OpenStack Support for Different VIF Types

Issue

This document describes a proposal for OpenStack to support multiple types of virtual machine network interface (henceforth referred to as VIF) configurations to connect to the underlying network. The need for this proposal arises from the observation that,

1. Today OpenStack only supports the "bridge" type VIF configuration. However, going forward there will be a need to support multiple VIF types. In the context of using libvirt, the interface types of current interest are "bridge", "ethernet", "direct" (and other types could be added as well).
2. Each of these configuration types differ in their format and, the parameters and values which constitute that format. In certain cases, the nova-compute service might not possess all the data needed to populate the specific values in a particular configuration since those values refer to resources which are actually not controlled by nova-compute, but by a network service such as Quantum.

Proposal for Generating the VIF Configuration Format

The virtualization driver currently follows the method of using a template file containing a combination of all supported libvirt configurations for a VM. The virtualization driver, based on internal logic, or configuration flags, decides which relevant configuration items to choose from the template and constructs the consolidated VM configuration. The same approach can be used for supporting multiple VIF types. A flag, such as "libvirtVIFType", can be introduced in the nova.conf file. Each supported value for this flag will have a corresponding template configuration in the template file. Based on the value of the flag, the appropriate template configuration can be chosen.

Proposal for Populating the VIF Configuration with Appropriate Values

Let us take the specific example of 802.1Qbh, to understand this part of the proposal. Assuming that the earlier proposed method to use a flag in the nova.conf file is used, the following template configuration for 802.1Qbh would be loaded from the template file:

```
<interface type='direct'>
  <mac address=${mac_address}/>
  <source dev=${device_name} mode='private'/>
  <virtualport type='802.1Qbh'>
    <parameters profileid=${profile_name}/>
  </virtualport>
  <model type='virtio'/>
</interface>
```

In this template configuration, there are three place holders, of which the mac address value is generated by the nova-compute service and is available to the virtualization driver. However, the values for the "device_name" and "profile_name" place holders are not available to the nova-compute system since these values represent resources which are controlled by the network subsystem[1].

To obtain these values, the nova-compute system would need to communicate with the network service. A proposal for setting up this communication is by introducing the notion of a VIF driver. One or more VIF drivers will be available for every VIF type. Another flag can be defined in the nova.conf file, which specifies the name of the VIF driver. The virtualization driver would load the appropriate VIF driver class using this driver name.

The VIF driver would implement a standard python interface, such that it receives the information about that VM, and in turn populates the network dictionary with values corresponding to the relevant place holders:

```
def getVIFConfiguration(self, instance, network)
    …
    return network
```

where, "instance" is the instance dictionary, and "network" is the network dictionary (dictionaries referenced here are with reference to their implementation in libvirt_conn.py file in the Cactus release). The VIF driver, in the case of 802.1Qbh would obtain the necessary values by communicating with the network service.

Once the "network" dictionary is obtained by the virtualization driver from the VIF driver, the virtualization driver can iterate through the keys of the dictionary (which correspond to the place holder strings in the VIF template), and replace them with the values in the dictionary (this is the same approach being followed today for customizing the template configuration).

Specifically with respect to the 802.1Qbh example, the getVIFConfiguration() would return a "network" dictionary such as:
{'source_device': "eth5", 'profile_name': "enterprise_xyz_dept"}
and the virtualization driver would simply replace the occurrence of "source_device" and "profile_name" occurrences in the VIF template with values of "eth5", and "enterprise_xyz_dept", such that the final VIF configuration would look like:

```
<interface type='direct'>
  <mac address=${mac_address}/>
  <source dev='eth5' mode='private'/>
  <virtualport type='802.1Qbh'>
```

[1] In 802.1Qbh the "source_device" refers to a eth device on the host (just like a physical eth device on a NIC, but it's not a physical device), such that the VIF directly attaches to this device. The "profile_name" refers to the name of a port profile, which contains attributes that define the configuration of the "source_device".

```
      <parameters profileid='enterprise_xyz_dept'/>
    </virtualport>
    <model type='virtio'/>
  </interface>
```

The advantages of this approach are as follows:

a) Clean separation of VIF's network-specific details from the virtualization driver implementation. The virtualization driver does not need to know the specifics of a particular VIF type, including its configuration format, and how the configuration is populated.
b) The virtualization driver follows the same processing flow for all the VIF types, so there is no need for multiple forks in implementation logic (minimizing the chance of errors).
c) The VIF driver can be developed independently by the entities supporting the VIF types. However, the modular approach ensures that those entities do not have to touch the virtualization driver implementation. This reduces the risk of possibly breaking the virtualization driver on account of multiple entities modifying it.
d) The VIF driver is free to implement any communication mechanism and logic to obtain the information it needs to populate the VIF configuration.

Summary

The proposal outlined in this document is to:

a) Support multiple VIF types via configuration , and,

b) Introduce VIF drivers to populate the VIF configuration.