Klever Documentation

Release 1.0

ISP RAS

Jul 13, 2018

Contents

Contents:

CHAPTER 1

Deployment

Klever does not support standard deployment means because it consists of several components that may require complicating setup, e.g. configuring and running a web service with a database access, running system services that perform some preliminary actions with superuser rights, etc. Also, Klever will likely always require several specific addons that can not be deployed in a normal way. Please, be ready to spend quite much time if you follow this instruction first time.

1.1 Hardware Requirements

We recommend following hardware to deploy and to run Klever:

- 64-bit CPU with 4 cores
- 16 GB of memory
- 100 GB of free disk space

Increasing specified hardware characteristics in 2-4 times can reduce total verification time very considerably.

1.2 Deployment Variants

There are several variants for deploying Klever:

1.2.1 Local Deployment

Warning: Do not deploy Klever at your workstation or valuable servers unless you are ready to lose some sensitive data or to have misbehaved software. We hope that one day Klever will be more safe and secure, so this warning will be redundant.

Local Deployment works for Debian 9. Also, you can try it for other versions of Debian as well as for various Debian derivatives, e.g. it works for Ubuntu 18.04 as well.

Prior to proceding to *Local Deployment*, it is necessary to perform deploy_common. Then you need to choose an appropriate deployment mode. One should select *development* only if one is going to develop Klever (see *Deployment for Development Purposes* in addition). Otherwise, please, choose *production*. Then you should *install* Klever:

```
$ sudo $KLEVER_SRC/deploys/bin/deploy-local --deployment-directory $KLEVER_DEPLOY_DIR␣
↪install production
```

After successfull installation one is able to *update* Klever multiple times to install new or to update alredy installed klever_addons and target_software as well as to update Klever itself:

```
$ sudo $KLEVER_SRC/deploys/bin/deploy-local --deployment-directory $KLEVER_DEPLOY_DIR␣
↪update production
```

To *uninstall* Klever, e.g. if something went wrong during installation, you need to run:

```
$ sudo $KLEVER_SRC/deploys/bin/deploy-local --deployment-directory $KLEVER_DEPLOY_DIR␣
↪uninstall production
```

So that a normal sequence of actions for *Local Deployment* is the following: "*install* → *update* → *update* → . . . → *update* → *uninstall*". In addition, there are several optional arguments which you can find out by running:

```
$ $KLEVER_SRC/deploys/bin/deploy-local --help
```

1.2.2 OpenStack Deployment

Althouth we would like to support different OpenStack environments, at the moment *OpenStack Deployment* likely works just for the ISP RAS one.

Prior to proceding to *OpenStack Deployment*, it is necessary to perform deploy_common. Additionally you need to install following Python3 packages:

- cinderclient.
- glanceclient.
- keystoneauth1.
- neutronclient.
- novaclient.
- paramiko.
- pycryptodome.

OpenStack Deployment supports 3 kinds of entities:

- *Klever Base Image* - usually this is a Debian 9 OpenStack image with installed packages and Python3 packages which will most likely required for Klever. Using *Klever Base Image* allows to substantially reduce a time for deploying other entities.
- *Klever Developer Instance* - an OpenStack instance for development purposes. For *Klever Developer Instance* many debug options are activated by default.
- *Klever Experimental Instances* - a specified number of OpenStack instances for performing various experiments.

In addition to arguments mentioned below, there are several optional arguments which you can find out by running:

```
$ $KLEVER_SRC/deploys/bin/deploy-openstack --help
```

Klever Base Image

For *Klever Base Image* you can execute actions *show*, *create* and *remove*. The normal workflow for *Klever Base Image* is "*create* → *remove*":

```
$ $KLEVER_SRC/deploys/bin/deploy-openstack --ssh-rsa-private-key-file $SSH_RSA_
↪PRIVATE_KEY_FILE create "Klever base image"
```

It is not necessary to *remove Klever Base Image* ever for allowing one to understand what images running OpenStack instances are based on. Unless specified, name *Klever Base* is used for new *Klever Base Image*. If there is already an image with such the name it will be renamed by adding suffix *deprecated* (indeed, this is done recursively with using ordinal numbers of images in addition, so, no images will be lost and there will not be any duplicates).

Klever Developer Instance

For *Klever Developer Instance* you can execute actions *show*, *create*, *update*, *ssh*, *remove*, *share* and *hide*. Basically you should perform actions with *Klever Developer Instance* in the following order "*create* → *update* → *update* → ... → *update* → *remove*" exactly as for *Local Deployment*:

```
$ $KLEVER_SRC/deploys/bin/deploy-openstack --ssh-rsa-private-key-file $SSH_RSA_
↪PRIVATE_KEY_FILE create "Klever developer instance"
```

In addition, between creating and removing you can also *share*/*hide* for/from the outside world *Klever Developer Instance* and open an SSH connection to it. By default a name for *Klever Developer Instance* is a concatenation of an OpenStack username and *-klever-dev*.

Klever Experimental Instances

For *Klever Experimental Instances* you can execute actions *show*, *create* and *remove*. The normal workflow for *Klever Experimental Instances* is "*create* → *remove*":

```
$ $KLEVER_SRC/deploys/bin/deploy-openstack --ssh-rsa-private-key-file $SSH_RSA_
↪PRIVATE_KEY_FILE --instances $INSTANCES create "Klever experimental instances"
```

Deployment Troubleshooting

If at running script `deploy-openstack` you met the following exception:

```
Traceback (most recent call last):
  File "./deploys/bin/deploy-openstack", line 27, in <module>
    sys.exit(deploys.openstack.main())
  File "./deploys/bin/../deploys/openstack/__init__.py", line 80, in main
    getattr(OSKleverDeveloperInstance(args, logger), args.action)()
  File "./deploys/bin/../deploys/openstack/openstack.py", line 296, in create
    base_image=base_image, flavor_name=self.args.flavor) as self.instance:
  File "./deploys/bin/../deploys/openstack/instance.py", line 75, in __enter__
    self._setup_keypair()
  File "./deploys/bin/../deploys/openstack/instance.py", line 171, in _setup_keypair
    public_key = RSA.import_key(private_key).publickey().exportKey('OpenSSH')
AttributeError: module 'Crypto.PublicKey.RSA' has no attribute 'import_key'.
```

Then you should check that you have properly installed Python3 package pycryptodome.

CHAPTER 2

Developer Documentation

2.1 How to Write This Documentation

This documentation is created using Sphinx from reStructuredText source files. To improve existing documentation or to develop the new one you need to read at least the following chapters of the Sphinx documentation:

1. Defining document structure.
2. Adding content.
3. Running the build.
4. reStructuredText Primer.
5. Sphinx Markup Constructs.
6. Sphinx Domains (you can omit language specific domains).

Please, follow these advises:

1. Do not think that other developers and especially users are so smart as you are.
2. Clarify ambiguous things and describe all the details without missing anything.
3. Avoid and fix misprints.
4. Write each sentence on a separate line.
5. Do not use blank lines except it is required.
6. Write a new line at the end of each source file.
7. Break sentences longer than 120 symbols to several lines if possible.

To develop documentation it is recommended to use some visual editor.

Warning: Please do not reinvent the wheel! If you are a newbie then examine carefully the existing documentation and create the new one on that basis. Just if you are a guru then you can suggest to improve the existing documentation.

2.2 Using Git Repository

Klever source code resides in the Git repository. There is plenty of very good documentation about Git usage. This section describes just rules specific for the given project.

2.2.1 Update

1. Periodically synchronize your local repository with the main development repository (it is available just internally at ISP RAS):

```
branch $ git fetch origin
branch $ git remote prune origin
```

Note: This is especially required when you are going to create a new branch or to merge some branch to the master branch.

2. Pull changes if so:

```
branch $ git pull --rebase origin branch
```

Warning: Forget about pulling without rebasing!

3. Resolve conflicts if so.

2.2.2 Fixing Bugs and Implementing New Features

1. One must create a new branch to fix each individual bug or implement a new feature:

```
master $ git checkout -b fix-conf
```

Warning: Do not intermix fixes and implementation of completely different bugs and features into one branch. Otherwise other developers will need to wait or to make some tricky things like cherry-picking and merging of non-master branches. Eventually this can lead to very unpleasant consequences, e.g. the master branch can be broken because of one will merge there a branch based on another non working branch.

2. Push all new branches to the main development repository. As well re-push them at least one time a day if you make some commits:

```
fix-conf $ git push origin fix-conf
```

3. Merge the master branch into your new branches if you need some recent bug fixes or features:

```
fix-conf $ git merge master
```

Note: Do not forget to update the master branch from the main development repository.

Note: Do not merge remote-tracking branches.

4. Ask senior developers to review and to merge branches to the master branch when corresponding bugs/features are fixed/implemented.
5. Delete merged branches:

```
master $ git branch -d fix-conf
```

2.3 Releases

Generally we follow the same rules as for development of the Linux kernel.

Each several months a new release will be issued, e.g. 0.1, 0.2, 1.0.

Just after this a merge window of several weeks will be opened. During the merge window features implemented after a previous merge window or during the given one will be merged to master.

After the merge window just bug fixes can be merged to the master branch. During this period we can issue several release candidates, e.g. 1.0-rc1, 1.0-rc2.

In addition, after issuing a new release we can decide to support a stable branch. This branch will start from a commit corresponding to the given release. It can contain just bug fixes relevant to an existing functionality and not to a new one which is supported within a corresponding merge window.

2.4 Deployment for Development Purposes

To deploy Klever for development purposes in addition to using mode *development* (see *Local Deployment*) one needs to specify command-line option *–allow-symbolic-links*.

2.5 Using PyCharm IDE

To use PyCharm IDE for developing Klever follow the following steps.

2.5.1 Installation

1. Download PyCharm Community from https://www.jetbrains.com/pycharm/download/ (below all settings are given for version 2017.1.1, you have to adapt them for your version by yourself).
2. Follow installation instructions provided at that site.

2.5.2 Setting Project

At the "Welcome to PyCharm" window:

1. Specify your preferences.
2. *Open*.
3. Specify the absolute path to directory `$KLEVER_SRC/bridge`.
4. *OK*.

2.5.3 Configuring the Python Interpreter

1. *File* → *Settings* → *Project: Bridge* → *Project Interpreter* → *Settings* → *More...*
2. Select Python 3.4 or higher from the list and press `Enter`.
3. Input *Python 3* in field *name*.
4. *OK*.
5. Ditto for *core*, *deploys*, *docs*, *scheduler* and *utils*.

2.5.4 Setting Run/Debug Configuration

Common run/debug configurations are included into the Klever project. Common configurations with names starting with **$** should be copied to configurations with names without **$** and adjusted in accordance with instructions below. If you want to adjust configurations with names that not starting with **$** you also have to copy them before.

1. *Run* → *Edit Configurations...*.

Klever Bridge Run/Debug Configuration

Note: This is available just for PyCharm Professional.

- Specify *0.0.0.0* in field *Host* if you want to share your Klever Bridge to the local network.
- Specify your preferred port in field *Port*.

Note: To make your Klever Bridge accessible from the local network you might need to set up your firewall accordingly.

Klever Core Run/Debug Configuration

This run/debug configuration is only useful if you are going to debug Klever Core.

- Extend existing value of environment variable `PATH` so that CIF (`cif` or `compiler`), Aspectator (`aspectator`) and CIL (`cilly.asm.exe`) binaries could be found (edit value of field *Environment variables*).
- Specify the absolute path to the working directory in field *Working directory*.

Note: Place Klever Core working directory somewhere outside the main development repository.

Note: Klever Core will search for its configuration file `core.json` in the specified working directory. Thus, the best workflow to debug Klever Core is to set its working directory to the one created previously when it was run without debugging. Besides, you can provide this file by passing its name as a first parameter to the script.

Documentation Run/Debug Configuration

Specify another representation of documenation in field *Command* if you need it.

2.5.5 Testing

Klever Bridge Testing

Note: This is available just for PyCharm Professional.

1. *Tools* → *Run manage.py Task...* :

```
manage.py@bridge > test
```

Note: To start tests from console:

```
$ cd bridge
$ python3 manage.py test
```

Note: Another way to start tests from console:

```
$ python3 path/to/klever/bridge/manage.py test bridge users jobs reports marks service
```

Note: The test database is created and deleted automatically. If the user will interrupt tests the test database will preserved and the user will be asked for its deletion for following testing.

Note: PyCharm has reach abilities to analyse tests and their results.

2.5.6 Additional documentation

A lot of usefull documentation for developing Django projects as well as for general using of the PyCharm IDE is available at the official site.

Index