Creating and starting a new container

As I mentioned in the previous posts, the LXD command line client comes pre-configured with a few image sources. Ubuntu is the best covered with official images for all its releases and architectures but there also are a number of unofficial images for other distributions. Those are community generated and maintained by LXC upstream contributors.

Ubuntu

If all you want is the best supported release of Ubuntu, all you have to do is:

```
lxc launch ubuntu:
```

Note however that the meaning of this will change as new Ubuntu LTS releases are released. So for scripting use, you should stick to mentioning the actual release you want (see below).

Ubuntu 14.04 LTS

To get the latest, tested, stable image of Ubuntu 14.04 LTS, you can simply run:

```
lxc launch ubuntu:14.04
```

In this mode, a random container name will be picked.
If you prefer to specify your own name, you may instead do:

```
lxc launch ubuntu:14.04 c1
```

Should you want a specific (non-primary) architecture, say a 32bit Intel image, you can do:

```
lxc launch ubuntu:14.04/i386 c2
```

Current Ubuntu development release

The “ubuntu:” remote used above only provides official, tested images for Ubuntu. If you instead want untested daily builds, as is appropriate for the development release, you’ll want to use the “ubuntu-daily:” remote instead.

```
lxc launch ubuntu-daily:devel c3
```

In this example, whatever the latest Ubuntu development release is will automatically be picked.

You can also be explicit, for example by using the code name:

```
lxc launch ubuntu-daily:xenial c4
```

Latest Alpine Linux

Alpine images are available on the “images:” remote and can be launched with:

```
lxc launch images:alpine/3.3/amd64 c5
```

And many more

A full list of the Ubuntu images can be obtained with:

```
lxc image list ubuntu:

lxc image list ubuntu-daily:
```

And of all the unofficial images:

```
lxc image list images:
```

A list of all the aliases (friendly names) available on a given remote can also be obtained with (for the “ubuntu:” remote):

```
lxc image alias list ubuntu:
```

Creating a container without starting it

If you want to just create a container or a batch of container but not also start them immediately, you can just replace “lxc launch” by “lxc init”. All the options are identical, the only different is that it will not start the container for you after creation.

```
lxc init ubuntu:
```

Information about your containers

Listing the containers

To list all your containers, you can do:

```
lxc list
```

There are a number of options you can pass to change what columns are displayed. On systems with a lot of containers, the default columns can be a bit slow (due to having to retrieve network information from the containers), you may instead want:

```
lxc list --fast
```

Which shows a different set of columns that require less processing on the server side.

You can also filter based on name or properties:

```
stgraber@dakara:~$ lxc list security.privileged=true

+------+---------+---------------------+-----------------------------
--------------------+------------+-----------+

| NAME |  STATE  |         IPV4          |                          IPV6
|    TYPE    | SNAPSHOTS |

+------+---------+---------------------+-----------------------------
--------------------+------------+-----------+

| suse | RUNNING | 172.17.0.105 (eth0) |
2607:f2c0:f00f:2700:216:3eff:fef2:aff4 (eth0) | PERSISTENT | 0
|

+------+---------+---------------------+-----------------------------
--------------------+------------+-----------+
```

In this example, only containers that are privileged (user namespace disabled) are listed.

```
stgraber@dakara:~$ lxc list --fast alpine
```

```
+-------------+---------+-------------+----------------------+----
------+------------+

|    NAME     |  STATE  | ARCHITECTURE |      CREATED AT      |
PROFILES |    TYPE    |

+-------------+---------+-------------+----------------------+----
------+------------+

| alpine      | RUNNING | x86_64       | 2016/03/20 02:11 UTC |
default  | PERSISTENT |

+-------------+---------+-------------+----------------------+----
------+------------+

| alpine-edge | RUNNING | x86_64       | 2016/03/20 02:19 UTC |
default  | PERSISTENT |

+-------------+---------+-------------+----------------------+----
------+------------+
```

And in this example, only the containers which have “alpine” in their names (complex regular expressions are also supported).

Getting detailed information from a container

As the list command obviously can’t show you everything about a container in a nicely readable way, you can query information about an individual container with:

```
lxc info <container>
```

For example:

```
stgraber@dakara:~$ lxc info zerotier

Name: zerotier

Architecture: x86_64
```

```
Created: 2016/02/20 20:01 UTC

Status: Running

Type: persistent

Profiles: default

Pid: 31715

Processes: 32

Ips:

 eth0: inet 172.17.0.101

 eth0: inet6 2607:f2c0:f00f:2700:216:3eff:feec:65a8

 eth0: inet6 fe80::216:3eff:feec:65a8

 lo: inet 127.0.0.1

 lo: inet6 ::1

 lxcbr0: inet 10.0.3.1

 lxcbr0: inet6 fe80::c0a4:ceff:fe52:4d51

 zt0: inet 29.17.181.59

 zt0: inet6 fd80:56c2:e21c:0:199:9379:e711:b3e1

 zt0: inet6 fe80::79:e7ff:fe0d:5123

Snapshots:

 zerotier/blah (taken at 2016/03/08 23:55 UTC) (stateless)
```

Life-cycle management commands

Those are probably the most obvious commands of any container or virtual machine manager but they still need to be covered.

Oh and all of them accept multiple container names for batch operation.

start

Starting a container is as simple as:

```
lxc start <container>
```

stop

Stopping a container can be done with:

```
lxc stop <container>
```

If the container isn't cooperating (not responding to SIGPWR), you can force it with:

```
lxc stop <container> --force
```

restart

Restarting a container is done through:

```
lxc restart <container>
```

And if not cooperating (not responding to SIGINT), you can force it with:

```
lxc restart <container> --force
```

pause

You can also "pause" a container. In this mode, all the container tasks will be sent the equivalent of a SIGSTOP which means that they will still be visible and will still be using memory but they won't get any CPU time from the scheduler.

This is useful if you have a CPU hungry container that takes quite a while to start but that you aren't constantly using. You can let it start, then pause it, then start it again when needed.

```
lxc pause <container>
```

delete

Lastly, if you want a container to go away, you can delete it for good with:

```
lxc delete <container>
```

Note that you will have to pass "–force" if the container is currently running.

Container configuration

LXD exposes quite a few container settings, including resource limitation, control of container startup and a variety of device pass-through options. The full list is far too long to cover in this post but it's available here.
As far as devices go, LXD currently supports the following device types:

- **disk**
  This can be a physical disk or partition being mounted into the container or a bind-mounted path from the host.
- **nic**
  A network interface. It can be a bridged virtual ethernet interrface, a point to point device, an ethernet macvlan device or an actual physical interface being passed through to the container.
- **unix-block**
  A UNIX block device, e.g. /dev/sda
- **unix-char**
  A UNIX character device, e.g. /dev/kvm
- **none**
  This special type is used to hide a device which would otherwise be inherited through profiles.

Configuration profiles

The list of all available profiles can be obtained with:

```
lxc profile list
```

To see the content of a given profile, the easiest is to use:

```
lxc profile show <profile>
```

And should you want to change anything inside it, use:

```
lxc profile edit <profile>
```

You can change the list of profiles which apply to a given container with:

```
lxc profile apply <container> <profile1>,<profile2>,<profile3>,...
```

Local configuration

For things that are unique to a container and so don't make sense to put into a profile, you can just set them directly against the container:

```
lxc config edit <container>
```

This behaves the exact same way as "profile edit" above.

Instead of opening the whole thing in a text editor, you can also modify individual keys with:

```
lxc config set <container> <key> <value>
```

Or add devices, for example:

```
lxc config device add my-container kvm unix-char path=/dev/kvm
```

Which will setup a /dev/kvm entry for the container named "my-container".

The same can be done for a profile using "lxc profile set" and "lxc profile device add".

Reading the configuration

You can read the container local configuration with:

```
lxc config show <container>
```

Or to get the expanded configuration (including all the profile keys):

```
lxc config show --expanded <container>
```

For example:

```
stgraber@dakara:~$ lxc config show --expanded zerotier

name: zerotier

profiles:

- default

config:

 security.nesting: "true"

 user.a: b

 volatile.base_image:
a49d26ce5808075f5175bf31f5cb90561f5023dcd408da8ac5e834096d46b2d8

 volatile.eth0.hwaddr: 00:16:3e:ec:65:a8

 volatile.last_state.idmap:
'[{"Isuid":true,"Isgid":false,"Hostid":100000,"Nsid":0,"Maprange":6
5536},{"Isuid":false,"Isgid":true,"Hostid":100000,"Nsid":0,"Maprang
e":65536}]'

devices:

 eth0:

  name: eth0

  nictype: macvlan

  parent: eth0
```

```
  type: nic
  limits.ingress: 10Mbit
  limits.egress: 10Mbit
 root:
  path: /
  size: 30GB
  type: disk
 tun:
  path: /dev/net/tun
  type: unix-char
ephemeral: false
```

That one is very convenient to check what will actually be applied to a given container.

Live configuration update

Note that unless indicated in the documentation, all configuration keys and device entries are applied to affected containers live. This means that you can add and remove devices or alter the security profile of running containers without ever having to restart them.

Getting a shell

LXD lets you execute tasks directly into the container. The most common use of this is to get a shell in the container or to run some admin tasks.

The benefit of this compared to SSH is that you're not dependent on the container being reachable over the network or on any software or configuration being present inside the container.

Execution environment

One thing that's a bit unusual with the way LXD executes commands inside the container is that it's not itself running inside the container, which means that it can't know what shell to use, what environment variables to set or what path to use for your home directory.

Commands executed through LXD will always run as the container's root user (uid 0, gid 0) with a minimal PATH environment variable set and a HOME environment variable set to /root.

Additional environment variables can be passed through the command line or can be set permanently against the container through the "environment.<key>" configuration options.

Executing commands

Getting a shell inside a container is typically as simple as:

```
lxc exec <container> bash
```

That's assuming the container does actually have bash installed.

More complex commands require the use of a separator for proper argument parsing:

```
lxc exec <container> -- ls -lh /
```

To set or override environment variables, you can use the "–env" argument, for example:

```
stgraber@dakara:~$ lxc exec zerotier --env mykey=myvalue env | grep
mykey

mykey=myvalue
```

Managing files

Because LXD has direct access to the container's file system, it can directly read and write any file inside the container. This can be very useful to pull log files or exchange files with the container.

Pulling a file from the container

To get a file from the container, simply run:

```
lxc file pull <container>/<path> <dest>
```

For example:

```
stgraber@dakara:~$ lxc file pull zerotier/etc/hosts hosts
```

Or to read it to standard output:

```
stgraber@dakara:~$ lxc file pull zerotier/etc/hosts -

127.0.0.1 localhost



# The following lines are desirable for IPv6 capable hosts

::1 ip6-localhost ip6-loopback

fe00::0 ip6-localnet

ff00::0 ip6-mcastprefix

ff02::1 ip6-allnodes

ff02::2 ip6-allrouters

ff02::3 ip6-allhosts
```

Pushing a file to the container

Push simply works the other way:

```
lxc file push <source> <container>/<path>
```

Editing a file directly

Edit is a convenience function which simply pulls a given path, opens it in your default text editor and then pushes it back to the container when you close it:

```
lxc file edit <container>/<path>
```

Snapshot management

LXD lets you snapshot and restore containers. Snapshots include the entirety of the container's state (including running state if –stateful is used), which means all container configuration, container devices and the container file system.

Creating a snapshot

You can snapshot a container with:

```
lxc snapshot <container>
```

It'll get named snapX where X is an incrementing number.

Alternatively, you can name your snapshot with:

```
lxc snapshot <container> <snapshot name>
```

Listing snapshots

The number of snapshots a container has is listed in "lxc list", but the actual snapshot list is only visible in "lxc info".

```
lxc info <container>
```

Restoring a snapshot

To restore a snapshot, simply run:

```
lxc restore <container> <snapshot name>
```

Renaming a snapshot

Renaming a snapshot can be done by moving it with:

```
lxc move <container>/<snapshot name> <container>/<new snapshot name>
```

Creating a new container from a snapshot

You can create a new container which will be identical to another container's snapshot except for the volatile information being reset (MAC address):

```
lxc copy <source container>/<snapshot name> <destination container>
```

Deleting a snapshot

And finally, to delete a snapshot, just run:

```
lxc delete <container>/<snapshot name>
```

Cloning and renaming

Getting clean distribution images is all nice and well, but sometimes you want to install a bunch of things into your container, configure it and then branch it into a bunch of other containers.

Copying a container

To copy a container and effectively clone it into a new one, just run:

```
lxc copy <source container> <destination container>
```

The destination container will be identical in every way to the source one, except it won't have any snapshot and volatile keys (MAC address) will be reset.

Moving a container

LXD lets you copy and move containers between hosts, but that will get covered in a later post.

For now, the “move” command can be used to rename a container with:

```
lxc move <old name> <new name>
```

The only requirement is that the container be stopped, everything else will be kept exactly as it was, including the volatile information (MAC address and such).

Conclusion

This pretty long post covered most of the commands you’re likely to use in day to day operation.

Obviously a lot of those commands have extra arguments that let you be more efficient or tweak specific aspects of your LXD containers. The best way to learn about all of those is to go through the help for those you care about (–help).