Linux Containers (LXC)

Qingwei Lan (404458762)
University of California, Los Angeles

June 9, 2016

1 Introduction to Linux Containers

Linux Containers is an implementation for the "containers" concept. Traditionally virtualization starts with the physical hardware at the bottom, then comes the kernel and operating system, and above that is the hypervisor. Atop the hypervisor are the virtual machine images that allow the user to run many different systems. A container is similar to the traditional virtualization method except that it doesn't have the hypervisor layer.

A major advantage of containers over traditional virtual machines is speed since containers do not run an individual operating system but shares the same host operating system. This allows it to make efficient use of system calls and hardware resources without going through the hypervisor layer.

Linux Containers provides a virtual envoronment complete with its own CPU, memory, and I/O, which is made possible by cgroups and namespaces in the Linux kernel.

2 Commonly Used LXC Commands

In this section I will introduce some of the most commonly used commands in the current stable 2.0 version of LXC. These command binaries can be found in the `src/lxc` directory after compilation through the command "`./autogen.sh && ./configure && make`".

`lxc-create, lxc-destroy`

These two commands are used to create and destroy containers, taking the container name as the parameter. For example to create and destroy a container named `p1`, we execute the following commands

```
$ lxc-create --name p1
$ lxc-destroy --name p1
```

`lxc-start, lxc-stop`

These two commands are used to start a container and stop a container, taking the container name as the parameter. We could also start a container in the background by setting an optional parameter `--daemon`. Stopping a container kills all processes within the container. For example to start and stop a container named `p1`, we execute the following commands

```
$ # runs in background if --daemon is set
$ lxc-start --name p1 [--daemon]
$ lxc-stop --name p1
```

`lxc-freeze, lxc-unfreeze`

These two commands are used to temporarily stop all processes in containers or resume the stopped processes. The two commands take the container name as the parameter. For example to freeze and unfreeze a container named `p1`, we execute the following commands

```
$ lxc-freeze --name p1
$ lxc-unfreeze --name p1
```

`lxc-ls`

This command is used to list all containers in the system. Because it is built on top of the GNU `ls` program, it can be run with the options of the `ls` program. Following are example commands

```
$ lxc-ls # list all containers
$ lxc-ls -C1 # list all containers in one column
$ lxc-ls -l # list all containers and permissions
```

`lxc-info`

This command is used for displaying the information of a specific container, taking the name of the container as the parameter. For example to show the information of a container named `p1`, we execute the following command

```
$ lxc-info --name p1
```

`lxc-monitor`

This command is used for monitoring the states of one or more containers. It takes the container names (in regular expression format) as the parameter. The command will output state changes for the specified containers. Following are example commands to monitor containers named `a` and `b` or all containers in the system

```
$ lxc-monitor --name a # monitor container a
$ lxc-monitor --name ‘a|b’ # monitor containers a and b
$ lxc-monitor --name ‘.*’ # monitor all containers
```

`lxc-wait`

This command waits for a container to reach one or more specific states and exits. It takes the container name and intended states as parameters. Following are examples to wait on a container called `p1`

```
$ # exits when RUNNING is reached
$ lxc-wait --name p1 --states RUNNING

$ # exits when RUNNING or STOPPED is reached
$ lxc-wait --name p1 --states RUNNING|STOPPED
```

`lxc-attach`

This command starts a process inside a running container, taking the name of container and the command to run as parameters. The container has to be running in order for this command to work. To spawn a new shell inside a container named `p1`, execute the following command

```
$ lxc-attach --name p1
```

`lxc-copy`

This command creates a copy of existing containers and optionally starts the copy. It takes the to-be-copied container name as a parameter. This command is intended to replace the deprecated commands `lxc-clone` and `lxc-start-ephemeral`.

3 New LXC Commands

For my project I added a direcory method to the outer layer of the container. The method is `lxc-pd` (print directories in a container).

Then I built an export system `lxc-export` where users can export a certain snapshot of a container so that it can be used as a base for future containers.

3.1 `lxc-pd` (print-directory)

This command is be used to print directories in a container specified by the `--name` or `-n` option. It will print recursively for `N` levels (default is print all levels) starting from the root directory. The number of levels can be specified with the option `--dir-level` or `-L` followed by the number of levels. An example usage will be

```
$ lxc-pd --name p1 --dir-level 5
```

This will print directories in container `p1` for up to 5 levels of recursion. To print only directories, we can specify the `--dir-only` or `-d` option. An example usage will be

```
$ lxc-pd --name p1 --dir-only
```

3.2 `lxc-export` (export-container)

This command will export a certain container or a container snapshot as a "base container" that can be used for future containers. In other words, the container in question can be used as the base for another container.

3.2.1 Export Container

To export a container we use the `--name` or `-n` option to specify the name of the container we want to export and the `--export` or `-e` option to specify the exported container name. An example usage would be

```
$ lxc-export --name p1 --export swift-dev
```

This create an exported container named `swift-dev` and saves the state of the container `p1`. Note that exported containers are static read-only and they cannot be started or modified.

3.2.2 Listing Exported Containers

In order to list all the exported containers we simply use the `--list` or `-L` option in the command. An example usage will be

```
$ lxc-export -L
```

This command lists all the previously exported containers.

3.2.3 Creating a New Container From Exported Containers

To create a new container from an exported container is one of the most crucial things of this project. To do that we use the `--name` or `-n` option to specify the name of the exported container and the `--create` or `-c` option to specify the name of the new container-to-be-created. An example usage will be

```
$ lxc-export --name swift-dev --create p2
```

This command creates a new container `p2` from the exported container `swift-dev`.

3.2.4 Deleting an Exported Container

To delete an exported container, we use the `--name` or `-n` option to specify the name of the exported container we want to delete and the `--delete` or `-D` option to specify that we want to delete it. An example usage would be

```
$ lxc-export --name swift-dev --delete
```

This command deletes the exported container named `swift-dev`.

4 Challenges of the Project

The main challenges of the project are listed below. Most of these challenges are the problem of a large code base.

- The biggest challenge was understanding the core implementation and flow of the implementation. LXC has many files working together and navigating through them was not easy. I found the '`grep`' command extremely helpful in this project.
- The core of LXC is hard to debug because of the lacking of debugging information. I tried using GDB step throughs to examine the assembly code but it was long and tedious. Valgrind did not help in this situation.
- The original core of LXC took an object-oriented approach in the C programming language that involved function pointers, which I have seen often in projects.
- When permissions are not met, the error messages are not explicit about it and it took me some time to figure out the permissions problem.
- Testing my implementation involved privileged operations so the code had to be installed in the root directory. I did not want to mess up my root directory so I made

a container using LXC and installed my version of LXC in the root directory of the container.

5 Future Development

The project is not production ready as it has not been throughly tested. There are also many aspects that can be further enhanced. Some of those are listed below.

- The public API can be further cleaned up. We can move the creation method to `lxc-create` and pass in an option to take an export as the base.
- This project can be further extended to export snapshots of a container. The basic idea would be similar and the only changes would be made to configuration files.
- Bindings for Python and Lua can be easily made for this extension since LXC ships the core functions as a shared library.

6 Conclusion

Linux Containers is an active project currently under heavy development. There is a lot of community support as container technology rises. It is a nice tool for testing and deployment since it creates a easily manangeable and easily reproduceable environment that is isolated from the host operating system.

There are many features that can be included into the project and I have tried integrating one of them. The results were satisfying as it introduces a new layer of abstraction in that we can know export a read-only container as a 'base image' that cannot be modified or started. We can save this container and create new containers from this 'stable version' of the container.

There are future development options as presented above as it is only a simple prototype that has not been tested throughly. Other features can be included into the project as well.