Linux Container Demo

Anand Jayaram
email: ajayara3@ncsu.edu , anandjyrm@gmail.com

July 23, 2017

This demo is meant to be an introduction into the world of Containers for networking students. After completing this demo, you would have the tools to deploy custom VM container networks and manage them remotely. This guide is not meant to be comprehensive. Send me an email in case of questions/clarifications/mistakes. For more information about linux container theory, refer to the Text. For official documention refer to Official LXC,Official LXD, Official Docker, Official Docker, Stephane Graber LXC blogs and Stephane Graber LXD blogs. May Google guide you.

LXC

Scope of this demo

Creating privileged containers and setting up a simple container network. For unprivileged containers, check out official documentation. Privileged containers require root access. So the lxc- commands will be prefaced by sudo. There could be other ways/commands to do the given demo tasks, feel free to explore them.

1. Installation

1.a Linux Environment Setup

We will be using Ubuntu 16.04 LTS on AWS EC2 machine for this demo. This is to ensure consistency and convenience, you can freely mess around in this environment. You can use any Linux Machine as long as you have terminal and Internet access. Refer to online docs, in case of issues regarding other distributions.

*** note on ubuntu version ***
Use `lsb_release -a (--all)` to check version of linux distribution Use `uname -a (--all)` to check kernel version

*** note on AWS EC2 setup ***
???

*** setting up VM with terminal and Internet Access ***
???

Other required packages:

- `bridge-utils` → `sudo apt install bridge-utils`
- text editor of your choice
- ip package → installed by default
- ssh package → installed by default
  Optional

- update (and upgrade everything)
- Remove all previous versions and packages of LXC, LXD, LXCFS and Docker using `sudo apt remove`

In case you have additional bridges/interfaces created on startup, you can remove them using the
`ip link set "interface" down`,
`ip link delete "interface"` and
`brctl delbr "bridge"`.
Verify Current setup using :
`ip link`
`brctl show`

1.b How to Install

`sudo apt install lxc`

1.c Default Configuration

Check output of :
`ip link`
`brctl show`
There should be a bridge called lxcbr0 currently UP with an IP address present. This is the default bridge created by LXC. The details of this bridge are stored in this location:
`/etc/default/lxc-net`
This bridge will be created and started each time you start the system, unless you modify this file. This file also configures the DHCP and DNS settings for the lxcbr0 bridge. Remember to create a backup of this file in case you are modifying it, since changes are permanent.

In case the bridge is absent, you might face issues while starting your container, explained in the coming sections.

2. Creating a container

`sudo lxc-create -t IMG -n CNTNR (--template and --name)`

Here IMG could be a image like ubuntu, alpine, etc. and CNTNR is the container name.
To verify
`sudo lxc-ls -f (--fancy)`
The output should show a stopped container called CNTNR. Issues you could face:

- Container already exists in which case create a container with a new name.

2.a Templates

The templates for creating the OS containers are stored in this location:

```
/usr/share/lxc/templates/lxc-
```

They are named starting with lxc-. The files are bash scripts which create rootfs directory for the container. In the above command the IMG can be replaced with any of the template names without the lxc-. The first time the command is used with a particular template the actual rootfs of the container CNTNR is created from the template and cached for later use at:

```
/var/cache/lxc/
```

Subsequent uses of lxc-create copies the rootfs from cache to

```
/var/lib/lxc/CNTNR/
```

where the actual container resides.

2.b Default configs

The config file for the container is created from a combination of files. The final version used in the container is stored in this location

```
/var/lib/lxc/CNTNR/config
```

This file can be modified before starting the container for desired effects. Modify or create

```
/etc/lxc/default.conf
```

for changing default network configurations of all newly created containers.

2.c Sharing Volumes

Sharing volumes between host and container. https://en.opensuse.org/User:Tsu2/LXC_mount_shared_directory

3. Accessing/Running Containers

To start the container:

```
sudo lxc-launch -n CNTNR
```

To verify:

```
sudo lxc-ls -f
```

Issues you could face:

- The config file has a bridge and the bridge is not actually up. Either change the config file/ bring up the bridge
- Already running same container. Create a container with new name

To execute commands on the container:

```
sudo lxc-execute -n CNTNR CMD
```

To accessing terminal of the container:

- `lxc-attach -n CNTNR`
  This gives you a terminal with root access. To return to host terminal type `exit`.
- `sudo lxc-console -n CNTNR` This gives you a console from the container. Login using the credentials obtained when creating container (eg. username Ubuntu, password Ubuntu). To exit type `Ctrl a + q`.

4. Configuring the container network

4.a Default containers

By default the /etc/default/default.conf dictates the network configuration. If the default file was empty, the config file for the new container would have an empty network configuration and the container would have only a loopback interface. If the config file has a veth interface and linked to lxcbr0, then the container would have an ethernet interface. By default lxc configures lxcbr0 with DHCP, NAT and DNS, therefore the container would have have network access to the host machine and internet.

4.b Containers with hanging veth interfaces

Modify the network configuration file before starting the container to get a hanging veth interface. You can add as many interfaces as you want here.

4.c Containers connected to linux bridges

Modiy the network configuration file before starting the container to connect to any linux bridge. You can add as many interfaces and bridges as you want.

5. Attaching interfaces to running containers

The previous step mentions how to attach interfaces to containers before starting them. This step lets you attach Virtual Ethernet Interfaces to Containers.

5.a Extracting namespace of containers

Each container is a process and each process runs in its own namespaces. most host processes run in the root namespaces. Check sudo ls -l /proc/"PID"/ns/. To allow manipulating the network namespace of a process we need to link the network namespace to the ip netns folder. we create /var/run/netns using mkdir -p /vr/run/netns get the pid of the container using ps -ef —grep "conainer name" link the n/w namespace of the container process to a netns namespace sudo ln -s /proc/PID/ns/net /var/run/netns/containerNamespaceName to verify ip netns show ip netns exec containerNamespaceName ifconfig -a will have the same network as the container.

5.b Creating Virtual Ethernet Interfaces

Virtual Ethernet Interfaces or veth pairs can be created to connect stuff. ip link add veth0 type veth peer name veth1

5.c Attaching Interfaces to everything

ip link set veth0 netns containerNamespacename brctl addif bridgename veth1 ip link set veth1 netns another namespace ovs-vsctl add-port br-int veth1

to verify run ip link , ovsvsctl show, brctl show, ip netns exec ns ip link etc on host or ip link on container try pinging stuff

Now we have created L2 and L3 datapaths for the containers.

5.d lxc-device add

6. Enabling DHCP, NAT and DNS for the containers

by default the lxcbr0 bridge does all this for the containers. You can set up a linux bridge of your own and enable all these from the hosts and connect a container to it. here are the steps for that ???

7. Managing containers

sudo lxc-stop sudo lxc-wait sudo lxc-pause

remove stuff using var/lib/lxc check all options here

????

8. Automation using scripts/Ansible

Since all these commands were run from the CLI, you can technically automate the whole process. You can remotely do this if you have ssh access to a machine. therefore you could even use ansible playbooks to do it.

python/bash script to create a 3 router network each router connected to a linux switch which in turn is connected to 2 containers each. Ansible playbook to do some other stuff on a remote machine

LXD

Scope of this demo

This demo is intended to show the CLI provided by LXD, which is a container management library which uses LXC. The LXC Commands all start with "lxc-...", the LXD commands to manipulate containers begin with "lxc ...". You also have "lxc image ..." commands to manipulate images and "lxd ..." commands for the lxd daemon itself.

Setup and Installation

Setup is the same as before, we can continue from the previous demo. Install lxd using sudo apt install lxd. The lxd-client package will also be installed along with it.

Creating and configuring containers

Creating new network profiles

Publishing containers

LXD management tools

Remote Management using REST APIs

Docker

Scope of this demo

Installation

Version

How to install

Default Configs

Starting VM containers

Docker run

Exposing ports

Sharing Volumes

Accessing Running Containers

Managing docker container networks

Managing Containers

Saving, stopping, pausing, push

Managing Images

Docker hub, publishing

Building from a Dockerfile

App containers

Remote Management using REST APIs

Automation using scripts/Ansible