master branch

The 2nd International KLEE Workshop on Symbolic Execution is coming!
Join us from 14-15 September 2020 in London.

#DOCKER

Using KLEE with Docker

Quick Start

Assuming you have Docker installed, you can run the following to try the latest release of KLEE:

```
$ docker pull klee/klee:2.0
$ docker run --rm -ti --ulimit='stack=-1:-1' klee/klee:2.0
```

What is Docker?

Docker provides tools for deploying applications within containers. Containers are (mostly) isolated from each other and the underlying system. This allows you to make a KLEE container, tinker with it and then throw it away when you're done without affecting the underlying system or other

master branch

A Docker container is built from a Docker image. A Docker image encapsulates an applicatio in this case is KLEE. This application level encapsulation is useful because it provides a "portabl and reproducible environment to run KLEE in.

Installing Docker

To run Docker natively on your machine you need to be using a Linux distribution with Docker installed. Follow these links for installation instructions on Ubuntu, OS X and Windows.

Getting the KLEE Docker image

There are two ways of obtaining the KLEE Docker image.

Pulling from the Docker Hub

Our GitHub repository is linked to the DockerHub so that changes to particular branches trigger an automatic rebuild of the corresponding Docker image.

To pull down the latest build of a particular Docker image run:

```
$ docker pull klee/klee
```

If you want to use a tagged revision of KLEE you should instead run:

```
$ docker pull klee/klee:<TAG>
```

Where `<TAG>` is one of the tags listed on the DockerHub. Typically this is either `latest`

master branch

Note this process pulls images containing code compiled by a third-party service. W accept responsibility for the contents of the image.

Building the Docker image locally

If you want to build the Docker image manually instead of pulling the image from DockerHub you can do so by running:

```
$ git clone https://github.com/klee/klee.git
$ cd klee
$ docker build -t klee/klee .
```

This will use the contents of the `Dockerfile` in the root of the repository to build the Docker image.

Creating a KLEE Docker container

Now that you have a KLEE Docker image you can try creating a container from the image.

The simplest thing to try is creating a temporary container and gain shell access. To do this run

```
$ docker run --rm -ti --ulimit='stack=-1:-1' klee/klee
```

Note if you wanted to use a tagged KLEE image replace `klee/klee` with `klee/klee:<TAG>` where `<TAG>` is the tag you wish to use

Note the `--ulimit` option sets an unlimited stack size inside the container. This is to avoid stack overflow issues when running KLEE.

≡ KLEE

master branch

```
klee@3c098b05ca85:~$ whoami
klee
klee@3c098b05ca85:~$
```

You can now try running KLEE inside the container, where you should see an output similar to:

```
klee@3c098b05ca85:~$ klee --version
KLEE 2.0 (https://klee.github.io)
  Build mode: RelWithDebInfo (Asserts: ON)
  Build revision: 938434b2521d4c1ec11af31f1e5e5fbafd2cb2cd

LLVM (http://llvm.org/):
  LLVM version 6.0.1
  Optimized build with assertions.
  Default target: x86_64-unknown-linux-gnu
  Host CPU: skylake
```

and Clang

```
$ clang --version
clang version 6.0.1 (branches/release_60 355598)
Target: x86_64-unknown-linux-gnu
Thread model: posix
InstalledDir: /tmp/llvm-60-install_O_D_A/bin
```

Now exit the container

≡ KLEE

master branch

Because the `--rm` flag was used with the `docker run` command the container was destroyed (not visible in `docker ps -a`) when the application running in the container (`/bin/bash`) terminated.

Persistent Containers

The earlier example didn't do anything interesting with KLEE and threw the created container away. Here is a simple example that does something slightly more interesting with KLEE and also shows how a container can be persisted.

To create and enter the container run:

```
$ docker run -ti --name=my_first_klee_container --ulimit='stack=-1
```

Notice we didn't use `--rm` so the container will not be destroyed when we exit it from it and we also gave the container a name using the `--name` flag.

Now inside the container you can try running the following:

```
klee@3c098b05ca85:~$ pwd
/home/klee
klee@3c098b05ca85:~$ echo "int main(int argn, char** argv) { retur
klee@3c098b05ca85:~$ clang -emit-llvm -g -c test.c -o test.bc
klee@3c098b05ca85:~$ klee --libc=uclibc --posix-runtime test.bc
KLEE: NOTE: Using klee-uclibc : /home/klee/klee_build/klee/Release
KLEE: NOTE: Using model: /home/klee/klee_build/klee/Release+Assert
KLEE: output directory is "/home/klee/klee-out-0"
```

```
KLEE: WARNING ONCE: calling __user_main with extra arguments

KLEE: done: total instructions = 5047
KLEE: done: completed paths = 1
KLEE: done: generated tests = 1
klee@3c098b05ca85:~$ ls
klee-last  klee-out-0  klee_build  klee_src  test.bc  test.c
```

Now exit the container

```
klee@3c098b05ca85:~$ exit
```

We can check that the container still exists by running

```
$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATE
1c408467bdf7        klee/klee           "/bin/bash"         About
```

You can restart the container by running

```
$ docker start -ai my_first_klee_container
klee@1c408467bdf7:~$ ls
klee-last  klee-out-0  klee_build  klee_src  test.bc  test.c
klee@1c408467bdf7:~$ exit
```

As you can see the files we created earlier are still present.

master branch

```
$ docker rm my_first_klee_container
```

to remove it.

Working with KLEE containers built from the KLEE Docker image

There are a few useful things to know about KLEE Docker containers created using the KLEE Docker image.

- The Docker image is based on an Ubuntu 16.04 LTS image.
- Inside the Docker image the `klee` user has sudo access (password is `klee`) so that you can install other applications inside the container (e.g. a text editor). Given that the default user has sudo access this image **should never be used in a production environment**.
- You may want files on your native filesystem available in the container. By default the host filesystem is not visible inside the container. You can use the `--volume=` option to `docker run` to mount directories on the host filesystem into the container.
- These Docker images use LLVM 6.0 so you need to use `clang` to create LLVM bitcode.
- `/home/klee/klee_src` contains the source code used to build KLEE.
- `/home/klee/klee_build` contains the build of KLEE built from `/home/klee/klee_src`
- All the previous examples implicitly run `/bin/bash` inside the container. This is the default but it is also possible to run KLEE directly (useful for scripting) by specifying the command line to use to `docker run`.

This should give you everything you need to start playing with KLEE using Docker. If you are unfamiliar with Docker and wish to learn more a good place to start is Docker's documentation.

master branch

Resources

Mailing List

Doxygen

GitHub

TravisCI

Documentation for KLEE master branch