VNF
Heat Template
Requirements
for
Pre-Amsterdam ONAP

Revision	2017-2
Revision Date	6/30/2017

Document Revision History

Date	Revision	Description
2/1/2017	1.0	Initial publication of VNF Heat Template Requirement for Pre-Amsterdam ONAP
6/30/2017	2.0	Updated and restructured document.

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 document is part of a hierarchy of documents that describes the overall Requirements and Guidelines for ONAP (Open Network Automation Platform). The diagram below identifies where this document fits in the hierarchy.

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

Figure 1 High level organization of ONAP documents

Document summary:

VNF Guidelines for Network Cloud and ONAP

- Describes VNF environment and overview of requirements

VNF Cloud Readiness Requirements for ONAP

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

VNF Management Requirements for ONAP

- Requirements for how VNFs interact and utilize ONAP

VNF Heat Template Requirements for ONAP

- This document is the ***VNF Heat Template Requirements for ONAP***.

The ***VNF Heat Template Requirements for ONAP*** provides requirements and recommendations for building Heat templates compatible with pre-Amsterdam ONAP release. Pre-Amsterdam ONAP assumes Network Cloud implementations are OpenStack based.

Feedback on or questions about the content of this document may be sent to the following email address: VNFGuidelines@list.att.com.

1.1 Definitions

This section defines common terms used in the document.

1.1.1 OpenStack Glossary Definitions

The following terms are defined in the OpenStack Glossary located at https://docs.openstack.org/user-guide/common/glossary.html.

- Heat: Codename for the Orchestration service.

- Heat Orchestration Template (HOT): Heat input in the format native to OpenStack.
- orchestration service (heat): The OpenStack service which orchestrates composite cloud applications using a declarative template format through an OpenStack-native REST API.
- project: Projects represent the base unit of “ownership” in OpenStack, in that all resources in OpenStack should be owned by a specific project. In OpenStack Identity, a project must be owned by a specific domain.
- RESTful: A kind of web *service API that uses REST, or Representational State Transfer. REST is the style of architecture for hypermedia systems that is used for the World Wide Web.
- stack: A set of OpenStack resources created and managed by the Orchestration service according to a given template (either an AWS Cloud Formation template or a Heat Orchestration Template (HOT)).
- tenant: A group of users; used to isolate access to Compute resources. An alternative term for a project.

1.1.2 Additional Definitions

This section contains some additional useful definitions.

- Heat: OpenStack Orchestration Service, which provides an orchestration engine to launch multiple composite cloud applications based on Heat templates, which are YAML text files, readable by humans, that describe the infrastructure for a cloud application, including the servers, floating IPs, volumes, security groups, users, etc. Heat templates also specify the relationships between resources (for example, this volume is connected to this server). This enables Heat to call out to OpenStack APIs to create all of the infrastructure in the correct order to completely launch a cloud application.
- Heat environment file: A YAML text file which is primarily used to provide values for parameters defined in a Heat Orchestration Template. The environment file (or ENV file) may also be used to map or override Heat resources with other resources, to control stack creation on a given resource, to specify event consumers, etc.
- Heat Stack: The collection of objects (that is, resources) created by a Heat service, including instances (such as VMs), networks, subnets, routers, ports, router interfaces, user, security groups, security group rules, auto-scaling rules, etc.
- environment file: The heat environment file affects the runtime behavior of a template. It provides a way to override the resource implementations and a mechanism to place parameters that the service needs (https://docs.openstack.org/developer/heat/template_guide/environment.html). In ONAP’s implementation, the environment file must only contain pseudo constants.
- Pseudo Constants: parameters in the environment file that do not change (i.e., remain constant) across multiple instantiations of the same VNF.
- YAML: YAML (YAML Ain't Markup Language) is a human friendly data serialization standard for all programming languages (http://www.yaml.org/).

- External network: ONAP defines an external network in relation to the VNF and not with regard to the Network Cloud site. External networks may also be referred to as “inter-VNF” networks. An external network connects VMs in a VNF to
    - VMs in another VNF or
    - an external gateway or router
- Internal network: ONAP defines an internal network in relation to the VNF and not with regard to the Network Cloud site. Internal networks may also be referred to as “intra-VNF” networks or “private” networks. An internal network only connects VMs in a single VNF. It must not connect to other VNFs or an external gateway or router.
- Cloud Assigned IP Address: A Cloud assigned IP address is an IPv4 or IPv6 address assigned by OpenStack’s DHCP Service.
- ONAP SDN-C Assigned IP Address: An ONAP SDN-C assigned IP address is an IPv4 or IPv6 IP address is assigned by the ONAP SDN-C controller. The IP can be
    - automatically assigned by SDN-C policies
    - manually assigned and preloaded into SDN-C
- Predetermined Static IP Address: A predetermined static IP address is enumerated in the Heat environment file. A predetermined static IP address is used to attach a VM to an internal network and the IP is local to the VNF. Therefore, the IP addresses can be re-used at every VNF instance.

2. General Guidelines

2.1 YAML Format

Heat Orchestration Templates must use valid YAML. YAML (YAML Ain't Markup Language) is a human friendly data serialization standard for all programming languages. See http://www.yaml.org/.

2.2 Heat Orchestration Template Format

Heat Orchestration templates must be defined in YAML.

YAML rules include:

- Tabs are NOT allowed, use spaces ONLY.
- You MUST indent your properties and lists with 1 or more spaces.
- All Resource IDs and resource property parameters are case-sensitive. (e.g., "ThIs", is not the same as "thiS")

2.2.1 Heat Orchestration Template Structure

Heat Orchestration template structure follows the following format, as defined by OpenStack at https://docs.openstack.org/developer/heat/template_guide/hot_spec.html.

```
heat_template_version: <date>

description:
  # a description of the template

parameter_groups:
  # a declaration of input parameter groups and order

parameters:
  # declaration of input parameters

resources:
  # declaration of template resources

outputs:
  # declaration of output parameters

conditions:
  # declaration of conditions
```

Heat Orchestration templates for ONAP must contain the following sections:

- `heat_template_version:`
- `description:`
- `parameters:`
- `resources:`

Heat Orchestration templates for ONAP may contain the following sections:

- `parameter_groups:`
- `outputs:`

2.2.1.1 heat_template_version

This section is ONAP mandatory. The `heat_template_version` must be set to a date that is supported by the OpenStack environment.

2.2.1.2 description

This ONAP mandatory section allows for a description of the template.

2.2.1.3 parameter_groups

This ONAP optional section allows for specifying how the input parameters should be grouped and the order to provide the parameters in.

2.2.1.4 parameters

The parameter section is ONAP mandatory. This section allows for specifying input parameters that have to be provided when instantiating the template. Each parameter is specified in a separated nested block with the name of the parameters defined in the first line and additional attributes (e.g., `type`, `label`) defined as nested elements.

The Pre-Amsterdam VNF Validation Program (i.e., ICE Project) process requires all parameters declared in a template to be used in a resource with the exception of the parameters for the `OS::Nova::Server` property `availability_zone`. See Section 5.4.4 for more details on `availability_zone`.

```
parameters:
  <param name>:
    type: <string | number | json | comma_delimited_list | boolean>
    label: <human-readable name of the parameter>
    description: <description of the parameter>
    default: <default value for parameter>
    hidden: <true | false>
    constraints:
      <parameter constraints>
    immutable: <true | false>
```

- param name:
  - The name of the parameter.
  - ONAP requires that the param name must contain only alphanumeric characters and "_" underscores. Special characters must not be used.
- type:
  - The type of the parameter. Supported types are `string`, `number`, `comma_delimited_list`, `json` and `boolean`.
  - This attribute must be provided per the OpenStack Heat Orchestration Template standard.
- label:
  - A human readable name for the parameter.
  - This attribute is optional.
- description:
  - A human readable description for the parameter.
  - This attribute is ONAP mandatory; it must be provided. (Note that this attribute is OpenStack optional.)
- default:
  - A default value for the parameter.
  - ONAP does not support this attribute; it must not be provided in the Heat Orchestration Template. If a parameter has a default value, it must be provided in the environment file. (Note that this attribute is OpenStack optional.)
- hidden:
  - Defines whether the parameters should be hidden when a user requests information about a stack created from the template. This attribute can be used to hide passwords specified as parameters.
  - This attribute is optional and defaults to false.
- constraints:
  - A list of constraints to apply. The constraints block of a parameter definition defines additional validation constraints that apply to the value of the parameter. The parameter

values provided in the Heat Orchestration Template are validated against the constraints at instantiation time. The constraints are defined as a list with the following syntax

```
constraints:
  - <constraint type>: <constraint definition>
    description: <constraint description>
```

- constraint type: Type of constraint to apply.
- constraint definition: The actual constraint, depending on the constraint type.
- description: A description of the constraint. The text is presented to the user when the value the user defines violates the constraint. If omitted, a default validation message is presented to the user. This attribute is optional.

o When the parameter type is set to `number`, the Heat Orchestration Template uploaded into ONAP must have constraints for `range` or `allowed_values.`

- `range`: The range constraint applies to parameters of type number. It defines a lower and upper limit for the numeric value of the parameter. The syntax of the range constraint is

  ```
  range: { min: <lower limit>, max: <upper limit> }
  ```

  It is possible to define a range constraint with only a lower limit or an upper limit.

- `allowed_values`: The allowed_values constraint applies to parameters of type string or number. It specifies a set of possible values for a parameter. At deployment time, the user-provided value for the respective parameter must match one of the elements of the list. The syntax of the allowed_values constraint is

  ```
  allowed_values: [ <value>, <value>, ... ]
  ```

  Alternatively, the following YAML list notation can be used

  ```
  allowed_values:
    - <value>
    - <value>
    - ...
  ```

o Other <constraint type> are optional, they may be used (e.g., length, modulo, allowed_pattern, custom_constraint, allowed_values (for `string` types))

o Note that constrains must not be defined for any parameter enumerated in a nested heat template.

o Some ONAP parameters must never have constraints defined. See Section 5 for the use cases where these exceptions exist.

- immutable:
  o Defines whether the parameter is updatable. Stack update fails, if this is set to true and the parameter value is changed.

- This attribute is optional and defaults to false.

2.2.1.5 resources

This section is ONAP mandatory; it must be provided. This section contains the declaration of the single resources of the template. This section with at least one resource must be defined in the Heat Orchestration Template, or the template would not create any resources when being instantiated.

Each resource is defined as a separate block in the resources section with the following syntax.

```
resources:
  <resource ID>:
    type: <resource type>
    properties:
      <property name>: <property value>
    metadata:
      <resource specific metadata>
    depends_on: <resource ID or list of ID>
    update_policy: <update policy>
   deletion_policy: <deletion policy>
    external_id: <external resource ID>
    condition: <condition name or expression or boolean>
```

- resource ID
  - A resource ID that must be unique within the resources section of the Heat Orchestration Template.
  - ONAP requires that the resource ID must be unique across all Heat Orchestration Templates that compose the VNF. This requirement also applies when a VNF is composed of more than one Heat Orchestration Template (See Section 3.1).
  - The naming convention for a resource ID is provided in Section 5.3.
- type
  - The resource type, such as `OS::Nova::Server` or `OS::Neutron::Port`. Note that the type may specify a nested heat file. This attribute is required.
- properties
  - A list of resource-specific properties. The property value can be provided in place, or via a function (e.g., Intrinsic functions). This section is optional.
  - The naming convention for property parameters is provided in Section 5.
- metadata
  - Resource-specific metadata. This section is optional, except for the resource `OS::Nova::Server`. See Section 5.4.
- depends_on
  - Dependencies of the resource on one or more resources of the template. This attribute is optional. See Section 9.7 for additional details.
- update_policy

- Update policy for the resource, in the form of a nested dictionary. Whether update policies are supported and what the exact semantics are depends on the type of the current resource. This attribute is optional.

- deletion_policy
  - Deletion policy for the resource. The allowed deletion policies are `Delete`, `Retain`, and `Snapshot`. Beginning with `heat_template_version 2016-10-14`, the lowercase equivalents `delete`, `retain`, and `snapshot` are also allowed. This attribute is optional; the default policy is to delete the physical resource when deleting a resource from the stack.

- external_id
  - Allows for specifying the resource_id for an existing external (to the stack) resource. External resources cannot depend on other resources, but we allow other resources to depend on external resource. This attribute is optional. Note: when this is specified, properties will not be used for building the resource and the resource is not managed by Heat. This is not possible to update that attribute. Also, resource won't be deleted by heat when stack is deleted.

- condition
  - Condition for the resource. The condition decides whether to create the resource or not. This attribute is optional.

2.2.1.6 outputs

This ONAP optional section allows for specifying output parameters available to users once the template has been instantiated. If the section is specified, it will need to adhere to specific requirements. See Section 3.4 and Section 5.8 for additional details.

2.3 Environment File Format

The environment file is a yaml text file.
(https://docs.openstack.org/developer/heat/template_guide/environment.html)

The environment file can contain the following sections:

- `parameters:` A list of key/value pairs.
- `resource_registry:` Definition of custom resources.
- `parameter_defaults:` Default parameters passed to all template resources.
- `encrypted_parameters:` List of encrypted parameters.
- `event_sinks:` List of endpoints that would receive stack events.
- `parameter_merge_strategies:` Merge strategies for merging parameters and parameter defaults from the environment file.

Environment files for ONAP must contain the following sections:

- `parameters:`

Environment files for ONAP may contain the following sections:

- `resource_registry:`
- `parameter_defaults:`
- `encrypted_parameters:`
- `event_sinks:`
- `parameter_merge_strategies:`

The use of an environment file in OpenStack is optional. In ONAP, it is mandatory. A Heat Orchestration Template uploaded to ONAP must have a corresponding environment file, even if no parameters are enumerated in the mandatory parameter section.

(Note that ONAP, the open source version of ONAP, does not programmatically enforce the use of an environment file.)

2.3.1 SDC Treatment of Environment Files

Parameter values enumerated in the environment file are used by SDC as the default value. However, the SDC user may use the SDC GUI to overwrite the default values in the environment file.

SDC generates a new environment file for distribution to MSO based on the uploaded environment file and the user provided GUI updates. The user uploaded environment file is discarded when the new file is created.

ONAP has requirements for what parameters must be enumerated in the environment file and what parameter must not be enumerated in the environment file. See Section 3.4 and Section 5 for more details.

3. ONAP Heat Orchestration Templates: Overview

ONAP supports a modular Heat Orchestration Template design pattern, referred to as *VNF Modularity.*

3.1 ONAP VNF Modularity Overview

With VNF Modularity, a single VNF may be composed from one or more Heat Orchestration Templates, each of which represents a subset of the overall VNF. These component parts are referred to as "*VNF Modules*". During orchestration, these modules are deployed incrementally to create the complete VNF.

A modular Heat Orchestration Template can be either one of the following types:

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

A VNF must be composed of one "base" module and may be composed of zero to many "incremental" modules. The base module must be deployed first, prior to the incremental modules.

ONAP also supports the concept of an optional, independently deployed Cinder volume via a separate Heat Orchestration Templates, referred to as a Cinder Volume Module. This allows the volume to persist after a Virtual Machine (VM) (i.e., `OS::Nova::Server`) is deleted, allowing the volume to be reused on another instance (e.g., during a failover activity).

The scope of a Cinder volume module, when it exists, must be 1:1 with a Base Module or Incremental Module.

A Base Module must have a corresponding environment file.

An Incremental Module must have a corresponding environment file.

A Cinder Volume Module must have a corresponding environment file.

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.

3.2 Nested Heat Orchestration Templates Overview

ONAP supports nested Heat Orchestration Templates per OpenStack specifications.

A Base Module may utilize nested templates.

An Incremental Module may utilize nested templates.

A Cinder Volume Module may utilize nested templates.

A nested template must not define parameter constraints in the parameter definition section.

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 Heat Orchestration Template may then reference these nested templates either statically (by repeated definition) or dynamically (via `OS::Heat::ResourceGroup`).

See Section 9.1 for additional details.

3.3 ONAP Heat Orchestration Template Filenames

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

In the examples below, <text> represents any alphanumeric string that must not contain any special characters and must not contain the word “base”.

3.3.1 Base Modules

The file name for the base module must include “base” in the filename and must match one of the following options:

- base_<text>.y[a]ml
- <text>_base.y[a]ml
- base.y[a]ml
- <text>_base_<text>.y[a]ml

The base module’s corresponding environment file must be named identical to the base module with “.y[a]ml” replaced with “.env”.

3.3.2 Incremental Modules

There is no explicit naming convention for the incremental modules. As noted above, <text> represents any alphanumeric string that must not contain any special characters and must not contain the word “base”.

- <text>.y[a]ml

The incremental module’s corresponding environment file must be named identical to the incremental module with “.y[a]ml” replaced with “.env”.

To clearly identify the incremental module, it is recommended to use the following naming options for modules:

- module_<text>.y[a]ml
- <text>_module.y[a]ml
- module.y[a]ml

3.3.3 Cinder Volume Modules

The file name for the Cinder volume module must be named the same as the corresponding module it is supporting (base module or incremental module) with "_volume" appended

- <base module name>_volume.y[a]ml
- <incremental module name>_volume.y[a]ml

The volume module's corresponding environment file must be named identical to the volume module with ".y[a]ml" replaced with ".env".

3.3.4 Nested Heat file

There is no explicit naming convention for nested Heat files with the following exceptions; the name should contain "nest". As noted above, <text> represents any alphanumeric string that must not contain any special characters and must not contain the word "base".

- <text>.y[a]m

Nested Heat files do not have corresponding environment files, per OpenStack specifications. All parameter values associated with the nested heat file must be passed in as properties in the resource definition defined in the parent heat template.

3.4 ONAP Parameter Classifications Overview

In order for ONAP to support workflow automation, Heat Orchestration Template resource property parameters must adhere to specific naming conventions and requirements.

Broadly, ONAP categorizes parameters into four categories:

1. ONAP metadata parameters
2. Instance specific parameters
3. Constant parameters
4. Output parameters.

3.4.1 ONAP Metadata Parameters

There are both mandatory and optional ONAP metadata parameters associated with the resource OS::Nova::Server.

- ONAP metadata parameters must not have parameter constraints defined.
- Both mandatory and optional (if specified) ONAP metadata parameter names must follow the ONAP metadata parameter naming convention.

Section 5.5 provides more details on the metadata parameters.

3.4.2 Instance specific parameters

The instance specific parameters are VNF instance specific. The value of the parameter will be different for every instance of a VNF (e.g., IP address). The instance specific parameters are subdivided into two categories: **ONAP Orchestration Parameters** and **VNF Orchestration Parameters**

3.4.2.1 ONAP Orchestration Parameters

ONAP Orchestration Parameters are per instance parameters where the value is assigned via ONAP automation. (Note that in some cases, automation is currently not available and the value is loaded into ONAP prior to instantiation.)

- ONAP orchestration parameters must not be enumerated in the environment file.

- When the ONAP orchestration parameter type is set to `number`, the parameter must have constraints for `range` and/or `allowed_values`.
- Parameter constraints for ONAP orchestration parameters are optional for all parameter types other than `number`. If constraints are specified, they must adhere to the OpenStack specifications.
- The ONAP orchestration parameter names must follow the ONAP orchestration parameter naming convention. Section 5 provides additional details.

3.4.2.2 VNF Orchestration Parameters

VNF Orchestration Parameters are per instance parameters where the values are assigned manually. They are not supported by ONAP automation. The per instance values are loaded into ONAP prior to VNF instantiation.

- VNF orchestration parameters must not be enumerated in the environment file.
- When the VNF orchestration parameter type is set to `number`, the parameter must have constraints for `range` or `allowed_values`.
- Parameter constraints for VNF orchestration parameters are optional for all parameter types other than `number`. If constraints are specified, they must adhere to the OpenStack specifications.
- The VNF orchestration parameter names should follow the VNF orchestration parameter naming convention. Section 5 provides additional details.

3.4.3 Constant Parameters

The constant parameters are parameters that remain constant across many VNF instances (e.g., image, flavor). The constant parameters are subdivided into two categories: **ONAP Constant Parameters** and **VNF Constant Parameters.**

3.4.3.1 ONAP Constant Parameters

- ONAP Constant Parameters must be enumerated in the environment file. These parameter values are not assigned by ONAP.
- When the ONAP Constant Parameter type is set to `number`, the parameter must have constraints for `range` and/or `allowed_values`.
- Parameter constraints for ONAP constant parameters are optional for all parameter types other than `number`. If constraints are specified, they must adhere to the OpenStack specifications.
- The ONAP Constant Parameter names must follow the ONAP orchestration parameter naming convention. Section 5 provides additional details.

3.4.3.2 VNF Constant Parameters

- VNF Constant Parameters must be enumerated in the environment file. These parameter values are not assigned by ONAP.
- When the VNF Constant Parameters type is set to `number`, the parameter must have constraints for `range` and/or `allowed_values`.
- Parameter constraints for ONAP constant parameters are optional for all parameter types other than `number`. If constraints are specified, they must adhere to the OpenStack specifications.
- The VNF Constant Parameters names should follow the ONAP orchestration parameter naming convention. Section 5 provides additional details.

3.4.4 Output Parameters

The output parameters are parameters defined in the output section of a Heat Orchestration Template. The ONAP output parameters are subdivided into three categories:

1. ONAP Base Module Output Parameters
2. ONAP Volume Module Output Parameters
3. ONAP Predefined Output Parameters.

3.4.4.1 ONAP Base Module Output Parameters

ONAP Base Module Output Parameters are declared in the `outputs:` section of the base module Heat Orchestration Template. A Base Module Output Parameter is available as an input parameter (i.e., declared in the "parameters:" section) to all incremental modules in the VNF.

- A Base Module Output Parameter may be used as an input parameter in an incremental module.
- The Output parameter name and type must match the input parameter name and type unless the Output parameter is of the type `comma_delimited_list`.
    - If the Output parameter has a `comma_delimited_list` value (e.g., a collection of UUIDs from a Resource Group), then the corresponding input parameter must be declared as type `json` and not a `comma_delimited_list`, which is actually a `string` value with embedded commas.
- When a Base Module Output Parameter is declared as an input parameter in an incremental module Heat Orchestration Template, parameter constraints must not be declared.

Additional details on ONAP Base Module Output Parameters are provided in Sections 5.8 and Section 6.

3.4.4.2 ONAP Volume Module Output Parameters

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

- ONAP Volume Module Output Parameters are declared in the "outputs:" section of the Cinder volume module Heat Orchestration Template
- An ONAP Volume Module Output Parameter is available as an input parameter (i.e., declared in the `parameters:` section) only for the module (base or incremental) that the Cinder volume module is associated with. The Output parameter name and type must match the input parameter name and type unless the Output parameter is of the type `comma_delimited_list`.
- If the Output parameter has a `comma_delimited_list` value (e.g., a collection of UUIDs from a Resource Group), then the corresponding input parameter must be declared as type `json` and not a `comma_delimited_list`, which is actually a `string` value with embedded commas.
- When an ONAP Volume Module Output Parameter is declared as an input parameter in a base module or incremental module, parameter constraints must not be declared.

Additional details on ONAP Base Module Output Parameters are provided in Sections 5.8 and Section 7.

3.4.4.3 ONAP Predefined Output Parameters

ONAP will look for a small set of pre-defined Heat output parameters to capture resource attributes for inventory in ONAP. These output parameters are optional and are specified in Section 5.8.3.1.

3.5 Support of heat stack update

VNF Heat Orchestration Templates must not be designed to utilize the OpenStack `heat stack-update` command for scaling (growth/de-growth). ONAP does not support the use of `heat stack-update` command for scaling.

It is important to note that ONAP only supports `heat stack-update` for image upgrades.

4. Networking

ONAP defines two types of networks: External Networks and Internal Networks.

ONAP defines an external network in relation to the VNF and not with regard to the Network Cloud site. External networks may also be referred to as “inter-VNF” networks. An external network connects VMs in a VNF to

- VMs in another VNF or
- an external gateway or router

ONAP defines an internal network in relation to the VNF and not with regard to the Network Cloud site. Internal networks may also be referred to as “intra-VNF” networks or “private” networks. An internal network only connects VMs in a single VNF. It must not connect to other VNFs or an external gateway or router.

4.1 External Networks

VNF Heat Orchestration Templates must not include any resources for external networks connected to the VNF. External networks must be orchestrated separately, as independent, stand-alone VNF Heat Orchestration Templates, 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}`. The `{network-role}` should describe the network (e.g., oam). The `{network-role}` while unique to the LCP, can repeat across LCPs.

An External Network may be a Neutron Network or a Contrail Network

External networks must be passed into the VNF Heat Orchestration Templates as parameters.

- Neutron Network-id (UUID)
- Neutron Network subnet ID (UUID)
- Contrail Network Fully Qualified Domain Name (FQDN)

ONAP enforces a naming convention for parameters associated with external networks. Section 5 provides additional details.

Parameter values associated with an external network will be generated and/or assigned by ONAP at orchestration time. Parameter values associated with an external network must not be enumerated in the environment file. Section 5 provides additional details.

VNFs may use **Cloud assigned IP addresses** or **ONAP SDN-C assigned IP addresses** when attaching VMs to an external network

- A Cloud assigned IP address is assigned by OpenStack’s DHCP Service.
- An ONAP SDN-C assigned IP address is assigned by the ONAP SDN-C controller
- Note that Neutron Floating IPs must not be used. ONAP does not support Neutron Floating IPs (e.g., `OS::Neutron::FloatingIP`)
- ONAP supports the property `allowed_address_pairs` in the resource `OS::Neutron:Port` and the property `virtual_machine_interface_allowed_address_pairs` in `OS::ContrailV2::VirtualMachineInterfaces`. This allows the assignment of a virtual IP (VIP) address to a set of VMs.

VNF Heat Orchestration Templates must pass the appropriate external network IDs into nested VM templates when nested Heat is used.

4.2 Internal Networks

The VNF Heat Orchestration Templates must include the resource(s) to create the internal network. The internal network must be either a Neutron Network or a Contrail Network.

In the modular approach, internal networks must be created in the Base Module, with their resource IDs exposed as outputs (i.e., ONAP Base Module Output Parameters) for use by all incremental modules. If the Network resource ID is required in the base template, then a `get_resource` must be used.

When the internal network is created, it should be assigned a unique {network-role} in the context of the VNF. Section 5 provides additional details.

VNFs may use **Cloud assigned IP addresses** or **predetermined static IPs** when attaching VMs to an internal network.

- A Cloud assigned IP address is assigned by OpenStack's DHCP Service.
- A predetermined static IP address is enumerated in the Heat environment file. Since an internal network is local to the VNF, IP addresses can be re-used at every VNF instance.
- Note that Neutron Floating IPs must not be used. ONAP does not support Neutron Floating IPs (e.g., `OS::Neutron::FloatingIP`)
- ONAP supports the property allowed_address_pairs in the resource `OS::Neutron:Port` and the property `virtual_machine_interface_allowed_address_pairs` in `OS::ContrailV2::VirtualMachineInterfaces`. This allows the assignment of a virtual IP (VIP) address to a set of VMs.

ONAP does not programmatically enforce a naming convention for parameters for internal network. However, a naming convention is provided that must be followed. Section 5 provides additional details.

5. ONAP Resource ID and Parameter Naming Convention

This section provides the ONAP naming requirements for

1. Resource IDs
2. Resource Property Parameters

5.1 {vm-type}

The Heat Orchestration Templates for a VNF must assign a VNF unique `{vm-type}` for each Virtual Machine type (i.e., `OS::Nova::Server`) instantiated in the VNF. While the `{vm-type}` must be unique to the VNF, it does not have to be globally unique across all VNFs that ONAP supports.

Any parameter that is associated with a unique Virtual Machine type in the VNF must include `{vm-type}` as part of the parameter name.

Any resource ID that is associated with a unique Virtual Machine type in the VNF must include `{vm-type}` as part of the resource ID.

Note that `{vm-type}` must not be a substring of `{network-role}`. A substring of a string is another string that occurs "in". For example, "oam" is a substring of "oam_protected". It will cause the Pre-Amsterdam VNF Validation Program (i.e., ICE Project) process to produce erroneous error messages.

The {vm-type} should not contain the string "_int" or "int_" or "_int_". It may cause the Pre-Amsterdam VNF Validation Program (i.e., ICE Project) process to produce erroneous error messages.

The {vm-type} must be the same case in all parameter names in the VNF.

The {vm-type} must be the same case in all Resource IDs in the VNF.

It is recommended that the {vm-type} case in the parameter names matches the {vm-type} case in the Resource IDs.

There are two exceptions to the above rules:

1. The six ONAP Metadata parameters must not be prefixed with a common {vm-type} identifier. They are *vnf_name*, *vnf_id*, *vf_module_id*, *vf_module_name, vm_role*. The ONAP Metadata parameters are described in Section 5.5.
2. The parameter referring to the OS::Nova::Server property availability_zone must not be prefixed with a common {vm-type} identifier. availability_zone is described in Section 5.4.4.

5.2 {network-role}

The assignment of a {network-role} is discussed in Section 4.

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

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

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

Any resource ID that is associated with an internal network must include int_{network-role} as part of the resource ID.

Note that {network-role} must not be a substring of {vm-type}. A substring of a string is another string that occurs "in". For example, "oam" is a substring of "oam_protected". It will cause the Pre-Amsterdam VNF Validation Program (i.e., ICE Project) process to produce erroneous error messages.

The {network-role} should not contain the string "_int" or "int_" or "_int_". It may cause the Pre-Amsterdam VNF Validation Program (i.e., ICE Project) process to produce erroneous error messages.

The {network-role} must be the same case in all parameter names in the VNF.

The {network-role} must be the same case in all Resource IDs in the VNF.

It is recommended that the {network-role} case in the parameter names matches the {network-role} case in the Resource IDs.

5.3 Resource IDs

Heat Orchestration Template resources are described in Section 2.2.1.5

A resource ID that must be unique within the resources section of a Heat Orchestration Template. This is an OpenStack Requirement.

When a VNF is composed of more than one Heat Orchestration Template (i.e., modules), ONAP requires that the resource ID must be unique across all modules that compose the VNF.

When a resource is associated with a single {vm-type}, the resource ID must contain {vm-type}.

When a resource is associated with a single external network, the resource ID must contain {network-role}.

When a resource is associated with a single internal network, the resource ID must contain int_{network-role}.

When a resource is associated with a single {vm-type} and a single external network, the resource ID must contain both the {vm-type} and {network-role}.

- The {vm-type} must appear before the {network-role} and must be separated by an underscore (i.e., {vm-type}_{network-role}).
- Note that an {index} value may separate the {vm-type} and the {network-role}. An underscore will separate the three values (i.e., {vm-type}_{index}_{network-role}).

When a resource is associated with a single {vm-type} and a single internal network, the resource ID must contain both the {vm-type} and int_{network-role}.

- The {vm-type} must appear before the int_{network-role} and must be separated by an underscore (i.e., {vm-type}_int_{network-role}).
- Note that an {index} value may separate the {vm-type} and the int_{network-role}. An underscore will separate the three values (i.e., {vm-type}_{index}_int_{network-role}).

When a resource is associated with more than one {vm-type} and/or more than one network, the resource ID

- must not contain the {vm-type} and/or {network-role}/int_{network-role}
- should contain the term “shared” and/or contain text that identifies the VNF.

Only alphanumeric characters and “_” underscores must be used in the resource ID. Special characters must not be used.

All {index} values must be zero based. That is, the {index} must start at zero and increment by one.

The table below provides example OpenStack Heat resource ID for resources only associated with one {vm-type} and/or one network.

Resource Type	Resource ID Format
OS::Cinder::Volume	{vm_type}_volume_{index}
OS::Cinder::VolumeAttachment	{vm_type}_volumeattachment_{index}
OS::Heat::CloudConfig	{vm_type}_RCC
OS::Heat::MultipartMime	{vm_type}_RMM
OS::Heat::ResourceGroup	{vm_type}_RRG
OS::Heat::SoftwareConfig	{vm_type}_RSC
OS::Neutron::Port	{vm_type}_{index}_{network_role}_{index}_port
	{vm_type}_{index}_int_{network_role}_{index}_port
OS::Neutron::SecurityGroup	{vm_type}_RSG
OS::Neutron::Subnet	{network_role}_subnet_{index}

OS::Nova::Server	{vm_type}_{index}
OS::Nova::ServerGroup	{vm_type}_RSG
OS::Swift::Container	{vm_type}_RSwiftC

Table 1: Example OpenStack Heat Resource ID

The table below provides example Contrail Heat resource ID for resources only associated with one `{vm-type}` and/or one network.

Resource Type	Resource ID Format
OS::ContrailV2::InstanceIp	{vm_type}_{index}_{network_role}_RII
OS::ContrailV2::InterfaceRouteTable	{network_role}_RIRT
OS::ContrailV2::NetworkIpam	{network_role}_RNI
OS::ContrailV2::PortTuple	{vm_type}_RPT
OS::ContrailV2::ServiceHealthCheck	{vm_type}_RSHC_{LEFT|RIGHT}
OS::ContrailV2::ServiceTemplate	{vm_type}_RST_{index}
OS::ContrailV2::VirtualMachineInterface	int_{network_role}_RVMI
OS::ContrailV2::VirtualNetwork	int_{network_role}_RVN

Table 2: Example Contrail Heat resource ID

5.4 Resource: OS::Nova::Server - Parameters

The resource `OS::Nova::Server` manages the running virtual machine (VM) instance within an OpenStack cloud. (See https://docs.openstack.org/developer/heat/template_guide/openstack.html#OS::Nova::Server.)

Four properties of this resource must follow the ONAP parameter naming convention. The four properties are:

1. `image`
2. `flavor`
3. `name`
4. `availability_zone`

The table below provides a summary. The sections that follow provides additional details.

Note that the `{vm_type}` must be identical across all four property parameter for a given `OS::Nova::Server` resource.

Resource OS::Nova::Server				
Property Name	ONAP Parameter Name	Parameter Type	Parameter Value Generation	ONAP Parameter Classification
image	{vm-type}_image_name	string	Environment File	ONAP Constant

flavor	{vm-type}_flavor_name	string	Environment File	ONAP Constant
name	{vm-type}_name_{index}	string	ONAP	ONAP Orchestration
	{vm-type}_names	CDL	ONAP	ONAP Orchestration
availability_zone	availability_zone_{index}	string	ONAP	ONAP Orchestration

Table 3 Resource Property Parameter Names

5.4.1 Property: image

The parameter associated with the property `image` is an ONAP Constant parameter.

The parameters must be named `{vm-type}_image_name` in the Heat Orchestration Template.

The parameter must be declared as `type: string`

The parameter must be enumerated in the Heat Orchestration Template environment file.

Each VM type (i.e., `{vm-type}`) must have a separate parameter for image, even if more than one `{vm-type}` shares the same image. This provides maximum clarity and flexibility.

Example Parameter Definition

```
parameters:
      {vm-type}_image_name:
            type: string
            description: {vm-type} server image
```

5.4.2 Property: flavor

The parameter associated with the property `flavor` is an ONAP Constant parameter.

The parameters must be named `{vm-type}_flavor_name` in the Heat Orchestration Template.

The parameter must be declared as `type: string`

The parameter must be enumerated in the Heat Orchestration Template environment file.

Each VM type (i.e., {vm-type}) must have a separate parameter for flavors, even if more than one {vm-type} shares the same flavor. This provides maximum clarity and flexibility.

Example Parameter Definition

```
parameters:
      {vm-type}_flavor_name:
            type: string
            description: {vm-type} flavor
```

5.4.3 Property: Name

The parameter associated with the property `name` is an ONAP Orchestration parameter.

The parameter value is provided to the Heat template by ONAP. The parameter must not be enumerated in the environment file.

The parameter must be declared as `type: string` or `type: comma_delimited_list`

If the parameter is declared as `type:string`, the parameter must be named `{vm-type}_name_{index}`, where `{index}` is a numeric value that starts at zero and increments by one.

If the parameter is declared as `type:comma_delimited_list`, the parameter must be named as `{vm-type}_names`

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

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

Example: Parameter Definition

```
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: comma_delimited_list

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 }
                  ...
```

5.4.3.1 Contrail Issue with Values for OS::Nova::Server Property Name

The Contrail GUI has a limitation displaying special characters. The issue is documented in https://bugs.launchpad.net/juniperopenstack/+bug/1590710. It is recommended that special characters be avoided. However, if special characters must be used, the only special characters supported are:

```
- “ ! $ ‘ ( ) = ~ ^ | @ ` { } [ ] > , . _
```

5.4.4 Property: availability_zone

The parameter associated with the property `availability_zone` is an ONAP Orchestration parameter.

The parameter value is provided to the Heat template by ONAP. The parameter must not be enumerated in the environment file.

The parameter must be named `availability_zone_{index}` in the Heat Orchestration Template. The `{index}` must start at zero. The `{index}` must increment by one. The parameter name must not include the `{vm-type}`.

The parameter must be declared as `type: string`

The parameter must not be declared as `type: comma_delimited_list`

5.4.5 Example

The example below depicts part of a Heat Orchestration Template that uses the four `OS::Nova::Server` properties discussed in this section.

In the Heat Orchestration Template below, four Virtual Machines (`OS::Nova::Server`) are created: two dns servers with `{vm-type}` set to “dns” and two oam servers with `{vm-type}` set to “oam”. Note that the parameter associated with the property `name` is a `comma_delimited_list` for dns and a string for oam.

```
parameters:
  dns_image_name:
    type: string
    description: dns server image
  dns_flavor_name:
    type: string
    description: dns server flavor
  dns_names:
    type: comma_delimited_list
    description: dns server names
  oam_image_name:
    type: string
    description: oam server image
  oam_flavor_name:
    type: string
    description: oam server flavor
  oam_name_0:
    type: string
    description: oam server name 0
  oam_name_1:
    type: string
    description: oam server name 1
  availability_zone_0:
    type: string
    description: availability zone ID or Name
  availability_zone_1:
    type: string
    description: availability zone ID or Name


resources:
  dns_server_0:
    type: OS::Nova::Server
    properties:
      name: { get_param: [ dns_names, 0 ] }
      image: { get_param: dns_image_name }
      flavor: { get_param: dns_flavor_name }
      availability_zone: { get_param: availability_zone_0 }
      . . .
```

```
dns_server_1:
  type: OS::Nova::Server
  properties:
    name: { get_param: [ dns_names, 1 ] }
    image: { get_param: dns_image_name }
    flavor: { get_param: dns_flavor_name }
    availability_zone: { get_param: availability_zone_1 }
    . . .

oam_server_0:
  type: OS::Nova::Server
  properties:
    name: { get_param: oam_name_0 }
    image: { get_param: oam_image_name }
    flavor: { get_param: oam_flavor_name }
    availability_zone: { get_param: availability_zone_0 }
    . . .

oam_server_1:
  type: OS::Nova::Server
  properties:
    name: { get_param: oam_name_1 }
    image: { get_param: oam_image_name }
    flavor: { get_param: oam_flavor_name }
    availability_zone: { get_param: availability_zone_1 }
    . . .
```

5.5 Resource: OS::Nova::Server – Metadata Parameters

The resource OS::Nova::Server has an OpenStack optional property `metadata`. The `metadata` property is mandatory for ONAP Heat Orchestration Templates; it must be included.

ONAP requires the following three mandatory metadata parameters for an `OS::Nova::Server` resource:

- `vnf_id`
- `vf_module_id`
- `vnf_name`

ONAP allows the following three optional metadata parameters for an `OS::Nova::Server` resource. They may be included

- `vm_role`
- `vf_module_name`

Note that the metadata parameters do not and must not contain `{vm-type}` in their name.

When Metadata parameters are past into a nested heat template, the parameter names must not change.

The table below provides a summary. The sections that follow provides additional details.

Metadata Parameter Name	Parameter Type	Mandatory/Optional	Parameter Value Generation
vnf_id	string	Mandatory	ONAP
vf_module_id	string	Mandatory	ONAP
vnf_name	string	Mandatory	ONAP
vf_module_name	string	Optional	ONAP
vm_role	string	Optional	YAML or Environment File

Table 4: ONAP Metadata

5.5.1 vnf_id

The `vnf_id` parameter is mandatory; it must be included in the Heat Orchestration Template.

The `vnf_id` parameter value will be supplied by ONAP. ONAP generates the UUID that is the `vnf_id` and supplies it to the Heat Orchestration Template at orchestration time.

The parameter must be declared as `type: string`

Parameter constraints must not be defined.

The parameter must not be enumerated in the Heat environment file.

Example Parameter Definition

```
parameters:
      vnf_id:
            type: string
            description: Unique ID for this VNF instance
```

5.5.2 vf_module_id

The `vf_module_id` parameter is mandatory; it must be included in the Heat Orchestration Template.

The `vf_module_id` parameter value will be supplied by ONAP. ONAP generates the UUID that is the `vf_module_id` and supplies it to the Heat Orchestration Template at orchestration time.

The parameter must be declared as `type: string`

Parameter constraints must not be defined.

The parameter must not be enumerated in the Heat environment file.

Example Parameter Definition

```
parameters:
      vnf_module_id:
            type: string
            description: Unique ID for this VNF module instance
```

5.5.3 vnf_name

The vnf_name parameter is mandatory; it must be included in the Heat Orchestration Template.

The vnf_name parameter value will be generated and/or assigned by ONAP and supplied to the Heat Orchestration Template by ONAP at orchestration time.

The parameter must be declared as type: string

Parameter constraints must not be defined.

The parameter must not be enumerated in the Heat environment file.

Example Parameter Definition

```
parameters:
      vnf_name:
            type: string
            description: Unique name for this VNF instance
```

5.5.4 vf_module_name

The vf_module_name parameter is optional; it may be included in the Heat Orchestration Template.

The vf_module_name parameter 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.

The parameter must be declared as type: string

Parameter constraints must not be defined.

The parameter must not be enumerated in the Heat environment file.

Example Parameter Definition

```
parameters:
      vf_module_name:
            type: string
            description: Unique name for this VNF Module instance
```

5.5.5 vm_role

The vm_role parameter is optional; it may be included in the Heat Orchestration Template.

Any roles tagged to the VMs via metadata will be stored in ONAP's A&AI system and available for use by other ONAP components and/or north bound systems.

The vm_role values must be either

- hard-coded into the Heat Orchestration Template or
- enumerated in the environment file.

Defining the vm_role as the {vm-type} is a recommended convention

The parameter must be declared as `type: string`

Parameter constraints must not be defined.

Example Parameter Definition

```
parameters:
      vm_role:
            type: string
            description: Unique role for this VM
```

Example Resource Definition: Hard Coded

In this example, the `{vm-role}` is hard coded in the Heat Orchestration Template.

```
resources:
  dns_servers:
    type: OS::Nova::Server
    properties:
      . . . .
      metadata:
        vm_role: lb
```

Example Resource Definition: get_param

In this example, the `{vm-role}` is enumerated in the environment file.

```
resources:
  dns_servers:
    type: OS::Nova::Server
    properties:
      . . . .
      metadata:
        vm_role: { get_param: vm_role }
```

5.5.6 Example

The example below depicts part of a Heat Orchestration Template that uses the five of the `OS::Nova::Server` metadata parameter discussed in this section. The `{vm-type}` has been defined as `lb` for load balancer.

```
parameters:
   lb_name_0
      type: string
      description: VM Name for lb VM 0
   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
   vm_role:
      type: string
      description: Unique role for this VM

resources:


   lb_vm_0:
      type: OS::Nova::Server
      properties:
      name: { get_param: lb_name_0 }
      ...
      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 }
         vm_role: lb
```

5.6 Resource: OS::Neutron::Port - Parameters

The resource OS::Neutron::Port is for managing Neutron ports (See https://docs.openstack.org/developer/heat/template_guide/openstack.html#OS::Neutron::Port.)

5.6.1 Introduction

Four properties of the resource `OS::Neutron::Port` that must follow the ONAP parameter naming convention. The four properties are:

1. `network`
2. `fixed_ips, ip_address`
3. `fixed_ips, subnet_id`
4. `allowed_address_pairs, ip_address`

The parameters associated with these properties may reference an external network or internal network. External networks and internal networks are defined in Section 4.

5.6.1.1 External Networks

When the parameter references an external network

- the parameter name must contain `{network-role}`

- the parameter must not be enumerated in the Heat environment file
- the parameter is classified as an ONAP Orchestration Parameter

Property Name	ONAP Parameter Name	Parameter Type
network	{network-role}_net_id	string
	{network-role}_net_name	string
fixed_ips, ip_address	{vm-type}_{network-role}_ip_{index}	string
	{vm-type}_{network-role}_ips	comma_delimited_list
	{vm-type}_{network-role}_v6_ip_{index}	string
	{vm-type}_{network-role}_v6_ips	comma_delimited_list
fixed_ips, subnet	{network-role}_subnet_id	string
	{network-role}_v6_subnet_id	string
allowed_address_pairs, ip_address	{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	comma_delimited_list
	{vm-type}_{network-role}_v6_ip_{index}	string
	{vm-type}_{network-role}_v6_ips	comma_delimited_list

Table 5: OS::Neutron::Port Resource Property Parameters (External Networks)

5.6.1.2 Internal Networks

When the parameter references an internal network

- the parameter name must contain `int_{network-role}`
- the parameter may be enumerated in the environment file.

Property	Parameter Name for Internal Networks	Parameter Type
network	int_{network-role}_net_id	string
	int_{network-role}_net_name	string
fixed_ips, ip_address	{vm-type}_int_{network-role}_ip_{index}	string
	{vm-type}_int_{network-role}_ips	comma_delimited_list
	{vm-type}_int_{network-role}_v6_ip_{index}	string
	{vm-type}_int_{network-role}_v6_ips	comma_delimited_list
fixed_ips, subnet	int_{network-role}_subnet_id	string
	int_{network-role}_v6_subnet_id	string
allowed_address_pairs, ip_address	{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	comma_delimited_list
	{vm-type}_int_{network-role}_v6_ip_{index}	string
	{vm-type}_int_{network-role}_v6_ips	comma_delimited_list

Table 6: Port Resource Property Parameters (Internal Networks)

5.6.2 Property: network

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.

5.6.2.1 External Networks

When the parameter associated with the property `network` is referencing an "external" network, the parameter must adhere to the following naming convention in the Heat Orchestration Template

- `{network-role}_net_id` for the Neutron network ID
- `{network-role}_net_name` for the network name in OpenStack

The parameter must be declared as `type: string`

The parameter must not be enumerated in the Heat environment file.

Example Parameter Definition

```
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
```

Example: One Cloud Assigned IP Address (DHCP) assigned to a network that has only one subnet

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. The Cloud Assigned IP Address uses the OpenStack DHCP service to assign IP addresses.

```
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 }
```

5.6.2.2 Internal Networks

When the parameter associated with the property `network` 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 parameter must be declared as `type: string`

The assumption is that internal networks are created in the base module. The Neutron Network ID will be passed as an output parameter (e.g., ONAP Base Module Output Parameter) to the incremental modules. In the incremental modules, it will be defined as input parameter.

Example Parameter Definition

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

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

5.6.3 Property: fixed_ips, Map Property: subnet_id

The property `fixed_ips` is used to assign IPs to a port. The Map Property `subnet_id` specifies the subnet the IP is assigned from.

The property `fixed_ips` and Map Property `subnet_id` must be used if a Cloud (i.e., DHCP) IP address assignment is being requested and the Cloud IP address assignment is targeted at a specific subnet when two or more subnets exist.

The property `fixed_ips` and Map Property `subnet_id` should not be used if all IP assignments are fixed, or if the Cloud IP address assignment does not target a specific subnet or there is only one subnet.

Note that DHCP assignment of IP addresses is also referred to as cloud assigned IP addresses.

5.6.3.1 Subnet of an External Networks

When the parameter is referencing a subnet of an “external” network, the property `fixed_ips` and Map Property `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

The parameter must be declared as `type: string`

The parameter must not be enumerated in the Heat environment file.

Example Parameter Definition

```
parameters:
    {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: One Cloud Assigned IPv4 Address (DHCP) assigned to a network that has two or more subnets subnet:

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. The Cloud Assigned IP Address uses the OpenStack DHCP service to assign IP addresses.

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

   oam_subnet_id:
      type: string
      description: Neutron subnet UUID for the oam network

resources:
   lb_port_1:
      type: OS::Neutron::Port
      network: { get_param: oam_net_id }
      fixed_ips:
       - subnet_id: { get_param: oam_subnet_id }
```

Example: One Cloud Assigned IPv4 address and one Cloud Assigned IPv6 address assigned to a network that has at least one IPv4 subnet and one IPv6 subnet

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

   oam_subnet_id:
      type: string
      description: Neutron subnet UUID for the oam network
```

```
   oam_v6_subnet_id:
      type: string
      description: Neutron subnet UUID for the oam network

resources:
   lb_port_1:
      type: OS::Neutron::Port
      properties:
        network: { get_param: oam_net_id }
        fixed_ips:
         - subnet_id: { get_param: oam_subnet_id }
         - subnet_id: { get_param: oam_v6_subnet_id }
```

5.6.3.2 Internal Networks

When the parameter is referencing the subnet of an "internal" network, the property `fixed_ips` and Map Property `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

The parameter must be declared as `type: string`

The assumption is that internal networks are created in the base module. The Neutron subnet network ID will be passed as an output parameter (e.g., ONAP Base Module Output Parameter) to the incremental modules. In the incremental modules, it will be defined as input parameter.

Example Parameter Definition

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

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

5.6.4 Property: fixed_ips, Map Property: ip_address

The property `fixed_ips` is used to assign IPs to a port. The Map Property `ip_address` specifies the IP address to be assigned to the port.

The property `fixed_ips` and Map Property `ip_address` must be used when statically assigning one or more IP addresses to a port. This is also referred to as ONAP SDN-C IP address assignment. ONAP's SDN-C provides the IP address assignment.

An IP address is assigned to a port on a VM (referenced by {vm-type}) that is connected to an external network (referenced by {network-role}) or internal network (referenced by int_{network-role}).

When a SDN-C IP assignment is made to a port connected to an external network, the parameter name must contain {vm-type} and {network-role}.

When a SDN-C IP assignment is made to a port connected to an internal network, the parameter name must contain {vm-type} and int_{network-role}.

5.6.4.1 IP Address Assignments on External Networks

When the property fixed_ips and Map Property ip_address is used to assign IP addresses to an external network, the parameter name is dependent on the parameter type (comma_delimited_list or string) and IP address type (IPv4 or IPv6).

When the parameter for property fixed_ips and Map Property ip_address is declared type: comma_delimited_list, the parameter must adhere to the following naming convention

- {vm-type}_{network-role}_ips for IPv4 address
- {vm-type}_{network-role}_v6_ips for IPv6 address

Each element in the IP list should be assigned to successive instances of {vm-type} on {network-role}.

The parameter must not be enumerated in the Heat environment file.

Example Parameter Definition

```
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
```

Example: comma_delimited_list parameters for IPv4 and IPv6 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 IPv4 assignments for db VMs on the oam network

   db_oam_v6_ips:
      type: comma_delimited_list
      description: Fixed IPv6 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 ]}}, {
“ip_address”: {get_param: [ db_oam_v6_ips, 0 ]}}]

   db_1_port_1:
      type: OS::Neutron::Port
      properties:
        network: { get_param: oam_net_id }
        fixed_ips:
           - “ip_address”: {get_param: [ db_oam_ips, 1 ]}
           - “ip_address”: {get_param: [ db_oam_v6_ips, 1 ]}
```

When the parameter for property `fixed_ips` and Map Property `ip_address` is declared `type: string`, the parameter must adhere to the following naming convention.

- `{vm-type}_{network-role}_ip_{index}` for an IPv4 address
- `{vm-type}_{network-role}_v6_ip_{index}` for an IPv6 address

The value for `{index}` must start at zero (0) and increment by one.

The parameter must not be enumerated in the Heat environment file.

Example Parameter Definition

```
parameters:

   {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: string parameters for IPv4 and IPv6 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: Fixed IPv4 assignment for db VM 0 on the OAM network

   db_oam_ip_1:
      type: string
      description: Fixed IPv4 assignment for db VM 1 on the OAM network

   db_oam_v6_ip_0:
      type: string
      description: Fixed IPv6 assignment for db VM 0 on the OAM network

   db_oam_v6_ip_1:
      type: string
      description: Fixed IPv6 assignment for db VM 1 on the OAM network

resources:
   db_0_port_1:
      type: OS::Neutron::Port
      properties:
        network: { get_param: oam_net_id }
        fixed_ips: [ { "ip_address": {get_param: db_oam_ip_0}}, {
"ip_address": {get_param: db_oam_v6_ip_0 ]}}]

   db_1_port_1:
      type: OS::Neutron::Port
      properties:
        network: { get_param: oam_net_id }
        fixed_ips:
          - "ip_address": {get_param: db_oam_ip_1}}]
          - "ip_address": {get_param: db_oam_v6_ip_1}}]
```

5.6.4.2 IP Address Assignment on Internal Networks

When the property `fixed_ips` and Map Property `ip_address` is used to assign IP addresses to an internal network, the parameter name is dependent on the parameter type (`comma_delimited_list` or `string`) and IP address type (IPv4 or IPv6).

When the parameter for property `fixed_ips` and Map Property `ip_address` is declared `type: comma_delimited_list`, the parameter must adhere to the following naming convention

- `{vm-type}_int_{network-role}_ips` for IPv4 address
- `{vm-type}_int_{network-role}_v6_ips` for IPv6 address

Each element in the IP list should be assigned to successive instances of `{vm-type}` on `{network-role}`.

The parameter must be enumerated in the Heat environment file. Since an internal network is local to the VNF, IP addresses can be re-used at every VNF instance.

Example Parameter Definition

```
parameters:

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

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

Example: comma_delimited_list parameters for IPv4 and IPv6 Address Assignments to an internal network

In this example, the `{network-role}` has been defined as `oam_int` to represent an oam network internal to the vnf. The role `oam_int` was picked to differentiate from an external oam network with a `{network-role}` of `oam`. The `{vm-type}` has been defined as `db` for database.

```
parameters:
   int_oam_int_net_id:
      type: string
      description: Neutron UUID for the oam internal network

   db_int_oam_int_ips:
      type: comma_delimited_list
      description: Fixed IPv4 assignments for db VMs on the oam internal
network
```

```
   db_int_oam_int_v6_ips:
      type: comma_delimited_list
      description: Fixed IPv6 assignments for db VMs on the oam internal
network

resources:
   db_0_port_1:
      type: OS::Neutron::Port
      properties:
      network: { get_param: int_oam_int_net_id }
      fixed_ips: [ { “ip_address”: {get_param: [ db_int_oam_int_ips, 0 ]}}, {
“ip_address”: {get_param: [ db_int_oam_int_v6_ips, 0 ]}}]

   db_1_port_1:
      type: OS::Neutron::Port
      properties:
      network: { get_param: int_oam_int_net_id }
      fixed_ips:
         - “ip_address”: {get_param: [ db_int_oam_int_ips, 1 ]}
         - “ip_address”: {get_param: [ db_int_oam_int_v6_ips, 1 ]}
```

When the parameter for property `fixed_ips` and Map Property `ip_address` is declared `type: string`, the parameter must adhere to the following naming convention.

- `{vm-type}_int_{network-role}_ip_{index}` for an IPv4 address
- `{vm-type}_int_{network-role}_v6_ip_{index}` for an IPv6 address

The value for `{index}` must start at zero (0) and increment by one.

The parameter must be enumerated in the Heat environment file. Since an internal network is local to the VNF, IP addresses can be re-used at every VNF instance.

Example Parameter Definition

```
parameters:

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

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

Example: string parameters for IPv4 and IPv6 Address Assignments to an internal network

In this example, the {network-role} has been defined as oam_int to represent an oam network internal to the vnf. The role oam_int was picked to differentiate from an external oam network with a {network-role} of oam. The {vm-type} has been defined as db for database.

```
parameters:
   int_oam_int_net_id:
      type: string
      description: Neutron UUID for an OAM internal network

   db_oam_int_ip_0:
      type: string
      description: Fixed IPv4 assignment for db VM on the oam_int network

   db_oam_int_ip_1:
      type: string
      description: Fixed IPv4 assignment for db VM 1 on the oam_int network

   db_oam_int_v6_ip_0:
      type: string
      description: Fixed IPv6 assignment for db VM 0 on the oam_int network

   db_oam_int_v6_ip_1:
      type: string
      description: Fixed IPv6 assignment for db VM 1 on the oam_int network

resources:
   db_0_port_0:
      type: OS::Neutron::Port
      properties:
        network: { get_param: int_oam_int_net_id }
        fixed_ips: [ { “ip_address”: {get_param: db_oam_int_ip_0}}, {
“ip_address”: {get_param: db_oam_int_v6_ip_0 ]}}]

   db_1_port_0:
      type: OS::Neutron::Port
      properties:
        network: { get_param: int_oam_int_net_id }
        fixed_ips:
          - “ip_address”: {get_param: db_oam_int_ip_1}}]
          - “ip_address”: {get_param: db_oam_int_v6_ip_1}}]
```

5.6.5 Property: allowed_address_pairs, Map Property: ip_address

The property allowed_address_pairs in the resource OS::Neutron::Port allows the user to specify a mac_address and/or ip_address that will 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. The map property ip_address specifies the IP address.

The `allowed_address_pairs` is an optional property. It is not required.

An ONAP Heat Orchestration Template allows the assignment of one IPv4 address `allowed_address_pairs` and/or one IPv6 address to a `{vm-type}` and `{network-role}/int_{network-role}` combination.

An ONAP Heat Orchestration Template allows the assignment of one IPv6 address `allowed_address_pairs` and/or one IPv6 address to a `{vm-type}` and `{network-role}/int_{network-role}` combination.

Note that 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. ONAP does not support Neutron-style Floating IPs.

5.6.5.1 External Networks

When the parameter is referencing an "external" network, the property `allowed_address_pairs` and Map Property `ip_address` parameter must adhere to the following naming convention.

- `{vm-type}_{network-role}_floating_ip` for an IPv4 address
- `{vm-type}_{network-role}_floating_v6_ip` for an IPv6 address

The parameter must be declared as `type: string`

The parameter must not be enumerated in the Heat environment file.

Example Parameter Definition

```
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:

   oam_net_id:
      type: string
```

```
      description: Neutron UUID for the oam network

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

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

resources:
  db_0_port_0:
   type: OS::Neutron::Port
   properties:
     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
   properties:
     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}}]
```

5.6.5.2 Internal Networks

When the parameter is referencing an “internal” network, the property `allowed_address_pairs` and Map Property `ip_address` parameter must adhere to the following naming convention.

- `{vm-type}_int_{network-role}_floating_ip` for an IPv4 address
- `{vm-type}_int_{network-role}_floating_v6_ip` for an IPv6 address

The parameter must be declared as `type: string`

The parameter must be enumerated in the Heat environment file.

Example Parameter Definition

```
parameters:

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

   {vm-type}_int_{network-role}_floating_v6_ip:
      type: string
```

```
      description: VIP for {vm-type} VMs on the int_{network-role} network
```

5.6.5.3 Multiple allowed_address_pairs for a {vm-type} / {network-role} combination

The parameter `{vm-type}_{network-role}_floating_ip` provides only one allowed address pair IPv4 address per `{vm-type}` and `{network-role}` pair.

The parameter `{vm-type}_{network-role}_floating_v6_ip` provides only one allowed address pair IPv6 address per `{vm-type}` and `{network-role}` pair.

If there is a need for multiple allowed address pair IPs for a given `{vm-type}` and `{network-role}` combination within a VNF, then the parameter names defined for the property `fixed_ips` and Map Property `ip_address` should be used with the `allowed_address_pairs` property. The examples below illustrate this.

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

In this example, there are two administrative VM pairs. Each pair has one VIP. The `{network-role}` has been defined as `oam` to represent an oam network and the `{vm-type}` has been defined as `admin` for an administrative VM.

Pair 1: Resources `admin_0_port_0` and `admin_1_port_0` share a unique VIP, `[admin_oam_ips,2]`

Pair 2: Resources `admin_2_port_0` and `admin_3_port_0` share a unique VIP, `[admin_oam_ips,5]`

```
parameters:
   oam_net_id:
      type: string
      description: Neutron UUID for the oam network
   admin_oam_ips:
      type: comma_delimited_list
      description: Fixed IP assignments for admin VMs on the oam network

resources:

  admin_0_port_0:
    type: OS::Neutron::Port
    properties:
      network: { get_param: oam_net_id }
      fixed_ips: [ { “ip_address”: {get_param: [admin_oam_ips,0] }}]
      allowed_address_pairs: [{ “ip_address”: {get_param: [admin_oam_ips,2]
}}]

  admin_1_port_0:
    type: OS::Neutron::Port
    properties:
```

```
      network: { get_param: oam_net_id }
      fixed_ips: [ { “ip_address”: {get_param: [admin_oam_ips,1] }}]
      allowed_address_pairs: [{ “ip_address”: {get_param: [admin_oam_ips,2]
}}]

  admin_2_port_0:
    type: OS::Neutron::Port
    properties:
      network: { get_param: oam_net_id }
      fixed_ips: [ { “ip_address”: {get_param: [admin_oam_ips,3] }}]
      allowed_address_pairs: [{ “ip_address”: {get_param: [admin_oam_ips,5]
}}]

  admin_3_port_0:
    type: OS::Neutron::Port
    properties:
      network: { get_param: oam_net_id }
      fixed_ips: [ { “ip_address”: {get_param: [admin_oam_ips,4] }}]
      allowed_address_pairs: [{ “ip_address”: {get_param: [admin_oam_ips,5]
}}]
```

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

In this example, there is one load balancer pairs. The pair has two VIPs. The `{network-role}` has been defined as `oam` to represent an oam network and the `{vm-type}` has been defined as `lb` for a load balancer VM.

```
resources:
   lb_0_port_0:
      type: OS::Neutron::Port
      properties:
        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
      properties:
        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.

5.6.5.4 ONAP SDN-C Assignment of allowed_address_pair IP Addresses

The following items must be taken into consideration when designing Heat Orchestration Templates that expect ONAP's SDN-C to assign `allowed_address_pair` IP addresses via automation.

The VMs must be of the same `{vm-type}`.

The VMs must be created in the same module (base or incremental).

5.7 Resource Property "name"

The parameter naming convention of the property `name` for the resource `OS::Nova::Server` has been defined in Section 5.3.3.

This section provides the requirements how the property `name` for non `OS::Nova::Server` resources must be defined when the property is used. Not all resources require the property `name` (e.g., it is optional) and some resources do not support the property.

When the property `name` for a non `OS::Nova::Server` resources is defined in a Heat Orchestration Template, the intrinsic function `str_replace` must be used in conjunction with the ONAP supplied metadata parameter `vnf_name` to generate a unique value. This prevents the enumeration of a unique value for the property `name` in a per instance environment file.

Note that

- In most cases, only the use of the metadata value `vnf_name` is required to create a unique property `name`
- the Heat Orchestration Template pseudo parameter `'OS::stack_name'` may also be used in the `str_replace` construct to generate a unique name when the `vnf_name` does not provide uniqueness

Example: Property `name` *for resource* `OS::Neutron::SecurityGroup`

```
resources:
  DNS_SECURITY_GROUP:
    type: OS::Neutron::SecurityGroup
    properties:
      description: vDNS security group
      name:
        str_replace:
            template: VNF_NAME_sec_grp_DNS
            params:
                VNF_NAME:    {get_param: vnf_name}
      rules: [. . . . .
             ]
```

Example: Property name for resource `OS::Cinder::Volume`

```
resources:
```

```
DNS_Cinder_Volume:
  type: OS::Cinder::Volume
  properties:
    description: Cinder Volume
    name:
      str_replace:
          template: VNF_NAME_STACK_NAME_dns_volume
          params:
              VNF_NAME:    {get_param: vnf_name}
              STACK_NAME: { get_param: 'OS::stack_name' }
    . . . .
```

5.7.1 Contrail Issue with Values for the Property Name

The Contrail GUI has a limitation displaying special characters. The issue is documented in https://bugs.launchpad.net/juniperopenstack/+bug/1590710. It is recommended that special characters be avoided. However, if special characters must be used, note that for the following resources:

- Virtual Machine
- Virtual Network
- Port
- Security Group
- Policies
- IPAM Creation

the only special characters supported are:

```
- “ ! $ ‘ ( ) = ~ ^ | @ ` { } [ ] > , . _
```

5.8 ONAP Output Parameter Names

ONAP defines three types of Output Parameters as detailed in Section 3.4.4.

5.8.1 ONAP Base Module Output Parameters:

ONAP Base Module Output Parameters do not have an explicit naming convention. The parameter name must contain `{vm-type}` and `{network-role}` when appropriate.

5.8.2 ONAP Volume Template Output Parameters:

ONAP Base Module Output Parameters do not have an explicit naming convention. The parameter name must contain `{vm-type}` when appropriate.

5.8.3 Predefined Output Parameters

ONAP currently defines one predefined output parameter the OAM Management IP Addresses.

5.8.3.1 OAM Management IP Addresses

A VNF may 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 ONAP. The IP address 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 use of this output parameters are optional.
- The Management IP Address should be defined only once per VNF, so it must 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. In this case, a IPv4 and/or IPv6 parameter must be defined in the parameter section of the YAML Heat template. The parameter maybe named `oam_management_v4_address` and/or `oam_management_v6_address` or may be named differently.
- If the IP for the admin VM is obtained via DHCP, it may be obtained from the resource attributes. In this case, `oam_management_v4_address` and/or `oam_management_v6_address` must not be defined in the parameter section of the YAML Heat template.

Example: SDN-C Assigned IP Address echoed as `oam_management_v4_address`

```
parameters:
   admin_oam_ip_0:
      type: string
      description: Fixed IPv4 assignment for admin VM 0 on the OAM network
   . . .

resources:
  admin_oam_net_0_port:
    type: OS::Neutron::Port
    properties:
      name:
        str_replace:
          template: VNF_NAME_admin_oam_net_0_port
          params:
              VNF_NAME: {get_param: vnf_name}
      network: { get_param: oam_net_id }
      fixed_ips: [{ "ip_address": { get_param: admin_oam_ip_0 }}]
      security_groups: [{ get_param: security_group }]

  admin_server:
    type: OS::Nova::Server
    properties:
      name: { get_param: admin_names }
      image: { get_param: admin_image_name }
```

```
      flavor: { get_param: admin_flavor_name }
      availability_zone: { get_param: availability_zone_0 }
      networks:
        - port: { get_resource: admin_oam_net_0_port }
      metadata:
        vnf_id: { get_param: vnf_id }
        vf_module_id: { get_param: vf_module_id }
        vnf_name: {get_param: vnf_name }

Outputs:
    oam_management_v4_address:
   value: {get_param: admin_oam_ip_0 }
```

Example: Cloud Assigned IP Address output as `oam_management_v4_address`

```
parameters:
   . . .
resources:
  admin_oam_net_0_port:
    type: OS::Neutron::Port
    properties:
      name:
        str_replace:
          template: VNF_NAME_admin_oam_net_0_port
          params:
              VNF_NAME: {get_param: vnf_name}
      network: { get_param: oam_net_id }
      security_groups: [{ get_param: security_group }]

    admin_server:
    type: OS::Nova::Server
    properties:
      name: { get_param: admin_names }
      image: { get_param: admin_image_name }
      flavor: { get_param: admin_flavor_name }
      availability_zone: { get_param: availability_zone_0 }
      networks:
        - port: { get_resource: admin_oam_net_0_port }
      metadata:
        vnf_id: { get_param: vnf_id }
        vf_module_id: { get_param: vf_module_id }
        vnf_name: {get_param: vnf_name }

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

5.9 Contrail Resource Parameters

ONAP requires the parameter names of certain Contrail Resources to follow specific naming conventions. This section provides these requirements.

5.9.1 Contrail Network Parameters

Contrail based resources may require references to a Contrail network using the network FQDN.

5.9.1.1 External Networks

When the parameter associated with the Contrail Network is referencing an “external” network, the parameter must adhere to the following naming convention in the Heat Orchestration Template

- `{network-role}_net_fqdn`

The parameter must be declared as `type: string`

The parameter must not be enumerated in the Heat environment file.

Example: Parameter declaration

```
parameters:
   {network-role}_net_fqdn:
      type: string
      description: Contrail FQDN for the {network-role} network
```

Example: Contrail Resource OS::ContrailV2::VirtualMachineInterface Reference to a Network FQDN.

In this example, the `{network-role}` has been defined as `oam` to represent an oam network and the `{vm-type}` has been defined as `fw` for firewall. The Contrail resource `OS::ContrailV2::VirtualMachineInterface` property `virtual_network_refs` references a contrail network FQDN.

```
FW_OAM_VMI:
  type: OS::ContrailV2::VirtualMachineInterface
  properties:
    name:
      str_replace:
        template: VM_NAME_virtual_machine_interface_1
        params:
          VM_NAME: { get_param: fw_name_0 }
    virtual_machine_interface_properties:
      virtual_machine_interface_properties_service_interface_type: {
get_param: oam_protected_interface_type }
    virtual_network_refs:
      - get_param: oam_net_fqdn
    security_group_refs:
      - get_param: fw_sec_grp_id
```

5.9.2 Interface Route Table Prefixes for Contrail InterfaceRoute Table

The parameter associated with the resource `OS::ContrailV2::InterfaceRouteTable` property `interface_route_table_routes`, map property `interface_route_table_routes_route_prefix` is an ONAP Orchestration Parameter.

The parameters must be named `{vm-type}_{network-role}_route_prefixes` in the Heat Orchestration Template.

The parameter must be declared as `type: json`

The parameter supports IP addresses in the format:

1. Host IP Address (e.g., 10.10.10.10)
2. CIDR Notation format (e.g., 10.0.0.0/28)

The parameter must not be enumerated in the Heat environment file.

Example Parameter Definition

```
parameters:
   {vm-type}_{network-role}_route_prefixes:
      type: json
      description: JSON list of Contrail Interface Route Table route prefixes
```

Example:

```
parameters:
  vnf_name:
    type: string
    description: Unique name for this VF instance
  fw_int_fw_route_prefixes:
    type: json
    description: prefix for the ServiceInstance InterfaceRouteTable
  int_fw_dns_trusted_interface_type:
    type: string
    description: service_interface_type for ServiceInstance


<resource name>:
  type: OS::ContrailV2::InterfaceRouteTable
  depends_on: [resource name of OS::ContrailV2::ServiceInstance]
  properties:
    name:
      str_replace:
        template: VNF_NAME_interface_route_table
        params:
          VNF_NAME: { get_param: vnf_name }
    interface_route_table_routes:
      interface_route_table_routes_route: { get_param:
fw_int_fw_route_prefixes }
```

```
    service_instance_refs:
     - get_resource: < resource name of OS::ContrailV2::ServiceInstance >
    service_instance_refs_data:
     - service_instance_refs_data_interface_type: { get_param:
int_fw_interface_type }
```

5.10 Parameter Names in Contrail Resources

Contrail Heat resource properties will use, when appropriate, the same naming convention as OpenStack Heat resources. For example, the resource `OS::ContrailV2::InstanceIp` has two properties that the parameter naming convention is identical to properties in `OS::Neutron::Port`.

Example: Contrail Resource OS::ContrailV2::InstanceIp, Property instance_ip_address

The property `instance_ip_address` uses the same parameter naming convention as the property `fixed_ips` and Map Property `ip_address` in `OS::Neutron::Port`. The resource is assigning an ONAP SDN-C Assigned IP Address. The `{network-role}` has been defined as `oam_protected` to represent an oam protected network and the `{vm-type}` has been defined as `fw` for firewall.

```
  CMD_FW_OAM_PROTECTED_RII:
    type: OS::ContrailV2::InstanceIp
    depends_on:
      - FW_OAM_PROTECTED_RVMI
    properties:
      virtual_machine_interface_refs:
        - get_resource: FW_OAM_PROTECTED_RVMI
      virtual_network_refs:
        - get_param: oam_protected_net_fqdn
      instance_ip_address: { get_param: [fw_oam_protected_ips, get_param: index ] }
```

Example: Contrail Resource OS::ContrailV2::InstanceIp, Property subnet_uuid

The property `instance_ip_address` uses the same parameter naming convention as the property `fixed_ips` and Map Property `subnet_id` in `OS::Neutron::Port`. The resource is assigning a Cloud Assigned IP Address. The `{network-role}` has been defined as "`oam_protected`" to represent an oam protected network and the `{vm-type}` has been defined as "`fw`" for firewall.

```
  CMD_FW_SGI_PROTECTED_RII:
    type: OS::ContrailV2::InstanceIp
    depends_on:
      - FW_OAM_PROTECTED_RVMI
    properties:
      virtual_machine_interface_refs:
        - get_resource: FW_OAM_PROTECTED_RVMI
      virtual_network_refs:
        - get_param: oam_protected_net_fqdn
      subnet_uuid: { get_param: oam_protected_subnet_id  }
```

6. ONAP VNF Modularity

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

A modular Heat Orchestration Template can be either one of the following types:

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

A VNF must be composed of one "base" module and may be composed of zero to many "incremental" modules. The base module must be deployed first, prior to the incremental modules.

ONAP also supports the concept of an optional, independently deployed Cinder volume via a separate Heat Orchestration Templates, referred to as a Cinder Volume Module. This allows the volume to persist after a VM (i.e., `OS::Nova::Server`) is deleted, allowing the volume to be reused on another instance (e.g., during a failover activity).

The scope of a Cinder volume module, when it exists, must be 1:1 with a Base module or Incremental Module.

A Base Module must have a corresponding environment file.

An Incremental Module must have a corresponding environment file.

A Cinder Volume Module must have a corresponding environment file.

A VNF module (base, incremental, cinder) may support nested templates.

A shared Heat Orchestration Template resource must be defined in the base module. A shared resource is a resource that that will be referenced by another resource that is defined in the Base Module and/or one or more incremental modules.

When the shared resource needs to be referenced by a resource in an incremental module, the UUID of the shared resource must be exposed by declaring an ONAP Base Module Output Parameter.

Note that a Cinder volume is not a shared resource. A volume template must correspond 1:1 with a base module or incremental module.

An example of a shared resource is the resource `OS::Neutron::SecurityGroup`. Security groups are sets of IP filter rules that are applied to a VNF's networking. The resource `OS::Neutron::Port` has a property security_groups which provides the security groups associated with port. The value of parameter(s) associated with this property must be the UUIDs of the resource(s) `OS::Neutron::SecurityGroup.`

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

6.1 Suggested Patterns for Modular VNFs

There are numerous variations of VNF modularity. Below are two suggested usage patterns.

Option 1: Modules per VNFC type

a. Base module contains only the shared resources.
b. Group all VMs (e.g., VNFCs) of a given type (i.e. `{vm-type}`) into its own incremental module. That is, the VNF has an incremental module for each `{vm-type}`.
c. For a given `{vm-type}` incremental module, the VNF may have
    i. One incremental module used for both initial turn up and re-used for scaling. This approach is used when the number of VMs instantiated will be the same for initial deployment and scaling.
    ii. Two incremental modules, where one is used for initial turn up and one is used for scaling. This approach is used when the number of VMs instantiated will be different for initial deployment and scaling.

Option 2: Base VNF with Incremental Growth Modules

a. Base module contains a complete initial VNF instance
b. Incremental 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

With no growth units, Option 2 is equivalent to the “One Heat Template per VNF” model.

Note that modularization of VNFs is not required. A single Heat Orchestration Template (a base module) may still define a complete VNF, which might be appropriate for smaller VNFs that do not have any scaling options.

6.2 Modularity Rules

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., ONAP Base Module 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 incremental modules 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 appropriate Base 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 incremental Module 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 incremental) may have (optional) an associated Cinder Volume Module (*see Section 2.5*)
    a. Volume modules must correspond 1:1 with a base module or incremental module

b. A Cinder volume may be embedded within the base module or incremental module if persistence is not required

4. Shared resource UUIDs are passed between the base module and incremental modules via Heat Outputs Parameters (i.e., Base Module Output Parameters)
    a. The output parameter name in the base must match the parameter name in the add-on module

6.3 VNF Modularity Examples

Example: Base Module creates SecurityGroup

A VNF has a base module, named `base.yaml`, that defines a `OS::Neutron::SecurityGroup`. The security group will be referenced by an `OS::Neutron::Port` resource in an incremental module, named `INCREMENTAL_MODULE.yaml`. The base module defines a parameter in the out section named `dns_sec_grp_id`. `dns_sec_grp_id` is defined as a parameter in the incremental module. ONAP captures the UUID value of `dns_sec_grp_id` from the base module output statement and provides the value to the incremental module.

Note that the example below is not a complete Heat Orchestration Template. The `{network-role}` has been defined as `oam` to represent an oam network and the `{vm-type}` has been defined as `dns`.

`base_MODULE.yaml`

```
parameters:
  . . .


resources:
  DNS_SECURITY_GROUP:
    type: OS::Neutron::SecurityGroup
    properties:
      description: vDNS security group
      name:
        str_replace:
            template: VNF_NAME_sec_grp_DNS
            params:
                VMF_NAME: {get_param: vnf_name}
      rules: [. . . . .
             ]
  . . .


outputs:
  dns_sec_grp_id:
    description: UUID of DNS Resource SecurityGroup
    value: { get_resource: DNS_SECURITY_GROUP }
```

INCREMENTAL_MODULE.yaml

```
parameters:
  dns_sec_grp_id:
    type: string
    description: security group UUID
  . . .

resources:

  dns_oam_0_port:
    type: OS::Neutron::Port
    properties:
      name:
         str_replace:
           template: VNF_NAME_dns_oam_port
           params:
               VNF_NAME: {get_param: vnf_name}
      network: { get_param: oam_net_name }
      fixed_ips: [{ "ip_address": { get_param: dns_oam_ip_0 }}]
      security_groups: [{ get_param: dns_sec_grp_id }]
```

Examples: Base Module creates an internal network

A VNF has a base module, named `base_module.yaml,` that creates an internal network. An incremental module, named `incremental_module.yaml`, will create a VM that will connect to the internal network. The base module defines a parameter in the out section named `int_oam_net_id`. `int_oam_net_id` is defined as a parameter in the incremental module. ONAP captures the UUID value of `int_oam_net_id` from the base module output statement and provides the value to the incremental module.

Note that the example below is not a complete Heat Orchestration Template. 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.

base.yaml

```
heat_template_version: 2013-05-23

resources:
   int_oam_network:
      type: OS::Neutron::Network
      properties:
         name: {… }
         . . .

outputs:
```

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

incremental.yaml

```
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. Cinder Volume Templates

ONAP supports the independent deployment of a Cinder volume via separate Heat Orchestration Templates, the Cinder Volume module. 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 Base Module or Incremental Module may have a corresponding volume module. Use of separate volume modules is optional. A Cinder volume may be embedded within the Base Module or Incremental Module if persistence is not required.

If a VNF Base Module or Incremental Module has an independent volume module, the scope of volume templates must be 1:1 with Base module or Incremental 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 Orchestration Template from the Base Module or Incremental Module.
    - A single Cinder volume module must include all Cinder volumes needed by the Base/Incremental module.
    - The volume template must define “outputs” for each Cinder volume resource universally unique identifier (UUID) (i.e. ONAP Volume Template Output Parameters).
- The VNF Incremental Module or Base Module must define input parameters that match each Volume output parameter (i.e., ONAP Volume Template Output Parameters).
    - ONAP will supply the volume template outputs automatically to the bases/incremental template input parameters.
- Volume modules may utilize nested Heat templates.

Examples: Volume Template

A VNF has a Cinder volume module, named `incremental_volume.yaml`, that creates an independent Cinder volume for a VM in the module `incremental.yaml`. The `incremental_volume.yaml` defines a parameter in the output section, `lb_volume_id_0` which is the UUID of the cinder volume. `lb_volume_id_0` is defined as a parameter in `incremental.yaml`. ONAP captures the UUID value of `lb_volume_id_0` from the volume module output statement and provides the value to the incremental module.

Note that the example below is not a complete Heat Orchestration Template. The `{vm-type}` has been defined as “`lb`” for load balancer

`incremental_volume.yaml`

```
parameters:
   vnf_name:
      type: string
   lb_volume_size_0:
      type: number
   ...

resources:
   dns_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: dns_volume_size_0}
```

```
   ...

outputs:
    lb_volume_id_0:
       value: {get_resource: dns_volume_0}
   ...
```

incremental.yaml

```
parameters:
   lb_name_0:
      type: string
   lb_volume_id_0:
      type: string
   ...

resources:
  lb_0:
     type: OS::Nova::Server
     properties:
        name: {get_param: dns_name_0}
        networks:
        ...

  lb_0_volume_attach:
     type: OS::Cinder::VolumeAttachment
     properties:
        instance_uuid: { get_resource: lb_0 }
        volume_id: { get_param: lb_volume_id_0 }
```

8. ONAP Support of Environment Files

The use of an environment file in OpenStack is optional. In ONAP, it is mandatory. A Heat Orchestration Template uploaded to ONAP must have a corresponding environment file, even if no parameters are required to be enumerated.

(Note that ONAP, the open source version of ONAP, does not programmatically enforce the use of an environment file.)

A Base Module Heat Orchestration Template must have a corresponding environment file.

An Incremental Module Heat Orchestration Template must have a corresponding environment file.

A Cinder Volume Module Heat Orchestration Template must have a corresponding environment file.

A nested heat template must not have an environment file; OpenStack does not support it.

The environment file must contain parameter values for the ONAP Orchestration Constants and VNF Orchestration Constants. These parameters are identical across all instances of a VNF type, and expected to change infrequently. The ONAP 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 (ONAP Orchestration Parameters, VNF Orchestration Parameters). These parameters are supplied to the Heat by ONAP at orchestration time.

8.1 SDC Treatment of Environment Files

Parameter values enumerated in the environment file are used by SDC as the default value. However, the SDC user may use the SDC GUI to overwrite the default values in the environment file.

SDC generates a new environment file for distribution to MSO based on the uploaded environment file and the user provided GUI updates. The user uploaded environment file is discarded when the new file is created. Note that if the user did not change any values via GUI updates, the SDC generated environment file will contain the same values as the uploaded file.

8.2 Use of Environment Files when using OpenStack “heat stack-create” CLI

When ONAP is instantiating the Heat Orchestration Template, certain parameter must not be enumerated in the environment file. This document provides the details of what parameters should not be enumerated.

If the Heat Orchestration Template is to be instantiated from the OpenStack Command Line Interface (CLI) using the command “heat stack-create”, all parameters must be enumerated in the environment file.

9. Heat Template Constructs

9.1 Nested Heat Templates

ONAP 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 VNF module may then reference these component templates either statically by repeated definition or dynamically by using the resource `OS::Heat::ResourceGroup`.

9.1.1 Nested Heat Template Requirements

ONAP supports nested Heat Orchestration Templates. A Base Module, Incremental Module, and Cinder Volume Module may use nested heat.

A Heat Orchestration Template may reference the nested heat statically by repeated definition.

A Heat Orchestration Template may reference the nested heat dynamically using the resource `OS::Heat::ResourceGroup`.

A Heat Orchestration template must have no more than three levels of nesting. ONAP supports a maximum of three levels.

Nested heat templates must be referenced by file name. The use of `resource_registry` in the environment file is not supported and must not be used.

A nested heat yaml file must have a unique file names within the scope of the VNF

ONAP does not support a directory hierarchy for nested templates. All templates must be in a single, flat directory (per VNF)

A nested heat template may be used by any module within a given VNF.

Note that:

- Constrains must not be defined for any parameter enumerated in a nested heat template.
- All parameters defined in nested heat must be passed in as properties of the resource calling the nested yaml file.
- When `OS::Nova::Server` metadata parameters are past into a nested heat template, the parameter names must not change
- With nested templates, outputs are required to expose any resource properties of the child templates to the parent template. Those would not explicitly be declared as parameters but simply referenced as `get_attribute` targets against the "parent" resource.

9.1.1.1 Nested Heat Template Example: Static

`incremental.yaml`

```
Resources:
  dns_server_0:
    type: nested.yaml
    properties:
          dns_image_name: { get_param: dns_image_name }
          dns_flavor_name: { get_param: dns_flavor_name }
          availability_zone: { get_param: availability_zone_0 }
          security_group: { get_param: DNS_shared_sec_grp_id }
          oam_net_id: { get_param: oam_protected_net_id }
          dns_oam_ip: { get_param: dns_oam_ip_0 }
          dns_name: { get_param: dns_name_0 }
          vnf_name: { get_param: vnf_name }
          vnf_id: { get_param: vnf_id }
          vf_module_id: {get_param: vf_module_id}

  dns_server_1:
    type: nested.yaml
    properties:
          dns_image_name: { get_param: dns_image_name }
          dns_flavor_name: { get_param: dns_flavor_name }
          availability_zone: { get_param: availability_zone_1 }
          security_group: { get_param: DNS_shared_sec_grp_id }
```

```
          oam_net_id: { get_param: oam_protected_net_id }
          dns_oam_ip: { get_param: dns_oam_ip_1 }
          dns_name: { get_param: dns_name_1 }
          vnf_name: { get_param: vnf_name }
          vnf_id: { get_param: vnf_id }
          vf_module_id: {get_param: vf_module_id}
```

nested.yaml

```
  dns_oam_0_port:
    type: OS::Neutron::Port
    properties:
      name:
         str_replace:
           template: VNF_NAME_dns_oam_port
           params:
               VNF_NAME: {get_param: vnf_name}
      network: { get_param: oam_net_id }
      fixed_ips: [{ "ip_address": { get_param: dns_oam_ip }}]
      security_groups: [{ get_param: security_group }]

  dns_servers:
    type: OS::Nova::Server
    properties:
      name: { get_param: dns_names }
      image: { get_param: dns_image_name }
      flavor: { get_param: dns_flavor_name }
      availability_zone: { get_param: availability_zone }
      networks:
        - port: { get_resource: dns_oam_0_port }
      metadata:
        vnf_id: { get_param: vnf_id }
        vf_module_id: { get_param: vf_module_id }
        vnf_name  {get_param: vnf_name }
```

9.1.2 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 ONAP 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 ResourceGroup:

```
type: OS::Heat::ResourceGroup
  resource_def:
    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_def:
    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} ] }
```

9.1.2.1 ResourceGroup Property count

ONAP requires that the OS::Heat::ResourceGroup property count be defined (even if the value is one) and that the value must be enumerated in the environment file. This is required for ONAP to build the TOSCA model for the VNF.

```
type: OS::Heat::ResourceGroup
  properties:
  count: { get_param: count }
  index_var: index
    resource_def:
      type: my_nested_vm_template.yaml
      properties:
        names: {get_param: vm_name_list}
      index: index
```

9.1.2.2 Availability Zone and ResourceGroups

The resource `OS::Heat::ResourceGroup` and the property `availability_zone` has been an "issue" with a few VNFs since ONAP only supports `availability_zone` as a `string` parameter and not a `comma_delimited_list`. This makes it difficult to use a `ResourceGroup` to create Virtual Machines in more than one availability zone.

There are numerous solutions to this issue. Below are two suggested usage patterns.

Option 1: create a CDL in the `OS::Heat::ResourceGroup`. In the resource type: `OS::Heat::ResourceGroup`, create a `comma_delimited_list availability_zones` by using the intrinsic function `list_join`.

```
<resource name>:
    type: OS::Heat::ResourceGroup
     properties:
      count: { get_param: node_count }
      index_var: index
      resource_def:
        type: nested.yaml
        properties:
          index: index
          avaialability_zones: { list_join: [',', [ { get_param:
availability_zone_0 }, { get_param: availability_zone_1 } ] ] }
```

In the nested heat

```
parameters:
  avaialability_zones:
    type: comma_delimited_list
    description:

resources:
  servers:
    type: OS::Nova::Server
    properties:
      name: { get_param: [ dns_names, get_param: index ] }
      image: { get_param: dns_image_name }
      flavor: { get_param: dns_flavor_name }
      availability_zone: { get_param: [ avaialability_zones, get_param:
index ] }
```

Option 2: Create a resource group per availability zone. A separate `OS::Heat::ResourceGroup` is created for each availability zone.

9.2 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, ONAP *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.

9.3 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 ONAP 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.
- ONAP 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

9.4 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 ONAP 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 ONAP). 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 instance-specific parameters. ONAP 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
   lb_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
```

9.5 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 Orchestration 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}`).

9.6 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_resource: db_server_group}

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

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

9.7 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.

```
resources:
  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}
```

10. 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

11. Post Orchestration & VNF Configuration

Heat templates should contain a minimum amount of post-orchestration configuration data. For instance, *do not* embed complex user-data scripts in the template with large numbers of configuration parameters to the Heat template.

- VNFs may provide configuration APIs for use after VNF creation. Such APIs will be invoked via application and/or SDN controllers.

Note: It is important to follow this convention to the extent possible even in the short-term as of the long-term direction.

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 ***2*** for the relationship between VNFC and VNFs.

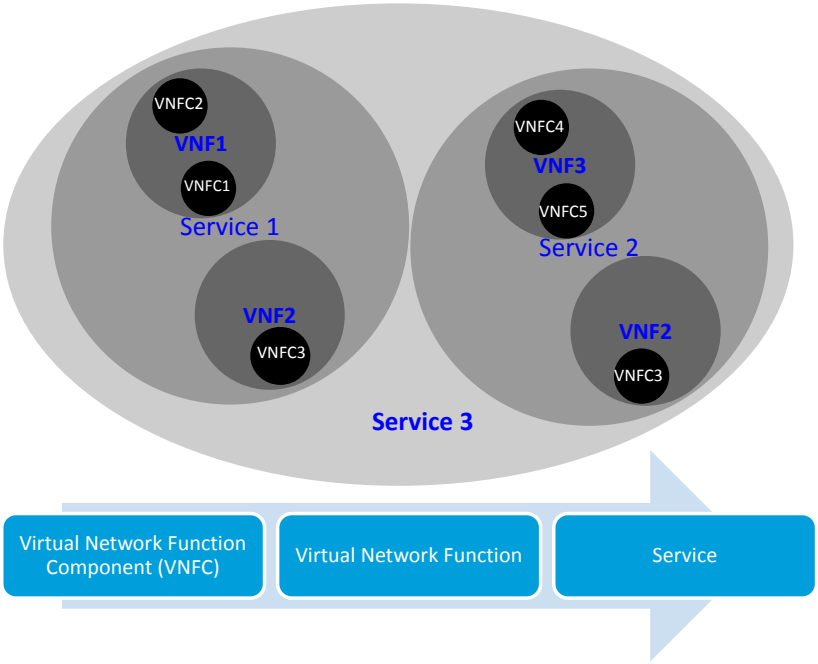

Figure 2. Virtual Function Entity Relationship