Tobiko

Release 0.7.2-26-g2945010

Tobiko's Team

Apr 23, 2024

CONTENTS

CHAPTER

ONE

TEST BIG CLOUD OPERATIONS

Tobiko is an OpenStack testing framework focusing on areas mostly complementary to Tempest. While Tempest main focus has been testing OpenStack rest APIs, the main Tobiko focus is to test OpenStack system operations while "simulating" the use of the cloud as the final user would.

Tobiko's test cases populate the cloud with workloads such as Nova instances; they execute disruption operations such as services/nodes restart; finally they run test cases to validate that the cloud workloads are still functional.

Tobiko's test cases can also be used, for example, for testing that previously created workloads are working right after OpenStack services update/upgrade operation.

CHAPTER

TWO

PROJECT REQUIREMENTS

Tobiko Python framework is being automatically tested with below Python versions:

- Python 3.8
- Python 3.9
- Python 3.10 (new)

and below Linux distributions:

- CentOS 9 / RHEL 8 (with Python 3.9)
- Ubuntu Focal (with Python 3.8)
- Ubuntu Jammy (with Python 3.10)

Tobiko has also been tested for development purposes with below OSes:

- OSX (with Python 3.6 to 3.10)

The Tobiko Python framework is being used to implement test cases. As Tobiko can be executed on nodes that are not part of the cloud to test against, this doesn't mean Tobiko requires cloud nodes have to run with one of above Python versions or Linux distributions.

There is also a Docker file that can be used to create a container for running test cases from any node that do support containers execution.

2.1 Main Project Goals

- To test OpenStack and Red Hat OpenStack Platform projects before they are released.
- To provide a Python framework to write system scenario test cases (create and test workloads).
- To verify previously created workloads are working fine after executing OpenStack nodes update/upgrade.
- To write white boxing test cases (to log to cloud nodes for internal inspection purpose).
- To write disruptive test cases (to simulate service disruptions like for example rebooting/interrupting a service to verify cloud reliability).
- To provide Ansible roles implementing a workflow designed to run an ordered sequence of test suites. For example a workflow could do below steps:
  - creates workloads;
  - run disruptive test cases (IE reboot OpenStack nodes or services);
  - verify workloads are still working.

The main use of these roles is writing continuous integration jobs for Zuul or other services like Jenkins (IE by using the Tobiko InfraRed plug-in).

- To provide tools to monitor and recollect the healthy status of the cloud as seen from user perspective (black-box testing) or from an inside point of view (white-box testing built around SSH client).

CHAPTER

THREE

REFERENCES

- Free software: Apache License, Version 2.0
- Documentation: https://tobiko.readthedocs.io/
- Release notes: https://docs.openstack.org/releasenotes/tobiko/
- Source code: https://opendev.org/x/tobiko
- Bugs: https://storyboard.openstack.org/#!/project/x/tobiko
- Code review: https://review.opendev.org/q/project:x/tobiko

3.1 Related projects

- OpenStack: https://www.openstack.org/
- Red Hat OpenStack Platform: https://www.redhat.com/en/resources/openstack-platform-datasheet
- Python: https://www.python.org/
- Testtools: https://github.com/testing-cabal/testtools
- Ansible: https://www.ansible.com/
- InfraRed: https://infrared.readthedocs.io/en/latest/
- DevStack: https://docs.openstack.org/devstack/latest/
- Zuul: https://docs.openstack.org/infra/system-config/zuul.html
- Jenkins: https://www.jenkins.io/

CHAPTER

FOUR

CONTENT

It includes the user, administrator and user guides. It assumes that you are already familiar with other OpenStack projects. If not, hop over to the OpenStack doc site.

For additional documentation you can also look at OpenStack wiki.

Source code for this documentation page is hosted at source tree.

4.1 User Guide

4.1.1 Quick Start

This document describes how to setup an environment and how to run test cases

See also

To install Tobiko inside a virutalenv please read *Install*.

To configure Tobiko please read *Configuration*.

To run Tobiko scenario test cases please look at *Tobiko Test Cases Execution Guide*.

Install test cases within a virtualenv

The safest way to run test cases is to do it within a Virtualenv. Here we are goint to see how to setup an environment with all test case dependencies.

In **RHEL**, **CentOS** or **Fedora** install the following packages:

```
sudo dnf install -y gcc git python3 python3-devel python3-pip which findutils
```

In **Debian** or **Ubuntu** install following packages:

```
sudo apt update
sudo apt install -y gcc git python3 python3-dev python3-pip
```

Ensure Pip is up-to-date:

```
python3 -m pip install --upgrade --user pip
```

Ensure Tox is installed and up-to-date:

```
python3 -m pip install --upgrade --user setuptools virtualenv wheel tox
```

Get source code using Git and enter into Tobiko source folder:

```
git clone https://opendev.org/x/tobiko.git
cd tobiko
```

Install remaining binary packages:

```
tools/install-bindeps.sh
```

Crate the virtual environment with Tox:

```
python3 -m tox -e py3 --notest
```

In case you want to activate the virtual environment you can then type:

```
. .tox/py3/bin/activate
```

At this point the environment should have all dependencies installed for running test cases.

Configure tobiko with tobiko.conf

Tobiko tries to load the '*tobiko.conf*' file from one of the below locations:

- current directory:

  ```
  ./tobiko.conf
  ```

- user home directory:

  ```
  ~/.tobiko/tobiko.conf
  ```

- system directory:

  ```
  /etc/tobiko/tobiko.conf
  ```

Configure Logging Options

Tobiko can configure a logging system to write messages to a log file. You can edit the below options in tobiko-conf to enable it as below:

```
[DEFAULT]
# Whenever to allow debugging messages to be written out or not
debug = true

# Name of the file where log messages will be appended.
log_file = tobiko.log

# The base directory used for relative log_file paths.
log_dir = .
```

The file 'tobiko.log' is the default file where test cases and the Python framework are going to write their logging messages. By setting debug as 'True' you ensure that messages with the lowest logging level are written there (DEBUG level). The log_file location specified above is relative to the tobiko.conf file location. In this example it is the Tobiko source files' directory itself.

Configure Tobiko Credentials

Tobiko needs to have Keystone credentials in order to run the OpenStack test cases. We are going to assume you are using one of the two OpenStack distributions supported by Tobiko:

- DevStack
- TripleO

Get credentials from a DevStack host

Copy the clouds.yaml file from your remote cloud to any one of the below locations:

- Tobiko source files directory
- ~/.config/openstack
- /etc/openstack

The clouds.yaml file contains valid Keystone credentials.

You can copy the file in the following way:

```
ssh <... connection options here ...> cat /etc/openstack/clouds.yaml > clouds.yaml
```

Get credentials from a TripleO undercloud host

Tobiko test cases will be able to setup some type of SSH tunneling to be able to reach the remote cloud, but for achieving it you are required to be able to connect to a remote SSH server that is able to connect the OpenStack services and hosts. We will refer to that server as the SSH proxy host.

Tobiko test cases will execute some commands on the SSH proxy host (like ping, nc, curl, etc). Those commands need to have direct connectivity to target cloud.

Test cases will use Python REST API clients configured to make HTTP requests coming out from such SSH server (mainly by using nc command) or SSH server direct connect feature.

Test cases will make all SSH connection to cloud nodes by using this SSH proxy host.

To resume the purpose of the SSH proxy, all network packages sent by Tobiko test cases to the tested cloud will come from the SSH proxy host, while all Tobiko test cases will be executed from the developer workstation.

SetUp SSH public key to connect to remote cloud

First of all we need to make sure we can connect to the SSH proxy server without requiring any password. We therefore need to have a local SSH key pair to be used by tobiko. This key by default is the same default one used by openSSH client: - default SSH private key filename: *~/.ssh/id_rsa* - default SSH public key filename: *~/.ssh/id_rsa.pub*

Note: In case that you already have a public key which does not require a password (it requires an empty passphrase), you can skip the ssh keypair creation (continue with *defining your ssh variables* **).**

In case that you don't, to avoid having problems with other uses of the same file, let's instead create our SSH key pair only for Tobiko in a sub-folder near to your tobiko.conf

Make sure you run the following commands in the tobiko directory. Ensure we do have this key pair on your workstation by typing:

```
mkdir -p .ssh
chmod 700 .ssh
ssh-keygen -v -f .ssh/id -N ''
chmod 600 .ssh/id .ssh/id.pub
```

Define the below SSH variables to later connect to your SSH server:

```
SSH_HOST=<your-ssh-proxy-address>
SSH_USERNAME=<your-ssh-proxy-user>
```

For example:

```
SSH_HOST=seal100.your.domain
SSH_USERNAME=root
```

Copy your SSH public key to your remote server:

```
ssh-copy-id -i .ssh/id "${SSH_USERNAME}@${SSH_HOST}"
```

Make sure the SSH key pair is working:

```
ssh -i .ssh/id "${SSH_USERNAME}@${SSH_HOST}" hostname
```

Now let's make sure Tobiko test cases will use the SSH key pair to connect to your SSH remote host. Add the following lines to tobiko.conf file:

```
[ssh]
proxy_jump = SSH_USERNAME@SSH_HOST
```

For example:

```
[ssh]
proxy_jump = root@seal100.your.domain
#proxy_jump = root@seal99.your.domain
#proxy_jump = root@seal98.your.domain
```

Tip: You could have multiple hosts in your tobiko.conf [ssh] section, where the ones you are not currently using are commented (as appear above). Moving your tobiko tests from one host to another will be as easy as commenting the host you are stop using and uncommenting the one you are start using (remember to copy your SSH key to your other remote hosts as well).

Create a virtual environment with tox

To execute commands from a virtualenv created by Tox, add the virtualenv name to tox.ini envlist variable in the following way (in this example, the virtual environment's name is 'venv'):

```
[tox]

envlist = other_environment_variables,venv
```

Run your commands as below:

```
tox -e venv -- <your-commands>
```

For example:

```
tox -e venv -- openstack network list
```

Run Test Cases

The next section is a quick guide about running some test cases. For more information, please see our *Tobiko Test Cases Execution Guide*

Before running test cases, make sure you configure tobiko logging according to your needs.

Note: Unlike other testing frameworks, **Tobiko does not delete its resources after test cases finish their execution**. You may clean up tobiko workloads after the execution manually, for example heat stacks and glance images.

Run Scenario Test Cases

Scenario test cases are used to create workloads that simulate real-world use of OpenStack. They create networks, virtual machines, ports, routers, etc. They also validate that these workloads are functioning.

Running Tobiko scenario test cases using Tox (may take some minutes to complete):

```
tox -e scenario
```

Scenario test cases are also used to check that previously created resources are still up and working as expected. To ensure test cases will not create those resources again we can set *TOBIKO_PREVENT_CREATE* environment variable before re-running test cases:

```
TOBIKO_PREVENT_CREATE=yes tox -e scenario
```

Run Disruptive Test Cases

Disruptive (or faults) test cases are used for testing that after inducing some critical disruption to the operation of the cloud, the services can get back to the expected state after a while. To execute them you can type:

```
tox -e faults
```

The faults induced by these test cases could be cloud nodes reboot, OpenStack services restart, virtual machines migrations, etc.

Please note that while scenario test cases are being executed in parallel (to speed up test case execution), disruptive test cases are only executed sequentially. This is because the operations executed by such cases could break some functionality for a short time and alter the regular state of the system which may be assumed by other test cases to be executed.

Run the Tobiko Workflow

Scenario and disruptive test cases, which are being executed in a specific sequence, could be used to uncover more issues with the cloud than disruptive test cases alone.

- First ensure there are workloads properly running by running scenario test cases:

  ```
  tox -e scenario
  ```

Note

As second step we may, instead, update or upgrade OpenStack nodes.

- Next we could execute disruptive test cases to "stress" the cloud:

  ```
  tox -e faults
  ```

- Finally we might re-run scenario test cases to check that everything is still running as expected:

  ```
  TOBIKO_PREVENT_CREATE=yes tox -e scenario
  ```

Test Cases Report Files

After executing test cases we can view the results in greater detail via a small set of files:

- **test_results.html**: A user-browsable HTML view of test case results.
- **test_results.log**: A log file with logging traces collected from every individual test case.
- **test_results.subunit**: The original subunit binary file generated by test runner.
- **test_results.xml**: An XML Junit file to be used, for example, to show test cases result by Jenkins CI server.

The names of the above files can be changed from the default value (*test_results*) to a custom one by setting the *TOX_REPORT_NAME* environment variable.

Legend

{toxinidir} stands for the Tobiko source files directory.

{envname} is the name of the Tox enviroment to be executed (IE scenario, faults, etc.)

The above files are saved into a folder that can be specified with *TOX_REPORT_DIR* environment variable.

By default the full path of the report directory is made from the below:

```
{toxinidir}/report/{envname}
```

4.1.2 Install

This document describes how to setup an environment for running test cases

See also

For a quick and simpler start you can jump to the *Quick Start*.

To configure Tobiko please read *Configuration*.

To run Tobiko scenario test cases please look at *Tobiko Test Cases Execution Guide*.

Install test cases within a virtualenv

The safest way to run test cases is to do it within a Virtualenv. Here we are goint to see how to setup an environment with all test case dependencies.

In **RHEL**, **CentOS** or **Fedora** install the following packages:

```
sudo dnf install -y gcc git python3 python3-devel python3-pip which findutils
```

In **Debian** or **Ubuntu** install following packages:

```
sudo apt update
sudo apt install -y gcc git python3 python3-dev python3-pip
```

Ensure Pip is up-to-date:

```
python3 -m pip install --upgrade --user pip
```

Ensure Tox is installed and up-to-date:

```
python3 -m pip install --upgrade --user setuptools virtualenv wheel tox
```

Get source code using Git and enter into Tobiko source folder:

```
git clone https://opendev.org/x/tobiko.git
cd tobiko
```

Install remaining binary packages:

```
tools/install-bindeps.sh
```

Crate the virtual environment with Tox:

```
python3 -m tox -e py3 --notest
```

In case you want to activate the virtual environment you can then type:

```
. .tox/py3/bin/activate
```

At this point the environment should have all dependencies installed for running test cases.

What's Next

To know how to configure Tobiko please read *Configuration*.

4.1.3 Configuration

This document describes how to configure Tobiko.

See also

For a quick and simpler start you can jump to the *Quick Start*.

To install Tobiko inside a virutalenv please read *Install*.

To run Tobiko scenario test cases please look at *Tobiko Test Cases Execution Guide*.

Configure Tobiko Framework

In order to make sure Tobiko tools can connect to OpenStack services via Rest API configuration parameters can be passed either via environment variables or via an INI configuration file (referred here as tobiko-conf). Please look at *Authentication Methods* for more details.

tobiko.conf

Tobiko tries to load the '*tobiko.conf*' file from one of the below locations:

- current directory:

  ```
  ./tobiko.conf
  ```

- user home directory:

  ```
  ~/.tobiko/tobiko.conf
  ```

- system directory:

  ```
  /etc/tobiko/tobiko.conf
  ```

Configure Logging

Tobiko can configure a logging system to write messages to a log file. You can edit the below options in tobiko-conf to enable it as below:

```
[DEFAULT]
# Whenever to allow debugging messages to be written out or not
debug = true

# Name of the file where log messages will be appended.
log_file = tobiko.log

# The base directory used for relative log_file paths.
log_dir = .
```

The file 'tobiko.log' is the default file where test cases and the Python framework are going to write their logging messages. By setting debug as 'True' you ensure that messages with the lowest logging level are written there (DEBUG level). The log_file location specified above is relative to the tobiko.conf file location. In this example it is the Tobiko source files' directory itself.

Configure Tobiko Credentials

Tobiko needs to have Keystone credentials in order to run the OpenStack test cases. We are going to assume you are using one of the two OpenStack distributions supported by Tobiko:

- DevStack
- TripleO

Get credentials from a DevStack host

Copy the clouds.yaml file from your remote cloud to any one of the below locations:

- Tobiko source files directory
- ~/.config/openstack
- /etc/openstack

The clouds.yaml file contains valid Keystone credentials.

You can copy the file in the following way:

```
ssh <... connection options here ...> cat /etc/openstack/clouds.yaml > clouds.yaml
```

Get credentials from a TripleO undercloud host

Tobiko test cases will be able to setup some type of SSH tunneling to be able to reach the remote cloud, but for achieving it you are required to be able to connect to a remote SSH server that is able to connect the OpenStack services and hosts. We will refer to that server as the SSH proxy host.

Tobiko test cases will execute some commands on the SSH proxy host (like ping, nc, curl, etc). Those commands need to have direct connectivity to target cloud.

Test cases will use Python REST API clients configured to make HTTP requests coming out from such SSH server (mainly by using nc command) or SSH server direct connect feature.

Test cases will make all SSH connection to cloud nodes by using this SSH proxy host.

To resume the purpose of the SSH proxy, all network packages sent by Tobiko test cases to the tested cloud will come from the SSH proxy host, while all Tobiko test cases will be executed from the developer workstation.

SetUp SSH public key to connect to remote cloud

First of all we need to make sure we can connect to the SSH proxy server without requiring any password. We therefore need to have a local SSH key pair to be used by tobiko. This key by default is the same default one used by openSSH client: - default SSH private key filename: *~/.ssh/id_rsa* - default SSH public key filename: *~/.ssh/id_rsa.pub*

Note: In case that you already have a public key which does not require a password (it requires an empty passphrase), you can skip the ssh keypair creation (continue with *defining your ssh variables* **).**

In case that you don't, to avoid having problems with other uses of the same file, let's instead create our SSH key pair only for Tobiko in a sub-folder near to your tobiko.conf

Make sure you run the following commands in the tobiko directory. Ensure we do have this key pair on your workstation by typing:

```
mkdir -p .ssh
chmod 700 .ssh
ssh-keygen -v -f .ssh/id -N ''
chmod 600 .ssh/id .ssh/id.pub
```

Define the below SSH variables to later connect to your SSH server:

```
SSH_HOST=<your-ssh-proxy-address>
SSH_USERNAME=<your-ssh-proxy-user>
```

For example:

```
SSH_HOST=seal100.your.domain
SSH_USERNAME=root
```

Copy your SSH public key to your remote server:

```
ssh-copy-id -i .ssh/id "${SSH_USERNAME}@${SSH_HOST}"
```

Make sure the SSH key pair is working:

```
ssh -i .ssh/id "${SSH_USERNAME}@${SSH_HOST}" hostname
```

Now let's make sure Tobiko test cases will use the SSH key pair to connect to your SSH remote host. Add the following lines to tobiko.conf file:

```
[ssh]
proxy_jump = SSH_USERNAME@SSH_HOST
```

For example:

```
[ssh]
proxy_jump = root@seal100.your.domain
#proxy_jump = root@seal99.your.domain
#proxy_jump = root@seal98.your.domain
```

Tip: You could have multiple hosts in your tobiko.conf [ssh] section, where the ones you are not currently using are commented (as appear above). Moving your tobiko tests from one host to another will be as easy as commenting the host you are stop using and uncommenting the one you are start using (remember to copy your SSH key to your other remote hosts as well).

Authentication Methods

Tobiko uses OpenStack client to connect to OpenStack services.

Skipping resources creation

In some cases, for example when Tobiko is run after an upgrade of a cloud, it may be expected that resources used for tests have already been created. Tobiko should not try to create resources than and just run tests using what has already been created. To configure Tobiko to not create test resources, the environment variable `TOBIKO_PREVENT_CREATE` can be used:

```
export TOBIKO_PREVENT_CREATE=True
```

If this is set to `True` or `1` then Tobiko will not try to create resources like VMs, networks, routers, or images and just run validations of what exists in the cloud already.

What's Next

To know how to run Tobiko scenario test cases you can look at *Tobiko Test Cases Execution Guide*

4.1.4 Tobiko Test Cases Execution Guide

This document describes how to execute Tobiko test cases.

See also

For a quick and simpler start you can jump to the *Quick Start*.

To install Tobiko inside a virtualenv please read *Install*.

To configure Tobiko please read *Configuration*.

Prepare Your System

Before running Tobiko test cases, you need to be sure you are doing it from Tobiko source files folder and that you have activated a virtualenv where Tobiko is, and its requirements are installed. Please refer to *Install* and *Configuration* to know how to setup your system before running test cases.

Prepare some logging (Optional and recommended)

To see if we are now being able to execute Tobiko test cases, please open **a new terminal** and keep it open, where you could watch tobiko.log receive logs on real time. Change directory to reach the directory where tobiko.log file is and run the following command:

```
tail -F tobiko.log
```

Run Tobiko Test Cases

Run Tobiko specific test cases

In the first terminal, execute some Tobiko test cases as below:

To run test cases you need a test runner able to execute Python test cases. Test cases delivered with Tobiko has been tested using stestr

From the Tobiko source folder you can run scenario test cases using the below command:

```
pytest tobiko/tests/scenario/
```

You could also use tox to run test cases:

```
tox -e <environment_variable> -- path/to/test/module
```

For example:

```
tox -e scenario -- tobiko/tests/scenario/neutron/test_router.py
```

Note that with tox, the <environment_variable> should match the directory where the test is (if the test is inside the 'scenario' directory, the environment variable has to be scenario).

You can also run only a class of test cases by running:

```
tox -e <tox-env-list> -- path/to/test/module::class
```

You can run only a specific test case (a method) by running:

```
tox -e <tox-env-list> -- path/to/test/module::class::test_case
```

Run Scenario Test Cases

Scenario test cases are used to create workloads that simulate real-world use of OpenStack. They create networks, virtual machines, ports, routers, etc. They also validate that these workloads are functioning.

Running Tobiko scenario test cases using Tox (may take some minutes to complete):

```
tox -e scenario
```

Scenario test cases are also used to check that previously created resources are still up and working as expected. To ensure test cases will not create those resources again we can set *TOBIKO_PREVENT_CREATE* environment variable before re-running test cases:

```
TOBIKO_PREVENT_CREATE=yes tox -e scenario
```

Running Disruptive Test Cases

Disruptive (or faults) test cases are used for testing that after inducing some critical disruption to the operation of the cloud, the services can get back to the expected state after a while. To execute them you can type:

```
tox -e faults
```

The faults induced by these test cases could be cloud nodes reboot, OpenStack services restart, virtual machines migrations, etc.

Please note that while scenario test cases are being executed in parallel (to speed up test case execution), disruptive test cases are only executed sequentially. This is because the operations executed by such cases could break some functionality for a short time and alter the regular state of the system which may be assumed by other test cases to be executed.

Run the Tobiko Workflow

Scenario and disruptive test cases, which are being executed in a specific sequence, could be used to uncover more issues with the cloud than disruptive test cases alone.

- First ensure there are workloads properly running by running scenario test cases:

  ```
  tox -e scenario
  ```

Note

As second step we may, instead, update or upgrade OpenStack nodes.

- Next we could execute disruptive test cases to "stress" the cloud:

  ```
  tox -e faults
  ```

- Finally we might re-run scenario test cases to check that everything is still running as expected:

  ```
  TOBIKO_PREVENT_CREATE=yes tox -e scenario
  ```

Test Cases Report Files

After executing test cases we can view the results in greater detail via a small set of files:

- **test_results.html**: A user-browsable HTML view of test case results.
- **test_results.log**: A log file with logging traces collected from every individual test case.
- **test_results.subunit**: The original subunit binary file generated by test runner.
- **test_results.xml**: An XML Junit file to be used, for example, to show test cases result by Jenkins CI server.

The names of the above files can be changed from the default value (*test_results*) to a custom one by setting the *TOX_REPORT_NAME* environment variable.

Legend

{toxinidir} stands for the Tobiko source files directory.

{envname} is the name of the Tox enviroment to be executed (IE scenario, faults, etc.)

The above files are saved into a folder that can be specified with *TOX_REPORT_DIR* environment variable.

By default the full path of the report directory is made from the below:

```
{toxinidir}/report/{envname}
```

4.2 Project Contributor Guide

See also

To configure your workstation to develop tobiko test cases please read *Tobiko Contributor Guide*.

4.2.1 Tobiko Contributor Guide

Document Overview

This document describes how to configure a developer workstation to run Tobiko test cases against a remote OpenStack cloud

See also

To just execute Tobiko test cases please read *Quick Start*.

To install Tobiko inside a virutalenv please read *Install*.

To configure Tobiko please read *Configuration*.

To run Tobiko scenario test cases please look at *Tobiko Test Cases Execution Guide*.

This tutorial will guide you to configure a developer workstation to be able to run Tobiko test cases locally without requiring direct network connectivity to a remote OpenStack cloud (DevStack based or TripleO based) so that the edit-try-debug cycle should get shorter and simpler than by editing test cases on one host that is not the same where test cases are being executed:

```
[ test cases host ] - SSH -> [SSH proxy host] - IP -> [OpenStack cloud]
```

Install Dependencies

Install Basic Python Packages

Make sure Git and Python 3 are installed on your system.

For instance on RedHat Linux / Fedora:

```
sudo dnf install -y git python3 which
```

Check your Python 3 version is 3.8 or greater:

```
python3 --version
```

Make sure pip is installed and up-to date:

```
curl https://bootstrap.pypa.io/get-pip.py | sudo python3
```

Check installed Pip version:

```
python3 -m pip --version
```

Make sure basic Python packages are installed and up-to-date:

```
sudo python3 -m pip install --upgrade setuptools wheel virtualenv tox six
```

Check installed Tox version:

```
tox --version
```

Clone the Tobiko repository

Clone the Tobiko repository using Git:

```
git clone https://opendev.org/x/tobiko.git
cd tobiko
```

Install Missing Binary Packages

Install required binary packages:

```
tools/install-bindeps.sh
```

Configure Logging Options

Test cases load most of the configuration parameters from an INI configuration file, typically found at one of the following locations:

- ./tobiko.conf (Tobiko source files directory)
- ~/.tobiko/tobiko.conf
- /etc/tobiko/tobiko.conf

Create it in the Tobiko source directory with the following (or as your preferences). Example:

```
[DEFAULT]
debug = true
log_file = tobiko.log
```

The file 'tobiko.log' is the default file where test cases and the Python framework are going to write their logging messages. By setting debug as 'true' you ensure that messages with the lowest logging level are written there (DEBUG level). The log_file location specified above is relative to the tobiko.conf file location. In this example it is the Tobiko source files directory itself because in case of a relative path the directory where the tobiko.conf file is used as current directory.

SetUp SSH public key to connect to remote cloud

Tobiko test cases will be able to setup some type of SSH tunneling to be able to reach the remote cloud, but for archiving it you are required to be able to connect to a remote SSH server that is able to connect to the OpenStack services and hosts. We will call that server here as the SSH proxy host.

Tobiko test cases will execute some commands on the SSH proxy host (like ping, nc, curl, etc.) as soon as these command need to have direct connectivity to target cloud.

Test case will use Python REST API clients configured to make HTTP requests coming out from such SSH server (mainly by using nc command) or SSH server direct connect feature.

Test cases will make all SSH connection to cloud nodes by using this SSH proxy host

To resume the purpose of the SSH proxy, all network packages sent by Tobiko test cases to the tested cloud will come from the SSH proxy host, while all Tobiko test cases will be executed from the developer workstation.

In order to archive it, first of all we need to make sure we can connect to the SSH proxy server without requiring any password. We therefore need to have a local SSH key pair to be used by tobiko. This key by default is the same default one used by openSSH client: - default SSH private key filename: *~/.ssh/id_rsa* - default SSH public key filename: *~/.ssh/id_rsa.pub* To avoid having problems with other uses of the same file, lets instead create our SSH key pair only for Tobiko in a sub-folder near to your tobiko.conf

Ensure we do have this key pair on your workstation by typing:

```
mkdir -p .ssh
chmod 700 .ssh
```

(continues on next page)

(continued from previous page)

```
ssh-keygen -v -f .ssh/id -N ''
chmod 600 .ssh/id .ssh/id.pub
```

Please note in case you already have this pair of files created before that, the key pair must have an empty passphrase. That means the SSH client will never ask you a password to connect to a remote server using that key pair.

Define below variables to later connect to your SSH server:

```
SSH_HOST=<your-ssh-proxy-address>
SSH_USERNAME=<your-ssh-proxy-user>
```

Ensure you can connect to the remote SSH server using our new key pair without a password:

```
ssh-copy-id -i .ssh/id "${SSH_USERNAME}@${SSH_HOST}"
```

Check the SSH key pair is working:

```
ssh -i .ssh/id "${SSH_USERNAME}@${SSH_HOST}" hostname
```

Create '.ssh/config' file with SSH proxy connection parameters:

```
echo "
Host ssh-proxy ${SSH_HOST}
    IdentityFile .ssh/id
    IdentitiesOnly yes
    HostName ${SSH_HOST}
    User ${SSH_USERNAME}
    StrictHostKeyChecking no
    PasswordAuthentication no
    UserKnownHostsFile /dev/null
" > .ssh/config
chmod 600 .ssh/config
```

Check the SSH config file is working:

```
ssh -F .ssh/config ssh-proxy hostname
```

Now let tell Tobiko test cases to use these SSH key pair and to connect to your SSH remote host by editing tobiko.conf file:

```
[ssh]
proxy_jump = ssh-proxy
config_files = .ssh/config
```

We also want to tell Tobiko to use the same key pair to connect to VMs created by Tobiko test cases:

```
[nova]
key_file = .ssh/id
```

Configure Tobiko Credentials

In order to run the OpenStack test cases Tobiko needs to have Keystone credentials. To make our life simpler we are going to assume you are using one of the two OpenStack distributions supported by Tobiko:

- DevStack
- TripleO

Get credentials from a DevStack host

Get the clouds.yaml file from a remote DevStack host:

```
ssh <... connection options here ...> cat /etc/openstack/clouds.yaml > clouds.yanl
```

If your SSH proxy host configured before is one of your DevStack cloud hosts then you can type:

```
ssh -F .ssh/config ssh-proxy cat /etc/openstack/clouds.yaml > clouds.yaml
```

Edit your tobiko.conf file to pick your DevStack based cloud:

```
[keystone]
cloud_name = devstack-admin
clouds_file_names = clouds.yaml
```

Get credentials from TripleO undercloud host

First of all let discover undercloud host IP by pinging it from ssh-proxy host:

```
UNDERCLOUD_IP=$(
    ssh -F .ssh/config ssh-proxy ping -c 1 undercloud-0 |
    awk '/^PING/{gsub(/\(|\)/,""); print $3}')
echo $UNDERCLOUD_IP
```

Tobiko should be able to get credentials directly from such undercloud node but it must know the address of the undercloud host, so we must edit tobiko.conf to let it know:

```
[tripleo]
undercloud_ssh_hostname=<undercloud-host-address>
```

Run Tobiko test cases

Running Scenario Test Cases

To see if we are now able to execute Tobiko test cases please keep open a new terminal where to watch tobiko.log file on the same folder of tobiko.conf file:

```
tail -F tobiko.log
```

Then in the first terminal execute some Tobiko test cases as below:

```
tox -v -e scenario -- -v tobiko/tests/scenario/neutron/test_floating_ip.
↪py::FloatingIPTest
```

Scenario test cases are used to create workloads that simulate real-world use of OpenStack. They create networks, virtual machines, ports, routers, etc. They also test validate that these workloads functioning.

Running Tobiko scenario test cases using Tox (may take several minutes to complete):

```
tox -e scenario
```

Listing Tobiko Workloads

To manage workloads created by Tobiko please log to remote cloud node

Listing Tobiko Workloads on DevStack

To list workloads generated by tobiko you can use glance and heat CLI from the SSH proxy host node:

```
ssh -i .ssh/config ssh-proxy
export OS_CLOUD=devstack-admin
openstack image list
openstack stack list
```

Listing Tobiko Workloads on DevStack

To list workloads generated by tobiko you can use glance and heat CLI from the undercloud-0 host node:

```
ssh -F .ssh/config ssh-proxy -t ssh stack@undercloud-0
source overcloudrc
openstack image list
openstack stack list
```

Verify Tobiko Workloads

Scenario test cases are also used to check that previously created resources are still up and working as expected. To ensure test cases will not create those resources again we can set *TOBIKO_PREVENT_CREATE* environment variable before re-running test cases:

```
TOBIKO_PREVENT_CREATE=yes tox -e scenario -- -v tobiko/tests/scenario/neutron/test_
↪floating_ip.py::FloatingIPTest
```

Cleaning Up Tobiko Workloads

Once Tobiko test cases have been executed, we may want to clean up all workloads remaining on the cloud so that we restore it to its original state.

Cleaning Up Heat Stacks

Because Tobiko is using Heat stacks for orchestrating the creation of most of the resources, deleting all stacks created with Tobiko will clean up almost all resources:

```
openstack stack list -f value -c ID | xargs openstack stack delete
```

Cleaning Up Glance Images

Because Heat doesn't support creation of Glance images, Tobiko implements some specific fixtures to download images from the Web and upload them to the Glance service:

```
openstack image list -f value -c ID | xargs openstack image delete
```

Running Disruptive Test Cases

Disruptive test cases are used for testing that after inducing some critical disruption to the operation of the cloud, the services return working as expected after a while. To execute them you can type:

```
tox -e faults
```

The faults induced by these test cases could be cloud nodes reboot, OpenStack services restart, virtual machines migrations, etc.

Please note that while scenario test cases are being executed in parallel (to speed up test case execution), disruptive test case are only executed sequentially. This is because the operations executed by such cases could break some functionality for a short time and alter the regular state of the system which may be assumed by other test cases to be executed.

Running the Tobiko Workflow

Scenario and disruptive test cases, being executed in a specific sequence could be used to uncover more issues with the cloud than disruptive test cases alone.

- First ensure there are workloads properly running by running scenario test cases:

  ```
  tox -e scenario
  ```

Note

As second step we may, instead, update or upgrade OpenStack nodes.

- Next we could execute disruptive test cases to "stress" the cloud:

```
tox -e faults
```

- Finally we might re-run scenario test cases to check that everything is still running as expected:

```
TOBIKO_PREVENT_CREATE=yes tox -e scenario
```

4.2.2 Tobiko Reviewer Guide

Document Overview

This document describes how to review changes proposed for Tobiko. You can find here information about where patches should be reviewed and what are some basic rules during the review and to merge patches.

See also

Tobiko uses OpenStack's gerrit to review patches. You can find many additional details about using Gerrit in the OpenStack Contributor Guide.

Reviewing changes

Every change proposed to one of the Tobiko repositories needs to be reviewed by someone else. Everyone who have account in the Openstack gerrit created can review every change there. As a reviewer You can comment on the proposed change and give one of the votes:

1. `+1` - when You think that change is good to be merged and don't need additional work,
2. `-1` - when change needs some additional work, You shouldn't give just `-1` to the change without any comments what is wrong in Your opinion there.
3. `0` - when You simply have some comment but don't want to give neither `+1` nor `-1` to the change.

Core reviewers

There is also Core reviewers team. Reviewers who are members of this team can additionally vote on the change with:

1. `+2` - when change is ready to be merged according to the core reviewer,
2. `-2` - which means `Do not merge that change` - it shouldn't be used often and only for good reason. That vote will not dissapear when new patch set will be proposed by the change owner. It can be only removed by the reviewer who gave it,
3. `+W` - which means that patch is approved and is going to be merged by Zuul after it will pass CI jobs.

To give `+W` to the patch and to merge it patch should have at least one `+2` vote for someone else than patch owner. If the change is trivial, like e.g. fixed typo, and made by one of the Tobiko core reviewers, it can be approved directly by the owner of the change to be merged quickly. Core reviewers shouldn't use that exception too much. General rule should always be that someone else should review the change, vote with `+2` and approve change with `+W`.

4.2.3 How to setup a tobiko workstation

This guide should help contributors to configure their workstation, so it would be easier for them to write tobiko test cases.

Configuring local test running and debugging

Clone the tobiko repository:

```
git clone https://opendev.org/x/tobiko.git
```

Create a tobiko.conf inside the tobiko directory, which will have:

```
[DEFAULT]

debug = true
```

Inside the Tobiko directory, run:

```
tox -e py3
```

Copy the path you see in the first line (it should contain something similar to *tobiko/.tox/py3*)

On Pycharm:

- Create a new (pycharm) project with *Location = <the tobiko path>*
- In interpreter options tab, choose "*New environment using Virtualenv*", with Location = *<tobiko-path>/.pycharm*, and then click on "*Create -> Create from Existing Sources*".
- Enter preferences (ctrl+alt+s in Linux)
- **In Project Interpreter tab:**
  - Click * -> Add
  - Choose "*Existing Environment*"
  - In the Interpreter field, paste the path you copied after running the tox command, and add the "/bin/python3" suffix.

    Example: the path *~/tobiko/.tox/py3/bin/python3* might be used, as tox provides the directory *~/tobiko/.tox/py3*, and we add */bin/python3*
  - In Python Integrated Tools tab, under *Testing* section, set the *Default test runner* to be "*pytest*"

Now verify your environment has the configuration options mentioned above by doing the following: Find tobiko/tests/unit/test_config.py in the left project window -> right click -> *Run pytest in test config*.

All tests should pass. You could also debug tests by setting a breakpoint.

Configuring proxy jump

Make sure you can ssh to your remote host without a password:

Define the below SSH variables to later connect to your SSH server:

```
SSH_HOST=<your-ssh-proxy-address>
SSH_USERNAME=<your-ssh-proxy-user>
```

For example:

```
SSH_HOST=seal100.your.domain
SSH_USERNAME=root
```

Copy your SSH public key to your remote server:

```
ssh-copy-id -i .ssh/id "${SSH_USERNAME}@${SSH_HOST}"
```

Make sure the SSH key pair is working:

```
ssh -i .ssh/id "${SSH_USERNAME}@${SSH_HOST}" hostname
```

Now let's make sure Tobiko test cases will use the SSH key pair to connect to your SSH remote host. Add the following lines to tobiko.conf file:

```
[ssh]
proxy_jump = SSH_USERNAME@SSH_HOST
```

For example:

```
[ssh]
proxy_jump = root@seal100.your.domain
#proxy_jump = root@seal99.your.domain
#proxy_jump = root@seal98.your.domain
```

Tip: You could have multiple hosts in your tobiko.conf [ssh] section, where the ones you are not currently using are commented (as appear above). Moving your tobiko tests from one host to another will be as easy as commenting the host you are stop using and uncommenting the one you are start using (remember to copy your SSH key to your other remote hosts as well).

And that's it!

4.3 Tobiko Framework Reference Guide

4.3.1 Tobiko package

tobiko

class tobiko.**BackgroundProcessFixture**(**args*, *target: Optional[Callable] = None*, *process_name: Optional[str] = None*, *pid_file: Optional[str] = None*, *retry_timeout: Optional[float] = None*, *retry_interval: Optional[float] = None*, *terminate_signal=Signals.SIGTERM*, *kill_signal=Signals.SIGTERM*, ***kwargs*)

Bases: *SharedFixture*

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

class tobiko.**CachedProperty**(*fget=None*, *fset=None*, *fdel=None*, *doc=None*, *cached_id=None*)

Bases: object

class tobiko.**CaptureLogFixture**(*test_case_id*, *logger=None*, *level=None*, *fmt=None*)

Bases: *SharedFixture*

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

class tobiko.**ExceptionInfo**(*type*, *value*, *traceback*)

Bases: *ExceptionInfo*

class tobiko.**FixtureManager**

Bases: object

exception tobiko.**InvalidVersion**(*message=None*, ***properties*)

Bases: *TobikoException*

exception tobiko.**MultipleObjectsFound**(*message=None*, ***properties*)

Bases: *TobikoException*

exception tobiko.**ObjectNotFound**(*message=None*, ***properties*)

Bases: *TobikoException*

class tobiko.**Operation**

Bases: *SharedFixture*

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

class tobiko.**RequiredFixture**(*cls: Type[G]*, *setup=True*, ***kwargs*)

Bases: property, Generic[G]

class tobiko.**Retry**(*count: Optional[int] = None*, *timeout: Optional[float] = None*, *sleep_time: Optional[float] = None*, *interval: Optional[float] = None*)

Bases: object

class tobiko.**RetryAttempt**(*number: int*, *start_time: float*, *elapsed_time: float*, *count: Optional[int] = None*, *timeout: Optional[float] = None*, *sleep_time: Optional[float] = None*, *interval: Optional[float] = None*)

Bases: object

exception tobiko.**RetryCountLimitError**(*message=None*, ***properties*)

Bases: *RetryLimitError*

exception tobiko.**RetryLimitError**(*message=None*, ***properties*)

Bases: RetryException

exception tobiko.**RetryTimeLimitError**(*message=None*, ***properties*)

Bases: *RetryLimitError*

class tobiko.**RunsOperations**

Bases: object

exception tobiko.**SecondsValueError**(*message=None*, ***properties*)

Bases: *TobikoException*

class tobiko.**Selection**(*iterable=()*, */*)

Bases: list, Generic[T]

classmethod create(*objects: Optional[Iterable[T]] = None*) → *Selection*[T]

class tobiko.**SharedFixture**

Bases: Fixture

Base class for fixtures intended to be shared between multiple tests

Make sure that fixture setUp method can be called more than once, but actually executing _setUp method only the first time. This allows the fixture to be passed to useFixture methods multiple times without caring about if has already been used before.

Fixture set up can anyway be forced by calling 'setup_shared_fixture' method.

Because cleanup policy in a shared fixture is different from a common fixture, cleanUp method simply doesn't nothing.

Actual fixture cleanup is executed by calling cleanup_shared_fixture method.

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

class tobiko.**TestCaseManager**(*start_time: Optional[float] = None*)

Bases: object

exception tobiko.**TobikoException**(*message=None*, ***properties*)

Bases: Exception

Base Tobiko Exception.

To use this class, inherit from it and define attribute 'message' string. If properties parameters is given, then it will format message string using properties as key-word arguments.

Example:

```
class MyException(TobikoException):
    message = "This exception occurred because of {reason}"

try:
    raise MyException(reason="something went wrong")
except MyException as ex:

    # It should print:
    #   This exception occurred because of something went wrong
    print(ex)

    # It should print:
    #   something went wrong
    print(ex.reason)
```

Attribute message
the message to be printed out.

exception tobiko.**VersionMismatch**(*message=None*, ***properties*)

Bases: *TobikoException*

tobiko.**tobiko_config**()

tobiko.**tobiko_config_dir**()

tobiko.**tobiko_config_path**(*path*)

tobiko.actors

class tobiko.actors.**Actor**(*actor_id: Optional[str] = None*, *loop: Optional[AbstractEventLoop] = None*, *log: Optional[LoggerAdapter] = None*, *requests: Optional[ActorRequestQueue] = None*)

Bases: ActorBase

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

class tobiko.actors.**ActorRef**(*actor: A*)

Bases: *CallProxyBase*, Generic[A]

class tobiko.actors.**CallProxy**(*handle_call: Callable*)

Bases: *CallProxyBase*, Generic[P]

class tobiko.actors.**CallProxyBase**

Bases: CallHandler, Generic[P], ABC

tobiko.docker

class tobiko.docker.**DockerClientFixture**(*base_urls: Optional[Iterable[str]] = None*, *ssh_client: None = None*, *sudo: Optional[bool] = None*)

Bases: *SharedFixture*

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

exception tobiko.docker.**DockerError**(*message=None*, ***properties*)

Bases: *TobikoException*

exception tobiko.docker.**DockerUrlNotFoundError**(*message=None*, ***properties*)

Bases: *TobikoException*

tobiko.http

tobiko.openstack

tobiko.openstack.designate

class tobiko.openstack.designate.**DesignateClientFixture**(*session=None*, *client=None*)

Bases: OpenstackClientFixture

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

tobiko.openstack.glance

class tobiko.openstack.glance.**CustomizedGlanceImageFixture**(*image_file=None*, *image_dir=None*, ***kwargs*)

Bases: *FileGlanceImageFixture*

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

class tobiko.openstack.glance.**FileGlanceImageFixture**(*image_file=None*, *image_dir=None*, ***kwargs*)

Bases: UploadGlanceImageFixture

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

class tobiko.openstack.glance.**GlanceClientFixture**(*session=None*, *client=None*)

Bases: OpenstackClientFixture

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

class tobiko.openstack.glance.**GlanceImageFixture**(*image_name: Optional[str] = None*, *username: Optional[str] = None*, *password: Optional[str] = None*)

Bases: HasGlanceClientMixin, *SharedFixture*

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

class tobiko.openstack.glance.**HasImageMixin**

Bases: HasGlanceClientMixin

class tobiko.openstack.glance.**URLGlanceImageFixture**(*image_url: Optional[str] = None*, ***kwargs*)

Bases: *FileGlanceImageFixture*

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

tobiko.openstack.heat

class tobiko.openstack.heat.**HeatClientFixture**(*session=None*, *client=None*)

Bases: OpenstackClientFixture

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

class tobiko.openstack.heat.**HeatStackFixture**(*stack_name: Optional[str] = None*, *template: Optional[HeatTemplateFixture] = None*, *parameters=None*, *wait_interval: Optional[float] = None*, *client: Union[None, Client, HeatClientFixture] = None*)

Bases: BaseResourceFixture

Manages Heat stacks.

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

outputs = <tobiko.openstack.heat._stack.HeatStackOutputsFixture object>

resources = <tobiko.openstack.heat._stack.HeatStackResourceFixture object>

exception tobiko.openstack.heat.**HeatStackNotFound**(*message=None*, ***properties*)

Bases: HeatStackError

class tobiko.openstack.heat.**HeatTemplateFileFixture**(*template_file: str*, *template_dirs: Optional[Iterable[str]] = None*)

Bases: *HeatTemplateFixture*

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

class tobiko.openstack.heat.**HeatTemplateFixture**(*template: Optional[Mapping[str, Any]] = None*, *template_files: Optional[Mapping] = None*)

Bases: *SharedFixture*

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

tobiko.openstack.ironic

exception tobiko.openstack.ironic.**WaitForNodePowerStateError**(*message=None*, ***properties*)

Bases: *TobikoException*

exception tobiko.openstack.ironic.**WaitForNodePowerStateTimeout**(*message=None*, ***properties*)

Bases: *WaitForNodePowerStateError*

tobiko.openstack.keystone

class tobiko.openstack.keystone.**CloudsFileKeystoneCredentialsFixture**(*credentials: KeystoneCredentials = None, connection: Union[ShellConnection, None, bool, SSHClientFixture, str] = None, environ: Dict[str, str] = None, cloud_name: str = None, directories: Iterable[str] = None, filenames: Iterable[str] = None*)

Bases: *KeystoneCredentialsFixture*

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

class tobiko.openstack.keystone.**ConfigKeystoneCredentialsFixture**(*credentials: KeystoneCredentials = None, connection: Union[ShellConnection, None, bool, SSHClientFixture, str] = None, environ: Dict[str, str] = None*)

Bases: *KeystoneCredentialsFixture*

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

class tobiko.openstack.keystone.**DelegateKeystoneCredentialsFixture**(*delegates: Iterable[KeystoneCredentialsFixture] = None, credentials: KeystoneCredentials = None, connection: Union[ShellConnection, None, bool, SSHClientFixture, str] = None, environ: Dict[str, str] = None*)

Bases: *KeystoneCredentialsFixture*

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

class tobiko.openstack.keystone.**EnvironKeystoneCredentialsFixture**(*credentials: KeystoneCredentials = None, connection: Union[ShellConnection, None, bool, SSHClientFixture, str] = None, environ: Dict[str, str] = None*)

Bases: *KeystoneCredentialsFixture*

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

exception tobiko.openstack.keystone.**InvalidKeystoneCredentials**(*message=None*, ***properties*)

Bases: *TobikoException*

class tobiko.openstack.keystone.**KeystoneClientFixture**(*session=None*, *client=None*)

Bases: OpenstackClientFixture

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

class tobiko.openstack.keystone.**KeystoneCredentials**(*auth_url*, *username*, *password*, *project_name*, *api_version*, *domain_name*, *user_domain_name*, *project_domain_name*, *project_domain_id*, *cacert*, *trust_id*)

Bases: NamedTuple

class tobiko.openstack.keystone.**KeystoneCredentialsFixture**(*credentials: KeystoneCredentials = None*, *connection: Union[ShellConnection, None, bool, SSHClientFixture, str] = None*, *environ: Dict[str, str] = None*)

Bases: *SharedFixture*

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

class tobiko.openstack.keystone.**KeystoneSessionFixture**(*credentials: Optional[Union[KeystoneCredentials, KeystoneCredentialsFixture, Type[KeystoneCredentialsFixture]]] = None*, *session: Optional[Session] = None*)

Bases: *SharedFixture*

VALID_CREDENTIALS_TYPES = (<class 'tobiko.openstack.keystone._credentials.KeystoneCredentials'>, <class 'tobiko.openstack.keystone._credentials.KeystoneCredentialsFixture'>, <class 'type'>)

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

class tobiko.openstack.keystone.**KeystoneSessionManager**

Bases: object

exception tobiko.openstack.keystone.**NoSuchKeystoneCredentials**(*message=None*, ***properties*)

Bases: *ObjectNotFound*

tobiko.openstack.metalsmith

class tobiko.openstack.metalsmith.**MetalsmithClientFixture**(*session=None*, *client=None*)

Bases: OpenstackClientFixture

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

tobiko.openstack.neutron

exception tobiko.openstack.neutron.**AgentNotFoundOnHost**(*message=None*, ***properties*)

Bases: *TobikoException*

exception tobiko.openstack.neutron.**EnsureNeutronQuotaLimitsError**(*message=None*, ***properties*)

Bases: *TobikoException*

class tobiko.openstack.neutron.**NeutronClientFixture**(*session=None*, *client=None*)

Bases: OpenstackClientFixture

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

exception tobiko.openstack.neutron.**NoSuchFloatingIp**(*message=None*, ***properties*)

Bases: *ObjectNotFound*

exception tobiko.openstack.neutron.**NoSuchNetwork**(*message=None*, ***properties*)

Bases: *ObjectNotFound*

exception tobiko.openstack.neutron.**NoSuchPort**(*message=None*, ***properties*)

Bases: *ObjectNotFound*

exception tobiko.openstack.neutron.**NoSuchRouter**(*message=None*, ***properties*)

Bases: *ObjectNotFound*

exception tobiko.openstack.neutron.**NoSuchSubnet**(*message=None*, ***properties*)

Bases: *ObjectNotFound*

exception tobiko.openstack.neutron.**NoSuchSubnetPool**(*message=None*, ***properties*)

Bases: *ObjectNotFound*

tobiko.openstack.nova

exception tobiko.openstack.nova.**EnsureNovaQuotaLimitsError**(*message=None*, ***properties*)

Bases: *TobikoException*

exception tobiko.openstack.nova.**GetServerError**(*message=None*, ***properties*)

Bases: *TobikoException*

class tobiko.openstack.nova.**HasNovaClientMixin**

Bases: object

class tobiko.openstack.nova.**HasServerMixin**

Bases: *HasNovaClientMixin*

class tobiko.openstack.nova.**KeyFileFixture**(*key_file: Optional[str] = None*, *key_type: Optional[str] = None*)

Bases: *SharedFixture*

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

exception tobiko.openstack.nova.**MigrateServerError**(*message=None*, ***properties*)

Bases: *TobikoException*

exception tobiko.openstack.nova.**NoValidHostFoundMigrateServerError**(*message=None*, ***properties*)

Bases: *MigrateServerError*

exception tobiko.openstack.nova.**NotInLocalStorageMigrateServerError**(*message=None*, ***properties*)

Bases: *MigrateServerError*

exception tobiko.openstack.nova.**NotInSharedStorageMigrateServerError**(*message=None*, ***properties*)

Bases: *MigrateServerError*

class tobiko.openstack.nova.**NovaClientFixture**(*session=None*, *client=None*)

Bases: OpenstackClientFixture

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

exception tobiko.openstack.nova.**PreCheckMigrateServerError**(*message=None*, ***properties*)

Bases: *MigrateServerError*

exception tobiko.openstack.nova.**ServerNotFoundError**(*message=None*, ***properties*)

Bases: *GetServerError*, *ObjectNotFound*

exception tobiko.openstack.nova.**WaitForCloudInitTimeoutError**(*message=None*, ***properties*)

Bases: InvalidCloudInitStatusError

exception tobiko.openstack.nova.**WaitForServerStatusError**(*message=None*, ***properties*)

Bases: *TobikoException*

exception tobiko.openstack.nova.**WaitForServerStatusTimeout**(*message=None*, ***properties*)

Bases: *WaitForServerStatusError*

tobiko.openstack.octavia

exception tobiko.openstack.octavia.**AmphoraMgmtPortNotFound**(*message=None*, ***properties*)

Bases: *TobikoException*

class tobiko.openstack.octavia.**OctaviaClientFixture**(*session=None*, *client=None*)

Bases: OpenstackClientFixture

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

exception tobiko.openstack.octavia.**RequestException**(*message=None*, ***properties*)

Bases: *TobikoException*

exception tobiko.openstack.octavia.**RoundRobinException**(*message=None*, ***properties*)

Bases: *TobikoException*

exception tobiko.openstack.octavia.**TimeoutException**(*message=None*, ***properties*)

Bases: *TobikoException*

exception tobiko.openstack.octavia.**TrafficTimeoutError**(*message=None*, ***properties*)

Bases: *TobikoException*

tobiko.openstack.openstackclient

exception tobiko.openstack.openstackclient.**OSPCliAuthError**(*message=None*, ***properties*)

Bases: OSPCliError

tobiko.openstack.openstackclient.**OSPCliError**

alias of *OSPCliAuthError*

tobiko.openstack.stacks

class tobiko.openstack.stacks.**AffinityServerGroupStackFixture**

Bases: *SharedFixture*

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

property server_group_stack: G

class tobiko.openstack.stacks.**AntiAffinityServerGroupStackFixture**

Bases: *SharedFixture*

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

property server_group_stack: G

class tobiko.openstack.stacks.**CirrosDifferentHostServerStackFixture**(*stack_name: Optional[str] = None*, *template: Optional[*HeatTemplateFixture*] = None*, *parameters=None*, *wait_interval: Optional[float] = None*, *client: Union[None, Client,* HeatClientFixture*] = None*)

Bases: *CirrosPeerServerStackFixture*, DifferentHostServerStackFixture

property flavor_stack: G

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

property image_fixture: G

property key_pair_stack: G

property network_stack: G

outputs = <tobiko.openstack.heat._stack.HeatStackOutputsFixture object>

property peer_stack: G

resources = <tobiko.openstack.heat._stack.HeatStackResourceFixture object>

template: *_template.HeatTemplateFixture* = **<tobiko.openstack.heat._template.HeatTemplateFileFixture object>**

Heat template file

class tobiko.openstack.stacks.**CirrosFlavorStackFixture**(*stack_name: Optional[str] = None*, *template: Optional[*HeatTemplateFixture*] = None*, *parameters=None*, *wait_interval: Optional[float] = None*, *client: Union[None, Client,* HeatClientFixture*] = None*)

Bases: *FlavorStackFixture*

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

outputs = <tobiko.openstack.heat._stack.HeatStackOutputsFixture object>

resources = <tobiko.openstack.heat._stack.HeatStackResourceFixture object>

template: *_template.HeatTemplateFixture* = **<tobiko.openstack.heat._template.HeatTemplateFileFixture object>**

class tobiko.openstack.stacks.**CirrosImageFixture**(*image_url: Optional[str] = None*, ***kwargs*)

Bases: *URLGlanceImageFixture*

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

class tobiko.openstack.stacks.**CirrosNoFipServerStackFixture**(*stack_name: Optional[str] = None*, *template: Optional[*HeatTemplateFixture*] = None*, *parameters=None*, *wait_interval: Optional[float] = None*, *client: Union[None, Client,* HeatClientFixture*] = None*)

Bases: *ExtraDhcpOptsCirrosServerStackFixture*

property flavor_stack: G

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

property image_fixture: G

property key_pair_stack: G

property network_stack: G

outputs = <tobiko.openstack.heat._stack.HeatStackOutputsFixture object>

resources = <tobiko.openstack.heat._stack.HeatStackResourceFixture object>

template: *_template.HeatTemplateFixture* = <tobiko.openstack.heat._template.HeatTemplateFileFixture object>

Heat template file

class tobiko.openstack.stacks.**CirrosPeerServerStackFixture**(*stack_name: Optional[str] = None*, *template: Optional[*HeatTemplateFixture*] = None*, *parameters=None*, *wait_interval: Optional[float] = None*, *client: Union[None, Client,* HeatClientFixture*] = None*)

Bases: *CirrosServerStackFixture*, PeerServerStackFixture

property flavor_stack: G

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

property image_fixture: G

property key_pair_stack: G

property network_stack: G

outputs = <tobiko.openstack.heat._stack.HeatStackOutputsFixture object>

property peer_stack: G

Peer server used to reach this one

resources = <tobiko.openstack.heat._stack.HeatStackResourceFixture object>

template: *_template.HeatTemplateFixture* = <tobiko.openstack.heat._template.HeatTemplateFileFixture object>

Heat template file

class tobiko.openstack.stacks.**CirrosSameHostServerStackFixture**(*stack_name: Optional[str] = None*, *template: Optional[*HeatTemplateFixture*] = None*, *parameters=None*, *wait_interval: Optional[float] = None*, *client: Union[None, Client,* HeatClientFixture*] = None*)

Bases: *CirrosPeerServerStackFixture*, SameHostServerStackFixture

property flavor_stack: G

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

property image_fixture: G

property key_pair_stack: G

property network_stack: G

outputs = <tobiko.openstack.heat._stack.HeatStackOutputsFixture object>

property peer_stack: G

resources = <tobiko.openstack.heat._stack.HeatStackResourceFixture object>

template: *_template.HeatTemplateFixture* = <tobiko.openstack.heat._template.HeatTemplateFileFixture object>

Heat template file

class tobiko.openstack.stacks.**CirrosServerStackFixture**(*stack_name: Optional[str] = None*, *template: Optional[*HeatTemplateFixture*] = None*, *parameters=None*, *wait_interval: Optional[float] = None*, *client: Union[None, Client,* HeatClientFixture*] = None*)

Bases: *ServerStackFixture*

property flavor_stack: G

Flavor used to create a Nova server instance

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

property image_fixture: G

Glance image used to create a Nova server instance

property key_pair_stack: G

property network_stack: G

outputs = <tobiko.openstack.heat._stack.HeatStackOutputsFixture object>

resources = <tobiko.openstack.heat._stack.HeatStackResourceFixture object>

template: *_template.HeatTemplateFixture* = <tobiko.openstack.heat._template.HeatTemplateFileFixture object>

Heat template file

class tobiko.openstack.stacks.**CirrosServerWithDefaultSecurityGroupStackFixture**(*stack_name: Optional[str] = None*, *template: Optional[HeatTemplateFixture] = None*, *parameters=None*, *wait_interval: Optional[float] = None*, *client: Union[None, Client, HeatClientFixture] = None*)

Bases: *CirrosServerStackFixture*

property flavor_stack: G

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

property image_fixture: G

property key_pair_stack: G

property network_stack: G

outputs = <tobiko.openstack.heat._stack.HeatStackOutputsFixture object>

resources = <tobiko.openstack.heat._stack.HeatStackResourceFixture object>

template: *_template.HeatTemplateFixture* = <tobiko.openstack.heat._template.HeatTemplateFileFixture object>

Heat template file

class tobiko.openstack.stacks.**CirrosShellConnection**(*ssh_client: Optional[SSHClientFixture] = None*)

Bases: *SSHShellConnection*

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

class tobiko.openstack.stacks.**CloudInitServerStackFixture**(*stack_name: Optional[str] = None*, *template: Optional[HeatTemplateFixture] = None*, *parameters=None*, *wait_interval: Optional[float] = None*, *client: Union[None, Client, HeatClientFixture] = None*)

Bases: *ServerStackFixture*, ABC

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

property key_pair_stack: G

property network_stack: G

outputs = <tobiko.openstack.heat._stack.HeatStackOutputsFixture object>

resources = <tobiko.openstack.heat._stack.HeatStackResourceFixture object>

template: _template.HeatTemplateFixture = <tobiko.openstack.heat._template.HeatTemplateFileFixture object>

Heat template file

class tobiko.openstack.stacks.**DesignateZoneStackFixture**(*stack_name: Optional[str] = None*, *template: Optional[*HeatTemplateFixture*] = None*, *parameters=None*, *wait_interval: Optional[float] = None*, *client: Union[None, Client,* HeatClientFixture*] = None*)

Bases: *HeatStackFixture*

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

outputs = <tobiko.openstack.heat._stack.HeatStackOutputsFixture object>

resources = <tobiko.openstack.heat._stack.HeatStackResourceFixture object>

template: _template.HeatTemplateFixture = <tobiko.openstack.heat._template.HeatTemplateFileFixture object>

class tobiko.openstack.stacks.**EvacuableCirrosImageFixture**(*image_url: Optional[str] = None*, ***kwargs*)

Bases: *CirrosImageFixture*

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

class tobiko.openstack.stacks.**EvacuableServerStackFixture**(*stack_name: Optional[str] = None*, *template: Optional[*HeatTemplateFixture*] = None*, *parameters=None*, *wait_interval: Optional[float] = None*, *client: Union[None, Client,* HeatClientFixture*] = None*)

Bases: *CirrosServerStackFixture*

property flavor_stack: G

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

property image_fixture: G

Glance image used to create a Nova server instance

property key_pair_stack: G

property network_stack: G

outputs = <tobiko.openstack.heat._stack.HeatStackOutputsFixture object>

resources = <tobiko.openstack.heat._stack.HeatStackResourceFixture object>

template: *_template.HeatTemplateFixture* = <tobiko.openstack.heat._template.HeatTemplateFileFixture object>

Heat template file

class tobiko.openstack.stacks.**ExtraDhcpOptsCirrosServerStackFixture**(*stack_name: Optional[str] = None*, *template: Optional[*HeatTemplateFixture*] = None*, *parameters=None*, *wait_interval: Optional[float] = None*, *client: Union[None, Client,* HeatClientFixture*] = None*)

Bases: *CirrosServerStackFixture*

property flavor_stack: G

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

property image_fixture: G

property key_pair_stack: G

property network_stack: G

outputs = <tobiko.openstack.heat._stack.HeatStackOutputsFixture object>

resources = <tobiko.openstack.heat._stack.HeatStackResourceFixture object>

template: *_template.HeatTemplateFixture* = <tobiko.openstack.heat._template.HeatTemplateFileFixture object>

Heat template file

class tobiko.openstack.stacks.**FlavorStackFixture**(*stack_name: Optional[str] = None*, *template: Optional[*HeatTemplateFixture*] = None*, *parameters=None*, *wait_interval: Optional[float] = None*, *client: Union[None, Client,* HeatClientFixture*] = None*)

Bases: *HeatStackFixture*

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

outputs = <tobiko.openstack.heat._stack.HeatStackOutputsFixture object>

resources = <tobiko.openstack.heat._stack.HeatStackResourceFixture object>

template: _template.HeatTemplateFixture = <tobiko.openstack.heat._template.HeatTemplateFileFixture object>

class tobiko.openstack.stacks.**FloatingIpStackFixture**(*stack_name: Optional[str] = None*, *network: Optional[Union[str, Dict[str, Any]]] = None*, *port: Optional[Union[str, Dict[str, Any]]] = None*, *device_id: Optional[str] = None*, *fixed_ip_address: Optional[str] = None*, *heat_client: Union[None, Client, HeatClientFixture] = None*, *neutron_client: Optional[Union[Client, NeutronClientFixture]] = None*)

Bases: *HeatStackFixture*

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

outputs = <tobiko.openstack.heat._stack.HeatStackOutputsFixture object>

resources = <tobiko.openstack.heat._stack.HeatStackResourceFixture object>

property router_stack: G

template: _template.HeatTemplateFixture = <tobiko.openstack.heat._template.HeatTemplateFileFixture object>

Heat template file

class tobiko.openstack.stacks.**KeyPairStackFixture**(*stack_name: Optional[str] = None*, *template: Optional[HeatTemplateFixture] = None*, *parameters=None*, *wait_interval: Optional[float] = None*, *client: Union[None, Client, HeatClientFixture] = None*)

Bases: *HeatStackFixture*

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

outputs = <tobiko.openstack.heat._stack.HeatStackOutputsFixture object>

resources = <tobiko.openstack.heat._stack.HeatStackResourceFixture object>

template: _template.HeatTemplateFixture = <tobiko.openstack.heat._template.HeatTemplateFileFixture object>

class tobiko.openstack.stacks.**L3haDifferentHostServerStackFixture**(*stack_name: Optional[str] = None*, *template: Optional[HeatTemplateFixture] = None*, *parameters=None*, *wait_interval: Optional[float] = None*, *client: Union[None, Client, HeatClientFixture] = None*)

Bases: *L3haPeerServerStackFixture*, DifferentHostServerStackFixture

property flavor_stack: G

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

property image_fixture: G

property key_pair_stack: G

property network_stack: G

outputs = <tobiko.openstack.heat._stack.HeatStackOutputsFixture object>

property peer_stack: G

resources = <tobiko.openstack.heat._stack.HeatStackResourceFixture object>

template: *_template.HeatTemplateFixture* = <tobiko.openstack.heat._template.HeatTemplateFileFixture object>

Heat template file

class tobiko.openstack.stacks.**L3haNetworkStackFixture**(*stack_name: Optional[str] = None*, *template: Optional[*HeatTemplateFixture*] = None*, *parameters=None*, *wait_interval: Optional[float] = None*, *client: Union[None, Client,* HeatClientFixture*] = None*)

Bases: *NetworkBaseStackFixture*

property gateway_stack: G

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

outputs = <tobiko.openstack.heat._stack.HeatStackOutputsFixture object>

resources = <tobiko.openstack.heat._stack.HeatStackResourceFixture object>

classmethod skip_if_router_is_distributed(*reason: Optional[str] = None*)

property subnet_pools_ipv4_stack: G

property subnet_pools_ipv6_stack: G

template: *_template.HeatTemplateFixture* = <tobiko.openstack.heat._template.HeatTemplateFileFixture object>

Heat template file

class tobiko.openstack.stacks.**L3haPeerServerStackFixture**(*stack_name: Optional[str] = None*, *template: Optional[*HeatTemplateFixture*] = None*, *parameters=None*, *wait_interval: Optional[float] = None*, *client: Union[None, Client,* HeatClientFixture*] = None*)

Bases: *L3haServerStackFixture*, PeerServerStackFixture

property flavor_stack: G

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

property image_fixture: G

property key_pair_stack: G

property network_stack: G

outputs = <tobiko.openstack.heat._stack.HeatStackOutputsFixture object>

property peer_stack: G

resources = <tobiko.openstack.heat._stack.HeatStackResourceFixture object>

template: *_template.HeatTemplateFixture* = <tobiko.openstack.heat._template.HeatTemplateFileFixture object>

Heat template file

class tobiko.openstack.stacks.**L3haSameHostServerStackFixture**(*stack_name: Optional[str] = None*, *template: Optional[HeatTemplateFixture] = None*, *parameters=None*, *wait_interval: Optional[float] = None*, *client: Union[None, Client, HeatClientFixture] = None*)

Bases: *L3haPeerServerStackFixture*, SameHostServerStackFixture

property flavor_stack: G

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

property image_fixture: G

property key_pair_stack: G

property network_stack: G

outputs = <tobiko.openstack.heat._stack.HeatStackOutputsFixture object>

property peer_stack: G

resources = <tobiko.openstack.heat._stack.HeatStackResourceFixture object>

template: *_template.HeatTemplateFixture* = <tobiko.openstack.heat._template.HeatTemplateFileFixture object>

Heat template file

class tobiko.openstack.stacks.**L3haServerStackFixture**(*stack_name: Optional[str] = None*, *template: Optional[HeatTemplateFixture] = None*, *parameters=None*, *wait_interval: Optional[float] = None*, *client: Union[None, Client, HeatClientFixture] = None*)

Bases: *CirrosServerStackFixture*

property flavor_stack: *G*

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

property image_fixture: *G*

property key_pair_stack: *G*

property network_stack: *G*

Heat stack for creating internal network with L3HA enabled

outputs = <tobiko.openstack.heat._stack.HeatStackOutputsFixture object>

resources = <tobiko.openstack.heat._stack.HeatStackResourceFixture object>

template: *_template.HeatTemplateFixture* = **<tobiko.openstack.heat._template.HeatTemplateFileFixture object>**

Heat template file

class tobiko.openstack.stacks.**MultiIPCirrosServerStackFixture**(*stack_name: Optional[str] = None*, *template: Optional[*HeatTemplateFixture*] = None*, *parameters=None*, *wait_interval: Optional[float] = None*, *client: Union[None, Client,* HeatClientFixture*] = None*)

Bases: *CirrosServerStackFixture*

property flavor_stack: *G*

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

property image_fixture: *G*

property key_pair_stack: *G*

property network_stack: *G*

outputs = <tobiko.openstack.heat._stack.HeatStackOutputsFixture object>

resources = <tobiko.openstack.heat._stack.HeatStackResourceFixture object>

template: *_template.HeatTemplateFixture* = **<tobiko.openstack.heat._template.HeatTemplateFileFixture object>**

Heat template file

class tobiko.openstack.stacks.**NetworkBaseStackFixture**(*stack_name: Optional[str] = None*, *template: Optional[*HeatTemplateFixture*] = None*, *parameters=None*, *wait_interval: Optional[float] = None*, *client: Union[None, Client,* HeatClientFixture*] = None*)

Bases: *HeatStackFixture*

Heat stack for creating internal network with a router to external

property gateway_stack: G

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

outputs = <tobiko.openstack.heat._stack.HeatStackOutputsFixture object>

resources = <tobiko.openstack.heat._stack.HeatStackResourceFixture object>

classmethod skip_if_router_is_distributed(*reason: Optional[str] = None*)

property subnet_pools_ipv4_stack: G

property subnet_pools_ipv6_stack: G

template: *_template.HeatTemplateFixture* = <tobiko.openstack.heat._template.HeatTemplateFileFixture object>

Heat template file

class tobiko.openstack.stacks.**NetworkStackFixture**(*stack_name: Optional[str] = None*, *template: Optional[*HeatTemplateFixture*] = None*, *parameters=None*, *wait_interval: Optional[float] = None*, *client: Union[None, Client,* HeatClientFixture*] = None*)

Bases: *NetworkBaseStackFixture*

property gateway_stack: G

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

outputs = <tobiko.openstack.heat._stack.HeatStackOutputsFixture object>

resources = <tobiko.openstack.heat._stack.HeatStackResourceFixture object>

classmethod skip_if_router_is_distributed(*reason: Optional[str] = None*)

property subnet_pools_ipv4_stack: G

property subnet_pools_ipv6_stack: G

template: *_template.HeatTemplateFixture* = <tobiko.openstack.heat._template.HeatTemplateFileFixture object>

Heat template file

class tobiko.openstack.stacks.**NetworkWithNetMtuWriteStackFixture**(*stack_name: Optional[str] = None*, *template: Optional[*HeatTemplateFixture*] = None*, *parameters=None*, *wait_interval: Optional[float] = None*, *client: Union[None, Client,* HeatClientFixture*] = None*)

Bases: *NetworkBaseStackFixture*

property gateway_stack: G

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

outputs = <tobiko.openstack.heat._stack.HeatStackOutputsFixture object>

resources = <tobiko.openstack.heat._stack.HeatStackResourceFixture object>

classmethod skip_if_router_is_distributed(*reason: Optional[str] = None*)

property subnet_pools_ipv4_stack: G

property subnet_pools_ipv6_stack: G

template: _template.HeatTemplateFixture = <tobiko.openstack.heat._template.HeatTemplateFileFixture object>

Heat template file

class tobiko.openstack.stacks.**OctaviaOtherServerStackFixture**(*stack_name: Optional[str] = None*, *template: Optional[*HeatTemplateFixture*] = None*, *parameters=None*, *wait_interval: Optional[float] = None*, *client: Union[None, Client,* HeatClientFixture*] = None*)

Bases: *OctaviaServerStackFixture*

property flavor_stack: G

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

property image_fixture: G

property key_pair_stack: G

property network_stack: G

outputs = <tobiko.openstack.heat._stack.HeatStackOutputsFixture object>

resources = <tobiko.openstack.heat._stack.HeatStackResourceFixture object>

template: _template.HeatTemplateFixture = <tobiko.openstack.heat._template.HeatTemplateFileFixture object>

Heat template file

property vlan_network_stack: G

class tobiko.openstack.stacks.**OctaviaServerStackFixture**(*stack_name: Optional[str] = None*, *template: Optional[*HeatTemplateFixture*] = None*, *parameters=None*, *wait_interval: Optional[float] = None*, *client: Union[None, Client,* HeatClientFixture*] = None*)

Bases: *UbuntuServerStackFixture*

property flavor_stack: G

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

property image_fixture: G

property key_pair_stack: G

property network_stack: G

outputs = <tobiko.openstack.heat._stack.HeatStackOutputsFixture object>

resources = <tobiko.openstack.heat._stack.HeatStackResourceFixture object>

template: *_template.HeatTemplateFixture* = <tobiko.openstack.heat._template.HeatTemplateFileFixture object>

Heat template file

property vlan_network_stack: G

class tobiko.openstack.stacks.**QosNetworkStackFixture**(*stack_name: Optional[str] = None*, *template: Optional[*HeatTemplateFixture*] = None*, *parameters=None*, *wait_interval: Optional[float] = None*, *client: Union[None, Client,* HeatClientFixture*] = None*)

Bases: *NetworkBaseStackFixture*

property gateway_stack: G

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

outputs = <tobiko.openstack.heat._stack.HeatStackOutputsFixture object>

property qos_stack: G

stack with the qos policy for the network

resources = <tobiko.openstack.heat._stack.HeatStackResourceFixture object>

classmethod skip_if_router_is_distributed(*reason: Optional[str] = None*)

property subnet_pools_ipv4_stack: G

property subnet_pools_ipv6_stack: G

template: *_template.HeatTemplateFixture* = <tobiko.openstack.heat._template.HeatTemplateFileFixture object>

Heat template file

class tobiko.openstack.stacks.**QosPolicyStackFixture**(*stack_name: Optional[str] = None*, *template: Optional[*HeatTemplateFixture*] = None*, *parameters=None*, *wait_interval: Optional[float] = None*, *client: Union[None, Client,* HeatClientFixture*] = None*)

Bases: *HeatStackFixture*

Heat stack with a QoS Policy and some QoS Policy Rules

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

outputs = <tobiko.openstack.heat._stack.HeatStackOutputsFixture object>

resources = <tobiko.openstack.heat._stack.HeatStackResourceFixture object>

template: _template.HeatTemplateFixture = <tobiko.openstack.heat._template.HeatTemplateFileFixture object>

Heat template file

class tobiko.openstack.stacks.**QosServerStackFixture**(*stack_name: Optional[str] = None*, *template: Optional[*HeatTemplateFixture*] = None*, *parameters=None*, *wait_interval: Optional[float] = None*, *client: Union[None, Client,* HeatClientFixture*] = None*)

Bases: *UbuntuServerStackFixture*

property flavor_stack: G

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

property image_fixture: G

property key_pair_stack: G

property network_stack: G

stack with the network with a qos policy

outputs = <tobiko.openstack.heat._stack.HeatStackOutputsFixture object>

resources = <tobiko.openstack.heat._stack.HeatStackResourceFixture object>

template: _template.HeatTemplateFixture = <tobiko.openstack.heat._template.HeatTemplateFileFixture object>

Heat template file

property vlan_network_stack: G

class tobiko.openstack.stacks.**RebootCirrosServerOperation**(*ssh_client:* SSHClientFixture, *timeout: Optional[float] = None*, *method: RebootHostMethod = RebootHostMethod.SOFT*)

Bases: *RebootHostOperation*

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

property stack: G

class tobiko.openstack.stacks.**RouterInterfaceStackFixture**(*stack_name: Optional[str] = None*, *router: Optional[Union[str, Dict[str, Any]]] = None*, *network: Optional[Union[str, Dict[str, Any]]] = None*, *subnet: Optional[Union[str, Dict[str, Any]]] = None*, *neutron_client: Optional[Union[Client, NeutronClientFixture]] = None*)

Bases: *HeatStackFixture*

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

outputs = <tobiko.openstack.heat._stack.HeatStackOutputsFixture object>

resources = <tobiko.openstack.heat._stack.HeatStackResourceFixture object>

template: _template.HeatTemplateFixture = <tobiko.openstack.heat._template.HeatTemplateFileFixture object>

Heat template file

class tobiko.openstack.stacks.**RouterStackFixture**(*neutron_client: Optional[Union[Client, NeutronClientFixture]] = None*)

Bases: ExternalNetworkStackFixture

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

outputs = <tobiko.openstack.heat._stack.HeatStackOutputsFixture object>

resources = <tobiko.openstack.heat._stack.HeatStackResourceFixture object>

template: _template.HeatTemplateFixture = <tobiko.openstack.heat._template.HeatTemplateFileFixture object>

class tobiko.openstack.stacks.**SecurityGroupsFixture**(*stateful: bool = True*)

Bases: *HeatStackFixture*

Heat stack with some security groups

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

outputs = <tobiko.openstack.heat._stack.HeatStackOutputsFixture object>

resources = <tobiko.openstack.heat._stack.HeatStackResourceFixture object>

template: _template.HeatTemplateFixture = <tobiko.openstack.heat._template.HeatTemplateFileFixture object>

Heat template file

class tobiko.openstack.stacks.**ServerGroupStackFixture**(*stack_name: Optional[str] = None*, *template: Optional[HeatTemplateFixture] = None*, *parameters=None*, *wait_interval: Optional[float] = None*, *client: Union[None, Client, HeatClientFixture] = None*)

Bases: *HeatStackFixture*

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

outputs = <tobiko.openstack.heat._stack.HeatStackOutputsFixture object>

resources = <tobiko.openstack.heat._stack.HeatStackResourceFixture object>

template: _template.HeatTemplateFixture = <tobiko.openstack.heat._template.HeatTemplateFileFixture object>

class tobiko.openstack.stacks.**ServerStackFixture**(*stack_name: Optional[str] = None*, *template: Optional[HeatTemplateFixture] = None*, *parameters=None*, *wait_interval: Optional[float] = None*, *client: Union[None, Client, HeatClientFixture] = None*)

Bases: *HeatStackFixture*, ABC

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

property key_pair_stack: G

stack with the key pair for the server instance

property network_stack: G

stack with the internal where the server port is created

outputs = <tobiko.openstack.heat._stack.HeatStackOutputsFixture object>

resources = <tobiko.openstack.heat._stack.HeatStackResourceFixture object>

template: _template.HeatTemplateFixture = <tobiko.openstack.heat._template.HeatTemplateFileFixture object>

Heat template file

class tobiko.openstack.stacks.**StatelessSecurityGroupFixture**(*description=None*, *rules=None*)

Bases: ResourceFixture

Neutron Stateless Security Group Fixture.

This SG will by default allow SSH and ICMP to the instance and also ingress traffic from the metadata service as it can't rely on conntrack.

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

class tobiko.openstack.stacks.**UbuntuExternalServerStackFixture**(*stack_name: Optional[str] = None*, *template: Optional[HeatTemplateFixture] = None*, *parameters=None*, *wait_interval: Optional[float] = None*, *client: Union[None, Client, HeatClientFixture] = None*)

Bases: *UbuntuServerStackFixture*, ExternalServerStackFixture

Ubuntu server with port on special external network

property flavor_stack: G

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

property image_fixture: G

property key_pair_stack: G

property network_stack: G

outputs = <tobiko.openstack.heat._stack.HeatStackOutputsFixture object>

resources = <tobiko.openstack.heat._stack.HeatStackResourceFixture object>

template: *_template.HeatTemplateFixture* = <tobiko.openstack.heat._template.HeatTemplateFileFixture object>

Heat template file

property vlan_network_stack: G

class tobiko.openstack.stacks.**UbuntuFlavorStackFixture**(*stack_name: Optional[str] = None*, *template: Optional[*HeatTemplateFixture*] = None*, *parameters=None*, *wait_interval: Optional[float] = None*, *client: Union[None, Client,* HeatClientFixture*] = None*)

Bases: *FlavorStackFixture*

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

outputs = <tobiko.openstack.heat._stack.HeatStackOutputsFixture object>

resources = <tobiko.openstack.heat._stack.HeatStackResourceFixture object>

template: *_template.HeatTemplateFixture* = <tobiko.openstack.heat._template.HeatTemplateFileFixture object>

class tobiko.openstack.stacks.**UbuntuImageFixture**(*ethernet_devide: Optional[str] = None*, ***kwargs*)

Bases: *UbuntuMinimalImageFixture*, *CustomizedGlanceImageFixture*

Ubuntu server image running an HTTP server

The server has additional installed packages compared to the minimal one:

- iperf3
- ping
- ncat
- nginx
- vlan

The image will also have below running services:

- nginx HTTP server listening on TCP port 80
- iperf3 server listening on TCP port 5201

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

class tobiko.openstack.stacks.**UbuntuMinimalImageFixture**(*image_file=None*, *image_dir=None*, ***kwargs*)

Bases: *FileGlanceImageFixture*

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

class tobiko.openstack.stacks.**UbuntuMinimalServerStackFixture**(*stack_name: Optional[str] = None*, *template: Optional[*HeatTemplateFixture*] = None*, *parameters=None*, *wait_interval: Optional[float] = None*, *client: Union[None, Client,* HeatClientFixture*] = None*)

Bases: *CloudInitServerStackFixture*

property flavor_stack: G

Flavor used to create a Nova server instance

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

property image_fixture: G

Glance image used to create a Nova server instance

property key_pair_stack: G

property network_stack: G

outputs = <tobiko.openstack.heat._stack.HeatStackOutputsFixture object>

resources = <tobiko.openstack.heat._stack.HeatStackResourceFixture object>

template: *_template.HeatTemplateFixture* = <tobiko.openstack.heat._template.HeatTemplateFileFixture object>

Heat template file

class tobiko.openstack.stacks.**UbuntuServerStackFixture**(*stack_name: Optional[str] = None*, *template: Optional[*HeatTemplateFixture*] = None*, *parameters=None*, *wait_interval: Optional[float] = None*, *client: Union[None, Client,* HeatClientFixture*] = None*)

Bases: *UbuntuMinimalServerStackFixture*, VlanServerStackFixture

Ubuntu server running an HTTP server

The server has additional commands compared to the minimal one:

- iperf3

- ping

property flavor_stack: G

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

property image_fixture: G

Glance image used to create a Nova server instance

property key_pair_stack: G

property network_stack: G

outputs = <tobiko.openstack.heat._stack.HeatStackOutputsFixture object>

resources = <tobiko.openstack.heat._stack.HeatStackResourceFixture object>

template: *_template.HeatTemplateFixture* = **<tobiko.openstack.heat._template.HeatTemplateFileFixture object>**

Heat template file

property vlan_network_stack: G

class tobiko.openstack.stacks.**VlanNetworkStackFixture**(*stack_name: Optional[str] = None*, *template: Optional[*HeatTemplateFixture*] = None*, *parameters=None*, *wait_interval: Optional[float] = None*, *client: Union[None, Client,* HeatClientFixture*] = None*)

Bases: *NetworkBaseStackFixture*

property gateway_stack: G

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

outputs = <tobiko.openstack.heat._stack.HeatStackOutputsFixture object>

resources = <tobiko.openstack.heat._stack.HeatStackResourceFixture object>

classmethod skip_if_router_is_distributed(*reason: Optional[str] = None*)

property subnet_pools_ipv4_stack: G

property subnet_pools_ipv6_stack: G

template: *_template.HeatTemplateFixture* = **<tobiko.openstack.heat._template.HeatTemplateFileFixture object>**

Heat template file

class tobiko.openstack.stacks.**VlanProxyServerStackFixture**(*stack_name: Optional[str] = None*, *template: Optional[*HeatTemplateFixture*] = None*, *parameters=None*, *wait_interval: Optional[float] = None*, *client: Union[None, Client,* HeatClientFixture*] = None*)

Bases: *CirrosServerStackFixture*

property flavor_stack: G

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

property image_fixture: G

property key_pair_stack: G

property network_stack: G

outputs = <tobiko.openstack.heat._stack.HeatStackOutputsFixture object>

resources = <tobiko.openstack.heat._stack.HeatStackResourceFixture object>

template: *_template.HeatTemplateFixture* = <tobiko.openstack.heat._template.HeatTemplateFileFixture object>

Heat template file

tobiko.openstack.topology

exception tobiko.openstack.topology.**NoSuchOpenStackTopologyNode**(*message=None*, ***properties*)

Bases: *TobikoException*

exception tobiko.openstack.topology.**NoSuchOpenStackTopologyNodeGroup**(*message=None*, ***properties*)

Bases: *TobikoException*

class tobiko.openstack.topology.**OpenStackTopology**

Bases: *SharedFixture*

property config: G

file_digger_class

alias of *JournalLogDigger*

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_agent_container_name(*agent_name: str*) → str

classmethod get_agent_service_name(*agent_name: str*) → str

classmethod get_fixture_manager() → *FixtureManager*

class tobiko.openstack.topology.**OpenStackTopologyConfig**

Bases: *SharedFixture*

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

class tobiko.openstack.topology.**OpenStackTopologyNode**(*topology: OpenStackTopology*, *name: str*, *ssh_client: Optional[SSHClientFixture]*, *addresses: Iterable[IPAddress]*, *hostname: str*)

Bases: object

exception tobiko.openstack.topology.**UnknowOpenStackContainerNameError**(*message=None*, ***properties*)

Bases: *TobikoException*

exception tobiko.openstack.topology.**UnknowOpenStackServiceNameError**(*message=None*, ***properties*)

Bases: *TobikoException*

exception tobiko.openstack.topology.**UnknownOpenStackConfigurationFile**(*message=None*, ***properties*)

Bases: *TobikoException*

tobiko.podman

class tobiko.podman.**PodmanClientFixture**(*ssh_client=None*)

Bases: *SharedFixture*

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

exception tobiko.podman.**PodmanError**(*message=None*, ***properties*)

Bases: *TobikoException*

exception tobiko.podman.**PodmanSocketNotFoundError**(*message=None*, ***properties*)

Bases: *TobikoException*

tobiko.run

tobiko.shell

tobiko.shell.ansible

class tobiko.shell.ansible.**AnsiblePlaybook**(*command: Union[*ShellCommand*, str, Iterable] = 'ansible-playbook'*, *inventory_filenames: Iterable[str] = None*, *playbook: str = 'main'*, *playbook_dirname: str = None*, *requirement_files: Iterable[str] = None*, *roles_path: Iterable[str] = None*, *ssh_client: Union[None, bool,* SSHClientFixture*] = None*, *verbosity: int = None*, *work_dir: str = None*)

Bases: *SharedFixture*

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

tobiko.shell.curl

class tobiko.shell.curl.**CurlHeader**(*head_line: Optional[str]*, **args*, ***kwargs*)

Bases: UserDict

classmethod fromkeys(*iterable*, *value=None*)

class tobiko.shell.curl.**CurlProcessFixture**(*url: str*, *continue_at: Optional[int] = None*, *create_dirs: Optional[bool] = None*, *file_name: Optional[str] = None*, *head_only: Optional[bool] = None*, *headers_file_name: Optional[str] = None*, *location: Optional[bool] = None*, *max_redirs: Optional[int] = None*, *ssh_client: None = None*, *sudo: Optional[bool] = None*)

Bases: *SharedFixture*

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

tobiko.shell.files

class tobiko.shell.files.**JournalLogDigger**(*filename: str*, *pattern: Optional[str] = None*, ***execute_params*)

Bases: *LogFileDigger*

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

class tobiko.shell.files.**LogFileDigger**(*filename: str*, *pattern: Optional[str] = None*, ***execute_params*)

Bases: *SharedFixture*

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

class tobiko.shell.files.**MultihostLogFileDigger**(*filename: str*, *ssh_clients: ~typing.Optional[~typing.Iterable[~typing.Union[None, bool, ~tobiko.shell.ssh._client.SSHClientFixture]]] = None*, *file_digger_class: ~typing.Type[~tobiko.shell.files._logs.LogFileDigger] = <class 'tobiko.shell.files._logs.LogFileDigger'>*, *pattern: ~typing.Optional[str] = None*, ***execute_params*)

Bases: *SharedFixture*

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

tobiko.shell.iperf3

tobiko.shell.ping

exception tobiko.shell.ping.**BadAddressPingError**(*message=None*, ***properties*)

Bases: *PingError*

Raised when passing wrong address to ping command

exception tobiko.shell.ping.**ConnectPingError**(*message=None*, ***properties*)

Bases: *PingError*

Raised when sendto error happens

exception tobiko.shell.ping.**LocalPingError**(*message=None*, ***properties*)

Bases: *PingError*

Raised when local error happens

exception tobiko.shell.ping.**PingError**(*message=None*, ***properties*)

Bases: *PingException*

Base ping error

exception tobiko.shell.ping.**PingException**(*message=None*, ***properties*)

Bases: *TobikoException*

Base ping command exception

exception tobiko.shell.ping.**PingFailed**(*message=None*, ***properties*)

Bases: *PingError*, AssertionError

Raised when ping timeout expires before reaching expected message count

class tobiko.shell.ping.**PingStatistics**(*source=None*, *destination=None*, *transmitted: int = 0*, *received: int = 0*, *undelivered: int = 0*, *begin_interval=None*, *end_interval=None*)

Bases: object

Ping command statistics

exception tobiko.shell.ping.**ReachableHostsException**(*message=None*, ***properties*)

Bases: *PingFailed*

exception tobiko.shell.ping.**SendToPingError**(*message=None*, ***properties*)

Bases: *PingError*

Raised when sendto error happens

exception tobiko.shell.ping.**UnknowHostError**(*message=None*, ***properties*)

Bases: *PingError*

Raised when unable to resolve host name

exception tobiko.shell.ping.**UnreachableHostsException**(*message=None*, ***properties*)

Bases: *PingFailed*

tobiko.shell.sh

exception tobiko.shell.sh.**CommandNotFound**(*message=None*, ***properties*)

Bases: *ObjectNotFound*

tobiko.shell.sh.**HostNameError**

alias of HostnameError

class tobiko.shell.sh.**ListNameserversFixture**(*ssh_client: Optional[SSHClientFixture] = None*, *filenames: Optional[Iterable[str]] = None*, ***execute_params*)

Bases: *SharedFixture*

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

class tobiko.shell.sh.**LocalExecutePathFixture**(*executable_dirs=None*, *environ=None*)

Bases: ExecutePathFixture

environ: Dict[str, str] = {'DEBIAN_FRONTEND': 'noninteractive', 'DOCUTILSCONFIG': '/home/docs/checkouts/readthedocs.org/user_builds/tobiko/checkouts/master/doc/source/docutils.conf', 'HOME': '/home/docs', 'HOSTNAME': 'build-24155497-project-432992-tobiko', 'LANG': 'C.UTF-8', 'NO_COLOR': '1', 'PATH': '/home/docs/checkouts/readthedocs.org/user_builds/tobiko/envs/master/bin:/home/docs/.asdf/shims:/home/docs/.asdf/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin', 'PWD': '/home/docs/checkouts/readthedocs.org/user_builds/tobiko/checkouts/master/doc/source', 'PYTHON': '/home/docs/checkouts/readthedocs.org/user_builds/tobiko/envs/master/bin/python', 'READTHEDOCS': 'True', 'READTHEDOCS_CANONICAL_URL': 'https://tobiko.readthedocs.io/en/master/', 'READTHEDOCS_GIT_CLONE_URL': 'https://opendev.org/x/tobiko', 'READTHEDOCS_GIT_COMMIT_HASH': '2945010168e9277baf52ab1db44d49b8157253ee', 'READTHEDOCS_GIT_IDENTIFIER': 'master', 'READTHEDOCS_LANGUAGE': 'en', 'READTHEDOCS_OUTPUT': '/home/docs/checkouts/readthedocs.org/user_builds/tobiko/checkouts/master/_readthedocs/', 'READTHEDOCS_PROJECT': 'tobiko', 'READTHEDOCS_VERSION': 'master', 'READTHEDOCS_VERSION_NAME': 'master', 'READTHEDOCS_VERSION_TYPE': 'branch', 'READTHEDOCS_VIRTUALENV_PATH': '/home/docs/checkouts/readthedocs.org/user_builds/tobiko/envs/master'}

executable_dirs: List[str] = ['/home/docs/checkouts/readthedocs.org/user_builds/tobiko/envs/master/bin', '/home/docs/checkouts/readthedocs.org/user_builds/tobiko/envs/master/bin']

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

class tobiko.shell.sh.**LocalShellConnection**

Bases: *ShellConnection*

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

class tobiko.shell.sh.**LocalShellProcessFixture**(***kwargs*)

Bases: *ShellProcessFixture*

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

property path_execute: G

exception tobiko.shell.sh.**PsError**(*message=None*, ***properties*)

Bases: *TobikoException*

class tobiko.shell.sh.**PsProcess**(*command: str*, *pid: int*, *ssh_client: Union[None, bool,* SSHClientFixture*]*, *is_cirros: Optional[bool] = None*)

Bases: PsProcessTuple, PsProcessBase

exception tobiko.shell.sh.**PsWaitTimeout**(*message=None*, ***properties*)

Bases: *PsError*

exception tobiko.shell.sh.**RebootHostError**(*message=None*, ***properties*)

Bases: *TobikoException*

class tobiko.shell.sh.**RebootHostOperation**(*ssh_client:* SSHClientFixture, *timeout: Optional[float] = None*, *method: RebootHostMethod = RebootHostMethod.SOFT*)

Bases: *Operation*

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

exception tobiko.shell.sh.**RebootHostTimeoutError**(*message=None*, ***properties*)

Bases: *RebootHostError*

class tobiko.shell.sh.**SSHShellConnection**(*ssh_client: Optional[*SSHClientFixture*] = None*)

Bases: *ShellConnection*

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

class tobiko.shell.sh.**SSHShellProcessFixture**(***kwargs*)

Bases: *ShellProcessFixture*

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

class tobiko.shell.sh.**ShellCommand**(*iterable=()*, */*)

Bases: tuple

exception tobiko.shell.sh.**ShellCommandFailed**(*message=None*, ***properties*)

Bases: *ShellError*

Raised when shell command exited with non-zero status

class tobiko.shell.sh.**ShellConnection**

Bases: *SharedFixture*

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

exception tobiko.shell.sh.**ShellError**(*message=None*, ***properties*)

Bases: *TobikoException*

class tobiko.shell.sh.**ShellExecuteResult**(*command*, *exit_status*, *timeout*, *status*, *login*, *stdin*, *stdout*, *stderr*)

Bases: *ShellExecuteResult*

class tobiko.shell.sh.**ShellProcessFixture**(***kwargs*)

Bases: *SharedFixture*

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

exception tobiko.shell.sh.**ShellProcessNotTerminated**(*message=None*, ***properties*)

Bases: *ShellError*

exception tobiko.shell.sh.**ShellProcessTerminated**(*message=None*, ***properties*)

Bases: *ShellError*

exception tobiko.shell.sh.**ShellStdinClosed**(*message=None*, ***properties*)

Bases: *ShellError*

class tobiko.shell.sh.**ShellStdout**(*delegate*, *fd=None*, *buffer_size=None*)

Bases: ShellReadable

exception tobiko.shell.sh.**ShellTimeoutExpired**(*message=None*, ***properties*)

Bases: *ShellError*

Raised when shell command timeouts and has been killed before exiting

exception tobiko.shell.sh.**SkipOnCommandNotFound**(*message=None*, ***properties*)

Bases: *CommandNotFound*, SkipTest

class tobiko.shell.sh.**SystemdUnit**(*unit*, *load*, *active*, *sub*, *description*, *data*)

Bases: NamedTuple

exception tobiko.shell.sh.**UptimeError**(*message=None*, ***properties*)

Bases: *TobikoException*

tobiko.shell.ssh

class tobiko.shell.ssh.**SSHClientFixture**(*host=None*, *proxy_client=None*, *host_config=None*, *config_files=None*, *schema=None*, ***kwargs*)

Bases: *SharedFixture*

property default: G

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

exception tobiko.shell.ssh.**SSHConnectFailure**(*message=None*, ***properties*)

Bases: *TobikoException*

class tobiko.shell.ssh.**SSHHostConfig**(*host*, *ssh_config*, *host_config*, *config_files*)

Bases: *SSHHostConfig*

property default: G

class tobiko.shell.ssh.**SSHTunnelForwarder**(*ssh_client*, ***kwargs*)

Bases: SSHTunnelForwarder

class tobiko.shell.ssh.**SSHTunnelForwarderFixture**(*ssh_client*)

Bases: *SharedFixture*

classmethod **get**(*manager=None*, *fixture_id=None*)

classmethod **get_fixture_manager**() → *FixtureManager*

tobiko.shell.tcpdump

tobiko.shell.find

exception tobiko.shell.find.**FilesNotFound**(*message=None*, ***properties*)

Bases: *TobikoException*

tobiko.shell.grep

exception tobiko.shell.grep.**NoMatchingLinesFound**(*message=None*, ***properties*)

Bases: *TobikoException*

tobiko.shell.ifconfig

exception tobiko.shell.ifconfig.**IfconfigError**(*message=None*, ***properties*)

Bases: *TobikoException*

tobiko.shell.ip

exception tobiko.shell.ip.**IpError**(*message=None*, ***properties*)

Bases: *TobikoException*

tobiko.shell.ss

class tobiko.shell.ss.**SockData**

Bases: dict

A single socket information parsed from output of ss command line tool

Output of ss command line tool should be parsed and stored in the dict with keys are items of the SockHeader object. In most of the cases it should contain the following keys:

- protocol (optional)
- state (optional)
- recv_q
- send_q
- local_addr (IP or filename)
- local_port
- remote_addr (IP or filename)
- remote_port
- process (list of processes names)

class tobiko.shell.ss.**SockHeader**(*header_str: str*)

Bases: object

class tobiko.shell.ss.**SockLine**

Bases: str

Single line from the output of ss command line tool

It should match with the corresponding table header that is presented by object of SockHeader class

exception tobiko.shell.ss.**SocketLookupError**(*message=None*, ***properties*)

Bases: *TobikoException*

tobiko.shiftstack

class tobiko.shiftstack.**ShiftStackCloudsFileFixture**(*local_clouds_file_path: Optional[str] = None*, *remote_clouds_file_path: Optional[str] = None*, *connection: Optional[*ShellConnection*] = None*)

Bases: *SharedFixture*

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

tobiko.tripleo

tobiko.tripleo.containers

class tobiko.tripleo.containers.**ContainerRuntimeFixture**

Bases: *SharedFixture*

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

tobiko.tripleo.nova

tobiko.tripleo.pacemaker

class tobiko.tripleo.pacemaker.**PacemakerResourcesStatus**

Bases: object

class to handle pcs resources checks

exception tobiko.tripleo.pacemaker.**PcsResourceException**(*message=None*, ***properties*)

Bases: *TobikoException*

tobiko.tripleo.processes

exception tobiko.tripleo.processes.**OvercloudProcessesException**(*message=None*, ***properties*)

Bases: *TobikoException*

class tobiko.tripleo.processes.**OvercloudProcessesStatus**

Bases: object

class to handle processes checks, checks that all of these are running in the overcloud: 'ovsdb-server','pcsd', 'corosync', 'beam.smp', 'mysqld', 'redis-server', 'haproxy', 'nova-conductor', 'nova-scheduler', 'neutron-server', 'nova-compute', 'glance-api'

tobiko.tripleo.services

exception tobiko.tripleo.services.**OvercloudServiceException**(*message=None*, ***properties*)

Bases: *TobikoException*

class tobiko.tripleo.services.**OvercloudServicesStatus**(*services_to_check: Optional[List[str]] = None*)

Bases: *SharedFixture*

class to handle services checks, checks that all of these are running in the overcloud: 'corosync.service','iptables.service','network.service','ntpd.service', 'pacemaker.service','rpcbind.service','sshd.service'

classmethod get(*manager=None*, *fixture_id=None*)

classmethod get_fixture_manager() → *FixtureManager*

tobiko.tripleo

- genindex
- search

4.4 Configuration Files

Tobiko accept below configuration files

4.4.1 tobiko.conf

Tobiko will look for configuration files at below paths:

- /etc/tobiko/tobiko.conf
- ./tobiko.conf

The format of the file is the same as other OpenStack *.conf ini files.

This section provides a list of all configuration options for tobiko.conf file. It is being auto-generated from Tobiko code when this documentation is being built.

It can also be viewed in raw format.

```
[DEFAULT]
```

4.5 Miscellaneous

4.5.1 From Tempest to Tobiko

This doc might help programmers to embrace/join the Tobiko project, after having some experience with the Tempest project.

A few differences between Tobiko and Tempest:

- **Tobiko uses heat template stacks**
  - A stack is being created *"lazily"* (only when it is needed in the code), unlike "resource setup" in Tempest that always creates all the resources.
- **Tobiko re-uses its resources**
  - Tobiko will reuse the same resources in any possible test it can.

    Ex: if one test will create a stack, and the same stack is going to be used in different tests, Tobiko will simply reuse it. It will also reuse the same stacks during other executions of the same Tobiko tests.
- **Tobiko's tests structure:**
  - Tests do not require idempotent_id, each method which starts with the prefix "test" is considered to be a test which will be run.
  - After a Tobiko test suite finishes, it **does not** delete Tobiko's resources (no teardown), and this is by design.
- **Tobiko uses a client stack**
  - Traffic is being sent from a client stack that the tester may create (or a default one might be created).
- **Tobiko uses Typing:**

  - Mainly as a means of code readability and consistency.

    But additional doc strings are welcome!

- **Tobiko tests workflow:**
  - Tobiko uses the following workflow for its sequential test execution process:

    resource-creation->disruptive-actions->resource-verification.

PYTHON MODULE INDEX

t

INDEX

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W