ERICSSON

OpenStack Orchestration API in CEE

Cloud Execution Environment

INTERWORK DESCRIPTION

1/155 19-CRA 119 1873/5 Uen A

Contents

1 Introduction

This document serves as an introduction to the Application Programming Interface (API) of the OpenStack Dashboard based cloud management component Atlas, in the Cloud Execution Environment (CEE).

The API is used for orchestration purposes in CEE.

The document describes the modifications made to the OpenStack version.

1.1 API Version

The Atlas orchestration API is based on OpenStack Orchestration API v1.

1.2 Document References

This section lists the official OpenStack API references.

1.2.1 API Design Base Reference

For a detailed description of the API operations and extensions of orchestration, refer to the section “Orchestration API v1” in the *OpenStack API Complete Reference*.

This is a stored copy of the OpenStack API reference document that was the base for the development of this version of CEE.

All references to this OpenStack API made in this document are based on this specific document version. The document may include API Extensions that are not supported in this version of CEE. For details, see Section 2.2 on page 2.

Note: It is possible that the date on the front page differs from the one above, since the former shows the date on which the document was generated.

1.2.2 API Template Reference

For a detailed description of the Heat template, refer to *OpenStack Heat*.

This is a stored copy of the OpenStack Heat template reference document that was the base for the development of this version of CEE.

Note: It is possible that the date on the front page differs from the one above, since the former shows the date on which the document was generated.

2 Supported Operations

This section contain information about the API operations and API extensions in CEE.

2.1 Basic OpenStack Operations

Atlas orchestration supports all basic Orchestration API operations.

For a detailed description of the API operations and extensions of orchestration, refer to the section ‘‘Orchestration API v1’’ in the *OpenStack API Complete Reference*.

2.1.1 Limitations

Not applicable.

2.1.2 OpenStack Template Reference

CEE Orchestration supports all Template Reference.

For a detailed description of the Heat template, refer to *OpenStack Heat*.

2.2 OpenStack Extensions

This section lists the OpenStack API extensions that are used in CEE.

CEE orchestration supports the Identity API v2.0 extensions documented in the section ‘‘Identity API v2.0 extensions’’ in the *OpenStack API Complete Reference*.

3 Heat Plug-ins

The required properties and resource types related to Heat are implemented as a plug-ins. The properties required for Ericsson-specific tags are added in the plug-ins.

The plug-ins are implemented as Python modules and are described in Section 3.1 on page 3, Section 3.2 on page 4,and Section 3.3 on page 5.

Installation and Configuration

The file `/etc/heat/heat.conf` contains a list of directories where the plug-ins are stored. To define additional directories, use the `plugin_dirs` option in file `/etc/heat/heat.conf`.

By default the `/usr/lib/python2.7/dist-packages/heat/plugins` directory is included and searched for plug-ins.

To install a plug-in, copy the Python modules to one of the configured plug-in directories.

Note: The Heat engine must be restarted after this in order to load the new plug-ins.

3.1 Ericsson::Nova::Flavor

The plug-in enables the use of `Nova Flavor`, as a resource in a Heat template. The Python module for the plug-in is `ericsson_flavor.py`.

The new resource has the type `Ericsson::Nova::Flavor` with the properties shown in Table 1.

Table 1 Nova Flavor Properties

Name	Type	Description
`ram`	Integer	Memory in MB for the flavor
`vcpus`	Integer	Number of VCPUs for the flavor
`disk`	Integer	Size of local disk in GB
`swap`	Integer	Swap space in MB
`ephemeral`	Integer	Size of a secondary ephemeral data disk
`rxtx_factor`	Number	Receiver Mode (RX)/Transmitter Mode (TX) factor

Name	Type	Description
extra_specs	Map	Key/Value pairs to extend the capabilities of the flavor
is_public	Boolean	Scope of flavor accessibility (can be public or private). Default value is True, meaning flavor accessibility is public, shared across all projects.

An Ericsson::Nova::Flavor resource uses the properties in Table 1 to dynamically create a flavor from the values, as shown in Example 1.

```yaml
heat_template_version: 2013-05-23
description:  Heat Flavor creation example
resources:
  test_flavor:
    type: Ericsson::Nova::Flavor
    properties:
      ram: 1024
      vcpus: 1
      disk: 20
      swap: 2
      extra_specs: {"quota:disk_read_bytes_sec": "10240000"}
```

Example 1 Nova Flavor Template

Note: Ericsson::Nova::Flavor is deprecated. Use OS::Nova::Flavor instead.

3.2 Ericsson::Heat::DelayPolicy

The plug-in enables the use of a Delay Policy, as a resource in a Heat template. The Python module for the plug-in is delay.py.

The resource enables delay in seconds between allocation of two consequent resources. A typical usage of this resource is waiting for one VM to finished configuring itself before invoking deployment of another one.

The new resource has the type Ericsson::Heat::DelayPolicy with the properties shown in Table 2.

Table 2 Delay Policy Properties

Name	Type	Description
delay	Number	Delay between two resources

An Ericsson::Heat::DelayPolicy resource uses the properties in Table 2 to dynamically create a delay between two resources, as shown in Example 2.

```yaml
heat_template_version: 2013-05-23
description:  Heat DelayPolicy creation example
resources:
  test_delay:
    type: Ericsson::DelayPolicy
    properties: {delay: 2}
```

Example 2 Delay Policy Template

3.3 Ericsson::Neutron::Port

The plug-in enables the use of `port`, as a resource in a Heat template. The Python module for the plug-in is `eport.py`.

The new resource has the type `Ericsson::Neutron::Port` with the properties shown in Table 3.

Table 3 Ericsson::Neutron::Port Properties

Name	Type	Description
`network_id`	String	Unique identifier for the network owning the port
`Network`	String	The network to which this port belongs
`Name`	String	A symbolic name for this port
`Value_specs`	Map	Extra parameters to include in the port object
`admin_state_up`	String	The administrative state of this port
`fixed_ips`	List	Desired IPs for this port
`mac_address`	String	MAC address for this port
`device_id`	String	Device ID of this port
`security_groups`	List	Security group IDs to associate with this port
`allowed_address_pairs`	List	Additional MAC/IP address pairs allowed to pass through the port
`device_owner`	String	Name of the network owning the port
`replacement_policy`	String	Policy on how to respond to a stack-update for this resource. **`REPLACE_ALWAYS`** will replace the port regardless of any property changes. **`AUTO`** will update the existing port for any changed update-allowed property.

Name	Type	Description
`Trunkport`	MAP	Trunk port properties
`1) trunk_port_type`	String	Defines if a port is a trunk port or sub port
`2) trunk_port_parent_id`	String	For sub ports the ID of the trunk port that sub port is connected to
`3) vid`	String	For sub ports VID that will be used to access this sub port from trunk port
`Binding`	MAP	Binding port properties
`1) host_id`	String	Defines host ID of a port
`2) vnic_type`	String	The vNIC type needs to be bound on the Neutron port. To support SR-IOV PCI passthrough networking request the neutron port to be realized as `normal` (vNIC), `direct` (PCI passthrough), or `macvtap` (a virtual interface with a tap-like software interface). This procedure only works for Neutron deployments that support the `bindings` extension.
`qos_policy`	String	The name or ID of QoS policy to be attached to this port
`port_security_enabled`	Boolean	Flag to enable or disable port security on this port. When disabled (set to `False`), there is no package filtering (for example `security_groups` or `address-pairs`).

An `Ericsson::Neutron::Port` resource uses the properties in Table 3 to dynamically create a port, as shown in Example 3.

```yaml
heat_template_version: 2013-05-23
description:  Heat Port creation example
resources:
  test_eport:
    type: Ericsson::Neutron::Port
    properties:
      network_id: net1234
      trunkport: {
        type: subport,
        parent_id: s0e951a6-h6b4-3f4c-8c88-b95e2a120b7m',
        vid: 101
                 }
```

Example 3 HOT Syntax for Ericsson::Neutron::Port Resource

3.4 Ericsson::Neutron::RouterInterface

This is a resource representation of an internal network interface to the specified router.

The new resource has the type `Ericsson::Neutron::RouterInterface` with the properties shown in Table 4.

Table 4 RouterInterface Properties

Name	Type	Description
`router`	String	Name of the router.
`router_id`	String	ID of the router.
`port`	String	Port name: either the subnet or the port should be specified.
`port_id`	String	ID of the port.
`subnet`	String	Subnet name: either the subnet or the port should be specified.
`subnet_id`	String	ID of the subnet.
`active_device_id`	String	The active device ID. Updates cause replacement. Optional property.

An `Ericsson::Neutron::RouterInterface` resource uses the properties in Table 4, as shown in Example 4, using HOT syntax.

```
heat_template_version: 2013-05-23
...
resources:
  ...
  the_resource:
    type: OS::Neutron::RouterInterface
    properties:
      port: String
      router: String
      subnet: String
      active_device_id: String
```

Example 4 HOT Syntax for Router Interface Resource

4 Limitations and Recommendations

This section describes CEE specific limitations and recommendations.

4.1 Limitations

Not applicable.

4.2 Recommendations

Not applicable.