# OPNFV FUNCTEST developer guide

***Release arno.2015.1.0 (b27154b)***

**OPNFV**

June 28, 2016

CONTENTS

CHAPTER

ONE

# INTRODUCTION

This document describes how feature projects aiming to run functional tests can be integrated into FuncTest framework.

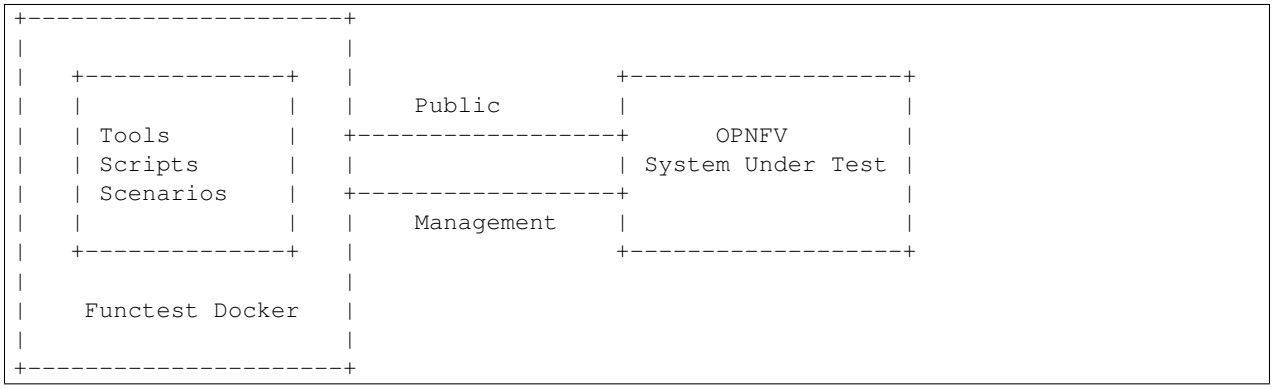
CHAPTER

TWO

# FUNCTEST HIGH LEVEL ARCHITECTURE

Functest is project delivering a test container dedicated to OPNFV. It includes the tools, the scripts and the test scenarios.

Functest can be described as follow:




Functest deals with internal and external test cases. The Internal test cases in Brahmaputra are:

- vPing_SSH
- vPing_userdata
- ODL
- Tempest
- vIMS
- Rally

The external tescases are:

- Promise
- Doctor
- Onos
- BGPVPN

see [2] for details.

Functest can also be considered as a framework that may include external OPNFV projects. This framework will ease the integration of the feature test suite to the CI.

CHAPTER

THREE

# HOW FUNCTEST WORKS

The installation and the launch of the Functest docker image is described in [1].

The Functest docker directories are:

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

| Directory | Subdirectory | Comments |
|---|---|---|
| opnfv | <project>/conf | All the useful configuration file(s) for <project> the openstack creds are put there for CI<br>It is recommended to push it there when passing the credentials to container through the -v option |
| | <project>/data | Usefull data, images for <projects><br>By default we put a cirros image: cirros-0.3.4-x86_64-disk.img<br>This image can be used by any projects |
| | <project>/results | Local result directory of project <project> |
| repos | bgpvpn | |
| | doctor | |
| | functest | |

```
|           +-------------------+                                          +
|           | odl_integration   |                                          |
|           +-------------------+    Clone of the useful repositories      +
|           | onos              |     These repositories may include:      |
|           +-------------------+                - tooling                 +
|           | promise           |                - scenario                |
|           +-------------------+                - scripts                 +
|           | rally             |                                          |
|           +-------------------+                                          +
|           | releng            |                                          |
|           +-------------------+                                          +
|           | vims-test         |                                          |
|           +-------------------+                                          +
|           | <your project>    |                                          |
+-----------+-------------------+------------------------------------------+
```

Before running the test suite, you must prepare the environement by running:

```
$ /home/opnfv/repos/functest/docker/prepare_env.sh
```

By running prepare_env.sh, you build the test environement required by the tests including the retrieval and sourcing of OpenStack credentials. This is an example of the env variables we have in the docker container:

- HOSTNAME=373f77816eb0
- INSTALLER_TYPE=fuel
- repos_dir=/home/opnfv/repos
- INSTALLER_IP=10.20.0.2
- PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
- PWD=/home/opnfv
- SHLVL=1
- HOME=/home/opnfv
- NODE_NAME=opnfv-jump-2
- creds=/home/opnfv/functest/conf/openstack.creds
- _=/usr/bin/env

The prepare_env.sh will source the credentials, so once run you should have access to the following env variables:

```
root@373f77816eb0:~# env|grep OS_
OS_REGION_NAME=RegionOne
OS_PROJECT_NAME=admin
OS_PASSWORD=admin
OS_AUTH_STRATEGY=keystone
OS_AUTH_URL=http://172.30.10.70:5000/v2.0
OS_USERNAME=admin
OS_TENANT_NAME=admin
OS_ENDPOINT_TYPE=internalURL
OS_NO_CACHE=true
```

Then you may run the test suite by running:

```
$ /home/opnfv/repos/functest/docker/run_tests.sh -t <your project>
```

see [2] for details.

CHAPTER

FOUR

# HOW TO INTEGRATE FUNCTEST

The files of the Functest repository you must modify to integrate Functest are:

- functest/docker/Dockerfile
- functest/docker/common.sh
- functest/docker/requirements.pip
- functest/docker/run_tests.sh
- functest/docker/prepare_env.sh
- functest/config_funtest.yaml

## 4.1 Dockerfile

This file lists the repositories to be cloned in the Functest container. The repositories can be internal or external:

```
RUN git clone https://gerrit.opnfv.org/gerrit/<your porject> ${repos_dir}/<your project>
```

## 4.2 Common.sh

This file can be used to declare the branch and commit variables of your projects:

```
<YOUR_PROJECT>_BRANCH=$(cat $config_file | grep -w <your project>_branch | awk 'END {print $NF}')
<YOUR_PROJECT>_COMMIT=$(cat $config_file | grep -w <your project>_commit | awk 'END {print $NF}')

echo "<YOUR_PROJECT>_BRANCH=${<YOUR_PROJECT>_BRANCH}"
echo "<YOUR_PROJECT>_COMMIT=${<YOUR_PROJECT>_COMMIT}"
```

## 4.3 requirements.pip

This file can be used to preloaded all the needed Python libraries (and avoid that each project does it) The current libraries used in Functest container are:

```
# cat requirements.pip
pyyaml==3.10
gitpython==1.0.1
python-neutronclient==2.6.0
python-novaclient==2.28.1
```

```
python-glanceclient==1.1.0
python-cinderclient==1.4.0
python-ceilometerclient==1.5.1
python-keystoneclient==1.6.0
virtualenv==1.11.4
pexpect==4.0
requests==2.8.0
robotframework==2.9.1
robotframework-requests==0.3.8
robotframework-sshlibrary==2.1.1
configObj==5.0.6
Flask==0.10.1
xmltodict==0.9.2
scp==0.10.2
paramiko==1.16.0
```

## 4.4 prepare_env.sh

This script can be adapted if you need to set up a specific environment before running the tests.

## 4.5 run_tests.sh

This script is used to run the tests. You must thus complete the cases with your own project:

```
;;
   "promise")
       info "Running PROMISE test case..."
       # TODO
   ;;
   "doctor")
       info "Running Doctor test..."
       python ${FUNCTEST_REPO_DIR}/testcases/features/doctor.py
   ;;
   "<your project>")
       info "Running <your project> test..."
       # your script that launchs your tests...
   ;;
```

And do not forget to update also the help line:

```
-t|--test         run specific set of tests
     <test_name>     one or more of the following separated by comma:
                        vping_ssh,vping_userdata,odl,rally,tempest,vims,onos,promise,ovno
```

## 4.6 config_funtest.yaml

This file is the main configuration file of Functest. You must add the references to your project:

```
general:
   directories:
       dir_repo_<your project>:      /home/opnfv/repos/<your project>
    repositories:
```

```
    # branch and commit ID to which the repos will be reset (HEAD)
    <your project>_branch:   master
    <your project>_commit:   latest
```

CHAPTER

FIVE

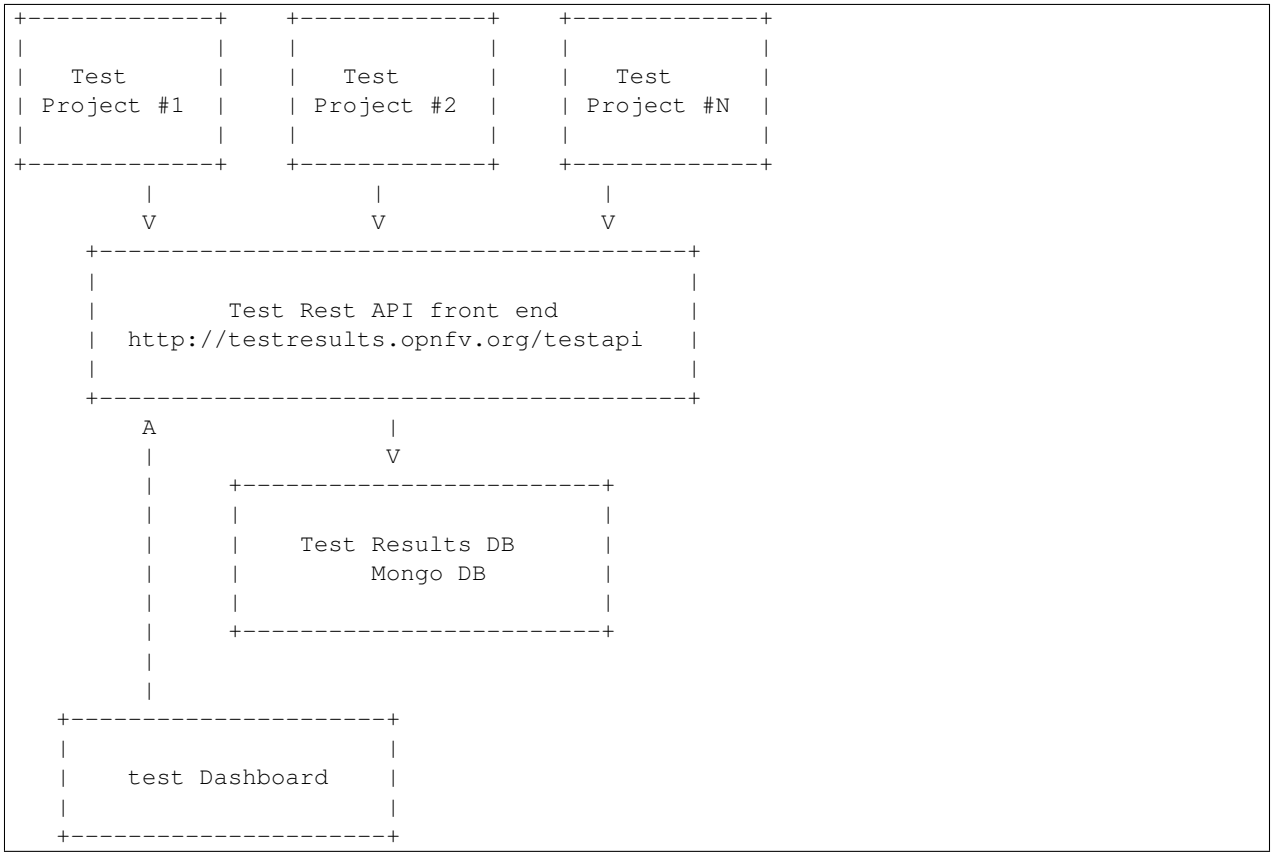
# TEST DASHBOARD & API

The OPNFV testing group created a test collection database to collect the test results from CI. Any test project running on any lab integrated in CI can push the results to this database. This database can be used afterwards to see the evolution of the tests and compare the results versus the installers, the scenario or the labs.

You can find more information about the dashboard from Testing Dashboard wiki page [3].

## 5.1 Overall Architecture

The Test result management in Brahmaputra can be summarized as follows:




The Test dashboard URL is: TODO LF A alternate Test dashboard has been realized to provide a view per installer and per scenario for Brahmaputra release:

```
http://testresults.opnfv.org/proto/brahma/
```

This Dashboard consumes the results retrieved thanks to the Test API.

## 5.2 Test API description

The Test API is used to declare pods, projects, test cases and test results. An additional method dashboard has been added to post-process the raw results. The data model is very basic, 4 objects are created:

- Pods
- Test projects
- Test cases
- Test results

Pods:

```
{
  "id": <ID>,
  "details": <URL description of the POD>,
  "creation_date": YYYY-MM-DD HH:MM:SS ,
  "name": <The POD Name>,
  "mode": <metal or virtual>
},
```

Test project:

```
{
  "id": <ID>,
  "name": <Name of the Project>,
  "creation_date": "YYYY-MM-DD HH:MM:SS",
  "description": <Short description>
},
```

Test case:

```
{
  "id": <ID>,
  "name":<Name of the test case>,
  "creation_date": "YYYY-MM-DD HH:MM:SS",
  "description": <short description>,
  "url":<URL for longer description>
},
```

Test results:

```
{
  "_id": <ID,
  "project_name": <Reference to project>,
  "pod_name": <Reference to POD where the test was executed>,
  "version": <Scenario on which the test was executed>,
  "installer": <Installer Apex or Compass or Fuel or Joid>,
  "description": <Short description>,
  "creation_date": "YYYY-MM-DD HH:MM:SS",
  "case_name": <Reference to the test case>
  "details":{
```

```
    <- the results to be put here ->
}
```

For Brahmaputra, we got:

- 16 pods
- 18 projects
- 101 test cases

The projects and the test cases have been frozen in December. But all were not ready for Brahmaputra.

The API can described as follows:

Version:

| Method | Path | Description |
|---|---|---|
| GET | /version | Get API version |

Pods:

| Method | Path | Description |
|---|---|---|
| GET | /pods | Get the list of declared Labs (PODs) |
| POST | /pods | Declare a new POD Content-Type: application/json { “name”: “pod_foo”, “creation_date”: “YYYY-MM-DD HH:MM:SS” } |

Projects:

| Method | Path | Description |
|---|---|---|
| GET | /test_projects | Get the list of test projects |
| GET | /test_projects/{project} | Get details on {project} |
| POST | /test_projects | Add a new test project Content-Type: application/json { “name”: “project_foo”, “description”: “whatever you want” } |
| PUT | /test_projects/{project} | Update a test project Content-Type: application/json { <the field(s) you want to modify> } |
| DELETE | /test_projects/{project} | Delete a test project |

Test cases:

| Method | Path | Description |
|---|---|---|
| GET | /test_projects/{project}/cases | Get the list of test cases of {project} |
| POST | /test_projects/{project}/cases | Add a new test case to {project} Content-Type: application/json { “name”: “case_foo”, “description”: “whatever you want” “creation_date”: “YYYY-MM-DD HH:MM:SS” “url”: “whatever you want” } |
| PUT | /test_projects/{project}/case_name={case} | Modify a test case of {project} Content-Type: application/json { <the field(s) you want to modify> } |
| DELETE | /test_projects/{project}/case_name={case} | Delete a test case |

Test Results:

| Method | Path | Description |
|---|---|---|
| GET | /results/project={project} | Get the test results of {project} |
| GET | /results/case={case} | Get the test results of {case} |
| GET | /results?pod={pod} | get the results on pod {pod} |
| GET | /results?installer={inst} | Get the test results of installer {inst} |
| GET | /results?version={version} | Get the test results of scenario {version}. Initially the version param was reflecting git version, in Functest it was decided to move to scenario |
| GET | /results?project={project} &case={case} &version={scenario} &installer={installer} &pod={pod} &period={days} | Get all the results of the test case {case} of the project {project} with version {scenario} installed by {installer} on POD {pod} stored since {days} days {project_name} and {case_name} are mandatory, the other parameters are optional. |
| POST | /results | Add a new test results Content-Type: application/json { “project_name”: “project_foo”, “case_name”: “case_foo”, “pod_name”: “pod_foo”, “installer”: “installer_foo”, “version”: “scenario_foo”, “details”: <your results> } |

Dashboard:

| Method | Path | Description |
|---|---|---|
| GET | /dashboard? &project={project} &case={case} &version={scenario} &installer={installer} &pod={pod} &period={days} | Get all the dashboard ready results of {case} of the project {project} version {scenario} installed by {installer} on POD {pod} stored since {days} days {project_name} and {case_name} are mandatory, the other parameters are optional. |

The results with dashboard method are post-processed from raw results. Please note that dashboard results are not stored. Only raw results are stored.

CHAPTER

SIX

# HOW TO PUSH YOUR RESULTS INTO THE TEST DATABASE

The test database is used to collect test results. By default it is enabled only in Continuous Integration. The architecture and associated API is described in [2]. If you want to push your results from CI, you just have to use the API at the end of your script.

You can also reuse a python function defined in functest_utils.py:

```
def push_results_to_db(db_url, case_name, logger, pod_name,version, payload):
  """
  POST results to the Result target DB
  """
  url = db_url + "/results"
  installer = get_installer_type(logger)
  params = {"project_name": "functest", "case_name": case_name,
            "pod_name": pod_name, "installer": installer,
            "version": version, "details": payload}

  headers = {'Content-Type': 'application/json'}
  try:
      r = requests.post(url, data=json.dumps(params), headers=headers)
      if logger:
          logger.debug(r)
      return True
  except Exception, e:
      print "Error [push_results_to_db('%s', '%s', '%s', '%s', '%s')]:" \
          % (db_url, case_name, pod_name, version, payload), e
      return False
```

CHAPTER

SEVEN

# REFERENCES

OPNFV main site: opnfvmain.

OPNFV functional test page: opnfvfunctest.

IRC support chan: #opnfv-testperf