VNF
Heat Template Requirements
for
OpenECOMP

Revision	1.0
Revision Date	2/1/2017

Document Revision History

Date	Revision	Description
2/1/2017	1.0	Initial publication of VNF Heat Template Requirements for OpenECOMP

Table of Contents

Definitions

Throughout the document, these terms have the following meaning:

MUST This word, or the terms "REQUIRED" or "SHALL", mean that the definition is an absolute requirement of the specification.

MUST NOT This phrase, or the phrase "SHALL NOT", mean that the definition is an absolute prohibition of the specification.

SHOULD This word, or the adjective "RECOMMENDED", mean that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.

SHOULD NOT This phrase, or the phrase "NOT RECOMMENDED" mean that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.

MAY This word, or the adjective "OPTIONAL", mean that an item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because the vendor feels that it enhances the product while another vendor may omit the same item. An implementation which does not include a particular option must be prepared to interoperate with another implementation which does include the option, though perhaps with reduced functionality. In the same vein an implementation which does include a particular option must be prepared to interoperate with another implementation which does not include the option (except, of course, for the feature the option provides.)

1. Introduction

This reference document is the ***VNF Heat Template Requirements for OpenECOMP*** and supports the first release of OpenECOMP.

1.1 Program and Document Structure

This document is part of a hierarchy of documents that describes the overall Requirements and Guidelines for OpenECOMP. The diagram below identifies where this document fits in the hierarchy.

<table>
<tr><td colspan="5">OpenECOMP Requirements and Guidelines</td></tr>
<tr><td colspan="4">VNF Guidelines for Network Cloud and OpenECOMP</td><td>Future OpenECOMP Subject Documents</td></tr>
<tr><td>VNF Cloud Readiness Requirements for OpenECOMP</td><td>VNF Management Requirements for OpenECOMP</td><td>VNF Heat Template Requirements for OpenECOMP</td><td>Future VNF Requirements Documents</td><td>Future Requirements Documents</td></tr>
</table>

Document Summary

VNF Guidelines for Network Cloud and OpenECOMP

- Describes VNF environment and overview of requirements

VNF Cloud Readiness Requirements for OpenECOMP

- Cloud readiness requirements for VNFs (Design, Resiliency, Security, and DevOps)

VNF Management Requirements for OpenECOMP

- Requirements for how VNFs interact and utilize OpenECOMP

VNF Heat Template Requirements for OpenECOMP

- Provides recommendations and standards for building Heat templates compatible with OpenECOMP– initial implementations of Network Cloud are assumed to be OpenStack based.

1.2 Intended Audience

This document is intended for persons developing Heat templates that will be orchestrated by OpenECOMP.

1.3 Scope

The first implementations of Network Cloud are assumed to be OpenStack based and thus OpenECOMP will be supporting Heat Orchestration Templates, also referred to as Heat templates or Heat in this document.

OpenECOMP requires the Heat Templates to follow a specific format. This document provides the mandatory, recommended, and optional requirements associated with this format.

In addition, the OpenStack version deployed in the Network Cloud may impose additional constraints on the Heat. These constraints are not covered in this document.

1.4 VNF Modularity Overview

OpenECOMP supports a modular Heat design pattern, referred to as *VNF Modularity.* With this approach, a single VNF may be composed from one or more Heat templates, each of which represents some subset of the overall VNF. These component parts are referred to as "*VNF Modules*". During orchestration, these modules may be deployed incrementally to build up the complete VNF.

A Heat template can be either one of the following types of modules:

1. Base Module
2. Incremental Modules
3. Independent Cinder Volume Modules

The OpenECOMP Heat template naming convention must be followed (Section 2.1). The naming convention identifies the module type.

A VNF must be composed of one "base" VNF module (also called a base module) and zero to many "incremental" or "add on" VNF modules. The base module must be deployed first, prior to the add-on modules.

A module can be thought of as equivalent to a Heat template, where a Heat template is composed of a YAML file and an environment file (also referred to as an ENV file). A given YAML file must have a corresponding environment file; OpenECOMP requires it.

A Heat template is used to create or deploy a Heat stack. Therefore, a module is also equivalent to a Heat Stack.

OpenECOMP supports the concept of an optional, independent deployment of a Cinder volume via separate Heat templates. This allows the volume to persist after VNF deletion so that the volume can be reused on another instance (e.g. during a failover activity).

The scope of a volume module, when it exists, must be 1:1 with the VNF Module (base or add-on). A single volume module must create only the volumes needed by a single VNF module (base or add-on).

These concepts will be described in more detail throughout the document. This overview is provided to set the stage and help clarify the concepts that will be introduced.

2. General Guidelines

The Heat templates supported by OpenECOMP must follow the requirements enumerated in this section.

2.1 Filenames

In order to enable OpenECOMP to understand the relationship between Heat files, the following Heat file naming convention must be followed.

- The file name for the base module Heat template must include "base" in the filename.
    - Examples: *base_xyz.yml* or *base_xyz.yaml*; *xyz_base.yml* or *xyz_base.yaml*

- There is no explicit naming convention for the add-on modules.
    - Examples: *module1.yml* or *module1.yaml*
- All Cinder volume templates must be named the same as the corresponding Heat template with "_volume" appended to the file name.
    - Examples: *base_xyz_volume.yml* or *base_xyz_volume.yaml*; *xyz_base_volume.yml* or *xyz_base_volume.yaml*; *module1_volume.yml* or *module1_volume.yaml* (referencing the above base module Heat template name)
- The file name of the environment files must fully match the corresponding Heat template filename and have *.env* or *.ENV* extension.
    - Examples: *base_xyz.env* or *base_xyz.ENV*; *xyz_base.env* or *xyz_base.ENV*; *base_xyz_volume.env* or *base_xyz_volume.ENV*; *module1.env* or *module1.ENV; module1_volume.env* or *module1_volume.ENV* (referencing the above base module Heat template name)
- A YAML file must have a corresponding ENV file, even if the ENV file enumerates no parameters. It is an OpenECOMP requirement.

2.2 Valid YAML Format

A Heat template (a YAML file and its corresponding environment file) must be formatted in valid YAML. For a description of YAML, refer to the following OpenStack wiki:
https://wiki.openstack.org/wiki/Heat/YAMLTemplates

A Heat template must follow a specific format. The OpenStack Heat Orchestration Template (HOT) specification explains in detail all elements of the HOT template format.
http://docs.openstack.org/developer/heat/template_guide/hot_spec.html

2.3 Parameter Categories & Specification

2.3.1 Parameter Categories

OpenECOMP requires the Heat template parameters to follow certain requirements in order for it to be orchestrated or deployed. OpenECOMP classifies parameters into eight broad categories.

- **OpenECOMP Metadata**: OpenECOMP mandatory and optional metadata parameters in the resource *OS::Nova::Server*.
    - OpenECOMP dictates the naming convention of these Metadata parameters and must be adhered to (See Section 4.4).
    - Metadata parameters must not be enumerated in the environment file.
    - The OpenECOMP Metadata are generated and/or assigned by OpenECOMP and supplied to the Heat by OpenECOMP at orchestration time.
- **OpenECOMP Orchestration Parameters**: The data associated with these parameters are VNF instance specific.
    - OpenECOMP enforces the naming convention of these parameters and must be adhered to (See Section 4).

    - These parameters must not be enumerated in the environment file.
    - The OpenECOMP Orchestration Parameters are generated and/or assigned by OpenECOMP and supplied to the Heat by OpenECOMP at orchestration time.

- **VNF Orchestration Parameters**: The data associated with these parameters are VNF instance specific.

    - While OpenECOMP does not enforce a naming convention, the parameter names should include {vm-type} and {network-role} when appropriate. (See Section 4)
    - These parameters must not be enumerated in the environment file.
    - The VNF Orchestration Parameters Heat are generated and/or assigned by OpenECOMP and supplied to the Heat by OpenECOMP at orchestration time.

- **OpenECOMP Orchestration Constants**: The data associated with these parameters must be constant across all VNF instances.

    - OpenECOMP enforces the naming convention of these parameters and must be adhered to (See Section 4).
    - These parameters must be enumerated in the environment file.

- **VNF Orchestration Constants**: The data associated with these parameters must be constant across all VNF instances.

    - While OpenECOMP does not enforce a naming convention, the parameter names should include {vm-type} and {network-role} when appropriate. (See Section 4)
    - These parameters must be enumerated in the environment file.

- **OpenECOMP Base Template Output Parameters** (also referred to as Base Template Output Parameters): The output section of the base template allows for specifying output parameters available to add-on modules once the base template has been instantiated. The parameter defined in the output section of the base must be identical to the parameter defined in the add-on module(s) where the parameter is used.

- **OpenECOMP Volume Template Output Parameters** (also referred to as Volume Template Output Parameters): The output section of the volume template allows for specifying output parameters available to the corresponding Heat template (base or add-on) once the volume template has been instantiated. The parameter defined in the output section of the volume must be identical to the parameter defined in the base or add-on module.

- **OpenECOMP Predefined Output Parameters** (also referred to as Predefined Output Parameters): OpenECOMP will look for a small set of pre-defined Heat output parameters to capture resource attributes for inventory in OpenECOMP. These parameters are specified in Section 4.6.

The table below summarizes the Parameter Types. If the user is orchestrating a manual spin up of Heat (e.g. OpenStack command line), the parameter values that OpenECOMP supplies must be enumerated in the environment file. However, when the Heat is to be loaded into OpenECOMP for orchestration, the

parameters that OpenECOMP supplies must be deleted or marked with a comment (i.e., a "#" placed at the beginning of a line).

Parameter Type	Naming Convention	Parameter Value Source
OpenECOMP Metadata	Explicit	OpenECOMP
OpenECOMP Orchestration Parameters	Explicit	OpenECOMP
VNF Orchestration Parameters	Recommended	OpenECOMP
OpenECOMP Orchestration Constants	Explicit	Environment File
VNF Orchestration Constants	Recommended	Environment File
OpenECOMP Base Template Output Parameters	Recommended	Heat Output Statement for base, OpenECOMP supplied to add-on modules
OpenECOMP Volume Template Output Parameters	Recommended	Heat Output Statement for volume, OpeneECOMP supplies to corresponding module
OpenECOMP Predefined Output Parameters	Explicit	Heat Output Statement

Table 1 Parameter Types

2.3.2 Parameter Specifications

2.3.2.1 OpenECOMP METADATA Parameters

OpenECOMP defines four "metadata" parameters: vnf_id, vf_module_id, vnf_name, vf_module_name. These parameters must not define any constraints in the Heat template, including length restrictions, ranges, default value and/or allowed patterns.

2.3.2.2 OpenECOMP Base Template & Volume Template Output Parameters

The base template and volume template output parameters are defined as input parameters in subsequent modules. When defined as input parameters, these parameters must not define any constraints in the Heat template, including length restrictions, ranges, default value and/or allowed patterns. The parameter name defined in the output statement of the Heat must be identical to the parameter name defined in the Heat that is to receive the value.

2.3.2.3 OpenECOMP Predefined Output Parameters

These parameters must not define any constraints in the Heat template, including length restrictions, ranges, default value and/or allowed patterns.

2.3.2.4 OpenECOMP Orchestration Parameters, VNF Orchestration Parameters, OpenECOMP Orchestration Constants, VNF Orchestration Constants

OpenECOMP Orchestration Parameters, VNF Orchestration Parameters, OpenECOMP Orchestration Constants, VNF Orchestration Constants must adhere to the following:

- All parameters should be clearly documented in the template, including expected values.
- All parameters should be clearly specified, including constraints and description.
- Numeric parameter constraints should include range and/or allowed values.

- When the parameter type is a string and the parameter name contains an index, the index must be zero based. That is, the index starts at zero.
- When the parameter type is a Comma Delimited List (CDL), the reference index must start at zero.
- Default values must only be supplied in a Heat environment file to keep the template itself as clean as possible.
- Special characters must not be used in parameter names, as currently only alphanumeric characters and "_" underscores are allowed.

2.4 Use of Heat Environments

A YAML file must have a corresponding environment file (also referred to as ENV file), even if the environment file defines no parameters. It is an OpenECOMP requirement.

The environment file must contain parameter values for the OpenECOMP Orchestration Constants and VNF Orchestration Constants. These parameters are identical across all instances of a VNF type, and expected to change infrequently. The OpenECOMP Orchestration Constants are associated with OS::Nova::Server image and flavor properties (See Section 4.3). Examples of VNF Orchestration Constants are the networking parameters associated with an internal network (e.g. private IP ranges) and Cinder volume sizes.

The environment file must not contain parameter values for parameters that are instance specific (OpenECOMP Orchestration Parameters, VNF Orchestration Parameters). These parameters are supplied to the Heat by OpenECOMP at orchestration time. The parameters are generated and/or assigned by OpenECOMP at orchestration time

2.5 Independent Volume Templates

OpenECOMP supports independent deployment of a Cinder volume via separate Heat templates. This allows the volume to persist after VNF deletion so that they can be reused on another instance (e.g. during a failover activity).

A VNF Incremental Module or Base Module may have an independent volume module. Use of separate volume modules is optional. A Cinder volume may be embedded within the Incremental or Base Module if persistence is not required.

If a VNF Incremental Module or Base Module has an independent volume module, the scope of volume templates must be 1:1 with Incremental module or Base module. A single volume module must create only the volumes required by a single Incremental module or Base module.

The following rules apply to independent volume Heat templates:

- Cinder volumes must be created in a separate Heat template from the Incremental and Base Modules.
    - A single volume module must include all Cinder volumes needed by the Incremental/Base module.
    - The volume template must define "outputs" for each Cinder volume resource universally unique identifier (UUID) (i.e. OpenECOMP Volume Template Output Parameters).

- The VNF Incremental Module or Base Module must define input parameters that match each Volume output parameter (i.e., OpenECOMP Volume Template Output Parameters).
    - OpenECOMP will supply the volume template outputs automatically to the bases/incremental template input parameters.
- Volume modules may utilize nested Heat templates.

Example (volume template):

```
 In this example, the {vm-type} has been left as a variable.  {vm-type} is described
 in section 4.1.  If the VM was a load balancer, the {vm-type} could be defined as
 “lb”

parameters:
   vm-typevnf_name:
      type: string
   {vm-type}_volume_size_0:
      type: number
   ...

resources:
   {vm-type}_volume_0:
      type: OS::Cinder::Volume
      properties:
         name:
            str_replace:
              template: VNF_NAME_volume_0
              params:
                VNF_NAME: { get_param: vnf_name }
         size: {get_param: {vm-type}_volume_size_0}
   ...
```
(+ additional volume definitions)

```
outputs:
    {vm-type}_volume_id_0:
       value: {get_resource: {vm-type}_volume_0}
   ...
```
(+ additional volume outputs)

Example (VNF module template):

```
parameters:
   {vm-type}_name_0:
      type: string
   {vm-type}_volume_id_0:
      type: string
   ...

resources:
  {vm-type}_0:
     type: OS::Nova::Server
     properties:
        name: {get_param: {vm-type}_name_0}
        networks:
        ...
  {vm-type}_0_volume_attach:
     type: OS::Cinder::VolumeAttachment
     properties:
        instance_uuid: { get_resource: {vm-type}_0 }
        volume_id: { get_param: {vm-type}_volume_id_0 }
```

2.6 Nested Heat Templates

OpenECOMP supports nested Heat templates per the OpenStack specifications. Nested templates may be suitable for larger VNFs that contain many repeated instances of the same VM type(s). A common usage pattern is to create a nested template for each VM type along with its supporting resources. The master VNF template (or VNF Module template) may then reference these component templates either statically (by repeated definition) or dynamically (via *OS::Heat::ResourceGroup*).

Nested template support in OpenECOMP is subject to the following limitations:

- Heat templates for OpenECOMP must only have one level of nesting. OpenECOMP only supports one level of nesting.
- Nested templates must be referenced by file name in the master template
    - i.e. use of *resource_registry* in the .env file is *not* currently supported
- Nested templates must have unique file names within the scope of the VNF
- OpenECOMP does not support a directory hierarchy for nested templates. All templates must be in a single, flat directory (per VNF)
- A nested template may be shared by all Modules (i.e., Heat templates) within a given VNF

3. Networking

3.1 External Networks

VNF templates must not include any resources for external networks connected to the VNF. In this context, “external” is in relation to the VNF itself (not with regard to the Network Cloud site). External networks may also be referred to as “inter-VNF” networks.

- External networks must be orchestrated separately, so they can be shared by multiple VNFs and managed independently. When the external network is created, it must be assigned a unique {network-role} (See section 4.2).
- External networks must be passed into the VNF template as parameters, including the network-id (i.e. the neutron network UUID) and optional subnet ID.
- VNF templates must pass the appropriate external network IDs into nested VM templates when nested Heat is used.
- VNFs may use DHCP assigned IP addresses or assign fixed IPs when attaching VMs to an external network.
- OpenECOMP enforces a naming convention for parameters associated with external networks.
- Parameter values associated with an external network will be generated and/or assigned by OpenECOMP at orchestration time.
- Parameter values associated with an external network must not be enumerated in the environment file.

3.2 Internal Networks

Orchestration activities related to internal networks must be included in VNF templates. In this context, "internal" is in relation to the VNF itself (not in relation to the Network Cloud site). Internal networks may also be referred to as "intra-VNF" networks or "private" networks.

- Internal networks must not attach to any external gateways and/or routers. Internal networks are for intra-VM communication only.
- In the modular approach, internal networks must be created in the Base Module template, with their resource IDs exposed as outputs (i.e., OpenECOMP Base Template Output Parameters) for use by all add-on module templates. When the external network is created, it must be assigned a unique {network-role} (See section 4.2).
- VNFs may use DHCP assigned IP addresses or assign fixed IPs when attaching VMs to an internal network.
- OpenECOMP does not enforce a naming convention for parameters for internal network, however, a naming convention is provided that should be followed.
- Parameter values associated with an internal network must either be passed as output parameter from the base template (i.e., OpenECOMP Base Template Output Parameters) into the add-on modules or be enumerated in the environment file.

3.3 IP Address Assignment

- VMs connect to external networks using either fixed (e.g. statically assigned) IP addresses or DHCP assigned IP addresses.
- VMs connect to internal networks using either fixed (e.g. statically assigned) IP addresses or DHCP assigned IP addresses.
- Neutron Floating IPs must not be used. OpenECOMP does not support Neutron Floating IPs.

- OpenECOMP supports the OS::Neutron::Port property “allowed_address_pairs.” See Section 4.4.3.

4. Parameter Naming Convention

4.1 {vm-type}

A common *{vm-type}* identifier must be used throughout the Heat template in naming parameters, for each VM type in the VNF with the following exceptions:

- The four OpenECOMP Metadata parameters must not be prefixed with a common {vm-type} identifier. They are *vnf_name*, *vnf_id*, *vf_module_id*, *vf_module_name*.
- Parameters only referring to a network or subnetwork must not be prefixed with a common {vm-type} identifier.
- The parameter referring to the OS::Nova::Server property availability_zone must not be prefixed with a common {vm-type} identifier.
- {vm-type} must be unique to the VNF. It does not have to be globally unique across all VNFs that OpenECOMP supports.

4.2 {network-role}

VNF templates must not include any resources for external networks connected to the VNF. In this context, “external” is in relation to the VNF itself (not with regard to the Network Cloud site). External networks may also be referred to as “inter-VNF” networks.

External networks must be orchestrated separately, so they can be shared by multiple VNFs and managed independently. When the external network is created, it must be assigned a unique {network-role}.

“External” networks must be passed into the VNF template as parameters. Examples include the network-id (i.e. the neutron network UUID) and optional subnet ID. See section 4.4.3.

Any parameter that is associated with an external network must include the {network-role} as part of the parameter name.

Internal network parameters must also define a {network-role}. Any parameter that is associated with an internal network must include int_{network-role} as part of the parameter name.

4.3 Resource: OS::Nova::Server - Parameters

The following OS::Nova::Server Resource Property Parameter Names must follow the OpenECOMP parameter Naming Convention. All the parameters associated with OS::Nova::Server are classified as OpenECOMP Orchestration Parameters.

OS::Nova::Server		
Property	OpenECOMP Parameter Naming Convention	Parameter Type
image	{*vm-type*}_image_name	string

flavor	{*vm-type*}_flavor_name	string
name	{*vm-type*}_name_{*index*}	string
	{vm-type}_names	CDL
availability_zone	availability_zone_{index}	string

Table 2 Resource Property Parameter Names

4.3.1 Property: image

Image is an OpenECOMP Orchestration Constant parameter. The image must be referenced by the Network Cloud Service Provider (NCSP) image name, with the parameter enumerated in the Heat environment file.

The parameters must be named *"{vm-type}_image_name"* in the VNF.

Each VM type (e.g., {vm-type}) should have a separate parameter for images, even if several share the same image. This provides maximum clarity and flexibility.

4.3.2 Property: flavor

Flavor is an OpenECOMP Orchestration Constant parameter. The flavors must be referenced by the Network Cloud Service Provider (NCSP) flavor name, with the parameter enumerated in the Heat environment file.

The parameters must be named *"{vm-type}_flavor_name"* for each *{vm-type}* in the VNF.

Each VM type should have separate parameters for flavors, even if more than one VM shares the same flavor. This provides maximum clarity and flexibility.

4.3.3 Property: Name

Name is an OpenEOMP Orchestration parameter; the value is provided to the Heat template by OpenECOMP.

VM names (hostnames) for assignment to VM instances must be passed to Heat templates either as

- an array (comma delimited list) for each VM type
- a set of fixed-index parameters for each VM type instance.

Each element in the VM Name list should be assigned to successive instances of that VM type.

The parameter names must reflect the VM Type (i.e., include the {vm-type} in the parameter name.) The parameter name format must be one of the following:

- If the parameter type is a comma delimited list: {*vm-type*}_names
- If the parameter type is a string with a fixed index: {*vm-type*}_name_{*index*}

If a VNF contains more than three instances of a given {vm-type}, the CDL form of the parameter name (i.e., *{vm-type}*_names} should be used to minimize the number of unique parameters defined in the Heat.

Examples:

```
parameters:
       {vm-type}_names:
              type: comma_delimited_list
              description: VM Names for {vm-type} VMs

       {vm-type}_name_{index}:
              type: string
              description: VM Name for {vm-type} VM {index}
```

Example (CDL):

```
In this example, the {vm-type} has been defined as “lb” for load balancer.

parameters:
       lb_names:
              type: comma_delimited_list
              description: VM Names for lb VMs

resources:
       lb_0:
              type: OS::Nova::Server
              properties:
                     name: { get_param: [lb_names, 0] }
                     ...

       lb_1:
              type: OS::Nova::Server
              properties:
                     name: { get_param: [lb_names, 1] }
                     ...
```

Example (fixed-index):

```
In this example, the {vm-type} has been defined as “lb” for load balancer.

parameters:
       lb_name_0:
              type: string
              description: VM Name for lb VM 0

       lb_name_1:
              type: string
              description: VM Name for lb VM 1

resources:
       lb_0:
              type: OS::Nova::Server
              properties:
                     name: { get_param: lb_name_0 }
                     ...

       lb_1:
              type: OS::Nova::Server
              properties:
                     name: { get_param: lb_name_1 }
                     ...
```

4.3.4 Property: availability_zone

Availability_zone is an OpenECOMP Orchestration parameter; the value is provided to the Heat template by OpenECOMP.

Availability zones must be passed as individual numbered parameters (not as arrays) so that VNFs with multi-availability zone requirements can clearly specify that in its parameter definitions.

The availability zone parameter must be defined as “availability_zone_{index}”, with the {index} starting at zero.

Example:

```
In this example, the {vm-type} has been defined as “lb” for load balancer.

parameters:
       lb_names:
              type: comma_delimited_list
              description: VM Names for lb VMs

       availability_zone_0:
              type: string
              description: First availability zone ID or Name

resources:
       lb_0:
              type: OS::Nova::Server
              properties:
                     name: { get_param: [lb_names, 0] }
                     availability_zone: { get_param: availability_zone_0 }
                     ...
```

4.4 Resource: OS::Nova::Server - Metadata

This section describes the OpenECOMP Metadata parameters.

OpenECOMP Heat templates must include the following three parameters that are used as metadata under the resource OS::Nova:Server: vnf_id, vf_module_id, vnf_name

OpenECOMP Heat templates may include the following parameter that is used as metadata under the resource OS::Nova:Server: vf_module_name.

These parameters are all classified as OpenECOMP Metadata.

Metadata Parameter Name	Parameter Type	Mandatory/Optional
vnf_id	string	mandatory
vf_module_id	string	mandatory
vnf_name	string	mandatory
vf_module_name	string	optional

Table 3 OpenECOMP Metadata

4.4.1 Required Metadata Elements

The vnf_id, vf_module_id, and vnf_name metadata elements are required (must) for *OS::Nova::Server* resources. The metadata parameters will be used by OpenECOMP to associate the servers with the VNF instance.

- vnf_id
  - *"vnf_id"* parameter value will be supplied by OpenECOMP. OpenECOMP generates the UUID that is the vnf_id and supplies it to the Heat at orchestration time.
- vf_module_id
  - "*vf_module_id"* parameter value will be supplied by OpenECOMP. OpenECOMP generates the UUID that is the vf_module_id and supplies it to the Heat at orchestration time.
- vnf_name
  - "*vnf_name"* parameter value will be generated and/or assigned by OpenECOMP and supplied to the Heat by OpenECOMP at orchestration time.

4.4.2 Optional Metadata Elements

The following metadata element is optional for *OS::Nova::Server* resources:

- *vf_module_name*
  - The vf_module_name is the name of the name of the Heat stack (e.g., <STACK_NAME>) in the command "Heat stack-create" (e.g. `Heat stack-create [-f <FILE>] [-e <FILE>] <STACK_NAME>`). The <STACK_NAME> needs to be specified as part of the orchestration process.
  - *"vf_module_name"* parameter value, when used, will be supplied by OpenECOMP to the Heat at orchestration time. The parameter will be generated and/or assigned by OpenECOMP and supplied to the Heat by OpenECOMP at orchestration time.

Example

```
In this example, the {vm-type} has been defined as “lb” for load balancer.

parameters:
       vnf_name:
              type: string
              description: Unique name for this VNF instance
       vnf_id:
              type: string
              description: Unique ID for this VNF instance
       vf_module_name:
              type: string
              description: Unique name for this VNF Module instance
       vf_module_id:
              type: string
              description: Unique ID for this VNF Module instance

resources:
       lb_server_group:
              type: OS::Nova::ServerGroup
              properties:
                     name:
                        str_replace:
                           template: VNF_NAME_lb_ServerGroup
                           params:
                             VNF_NAME: { get_param: VNF_name }
                     policies: [ ‘anti-affinity’ ]

       lb_vm_0:
              type: OS::Nova::Server
              properties:
                     name: { get_param: lb_name_0 }
                     scheduler_hints:
                     group: { get_resource: lb_server_group }
                     metadata:
                         vnf_name: { get_param: vnf_name }
                         vnf_id: { get_param: vnf_id }
                         vf_module_name: { get_param: vf_module_name }
                         vf_module_id: { get_param: vf_module_id }
                     ...
```

4.5 Resource: OS::Neutron::Port - Parameters

The following four OS::Neutron::Port Resource Property Parameters must adhere to the OpenECOMP parameter naming convention.

- network
- subnet
- fixed_ips
- allowed_address_pairs

These four parameters reference a network, which maybe an external network or an internal network. Thus the parameter will include {network-role} in its name.

When the parameter references an external network, the parameter is an OpenECOMP Orchestration Parameter. The parameter value must be supplied by OpenECOMP. The parameters must adhere to the OpenECOMP parameter naming convention.

OS::Neutron::Port		
Property	Parameter Name for External Networks	Parameter Type
Network	{network-role}_net_id	string
	{network-role}_net_name	string
Subnet	{network-role}_subnet_id	string
	{network-role}_v6_subnet_id	string
fixed_ips	{vm-type}_{network-role}_ip_{index}	string
	{vm-type}_{network-role}_ips	CDL
	{vm-type}_{network-role}_v6_ip_{index}	string
	{vm-type}_{network-role}_v6_ips	CDL
allowed_address_pairs	{vm-type}_{network-role}_floating_ip	string
	{vm-type}_{network-role}_floating_v6_ip	string
	{vm-type}_{network-role}_ip_{index}	string
	{vm-type}_{network-role}_ips	CDL
	{vm-type}_{network-role}_v6_ip_{index}	string
	{vm-type}_{network-role}_v6_ips	CDL

Table 4 Port Resource Property Parameters (External Networks)

When the parameter references an internal network, the parameter is a VNF Orchestration Parameters. The parameter value(s) must be supplied either via an output statement(s) in the base module (i.e., OpenECOMP Base Template Output Parameters) or be enumerated in the environment file. The parameters must adhere to the following parameter naming convention.

OS::Neutron::Port		
Property	Parameter Name for Internal Networks	Parameter Type
Network	int_{network-role}_net_id	string
	int_{network-role}_net_name	string
Subnet	int_{network-role}_subnet_id	string
	Int_{network-role}_v6_subnet_id	string
fixed_ips	{vm-type}_int_{network-role}_ip_{index}	string
	{vm-type}_int_{network-role}_ips	CDL
	{vm-type}_int_{network-role}_v6_ip_{index}	string
	{vm-type}_int_{network-role}_v6_ips	CDL
allowed_address_pairs	{vm-type}_int_{network-role}_floating_ip	string
	{vm-type}_int_{network-role}_floating_v6_ip	string
	{vm-type}_int_{network-role}_ip_{index}	string
	{vm-type}_int_{network-role}_ips	CDL
	{vm-type}_int_{network-role}_v6_ip_{index}	string
	{vm-type}_int_{network-role}_v6_ips	CDL

Table 5 Port Resource Property Parameters (Internal Networks)

4.5.1 Property: network & subnet

The property “networks” in the resource OS::Neutron::Port must be referenced by Neutron Network ID, a UUID value, or by the network name defined in OpenStack.

When the parameter is referencing an “external” network, the parameter must adhere to the following naming convention

- *“{*network-role}_net_id”, for the Neutron network ID
- “{network-role}_net_name”, for the network name in OpenStack

When the parameter is referencing an “internal” network, the parameter must adhere to the following naming convention.

- “*int_{network-role}_net_id*”, for the Neutron network ID
- “*int_{network-role}_net_name*”, for the network name in OpenStack

The property “subnet_id” must be used if a DHCP IP address assignment is being requested and the DHCP IP address assignment is targeted at a specific subnet.

The property “subnet_id” should not be used if all IP assignments are fixed, or if the DHCP assignment does not target a specific subnet

When the parameter is referencing an “external” network subnet, the “subnet_id” parameter must adhere to the following naming convention.

- “*{network-role}_subnet_id*” if the subnet is an IPv4 subnet
- “*{network-role}_v6_subnet_id”* if the subnet is an IPv6 subnet

When the parameter is referencing an “internal” network subnet, the “subnet_id” parameter must adhere to the following naming convention.

- “*int_{network-role}_subnet_id*” if the subnet is an IPv4 subnet
- “*int_{network-role}_v6_subnet_id*” if the subnet is an IPv6 subnet

Example:

```
parameters:
       {network-role}_net_id:
              type: string
              description: Neutron UUID for the {network-role} network

       {network-role}_net_name:
              type: string
              description: Neutron name for the {network-role} network

       {network-role}_subnet_id:
              type: string
              description: Neutron subnet UUID for the {network-role} network

       {network-role}_v6_subnet_id:
              type: string
              description: Neutron subnet UUID for the {network-role} network
```

Example:

```
In this example, the {network-role} has been defined as “oam” to represent an oam
network and the {vm-type} has been defined as “lb” for load balancer.

parameters:
       oam_net_id:
              type: string
              description: Neutron UUID for the oam network

resources:
       lb_port_1:
              type: OS::Neutron::Port
              network: { get_param: oam_net_id }
```

4.5.2 Property: fixed_ips

The property “fixed_ips” in the resource OS::Neutron::Port must be used when statically assigning IP addresses.

An IP address is assigned to a port on a type of VM (i.e., {vm-type}) that is connected to a type of network (i.e., {network-role}). These two tags are components of the parameter name.

When the “fixed_ips” parameter is referencing an “external” network, the parameter must adhere to the naming convention below. The parameter may be a comma delimited list or a string.

There must be a different parameter name for IPv4 IP addresses and IPv6 addresses

- **Comma-delimited list:** Each element in the IP list should be assigned to successive instances of that VM type on that network.
  - *Format for IPv4 addresses:* {vm-type}_{network-role}_ips
  - *Format for IPv6 addresses:* {vm-type}_{network-role}_v6_ips
- **A set of fixed-index parameters:** In this case, the parameter should have “*type: string*” and must be repeated for every IP expected for each {vm-type} + {network-role} pair.
  - *Format for IPv4 addresses:* {vm-type}_{network-role}_ip_{index}

- *Format for IPv6 addresses:* {vm-type}_{network-role}_v6_ip_{index}

When the “fixed_ips” parameter is referencing an “internal” network, the parameter must adhere to the naming convention below. The parameter may be a comma delimited list or a string.

There must be a different parameter name for IPv4 IP addresses and IPv6 addresses

- **Comma-delimited list:** Each element in the IP list should be assigned to successive instances of that VM type on that network.
  - *Format for IPv4 addresses:* {vm-type}_int_{network-role}_ips
  - *Format for IPv6 addresses:* {vm-type}_int_{network-role}_v6_ips
- **A set of fixed-index parameters:** In this case, the parameter should have “*type: string*” and must be repeated for every IP expected for each {vm-type} and {network-role}pair.
  - *Format for IPv4 addresses:* {vm-type}_int_{network-role}_ip_{index}
  - *Format for IPv6 addresses:* {vm-type}_int_{network-role}_v6_ip_{index}

If a VNF contains more than three IP addresses for a given {vm-type} and {network-role} combination, the CDL form of the parameter name should be used to minimize the number of unique parameters defined in the Heat.

Example (external network)

```
parameters:

       {vm-type}_{network-role}_ips:
              type: comma_delimited_list
              description: Fixed IPv4 assignments for {vm-type} VMs on the {network-
role} network

       {vm-type}_{network-role}_v6_ips:
              type: comma_delimited_list
              description: Fixed IPv6 assignments for {vm-type} VMs on the {network-
role} network

       {vm-type}_{network-role}_ip_{index}:
              type: string
              description: Fixed IPv4 assignment for {vm-type} VM {index} on the
              {network-role} network

       {vm-type}_{network-role}_v6_ip_{index}:
              type: string
              description: Fixed IPv6 assignment for {vm-type} VM {index} on the
              {network-role} network
```

Example (CDL parameter for IPv4 Address Assignments to an external network):

```
In this example, the {network-role} has been defined as “oam” to represent an oam
network and the {vm-type} has been defined as “db” for database.

parameters:
       oam_net_id:
              type: string
              description: Neutron UUID for a oam network

       db_oam_ips:
              type: comma_delimited_list
              description: Fixed IP assignments for db VMs on the oam network

resources:
       db_0_port_1:
              type: OS::Neutron::Port
              network: { get_param: oam_net_id }
              fixed_ips: [ { “ip_address”: {get_param: [ db_oam_ips, 0]
              }}]

       db_1_port_1:
              type: OS::Neutron::Port
              network: { get_param: oam_net_id }
              fixed_ips: [ { “ip_address”: {get_param: [ db_oam_ips, 1]
              }}]
```

Example (string parameters for IPv4 Address Assignments to an external network):

```
In this example, the {network-role} has been defined as “oam” to represent an oam
network and the {vm-type} has been defined as “db” for database.

parameters:
       oam_net_id:
              type: string
              description: Neutron UUID for an OAM network

       db_oam_ip_0:
              type: string
              description: First fixed IP assignment for db VMs on the OAM network

       db_oam_ip_1:
              type: string
              description: Second fixed IP assignment for db VMs on the OAM network

resources:
       db_0_port_1:
              type: OS::Neutron::Port
              network: { get_param: oam_net_id }
              fixed_ips: [ { “ip_address”: {get_param: db_oam_ip_0}}]

       db_1_port_1:
              type: OS::Neutron::Port
              network: { get_param: oam_net_id }
              fixed_ips: [ { “ip_address”: {get_param: db_oam_ip_1}}]
```

4.5.3 Property: allowed_address_pairs

The property “allowed_address_pairs” in the resource OS::Neutron::Port allows the user to specify mac_address/ip_address (CIDR) pairs that pass through a port regardless of subnet. This enables the use of protocols such as VRRP, which floats an IP address between two instances to enable fast data plane failover. An “allowed_address_pairs” is unique to a {vm-type} and {network-role} combination. The management of these IP addresses (i.e. transferring ownership between active and standby VMs) is the responsibility of the application itself.

Note that these parameters are *not* intended to represent Neutron “Floating IP” resources, for which OpenStack manages a pool of public IP addresses that are mapped to specific VM ports. In that case, the individual VMs are not even aware of the public IPs, and all assignment of public IPs to VMs is via OpenStack commands. OpenECOMP does not support Neutron-style Floating IPs.

Both IPv4 and IPv6 “allowed_address_pairs” addresses are supported.

If property “allowed_address_pairs” is used with an external network, the parameter name must adhere to the following convention:

- *Format for IPv4 addresses: {vm-type}_{network-role}_floating_ip*
- *Format for IPv6 addresses: {vm-type}_{network-role}_floating_v6_ip*

Example:

```
parameters:

       {vm-type}_{network-role}_floating_ip:
              type: string
              description: VIP for {vm-type} VMs on the {network-role} network

       {vm-type}_{network-role}_floating_v6_ip:
              type: string
              description: VIP for {vm-type} VMs on the {network-role} network
```

Example:

```
In this example, the {network-role} has been defined as “oam” to represent an oam
network and the {vm-type} has been defined as “db” for database.

parameters:
       db_oam_ips:
              type: comma_delimited_list
              description: Fixed IPs for db VMs on the oam network

       db_oam_floating_ip:
              type: string
              description: Floating IP for db VMs on the oam network

resources:
       db_0_port_0:
              type: OS::Neutron::Port
              network: { get_param: oam_net_id }
              fixed_ips: [ { “ip_address”: {get_param: [db_oam_ips,0] }}]
              allowed_address_pairs: [
                     { “ip_address”: {get_param: db_oam_floating_ip}}]

db_1_port_0:
              type: OS::Neutron::Port
              network: { get_param: oam_net_id }
              fixed_ips: [ { “ip_address”: {get_param: [db_oam_ips,1] }}]
              allowed_address_pairs: [
                     { “ip_address”: {get_param: db_oam_floating_ip}}]
```

If property “allowed_address_pairs” is used with an internal network, the parameter name should adhere to the following convention:

- *Format for IPv4 addresses: {vm-type}_int_{network-role}_floating_ip*
- *Format for IPv6 addresses: {vm-type}_int_{network-role}_floating_v6_ip*

Using the parameter *{vm-type}_{network-role}_floating_ip* or *{vm-type}_{network-role}_floating_v6_ip* provides only one floating IP per Vm-type{vm-type} and {network-role} pair. If there is a need for multiple floating IPs (e.g., Virtual IPs (VIPs)) for a given {vm-type} and {network-role} combination within a VNF, then the parameter names defined for the “fixed_ips” should be used with the “allowed_address_pairs” property. The examples below illustrate this.

Below example reflects two load balancer pairs in a single VNF. Each pair has one VIP.

Example: A VNF has four load balancers. Each pair has a unique VIP.

Pair 1: lb_0 and lb_1 share a unique VIP
Pair 2: lb_2 and lb_3 share a unique VIP

```
In this example, the {network-role} has been defined as “oam” to represent an oam
network and the {vm-type} has been defined as “lb” for load balancer.

resources:
      lb_0_port_0:
              type: OS::Neutron::Port
              network: { get_param: oam_net_id }
              fixed_ips: [ { “ip_address”: {get_param: [lb_oam_ips,0] }}]
              allowed_address_pairs: [{ “ip_address”: {get_param: [lb_oam_ips,2] }}]

      lb_1_port_0:
              type: OS::Neutron::Port
              network: { get_param: oam_net_id }
              fixed_ips: [ { “ip_address”: {get_param: [lb_oam_ips,1] }}]
              allowed_address_pairs: [{ “ip_address”: {get_param: [lb_oam_ips,2] }}]

      lb_2_port_0:
              type: OS::Neutron::Port
              network: { get_param: oam_net_id }
              fixed_ips: [ { “ip_address”: {get_param: [lb_oam_ips,3] }}]
              allowed_address_pairs: [{ “ip_address”: {get_param: [lb_oam_ips,5] }}]

      lb_3_port_0:
              type: OS::Neutron::Port
              network: { get_param: oam_net_id }
              fixed_ips: [ { “ip_address”: {get_param: [lb_oam_ips,4] }}]
              allowed_address_pairs: [{ “ip_address”: {get_param: [lb_oam_ips,5] }}]
```

Below example reflects a single app VM pair within a VNF with two VIPs:

Example: A VNF has two load balancers. The pair of load balancers share two VIPs.

```
In this example, the {network-role} has been defined as “oam” to represent an oam
network and the {vm-type} has been defined as “lb” for load balancer.

resources:
       lb_0_port_0:
              type: OS::Neutron::Port
              network: { get_param: oam_net_id }
              fixed_ips: [ { “ip_address”: {get_param: [lb_oam_ips,0] }}]
              allowed_address_pairs: [{ "ip_address": {get_param: [lb_oam_ips,2] },
{get_param: [lb_oam_ips,3] }}]

       lb_1_port_0:
              type: OS::Neutron::Port
              network: { get_param: oam_net_id }
              fixed_ips: [ { “ip_address”: {get_param: [lb_oam_ips,1] }}]
              allowed_address_pairs: [{ "ip_address": {get_param: [lb_oam_ips,2] },
{get_param: [lb_oam_ips,3] }}]
```

As a general rule, provide the fixed IPs for the VMs indexed first in the CDL and then the VIPs as shown in the examples above.

4.6 Resource Property: name

The parameter naming standard for the resource OS::Nova::Server has been defined in Section 4.3.3. This section describes how the name property of all other resources must be defined.

Heat templates must use the Heat “str_replace” function in conjunction with the OpenECOMP supplied metadata parameter *vnf_name* or *vnf_module_id* to generate a unique name for each VNF instance. This prevents the use of unique parameters values for resource “name” properties to be enumerated in a per instance environment file.

Note that

- In most cases, only the use of the vnf_name is necessary to create a unique name
- the Heat pseudo parameter 'OS::stack_name’ can also be used in the ‘str_replace’ construct to generate a unique name when the vnf_name does not provide uniqueness

```
type: OS::Cinder::Volume
  properities:
    name:
      str_replace:
        template: VF_NAME_STACK_NAME_oam_volume
        params:
          VF_NAME: { get_param: vnf_name }
          STACK_NAME: { get_param: 'OS::stack_name'  }


  type: OS::Neutron::SecurityGroup
  properties:
    description: Security Group of Firewall
    name:
      str_replace:
        template: VNF_NAME_Firewall_SecurityGroup
        params:
          VNF_NAME: { get_param: vnf_name }
```

4.7 Output Parameters

OpenECOMP defines three type of Output Parameters.

4.7.1 Base Template Output Parameters:

The base template output parameters are available for use as input parameters in all add-on modules. The add-on modules may (or may not) use these parameters.

4.7.2 Volume Template Output Parameters:

The volume template output parameters are only available only for the module (base or add on) that the volume is associated with.

4.7.3 Predefined Output Parameters

OpenECOMP currently defines one predefined output parameter.

4.7.3.1 OAM Management IP Addresses

Many VNFs will have a management interface for application controllers to interact with and configure the VNF. Typically, this will be via a specific VM that performs a VNF administration function. The IP address of this interface must be captured and inventoried by OpenECOMP. This might be a VIP if the VNF contains an HA pair of management VMs, or may be a single IP address assigned to one VM.

The Heat template may define either (or both) of the following Output parameters to identify the management IP address.

- *oam_management_v4_address*
- *oam_management_v6_address*

Notes:

- The Management IP Address should be defined only once per VNF, so it would only appear in one Module template
- If a fixed IP for the admin VM is passed as an input parameter, it may be echoed in the output parameters
- If the IP for the admin VM is obtained via DHCP, it may be obtained from the resource attributes

Example:

```
resources:
    admin_server:
       type: OS::Nova::Server
       properties:
              networks:
                     - network: {get_param: oam_net_id }
              ...

Outputs:
    oam_management_v4_address:
       value: {get_attr: [admin_server, networks, {get_param: oam_net_id}, 0] }
```

5. Heat Template Constructs

5.1 External References

Heat templates *should not* reference any HTTP-based resource definitions, any HTTP-based nested configurations, or any HTTP-based environment files.

- During orchestration, OpenECOMP *should not* retrieve any such resources from external/untrusted/unknown sources.
- VNF images should not contain such references in user-data or other configuration/operational scripts that are specified via Heat or encoded into the VNF image itself.

Note: HTTP-based references are acceptable if the HTTP-based reference is accessing information with the VM private/internal network.

5.2 Heat Files Support (get_file)

Heat Templates may contain the inclusion of text files into Heat templates via the Heat “get_file” directive. This may be used, for example, to define a common “user-data” script, or to inject files into a VM on startup via the “personality” property.

Support for Heat Files is subject to the following limitations:

- The ‘get_files’ targets must be referenced in Heat templates by file name, and the corresponding files should be delivered to OpenECOMP along with the Heat templates.
    - URL-based file retrieval must not be used; it is not supported.
- The included files must have unique file names within the scope of the VNF.
- OpenECOMP does not support a directory hierarchy for included files.
    - All files must be in a single, flat directory per VNF.
- Included files may be used by all Modules within a given VNF.
- get_file directives may be used in both non-nested and nested templates

5.3 Use of Heat ResourceGroup

The *OS::Heat::ResourceGroup* is a useful Heat element for creating multiple instances of a given resource or collection of resources. Typically it is used with a nested Heat template, to create, for example, a set of identical *OS::Nova::Server* resources plus their related *OS::Neutron::Port* resources via a single resource in a master template.

ResourceGroup may be used in OpenECOMP to simplify the structure of a Heat template that creates multiple instances of the same VM type. However, there are important caveats to be aware of.

ResourceGroup does not deal with structured parameters (comma-delimited-list and json) as one might typically expect. In particular, when using a list-based parameter, where each list element corresponds to one instance of the *ResourceGroup*, it is not possible to use the intrinsic “loop variable” %index% in the *ResourceGroup* definition.

For instance, the following is **not** valid Heat for a *ResourceGroup*:

```
type: OS::Heat::ResourceGroup
resource:
     type: my_nested_vm_template.yaml
     properties:
         name: {get_param: [vm_name_list, %index%]}
```

Although this appears to use the nth entry of the *vm_name_list* list for the nth element of the *ResourceGroup*, it will in fact result in a Heat exception. When parameters are provided as a list (one for each element of a *ResourceGroup*), you must pass the complete parameter to the nested template along with the current index as separate parameters.

Below is an example of an **acceptable** Heat Syntax for a *ResourceGroup*:

```
type: OS::Heat::ResourceGroup
resource:
    type: my_nested_vm_template.yaml
    properties:
        names: {get_param: vm_name_list}
        index: %index%
```

You can then reference within the nested template as:

```
{ get_param: [names, {get_param: index} ] }
```

Note that this is workaround has very important limitations. Since the entire list parameter is passed to the nested template, any change to that list (e.g., adding an additional element) will cause Heat to treat the entire parameter as updated within the context of the nested template (i.e., for each *ResourceGroup* element). As a result, if *ResourceGroup* is ever used for scaling (e.g., increment the count and include an additional element to each list parameter), Heat will often rebuild every existing element in addition to adding the "deltas". For this reason, use of *ResourceGroup* for scaling in this manner is not supported.

5.4 Key Pairs

When Nova Servers are created via Heat templates, they may be passed a "keypair" which provides an ssh key to the 'root' login on the newly created VM. This is often done so that an initial root key/password does not need to be hard-coded into the image.

Key pairs are unusual in OpenStack, because they are the one resource that is owned by an OpenStack User as opposed to being owned by an OpenStack Tenant. As a result, they are usable only by the User that created the keypair. This causes a problem when a Heat template attempts to reference a keypair by name, because it assumes that the keypair was previously created by a specific OpenECOMP user ID.

When a keypair is assigned to a server, the SSH public-key is provisioned on the VMs at instantiation time. They keypair itself is not referenced further by the VM (i.e. if the keypair is updated with a new public key, it would only apply to subsequent VMs created with that keypair).

Due to this behavior, the recommended usage of keypairs is in a more generic manner which does not require the pre-requisite creation of a keypair. The Heat should be structured in such a way as to:

- Pass a public key as a parameter value instead of a keypair name
- Create a new keypair within the VNF Heat templates (in the base module) for use within that VNF

By following this approach, the end result is the same as pre-creating the keypair using the public key – i.e., that public key will be provisioned in the new VM. However, this recommended approach also makes sure that a known public key is supplied (instead of having OpenStack generate a public/private pair to be saved and tracked outside of OpenECOMP). It also removes any access/ownership issues over the created keypair.

The public keys may be enumerated as a VNF Orchestration Constant in the environment file (since it is public, it is not a secret key), or passed at run-time as an instance-specific parameters. OpenECOMP will never automatically assign a public/private key pair.

Example (create keypair with an existing ssh public-key for {vm-type} of lb (for load balancer)):

```
parameters:
   vnf_name:
      type: string
   ssh_public_key:
      type: string

resources:
  my_keypair:
     type: OS::Nova::Keypair
     properties:
        name:
          str_replace:
            template: VNF_NAME_key_pair
            params:
            VNF_NAME: { get_param: vnf_name }
        public_key: {get_param: lb_ssh_public_key}
        save_private_key: false
```

5.5 Security Groups

OpenStack allows a tenant to create Security groups and define rules within the security groups.

Security groups, with their rules, may either be created in the Heat template or they can be pre-created in OpenStack and referenced within the Heat template via parameter(s). There can be a different approach for security groups assigned to ports on internal (intra-VNF) networks or external networks (inter-VNF). Furthermore, there can be a common security group across all VMs for a specific network or it can vary by VM (i.e., {vm-type}) and network type (i.e., {network-role}).

5.6 Anti-Affinity and Affinity Rules

Anti-affinity or affinity rules are supported using normal OpenStack *"OS::Nova::ServerGroup"* resources. Separate ServerGroups are typically created for each VM type to prevent them from residing on the same host, but they can be applied to multiple VM types to extend the affinity/anti-affinity across related VM types as well.

Example:

```
In this example, the {network-role} has been defined as “oam” to represent an oam
network and the {vm-type} have been defined as “lb” for load balancer and “db” for
database.

resources:
       db_server_group:
           type: OS::Nova::ServerGroup
           properties:
             name:
               str_replace:
                 params:
                      $vnf_name: {get_param: vnf_name}
                 template: $vnf_name-server_group1
             policies:
               - anti-affinity

       lb_server_group:
           type: OS::Nova::ServerGroup
           properties:
             name:
               str_replace:
                 params:
                      $vnf_name: {get_param: vnf_name}
                 template: $vnf_name-server_group2
             policies:
               - affinity


       db_0:
           type: OS::Nova::Server
           properties:
           ...
           scheduler_hints:
               group: {get_param: db_server_group}

       db_1:
           type: OS::Nova::Server
           properties:
           ...
           scheduler_hints:
               group: {get_param: db_server_group}

       lb_0:
           type: OS::Nova::Server
           properties:
           ...
           scheduler_hints:
           group: {get_param: lb_server_group}
```

6. Design Pattern: VNF Modularity

OpenECOMP supports the concept of *VNF Modularity*. With this approach, a single VNF may be composed from one or more Heat templates, each of which represents some subset of the overall VNF. These component parts are referred to as “*VNF Modules*”. During orchestration, these modules may be deployed incrementally to build up the complete VNF.

A Heat template can be either one for the following types of modules

1. Base Module
2. Incremental Modules
3. Independent Cinder Volume Modules

The OpenECOMP Heat template naming convention must be followed (Section 2.1). The naming convention identifies the module type.

A VNF must be composed of one “base” VNF module (also called a base module) and zero to many “incremental” or “add on” VNF modules. The base module must be deployed first prior to the add-on modules.

A module can be thought of as equivalent to a Heat template, where a Heat template is composed of a YAML file and an environment file. A given YAML file must have a corresponding environment file; OpenECOMP requires it. A Heat template is used to create or deploy a Heat stack. Therefore, a module is also equivalent to a Heat Stack.

However, there are cases where a module maybe composed of more than one Heat stack and/or more than one YAML file.

As discussed in Section 2.5, Independent Volume Templates, each VNF Module may have an associated Volume template.

- When a volume template is utilized, it must correspond 1:1 with add-on module template or base template it is associated with
- A Cinder volume may be embedded within the add-on module template and/or base template if persistence is not required, thus not requiring the optional Volume template.

A VNF module may support nested templates. In this case, there will be one or more additional YAML files.

Any shared resource defined in the base module template and used across the entire VNF (e.g., private networks, server groups), must be exposed to the incremental or add-on modules by declaring their resource UUIDs as Heat outputs (i.e., OpenECOMP Base Template Output Parameter in the output section of the Heat template). Those outputs will be provided by OpenECOMP as input parameter values to all add-on module Heat templates in the VNF that have declared the parameter in the template.

Note: A Cinder volume is <u>not</u> considered a shared resource. A volume template must correspond 1:1 with a base template or add-on module template.

There are two suggested usage patterns for modular VNFs, though any variation is supported.

A Modules per VNFC type

a. Group all VMs (VNFCs) of a given type into its own module
b. Build up the VNF one VNFC type at a time
c. Base module contains only the shared resources (and possibly initial Admin VMs)
d. Suggest one or two modules per VNFC type
    i. one for initial count
    ii. one for scaling increment (if different from initial count)

B Base VNF + Growth Units

a. Base module (template) contains a complete initial VNF instance
b. Growth modules for incremental scaling units
    i. May contain VMs of multiple types in logical scaling combinations
    ii. May be separated by VM type for multi-dimensional scaling
c. With no growth units, this is equivalent to the "*One Heat Template per VNF*" model

Note that modularization of VNFs is not required. A single Heat template (a base template) may still define a complete VNF, which might be appropriate for smaller VNFs without a lot of scaling options.

There are some rules to follow when building modular VNF templates:

1. All VNFs must have one Base VNF Module (template) that must be the first one deployed. The base template:
    a. Must include all shared resources (e.g., private networks, server groups, security groups)
    b. Must expose all shared resources (by UUID) as "outputs" in its associated Heat template (i.e., OpenECOMP Base Template Output Parameters)
    c. May include initial set of VMs
    d. May be operational as a stand-alone "minimum" configuration of the VNF
2. VNFs may have one or more Add-On VNF Modules (templates) which:
    a. Defines additional resources that can be added to an existing VNF
    b. Must be complete Heat templates
        i. i.e. not snippets to be incorporated into some larger template
    c. Should define logical growth-units or sub-components of an overall VNF
    d. On creation, receives all Base VNF Module outputs as parameters
        i. Provides access to all shared resources (by UUID)
        ii. must not be dependent on other Add-On VNF Modules
    e. Multiple instances of an Add-On VNF Module type may be added to the same VNF (e.g. incrementally grow a VNF by a fixed "add-on" growth units)
3. Each VNF Module (base or add-on) may have (optional) an associated Volume template (*see Section 2.5*)
    a. Volume templates should correspond 1:1 with Module (base or add-on) templates
    b. A Cinder volume may be embedded within the Module template (base or add-on) if persistence is not required
4. Shared resource UUIDs are passed between the base template and add-on template via Heat Outputs Parameters (i.e., Base Template Output Parameters)
    a. The output parameter name in the base must match the parameter name in the add-on module

Examples:

```
In this example, the {vm-type} have been defined as "lb" for load balancer and "admin"
for admin server.
```

1. Base VNF Module Heat Template (partial)

Heat_template_version: 2013-05-23

```
parameters:
       admin_name_0:
              type: string

resources:
       int_oam_network:
              type: OS::Neutron::Network
              properties:
                     name: {… }
       admin_server:
              type: OS::Nova::Server
              properties:
                     name: {get_param: admin_name_0}
                     image: ...

outputs:
       int_oam_net_id:
              value: {get_resource: int_oam_network }
```

2. Add-on VNF Module Heat Template (partial)

```
Heat_template_version: 2013-05-23

Parameters:
       int_oam_net_id:
              type: string
              description: ID of shared private network from Base template
       lb_name_0:
              type: string
              description: name for the add-on VM instance

Resources:
       lb_server:
              type: OS::Nova::Server
              properties:
                     name: {get_param: lb_name_0}
                     networks:
                            - port: { get_resource: lb_port }
                            ...
       lb_port:
              type: OS::Neutron::Port
              properties:
                     network_id: { get_param: int_oam_net_id }
...
```

7. Scaling Considerations

Scaling of a VNF may be manually driven to add new capacity (**static scaling**) or it may be driven in near real-time by the OpenECOMP controllers based on a real-time need **(dynamic scaling).**

With VNF Modularity, the recommended approach for scaling is to provide additional “growth unit” templates that can be used to create additional resources in logical scaling increments. This approach is very straightforward, and has minimal impact on the currently running VNFCs and must comply with the following:

- Combine resources into reasonable-sized scaling increments; do not just scale by one VM at a time in potentially large VNFs.

- Combine related resources into the same growth template where appropriate, e.g. if VMs of different types are always deployed in pairs, include them in a single growth template.
- Growth templates can use the private networks and other shared resources exposed by the Base Module template.

VNF Modules may also be updated “in-place” using the OpenStack Heat Update capability, by deploying an updated Heat template with different VM counts to an existing stack. This method requires another VNF module template that includes the new resources *in addition to all resources contained in the original module template*. Note that this also requires re-specification of all existing parameters as well as new ones.

For this approach:

- Use a fixed number of pre-defined VNF module configurations
- Successively larger templates must be identical to the next smaller one, plus add the additional VMs of the scalable type(s)
- VNF is scalable by sending a stack-update with a different template

Please do note that:

- If properties do not change for existing VMs, those VMs should remain unchanged
- If the update is performed with a smaller template, the Heat engine recognizes and deletes no-longer-needed VMs (and associated resources)
- Nested templates for the various server types will simplify reuse across multiple configurations
- Per the section on Use of Heat ResourceGroup, if *ResourceGroup* is ever used for scaling (e.g. increment the count and include an additional element to each list parameter), Heat will often rebuild every existing element in addition to adding the “deltas”. For this reason, use of *ResourceGroup* for scaling in this manner is not supported.

8. High Availability

VNF/VM parameters may include availability zone IDs for VNFs that require high availability.

The Heat must comply with the following requirements to specific availability zone IDs:

- The Heat template should spread Nova and Cinder resources across the availability zones as desired

9. Resource Data Synchronization

For cases where synchronization is required in the orchestration of Heat resources, two approaches are recommended:

- Standard Heat *“depends_on”* property for resources
    - Assures that one resource completes before the dependent resource is orchestrated.

- Definition of completeness to OpenStack may not be sufficient (e.g., a VM is considered complete by OpenStack when it is ready to be booted, not when the application is up and running).

- Use of Heat Notifications
  - Create *OS::Heat::WaitCondition* and *OS::Heat::WaitConditionHandle* resources.
  - Pre-requisite resources issue *wc_notify* commands in user_data.
  - Dependent resource define *"depends_on"* in the *OS::Heat::WaitCondition* resource.

Example: "depends_on" case

```
In this example, the {network-role} has been defined as "oam" to represent an oam
network and the {vm-type} has been defined as "oam" to represent an oam server.

oam_server_01:
    type: OS::Nova::Server
    properties:
      name: {get_param: [oam_ names, 0]}
      image: {get_param: oam_image_name}
      flavor: {get_param: oam_flavor_name}
      availability_zone: {get_param: availability_zone_0}
      networks:
        - port: {get_resource: oam01_port_0}
        - port: {get_resource: oam01_port_1}
      user_data:
      scheduler_hints: {group: {get_resource: oam_servergroup}}
      user_data_format:  RAW

  oam_01_port_0:
    type: OS::Neutron::Port
    properties:
      network: {get_resource: oam_net_name}
      fixed_ips: [{"ip_address": {get_param: [oam_oam_net_ips, 1]}}]
      security_groups: [{get_resource: oam_security_group}]

  oam_01_port_1:
    type: OS::Neutron::Port
    properties:
      network: {get_param: oam_net_name}
      fixed_ips: [{"ip_address": {get_param: [oam_oam_net_ips, 2]}}]
      security_groups: [{get_resource: oam_security_group}]

  oam_01_vol_attachment:
    type: OS::Cinder::VolumeAttachment
    depends_on: oam_server_01
    properties:
      volume_id: {get_param: oam_vol_1}
      mountpoint: /dev/vdb
      instance_uuid: {get_resource: oam_server_01}
```

Appendix A - Glossary

VM Virtual Machine (VM) is a virtualized computation environment that behaves very much like a physical computer/server. A VM has all its ingredients (processor, memory/storage, interfaces/ports) of a physical computer/server and is generated by a hypervisor, which partitions the underlying physical resources and allocates them to VMs. Virtual Machines are capable of hosting a virtual network function component (VNFC).

VNF Virtual Network Function (VNF) is the software implementation of a function that can be deployed on a Network Cloud. It includes network functions that provide transport and forwarding. It also includes other functions when used to support network services, such as network-supporting web servers and database.

VNFC Virtual Network Function Component (VNFC) are the sub-components of a VNF providing a VNF Provider a defined sub-set of that VNF's functionality, with the main characteristic that a single instance of this component maps 1:1 against a single Virtualization Container. See Figure ***1*** for the relationship between VNFC and VNFs.

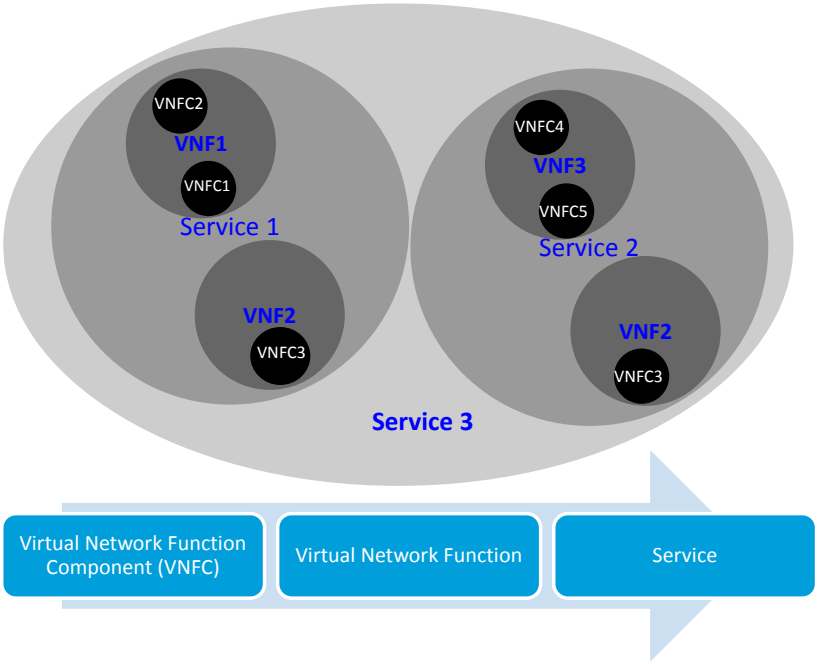

Figure 1. Virtual Function Entity Relationship