Apexlake Framework Documentation

Release draft (ff5cb95)

OPNFV

February 08, 2016

CONTENTS

CHAPTER

ONE

APEXLAKE INSTALLATION GUIDE

ApexLake is a framework that provides automatic execution of experiments and related data collection to enable a user validate infrastructure from the perspective of a Virtual Network Function (VNF). In the context of Yardstick, a virtual Traffic Classifier (vTC) network function is utilized.

1.1 Framework Hardware Dependencies

In order to run the framework there are some hardware related dependencies for ApexLake.

The framework needs to be installed on the same physical node where DPDK-pktgen is installed. The installation requires the physical node hosting the packet generator must have 2 NICs which are DPDK compatible.

The 2 NICs will be connected to the switch where the OpenStack VM network is managed.

The switch used must support multicast traffic and IGMP snooping. Further details about the configuration are provided at the following here.

The corresponding ports to which the cables are connected need to be configured as VLAN trunks using two of the VLAN IDs available for Neutron. Note the VLAN IDs used as they will be required in later configuration steps.

1.2 Framework Software Dependencies

Before starting the framework, a number of dependencies must first be installed. The following describes the set of instructions to be executed via the Linux shell in order to install and configure the required dependencies.

1. Install Dependencies.

To support the framework dependencies the following packages must be installed. The example provided is based on Ubuntu and needs to be executed in root mode.

```
apt-get install python-dev
apt-get install python-pip
apt-get install python-mock
apt-get install tcpreplay
apt-get install libpcap-dev
```

2. Install the Framework on the Target System.

After entering the Apexlake directory, run the following command.

```
python setup.py install
```

3. Source OpenStack openrc file.

```
source openrc
```

4. Create Two Networks based on VLANs in Neutron.

To enable network communications between the packet generator and the compute node, two networks must be created via Neutron and mapped to the VLAN IDs that were previously used in the configuration of the physical switch. The following shows the typical set of commands required to configure Neutron correctly.

```
VLAN_1=2025
VLAN_2=2021
neutron net-create apexlake_inbound_network \
        --provider:network_type vlan \
        --provider:segmentation_id $VLAN_1 \
        --provider:physical_network physnet1

neutron subnet-create apexlake_inbound_network \
        192.168.0.0/24 --name apexlake_inbound_subnet

neutron net-create apexlake_outbound_network \
        --provider:network_type vlan \
        --provider:physical_network physnet1

neutron net-create apexlake_inbound_network \
        --provider:network_type vlan \
        --provider:segmentation_id $VLAN_2 \
        --provider:physical_network physnet1

neutron subnet-create apexlake_outbound_network 192.168.1.0/24 \
        --name apexlake_outbound_subnet
```

5. Configure the Test Cases

The VLAN tags must also be included in the test case Yardstick yaml file as parameters for the following test cases:

- TC 006
- TC 007
- TC 020
- TC 021

1.2.1 Install and Configure DPDK Pktgen

Execution of the framework is based on DPDK Pktgen. If DPDK Pktgen has not installed, it is necessary to download, install, compile and configure it. The user can create a directory and download the dpdk packet generator source code:

```
cd experimental_framework/libraries
mkdir dpdk_pktgen
git clone https://github.com/pktgen/Pktgen-DPDK.git
```

For instructions on the installation and configuration of DPDK and DPDK Pktgen please follow the official DPDK Pktgen README file. Once the installation is completed, it is necessary to load the DPDK kernel driver, as follow:

```
insmod uio
insmod DPDK_DIR/x86_64-native-linuxapp-gcc/kmod/igb_uio.ko
```

It is necessary to set the configuration file to support the desired Pktgen configuration. A description of the required configuration parameters and supporting examples is provided in the following:

```
[PacketGen]
packet_generator = dpdk_pktgen

# This is the directory where the packet generator is installed
# (if the user previously installed dpdk-pktgen,
# it is required to provide the director where it is installed).
pktgen_directory = /home/user/software/dpdk_pktgen/dpdk/examples/pktgen/

# This is the directory where DPDK is installed
dpdk_directory = /home/user/apexlake/experimental_framework/libraries/Pktgen-DPDK/dpdk/

# Name of the dpdk-pktgen program that starts the packet generator
program_name = app/app/x86_64-native-linuxapp-gcc/pktgen

# DPDK coremask (see DPDK-Pktgen readme)
coremask = 1f

# DPDK memory channels (see DPDK-Pktgen readme)
memory_channels = 3

# Name of the interface of the pktgen to be used to send traffic (vlan_sender)
name_if_1 = p1p1

# Name of the interface of the pktgen to be used to receive traffic (vlan_receiver)
name_if_2 = p1p2

# PCI bus address correspondent to if_1
bus_slot_nic_1 = 01:00.0

# PCI bus address correspondent to if_2
bus_slot_nic_2 = 01:00.1
```

To find the parameters related to names of the NICs and the addresses of the PCI buses the user may find it useful to run the DPDK tool nic_bind as follows:

```
DPDK_DIR/tools/dpdk_nic_bind.py --status
```

Lists the NICs available on the system, and shows the available drivers and bus addresses for each interface. Please make sure to select NICs which are DPDK compatible.

1.2.2 Installation and Configuration of smcroute

The user is required to install smcroute which is used by the framework to support multicast communications. The following is the list of commands required to download and install smroute.

```
cd ~
git clone https://github.com/troglobit/smcroute.git
cd smcroute
sed -i 's/aclocal-1.11/aclocal/g' ./autogen.sh
sed -i 's/automake-1.11/automake/g' ./autogen.sh
./autogen.sh
./configure
make
sudo make install
cd ..
```

It is also requires the creation a configuration file using the following command:

SMCROUTE_NIC=(name of the nic)

where name of the nic is the name used previously for the variable “name_if_2”. For example:

```
SMCROUTE_NIC=p1p2
```

Then create the smcroute configuration file /etc/smcroute.conf

```
echo mgroup from $SMCROUTE_NIC group 224.192.16.1 > /etc/smcroute.conf
```

At the end of this procedure it will be necessary to perform the following actions to add the user to the sudoers:

```
adduser USERNAME sudo
echo "user ALL=(ALL) NOPASSWD: ALL" >> /etc/sudoers
```

1.2.3 Experiment using SR-IOV Configuration on the Compute Node

To enable SR-IOV interfaces on the physical NIC of the compute node, a compatible NIC is required. NIC configuration depends on model and vendor. After proper configuration to support SR-IOV, a proper configuration of OpenStack is required. For further information, please refer to the _SRIOV configuration guide

CHAPTER

TWO

APEXLAKE API INTERFACE DEFINITION

The API interface provided by the framework to enable the execution of test cases is defined as follows.

2.1 init

static init()

Initializes the Framework

Returns None

2.2 execute_framework

static execute_framework (test_cases,

iterations,

heat_template,

heat_template_parameters,

deployment_configuration,

openstack_credentials)

Executes the framework according the specified inputs

Parameters

- **test_cases**

  Test cases to be run with the workload (dict() of dict())

  Example: test_case = dict()

  test_case['name'] = 'module.Class'

  test_case['params'] = dict()

  test_case['params']['throughput'] = '1'

  test_case['params']['vlan_sender'] = '1000'

  test_case['params']['vlan_receiver'] = '1001'

  test_cases = [test_case]

- **iterations** Number of test cycles to be executed (int)

- **heat_template** (string) File name of the heat template corresponding to the workload to be deployed. It contains the parameters to be evaluated in the form of #parameter_name. (See heat_templates/vTC.yaml as example).
- **heat_template_parameters** (dict) Parameters to be provided as input to the heat template. See http://docs.openstack.org/developer/heat/ template_guide/hot_guide.html section "Template input parameters" for further info.
- **deployment_configuration** (dict[string] = list(strings))) Dictionary of parameters representing the deployment configuration of the workload.

  The key is a string corresponding to the name of the parameter, the value is a list of strings representing the value to be assumed by a specific param. The parameters are user defined: they have to correspond to the place holders (#parameter_name) specified in the heat template.

Returns dict() containing results