Heat Templates Guide for vMX

Anjali Kulkarni (anjali@juniper.net)
Pratik Maru (pmaru@juniper.net)

Introduction

This guide will demonstrate using heat templates to bring up vMX on an openstack system. vMX requires certain configuration and setup operations to be performed before using these heat templates. These are documented in the vMX Quick Start Guide, shown in references below. Ensure that you have read and configured your system according to the guide below.

This guide should be compatible with Openstack versions Liberty and upwards.
Junos vMX version for vRE – 15.1F6 onwards.

References

1. vMX Quick Start Guide

https://git.juniper.net/anjali/openstack-heat/blob/master/vMX-Openstack-QuickStartGuide.pdf

Openstack bundle

To download the scripts and heat templates, do:

```
git clone https://github.com/Juniper/vmx-heat-templates.git
cd vmx-heat-templates/openstack
```

The layout of this directory is as follows:

```
openstack> ls
1vmx.yaml
1vmx.env
vmx.yaml
scripts
vmx_contrail.yaml
vmx-topologies
vmx-components
kilo
```

See below for detailed explanation of the various directories.

vMX instance definition and vMX Topology yaml heat templates

There is a vMX instance definition heat template, which is vmx.yaml. You can define topologies using this vMX instance template. The topology heat template defined by the user could have multiple vMX instances, which could be inter-connected to each other, using bridges, as described in the “Topology Heat Templates” section.

vMX instance template

At the top level under openstack/ hierarchy, the 1vmx.yaml and 1vmx.env define the heat template and environment file to bring up a single vMX instance.
To bring up a single vMX instance:

- Modify the 1vmx.env file according to your environment variables. The 1vmx.env is shown below:

```
parameters:
    net_id1:

parameter_defaults:
    public_network:
```

```
    vfp_image:
    vfp_flavor:
    vcp_flavor:
    vcp_image:
    project_name:
    gateway_ip:
```

`net_id1`: is the network ID of a pre-created network, on which the wan port will exist.
`public_network`: ID of public network for management port
`vfp_image`: Glance image name of vFPC
`vfp_flavor`: Nova flavor name of vFPC
`vcp_flavor`: Glance image name of vRE
`vcp_image`: Nova flavor name of vFPC
`project_name`: Any project name. All resources will have this prefix
`gateway_ip`: IP of gateway.

The vfp_image, vfp_flavor, vcp_image and vcp_flavor are created using vmx scripts provided, as described in “vMX Installation Steps” section in Reference [1].

- Then, use the following heat stack-create command:
  `heat stack-create -f 1vmx.yaml -e 1vmx.env <vmx-name>`

This will bring up a single vMX instance, which has 1 wan port, 1 FPC.
To define a custom vMX instance, ie to change the number of ports or SRIOV/virtio port type, or FPC or dual-RE or other parameters of the vMX instance, we can modify the vmx.yaml file, which defines the vMX instance, and is encapsulated in the 1vmx.yaml file.

The vmx.yaml heat template defines :- vRE, vPFE, the bridge between vRE-vPFE, fabric bridge to connect multiple vPFE (if present), any wan bridges and ports. These components can be found under the openstack/vmx-components directory as follows:

```
openstack/vmx-components> ls
bridges  contrail  ports  vms
```

The vmx.yaml template with its components is shown below:

```
heat_template_version: 2015-10-15
parameters:
  net_id1: {description: ID of ge-network, type: string}
  stack_name: {description: name,  type: string}

resources:
  vmx_int_network:
    type: OS::Networking::VmxInternalNet
    properties:
      stack_name: {get_param: stack_name}

  re0:
    type: OS::Nova::VmxRe
    properties:
      re_pfe_network: {get_attr: [vmx_int_network,
re_pfe_network]}
      stack_name: {get_param: stack_name}

  fpc0:
    type: OS::Nova::VmxFpcSingle
    properties:
      id: 0
      re0_ip: {get_attr: [fpc0_fixed_net, external_ip]}
      all_ports: [{get_attr: [fpc0_fixed_net, external_port]},
                  {get_attr: [fpc0_fixed_net, internal_port]},
                  {get_attr: [fpc0_ge_port1, port]}]
      stack_name: {get_param: stack_name}

  fpc0_fixed_net:
    type: OS::Networking::VmxFpcFixedNet
    properties:
      re_pfe_network: {get_attr: [vmx_int_network,
re_pfe_network]}
      fabric_network: {get_attr: [vmx_int_network,
fabric_network]}
      id: 0
      internal_ip: 128.0.0.16
      stack_name: {get_param: stack_name}

  fpc0_ge_port1:
    type: OS::Networking::VmxPort
    properties:
      vnetwork_id: {get_param: net_id1}
      pname: fpc0_WAN_0
      stack_name: {get_param: stack_name}
```

Following resources form the components of a single vMX instance:

1. `OS::Networking::VmxInternalNet`

This resource defines the bridge connecting RE-PFE and the fabric bridge. It is located in the file .

2. `OS::Nova::VmxRe`
   This resource defines the vRE component of the vMX instance. It needs input as the re pfe network id, which is obtained via `{get_attr: [vmx_int_network, re_pfe_network]}` , and the stack_name.

3. A single FPC is defined by 2 resources:
   `A) OS::Nova::VmxFpcSingle`
   This resource defines the FPC of the vMX instance. It takes as input:
   a) id: The FPC id – ranges from 0 to 11.
   b) re0_ip: This is the IP address assigned to the PFE on management network.
   c) All_ports: This is the list of all ports for the FPC, which is the management (external) port, internal RE-FPC port and any wan (ge/xe) ports. Note, the external and internal port IDs are obtained from the `VmxFpcFixedNet` resource, which is specific to this FPC, so it must reference the name of this FPC's `VmxFpcFixedNet.` They are matched together via the "id" field, which should match for the FPC and VmxFpcFixedNet.
   d) Stack name

   `B) OS::Networking::VmxFpcFixedNet`
   This resource defines the ports of the FPC instance – the external port, internal RE-FPC port, and fabric port, if any. It takes as input:
   a) RE PFE network's id, obtained via {`get_attr: [vmx_int_network, re_pfe_network]}`
   b) Fabric network id, obtained via {`get_attr: [vmx_int_network, fabric_network]}`
   c) ID of the fpc
   d) Internal IP address of the FPC – this ranges from 128.0.0.16 to 128.0.0.27 for each of the 11 FPC
   e) Stack name

Note, we currently don't support multiple FPC, due to a bug in vMX code; thought heat templates are capable of supporting it.

Also, the above 2 resources may be later combined into a single resource for an FPC.

4. `OS::Networking::VmxPort`
   This resource defines the wan port of the FPC. It will appear on vmx either as ge-x/x/x or xe-0/0/0. It takes as input:
   a) `vnetwork_id:` Network ID of the network on which the wan port will be added. The network is assumed to be pre-created.
   b) `pname:` Name of the port
   c) Stack name

You can customize the vmx.yaml template by adding/deleting ports, change between virtio and sriov ports, add multiple FPC, add dual RE etc. as described below.

Add or delete a virtio port

In above vmx.yaml template, add the following entry of type **`VmxPort`** for fpc0_ge_port2, which denotes the second virtio port:

```
  fpc0_ge_port2:
    type: OS::Networking::VmxPort
    properties:
      vnetwork_id: {get_param: net_id2}
      pname: fpc0_WAN_1
      stack_name: {get_param: stack_name}
```

Make sure vnetwork_id takes net_id2 as a parameter, which is input from the .env file, and change pname to the name of the new port. Stack_name remains the same for all ports.

In addition, at the top of the heat template file, add an entry for net_id2 as shown below (the bold line is added new for this port):

```
heat_template_version: 2015-10-15
parameters:
  net_id1: {description: ID of ge-network, type: string}
  net_id2: {description: ID of ge-network, type: string}
```

Similarly, in the vmx.env add an entry for net_id2 under parameters: as shown below (in bold):

```
parameters:
    net_id1: <id of net1>
net_id2: <id of net2>
```

To delete a port, simply remove all above added lines.

Add or delete a SRIOV port

In above vmx.yaml template (assume it already has 2 virtio ports), add the following entry of type **`VmxSriovPort`** for fpc0_ge_port3, which denotes the sriov wan port:

```
  fpc0_ge_port3:
    type: OS::Networking::VmxSriovPort
    properties:
      vnetwork_id: {get_param: net_id3}
      pname: fpc0_WAN_2
      stack_name: {get_param: stack_name}
```

Make sure vnetwork_id takes net_id3 as a parameter, which is input from the .env file, and change pname to the name of the new port. Stack_name remains the same for all ports.

In addition, at the top of the heat template file, add an entry for net_id3 as shown below (the bold line is added new for this port):

```
heat_template_version: 2015-10-15
parameters:
  net_id1: {description: ID of ge-network, type: string}
  net_id2: {description: ID of ge-network, type: string}
  net_id3: {description: ID of ge-network, type: string}
```

Similarly, in the vmx.env add an entry for net_id3 under parameters: as shown below (in bold):

```
parameters:
    net_id1: <id of net1>
    net_id2: <id of net2>
    net_id3: <id of net3>
```

To delete a port, simply remove all above added lines.

Note, please make sure you have configured the items needed to use SRIOV as described in the Phase 2 Quick Start guide, before using SRIOV port.

Add or delete an FPC

Not supported yet due to a bug in vMX riot code.

Add dual RE

TBD

Topology heat templates

The custom vmx instance yaml and env file created above can be used along with a topology file, which defines multiple vmx instances and a way to interconnect them using pre-existing networks (which could be provider networks) or bridges/networks defined and created within the heat templates. The 1vmx.yaml under openstack/ is the most basic topology heat template, which defines a single vmx instance.

The vmx-topologies/osp-topologies directory shows some sample topologies, which can interconnect vmx instances. See below:

```
openstack/vmx-topologies/osp-topologies> ls
1vmx1net.env
1vmx1net.yaml
2vmx1net.env
2vmx1net.yaml
2vmx2net.yaml
```

The different topologies are described under “Topology:” headings below.

Topology: 1 vMX instance with a port on an OVS bridge with wan network in the heat template

In the above directory, the 1vmx1net.yaml defines a vMX instance along with a bridge network, as shown below:

```
heat_template_version: 2015-10-15
parameters:
  ge_cidr1: {default: 10.10.15.0/24, description: CIDR GE net,
type: string}
  n1: {description: name of vmx1, type: string}

resources:
  br1:
    type: OS::Networking::VmxNet
    properties:
      net_cidr: {get_param: ge_cidr1}
      bname: br1
      stack_name: {get_param: 'OS::stack_name'}

  vmx1:
    type: OS::Nova::Vmx
    properties:
      net_id1: {get_attr: [br1, bridge_network]}
      stack_name: {list_join: ['-', [{get_param:
'OS::stack_name'}, {get_param: n1}]]}
```

Above, 2 objects are defined which make a topology:

1. `OS::Networking::VmxNet`
   This defines an OVS bridge instance. It takes the cidr of the network as an input parameter (net_cidr), from corresponding env file, the name of the bridge (bname), and lastly the stack_name is always set to `get_param: 'OS::stack_name'`.
2. `OS::Nova::Vmx`
   This defines a vmx instance, which is encapsulated via vmx.yaml, which is described earlier. It takes input paramters as the network id of the bridge br1, and the stack name.
   Let's say you wanted to define another vmx instance with its own bridge in this topology file, you would then define the instance vmx2 as:

   ```
   vmx2:
       type: OS::Nova::Vmx
       properties:
         net_id1: {get_attr: [br2, bridge_network]}
         stack_name: {list_join: ['-', [{get_param:
   'OS::stack_name'}, {get_param: n2}]]}
   ```

   Above, the highlighted blue text shows the parameters to change from vmx1 instance – you would define net_id1 from

br2, and stack_name would have the name of vmx2, n2 (again input from env file)

The corresponding .env file for 1vmx1net.yaml looks as follows:

```
parameters:
    n1: vmx1

parameter_defaults:
    public_network:
     vfp_image:
     vfp_flavor:
     vcp_flavor:
    vcp_image:
    project_name:
    gateway_ip:

resource_registry:
      ....
```

Topology: 1 vMX instance with a port on an OVS bridge with existing wan network

In this topology, the wan network is already pre-created, and it's network id needs to be input in the .env file. The 1vmx.yaml file illustrates this topology:

```
heat_template_version: 2015-10-15
parameters:
  net_id1: {description: ID of ge-network, type: string}

resources:
  vmx1:
    type: OS::Nova::Vmx
    properties:
      net_id1: {get_param: net_id1}
      stack_name: {get_param: 'OS::stack_name'}
```

Its corresponding .env file is vmx.env and looks like as shown below. Enter the pre-existing network's network id in the net_id1 field below:

```
parameters:
    net_id1: <net-id>
```

```
parameter_defaults:
    public_network: <id of public network>
    vfp_image: <fpc-img>
    vfp_flavor: <pfe-flavor>
    vcp_flavor: <re-flvavor>
    vcp_image: <re-img>
    project_name: <project-name>
    gateway_ip: <gateway-ip>
```

Topology: 2 vMX instances, each with a wan port connected to each other via common OVS bridge

Let's say you wanted to connect 2 vMX instances via a private link on the same bridge. You will use the topology file 2vmx1net.yaml as shown below. It will define 2 vmx instances and share a bridge for the wan port on each vmx instance.

```
heat_template_version: 2015-10-15
parameters:
  ge_cidr1: {default: 10.10.15.0/24, description: CIDR GE net,
type: string}
  n1: {description: name of vmx1, type: string}
  n2: {description: name of vmx2, type: string}

resources:
  br1:
    type: OS::Networking::VmxNet
    properties:
      net_cidr: {get_param: ge_cidr1}
      bname: br1
      stack_name: {get_param: 'OS::stack_name'}

  vmx1:
    type: OS::Nova::Vmx
    properties:
      net_id1: {get_attr: [br1, bridge_network]}
      stack_name: {list_join: ['-', [{get_param:
'OS::stack_name'}, {get_param: n1}]]}

  vmx2:
    type: OS::Nova::Vmx
    properties:
      net_id1: {get_attr: [br1, bridge_network]}
       stack_name: {list_join: ['-', [{get_param:
      'OS::stack_name'}, {get_param: n2}]]}
```

Above, the OVS bridge is defined via `OS::Networking::VmxNet`, and 2 vMX instances via `OS::Nova::Vmx`.

The property `net_id1: {get_attr: [br1, bridge_network]}` defines the network id of the bridge, on which the wan port of that vMX instance is added. br1 is the name of the bridge, which is found under "`resources:`" section in above heat template.
The property `stack_name: {list_join: ['-', [{get_param: 'OS::stack_name'}, {get_param: n1}]]}` concatenates the stack name of the template, to the user provided name for vmx, input as "n1" from .env file. Thus, if the stack name while creating the stack was given as "vBNG", and the name n1 was input as vmx2, the stack_name passed to vMX instance will be "vBNG-vmx2". The stack name is concatenated to the vmx name to get a unique name for each vmx instance in the topology.
The corresponding .env file for the above topology is given below:

```
parameters:
    n1: vmx1
    n2: vmx2

parameter_defaults:
    public_network:
    vfp_image:
    vfp_flavor:
    vcp_flavor:
    vcp_image:
    project_name:
    gateway_ip:

resource_registry:
     …
```

Topology: 2 vMX instances, each with 2 wan ports connected to each other via 2 OVS bridges

This topology demonstrates 2 vMX instance, each with 2 wan ports, with one port of each instance connected by bridge br1 and second port of each vmx instance connected by a second bridge br2 – thus there are 2 private links/bridges between the 2 vMX.

```
heat_template_version: 2015-10-15
parameters:
```

```
  ge_cidr1: {default: 20.10.15.128/24, description: CIDR GE net,
type: string}
  ge_cidr2: {default: 30.10.15.128/24, description: CIDR GE net,
type: string}
  n1: {description: name of vmx1, type: string}
  n2: {description: name of vmx2, type: string}

resources:
  br1:
    type: OS::Networking::VmxNet
    properties:
      net_cidr: {get_param: ge_cidr1}
      bname: br1
      stack_name: {get_param: 'OS::stack_name'}

  br2:
    type: OS::Networking::VmxNet
    properties:
      net_cidr: {get_param: ge_cidr2}
      bname: br2
      stack_name: {get_param: 'OS::stack_name'}

  vmx1:    type: OS::Nova::Vmx
    properties:
      net_id1: {get_attr: [br1, bridge_network]}
      net_id2: {get_attr: [br2, bridge_network]}
      stack_name: {list_join: ['-', [{get_param:
'OS::stack_name'}, {get_param: n1}]]}

  vmx2:
    type: OS::Nova::Vmx
    properties:
      net_id1: {get_attr: [br1, bridge_network]}
      net_id2: {get_attr: [br2, bridge_network]}
      stack_name: {list_join: ['-', [{get_param:
'OS::stack_name'}, {get_param: n2}]]}
```

Above, 2 bridges are br1 and b2, and 2 vMX instances vmx1 and vmx2. Vmx1 and vmx2 each need net_id1 and net_id2, derived from bridges br1 and br2 as shown above, and their names n1 and n2 are input to list join of stack_name.

Corresponding .env file is shown below:

```
parameters:
    n1: vmx1
    n2: vmx2

parameter_defaults:
    public_network:
    vfp_image:
```

```
    vfp_flavor:
    vcp_flavor:
    vcp_image:
    project_name:
    gateway_ip:

resource_registry:
...
```