Configuring Deployment Parameters

Deployment Parameters

A VNF deployment is initiated as a service request through the northbound interface or the ESC portal. The service request comprises of templates that consist of XML payloads and deployment parameters. Deployment parameters are rules, policies or day 0 configuration that determine properties of the VNF and its lifecycle. The table below lists the complete list of deployment parameters and how they interoperate on OpenStack or VMware vCenter:

Deployment Parameters	OpenStack	VMware vCenter	VMware vCloud Director
Day 0 Configuration	Day 0 configuration is done in one of the following ways: • NETCONF API • REST API • ESC Portal	Day 0 configuration is done in one of the following ways: • NETCONF API • REST API • ESC Portal	• NETCONF API • REST API • ETSI API

Deployment Parameters	OpenStack	VMware vCenter	VMware vCloud Director
Deploying VNFs	Configuration of Individual and Composite VNFs is done in one of the following ways: • NETCONF API • REST API • ESC Portal (You can deploy using the Deployment Template.)	Configuration of Individual and Composite VNFs is done in one of the following ways: • NETCONF API • REST API • ESC Portal (You can configure the VNF settings through the Deployment Form, or the Deployment Template.)	Configuration of Individual and Composite VNFs is done in one of the following ways: • NETCONF API • REST API • ETSI API
Undeploy Virtual Network Functions	Undeploying is done in one of the following ways: • NETCONF API • REST API • ESC Portal	Undeploying VNFs is done in one of the following ways: • NETCONF API • REST API • ESC Portal	Undeploying VNFs is done in one of the following ways: • NETCONF API • REST API • ETSI API
Affinity and anti-affinity Rule	Creating and deleting affinity and anti-affinity rule definitions is done in one of the following ways: • NETCONF API • REST API	Creating and deleting affinity rule definition in one of the following ways: • NETCONF API • REST API • ESC Portal (You can set up affinity and anti-affinity using the Deployment Form.)	Creating and deleting affinity and anti-affinity rule definitions is done in one of the following ways: • NETCONF API • REST API • ETSI API
VNF Operations	VNF Operations are done in one of the following ways: • REST API • NETCONF API • ESC Portal	VNF Operations are done in one of the following ways: • REST API • NETCONF API • ESC Portal For more information, see the Elastic Services Controller Portal.	VNF Operations are done in one of the following ways: • REST API • NETCONF API • ETSI API

Deployment Parameters	OpenStack	VMware vCenter	VMware vCloud Director
Multi Cluster	Not applicable	Multi Cluster configuration is done in one of the following ways: • REST API • ESC Portal For more information, see the Deploying VNFs on VMware vCenter using ESC Portal.	Not applicable
Multiple Virtual Datacenter (Multi VDC)	Not applicable	Multiple Virtual Datacenter selection is done in one of the following ways: • REST API • ESC Portal	Not applicable
Hardware Acceleration	Hardware Acceleration is supported in one of the following ways: • NETCONF API • REST API For more information, see the Hardware Acceleration Support (OpenStack Only), on page 54.	Not applicable	Not applicable
Single Root I/O Virtualization	Configuration of Single Root I/O Virtualization is done in one of the following ways: • NETCONF API • REST API	Configuration of Single Root I/O Virtualization is done in one of the following ways: • NETCONF API • REST API	Not applicable

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

Day Zero Configuration

The initial or day 0 configuration of a VNF is based on the VM type. A VNF administrator configures the initial template for each VM type at the time of VNF deployment. The same configuration template is applied to all deployed and new VMs of that VM type. The template is processed at the time of individual VM

deployment. The day 0 configuration continues to persists, so that all initial deployment, healing and scaling of VMs have the same day 0 template.

Some of the day 0 configuration tasks include bringing up the interface, managing the network, support for static or dynamic IP (DHCP, IPAM), SSH keys, and NetConf enabled configuration support on VNF.

Note ESC does not support day 0 configuration of interfaces added during service update. In case of recovery for day 0 configuration, all the interfaces with Network Interface Card IDs will be configured.

Day Zero in the configuration data model

The day 0 configuration file can be specified in different ways in the data model, but you can use only one of the options at a time.

- <file> url </file>—The url specifies a file on the ESC VM file system or file hosted on report http server. ESC downloads the file specified by the URL. This file is used as a template to replace the tokens specified in this template with the values specified in the variables section. This template is used to generate the day 0 configuration.
- <data> inline config content </data>—Specifies URL for the template. This allows the use of inline text as the template.
- <encrypted_data> inline config content</encrypted_data>—The inline configuration content will be encrypted based on the data.
- <file_locators> list of file locators </file_locators>—Similar to file, a file_locator defines file to download from a remote server with basic authentication (if required).

  

  Note The <file_locators> is deprecated in ESC Release 4.0.

- <file_locator_name> deployment defined file_locator </file_locator_name>—Similar to file, the file_locator_name is used to download the file from a remote server with basic authentication (if required).

Day 0 configuration is defined in the datamodel under the config_data tag. Each user data and the configuration drive file is defined under the configuration tag. The contents are in the form of a template. ESC processes the template through the Apache Velocity Template Engine before passing to the VM.

The config_data tag is defined for each vm_group. The same configuration template is applied to all VMs in the vm_group. The template file is retrieved and stored at deployment initialization. Template processing is applied at time of VM deployment. The content of the config file can be retrieved from the file or data.

```
<file> url </file>
<data> inline config content </data>
```

A destination name is assigned to the config by <dst>. User Data is a treated as a special case with <dst>--user-data</dst>.

A sample config data model,

```
<config_data>
    <configuration>
        <file>file://cisco/userdata_file.txt</file>
        <dst>--user-data</dst>
```

```
        <variable>
            <name>CUSTOM_VARIABLE_FOR_USERDATA</name>
            <val>SOME_VALUE_XXX</val>
        </variable>
    </configuration>
    <configuration>
        <file>file://cisco/config.sh</file>
        <dst>config.sh</dst>
        <variable>
            <name>CUSTOM_VARIABLE_FOR_CONFIG</name>
            <val>SOME_VALUE_XXX</val>
        </variable>
    </configuration>
</config_data>
```

Custom variable can be specified in the variables tag within the configuration. Zero or more variables can be included in each configuration. Each variable can have multiple values. Multiple values are only useful when creating more than one VM per vm_group. Also, when performing scale in and scale out, additional VMs can be added and removed from the VM group.

Note Note the following while providing multiple values for the variable tag.

- The variable values assigned to the initially deployed VMs are unique and from the pool. There is no order followed for assigning the values from the pool. That is, the first VM can use the second value from the pool.
- A scaled out VM should have a unique variable value and from the pool.
- A recovered VM (after undeploy or redeploy) must retain the same value it had before.

The contents of <file> are a template that is processed by the Velocity Template Engine. ESC populates a set of variables for each interface before processing the configuration template:

NICID_n_IP_ALLOCATION_TYPE	string containing FIXED | DHCP
NICID_n_NETWORK_ID	string containing neutron network uuid
NICID_n_IP_ADDRESS	ipv4 or ipv6 address
NICID_n_MAC_ADDRESS	string
NICID_n_GATEWAY	ipv4 or ipv6 gateway address
NICID_n_CIDR_ADDRESS	ipv4 or ipv6 cidr prefix address
NICID_n_CIDR_PREFIX	integer with prefix-length
NICID_n_NETMASK	If an ipv4 CIDR address and prefix are present, ESC will automatically calculate and populate the netmask variable. This is not substituted in the case of an IPv6 address and should not be used.
NICID_n_ANYCAST_ADDRESS	string with ipv4 or ipv6
NICID_n_IPV4_OCTETS	string with last 2 octets of ip address, such as 16.66, specific to CloudVPN

Where n is the interface number from the data model, for example, 0, 1, 2, 3

Note The interface number, n starts with 0 for OpenStack, and 1 for VMware.

Example

```
NICID_0_IP_ALLOCATION_TYPE: FIXED
NICID_0_NETWORK_ID: 9f8d9a97-d873-4a1c-8e95-1a123686f038
NICID_0_IP_ADDRESS: 2a00:c31:7fe2:1d:0:0:1:1000
NICID_0_MAC_ADDRESS: null
NICID_0_GATEWAY: 2a00:c31:7fe2:1d::1
NICID_0_CIDR_ADDRESS: 2a00:c31:7fe2:1d::
NICID_0_CIDR_PREFIX: 64
NICID_0_ANYCAST_ADDRESS: null
NICID_0_IPV4_OCTETS: 16.0
NICID_1_IP_ALLOCATION_TYPE: DHCP
NICID_1_NETWORK_ID: 0c468d8e-2385-4641-b1db-9080c170cb1a
NICID_1_IP_ADDRESS: 6.0.0.2
NICID_1_MAC_ADDRESS: null
NICID_1_GATEWAY: 6.0.0.1
NICID_1_CIDR_ADDRESS: 6.0.0.0
NICID_1_CIDR_PREFIX: 24
NICID_1_ANYCAST_ADDRESS: null
NICID_1_NETMASK: 255.255.255.0
```

By default, ESC substitutes the $ variable in the day 0 configuration file with the actual value during deployment. You can enable or disable the $ variable substitution for each configuration file.

Add the following field to the configuration data model:

```
<template_engine>VELOCITY | NONE</template_engine> field to configuration
```

where,

- VELOCITY enables variable substitution.
- NONE disables variable substitution.

If no value is set the default option is VELOCITY, and the $ variable substitution takes place. When set to NONE, the $ variable substitution does not take place.

You must follow these tips while processing the template through the velocity template engine.

- To escape dollar sign in the template insert,

  ```
  #set ( $DS = "$" )
  ```

  then replace the variable with

  ```
  passwd: ${DS}1${DS}h1VxC40U${DS}uf2qLUwGTjHgZplkP78xA
  ```

- To escape a block in the template, insert #[[and #]]. For example,

  ```
  #[[ passwd: $1$h1VxC40U$uf2qLUwGTjHgZplkP78xA ]]#
  ```

File locator

To fetch external configuration files, a file locator is added to the day 0 configuration. The file locator contains a reference to the file server, and the relative path to the file to be downloaded.

Note The file locator attribute is defined at the deployment level, that is, directly under the deployment container instead of policy actions and day 0 configuration sections. For updated data model see Fetching Files From Remote Server.

Example of day 0 configuration with a file locator:

```
<esc_datamodel xmlns="http://www.cisco.com/esc/esc">
    <tenants>
        <tenant>
            <name>sample-tenant</name>
            <deployments>
                <deployment>
                    <name>sample-deployment</name>
                    <vm_group>
                        <name>sample-vm-group</name>
                        <config_data>
                            <!-- exisiting configuration example - remains valid -->
                            <configuration>
                                <file>file:///cisco/config.sh</file>
                                <dst>config.sh</dst>
                            </configuration>
                            <!-- new configuration including use of file locators -->
                            <configuration>
                                <dst>ASA_config_0</dst>
                                <file_locators>
                                    <file_locator>
                                        <name>configlocator-1</name>
                                        <!-- unique name -->
                                        <remote_file>
                                            <file_server_id>server-1</file_server_id>

<remote_path>/share/users/configureScript.sh</remote_path>
                                             <!-- optional user specified local silo directory
 -->

<local_target>day0/configureScript.sh</local_target>
                                            <!-- persistence is an optional parameter -->
                                            <persistence>FETCH_ALWAYS</persistence>
                                            <!-- properties in the file_locator are only
used for
                                fetching the file not for running scripts -->
                                            <properties>
                                                <property>
                                                    <!-- the property name
"configuration_file" with value "true" indictates this is the
                                    script to be used just as using the <file> member case of
the configuration -->
                                                    <name>configuration_file</name>
                                                    <value>true</value>
                                                </property>
                                                <property>
                                                    <name>server_timeout</name>
                                                    <value>120</value>
                                                  <!-- timeout value in seconds, overrides
 the file_server property -->
                                                </property>
                                            </properties>
                                        </remote_file>
                                        <!-- checksum is an optional parameter.
                                The following algorithms are supported: SHA-1, SHA-224, SHA-256,
```

```
SHA-384, SHA-512 -->
                                        <checksum>SHA256 (configureScript.sh) =
dd526bb2c0711238ec2649c4b91598fb9a6cf1d2cb8559c337c5f3dd5ea1769e</checksum>
                                    </file_locator>
                                    <file_locator>
                                        <name>configlocator-2</name>
                                        <remote_file>
                                            <file_server_id>server-2</file_server_id>

<remote_path>/secure/requiredData.txt</remote_path>
                                        <local_target>day0/requiredData.txt</local_target>

                                            <persistence>FETCH_ALWAYS</persistence>
                                            <properties />
                                        </remote_file>
                                    </file_locator>
                                </file_locators>
                            </configuration>
                        </config_data>
                    </vm_group>
                </deployment>
            </deployments>
        </tenant>
    </tenants>
</esc_datamodel>
```

The file locator parameters include:

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

For more information, see Fetching Files From Remote Server.

To encrypt the files see, Encrypting Configuration Data.

Day 0 Configuration for vCD Deployment

The day 0 configuration for vCD deployment can be passed in different ways:

- Constructing an ISO file
- OVF properties
- Pre-existing ISO file in a catalog (OOB ISO file)

Note

- For initial deployment, the number of VM group(s) defined in the datamodel must be the same as the number of VM(s) in the vApp template. In a deployment, the image value of each VM group should be unique.
- The out of band (OOB) ISO file cannot be used along with constructing an ISO file method, as the VM can consider any one. The ovf property can be used with OOB ISO or constructing ISO together.

Day 0 configuration through constructing an ISO file:

```
</rules>
            <config_data>
              <!-- take content from the file path and save it as config.sh into the ISO
file -->
              <configuration>
                <dst>config.sh</dst>
                <file>file:///cisco/config.sh</file>
              </configuration>
             <!-- take content from the file path, replace variables with values, and save
 it as data/config.sh into the ISO file -->
              <configuration>
                <dst>data/params.cfg</dst>
                <file>file:///cisco/template.cfg</file>
                <variable>
                  <name>CF_VIP_ADDR</name>
                  <val>10.0.0.9</val>
                </variable>
                <variable>
                  <name>CF_DOMAIN_NAME</name>
                  <val>cisco.com</val>
                </variable>
                <variable>
                  <name>CF_NAME_SERVER</name>
                  <val>172.16.180.7</val>
                </variable>
              </configuration>
             <!-- take the data section as the content of the file, replace variables with
 values, and save it as user-data.txt into the ISO file-->
              <configuration>
                <dst>user-data.txt</dst>
                <data>#cloud-config
manage_etc_hosts: true
hostname: $HOST_NAME
local-hostname: $HOST_NAME
</data>
                <variable>
                  <name>$HOST_NAME</name>
                  <val>something.cisco.com</val>
                </variable>
              </configuration>
            </config_data>
```

Day 0 configuration through OOB ISO file:

```
         </rules>
            <config_data>
              <configuration>
                <!-- ISO file stored in catalog-1 -->
                <dst>vcdCatalog:catalog-1</dst>
```

```
            <data>h2.iso</data>
          </configuration>
        </config_data>
```

Day 0 configuration through OVF properties:

```
<config_data>
   <configuration>
     <!-- ovf properties as day0 -->
     <dst>ovfProperty:mgmt-ipv4-addr</dst>
     <data>$NICID_0_IP_ADDRESS/24</data>
   </configuration>
 </config_data>
```

For information on deploying VNFs on vCD, see Deploying Virtual Network Functions on VMware vCloud Director (vCD).

KPIs, Rules and Metrics

Cisco Elastic Services Controller VNF monitoring is done through the definition of Key Performance Indicators (KPIs) metrics. Core metrics are preloaded with ESC, a programmable interface gives to the end-user the ability to add and remove metrics, but also to define the actions to be triggered on specified conditions.These metrics and actions are defined at the time of deployment.

The ESC metrics and actions datamodel is divided into 2 sections:

1. KPI—Defines the type of monitoring, events, polling interval and other parameters. This includes the event_name, threshold and metric values. The event_name is user defined. The metric_values specify threshold conditions and other details. An event is triggered when the threshold condition is reached.
2. Rule—Defines the actions when the KPI monitoring events are triggered. The action element defines the actions to be performed when an event corresponding to the event_name is triggered.

Rules

The ESC object model defines for each vm_group a section where the end-user can specify the administrative rules to be applied based on the outcome of the KPIs selected metric collector.

```
<rules>
    <admin_rules>
        <rule>
            <event_name>VM_ALIVE</event_name>
            <action>TRUE esc_vm_alive_notification</action>
            <action>FALSE recover autohealing</action>
        </rule>
        : : : : : : : : : : : : : : : : :
    </admin_rules>
</rules>
```

As mentioned within the KPIs section, correlation between KPIs and Rules is done based on the value of the <event_name> tag.

In the Rules section above, if the outcome of the KPIs defining event_name is VM_ALIVE, and the selected metric collector is TRUE, then the action identified by the key, TRUE esc_vm_alive_notification is selected for execution.

If the outcome of the KPIs defining event_name is VM_ALIVE, and the selected metric collector is FALSE, then the action identified by the key, FALSE recover autohealing is selected for execution.

For information on updating KPIs and Rules, see Updating the KPIs and Rules.

Metrics and Actions

ESC Metrics and Actions (Dynamic Mapping) framework is the foundation of the kpis and rules sections. As described in the KPIs section the metric type uniquely identifies a metric and its metadata.

The metrics and actions is as follows:

```
<metrics>
    <metric>
        <name>ICMPPING</name>
        <userLabel>ICMP Ping</userLabel>
        <type>MONITOR_SUCCESS_FAILURE</type>
        <metaData>
            <type>icmp_ping</type>
            <properties>
                <property>
                    <name>ip_address</name>
                    <value />
                </property>
                <property>
                    <name>enable_events_after_success</name>
                    <value>true</value>
                </property>
                <property>
                    <name>vm_gateway_ip_address</name>
                    <value />
                </property>
                <property>
                    <name>enable_check_interface</name>
                    <value>true</value>
                </property>
            </properties>
        </metaData>
    </metric>
    : : : : : : : : :
</metrics>
```

The above metric is identified by its unique name ICMPPING. The <type> tag identifies the metric type.

Currently ESC supports two types of metrics:

- MONITOR_SUCCESS_FAILURE
- MONITOR_THRESHOLD

The <metadata> section defines the attributes and properties that is processed by the monitoring engine.

The metric_collector type in the KPI show the following behavior:

At regular intervals of 3 seconds the behavior associated with the ICMPPING identifier is triggered. The ICMPPING metric is of type MONITOR_SUCCESS_FAILURE, that is the outcome of the monitoring action is either a success or a failure. In the sample above, an icmp_ping is performed using the <ip_address> field defined in the <metadata> section. In case of SUCCESS the rule action(s) with the TRUE prefix will be selected for execution. In case of FAILURE the rule action(s) with the FALSE prefix is selected for execution.

```
<actions>
    <action>
```

```
            <name>TRUE servicebooted.sh esc_vm_alive_notification</name>
            <type>ESC_POST_EVENT</type>
            <metaData>
                <type>esc_post_event</type>
                <properties>
                    <property>
                        <name>esc_url</name>
                        <value />
                    </property>
                    <property>
                        <name>vm_external_id</name>
                        <value />
                    </property>
                    <property>
                        <name>vm_name</name>
                        <value />
                    </property>
                    <property>
                        <name>event_name</name>
                        <value />
                    </property>
                    <property>
                        <name>esc_event</name>
                        <value>SERVICE_BOOTED</value>
                    </property>
                </properties>
            </metaData>
        </action>
        : : : : : : : : :
</actions>
```

The action sample above describes the behavior associated with the SUCCESS value. The ESC rule action name TRUE servicebooted.sh esc_vm_alive_notification specifies the action to be selected. Once selected the action <type> ESC_POST_EVENT identifies the action that the monitoring engine selects.

Metrics and Actions APIs

In Cisco ESC Release 2.1 and earlier, mapping the actions and metrics defined in the datamodel to the valid actions and metrics available in the monitoring agent was enabled using the *dynamic_mappings.xml* file. The file was stored in the ESC VM and was modified using a text editor. ESC 2.2 and later do not have an *esc-dynamic-mapping* directory and *dynamic_mappings.xml* file. However, if you have an existing dynamic_mapping xml file that you want to add to the ESC VM, do the following:

1. Backup this file to a location outside of ESC, such as, your home directory.
2. Create *esc-dynamic-mapping* directory on your ESC VM. Ensure that the read permissions are set.
3. Install on your ESC VM using the following bootvm argument:

```
 --file
root:root:/opt/cisco/esc/esc-dynamic-mapping/dynamic_mappings.xml:<path-to-local-copy-of-dynamic-mapping.xml>
```

The CRUD operations for mapping the actions and the metrics are available through REST API. Refer to the API tables below for mapped metrics and actions definition.

To update an existing mapping, delete and add a new mapping through the REST API.

Note While upgrading any earlier version of ESC to ESC 2.2 and later, to maintain the VNF monitoring rules, you must back up the *dynamic_mappings.xml* file and then restore the file in the upgraded ESC VM. For more information upgrading monitoring rules, see Upgrading VNF Monitoring Rules section in the *Cisco Elastic Services Controller Install and Upgrade Guide*. Cisco ESC Release 2.3.2 and later, the dynamic mapping API is accessible locally only on the ESC VM.

Table 1: Mapped Actions

User Operation	Path	HTTP Operation	Payload	Response	Description
Read	internal/dynamic_mapping/actions/ <action_name>	GET	N/A	Action XML	Get action by name
Read All	internal/dynamic_mapping/actions	GET	N/A	Action XML	Get all actions defined
Write	internal/dynamic_mapping/actions	POST	Actions XML	Expected Action XML	Create one or multiple actions
Delete	internal/dynamic_mapping/actions/ <action_name>	DELETE	N/A	N/A	Delete action by name
Clear All	internal/dynamic_mapping/actions	DELETE	N/A	N/A	Delete all non-core actions

The response for the actions APIs is as follows:

```
<actions>
    <action>
        <name>{action name}</name>
        <type>{action type}</type>
        <metaData>
            <type>{monitoring engine action type}</type>
            <properties>
                <property>
                    <name />
                    <value />
                </property>
                : : : : : : :
            </properties>
        </metaData>
    </action>
    : : : : : : : :
</actions>
```

Where,

{action name}: Unique identifier for the action. Note that in order to be compliant with the ESC object model, for success or failure actions, the name must start with either TRUE or FALSE.

{action type}: Action type in this current release can be either ESC_POST_EVENT, SCRIPT or CUSTOM_SCRIPT.

{monitoring engine action type}: The monitoring engine type are the following: icmp_ping, icmp4_ping, icmp6_ping, esc_post_event, script, custom_script, snmp_get. See Monitoring the VNFs for more details.

Core and Default Actions List

Table 2: Core and Default Actions List

Name	Type	Description
TRUE esc_vm_alive_notification	Core	Start Service
TRUE servicebooted.sh	Core/Legacy	Start Service
FALSE recover autohealing	Core	Recover Service
TRUE servicescaleup.sh	Core/Legacy	Scale Out
TRUE esc_vm_scale_out_notification	Core	Scale Out
TRUE servicescaledown.sh	Core/Legacy	Scale In
TRUE esc_vm_scale_in_notification	Core	Scale In
TRUE apply_netscaler_license.py	Default	Apply Netscaler License

The core actions and metrics are defined by ESC and cannot be removed or updated.

The default actions or metrics are defined by ESC and exist to supplement core actions or metrics for more complex monitoring capabilities. These can be deleted and modified by the user. The default actions or metrics are reloaded on ESC startup every time an action or a metric with the same name cannot be found in the database.

Metric APIs

Table 3: Mapped Metrics

User Operation	Path	HTTP Operation	Payload	Response	Description
Read	internal/dynamic_mapping/actions/<metric_name>	GET	N/A	Metric XML	Get metrics by name
Read All	internal/dynamic_mapping/metrics/	GET	N/A	Metric XML	Get all metrics defined
Write	internal/dynamic_mapping/metrics/	POST	Metrics XML	Expected Metrics XML	Create one or multiple metrics
Delete	internal/dynamic_mapping/actions/<metric_name>	DELETE	N/A	N/A	Delete metric by name
Clear All	internal/dynamic_mapping/metrics	DELETE	N/A	N/A	Delete all non-core metrics

The response for the Metric APIs is as follows:

```
<metrics>
    <metric>
        <name>{metric name}</name>
        <type>{metric type}</type>
        <metaData>
            <type>{monitoring engine action type}</type>
            <properties>
                <property>
                    <name />
                    <value />
                </property>
                : : : : : : :
            </properties>
        </metaData>
    </metric>
    : : : : : : : : :
</metrics>
```

Where,

{metric name}: Unique identifier for the metric.

{metric type}: Metric type can be either MONITOR_SUCCESS_FAILURE, MONITOR_THRESHOLD or MONITOR_THRESHOLD_COMPUTE.

{monitoring engine action type}: The monitoring engine type are the following: icmp_ping, icmp4_ping, icmp6_ping, esc_post_event, script, custom_script, snmp_get. See Monitoring for more details.

Core and Default Metrics List

Table 4: Core and Default Metrics List

Name	Type	Description
ICMPPING	Core	ICMP Ping
MEMORY	Default	Memory compute percent usage
CPU	Default	CPU compute percent usage
CPU_LOAD_1	Default	CPU 1 Minute Average Load
CPU_LOAD_5	Default	CPU 5 Minutes Average Load
CPU_LOAD_15	Default	CPU 15 Minutes Average Load
PROCESSING_LOAD	Default	CSR Processing Load
OUTPUT_TOTAL_BIT_RATE	Default	CSR Total Bit Rate
SUBSCRIBER_SESSION	Default	CSR Subscriber Session

ESC Service Deployment

The KPI section defines the new KPI using the monitoring metrics.

```
<kpi>
    <event_name>DEMO_SCRIPT_SCALE_OUT</event_name>
```

```
    <metric_value>20</metric_value>
    <metric_cond>GT</metric_cond>
    <metric_type>UINT32</metric_type>
    <metric_collector>
        <type>custom_script_count_sessions</type>
        <nicid>0</nicid>
        <poll_frequency>15</poll_frequency>
        <polling_unit>seconds</polling_unit>
        <continuous_alarm>false</continuous_alarm>
    </metric_collector>
</kpi>
<kpi>
    <event_name>DEMO_SCRIPT_SCALE_IN</event_name>
    <metric_value>1</metric_value>
    <metric_cond>LT</metric_cond>
    <metric_type>UINT32</metric_type>
    <metric_occurrences_true>1</metric_occurrences_true>
    <metric_occurrences_false>1</metric_occurrences_false>
    <metric_collector>
        <type>custom_script_count_sessions</type>
        <nicid>0</nicid>
        <poll_frequency>15</poll_frequency>
        <polling_unit>seconds</polling_unit>
        <continuous_alarm>false</continuous_alarm>
    </metric_collector>
</kpi>
```

In the above sample, in the first KPI section, the metric identified by *custom_script_count_sessions* is executed at regular interval of 15 seconds. If the value returned by the metric is greater than 20, then the event name DEMO_SCRIPT_SCALE_OUT is triggered to be processed by the rules section.

In the above sample, in the second KPI section,The metric identified by custom_script_count_sessions is executed at regular interval of 15 seconds. If the value returned by the metric is less than 1, then the event name DEMO_SCRIPT_SCALE_IN is triggered to be processed by the rules section.

The rules section defines rules using the event_name that have been used by kpis. The action tag will define an action that will be executed when the event_name is triggered. In the example below, the action identified by the TRUE ScaleOut identifier is executed when the event DEMO_SCRIPT_SCALE_OUT is triggered.

```
<rule>
    <event_name>DEMO_SCRIPT_SCALE_OUT</event_name>
    <action>ALWAYS log</action>
    <action>TRUE ScaleOut</action>
</rule>
<rule>
    <event_name>DEMO_SCRIPT_SCALE_IN</event_name>
    <action>ALWAYS log</action>
    <action>TRUE ScaleIn</action>
</rule>
```

Script Actions

There are two types of actions supported:

1. Pre-Defined actions
2. Script actions

You can specify script execution as part of the Policy-driven data model. The *script_filename* property is mandatory to script actions, which specifies the absolute path to the script on the ESC VM. The following XML snippet shows a working example of a script action:

```
<action>
    <name>GEN_VPC_CHASSIS_ID</name>
    <type>SCRIPT</type>
    <properties>
        <property>
            <name>script_filename</name>
            <value>/opt/cisco/esc/esc-scripts/esc_vpc_chassis_id.py</value>
        </property>
        <property>
            <name>CHASSIS_KEY</name>
            <value>164c03a0-eebb-44a8-87fa-20c791c0aa6d</value>
        </property>
    </properties>
</action>
```

The script timeout is 15 minutes by default. However, you can specify a different timeout value for each script by adding a *wait_max_timeout* property to the properties section. The following example shows how to set the timeout to 5 minutes only for this script:

```
<action>
    <name>GEN_VPC_CHASSIS_ID</name>
    <type>SCRIPT</type>
    <properties>
        <property>
            <name>script_filename</name>
            <value>/opt/cisco/esc/esc-scripts/esc_vpc_chassis_id.py</value>
        </property>
        <property>
            <name>CHASSIS_KEY</name>
            <value>164c03a0-eebb-44a8-87fa-20c791c0aa6d</value>
        </property>
        <property>
            <name>wait_max_timeout</name>
            <value>300</value>
        </property>
    </properties>
</action>
```

In the above example, GEN_VPC_CHASSIS_ID will have a timeout value of 300 seconds, i.e. 5 mins. ESC also has a global parameter specifying the default timeout time for all the scripts that are being executed, called SCRIPT_TIMEOUT_SEC in the MONA category. This serves as the default value unless a wait_max_timeout property is defined in the script.

Triggering Pre-defined Actions

ESC introduces a new REST API to trigger the existing (pre-defined) actions defined through the Dynamic Mapping API, when required. For more information on the Metrics and Actions APIs, see Metrics and Actions APIs, on page 12.

A sample predefined action is as follows:

```
<actions>
    <action>
        <name>SaidDoIt</name>
        <userlabel>My Friendly Action</userlabel>
        <type>SCRIPT</type>
        <metaData>
            <type>script</type>
            <properties>
```

```
                <property>
                    <name>script_filename</name>
                    <value>/opt/cisco/esc/esc-scripts/do_somethin.py</value>
                </property>
                <property>
                    <name>arg1</name>
                    <value>some_val</value>
                </property>
                <property>
                    <name>notification</name>
                    <value>true</value>
                </property>
            </properties>
        </metaData>
    </action>
</actions>
```

Note A script file located on a remote server is also supported. You must provide the details in the <value> tag, for example,
http://myremoteserverIP:80/file_store/do_somethin.py</value>http://myremoteserverIP:80/file_store/do_somethin.py</value>

The pre-defined action mentioned above is triggered using the trigger API.

Execute the following HTTP or HTTPS POST operation:

POST `http://<IP_ADDRESS>:8080/ESCManager/v0/trigger/action/`

POST `https://<IP_ADDRESS>:8443/ESCManager/v0/trigger/action/`

The following payload shows the actions triggered by the API, and the response received:

```
<triggerTarget>
  <action>SaidDoIt</action>
  <properties>
   <property>
     <name>arg1</name>
     <value>real_value</value>
   </property>
  </properties>
</triggerTarget>
```

The response,

```
<triggerResponse>
    <handle>c11be5b6-f0cc-47ff-97b4-a73cce3363a5</handle>
    <message>Action : 'SAIDDOIT' triggered</message>
</triggerResponse>
```

ESC accepts the request, and returns a response payload and status code.

An http status code of 200 indicates that the action triggered exists, and is triggered successfully. An http status codes of 400 or 404 indicate that the action to be triggered is not found.

You can determine the status using the custom script notifications sent to NB at various lifecycle stages.

ESC sends the MANUAL_TRIGGERED_ACTION_UPDATE callback event to NB with a status message that describes the success or failure of the action execution.

The notification is as follows:

```
<esc_event xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
```

```
    <event_type>MANUAL_TRIGGERED_ACTION_UPDATE</event_type>
    <properties>
        <property>
            <name>handle</name>
            <value>c11be5b6-f0cc-47ff-97b4-a73cce3363a5</value>
        </property>
        <property>
            <name>message</name>
            <value>Action execution success</value>
        </property>
        <property>
            <name>exit_code</name>
            <value>0</value>
        </property>
        <property>
            <name>action_name</name>
            <value>SAIDDOIT</value>
        </property>
    </properties>
</esc_event>
```

Note The script_filename property cannot be overwritten by the trigger API request. The trigger API must not contain any additional properties that do not exist in the predefined action.

The new API allows to overrides some of the special properties (of the actions) listed below:

- Notification—Set this if your script generates progress notifications at run time. The default value is false. This value can be set to true in the action or trigger payload.
- wait_max_timeout—Wait for the script to complete the execution before terminating. The default wait timeout is 900 seconds.

Note

- The trigger API supports only script type actions.
- Ensure that the script action located on the ESC VM is copied to the same path on both the Master and Backup HA instances. For more information, see the High Availability chapter in the *Cisco Elastic Services Controller Install and Upgrade Guide*.
- The script execution terminates if there is a failover, shutdown, or reboot of the ESC services.

Configuring Custom Script Metric Monitoring KPIs and Rules

Custom Script Metric Monitoring can be performed as follows:

1. Create Script
2. Add Metric
3. Add Action
4. Define Deployment
5. Update KPI data or Rules

6. Authenticating Remote Server Using KPIs and Rules

The script to be executed has to be compliant with the rules specified for a MONITOR_THRESHOLD action. Threshold crossing evaluation will be based on the exit value from the script execution. In the sample script below, the return value is the number of IP sessions.

```
#!/usr/bin/env python
import pexpect
import re
import sys
ssh_newkey = 'Are you sure you want to continue connecting'
# Functions
def get_value(key):
    i = 0
    for arg in sys.argv:
        i = i + 1
        if arg == key:
            return sys.argv[i]
    return None
def get_ip_addr():
    device_ip = get_value("vm_ip_address")
    return device_ip
# Main
CSR_IP = get_ip_addr()

p=pexpect.spawn('ssh admin@' + CSR_IP + ' show ip nat translations total')
i=p.expect([ssh_newkey,'assword:',pexpect.EOF])
if i==0:
    p.sendline('yes')
    i=p.expect([ssh_newkey,'assword:',pexpect.EOF])
if i==1:
    p.sendline("admin")
    p.expect(pexpect.EOF)
elif i==2:
    pass
n = p.before
result = re.findall(r'\d+', n)[0]
sys.exit(int(result))
```

The ESC monitoring and action engine processes the script exit value.

The script has to be installed into the following ESC VM directory: `/opt/cisco/esc/esc-scripts/`

The following payload describes a metric using a custom_script defined in the script

```
<!-- Demo Metric Counting Sessions -->
<metrics>
    <metric>
        <name>custom_script_count_sessions</name>
        <type>MONITOR_THRESHOLD</type>
        <metaData>
            <properties>
                <property>
                    <name>script_filename</name>
                    <value>/cisco/esc-scripts/countSessions.py</value>
                </property>
                <property>
                    <name>for_threshold</name>
                    <value>true</value>
                </property>
            </properties>
```

```
            <type>custom_script_threshold</type>
        </metaData>
    </metric>
</metrics>
<!-- -->
```

The metric payload has to be added to the list of supported ESC metrics by using the Mapping APIs.

Execute a HTTP POST operation on the following URI:

http://<my_esc_ip>:8080/ESCManager/internal/dynamic_mapping/metrics

The following payload describes custom actions that can be added to the list of supported ESC actions by using the Mapping APIs.

```
<actions>
    <action>
        <name>TRUE ScaleOut</name>
        <type>ESC_POST_EVENT</type>
        <metaData>
            <type>esc_post_event</type>
            <properties>
                <property>
                    <name>esc_url</name>
                    <value />
                </property>
                <property>
                    <name>vm_external_id</name>
                    <value />
                </property>
                <property>
                    <name>vm_name</name>
                    <value />
                </property>
                <property>
                    <name>event_name</name>
                    <value />
                </property>
                <property>
                    <name>esc_event</name>
                    <value>VM_SCALE_Out</value>
                </property>
                <property>
                    <name>esc_config_data</name>
                    <value />
                </property>
                <properties />
            </properties>
        </metaData>
    </action>
    <action>
        <name>TRUE ScaleIn</name>
        <type>ESC_POST_EVENT</type>
        <metaData>
            <type>esc_post_event</type>
            <properties>
                <property>
                    <name>esc_url</name>
                    <value />
                </property>
                <property>
                    <name>vm_external_id</name>
                    <value />
                </property>
```

```
                <property>
                    <name>vm_name</name>
                    <value />
                </property>
                <property>
                    <name>event_name</name>
                    <value />
                </property>
                <property>
                    <name>esc_event</name>
                    <value>VM_SCALE_IN</value>
                </property>
                <properties />
            </properties>
        </metaData>
    </action>
</actions>
```

Execute a HTTP POST operation on the following URI:

http://<IP_ADDRESS>:8080/ESCManager/internal/dynamic_mapping/actions

Custom Script Notification

ESC now supports sending notification to northbound about customized scripts run as part of the deployment at a certain lifecycle stage. You can also determine the progress of the script executed through this notification. To execute a custom script with notification, define action type attribute as *SCRIPT*, and property attribute name as *notification*, and set the value to true.

For example, in the datamodel below, the action is to run a customized script located at `/opt/cisco/esc/esc-scripts/senotification.py` with notification, when the deployment reaches POST_DEPLOY_ALIVE stage.

```
<policies>
    <policy>
        <name>PCRF_POST_DEPLOYMENT</name>
        <conditions>
            <condition>
                <name>LCS::POST_DEPLOY_ALIVE</name>
            </condition>
        </conditions>
        <actions>
            <action>
                <name>ANY_NAME</name>
                <type>SCRIPT</type>
                <properties>
                    <property>
                        <name>script_filename</name>
                        <value>/opt/cisco/esc/esc-scripts/senotification.py</value>
                    </property>
                    <property>
                        <name>notification</name>
                        <value>true</value>
                    </property>
                </properties>
            </action>
        </actions>
    </policy>
</policies>
```

You can notify northbound about the script execution progress using the following outputs:

- Standard JSON output

- REST API call

Standard JSON Output

The standard JSON output follows the MONA notification convention. MONA captures entries in this to generate notification.

```
{"esc-notification":{"items":{"properties":
[{"name":"name1","value":"value1"},{"name":"name2","value":"value2"}...]}}}
```

The items are listed in the table below.

Table 5: Item list

Name	Description
type	Describes the type of notification. `progress_steps | progress_percentage | log | alert | error`
progress **Note** Progress item is required only when the type is progress-steps or progress-percentage.	For progress-steps type, `{current_step}|{total_steps}`
	For progress-percentage type, `{percentage}`
msg	Notification message.

Example JSON output is as follows:

```
{"esc-notification":{"items":{"properties": [{"name":"type",
"value":"progress_percentage"},{"name":"progress","value":"25"},{"name":"msg","value":"Installation
in progress."}]}}}
```

Note If the custom script is written in Python, because standard output is buffered by default, after each notification print statement, the script is required to call sys.stdout.flush() to flush the buffer (for pre Python 3.0). Otherwise MONA cannot process the script stdout in a real-time. `print '{"esc-notification":{"items":{"properties": [{"name":"type", "value":"progress_percentage"},{"name":"progress","value":"25"},{"name":"msg","value":"Installation in progress."}]}}}'sys.stdout.flush()`

REST API Call

```
http://localhost:8090/mona/v1/actions/notification
```

For REST API, the script must accept a script handle as the last parameter. The script handle can be UUID, MONA action or execution job Id. For example, if the script originally accepts 3 command line parameters, to support MONA notification, the script considers an additional parameter for the handle UUID. This helps MONA to identify the notification source. For every notification, the script is responsible for constructing a POST REST call to MONA's endpoint inside the script:

The payload is as follows:

```
{
 "esc-notification" : {
  "items" : {
   "properties" : [{
     "name" : "type",
     "value" : "log",
     "hidden" : false
    }, {
     "name" : "msg",
     "value" : "Log info",
     "hidden" : false
    }
   ]
  },
  "source" : {
   "action_handle" : "f82fe86d-6625-4b13-99f7-89d169e427ad"
  }
 }
}
```

Note The action_handle value is the handle UUID MONA passes into the script.

Policy-Driven Data model

ESC supports a new policy-driven datamodel. A new <policy> section is introduced under <policies> at both deployment and VM group level.

Using the Policy Data model, a user can perform actions based on conditions. ESC supports predefined actions, or customized scripts during a deployment based on certain Lifecycle Stage (LCS). For example, the redeployment policy uses predefined actions based on lifecycle stages (LCS) to redeploy VMs. For more information, see Redeployment Policy.

Policy Data model

The policy data model consists of conditions and actions. The condition is a Lifecycle Stage (LCS) in a deployment. The action is predefined or custom script.

- Predefined action—The action is predefined and executed when the condition is met.

  In the datamodel below, when condition2 is met, Action2 is performed. The action <type> is predefined.

- Custom Script—The action is a custom script, and executed when the condition is met.

  In the datamodel below, when condition1 is met, Action1-1 and Action 1-2 are executed. The action <type> is script.

```
<policies>
    <policy>
        <name>Name1</name>
        <conditions>
            <condition>
                <name>Condition1</name>
            </condition>
        </conditions>
        <actions>
            <action>
                <name>Action1-1</name>
```

```
                <type>SCRIPT</type>
            </action>
            <action>
                <name>Action1-2</name>
                <type>SCRIPT</type>
            </action>
        </actions>
    </policy>
    <policy>
        <name>Name2</name>
        <conditions>
            <condition>
                <name>Condition2</name>
            </condition>
        </conditions>
        <actions>
            <action>
                <name>Action2</name>
                <type>PRE-DEFINED</type>
            </action>
        </actions>
    </policy>
</policies>
```

For more information on Predefined actions, and scripts, see Recovery and Redeployment Policies.

The table below shows the LCS in a deployment, and its description. The recovery and redeployment policies, and VNF software upgrade policies use the policy-driven data model. These policies are supported on both single deployment and multi VIM deployment. For more information, see "Deploying Virtual Network Functions". For details on configuring the recovery and redeployment policies using the policy framework, see Recovery and Redeployment Policies. For details on upgrading the VNF software upgrade policies, see Upgrading the Virtual Network Function Software Using Lifecycle Stages.

Supported Lifecycle Stages (LCS)

Table 6: Conditions and their Scope

Condition Name	Scope	Description
LCS::PRE_DEPLOY	Deployment	Occurs just before deploying VMs of the deployment.
LCS::POST_DEPLOY_ALIVE	Deployment	Occurs immediately after the deployment is active.
LCS::DEPLOY_ERR	Deployment	Occurs immediately after the deployment fails.
LCS::POST_DEPLOY:: VM_RECOVERY_ERR	Deployment	Occurs immediately after the recovery of one VM fails. (This is specified at deployment level and applies to all VM groups)

LCS::POST_DEPLOY:: VM_RECOVERY _REDEPLOY_ERR	Deployment	Occurs immediately after the redeployment of one VM fails. (This is specified at deployment level and applies to all VM groups)
LCS::DEPLOY_UPDATE::VM_ PRE_VOLUME_DETACH	Deployment	Triggered just before the ESC detaches a volume. (This is specified for a group of individual VMs and specified under <vm_group> rather than the entire deployment.)
LCS::DEPLOY_UPDATE:: VM_VOLUME_ATTACHED	Deployment	Triggered immediately after ESC has attached a new volume (This is specified for a group of individual VMs and specified under <vm_group> rather than the entire deployment.)
LCS::DEPLOY_UPDATE:: VM_SOFTWARE_VERSION_UPDATED	Deployment	Triggered immediately after ESC has updated the software version of the VM (This is specified for a group of individual VMs and specified under <vm_group> rather than the entire deployment.)

Fetching Files From Remote Server Using LCS Actions

Prior to ESc Release 4.0, a file locator is added to the LCS action scripts to fetch external configuration files. The file locator contains a reference to the file server, and the relative path to the file to be downloaded. Starting from ESC Release 4.0, the file locator attribute is defined at the deployment level, that is, directly under the deployment container instead of policy actions and day 0 configuration sections.

```
<esc_datamodel xmlns="http://www.cisco.com/esc/esc">
  <tenants>
    <tenant>
      <name>test-tenant</name>
      <deployments>
        <deployment>
          <name>test-deployment</name>
          <file_locators>
            <file_locator>
              <name>custom_bool_action</name>
              <remote_file>
                <file_server_id>http-my-server</file_server_id>
                <remote_path>share/qatest/custom_bool_action.sh</remote_path>
              </remote_file>
            </file_locator>
            <file_locator>
              <name>custom_bool_metric</name>
              <remote_file>
                <file_server_id>http-my-server</file_server_id>
                <remote_path>/share/qatest/custom_bool_metric.sh</remote_path>
              </remote_file>
            </file_locator>
          </file_locators>
          <!-- truncated for brevity -->
          <vm_group>
            <name>ASA-group</name>
```

```
            <!-- truncated for brevity -->
            <kpi_data>
              <kpi>
                <event_name>MY_CUSTOM_BOOL_ACTION</event_name>
                <metric_value>5</metric_value>
                <metric_cond>LT</metric_cond>
                <metric_type>UINT32</metric_type>
                <metric_occurrences_true>1</metric_occurrences_true>
                <metric_occurrences_false>1</metric_occurrences_false>
                <metric_collector>
                  <type>MY_CUSTOM_BOOL_METRIC</type>
                  <nicid>0</nicid>
                  <poll_frequency>3</poll_frequency>
                  <polling_unit>seconds</polling_unit>
                  <continuous_alarm>false</continuous_alarm>
                  <properties>
                    <!-- Add file locator reference here -->
                    <property>
                      <name>file_locator_name</name>
                      <value>custom_bool_action</value>
                    </property>
                  </properties>
                </metric_collector>
              </kpi>
            </kpi_data>
            <rules>
              <admin_rules>
                <rule>
                  <event_name>MY_CUSTOM_BOOL_ACTION</event_name>
                  <action>ALWAYS log</action>
                  <action>TRUE my_custom_bool_action</action>
                  <properties>
                    <!-- Add file locator reference here -->
                    <property>
                      <name>file_locator_name</name>
                      <value>custom_bool_action</value>
                    </property>
                  </properties>
                </rule>
              </admin_rules>
            </rules>
          </vm_group>
        </deployment>
      </deployments>
    </tenant>
  </tenants>
</esc_datamodel>
```

See Fetching Files From Remote Server for more information.

To encrypt the files see, Encrypting Configuration Data.

Lifecycle Stage (LCS) Policy Conditions Defined at Different Stages

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

Table 7: LifeCycle Stages

Condition Name	Scope
LCS::VM::PRE_VM_DEPLOY	VM
LCS::VM::POST_VM_DEPLOYED	VM

Condition Name	Scope
LCS::VM::POST_VM_ALIVE	VM
Lifecycle Stages in Deployment	
LCS::PRE_DEPLOY	VM / Deployment
LCS::DEPLOY:: POST_VM_DEPLOYED	VM
LCS::POST_DEPLOY_ALIVE	Deployment
LCS::DEPLOY_ERR	Deployment
Lifecycle Stages in Deployment Update	
LCS::DEPLOY_UPDATE::POST_VM_ALIVE	VM
LCS::DEPLOY_UPDATE::	VM
LCS::DEPLOY_UPDATE:: POST_VM_VOLUME_DETACHED	VM
LCS::DEPLOY_UPDATE:: POST_VM_VOLUME_ATTACHED	VM
LCS::DEPLOY_UPDATE:: PRE_VM_SOFTWARE_VERSION_UPDATED	VM
Lifecycle Stages in Recovery	
LCS::POST_DEPLOY:: POST_VM_RECOVERY_COMPLETE	VM
LCS::POST_DEPLOY:: VM_RECOVERY_ERR	VM
Lifecycle Stages in Recovery and Redeploy	
LCS::POST_DEPLOY:: VM_RECOVERY_REDEPLOY_ERR	VM

Affinity and Anti-Affinity Rules

Affinity and anti-affinity rules create relationship between virtual machines (VMs) and hosts. The rule can be applied to VMs, or a VM and a host. The rule either keeps the VMs and hosts together (affinity) or separated (anti-affinity).

Policies are applied during individual VM deployment. You can deploy a single VNF or multiple VNFs together through ESC portal by uploading an existing deployment datamodel or by creating a new deployment datamodel. For more information, see ESC Portal Dashboard.

Affinity and anti-affinity policy streamlines the deployment process.

Affinity and anti-affinity rules are created and applied on VMs at the time of deployment. VM receives the placement policies when the deploy workflow is initialized.

During a composite VNF deployment, if a couple of VMs need to communicate with each other constantly, they can be grouped together (affinity rule) and placed on the same host.

If two VMs are over-loading a network, they can be separated (anti-affinity rule) and placed on different hosts to balance the network.

Grouping or separating VMs and hosts at the time of deployment helps ESC to manage load across the VMs and hosts in the network. Recovery and scale out of these VMs do not impact the affinity and anti-affinity rules.

The anti-affinity rule can also be applied between VMs within the same group and on a different host. These VMs perform similar functions and support each other. When one host is down, the VM on the other host continues to run preventing any loss of service.

The table shows the types of affinity and anti-affinity policies in a deployment.

Table 8: Intra and Inter group affinity and anti-affinity policies

Policy	Policy	VM group	Host	Zone
affinity	Intra group affinity	same VM group	same host	same zone
	Inter group affinity	different VM group	same host	same zone
anti-affinity	Intra group anti-affinity	same VM group	different host	same zone
	Inter group anti-affinity	different VM group	different host	same zone

Note If the zone is not specified on OpenStack, VMs will be placed on different hosts and different zones for inter and intra group anti-affinity rules.

Affinity and Anti-Affinity Rules on OpenStack

The following sections describe affinity and anti-affinity policies with examples.

Intra Group Affinity Policy

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

Example for Intra Group Affinity Policy:

```
<vm_group>
     <name>affinity-test-gp</name>
     <placement>
 <type>affinity</type>
 <enforcement>strict</enforcement>
     </placement>
...
```

The type *zone-host* is used to deploy VNFs in the same host, or into the same availability zone.

Zone or Host Based Placement

The VNFs are within the same VM group and deployed on the same host or the same available zone. The *host* tag is used to deploy VMs on the same host and the *zone* tag is used to deploy VMs in the same available zone. Before deploying, you need to make sure that the host exists in OpenStack. ESC validates the specified host on OpenStack. The *zone-host* tag specifies the type of placement. Hence, if a host or a zone is not specified during a deployment, the deployment fails.

Important You cannot specify both the host and zone tags to deploy VM on the same host or the same available zone.

Example for host placement:

```
<vm_group>
     <name>zone-host-test-gp1</name>
     <placement>
 <type>zone_host</type>
 <enforcement>strict</enforcement>
 <host>my-server</host>
     </placement>
...
```

Example for zone placement:

```
<vm_group>
     <name>zone-host-test-gp2</name>
     <placement>
 <type>zone_host</type>
 <enforcement>strict</enforcement>
        <zone>dt-zone</zone>
     </placement>
...
```

Intra Group Anti-Affinity Policy

The VNFs within the same VM group are explicitly deployed on different hosts. For example, back-up VNFs.

Example for Intra Group anti-affinity Policy:

```
<vm_group>
        <name>anti-affinity-test-gp</name>
        <placement>
            <type>anti_affinity</type>
            <enforcement>strict</enforcement>
        </placement>
...
```

Inter Group Affinity Policy

The VNFs in the same deployment but different VM groups can be explicitly deployed in the same host. For example VNF bundles. Multiple VM groups can follow this policy by adding the vm_group_ref tag and providing the VM group name as the value.

Note You can use one or more vm_group_ref tag, type tag and enforcement tag under the placement tag. The host or zone cannot be specified.

Example for Inter Group Affinity Policy:

```
<deployments>
            <deployment>
            <name>intergroup-affinity-dep</name>
             <policies>
              <placement>
                <target_vm_group_ref>affinity-test-gp1</target_vm_group_ref>
                <type>affinity</type>
                <vm_group_ref>affinity-test-gp2</vm_group_ref>
                <enforcement>strict</enforcement>
               </placement>
          </policies>
…
```

Inter Group Anti-Affinity Policy

The VNFs in the same deployment but different VM Groups can be explicitly deployed in different hosts. For example back-up VNFs or High-availability VNFs. Multiple VM groups can follow this policy by adding the vm_group_ref tag, and providing the VM group name as the value.

Note You can only use one <target_vm_group_ref> tag, type tag and enforcement tag under the placement tag. The host or zone cannot be specified.You can use multiple <vm_group_ref> tags, however the anti-affinity policy only applies between each <vm_group_ref> and their <target_vm_group_ref>, which means that 2 or more <vm_group_ref> can be deployed on the same host, as long as each of them are deployed on a different host from their <target_vm_group_ref> that is acceptable.

Example for Inter Group anti-affinity Policy:

```
<deployments>
           <deployment>
            <name>intergroup-anti_affinity-dep</name>
             <policies>
              <placement>
                  <target_vm_group_ref>affinity-test-gp1</target_vm_group_ref>
                  <type>anti_affinity</type>
                  <vm_group_ref>affinity-test-gp2</vm_group_ref>
                  <enforcement>strict</enforcement>
                </placement>
          </policies>
…
```

In a multiple VIM deployment, the VM groups of a placement policy must belong to the same VIM. That is, the VIM connector must be the same for the VM groups (specified in the vim_id attribute in the locator tag

of the VM group). ESC rejects a deployment if the affinity and anti-affinity policies between VM groups are on different VIMs. For more details on deploying VMs on multiple deployments, see "Deploying VNFs onMultiple OpenStack VIMs".

A placement group tag is added under policies. Each <placement_group> contains the following:

- name—name unique per deployment.
- type—affinity or anti_affinity
- enforcement—strict
- vm_group—the content of each vm_group must be a vm group name listed under the same deployment.

The placement group tag is placed within the placement policy. The placement policy describes the relationship between the target vm group and the vm group members. The placement_group policy describes mutual relationship among all vm group members. The placement group policy is not applicable for target vm group.

The datamodel is as follows:

```
<policies>
 <placement_group>
  <name>placement-affinity-1</name>
  <type>affinity</type>
  <enforcement>strict</enforcement>
  <vm_group>t1g1</vm_group>
  <vm_group>t1g2</vm_group>
  <vm_group>t1g7</vm_group>
 </placement_group>
 <placement_group>
  <name>placement-affinity-2</name>
  <type>affinity</type>
  <enforcement>strict</enforcement>
  <vm_group>t1g3</vm_group>
  <vm_group>t1g4</vm_group>
 </placement_group>
 <placement_group>
  <name>placement-affinity-3</name>
  <type>affinity</type>
  <enforcement>strict</enforcement>
  <vm_group>t1g5</vm_group>
  <vm_group>t1g6</vm_group>
 </placement_group>
 <placement_group>
  <name>placement-anti-affinity-1</name>
  <type>anti_affinity</type>
  <enforcement>strict</enforcement>
  <vm_group>t1g1</vm_group>
  <vm_group>t1g3</vm_group>
  <vm_group>t1g5</vm_group>
 </placement_group>
</policies>
```

Note In the new placement group tag under policies, the <target_vm_group_ref> and <vm_group_ref> are replaced with <vm_group>. The ref based affinity and antiaffinity tags are deprecated.

The placement group policy is applicable for inter group affinity and anti-affinity policies only.

You cannot use both placement and placement group tags together in the inter group affinity and anti-affinity policies.

The placement group name tag must be unique for each placement group policy.

Inter Deployment Anti-Affinity Policy

Inter Deployment anti-affinity rules define relationships between different deployments with respect to the host placement. Anti-affinity between deployments is defined such that any VM from one deployment is not co-located on the same host as any other VM from the other deployment.

Note Inter Deployment anti-affinity is supported on OpenStack only. Inter Deployment anti-affinity does not work with host-placement (affinity or anti-affinity) as the latter takes precedence over inter deployment anti-affinity rules.

In the ESC datamodel, inter deployment anti-affinity is defined using anti-affinity groups. All member deployments of an anti-affinity group have an anti-affinity relationship between them. For example, in an anti-affinity group called default-anti with 3 deployments dep-1, dep-2 and dep-3, dep-1 is anti-affinity to dep-2 and dep-3 deployments, dep-2 is anti-affinity to dep-1 and dep-3 deployments, dep-3 is anti-affinity dep-1 and dep-2. A deployment specifies its membership in an anti-affinity group by referencing to all group names it pertains to as shown below.

```
<deployment>
<name>VPC-dep</name>
 <deployment_groups>
  <anti_affinity_group>VPC-ANTI-AFFINITY</anti_affinity_group>
  <anti_affinity_group>VPNAAS-ANTI-AFFINITY</anti_affinity_group>
 </deployment_groups>
….
</deployment>
```

In the above example, VPC-dep is in 2 anti-affinity groups; any other deployment that references one of these 2 groups will have an anti-affinity relationship with VPC-dep.

Inter-deployment Placement Groups

Anti-affinity group is an example of placement group. Anti-affinity group has the following properties in ESC:

- The placement group need not be created or deleted.
- Placement groups can be referenced for the first time by one deployment as well as multiple deployments in parallel.
- Placement rules are applicable during any deployment phase of a service including:
    - Initial deployment

- Scale Out
- VM group update addition
- VM group minimum scaling update (increasing minimum scaling to add VMs)
- Recovery

A multiple VIM deployment, supports Inter-deployment anti-affinity. However, ESC rejects a deployment

- If the inter-deployment anti-affinity policy is defined between a multiple VIM deployment (with locators within VM groups) and a default VIM deployment (without locators).
- If all the deployments of an inter-deployment anti-affinity group are not deployed on the same VIM (with same vim_id). For more details on a multiple VIM deployment, see Deploying VNFs on Multiple OpenStack VIMs.

Affinity and Anti-Affinity Rules on VMware vCenter

The affinity and anti-affinity rules for VMware vCenter is explained with examples. These rules are created for a cluster and a targeted host.

All VMware vCenter deployments must always be accompanied with zone-host placement policy. The zone-host defines the target VM group which is either the cluster or the host.

Intra Group Affinity Policy

The VNFs with the same VM group can be deployed on the same host.

During deployment, ESC deploys the first VM as an anchor VM for affinity. All the other VMs that follow the same affinity rule will be deployed to the same host as the anchor VM. The anchor VM deployment helps to optimize the resource usage.

Example for Intra Group Affinity Policy:

```
…
<vm_group>
<name>vm-gp</name>
…
<placement>
<type>zone_host</type>
<enforcement>strict</enforcement>
<zone>cluster1</zone>
</placement>
<placement>
<type>affinity</type>
<enforcement>strict</enforcement>
</placement>
…
```

Note Only *strict* attribute is supported for enforcement.

Note Affinity and anti-affinity policy with a host placement policy is incorrect and may cause deployment failure. Host placement alone (without affinity and anti-affinity placement policy within a VM group) can be used to achieve intra group affinity.

Intra Group Anti-Affinity

The VNFs with the same VM group can be deployed in different hosts. During deployment ESC deploys VNFs with the same VM group one after the other. At the end of each VNF deployment, ESC records its host to a list. At the beginning of each VNF's deployment, ESC deploys the VNF to a computing-host that is not in the list. If all the available computing-host(s) are in the list, ESC fails the whole deployment.

Example for Intra Group Anti-Affinity Policy:

```
…
<vm_group>
<name>vm-gp</name>
…
<placement>
<type>zone_host</type>
<enforcement>strict</enforcement>
<zone>cluster1</zone>
</placement>
<placement>
<type>anti_affinity</type>
<enforcement>strict</enforcement>
</placement>
```

Cluster Placement

All VMs in a VM group can be deployed to a cluster. For example, all VMs in a vm group CSR-gp1 can be deployed to cluster dc-cluster2.

Note The VMware vCenter cluster must be created by the administrator.

Example for cluster placement:

```
<name>CSR-gp1</name>
          <placement>
               <type>zone_host</type>
               <enforcement>strict</enforcement>
               <zone>dc-cluster2</zone>
           </placement>
```

Host Placement

All VMS in a VM group can be deployed to a host. For example, all VMs in the vm group CSR-gp1 will be deployed to host 10.2.0.2.

```
<name>CSR-gp1</name>
           <placement>
             <type>zone_host</type>
             <enforcement>strict</enforcement>
              <host>10.2.0.2</host>
            </placement>
```

Inter Group Affinity Policy

The VMs in different VM groups can be deployed to the same host. For example, all VMs in the VM group ASA-gp1 can be deployed to the same host as the VMs in the VM group CSR-gp1.

During deployment ESC deploys the first VM as an anchor VM. All other VMs that follow the same affinity rule will be deployed to the same host as the anchor VM.

Note To ensure that the inter-group affinity rules are applied within a single cluster, verify that all VM groups in a deployment are specified to the same cluster (<zone> in esc data_model).

Example for Inter Group Affinity Policy:

```
<deployment>
<deployment>
<name>test-affinity-2groups</name>
<policies>
<placement>
<target_vm_group_ref>CSR-gp1</target_vm_group_ref>
<type>affinity</type>
<vm_group_ref>CSR-gp2</vm_group_ref>
<vm_group_ref>ASA-gp1</vm_group_ref>
<enforcement>strict</enforcement>
</placement>
</policies>
```

Inter Group Anti-Affinity Policy

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

Example for Inter Group Anti-Affinity Policy:

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

A placement group tag is added under policies. Each <placement_group> contains the following:

- name—name unique per deployment.
- type—affinity or anti_affinity
- enforcement—strict
- vm_group—the content of each vm_group must be a vm group name listed under the same deployment.

The placement group tag is placed within the placement policy. The placement policy describes the relationship between the target vm group and the vm group members. The placement_group policy describes mutual relationship among all vm group members. The placement group policy is not applicable for target vm group.

The datamodel is as follows:

```
<policies>
 <placement_group>
  <name>placement-affinity-1</name>
  <type>affinity</type>
  <enforcement>strict</enforcement>
  <vm_group>t1g1</vm_group>
  <vm_group>t1g2</vm_group>
  <vm_group>t1g7</vm_group>
 </placement_group>
 <placement_group>
  <name>placement-affinity-2</name>
  <type>affinity</type>
  <enforcement>strict</enforcement>
  <vm_group>t1g3</vm_group>
  <vm_group>t1g4</vm_group>
 </placement_group>
 <placement_group>
  <name>placement-affinity-3</name>
  <type>affinity</type>
  <enforcement>strict</enforcement>
  <vm_group>t1g5</vm_group>
  <vm_group>t1g6</vm_group>
 </placement_group>
 <placement_group>
  <name>placement-anti-affinity-1</name>
  <type>anti_affinity</type>
  <enforcement>strict</enforcement>
  <vm_group>t1g1</vm_group>
  <vm_group>t1g3</vm_group>
  <vm_group>t1g5</vm_group>
 </placement_group>
</policies>
```

Note In the new placement group tag under policies, the <target_vm_group_ref> and <vm_group_ref> are replaced with <vm_group>. The ref based affinity and antiaffinity tags are deprecated.

The placement group policy is applicable for inter group affinity and anti-affinity policies only.

You cannot use both placement and placement group tags together in the inter group affinity and anti-affinity policies.

The placement group name tag must be unique for each placement group policy.

Limitations

Following are the limitations when affinity and anti-affinity rules are applied on VMware vCenter:

- All Affinity rules defined on VMware vCenter are implemented in a cluster.
- DPM, HA and vMotion must be turned off.
- VM deployment and recovery are managed by ESC.

- DRS must be set to manual mode if it is turned on.
- To leverage DRS deployment, shared storage is required.
- Supported value for <enforcement> tag should be 'strict'.
- <zone_host> must be used for any VM group.

Affinity and Anti-Affinity Rules on VMware vCloud Director

ESC supports affinity and anti-affinity placement policy for vCD. However, it does not support zone-host placement policy.

The affinity and anti-affinity implementation in ESC depends on the affinity rule (VM-VM affinity rule in vSphere) in the vCloud Director. The example below shows affinity and anti-affinity rules in the vCD VNF deployment datamodel.

```
<deployments>
        <deployment>
          <!-- vApp instance name -->
          <name>d1</name>
          <policies>
            <placement_group>
              <name>d1-placement-affinity-1</name>
              <type>affinity</type>
              <enforcement>strict</enforcement>
              <vm_group>g1</vm_group>
              <vm_group>g2</vm_group>
            </placement_group>
          </policies>
         ….…….…….…...
         ….…….…….…….
       </deployment>
    </deployments>
```

For vCD deployment, see Deploying Virtual Network Functions on VMware vCloud Director (vCD).

Configuring Custom VM Name

You can customize VM names if you do not want ESC to auto-generate VM names. To customize VM names, specify the `vim_vm_name` in the VM group section of the deployment datamodel. If `vim_vm_name` is not specified, ESC will auto-generate the VM names.

While specifying a custom name, if a VM group has more than one VM, an "_<index>" is appended to the custom VM name in the output. For example, the first VM in the group is named as specified in the `vim_vm_name`, and second VM onwards an index "_1", "_2" is appended to the custom name. For a custom name specified as ABC, the output will display the VM names as VMname, VMname_1, VMname_2, and so on. If a VM group only has a single VM, then there is no "_<index>" appended to the custom VM name.

A single deployment can contain multiple VM groups, and each individual VM group can specify a different `vim_vm_name` value, if required. For example, a deployment could have two VM groups: the first group specifies a `vim_vm_name` and all VMs have their names generated as described above. The second VM group does not specify a `vim_vm_name`, therefore all VM names created from this group are auto generated.

Custom VM names only have to be unique within the deployment and tenant for an OpenStack deployment. In other words, custom VM names can be duplicated across different tenants - or even duplicated within the

same tenant as long as it is for a different deployment. For a VMware deployment, the custom VM name must be unique throughout the entire vCenter server. In other words, no duplicate VM names are permitted.

Note You can use a maximum of 63 characters for the custom name. A VM name should not contain special characters and can only contain alphanumeric characters and "_" and "-".

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

    <vm_group>
    <name>CIRROS</name>
    <image>Automation-Cirros-Image</image>
    <flavor>Automation-Cirros-Flavor</flavor>
    <vim_vm_name>VMname</vim_vm_name>
    <scaling>
        <min_active>1</min_active>
        <max_active>2</max_active>
        <elastic>true</elastic>
    </scaling>
</vm_group>
```

Note

- The ESC Portal does not display the VM Name that was configured during the deployment time.
- Duplicate VM Names are not supported on VMWare.
- VM names cannot be modified after a deployment is complete.

The following are some output samples with the custom VM name. If the vim_vm_name was set during the deployment, the same value will be shown in the output. If this value was not set during the deployment, ESC will auto-generate the VM name.

- Below is an example of the output operational data fetched using the esc_nc_cli script after adding a custom VM name. A new element called <vmname> will be shown under the vm_group element. The value in the <status_message> field is also updated to reflect the custom VM name.

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="1">
  <data>
    <esc_datamodel xmlns="http://www.cisco.com/esc/esc">
      <opdata>
        <tenants>
          <tenant>
            <name>xyzzy</name>
            <deployments>
              <deployment_name>my-deployment-123</deployment_name>
              <deployment_id>78d48bf8-5f67-45fc-8d92-5ad4676yf57</deployment_id>
              <vm_group>
                <name>Grp1</name>
                <vm_instance>
                  <vm_id>df108144-ec4f-4d66-a62f-98096ecddef0</vm_id>
                  <name>VMname</name>
              </vm_group>
```

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

```
GET http://localhost:8080/ESCManager/v0/deployments/example-deployment-123
| xmllint --format -
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<deployment xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <datacenter>
    <default>false</default>
  </datacenter>
  <deployment_details>
    <host_uuid>8623f1476302a5815608dbd4c2f836c570e8c74cbfbaff41c78564b1</host_uuid>
    <host_name>my-server</host_name>
    <vm_uuid>e7e5a905-e0c7-4652-ae1f-23a409a58219</vm_uuid>
    <interfaces>
      <interface>

      </interface>
    </interfaces>
    <vm_group_name>Grp1</vm_group_name>
    <vm_name>VMname_1</vm_name><-- ##### custom vm name, single VM in the VM group, so
 no appended "_<index>" -->
    <vm_state_machine_state>VM_ALIVE_STATE</vm_state_machine_state>
  </deployment_details>
</deployment>
```

Interface Configurations

The Interface configuration allows to choose various configuration for the interface including network, subnet, ip address, mac address, vim interface name, model, and so on.

This section describes these basic and advance interface configurations for Elastic Services Controller (ESC) and procedures to configure these.

Basic Interface Configurations

In ESC Datamodel, Interface refers to the VNIC attached to the VM. We can add one or more Interface under a VM Group. The interface section will have details to configure the VNIC.

This section describes basic interface configurations for Elastic Services Controller (ESC).

Configuring Basic Interface Settings

This section describes basic interface configurations, such as:

- Network
- Subnet
- IP address
- MAC address
- VIM interface name, and so on for Elastic Services Controller (ESC).

Configuring an Interface Name

To configure VIM interface name, specify attribute `<vm_interface_name>` for an interface in the Deployment XML file. Use `<vm_interface_name>` to use a specific name when generating an interface name. If these attribute is not specified, ESC will auto-generate an interface name, which is a combination of the deployment_name, group_name, and a random UUID string. For example: my-deployment-na_my-gro_0_8053d7gf-hyt33-4676-h9d4-9j4a5599472t.

Note This feature is currently supported only on OpenStack.

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

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

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

Note You can use a maximum of 61 characters for an interface name should not contain special characters and can only contain alphanumeric characters and "_" and "-". The following are some output samples with the custom port name. If the `vim_interface_name` was set during the deployment, the same value will be shown in the output. If this value was not set during the deployment, ESC will auto-generate the port name.

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

```
[admin@esc-3-1-xxx]$ esc_nc_cli get esc_datamodel/opdata
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="1">
. . .
    <interface>
        <nicid>0</nicid>
```

```
        <type>virtual</type>
        <port_id>e4111069-5d00-493b-8ea9-1a2ca134b5c8</port_id>
        <vim_interface_name>interface_1</vim_interface_name>      <!-- NEW IN OUTPUT
-->
        <network>c7fafeca-aa53-4349-9b60-1f4b92605420</network>
        <subnet>255.255.255.0</subnet>
        <ip_address>192.168.2.1</ip_address>
        <mac_address>fa:16:3e:d7:5e:da</mac_address>
        <netmask>255.255.240.0</netmask>
        <gateway>192.168.2.255</gateway>
    </interface>
```

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

```
GET http://localhost:8080/ESCManager/v0/deployments/example-deployment-123
| xmllint --format -
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<deployments>
. . .
    <interface>
        <network_uuid>c7fafeca-aa53-4349-9b60-1f4b92605420</network_uuid>
        <gateway>172.16.0.1</gateway>
        <ip_address>172.16.12.251</ip_address>
        <mac_address>fa:16:3e:30:0c:99</mac_address>
        <netmask>255.255.240.0</netmask>
        <nic_id>0</nic_id>
        <port_forwarding/>
        <port_uuid>1773cdbf-fe5f-4af1-adff-3a9c1dd1c47d</port_uuid>
       <vim_interface_name>interface_1</vim_interface_name>        <!-- NEW IN OUTPUT
 -->
        <security_groups/>
        <subnet_uuid>7b2ce63b-eb20-4ff8-8d49-e46ee8dde0f5</subnet_uuid>
        <type>virtual</type>
    </interface>
```

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

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

Assigning the MAC Address

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

You can assign MAC address in the following ways:

Using the Interface

```
<interfaces>
    <interface>
 <nicid>1</nicid>
 <network>MANAGEMENT_NETWORK</network>
 <ip_address>172.16.0.11</ip_address>
 <mac_address>fa:16:3e:73:19:a0</mac_address>
    </interface>
</interfaces>
```

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

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

```
 <elastic>true</elastic>
 <static_ip_address_pool>
  <network>MANAGEMENT_NETWORK</network>
  <ip_address>172.16.0.11</ip_address>
  <ip_address>172.16.0.12</ip_address>
  <ip_address>172.16.0.13</ip_address>
 </static_ip_address_pool>
 <static_mac_address_pool>
  <network>MANAGEMENT_NETWORK</network>
  <mac_address>fa:16:3e:73:19:a0</mac_address>
  <mac_address>fa:16:3e:73:19:a1</mac_address>
  <mac_address>fa:16:3e:73:19:a2</mac_address>
 </static_mac_address_pool>
</scaling>
```

Assign MAC address using MAC address range.

```
<scaling>
 <min_active>2</min_active>
 <max_active>2</max_active>
 <elastic>true</elastic>
 <static_ip_address_pool>
  <network>MANAGEMENT_NETWORK</network>
  <ip_address_range>
   <start>172.16.0.25</start>
   <end>172.16.0.27</end>
  </ip_address_range>
 </static_ip_address_pool>
 <static_mac_address_pool>
  <network>MANAGEMENT_NETWORK</network>
  <mac_address_range>
   <start>fa:16:3e:73:19:b0</start>
   <end>fa:16:3e:73:19:b2</end>
  </mac_address_range>
 </static_mac_address_pool>
</scaling>
```

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

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

Configuring Subnet for an Interface

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

```
<interface>
   <nicid>0</nicid>
   <network>my-network</network>
  <subnet>my-subnet</subnet>
    </interface>
```

The `no_gateway` attribute allows ESC to create a subnet with the gateway disabled. In the example below, the `no_gateway` attribute is set to true to create a subnet without gateway.

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

Configuring an Out-of-Band Port

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

Note While undeploying or restoring a VNF, the ports attached to that VNF will only be detached and not deleted. ESC does not allow scaling while using OOB port for a VM group. You can configure only one instance of VM for the VM group.Updating the scaling value for a VM group, while using the out of band port is not allowd during a deployment update.

```
<esc_datamodel xmlns="http://www.cisco.com/esc/esc">
    <name>tenant</name>
    <deployments>
        <deployment>
            <name>depz</name>
            <vm_group>
                <name>g1</name>
                <image>Automation-Cirros-Image</image>
                <flavor>Automation-Cirros-Flavor</flavor>
                <bootup_time>100</bootup_time>
                <reboot_time>30</reboot_time>
                <recovery_wait_time>10</recovery_wait_time>
                <interfaces>
                    <interface>
                        <nicid>0</nicid>
                        <port>057a1c22-722e-44da-845b-a193e02807f7</port>
                        <network>my-network</network>
                    </interface>
                </interfaces>
            </vm_group>
        </deployment>
    </deployments>
</esc_datamodel>
```

Dual Stack Support

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

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

ESC supports the following for dual stack:

- Configuring the network and list of subnet

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

Note Currently, ESC supports dual stack only on OpenStack.

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

The container address is as follows:

```
container addresses {
  list address {
    key "address_id";
    leaf address_id {
      description "Id for the address in address list.";
      type uint16;
      mandatory true;
    }
    leaf subnet {
      description "Subnet name or uuid for allocating IP to this port";
      type types:escnetname;
    }
    leaf ip_address {
      description "Static IP address for this specific subnet";
      type types:escipaddr;
      must "../../../../../scaling/max_active = 1"
      {
         error-message "Only single VM per group supported with multiple address option.";
      }
    }
  }
}
```

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

```
…
    <interface>
        <nicid>1</nicid>
        <network>demo-net</network>
        <addresses>
            <address>
                <address_id>0</address_id>
                <subnet>demo-subnet</subnet>
            </address>
        </addresses>
    </interface>
 <kpi_data>
              <kpi>
                <event_name>VM_ALIVE</event_name>
                <metric_value>1</metric_value>
                <metric_cond>GT</metric_cond>
                <metric_type>UINT32</metric_type>
```

```
            <metric_occurrences_true>5</metric_occurrences_true>
                 <metric_occurrences_false>5</metric_occurrences_false>
                <metric_collector>
                    <type>ICMPPing</type>
                    <nicid>1</nicid>
                    <address_id>0</address_id>
                    <poll_frequency>10</poll_frequency>
                    <polling_unit>seconds</polling_unit>
                   <continuous_alarm>false</continuous_alarm>
                  </metric_collector>
                </kpi>
              </kpi_data>
...
```

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

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

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

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

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

NICID_n_a_IP_ALLOCATION_TYPE	string containing FIXED | DHCP	ipv4 or ipv6
NICID_n_a_IP_ADDRESS	IP address	ipv4 or ipv6
NICID_n_a_GATEWAY	Gateway address	ipv4 or ipv6
NICID_n_a_CIDR_ADDRESS	CIDR prefix address	ipv4 or ipv6
NICID_n_a_CIDR_PREFIX	Integer with CIDR prefix-length	ipv4 or ipv6
NICID_n_a_NETMASK	If an ipv4 CIDR address and prefix are present, ESC will automatically calculate and populate the netmask variable. This is not substituted in the case of an IPv6 address and should not be used.	ipv4 only

For information on day-0 confirguration for single IP address, see Day Zero Configuration, on page 3.

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

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

NICID_0_0_IP_ALLOCATION_TYPE=${NICID_0_0_IP_ALLOCATION_TYPE}
NICID_0_0_IP_ADDRESS=${NICID_0_0_IP_ADDRESS}
NICID_0_0_GATEWAY=${NICID_0_0_GATEWAY}
```

```
NICID_0_0_CIDR_ADDRESS=${NICID_0_0_CIDR_ADDRESS}
NICID_0_0_CIDR_PREFIX=${NICID_0_0_CIDR_PREFIX}
NICID_0_0_NETMASK=${NICID_0_0_NETMASK}

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

The datamodel is as follows:

```
<?xml version="1.0" encoding="ASCII"?>
<esc_datamodel xmlns="http://www.cisco.com/esc/esc">
  <tenants>
    <tenant>
      <name>dep-tenant</name>
      <deployments>
        <deployment>
          <name>cirros-dep</name>
          <vm_group>
            <name>Grp1</name>
            <bootup_time>600</bootup_time>
            <recovery_wait_time>30</recovery_wait_time>
            <flavor>Automation-Cirros-Flavor</flavor>
            <image>Automation-Cirros-Image</image>
           <interfaces>
              <interface>
                <!-- No dual stack support on mgmt interface in ESC 4.1 -->
                <nicid>0</nicid>
                <network>my-network</network>
              </interface>
              <interface>
                <nicid>1</nicid>
                <network>ent-network1</network>
                <addresses>
                  <address>
                    <!-- IPv4 Dynamic -->
                    <address_id>0</address_id>
                    <subnet>v4-subnet_A</subnet>
                  </address>
                  <address>
                    <!-- IPv6 Dynamic -->
                    <address_id>1</address_id>
                    <subnet>v6-subnet_B</subnet>
                  </address>
                </addresses>
              </interface>
              <interface>
                <nicid>2</nicid>
                <network>ent-network2</network>
                <addresses>
                  <address>
                    <!-- IPv4 Static -->
                    <address_id>0</address_id>
                    <subnet>v4-subnet_C</subnet>
                    <ip_address>172.16.87.8</ip_address>
                  </address>
                  <address>
                    <!-- IPv6 Static -->
                    <address_id>1</address_id>
                    <subnet>v6-subnet_D</subnet>
                    <ip_address>fd07::110</ip_address>
                  </address>
```

```
            </addresses>
          </interface>
          <interface>
            <nicid>3</nicid>
            <network>ent-network3</network>
            <addresses>
              <address>
                <!--  Only ip config -  ipv6 but no subnet  -->
                <address_id>0</address_id>
                <ip_address>fd07::110</ip_address>
              </address>
              <address>
                <!--  Only ip config - ipv4 but no subnet  -->
                <address_id>1</address_id>
                <ip_address>172.16.88.9</ip_address>
              </address>
            </addresses>
          </interface>
          <interface>
            <nicid>4</nicid>
            <network>ent-network4</network>
            <addresses>
              <address>
                <!-- ipv4 same subnet as address_id 6 -->
                <address_id>0</address_id>      //
                <subnet>v4-subnet_F</subnet>
                <ip_address>172.16.86.10</ip_address>
              </address>
              <address>
                <!-- ipv4 same subnet as id 5 -->
                <address_id>1</address_id>
                <subnet>v4-subnet_F</subnet>
                <ip_address>172.16.86.11</ip_address>
              </address>
            </addresses>
          </interface>
        </interfaces>
        <kpi_data>
...
```

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

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

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

Notification:

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

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

```
                    <hostname>my-server</hostname>
                    <interfaces>
                      <interface>
                        <nicid>0</nicid>
                        <type>virtual</type>

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

                        <port_id>4d57d4a5-3150-455a-ad39-c32fffbb10b1</port_id>
                        <mac_address>fa:16:3e:d2:50:a5</mac_address>
                        <network>45638651-2e92-45fb-96ce-9efdd9ea343e</network>
                        <address>
                          <address_id>0<address_id>
                          <subnet>6ac36430-4f58-454b-9dc1-82f7a796e2ff</subnet>
                          <ip_address>172.16.0.22</ip_address>
                          <prefix>24</prefix>
                          <gateway>172.16.0.1</gateway>
                        </address>
                        <address>
                          <address_id>1<address_id>
                          <subnet>8dd9f501-19d4-4782-8335-9aa9fbd4dab9</subnet>
                          <ip_address>2002:dc7::4</ip_address>
                          <prefix>48</prefix>
                          <gateway>2002:dc7::1</gateway>
                        </address>
                        <address>
                          <address_id>2<address_id>
                          <subnet>a234501-19d4-4782-8335-9aa9fbd4caf6</subnet>
                          <ip_address>172.16.87.8</ip_address>
                          <prefix>20</prefix>
                          <gateway>172.16.87.1</gateway>
                        </address>
                      </interface>
```

Sample opdata:

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

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

      <port_id>4d57d4a5-3150-455a-ad39-c32fffbb10b1</port_id>
      <mac_address>fa:16:3e:d2:50:a5</mac_address>
      <network>45638651-2e92-45fb-96ce-9efdd9ea343e</network>
      <address>
         <address_id>0</address_id>
         <subnet>6ac36430-4f58-454b-9dc1-82f7a796e2ff</subnet>
         <ip_address>172.16.0.22</ip_address>
         <prefix>24</prefix>
         <gateway>172.16.0.1</gateway>
      </address>
      <address>
         <address_id>1</address_id>
         <subnet>8dd9f501-19d4-4782-8335-9aa9fbd4dab9</subnet>
         <ip_address>2002:dc7::4</ip_address>
         <prefix>48</prefix>
         <gateway>2002:dc7::1</gateway>
      </address>
```

```
   </interface>
</interfaces>
```

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

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

NICID_0_0_IP_ALLOCATION_TYPE=DHCP
NICID_0_0_IP_ADDRESS=172.16.0.22
NICID_0_0_GATEWAY=172.16.0.1
NICID_0_0_CIDR_ADDRESS=172.16.0.0
NICID_0_0_CIDR_PREFIX=24
NICID_0_0_NETMASK=255.255.255.0

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

Dual Stack with Static IP Support

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

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

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

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

```
  <?xml version="1.0" encoding="ASCII"?>
<esc_datamodel xmlns="http://www.cisco.com/esc/esc">
  <tenants>
    <tenant>
      <name>dep-tenant</name>
      <deployments>
        <deployment>
          <name>cirros-dep</name>
          <vm_group>
            <name>Grp1</name>
            <bootup_time>600</bootup_time>
            <recovery_wait_time>30</recovery_wait_time>
            <flavor>Automation-Cirros-Flavor</flavor>
            <image>Automation-Cirros-Image</image>
            <interfaces>
              <interface>
                <nicid>0</nicid>
                <network>ent-network2</network>
                <addresses>
                  <address>
                    <!-- IPv4 Static -->
                    <address_id>0</address_id>
                    <subnet>v4-subnet_C</subnet>
                    <ip_address>172.16.87.8</ip_address>
                  </address>
                  <address>
```

```
                <!-- IPv6 Static -->
                <address_id>1</address_id>
                <subnet>v6-subnet_D</subnet>
                <ip_address>fd07::110</ip_address>
              </address>
            </addresses>
          </interface>
        </interfaces>
      <kpi_data>
```

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

Advanced Interface Configurations

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

For information on basic interface settings, see Basic Interface Configurations.

Configuring Advance Interface Settings

Configuring SR-IOV in ESC

Single Root I/O Virtualization (SR-IOV) allows multiple VMs running a variety of guest operating systems to share a single PCIe network adapter within a host server. It also allows a VM to move data directly to and from the network adapter, bypassing the hypervisor for increased network throughput and lower server CPU burden.

Configuring SR-IOV in ESC for OpenStack

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

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

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

Configuring SR-IOV in ESC for VMware

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

- Enable SR-IOV Physical Functions on desired ESXi hosts. For more information, see VMware documentation.
- Consider the following important points before enabling SR-IOV:
  - Review the list of physical network adaptors that VMware supports for SR-IOV. See VMware documentation.
  - Review the list of VM features that are not supported on a VM with SR-IOV configured. See VMware documentation.

- In a cluster deployment (defined by "zone" in the datamodel) with SR-IOV, make sure that each ESXi host has identical Physical Functions enabled for SR-IOV selection. For example, if a VM is going to use vmnic7 as the Physical Function, make sure that each host has vmnic7 and SR-IOV status for each vmnic7 is enabled.

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

```
<vm_group>
...
<interface>
  <nicid>2</nicid>
  <network>MgtNetwork</network>
  <type>direct</type>
</interface>
<interface>
  <nicid>3</nicid>
  <network>MgtNetwork</network>
  <type>direct</type>
</interface>
...
<extensions>
 <extension>
 <name>sriov_pf_selection</name>
 <properties>
 <property>
 <name>nicid-2</name>
 <value>vmnic1,vmnic2</value>
 </property>
 <property>
 <name>nicid-3</name>
 <value>vmnic3,vmnic4</value>
 </property>
 </properties>
 </extension>
 </extensions>
</vm_group>
```

Configuring Allowed Address Pair

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

The address pair is configured in the following ways:

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

```
<interface>
  <nicid>1</nicid>
  <network>network1</network>
  <allowed_address_pairs>
  <network>
  <name>bb8c5cfb-921c-46ea-a95d-59feda61cac1</name>
  </network>
  <network>
```

```
  <name>6ae017d0-50c3-4225-be10-30e4e5c5e8e3</name>
  </network>
  </allowed_address_pairs>
  </interface>
</interfaces>
```

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

```
<interface>
  <nicid>0</nicid>
  <network>my-network</network>
  <allowed_address_pairs>
  <address>
  <ip_address>172.16.10.10</ip_address>
  <netmask>255.255.255.0</netmask>
  </address>
  <address>
  <ip_address>172.16.20.10</ip_address>
  <netmask>255.255.255.0</netmask>
  </address>
  </allowed_address_pairs>
</interface>
```

Configuring Port Security

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

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

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

Configuring Security Group Rules

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

To configure security group rules, do the following:

Before you begin

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

Procedure

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

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

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

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

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

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

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

Hardware Acceleration Support (OpenStack Only)

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

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

- **OpenStack support for PCIe Passthrough interface**—enables assigning a PCI device to an instance on OpenStack.

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

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

Configuring PCI or PCIe Device Passthrough on VMware vCenter

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

Before You Begin

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

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

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

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

     <vm_group>
      <name>test-g1</name>
      <image>uLinux</image>
      <bootup_time>300</bootup_time>
      <recovery_wait_time>10</recovery_wait_time>
      <interfaces>
       <interface>
        <nicid>1</nicid>
        <network>MgtNetwork</network>
```

```
 <ip_address>192.168.0.102</ip_address>
</interface>
<interface>
 <nicid>2</nicid>
 <network>VM Network</network>
 <type>passthru</type>
 <ip_address>172.16.0.0</ip_address>
</interface>
<interface>
 <nicid>3</nicid>
 <network>VM Network</network>
 <type>passthru</type>
 <ip_address>192.168.46.117</ip_address>
</interface>
</interfaces>
```

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

Auto Selecting PCI or PCIe PassThrough Device

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

Note

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