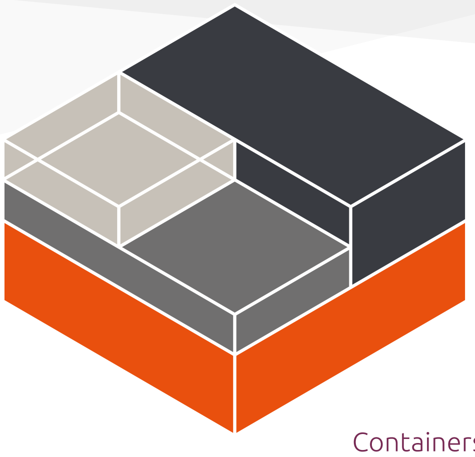
CANONICAL

How to build a lightweight system container cluster for your team

February 2019



Introduction

LXD is a system container manager developed by Canonical and shipped by default with Ubuntu. It makes it possible to create many containers of various Linux distributions and manage them in much the same way virtual machines would but without the overhead or cost usually associated to them.

Containers, unlike VMs, get all the benefits of using a shared kernel:

- Kernel security updates in the Ubuntu kernel
- Livepatch support
- Minimal memory footprint
- Ease of sharing resources (devices, disks, network…)
- Extremely low CPU usage/wakeups at idle

This means that in most cases, for non CPU intensive workloads, users can immediately enjoy 10x the density running the exact same workloads in containers vs virtual machines (VMs). For example; an average laptop will easily run 100 to 200 Ubuntu 18.04 containers but will typically struggle at running more than 20 or so virtual machines.

The focus of this paper is using LXD containers as part of a team development environment, effectively setting up a shared lab, either on physical hardware or in the cloud.

Such an environment is beneficial for a number of reasons:

- Reduces the time spent by team members getting a functional work environment
- Makes it easy to collaborate with colleagues, accessing their containers if needed
- Makes it possible to access the work environment of team members who are on leave
- Better use and control of resources by using shared systems
- Easy to implement snapshots and backups, huge time savers when a mistake happens

All team members will have access to the LXD setup and will be able to create containers as they see fit.

Limitations

A few things to keep in mind before we go deeper into how this works:

- As we will be using containers, only Linux workloads will work in this environment
- Workloads that require building/loading of custom kernel modules should also be considered out of scope for this
- LXD doesn't provide access control at this point, so all users of this setup will effectively have full access to LXD. This shouldn't be a problem in most small lab environments as described here
- For simplicity, the setup covered in this paper makes use of clustering but doesn't provide a highly available environment, if a cluster member goes offline, those containers will only be recoverable once it comes back up

Suggested configuration

Networking

LXD supports many different setups for networking, from passing a raw network interface, to allocating new ones using SR-IOV, to using Infiniband or simple point to point interfaces.

But the most common setup by far is to have LXD manage one or more bridges. Those bridges act like virtual switches with containers connected to them using a virtual ethernet connection.

You can then connect those bridges between multiple machines by attaching them to a physical interface (or VLAN) or a tunnelling protocol like VXLAN or GRE.

For the cloud setup, that will be discussed later, we'll be using the Ubuntu Fan overlay network which will provide one bridge per cluster member and routing between those bridges so that containers don't have to care what server they're running on.

Storage

LXD currently supports storing data on:

- ZFS
- btrfs
- plain directory
- LVM2 (block based with btrfs, ext4 or xfs on top)
- CEPH (block based with btrfs, ext4 or xfs on top)

For all except CEPH, LXD supports setting up a simple loop storage, which lets you set everything up without having to think about disks and partitions. This, however, is not something that's recommended in production and a separate disk or partition should be used instead.

All storage backends come with their pros and cons and a good overview of those can be found here.

CEPH deserves a special mention here as that is the only option that would provide an HA environment for a LXD cluster by having all containers and images replicated, it is however, not trivial to setup and will be slower than the other alternatives due to network latency and replication delays.

In our case, we'll be using ZFS with a dedicated disk for container storage. ZFS is very fast, supports quotas and can be further tweaked and optimised easily after setup.

Resource limits & isolation

The last thing wanted in a shared hosting environment is for one user's container to be able to take the entire system down by running it out of RAM, CPU or disk space.

To prevent this from happening, we'll be setting up a few limits in the **default** profile. This profile is automatically applied to all new containers. Containers can then be configured to raise any relevant limit to better match their needs.

A reasonable set of initial limits would be:

- 5GB disk space (ZFS referenced space)
- 512MB of RAM
- 2 CPU cores
- 1000 processes
- 100Mbit/s network bandwidth

We'll also configure containers to be unprivileged (default) and use an isolated range of uid/gid.

The latter part is important in order to avoid intentional or accidental denial of service attacks by running the kernel out of a resource controlled by user limits.

However, it does come with the downside of making it harder to share data between containers, so if data sharing between containers is important to you, you may want to turn that off on a per-container basis.

Authentication

LXD supports two ways to authenticate users.

The default method is to use TLS with client certificates. The command line client will generate one on first use which then needs to be added to the server's trust store. This can be done either by using a "trust" password that's shared with the user and allows them to self-add their certificate, or the user's certificate can be provided to an administrator that will then directly add it to the trust store.

The new method is to use Candid, a separate authentication gateway developed by Canonical which supports a variety of identity providers. With this in place, a new user will be redirected to a web page on first connection where they will have to authenticate using their company credentials before being allowed to interact with LXD.

Projects

One frustrating and time consuming element of a shared environment can be figuring out what belongs to who and avoiding potential naming clashes.

LXD projects are a great solution for this. Each project contains its own set of containers and may also contain its own profiles and images.

In our setup, we'll be using the "default" project only for global services and then create one project per user, keeping containers nicely organised.

For ease of management, separate images and profiles won't be used, instead these will be kept common to all projects. If you expect your users to generate their own images from their containers or to have enough containers that moving shared configuration into their own profiles makes sense, you may want to enable those features.

Clustering

LXD clustering was the headline feature of LXD 3.0, it allows turning up to 100 individual systems into one big LXD cluster.

This is ideal in this case as it means team members will only interact with one thing, the cluster, which can then be grown or shrunk as needed. It's also much easier to manage as configuration only needs to happen once for the entire cluster and LXD will take care of load-balancing containers as they get created.

Interaction from clients is identical to interacting with a standalone server except for a few extra fields here and there indicating on what cluster member the container is actually stored.

For example, a container listing on a cluster looks like:

```
stgraber@lxd05:~$ lxc list
+-------+---------+--------------------+-------+------------+-----------+----------+
| NAME  |  STATE  |        IPV4        | IPV6  |    TYPE    | SNAPSHOTS | LOCATION |
+-------+---------+--------------------+-------+------------+-----------+----------+
| u1    | RUNNING | 240.0.2.70 (eth0)  |       | PERSISTENT |           | lxd01    |
+-------+---------+--------------------+-------+------------+-----------+----------+
| u2    | RUNNING | 240.0.3.118 (eth0) |       | PERSISTENT |           | lxd02    |
+-------+---------+--------------------+-------+------------+-----------+----------+
| u3    | RUNNING | 240.0.5.54 (eth0)  |       | PERSISTENT |           | lxd04    |
+-------+---------+--------------------+-------+------------+-----------+----------+
| u4    | RUNNING | 240.0.4.160 (eth0) |       | PERSISTENT |           | lxd03    |
+-------+---------+--------------------+-------+------------+-----------+----------+
| u5    | RUNNING | 240.0.6.31 (eth0)  |       | PERSISTENT |           | lxd05    |
+-------+---------+--------------------+-------+------------+-----------+----------+
```

(note the Location column indicating what machine the container is running on)

Note that if needed, it is possible to directly request a particular container be placed on a particular cluster member. This may be of particular use if only some of the cluster members have GPUs or similar additional hardware.

On the networking side, the use of the Ubuntu Fan makes container to container interactions within the cluster completely seamless, routing all traffic as needed and offering unified DNS.

LXD clustering doesn't require any external component, LXD's database will automatically get replicated on the three first nodes of the cluster, allowing for maintenance without bringing the entire cluster offline.

Setup on Google Cloud Platform

Now that we've covered everything we want to do, let's take a look at implementing it all.

In this case, we'll be using Google Cloud Platform as a readily available cloud provider which is known to work very well for these kinds of deployment. But the same concepts and instructions should apply to other cloud providers or even to a local deployment on physical hardware.

Instance types, storage and networking

The target is to provide enough resources for a team of 20 people, where each person is running a dozen containers, with minimal risks of things slowing down due to THE actions of a single team member.

Thanks to containers having a pretty low memory footprint, we expect to need more CPU than we do RAM in many cases and thanks to ZFS copy-on-write and built-in compression, we also don't need a whole lot of storage.

For this setup, we'll be getting five **n1-highcpu-8** instances, they each come with:

- Eight vCPUs
- 7.2GB of RAM

These will use the Ubuntu 18.04 server image and the standard 10GB of disk. Then we'll be attaching an additional 500GB SSD-backed disk to host LXD itself.

The estimated cost at time of writing is around $1000/month. That's for the five instances described above, including SSD storage and running in the us-east1-b region.

This configuration will provide us with a total of about 30GB of usable RAM, 2.5TB of storage and 40 vCPUs, which will very easily host our planned 240 containers and should be able to host several times that many - so long as most are idle.

For networking, the Ubuntu Fan bridge requires VXLAN communication between instances, this is usually achieved by making sure that they're on the same VPC and that the firewall won't interfere with that internal traffic. We also need to allow tcp/8443 from the outside to interact with the LXD cluster.

Networking

Assuming you're just using the default VPC, you'll need to allow ingress access to LXD with:

```
gcloud compute firewall-rules create allow-lxd --network default --action allow
--direction ingress --rules tcp:8443
```

Spawning the instances

First create a file called "lxd.yaml" containing the following cloud-init configuration:

```
#cloud-config
snap:
  commands:
    00: "apt-get remove --purge --yes lxd lxd-client liblxc1 lxcfs"
    01: "snap install lxd"

write_files:
  - content: |-
      # LXD specific sysctl tweaks (taken from upstream doc)
      fs.inotify.max_queued_events=1048576
      fs.inotify.max_user_instances=1048576
      fs.inotify.max_user_watches=1048576
      vm.max_map_count=262144
      kernel.dmesg_restrict=1
      net.ipv4.neigh.default.gc_thresh3=8192
      net.ipv6.neigh.default.gc_thresh3=8192
      kernel.keys.maxkeys=2000

    path: /etc/sysctl.d/99-lxd.conf
    permissions: "0644"

runcmd:
  - "sysctl -p /etc/sysctl.d/99-lxd.conf"
```

Copy code

Then create the actual instances using the Google Cloud SDK:

```
for i in $(seq -w 01 05); do
    gcloud compute instances create \
    "lxd${i}" --description="LXD cluster - member #${i}" \
    --boot-disk-size="20GB" \
    --create-disk=name="lxd${i}-zfs,size=500GB,type=pd-ssd,auto-delete=yes" \
    --machine-type="n1-highcpu-8" \
    --image-family="ubuntu-1804-lts" --image-project="ubuntu-os-cloud" \
    --zone="us-east1-b" \
    --metadata-from-file="user-data=lxd.yaml"
done
```

Copy code

This will get you five instances with internal and public IPs - as can be confirmed with "gcloud compute instances list":

```
stgraber@castiana:~$ gcloud compute instances list
NAME   ZONE        MACHINE_TYPE  PREEMPTIBLE  INTERNAL_IP  EXTERNAL_IP     STATUS
lxd01  us-east1-b  n1-highcpu-8               10.142.0.2   35.196.213.247  RUNNING
lxd02  us-east1-b  n1-highcpu-8               10.142.0.3   35.237.28.231   RUNNING
lxd03  us-east1-b  n1-highcpu-8               10.142.0.4   35.196.151.239  RUNNING
lxd04  us-east1-b  n1-highcpu-8               10.142.0.5   35.196.249.182  RUNNING
lxd05  us-east1-b  n1-highcpu-8               10.142.0.6   35.237.231.120  RUNNING
```

Configuring the first instance

Connect to the first instance using SSH or the built-in remote shell, then run:

```
stgraber@lxd01:~$ sudo lxd init
Would you like to use LXD clustering? (yes/no) [default=no]: yes
What name should be used to identify this node in the cluster? [default=lxd01]:
What IP address or DNS name should be used to reach this node?
[default=10.142.0.2]:
Are you joining an existing cluster? (yes/no) [default=no]:
Setup password authentication on the cluster? (yes/no) [default=yes]:
Trust password for new clients:
Again:
Do you want to configure a new local storage pool? (yes/no) [default=yes]:
Name of the storage backend to use (btrfs, dir, lvm, zfs) [default=zfs]:
Create a new ZFS pool? (yes/no) [default=yes]:
Would you like to use an existing block device? (yes/no) [default=no]: yes
Path to the existing block device: /dev/sdb
Do you want to configure a new remote storage pool? (yes/no) [default=no]:
Would you like to connect to a MAAS server? (yes/no) [default=no]:
Would you like to configure LXD to use an existing bridge or host interface?
(yes/no) [default=no]:
Would you like to create a new Fan overlay network? (yes/no) [default=yes]:
What subnet should be used as the Fan underlay? [default=auto]: 10.142.0.0/16
Would you like stale cached images to be updated automatically? (yes/no)
[default=yes]
Would you like a YAML "lxd init" preseed to be printed? (yes/no) [default=no]:
```

You'll notice that the default values are usually sufficient, except for:

- Opting into clustering
- Choosing a trust password
- Selecting the ZFS block device
- Selecting the VPC subnet for the Ubuntu Fan (enter as /16 even if it's a /20)

And that's it, you have yourself a cluster of one node, now let's add more!

Configuring the other four instances

Now for each of the remaining four instances, connect to them and run:

```
stgraber@lxd02:~$ sudo lxd init
Would you like to use LXD clustering? (yes/no) [default=no]: yes
What name should be used to identify this node in the cluster? [default=lxd02]:
What IP address or DNS name should be used to reach this node?
[default=10.142.0.3]:
Are you joining an existing cluster? (yes/no) [default=no]: yes
IP address or FQDN of an existing cluster node: 10.142.0.2
Cluster fingerprint:
e5358b6f7d223a8d8ec1d2c8c681f07a3675e7fc29bc95340a046c1d3d4e75a6
You can validate this fingerprint by running "lxc info" locally on an existing
node.
Is this the correct fingerprint? (yes/no) [default=no]: yes
Cluster trust password:
All existing data is lost when joining a cluster, continue? (yes/no)
[default=no] yes
Choose "source" property for storage pool "local": /dev/sdb
Choose "volatile.initial_source" property for storage pool "local": /dev/sdb
Choose "zfs.pool_name" property for storage pool "local": local
Would you like a YAML "lxd init" preseed to be printed? (yes/no) [default=no]:
```

To confirm you're all done, run "lxc cluster list" from any of them:

```
stgraber@lxd05:~$ lxc cluster list
To start your first container, try: lxc launch ubuntu:18.04

+-------+-------------------------+----------+--------+-------------------+
| NAME  |           URL           | DATABASE | STATE  |      MESSAGE      |
+-------+-------------------------+----------+--------+-------------------+
| lxd01 | https://10.142.0.2:8443 | YES      | ONLINE | fully operational |
+-------+-------------------------+----------+--------+-------------------+
| lxd02 | https://10.142.0.3:8443 | YES      | ONLINE | fully operational |
+-------+-------------------------+----------+--------+-------------------+
| lxd03 | https://10.142.0.4:8443 | NO       | ONLINE | fully operational |
+-------+-------------------------+----------+--------+-------------------+
| lxd04 | https://10.142.0.5:8443 | YES      | ONLINE | fully operational |
+-------+-------------------------+----------+--------+-------------------+
| lxd05 | https://10.142.0.6:8443 | NO       | ONLINE | fully operational |
+-------+-------------------------+----------+--------+-------------------+
```

And congratulations, you're all done with the remote server configuration, from this point on, everything will happen through LXD!

Configuring your client

Install the LXD client ("lxc") on your machine. On Ubuntu and any other distribution supporting snaps, use "snap install lxd". The LXD client is also available for Windows and MacOS.

Once installed, add your cluster with "lxc remote add", using the public address of one of your instances (doesn't matter which):

```
stgraber@castiana:~$ lxc remote add cluster 35.196.151.239
Certificate fingerprint:
e5358b6f7d223a8d8ec1d2c8c681f07a3675e7fc29bc95340a046c1d3d4e75a6
ok (y/n)? y
Admin password for cluster:
Client certificate stored at server: cluster
stgraber@castiana:~$ lxc remote set-default cluster
```

This also makes it the default remote so you don't need to specifically target it.

Configuring the resource limits

The next step is to configure our default profile to apply the limits we discussed before.

```
stgraber@castiana:~$ lxc profile set default limits.cpu 2
stgraber@castiana:~$ lxc profile set default limits.memory 512MB
stgraber@castiana:~$ lxc profile set default limits.processes 1000
stgraber@castiana:~$ lxc profile device set default root size 5GB
stgraber@castiana:~$ lxc profile device set default eth0 limits.max 100Mbit
stgraber@castiana:~$ lxc storage set local volume.zfs.use_refquota true
```

Creating projects

Now that our configuration is done, all that's left is creating some projects. You may want to create one per user or just let your users create them as they see fit, up to you.

For now, let's create a project called "test" and only have it include containers, then "switch" to it so that container operations from this point on use that project:

```
stgraber@castiana:~$ lxc project create test -c features.profiles=false -c
features.images=false
Project test created
stgraber@castiana:~$ lxc project switch test
```

Using LXD

Now that it is all all setup and and the project is selected, you can start creating some containers. For now, let's just create five Ubuntu containers and five CentOS containers:

```
stgraber@castiana:~$ lxc launch ubuntu:18.04 u1
Creating u1
Starting u1
stgraber@castiana:~$ lxc launch ubuntu:18.04 u2
Creating u2
Starting u2
stgraber@castiana:~$ lxc launch ubuntu:18.04 u3
Creating u3
Starting u3
stgraber@castiana:~$ lxc launch ubuntu:18.04 u4
Creating u4
Starting u4
stgraber@castiana:~$ lxc launch ubuntu:18.04 u5
Creating u5
Starting u5
stgraber@castiana:~$ lxc launch images:centos/7 c1
Creating c1
Starting c1
stgraber@castiana:~$ lxc launch images:centos/7 c2
Creating c2
Starting c2
stgraber@castiana:~$ lxc launch images:centos/7 c3
Creating c3
Starting c3
stgraber@castiana:~$ lxc launch images:centos/7 c4
Creating c4
Starting c4
stgraber@castiana:~$ lxc launch images:centos/7 c5
Creating c5
Starting c5
```

You may notice creation being a bit slower than you may expect, that is because the containers get spread on the different cluster members and so the image also needs to be replicated.

That only happens the first time a particular image is used on a particular cluster member, so subsequent uses will be significantly faster.

You can then list the containers which will show what instance they're running on:

```
stgraber@lxd05:~$ lxc list
+-------+---------+----------------------+------+------------+-----------+----------+
| NAME  |  STATE  |         IPV4         | IPV6 |    TYPE    | SNAPSHOTS | LOCATION |
+-------+---------+----------------------+------+------------+-----------+----------+
| c1    | RUNNING | 240.0.2.175 (eth0)   |      | PERSISTENT |           | lxd01    |
+-------+---------+----------------------+------+------------+-----------+----------+
| c2    | RUNNING | 240.0.3.16 (eth0)    |      | PERSISTENT |           | lxd02    |
+-------+---------+----------------------+------+------------+-----------+----------+
| c3    | RUNNING | 240.0.5.239 (eth0)   |      | PERSISTENT |           | lxd04    |
+-------+---------+----------------------+------+------------+-----------+----------+
| c4    | RUNNING | 240.0.4.233 (eth0)   |      | PERSISTENT |           | lxd03    |
+-------+---------+----------------------+------+------------+-----------+----------+
| c5    | RUNNING | 240.0.6.160 (eth0)   |      | PERSISTENT |           | lxd05    |
+-------+---------+----------------------+------+------------+-----------+----------+
| u1    | RUNNING | 240.0.2.70 (eth0)    |      | PERSISTENT |           | lxd01    |
+-------+---------+----------------------+------+------------+-----------+----------+
| u2    | RUNNING | 240.0.3.118 (eth0)   |      | PERSISTENT |           | lxd02    |
+-------+---------+----------------------+------+------------+-----------+----------+
| u3    | RUNNING | 240.0.5.54 (eth0)    |      | PERSISTENT |           | lxd04    |
+-------+---------+----------------------+------+------------+-----------+----------+
| u4    | RUNNING | 240.0.4.160 (eth0)   |      | PERSISTENT |           | lxd03    |
+-------+---------+----------------------+------+------------+-----------+----------+
| u5    | RUNNING | 240.0.6.31 (eth0)    |      | PERSISTENT |           | lxd05    |
+-------+---------+----------------------+------+------------+-----------+----------+
```

And from that point on, everything works exactly as if those containers were local to your machine. You can access their console, get a shell inside them, transfer files, add devices, make snapshots, make backups and more.

Giving access to team members

For your initial client, you've used the "trust password", same as was used to bootstrap the cluster. You can keep using that mechanism with anyone else who needs access to your new LXD cluster, but if you don't like passwords, you also have the option to manually trust the client certificate of whoever you want to add.

To do that, have them send you their "client.crt" file, which is present either:

`~/snap/lxd/current/.config/lxc/client.crt`

`~/.config/lxc/client.crt`

Then on your own client, run "`lxc config trust add /path/to/client.crt`"

If using this over the trust password, you'll likely want to unset the trust password entirely with "`lxc config unset core.trust_password`"

Additional configuration to consider

Clustered storage with CEPH

The cluster as setup in this document will have a clustered database spread across three instances so the record of what's where, your configuration, will all be safe.

The data however is stored on ZFS directly attached to the different instances.

Should one of those get corrupted or an instance lost, all containers that were stored on it will be gone forever.

This may be reasonable for a lab or test environment, for others, you'll either want to setup backups for that ZFS pool or consider setting up a CEPH cluster and then have LXD use CEPH for container storage.

When combined with CEPH, a LXD cluster will perfectly tolerate the loss of a server.

If that happens, you can use "lxc move" to move any container which was on that server onto one of the other servers in the cluster, then "lxc start" it and you'll be back in business, no data recovery needed.

Automated snapshots

Taking snapshots of your containers, when running on top of ZFS, is near instantaneous and only uses minimal disk space. They are therefore a cheap way to recover from a mistake.

LXD recently introduced support for automated snapshots, where a CRON time pattern is set through "snapshots.schedule", either in individual container's configuration or in a profile.

This makes it easy to have hourly or daily snapshots taken for all your containers.

Multiple networks

In the scenario we ran, we've only used a single network, "fanbr0" with all containers attached directly to it. This simple setup works well in this case but for more complex environments you may want to create additional networks then have some containers use different networks.

This can be useful to setup a separate network directly connecting a few containers or to set up separate networks for every project on your cluster.

All this can be done through the "lxc network" command at any point in time.

Centralised authentication

For now we've opted to use the default TLS based authentication which tends to be easiest, especially in a cloud environment where access to your company's authentication system may be tricky.

But as you grow the set of people with access to the cluster, it will no doubt be a good idea to setup a Candid server, have it configured to authenticate against your company's authentication system and then use that to control who can interact with the cluster.

A tutorial on setting up Candid can be found here:
https://tutorials.ubuntu.com/tutorial/candid-authentication-lxd

Conclusion

For less than $1000/month, you can get your team onto a LXD cluster running in the cloud, with easy access from anywhere and using the exact same tools they'd use locally.

Those LXD containers can run any number of Linux distributions and releases of those, they're easy to manage and very low on resource consumption.

It's a great way to standardise your development or test environment onto a smaller number of cloud instances, reducing cost, simplifying cloud usage accounting and reducing the amount of time a team member needs to get up and running.

LXD on Ubuntu 18.04 LTS is fully supported by Canonical, for both host and guest and can be combined with kernel live-patching to reduce the amount of maintenance needed.

Documentation & support

The official LXD documentation can be found here: https://lxd.readthedocs.io/en/latest/

A community forum is available here: https://discuss.linuxcontainers.org/

Bug reports can be filed at: https://github.com/lxc/lxd/issues

And support for LXD comes as part of the Standard Ubuntu Advantage Server subscription:

https://www.ubuntu.com/support/plans-and-pricing#ua-support

ubuntu
Delivered by Canonical