OPNFV FUNCTEST Configuration Guide

Release arno.2015.1.0 (604d2fc)

OPNFV

August 19, 2016

CONTENTS

CHAPTER

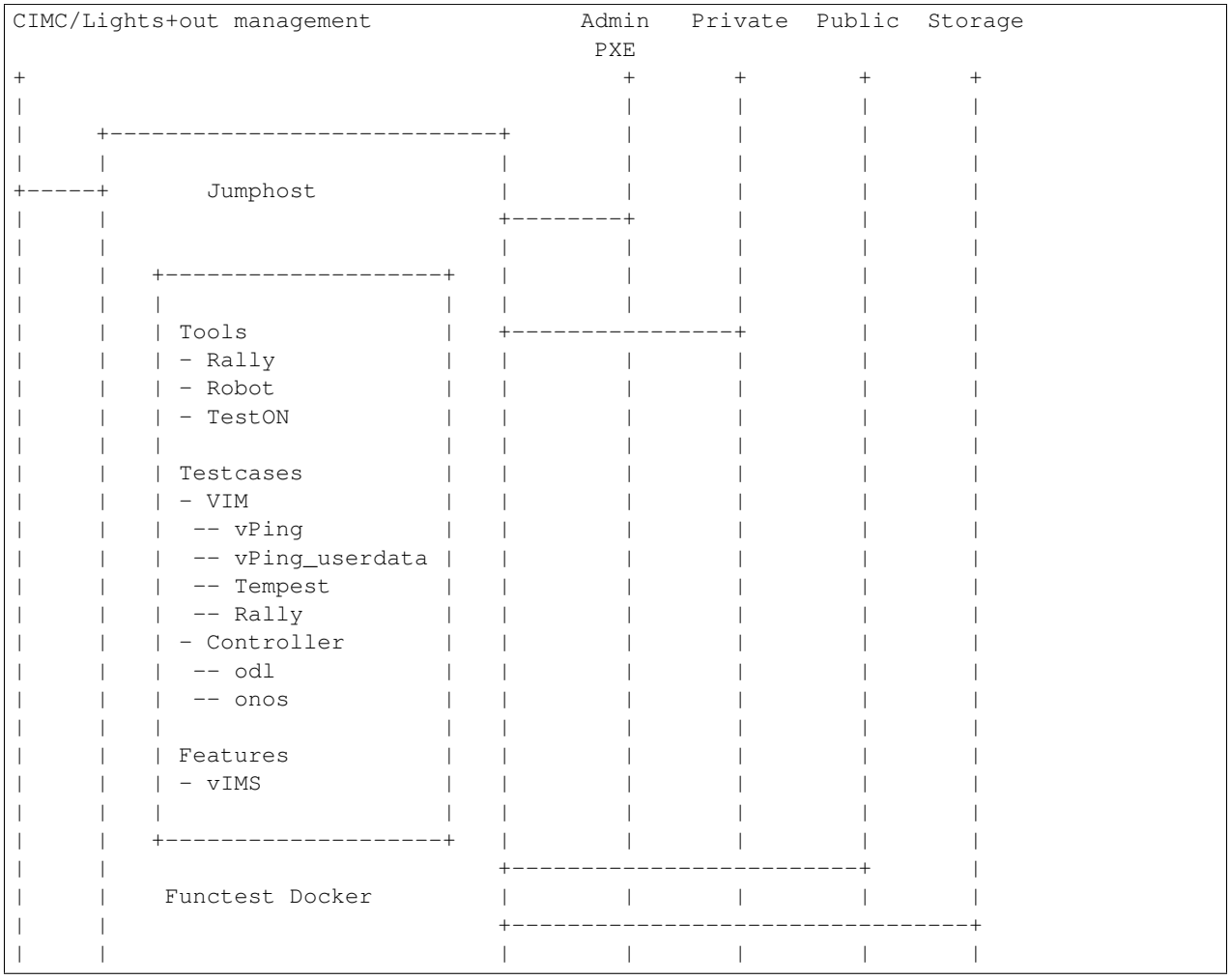
ONE

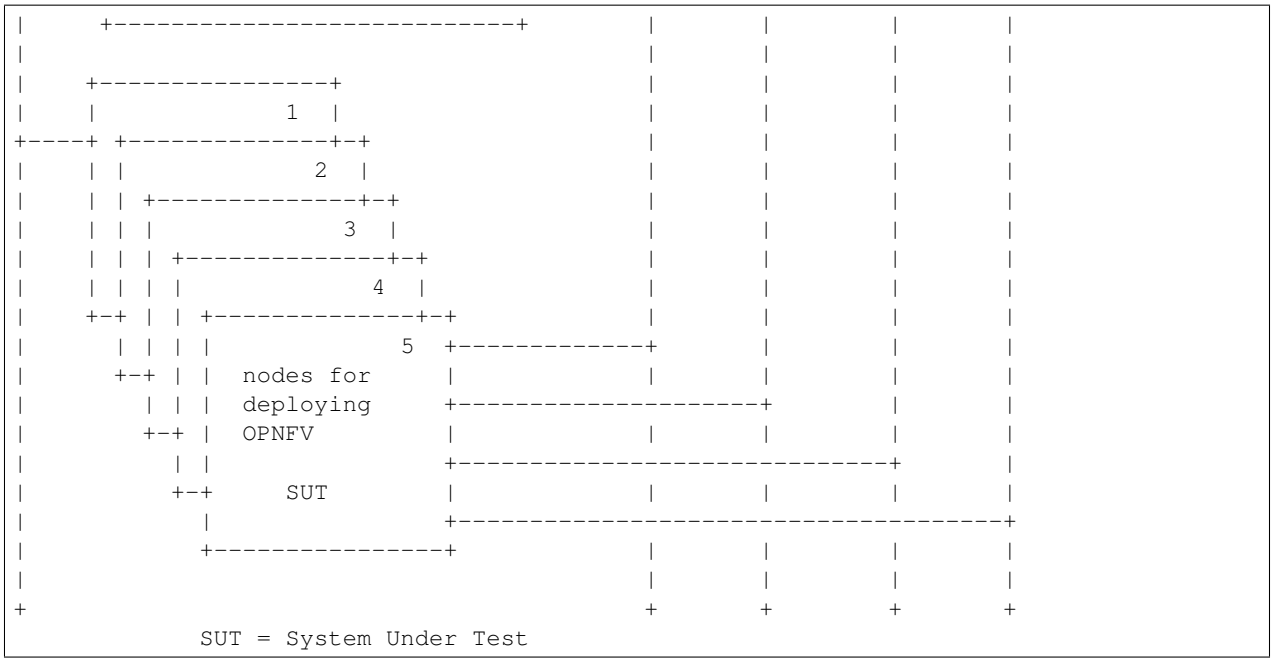
INTRODUCTION

This document describes how to install and configure Functest in OPNFV. The Functest CLI is utilized during the Functest environment preparation step. The given example commands should work in both virtual and bare metal cases alike.

1.1 High level architecture

The high level architecture of Functest within OPNFV can be described as follows:





All the libraries and dependencies needed by all of the Functest tools are pre-installed into the Docker image. This allows running Functest on any platform on any Operating System.

The automated mechanisms inside the Functest Docker container will:

- retrieve OpenStack credentials
- prepare the environment according to the SUT
- perform the appropriate functional tests
- push the test results into the OPNFV test result database

This Docker image can be integrated into CI or deployed independently.

Please note that the Functest Docker container has been designed for OPNFV, however, it would be possible to adapt it to any VIM + controller environment, since most of the test cases are integrated from upstream communities.

The functional test cases are described in the Functest User Guide [2]

CHAPTER

TWO

PREREQUISITES

The OPNFV deployment is out of the scope of this document but it can be found in [4]. The OPNFV platform is considered as the System Under Test (SUT) in this document.

Several prerequisites are needed for Functest:

1. A Jumphost to run Functest on
2. A Docker daemon shall be installed on the Jumphost
3. A public/external network created on the SUT
4. An admin/management network created on the SUT
5. Connectivity from the Jumphost to the SUT public/external network
6. Connectivity from the Jumphost to the SUT admin/management network

WARNING: Connectivity from Jumphost is essential and it is of paramount importance to make sure it is working before even considering to install and run Functest. Make also sure you understand how your networking is designed to work.

NOTE: **Jumphost** refers to any server which meets the previous requirements. Normally it is the same server from where the OPNFV deployment has been triggered previously.

NOTE: If your Jumphost is operating behind a company http proxy and/or Firewall, please consult first the section *Proxy Support*, towards the end of this document. The section details some tips/tricks which *may* be of help in a proxified environment.

2.1 Docker installation

Docker installation and configuration is only needed to be done once through the life cycle of Jumphost.

If your Jumphost is based on Ubuntu, RHEL or CentOS linux, please consult the references below for more detailed instructions. The commands below are offered as a short reference.

Tip: For running docker containers behind the proxy, you need first some extra configuration which is described in section *Docker Installation on CentOS behind http proxy*. You should follow that section before installing the docker engine.

Docker installation needs to be done as root user. You may use other userid's to create and run the actual containers later if so desired. Log on to your Jumphost as root user and install the Docker Engine (e.g. for CentOS family):

```
curl -sSL https://get.docker.com/ | sh
systemctl start docker
```

```
*Tip:* If you are working through proxy, please set the https_proxy
environment variable first before executing the curl command.
```

Add your user to docker group to be able to run commands without sudo:

```
sudo usermod -aG docker <your_user>
```

A reconnect is needed. There are 2 ways for this:

1. Re-login to your account
2. su - <username>

References - Installing Docker Engine on different Linux Operating Systems:

- Ubuntu
- RHEL
- CentOS

2.2 Public/External network on SUT

Some of the tests against the VIM (Virtual Infrastructure Manager) need connectivity through an existing public/external network in order to succeed. This is needed, for example, to create floating IPs to access VM instances through the public/external network (i.e. from the Docker container).

By default, the four OPNFV installers provide a fresh installation with a public/external network created along with a router. Make sure that the public/external subnet is reachable from the Jumphost.

Hint: For the given OPNFV Installer in use, the IP sub-net address used for the public/external network is usually a planning item and should thus be known. Consult the OPNFV Configuration guide [4], and ensure you can reach each node in the SUT, from the Jumphost using the 'ping' command using the respective IP address on the public/external network for each node in the SUT. The details of how to determine the needed IP addresses for each node in the SUT may vary according to the used installer and are therefore ommitted here.

2.3 Connectivity to Admin/Management network on SUT

Some of the Functest tools need to have access to the OpenStack admin/management network of the controllers [1].

For this reason, check the connectivity from the Jumphost to all the controllers in cluster in the OpenStack admin/management network range.

CHAPTER

THREE

INSTALLATION AND CONFIGURATION

3.1 Pulling the Docker image

Pull the Functest Docker image ('opnfv/functest') from the public dockerhub registry under the OPNFV account: [dockerhub], with the following docker command:

```
docker pull opnfv/functest:<TagIdentifier>
```

where <TagIdentifier> identifies a release of the Functest docker container image in the public dockerhub registry. There are many tags created automatically by the CI mechanisms, and you must ensure you pull an image with the **correct tag** to match the OPNFV software release installed in your environment. All available tagged images can be seen from location [FunctestDockerTags]. For example, when running on the first official release of the OPNFV Colorado system platform, tag "colorado.1.0" is needed. Pulling other tags might cause some problems while running the tests. If you need to specifically pull the latest Functest docker image, then omit the tag argument:

```
docker pull opnfv/functest
```

After pulling the Docker image, check that it is available with the following docker command:

```
[functester@jumphost ~]$ docker images
REPOSITORY     TAG              IMAGE ID      CREATED      SIZE
opnfv/functest latest           8cd6683c32ae  2 weeks ago  1.611 GB
opnfv/functest brahmaputra.3.0 94b78faa94f7  4 weeks ago  874.9 MB
hello-world    latest           94df4f0ce8a4  7 weeks ago    967 B
```

Docker images pulled without a tag specifier bear the implicitly assigned label "latest", as seen above.

The Functest docker container environment can -in principle- be also used with non-OPNFV official installers (e.g. 'devstack'), with the **disclaimer** that support for such environments is outside of the scope and responsibility of the OPNFV project.

3.2 Accessing the Openstack credentials

OpenStack credentials are mandatory and can be retrieved in different ways. From inside the running Functest docker container the "functest env prepare" command will automatically look for the Openstack credentials file "/home/opnfv/functest/conf/openstack.creds" and retrieve it unless the file already exists. This Functest environment preparation step is described later in this document.

WARNING: When the installer type is "joid" you have to have the credentials file inside the running container **before** initiating the functest environment preparation. For that reason you have to choose either one of the options below, since the automated copying does not work for "joid".

You can also specifically pass in the needed file prior to running the environment preparation either:

- by using the -v option when creating the Docker container. This is referred to in docker documentation as "Bind Mounting". See the usage of this parameter in the following chapter.
- or creating a local file '/home/opnfv/functest/conf/openstack.creds' inside the running container with the credentials in it. Consult your installer guide for further details. This is however not instructed in this document.

NOTE: When the installer type is "fuel" and virtualized deployment is used, there you have to explicitly fetch the credentials file executing the following sequence

1. Create a container as described in next chapter but do not "Bind Mount" the credentials
2. Log in to container and execute the following command. Replace the IP with installer address after the "-a" parameter:

```
$repos_dir/releng/utils/fetch_os_creds.sh \
-d /home/opnfv/functest/conf/openstack.creds \
-i fuel \
-a 10.20.0.2 \
-v
( -d specifies the full path to the Openstack credential file
-i specifies the INSTALLER_TYPE
-a specifies the INSTALLER_IP
-v indicates a virtualized environment and takes no arguments )
```

3. Continue with your testing, initiate functest environment preparation, run tests etc.

In proxified environment you may need to change the credentials file. There are some tips in chapter: *Proxy support*

3.3 Functest Docker parameters

This chapter explains how to run a container for executing functest test suites. Numbered list below explains some details of the recommended parameters for invoking docker container

1. It is a good practice to assign a precise container name through the **–name** option.
2. Assign parameter for installer type:

```
-e "INSTALLER_TYPE=<type>"
# Use one of following apex, compass, fuel or joid
```

3. Functest needs to know the IP of the installer:

```
-e "INSTALLER_IP=<Specific IP Address>"
```

4. Credentials for accessing the Openstack. Most convenient way of passing them to container is by having a local copy of the credentials file in Jumphost and then using the **-v** option. In the example we have local file by the name of "overcloudrc" and we are using that as an argument:

```
-v ~/overcloudrc:/home/opnfv/functest/conf/openstack.creds

The credentials file needs to exist in the Docker container
under the path: '/home/opnfv/functest/conf/openstack.creds'.
```

   WARNING: If you are using the Joid installer, you must pass the credentials using the **-v** option. See the section *Accessing the Openstack credentials* above.

5. Passing deployment scenario When running Functest against any of the supported OPNFV scenarios, it is recommended to include also the environment variable **DEPLOY_SCENARIO**. The **DEPLOY_SCENARIO** environment variable is passed with the format:

```
    -e "DEPLOY_SCENARIO=os-<controller>-<nfv_feature>-<ha_mode>"
    where:
    os = OpenStack (No other VIM choices currently available)
    controller is one of ( nosdn | odl_l2 | odl_l3 | onos )
    nfv_feature is one or more of ( ovs | kvm | sfc | bgpvpn | nofeature )
             If several features are pertinent then use the underscore
             character '_' to separate each feature (e.g. ovs_kvm)
             'nofeature' indicates no NFV feature is deployed
    ha_mode is one of ( ha | noha )
```

NOTE: Not all possible combinations of "DEPLOY_SCENARIO" are supported. The name passed in to the Functest Docker container must match the scenario used when the actual OPNFV platform was deployed.

Putting all above together, when using installer 'fuel' and an invented INSTALLER_IP of '10.20.0.2', the recommended command to create the Functest Docker container is as follows:

```
docker run --name "FunctestContainer" -it \
-e "INSTALLER_IP=10.20.0.2" \
-e "INSTALLER_TYPE=fuel" \
-e "DEPLOY_SCENARIO=os-odl_l2-ovs_kvm-ha" \
-v ~/overcloudrc:/home/opnfv/functest/conf/openstack.creds \
opnfv/functest /bin/bash
```

After the *run* command, a new prompt appears which means that we are inside the container and ready to move to the next step.

For tips on how to set up container with installer Apex, see chapter *Apex Installer Tips*.

Finally, three additional environment variables can also be passed in to the Functest Docker Container, using the -e "<EnvironmentVariable>=<Value>" mechanism. The first two of these are only relevant to Jenkins CI invoked testing and **should not be used** when performing manual test scenarios:

```
-e "NODE_NAME=<Test POD Name>" \
-e "BUILD_TAG=<Jenkins Build Tag>" \
-e "CI_DEBUG=<DebugTraceValue>"
where:
<Test POD Name> = Symbolic name of the POD where the tests are run.
                  Visible in test results files, which are stored
                  to the database. This option is only used when
                  tests are activated under Jenkins CI control.
                  It indicates the POD/hardware where the test has
                  been run. If not specified, then the POD name is
                  defined as "Unknown" by default.
                  DO NOT USE THIS OPTION IN MANUAL TEST SCENARIOS.
<Jenkins Build tag> = Symbolic name of the Jenkins Build Job.
                      Visible in test results files, which are stored
                      to the database. This option is only set when
                      tests are activated under Jenkins CI control.
                      It enables the correlation of test results,
                      which
                      are independently pushed to the results datbase
                      from different Jenkins jobs.
                      DO NOT USE THIS OPTION IN MANUAL TEST SCENARIOS.
<DebugTraceValue> = "true" or "false"
                    Default = "false", if not specified
                    If "true" is specified, then additional debug trace
                    text can be sent to the test results file / log files
                    and also to the standard console output.
```

3.4 Apex Installer Tips

Some specific tips are useful for the Apex Installer case. If not using Apex Installer; ignore this section.

In case of Triple-O based installer (like Apex) the docker container needs to connect to the installer VM, so it is then required that some known SSH keys are present in docker container. Since the Jumphost root SSH keys are already known, easiest way is to use those using the 'Bind mount' method. See below for sample parameter:

```
-v /root/.ssh/id_rsa:/root/.ssh/id_rsa

NOTE: You need the "sudo" when creating the container to access root
users ssh credentials even the docker command itself might not
require that.
```

HINT! In case of Triple-O installers you can find value for the INSTALLER_IP parameter by executing command and note the returned IP address:

```
inst=$(sudo virsh list | grep -iEo "undercloud|instack")
sudo virsh domifaddr ${inst}

NOTE: In releases prior to Colorado, the name 'instack' was
used. Currently the name 'undercloud' is used.
```

You can copy the credentials file from the "stack" users home directory in installer VM to Jumphost. Please check the correct IP from the command above. In the example below we are using invented IP address "192.168.122.89":

```
scp stack@192.168.122.89:overcloudrc .
```

Here is an example of the full docker command invocation for an Apex installed system, using latest Functest docker container, for illustration purposes:

```
sudo docker run -it --name "ApexFuncTestODL" \
-e "INSTALLER_IP=192.168.122.89" \
-e "INSTALLER_TYPE=apex" \
-e "DEPLOY_SCENARIO=os-odl_l2-nofeature-ha" \
-v /root/.ssh/id_rsa:/root/.ssh/id_rsa \
-v ~/overcloudrc:/home/opnfv/functest/conf/openstack.creds \
opnfv/functest /bin/bash
```

3.5 Compass installer local development env usage Tips

In the compass-functest local test case check and development environment, in order to get openstack service inside the functest container, some parameters should be configured during container creation, which are hard to guess for freshman. This section will provide the guideline, the parameters values are defaults here, which should be adjusted according to the settings, the complete steps are given here so as not to appear too abruptly.

1, Pull Functest docker image from public dockerhub:

```
docker pull opnfv/functest:<Tag>
```

<Tag> here can be "brahmaputra.1.0", "colorado.1.0", etc. Tag omitted means the latest docker image:

```
docker pull opnfv/functest
```

2, Functest Docker container creation

To make a file used for the environment, such as 'functest-docker-env':

```
OS_AUTH_URL=http://172.16.1.222:35357/v2.0
OS_USERNAME=admin
OS_PASSWORD=console
OS_TENANT_NAME=admin
OS_VOLUME_API_VERSION=2
OS_PROJECT_NAME=admin
INSTALLER_TYPE=compass
INSTALLER_IP=192.168.200.2
EXTERNAL_NETWORK=ext-net
```

Note: please adjust the content according to the environment, such as 'TENANT_ID' maybe used for some special cases.

Then to create the Functest docker:

```
docker run --privileged=true --rm -t \
--env-file functest-docker-env \
--name <Functest_Container_Name> \
opnfv/functest:<Tag> /bin/bash
```

Note: it is recommended to be run on jumpserver.

3, To attach Functest container

Before trying to attach the Functest container, the status can be checked by:

```
docker ps -a
```

to attach the 'Up' status Functest container and start bash mode:

```
docker exec -it <Functest_Container_Name> bash
```

4, Functest environemnt preparation and check

To see the Section below *Preparing the Functest environment*.

3.6 Functest docker container directory structure

Inside the Functest docker container, the following directory structure should now be in place:

```
`-- home
    `-- opnfv
      |-- functest
      |   |-- conf
      |   |-- data
      |   `-- results
      `-- repos
          |-- bgpvpn
          |-- doctor
          |-- functest
          |-- odl_integration
          |-- onos
          |-- promise
          |-- rally
          |-- releng
          `-- vims-test
```

Underneath the '/home/opnfv/' directory, the Functest docker container includes two main directories:

- The **functest** directory stores configuration files (e.g. the OpenStack creds are stored in path '/home/opnfv/functest/conf/openstack.creds'), the **data** directory stores a 'cirros' test image used in some functional tests and the **results** directory stores some temporary result log files
- The **repos** directory holds various repositories. The directory '/home/opnfv/repos/functest' is used to prepare the needed Functest environment and to run the tests. The other repository directories are used for the installation of the needed tooling (e.g. rally) or for the retrieval of feature projects scenarios (e.g. promise)

The structure under the **functest** repository can be described as follows:

```
. |-- INFO
  |-- LICENSE
  |-- __init__.py
  |-- ci
  |   |-- __init__.py
  |   |-- check_os.sh
  |   |-- config_functest.yaml
  |   |-- exec_test.sh
  |   |-- prepare_env.py
  |   |-- run_tests.py
  |   |-- testcases.yaml
  |   |-- tier_builder.py
  |   `-- tier_handler.py
  |-- cli
  |   |-- __init__.py
  |   |-- cli_base.py
  |   |-- commands
  |   |-- functest-complete.sh
  |   `-- setup.py
  |-- commons
  |   |-- ims
  |   |-- mobile
  |   `--traffic-profile-guidelines.rst
  |-- docker
  |   |-- Dockerfile
  |   |-- config_install_env.sh
  |   `-- requirements.pip
  |-- docs
  |   |-- com
  |   |-- configguide
  |   |-- devguide
  |   |-- images
  |   |-- release-notes
  |   |-- results
  |   `--userguide
  |-- testcases
  |   |-- Controllers
  |   |-- OpenStack
  |   |-- __init__.py
  |   |-- features
  |   |-- security_scan
  |   `-- vIMS
  `-- utils
      |-- __init__.py
      |-- functest_logger.py
      |-- functest_utils.py
      |-- openstack_clean.py
      |-- openstack_snapshot.py
      `-- openstack_utils.py
```

```
  (Note: All *.pyc files removed from above list for brevity...)
```

We may distinguish 7 different directories:

- **ci**: This directory contains test structure definition files (e.g <filename>.yaml) and bash shell/python scripts used to configure and execute Functional tests. The test execution script can be executed under the control of Jenkins CI jobs.
- **cli**: This directory holds the python based Functest CLI utility source code, which is based on the Python 'click' framework.
- **commons**: This directory is dedicated for storage of traffic profile or any other test inputs that could be reused by any test project.
- **docker**: This directory includes the needed files and tools to build the Funtest Docker container image.
- **docs**: This directory includes documentation: Release Notes, User Guide, Configuration Guide and Developer Guide. Test results are also located in a sub–directory called 'results'.
- **testcases**: This directory includes the scripts required by Functest internal test cases and other feature projects test cases.
- **utils**: this directory holds Python source code for some general purpose helper utilities, which testers can also re-use in their own test code. See for an example the Openstack helper utility: 'openstack_utils.py'.

3.7 Useful Docker commands

When typing **exit** in the container prompt, this will cause exiting the container and probably stopping it. When stopping a running Docker container all the changes will be lost, there is a keyboard shortcut to quit the container without stopping it: <CTRL>-P + <CTRL>-Q. To reconnect to the running container **DO NOT** use the *run* command again (since it will create a new container), use the *exec* or *attach* command instead:

```
docker ps  # <check the container ID from the output>
docker exec -ti <CONTAINER_ID> /bin/bash
```

There are other useful Docker commands that might be needed to manage possible issues with the containers.

List the running containers:

```
docker ps
```

List all the containers including the stopped ones:

```
docker ps -a
```

Start a stopped container named "FunTest":

```
docker start FunTest
```

Attach to a running container named "StrikeTwo":

```
docker attach StrikeTwo
```

It is useful sometimes to remove a container if there are some problems:

```
docker rm <CONTAINER_ID>
```

Use the *-f* option if the container is still running, it will force to destroy it:

```
docker -f rm <CONTAINER_ID>
```

Check the Docker documentation dockerdocs for more information.

3.8 Preparing the Functest environment

Once the Functest docker container is up and running, the required Functest environment needs to be prepared. A custom built **functest** CLI utility is available to perform the needed environment preparation action. Once the environment is prepared, the **functest** CLI utility can be used to run different functional tests. The usage of the **functest** CLI utility to run tests is described further in the Functest User Guide OPNFV_FuncTestUserGuide

Prior to commencing the Functest environment preparation, we can check the initial status of the environment. Issue the **functest env status** command at the prompt:

```
functest env status
Functest environment is not installed.

Note: When the Functest environment is prepared, the command will
return the status: "Functest environment ready to run tests."
```

To prepare the Functest docker container for test case execution, issue the **functest env prepare** command at the prompt:

```
functest env prepare
```

This script will make sure that the requirements to run the tests are met and will install the needed libraries and tools by all Functest test cases. It should be run only once every time the Functest docker container is started from scratch. If you try to run this command, on an already prepared enviroment, you will be prompted whether you really want to continue or not:

```
functest env prepare
It seems that the environment has been already prepared.
Do you want to do it again? [y|n]

(Type 'n' to abort the request, or 'y' to repeat the
 environment preparation)
```

To list some basic information about an already prepared Functest docker container environment, issue the **functest env show** at the prompt:

```
functest env show
+=========================================================+
| Functest Environment info                               |
+=========================================================+
|  INSTALLER: apex, 192.168.122.89                        |
|   SCENARIO: os-odl_l2-nofeature-ha                      |
|        POD: localhost                                   |
| GIT BRANCH: master                                      |
|   GIT HASH: 5bf1647dec6860464eeb082b2875798f0759aa91    |
| DEBUG FLAG: false                                       |
+---------------------------------------------------------+
|     STATUS: ready                                       |
+---------------------------------------------------------+

Where:

INSTALLER:  Displays the INSTALLER_TYPE value
```

```
            - here = "apex"
            and the INSTALLER_IP value
            - here = "192.168.122.89"
SCENARIO:   Displays the DEPLOY_SCENARIO value
            - here = "os-odl_l2-nofeature-ha"
POD:        Displays the value passed in NODE_NAME
            - here = "localhost"
GIT BRANCH: Displays the git branch of the OPNFV Functest
            project repository included in the Functest
            Docker Container.
            - here = "master"
                     (In first official colorado release
                      would be "colorado.1.0")
GIT HASH:   Displays the git hash of the OPNFV Functest
            project repository included in the Functest
            Docker Container.
            - here = "5bf1647dec6860464eeb082b2875798f0759aa91"
DEBUG FLAG: Displays the CI_DEBUG value
            - here = "false"

NOTE: In Jenkins CI runs, an additional item "BUILD TAG"
      would also be listed. The valaue is set by Jenkins CI.
```

Finally, the **functest** CLI has a **–help** options:

Some examples:

```
functest --help Usage: functest [OPTIONS] COMMAND [ARGS]...

Options:
  --version  Show the version and exit.
  -h, --help Show this message and exit.

Commands:
  env
  openstack
  testcase
  tier

functest env --help
Usage: functest env [OPTIONS] COMMAND [ARGS]...

Options:
  -h, --help Show this message and exit.

Commands:
  prepare  Prepares the Functest environment.
  show     Shows information about the current...
  status   Checks if the Functest environment is ready...
```

3.9 Checking Openstack and credentials

It is recommended and fairly straightforward to check that Openstack and credentials are working as expected.

Once the credentials are there inside the container, they should be sourced before running any Openstack commands:

```
source /home/opnfv/functest/conf/openstack.creds
```

After this, try to run any OpenStack command to see if you get any output, for instance:

```
openstack user list
```

This will return a list of the actual users in the OpenStack deployment. In any other case, check that the credentials are sourced:

```
env|grep OS_
```

This command must show a set of environment variables starting with *OS_*, for example:

```
OS_REGION_NAME=RegionOne
OS_DEFAULT_DOMAIN=default
OS_PROJECT_NAME=admin
OS_PASSWORD=admin
OS_AUTH_STRATEGY=keystone
OS_AUTH_URL=http://172.30.10.3:5000/v2.0
OS_USERNAME=admin
OS_TENANT_NAME=admin
OS_ENDPOINT_TYPE=internalURL
OS_NO_CACHE=true
```

If the OpenStack command still does not show anything or complains about connectivity issues, it could be due to an incorrect url given to the OS_AUTH_URL environment variable. Check the deployment settings.

3.10 SSL Support

If you need to connect to a server that is TLS-enabled (the auth URL begins with "https") and it uses a certificate from a private CA or a self-signed certificate, then you will need to specify the path to an appropriate CA certificate to use, to validate the server certificate with the environment variable OS_CACERT:

```
echo $OS_CACERT
/etc/ssl/certs/ca.crt
```

However, this certificate does not exist in the container by default. It has to be copied manually from the OpenStack deployment. This can be done in 2 ways:

1. Create manually that file and copy the contents from the OpenStack controller.
2. (Recommended) Add the file using a Docker volume when starting the container:

```
    -v <path_to_your_cert_file>:/etc/ssl/certs/ca.cert
```

You might need to export OS_CACERT environment variable inside the container:

```
export OS_CACERT=/etc/ssl/certs/ca.crt
```

Certificate verification can be turned off using OS_INSECURE=true. For example, Fuel uses self-signed cacerts by default, so an pre step would be:

```
export OS_INSECURE=true
```

3.11 Proxy support

If your Jumphost node is operating behind a http proxy, then there are 2 places where some special actions may be needed to make operations succeed:

1. Initial installation of docker engine First, try following the official Docker documentation for Proxy settings. Some issues were experienced on CentOS 7 based Jumphost. Some tips are documented in section: *Docker Installation on CentOS behind http proxy* below.
2. Execution of the Functest environment preparation inside the created docker container Functest needs internet access to download some resources for some test cases. This might not work properly if the Jumphost is connecting to internet through a http Proxy.

If that is the case, make sure the resolv.conf and the needed http_proxy and https_proxy environment variables, as well as the 'no_proxy' environment variable are set correctly:

```
# Make double sure that the 'no_proxy=...' line in the
# 'openstack.creds' file is commented out first. Otherwise, the
# values set into the 'no_proxy' environment variable below will
# be ovewrwritten, each time the command
# 'source ~/functest/conf/openstack.creds' is issued.

cd ~/functest/conf/
sed -i 's/export no_proxy/#export no_proxy/' openstack.creds
source ./openstack.creds

# Next calculate some IP addresses for which http_proxy
# usage should be excluded:

publicURL_IP=$(echo $OS_AUTH_URL | grep -Eo "([0-9]+\.){3}[0-9]+")

adminURL_IP=$(openstack catalog show identity | \
grep adminURL | grep -Eo "([0-9]+\.){3}[0-9]+")

export http_proxy="<your http proxy settings>"
export https_proxy="<your https proxy settings>"
export no_proxy="127.0.0.1,localhost,$publicURL_IP,$adminURL_IP"

# Ensure that "git" uses the http_proxy
# This may be needed if your firewall forbids SSL based git fetch
git config --global http.sslVerify True
git config --global http.proxy <Your http proxy settings>
```

Validation check: Before running **'functest env prepare'** CLI command, make sure you can reach http and https sites from inside the Functest docker container.

For example, try to use the **nc** command from inside the functest docker container:

```
nc -v google.com 80
Connection to google.com 80 port [tcp/http] succeeded!

nc -v google.com 443
Connection to google.com 443 port [tcp/https] succeeded!
```

Note: In a Jumphost node based on the CentOS family OS, the **nc** commands might not work. You can use the **curl** command instead.

curl http://www.google.com:80 <HTML><HEAD><meta http-equiv="content-type" . . </BODY></HTML>

curl https://www.google.com:443 <HTML><HEAD><meta http-equiv="content-type" . . </BODY></HTML>

(Ignore the content. If command returns a valid HTML page, it proves the connection.)

3.12 Docker Installation on CentOS behind http proxy

This section is applicable for CentOS family OS on Jumphost which itself is behind a proxy server. In that case, the instructions below should be followed **before** installing the docker engine:

```
1) # Make a directory '/etc/systemd/system/docker.service.d'
   # if it does not exist
   sudo mkdir /etc/systemd/system/docker.service.d

2) # Create a file called 'env.conf' in that directory with
   # the following contents:
   [Service]
   EnvironmentFile=-/etc/sysconfig/docker

3) # Set up a file called 'docker' in directory '/etc/sysconfig'
   # with the following contents:
   HTTP_PROXY="<Your http proxy settings>"
   HTTPS_PROXY="<Your https proxy settings>"
   http_proxy="${HTTP_PROXY}"
   https_proxy="${HTTPS_PROXY}"

4) # Reload the daemon
   systemctl daemon-reload

5) # Sanity check - check the following docker settings:
   systemctl show docker | grep -i env

   Expected result:
   ----------------
   EnvironmentFile=/etc/sysconfig/docker (ignore_errors=yes)
   DropInPaths=/etc/systemd/system/docker.service.d/env.conf
```

Now follow the instructions in [InstallDockerCentOS] to download and install the **docker-engine**. The instructions conclude with a "test pull" of a sample "Hello World" docker container. This should now work with the above pre-requisite actions.

CHAPTER

FOUR

INTEGRATION IN CI

In CI we use the Docker image and execute the appropriate commands within the container from Jenkins.

Docker creation in set-functest-env builder [3]:

```
envs="INSTALLER_TYPE=${INSTALLER_TYPE} -e INSTALLER_IP=${INSTALLER_IP} -e NODE_NAME=${NODE_NAME}"
[...]
docker pull opnfv/functest:latest_stable
cmd="docker run -id -e $envs ${labconfig} ${sshkey} ${res_volume} opnfv/functest:latest_stable /bin/b
echo "Functest: Running docker run command: ${cmd}"
${cmd}
docker ps -a
sleep 5
container_id=$(docker ps | grep 'opnfv/functest:latest_stable' | awk '{print $1}' | head -1)
echo "Container ID=${container_id}"
if [ -z ${container_id} ]; then
    echo "Cannot find opnfv/functest container ID ${container_id}. Please check if it is existing."
    docker ps -a
    exit 1
fi
echo "Starting the container: docker start ${container_id}"
docker start ${container_id}
sleep 5
docker ps
if [ $(docker ps | grep 'opnfv/functest:latest_stable' | wc -l) == 0 ]; then
    echo "The container opnfv/functest with ID=${container_id} has not been properly started. Exiting
    exit 1
fi
cmd="${FUNCTEST_REPO_DIR}/docker/prepare_env.sh"
echo "Executing command inside the docker: ${cmd}"
docker exec ${container_id} ${cmd}
```

Test execution in functest-all builder [3]:

```
echo "Functest: run $FUNCTEST_SUITE_NAME"
cmd="${FUNCTEST_REPO_DIR}/docker/run_tests.sh --test $FUNCTEST_SUITE_NAME ${flag}"
container_id=$(docker ps -a | grep opnfv/functest | awk '{print $1}' | head -1)
docker exec $container_id $cmd
```

Docker clean in functest-cleanup builder [3]:

```
echo "Cleaning up docker containers/images..."
# Remove previous running containers if exist
if [[ ! -z $(docker ps -a | grep opnfv/functest) ]]; then
echo "Removing existing opnfv/functest containers..."
docker ps | grep opnfv/functest | awk '{print $1}' | xargs docker stop
docker ps -a | grep opnfv/functest | awk '{print $1}' | xargs docker rm
```

```
fi

# Remove existing images if exist
if [[ ! -z $(docker images | grep opnfv/functest) ]]; then
echo "Docker images to remove:"
docker images | head -1 && docker images | grep opnfv/functest
image_tags=($(docker images | grep opnfv/functest | awk '{print $2}'))
for tag in "${image_tags[@]}"; do
    echo "Removing docker image opnfv/functest:$tag..."
    docker rmi opnfv/functest:$tag
done
fi
```

CHAPTER

FIVE

REFERENCES

OPNFV main site: opnfvmain.

OPNFV functional test page: opnfvfunctest.

IRC support channel: #opnfv-functest