting to recover through Reboot.

The following NetConf notification is triggered:

```xml
<type>VM_RECOVERY_COMPLETE</type>
<status>FAILURE</status>
```
### Scenario: 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:

```xml
<type>SERVICE_ALIVE</type>  
<status>SUCCESS</status>  
```

After ESC receives VM down event, the following NetConf notification is triggered:

```xml
<type>VM_RECOVERY_INIT</type>  
<status>SUCCESS</status>  
```

ESC fails to recover the VM by **Reboot** and proceeds with recovery by **Undeploy** or **Redeploy**.

It un-sets monitoring and un-deploys the VM.

The following NetConf notification is triggered:

```xml
<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:

```xml
<type>VM_RECOVERY_COMPLETE</type>  
<status>FAILURE</status>  
```

### Scenario: 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:

```xml
<type>SERVICE_ALIVE</type>  
<status>SUCCESS</status>  
```

After ESC receives VM down event, the following NetConf notification is triggered:

```xml
<type>VM_RECOVERY_INIT</type>  
<status>SUCCESS</status>  
```

ESC un-sets monitoring and un-deploys the VM. Recovery is performed by **Undeploy** or **Redeploy**.

The following NetConf notification is triggered:

```xml
<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:

```xml
<type>VM_RECOVERY_COMPLETE</type>  
<status>FAILURE</status>  
<type>SERVICE_ALIVE</type>  
<status>FAILURE</status>  
```
Virtual Network Function Operations

VNF Operations

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

A payload is required for VNF operations:
POST ESCManager/v0/{internal_tenant_id}/deployments/service/{internal_deployment_id}

Example,
```xml
<?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 up, scale down. 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
<?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.
## Error Handling

- Error Conditions for ESC Operations, page 55

### 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.

**Notification 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.

<table>
<thead>
<tr>
<th>Error Condition</th>
<th>Recovery</th>
</tr>
</thead>
<tbody>
<tr>
<td>Failed create tenant request</td>
<td>NB client (ESC User) has to send in a delete tenant request before attempting to send in the same create tenant request</td>
</tr>
<tr>
<td>Failed create network request</td>
<td>NB client (ESC User) has to send in a delete network request before attempting to send in the same create network request</td>
</tr>
<tr>
<td>Failed create subnet request</td>
<td>NB client (ESC User) has to send in a delete subnet request before attempting to send in the same create subnet request</td>
</tr>
<tr>
<td>Failed service registration request</td>
<td>NB client (ESC User) has to send in an unregister service request before attempting to send in the same service registration request</td>
</tr>
<tr>
<td>Error Condition</td>
<td>Recovery</td>
</tr>
<tr>
<td>-----------------------------------------------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Failed deployment request</td>
<td>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.</td>
</tr>
<tr>
<td>Failed Recovery</td>
<td>The existing deployment is not usable anymore. NB client (ESC User) has to send in an undeploy request then the same deploy request.</td>
</tr>
<tr>
<td>Failed Scale Out/In</td>
<td>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.</td>
</tr>
<tr>
<td>Failed Service Update</td>
<td>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.</td>
</tr>
<tr>
<td>Failed VM Operations (Start, Stop, Reboot, Enable Monitor, Disable Monitor)</td>
<td>No action required. The existing deployment is still functional. NB client (ESC User) can retry the failed operation.</td>
</tr>
<tr>
<td>Failed VNF/Service Operations (Start, Stop, Reboot, Enable Monitor, Disable Monitor)</td>
<td>No action required. The existing deployment is still functional. NB client (ESC User) can retry the failed operation.</td>
</tr>
<tr>
<td>Failed delete tenant request</td>
<td>Possibility of leaking resource in VIM. Manual intervention might be needed to clean up leaking resources on VIM.</td>
</tr>
<tr>
<td>Failed delete network request</td>
<td>Possibility of leaking resource in VIM. Manual intervention might be needed to clean up leaking resources on VIM.</td>
</tr>
<tr>
<td>Failed delete subnet request</td>
<td>Possibility of leaking resource in VIM. Manual intervention might be needed to clean up leaking resources on VIM.</td>
</tr>
<tr>
<td>Failed service unregistration request</td>
<td>Possibility of leaking resource in VIM. Manual intervention might be needed to clean up leaking resources on VIM.</td>
</tr>
</tbody>
</table>
Error Handling

Error Conditions for ESC Operations

<table>
<thead>
<tr>
<th>Error Condition</th>
<th>Recovery</th>
</tr>
</thead>
<tbody>
<tr>
<td>Failed undeployment request</td>
<td>Possibility of leaking resource in VIM. Manual intervention might be needed to clean up leaking resources on VIM</td>
</tr>
</tbody>
</table>

**ESC System Logs**

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

The log message format is as follows:

```plaintext
timestamp threadname loglevel [filename:methodname():line#] [transactionId] [classification] [tags] message
```

The following classifications are supported:

- **SM**—stands for StateMachine. This classification indicates logs in the StateMachine category.
- **REST_EVENT**—indicates REST EVENTS in logs. This is used by ESC UI.

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 log analysis tools.

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