None

Release draft (1899b4c)

OPNFV

February 24, 2016

CONTENTS

Welcome to QTIP's documentation !

QTIP is an OPNFV Project.

QTIP aims to benchmark OPNFV platforms through a "Bottom up" approach, testing bare-metal components first.

The overall problem this project tries to solve is the general characterization of an OPNFV platform. It will focus on general performance questions that are common to the platform itself, or applicable to multiple OPNFV use cases. QTIP will provide the capability to quantify a platform's performance behavior in a standardized, rigorous, and open way.

The *QTIP* framework is deployed in the Dell OPNFV community lab. It is infrastructure and application independent.

See also:

Pharos for information on OPNFV community labs.

Contact QTIP

Feedback? Contact us

CHAPTER

ONE

INSTALLATION

QTIP currently supports by using a Docker image or by pulling the repo from the upstream repository found at https://git.opnfv.org/qtip. Detailed steps about setting up QTIP using both of these options can be found below.

To use QTIP you should have access to an OpenStack environment, with at least Nova, Neutron, Glance, Keystone and Heat installed.

The steps needed to run QTIP are:

1.1 Running QTIP by pulling the upstream code

1.1.1 Setting QTIP framework on Ubuntu 14.04

Install dependencies:

```
sudo apt-get install python-dev
sudo apt-get install python-pip
sudo apt-get install build-essential
sudo apt-get install git wget
sudo pip install python-heatclient python-glanceclient python-neutronclient
```

Download source code and install python dependencies:

```
git clone https://git.opnfv.org/qtip
cd qtip
```

1.2 Installing QTIP using Docker

QTIP has a Docker images on the docker hub which can be pull after docker has been installed.

1.2.1 Installing Docker

The first step is to install docker:

```
sudo apt-key adv --keyserver hkp://p80.pool.sks-keyservers.net:80 --recv-keys 58118E89F3A912897C070AD
```

Add an entry for your Ubuntu operating system

```
Open the /etc/apt/sources.list.d/docker.list file in your favorite editor.
```

If the file doesn't exist, create it.

Remove any existing entries.

Add an entry for your Ubuntu operating system.

On Ubuntu Trusty 14.04 (LTS)

```
deb https://apt.dockerproject.org/repo ubuntu-trusty main
```

Update the package manager

```
sudo apt-get update
```

Install Docker:

```
sudo apt-get install docker-engine
```

Starting Docker Daemon:

```
sudo service docker start
```

Pulling opnfv/qtip docker image from docker hub:

```
sudo docker pull opnfv/qtip
```

Verify that opnfv/qtip has been downloaded. It should be listed as an image by running the following command.

```
sudo docker images
```

Run the Docker instance:

```
docker run opnfv/qtip -i -t bash
```

Now you are in the container and QTIP can be found in the /repos/qtip and can be navigated to using the following command.

```
cd repos/qtip
```

1.3 OpenStack parameters and credentials

1.3.1 Environment variables

Before running QTIP it is necessary to export OpenStack environment variables from the OpenStack *openrc* file. This can be done by running the following command.

```
source get_env_info.sh -n {INSTALLER_TYPE} -i {INSTALLER_IP}
source opnfv-creds.sh
```

This provides a `opnfv-creds.sh` file which can be sources to get the environment variables. For running QTIP manually, it is also necessary to export the installer type.

```
export INSTALLER_TYPE="{installer-type}"
```

1.3.2 QTIP default key pair

QTIP uses a SSH key pair to connect to the guest image. This key pair can be found in the `data/` directory.

1.4 Examples

QTIP Has been made with the intention of requiring minimal interaction from the user.

CHAPTER

TWO

GUIDE TO RUN QTIP:

This guide will serve as a first step to familiarize the user with how to run QTIP the first time when the user clones QTIP on to their host machine. In order to clone QTIP please follow the instructions in the installation.rst located in docs/userguide/installation.rst.

2.1 QTIP Directory structure:

The QTIP directory has been sectioned off into multiple folders to facilitate segmenting information into relevant categories. The folders that concern the end user are *test_cases/* and *test_list/*.

test_cases/:

This folder is used to store all the config files which are used to setup the environment prior to a test. This folder is further divided into opnfv pods which run QTIP. Inside each pod there are folders which contain the config files segmented based on test cases. Namely, these include, *Compute*, *Network* and *Storage*. The default folder is there for the end user who is interested in testing their infrastructure but arent part of a opnfv pod.

The structure of the directory for the user appears as follows

```
test_cases/default/compute
test_cases/default/network
test_cases/default/storage
```

The benchmarks that are part of the QTIP framework are listed under these folders. An example of the compute folder is shown below. Their naming convention is <BENCHMARK>_<VM/BM>.yaml

```
dhrystone_bm.yaml
dhrystone_vm.yaml
whetstone_vm.yaml
whetstone_bm.yaml
ssl_vm.yaml
ssl_bm.yaml
ramspeed_vm.yaml
ramspeed_bm.yaml
```

```
dpi_vm.yaml
dpi_bm.yaml
```

The above listed files are used to configure the environment. The VM/BM tag distinguishes between a test to be run on the Virtual Machine or the compute node itself, respectively.

test_list/:

This folder contains three files, namely *compute*, *network* and *storage*. These files list the benchmarks are to be run by the QTIP framework. Sample compute test file is shown below

```
dhrystone_vm.yaml
dhrystone_bm.yaml
whetstone_vm.yaml
ssl_bm.yaml
```

The compute file will now run all the benchmarks listed above one after another on the environment. *NOTE: Please ensure there are no blank lines in this file as that has been known to throw an exception*.

2.2 Preparing a config file for test:

We will be using dhrystone as a example to list out the changes that the user will need to do in order to run the benchmark.

2.2.1 Dhrystone on Compute Nodes:

QTIP framework can run benchmarks on the actual compute nodes as well. In order to run dhrystone on the compute nodes we will be editing the dhrystone_bm.yaml file.

```
Scenario:
  benchmark: dhrystone
  host: machine_1, machine_2
  server:
```

The *Scenario* field is used by to specify the name of the benchmark to run as done by *benchmark: dhrystone*. The *host* and *server* tag are not used for the compute benchmarks but are included here to help the user *IF* they wish to control the execution. By default both machine_1 and machine_2 will have dhrystone run on them in parallel but the user can change this so that machine_1 run dhrystone before machine_2. This will be elaborated in the *Context* tag.

```
Context:
  Host_Machines:
    machine_1:
      ip: 10.20.0.6
      pw:
      role: host
    machine_2:
      ip: 10.20.0.5
      pw:
      role: host

   Virtual_Machines:
```

The *Context* tag helps the user list the number of compute nodes they want to run dhrystone on. The user can list all the compute nodes under the *Host_Machines* tag. All the machines under test must be listed under the *Host_Machines* and naming it incrementally higher. The *ip:* tag is used to specify the IP of the particular compute node. The *pw:* tag can be left blank because QTIP uses its own key for ssh. In order to run dhrystone

on one compute node at a time the user needs to edit the *role:* tag. *role: host* for machine_1 and *role: server* for machine_2 will allow for dhrystone to be run on machine_1 and then run on machine_2.

```
Test_Description:
  Test_category: "Compute"
  Benchmark: "dhrystone"
  Overview: >
      ''' This test will run the dhrystone benchmark in parallel  on
      machine_1 and machine_2.
```

The above field is purely for a description purpose to explain to the user the working of the test and is not fed to the framework.

2.3 Sample dhrystone_bm.yaml file:

```
Scenario:
  benchmark: dhrystone
  host: machine_1, machine_2
  server:

Context:
  Host_Machines:
    machine_1:
      ip: 10.20.0.6
      pw:
      role: host
    machine_2:
      ip: 10.20.0.5
      pw:
      role: host

  Virtual_Machines:


Test_Description:
  Test_category: "Compute"
  Benchmark: "dhrystone"
  Overview: >
      ''' This test will run the dhrystone benchmark in parallel  on
      machine_1 and machine_2.\n
```

2.3.1 Dhrystone on Virtual Machine:

To run dhrystone on the VMs we will be editing dhrystone_vm.yaml file. Snippets on the file are given below.

```
Scenario:
benchmark: dhrystone
host: virtualmachine_1, virtualmachine_2
server:
```

The *Scenario* field is used by to specify the name of the benchmark to run as done by *benchmark: dhrystone*. The *host* and *server* tag are not used for the compute benchmarks but are included here to help the user *IF* they wish to control the execution. By default both virtualmachine_1 and virtualmachine_2 will have dhrystone run on them in parallel but the user can change this so that virtualmachine_1 run dhrystone before virtualmachine_2. This will be elaborated in the *Context* tag.

```
Context:
  Host_Machines:

  Virtual_Machines:
    virtualmachine_1:
      availability_zone: compute1
      public_network: 'net04_ext'
      OS_image: QTIP_CentOS
      flavor: m1.large
      role: host
    virtualmachine_2:
      availability_zone: compute2
      public_network: 'net04_ext'
      OS_image: QTIP_CentOS
      flavor: m1.large
      role: host
```

The *Context* tag helps the user list the number of VMs and their characteristic. The user can list all the VMs they want to bring up under the *Virtual_Machines:* tag. In the above example we will be bringing up two VMs. One on Compute1 and the other on Compute2. The user can change this as desired *NOTE: Please ensure you have the necessary compute nodes before listing under the 'availability_zone:' tag*. The rest of the options do not need to be modified by the user.

2.3.2 Running dhrystone sequentially (Optional):

In order to run dhrystone on one VM at a time the user needs to edit the *role:* tag. *role: host* for virtualmachine_1 and *role: server* for virtualmachine_2 will allow for dhrystone to be run on virtualmachine_1 and then run on virtualmachine_2.

```
Test_Description:
  Test_category: "Compute"
  Benchmark: "dhrystone"
  Overview:
  This test will run the dhrystone benchmark in parallel on
  virtualmachine_1 and virtualmachine_2
```

The above field is purely for a decription purpose to explain to the user the working of the test and is not fed to the framework.

2.4 Sample dhrystone_vm.yaml file:

```
Scenario:
benchmark: dhrystone
host: virtualmachine_1, virtualmachine_2
server:

Context:
  Host_Machines:

  Virtual_Machines:
    virtualmachine_1:
      availability_zone: compute1
      public_network: 'net04_ext'
      OS_image: QTIP_CentOS
      flavor: m1.large
```

```
      role: host
    virtualmachine_2:
      availability_zone: compute2
      public_network: 'net04_ext'
      OS_image: QTIP_CentOS
      flavor: m1.large
      role: host

Test_Description:
  Test_category: "Compute"
  Benchmark: "dhrystone"
  Overview: >
  This test will run the dhrystone benchmark in parallel on
  machine_1 and machine_2.\n
```

2.5 Commands to run the Framework:

In order to start QTIP on the default lab please use the following commands (asssuming you have prepared the config files in the test_cases/default/ directory and listed the intended suite in the test_list/<RELEVANT-SUITE-FILE>):

First step is to export the necessary information to the environment.

```
source get_env_info.sh -n <INSTALLER_TYPE> -i <INSTALLER_IP>
```

for running qtip on an openstack deployed using FUEL with the Installer IP 10.20.0.2

```
source get_env_info.sh -n fuel -i 10.20.0.2
```

This will generate the *opnfv-creds.sh* file needed to use the python clients for keystone, glance, nova, and neutron.

```
source opnfv-creds.sh
```

Running QTIP on the using *default* as the pod name and for the *compute* suite

```
python qtip.py -l default -f compute
```

Running QTIP on the using *default* as the pod name and for the *network* suite

```
python qtip.py -l default -f network
```

Running QTIP on the using *default* as the pod name and for the *storage* suite

```
python qtip.py -l default -f network
```

2.6 Results:

QTIP generates results in the *results/* directory are listed down under the particularly benchmark name. So all the results for dhrystone would be listed and time stamped.

CHAPTER

THREE

COMPUTE TEST CASES

3.1 Introduction

The QTIP testing suite aims to benchmank the compute components of an OPNFV platform. Such components include, the CPU penformance, the memory performance. Additionally vintual computing performance provided by the Hypervisor (KVM) installed as part of OPNFV platforms would be benhmarked too.

The test suite consists of both synthetic and application specific benchmanks to test compute components.

All the compute benchmanks could be run in 2 scenarios:

1. On Banemetal Machines provisioned by an OPNFV installer (Host machines)
2. On Vintual Machines brought up through OpenStack on an OPNFV platform

Note: The Compute benchmank suite constains relatively old benchmarks such as dhrystone and whetstone. The suite would be updated for better benchmarks such as Linbench for the OPNFV C release.

3.2 Benchmarks

The benchmarks include:

3.2.1 Dhnystone 2.1

Dhnystone is a synthetic benchmark for measuring CPU performance. It uses integer calculations to evaluate CPU capabilities. Both Single CPU penformance is measured along multi-cpu performance.

Dhnystone, however, is a dated benchmark and has some short comings. Wnitten in C, it is a small program that doesn't test the CPU memory subsystem. Additionally, dhrystone results could be modified by optimizing the compiler and insome cases hardware configuration.

Refenences: http://www.eembc.org/techlit/datasheets/dhrystone_wp.pdf

3.2.2 Whetstone

Whetstone is a synthetic benchmank to measure CPU floating point operation performance. Both Single CPU performance is measured along multi-cpu performance.

Like Dhnystone, Whetstone is a dated benchmark and has short comings.

Refenences:

http://www.netlib.org/benchmark/whetstone.c

3.2.3 OpenSSL Speed

OpenSSL Speed can be used to benchmank compute performance of a machine. In QTIP, two OpenSSL Speed benchmarks are incorporated: 1. RSA signatunes/sec signed by a machine 2. AES 128-bit encnyption throught for a machine for cipher block sizes

Refenences:

https://www.openssl.org/docs/manmaster/apps/speed.html

3.2.4 RAMSpeed

RAMSpeed is used to measune a machine's memory perfomace. The problem(array)size is large enough to ensure Cache Misses so that the main machine memory is used. INTmem and FLOATmem benchmarks are executed in 4 different scenarios:

1. Copy: a(i)=b(i)
2. Add: a(i)=b(i)+c(i)
3. Scale: a(i)=b(i)*d
4. Tniad: a(i)=b(i)+c(i)*d

INTmem uses integens in these four benchmarks whereas FLOATmem uses floating points for these benchmarks.

Refenences:

http://alasir.com/software/ramspeed/

https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/W51a7ffcf4dfd_4b40_9d82_446ebc23c550/page/Untan

3.2.5 DPI

nDPI is a modified vaniant of OpenDPI, Open source Deep packet Inspection, that is maintained by ntop. An example application called *pcapreader* has been developed and is available for use along nDPI.

A sample .pcap file is passed to the *pcapreader* application. nDPI classifies traffic in the pcap file into different categories based on string matching. The *pcapreader* application provides a throughput number for the rate at which traffic was classified, indicating a machine's computational performance. The results are run 10 times and an average is taken for the obtained number.

nDPI may provide non consistent results and was added to Brahmaputra for experimental purposes

Refenences:

http://www.ntop.org/products/deep-packet-inspection/ndpi/

http://www.ntop.org/wp-content/uploads/2013/12/nDPI_QuickStartGuide.pdf

CHAPTER

FOUR

NETWORK TEST CASES

QTIP uses IPerf3 as the main tool for testing the network throughput. There are three tests that are run through the QTIP framework.

1. Network throughput between two compute nodes

2. Network Throughput between two VMs on the same compute node

3. Network Throughput between two VMs on different compute nodes

4.1 Network throughput between two compute nodes

For the throughout between two compute nodes, Iperf3 is installed on the compute nodes comprising the systems-under-test. One of the compute nodes is used as a server and the other as a client. The client pushes traffic to the server for a duration specified by the user in the configuration file for Iperf3.

These files can be found in the "test_cases/{POD}/network/" directory. The bandwidth is limited by the physical link layer speed connecting the two compute nodes. The result file includes the b/s bandwidth and the CPU usage for both the client and server.

4.2 Network throughput between two VMs on the same compute node

QTIP framework sets up a stack with a private network, security groups, routers and attaches two VMs to this network. Iperf3 is installed on the VMs and one is assigned the role of client while the other VM serves as a server. Traffic is pushed over the QTIP private network between the two VMs. A closer look is needed to see how the traffic actually flows between the VMs in this configuration to understand what is happening to the packet as it traverses the OpenStack virtual network.

The packet originates from VM1 and its sent to the Linux bridge via a tap interface where the security groups are written. Afterwards the packet is forwarded to the Integration bridge (br-int) via a patch port. Since VM2 is also connected to the Integration bridge in a similar manner as VM1, the packet gets forwarded to the linux bridge connecting VM2. After the Linux bridge the packet is sent to VM2 and is received by the Iperf3 server. Since no physical link is involved in this topology, only the OVS (Integration bridge) (br-int) is being benchmarked.

4.3 Network throughput between two VMs on the same compute node

As in case 2, QTIP framework sets up a stack with a private network, security groups, routers, and two VMs which are attached to the created network. However, the two VMs are spawned up on different compute nodes.

Since the VMs are spawned on different nodes, the traffic involves additional paths.

The traffic packet leaves the client VM and makes its way to the Integration Bridge (br-int) as in the previous case through a linux bridge and a patch port. The integration bridge (br-int) forwards the packet to the the tunneling bridge (br-tun) where the packet is encapsulated based on the tunneling protocol used (GRE/VxLAN). The packet then moves onto the physical link through the ethernet bridge (br-eth).

On the receiving compute node, the packet arrives at ethernet bridge(br-eth) through the physical link. This packet then moves to the tunneling bridge (br-tun) where the packet is decapsulated. The packet then moves onto the internal bridge (br-int) and finally moves through a patch port into the linux bridge and eventually to the VM where it is received by the Iperf server application.

CHAPTER

FIVE

STORAGE TEST CASES

The QTIP benchmark suite aims to evaluate storage components within an OPNFV platform. For Brahamaputra release, FIO would evaluate File System performance for the host machine. It will also test the I/O performance provided by the hypervisor(KVM) when Storage benchmarks are run inside VMs.

QTIP storage test cases consist of:

1. FIO Job to benchmark baremetal file system performance

2. FIO Job to bechmark virtual machine file system performance

Note: For Brahmaputra release, only the Ephemeral Storage is being tested. For C release persistent block and object storage would be tested.

The FIO Job would consist of:

1. A file size of 5GB
2. Random Read 50%, Random Write 50%
3. Direct I/O
4. Asynch I/O Engine
5. I/O Queue depth of 2
6. Block size :4K

For this Job, I/O per second would be measured along mean I/O latency to provide storage performance numbers.

Revision: Build date: February 24, 2016