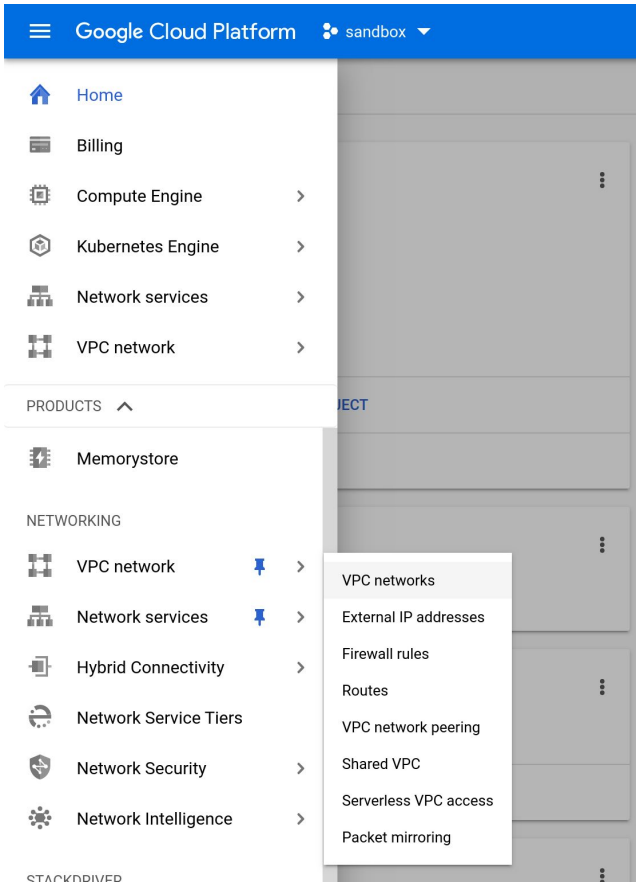
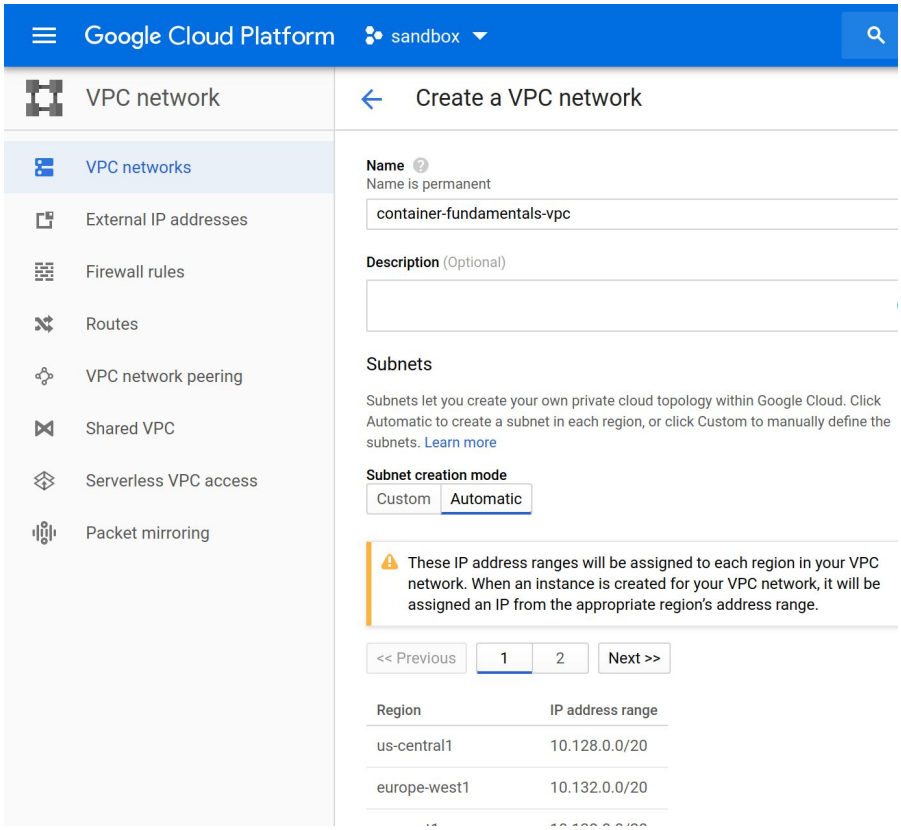
Lab 1.1 - Google Compute Engine Environment Setup

This guide exemplifies the setup steps for the lab environment on the Google Cloud Platform (GCP), more precisely the creation of a new VPC network with a new firewall rule, followed by a new Google Compute Engine (GCE) VM instance. The following setup steps are similar for either GCP account type.

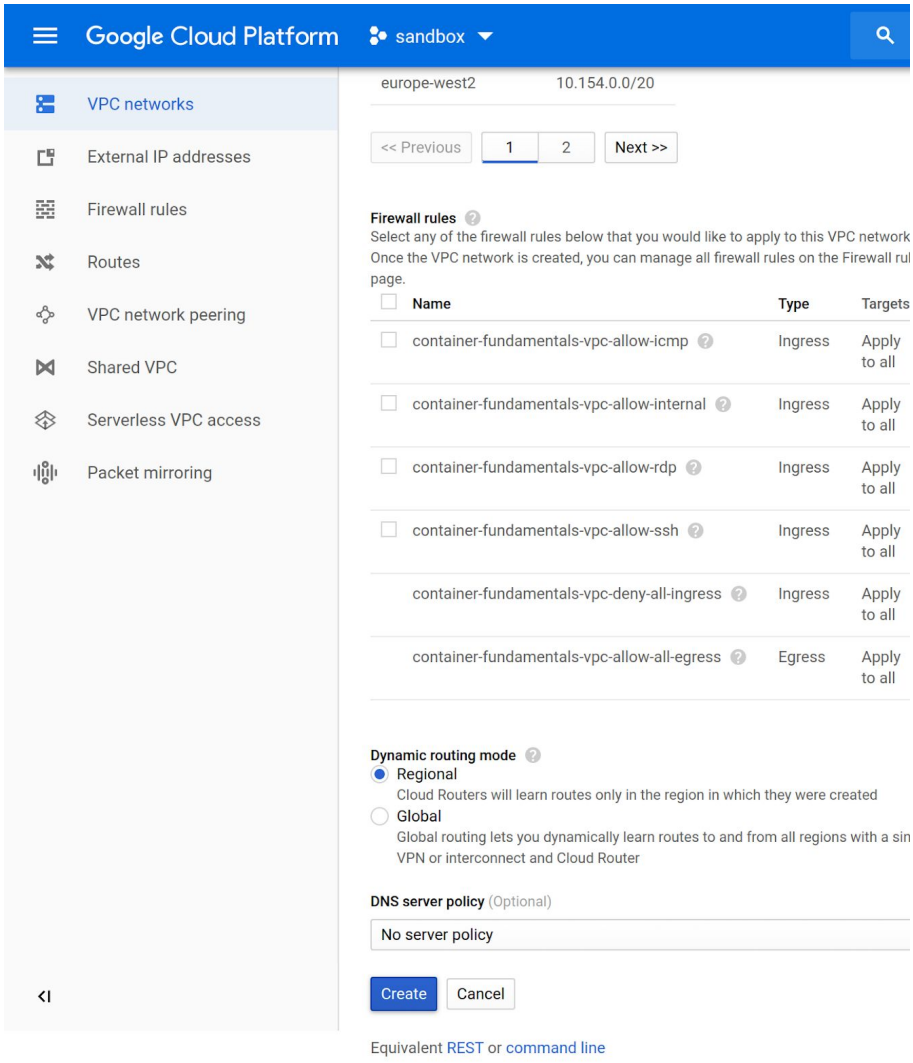
On the GCP Console select the **Navigation menu** from the top left corner.
Scroll down to the **NETWORKING** section, select **VPC network**, then select **VPC networks** from the sub-menu.



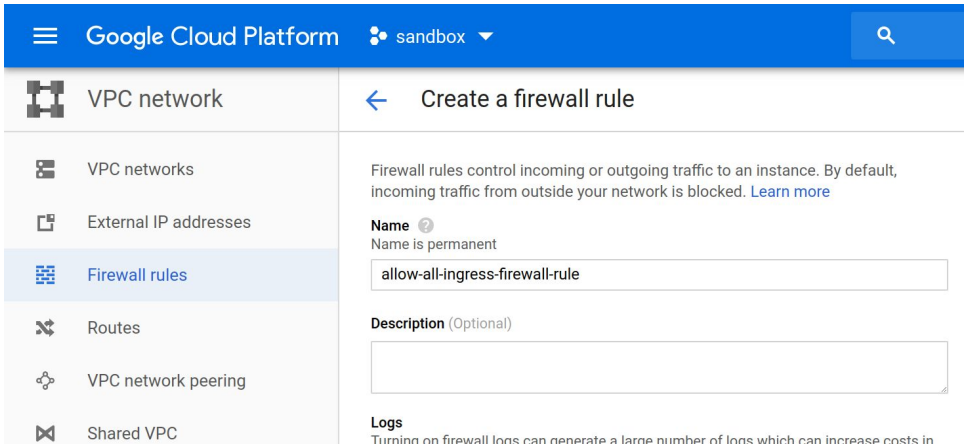
On the VPC network screen select **CREATE VPC NETWORK**.
Provide a name.
Select **Automatic** Subnet creation mode.



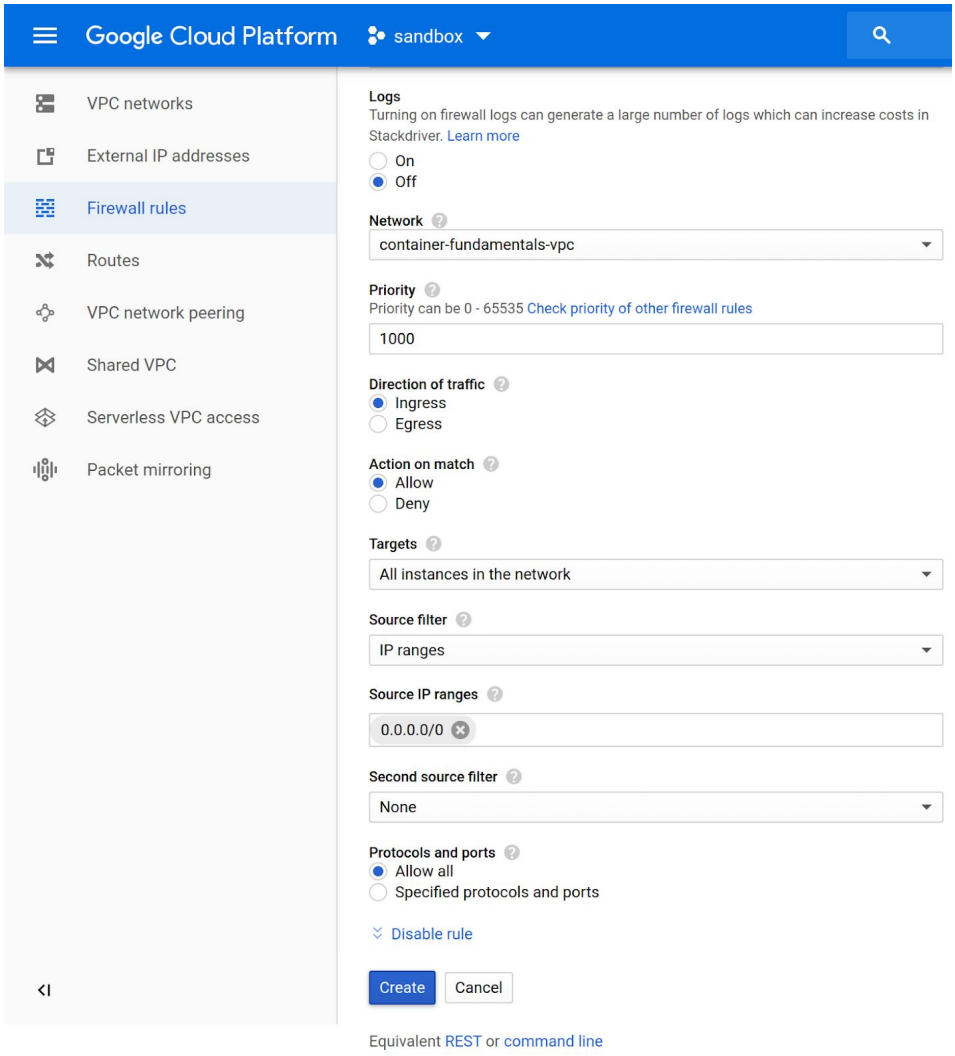
Leave Firewall rules unchecked. We will create a firewall rule later. Select **Create** to finalize the creation of the VPC network.



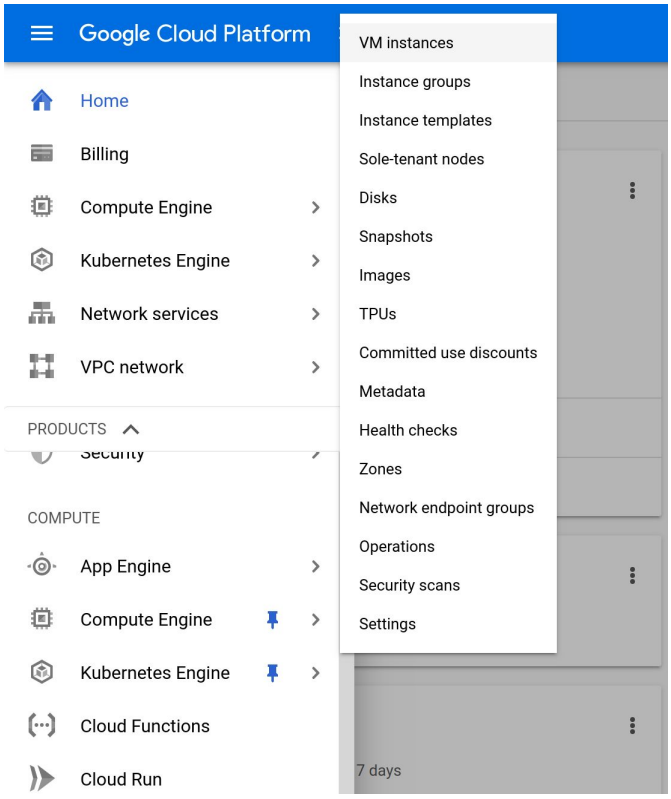
Select from the menu **Firewall rules**.
On the Firewall rules screen select **CREATE FIREWALL RULE**.
Provide a name.



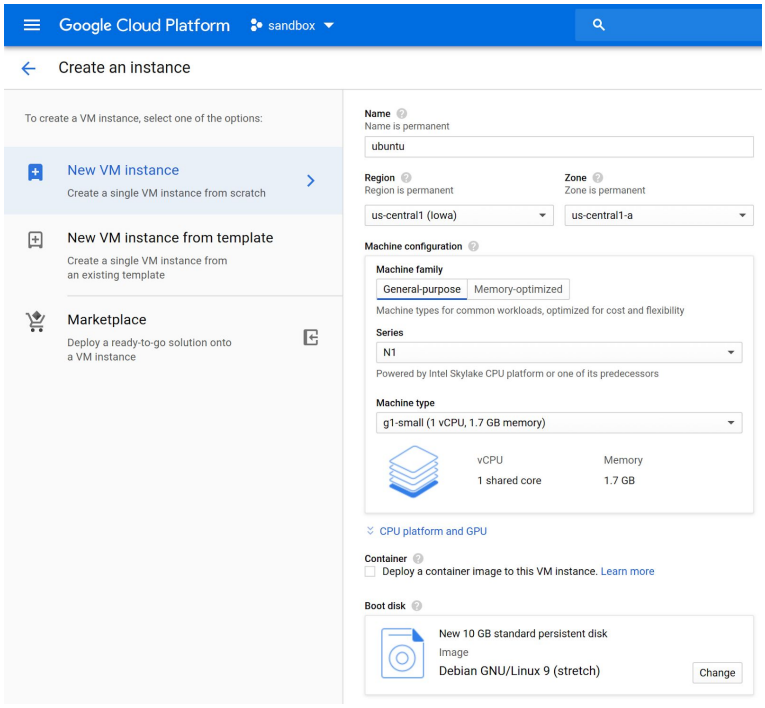
Select from the Network dropdown list the VPC network created earlier.
Direction of traffic: **Ingress**.
Action: **Allow**.
Source IP ranges: **0.0.0.0/0**
Protocols and Ports: **Allow all**.
Select **Create** to finalize the creation of the Firewall rule.



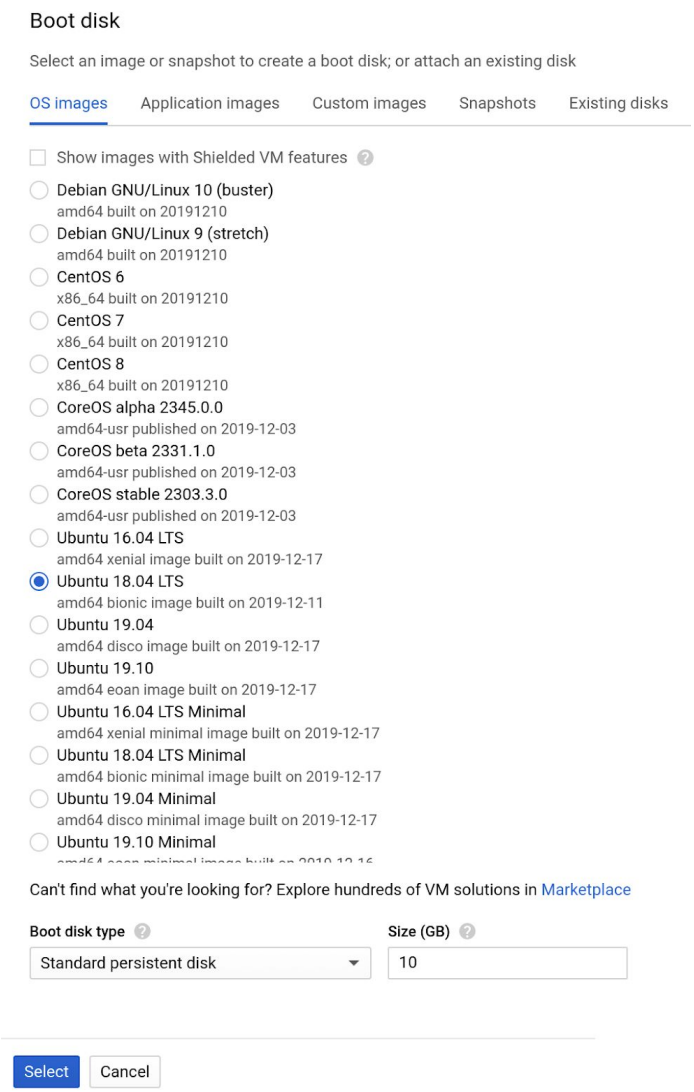
On the GCP Console select the **Navigation menu** from the top left corner.
Scroll down to the **COMPUTE** section, select **Compute Engine**, then select **VM instances** from the sub-menu.



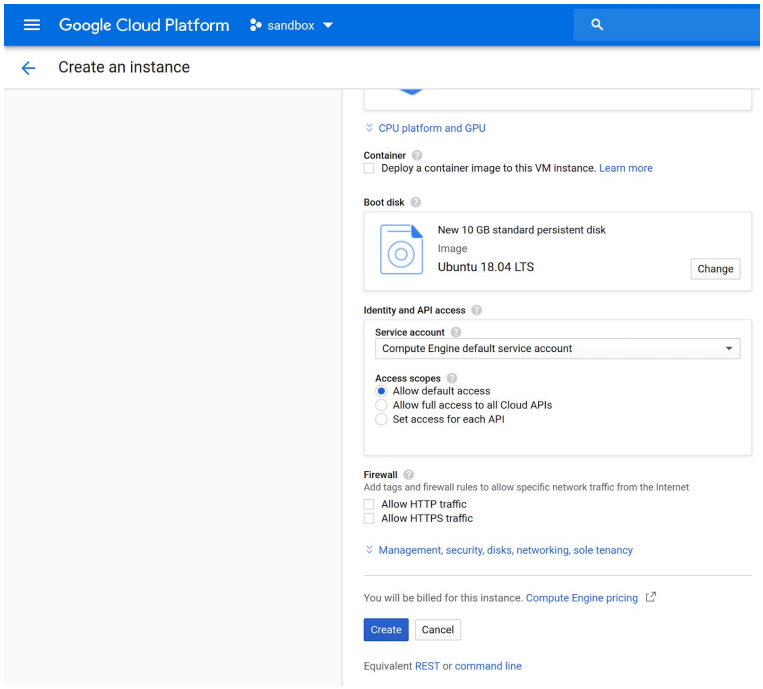
On the VM instances screen select **CREATE INSTANCE**.
Provide a name.
Select from the Region dropdown list the region closest to your location.
Select **General-purpose** Machine family.
Select from the Machine type dropdown list **g1-small (1 vCPU, 1.7 GB memory)**.
Select to **Change** the Boot disk to modify the instance OS image.



From the Boot disk OS images list select **Ubuntu 18.04 LTS**. Click **Select**.

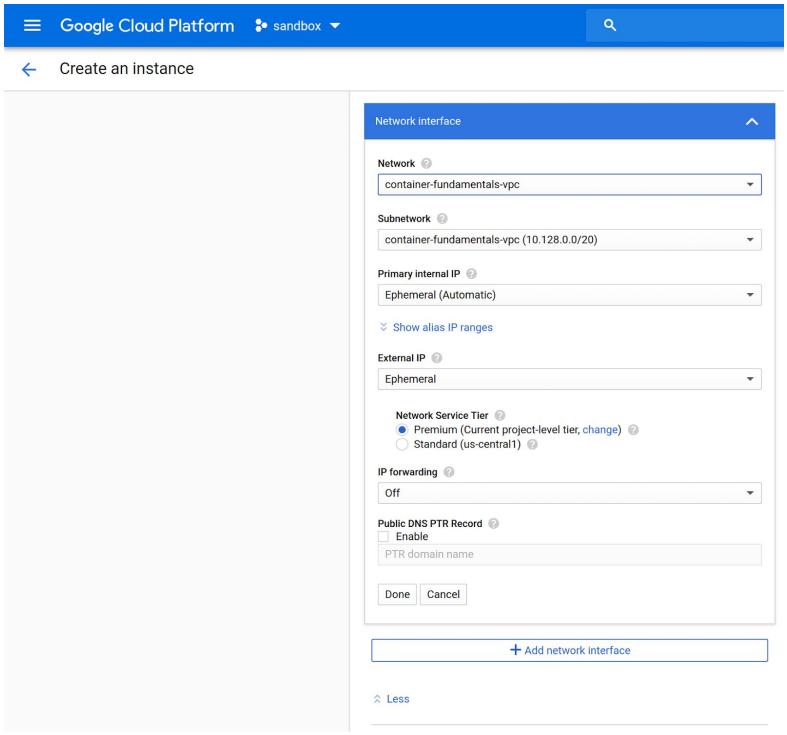


Expand the link **Management, security, disks, networking, sole tenancy**.
Select the **Networking** tab.

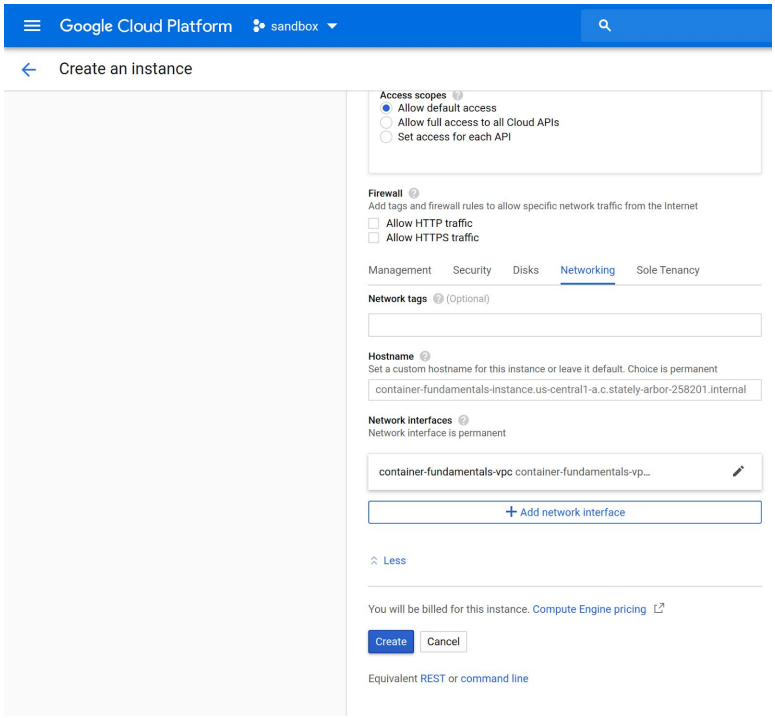


Select to edit the **Network interfaces** field.

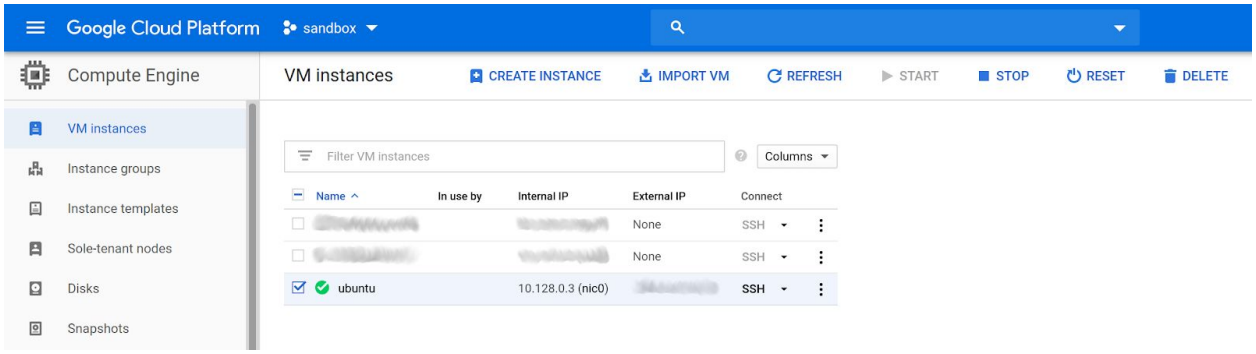
From the Network dropdown list select VPC network created earlier.
Select **Done**.



Select **Create** to finalize the creation of the VM instance.



From the VM instances page select the newly created instance followed by the desired action from the top menu: Start, Stop, Reset, Delete.
Running VM instance can be accessed directly by selecting **SSH**.



Lab 2.1 - Cgroups

Cgroups allow users to bundle processes together and limit, account and isolate the group's resources, such as CPU, memory, disk I/O, network, devices, and hugepages.

First, as **root**, let's install a tool that allows us to interact with cgroups:

```
student@ubuntu:~$ sudo -i
root@ubuntu:~# apt install -y cgroup-tools
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer
required:
  grub-pc-bin libnuma1
Use 'apt autoremove' to remove them.
The following additional packages will be installed:
  libcgroup1
The following NEW packages will be installed:
  cgroup-tools libcgroup1
0 upgraded, 2 newly installed, 0 to remove and 33 not upgraded.
Need to get 108 kB of archives.
After this operation, 393 kB of additional disk space will be used.
Get:1 http://us-central1.gce.archive.ubuntu.com/ubuntu bionic/universe amd64
libcgroup1 amd64 0.41-8ubuntu2 [42.0 kB]
Get:2 http://us-central1.gce.archive.ubuntu.com/ubuntu bionic/universe amd64
cgroup-tools amd64 0.41-8ubuntu2 [66.2 kB]
Fetched 108 kB in 0s (699 kB/s)
Selecting previously unselected package libcgroup1:amd64.
(Reading database ... 90813 files and directories currently installed.)
Preparing to unpack .../libcgroup1_0.41-8ubuntu2_amd64.deb ...
Unpacking libcgroup1:amd64 (0.41-8ubuntu2) ...
Selecting previously unselected package cgroup-tools.
Preparing to unpack .../cgroup-tools_0.41-8ubuntu2_amd64.deb ...
Unpacking cgroup-tools (0.41-8ubuntu2) ...
```

```
Setting up libcgroup1:amd64 (0.41-8ubuntu2) ...
Setting up cgroup-tools (0.41-8ubuntu2) ...
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
Processing triggers for libc-bin (2.27-3ubuntu1) ...
```

Now we can list all cgroups on our system:

```
root@ubuntu:~# lscgroup
rdma:/
net_cls,net_prio:/
hugetlb:/
memory:/
memory:/user.slice
memory:/system.slice
memory:/system.slice/systemd-update-utmp.service
memory:/system.slice/snap-core-8268.mount
memory:/system.slice/lvm2-monitor.service
...
```

And also list cgroups associated with a process:

```
root@ubuntu:~# cat /proc/<PID>/cgroup

root@ubuntu:~# cat /proc/1/cgroup
12:freezer:/
11:blkio:/
10:devices:/
9:perf_event:/
8:cpuset:/
7:pids:/
6:cpu,cpuacct:/
5:memory:/
4:hugetlb:/
3:net_cls,net_prio:/
2:rdma:/
1:name=systemd:/init.scope
0::/init.scope
```

Let's explore a particular cgroup, called **freezer**, which allows a group of tasks to be suspended and then resumed. We will demonstrate that a **frozen** (suspended) process does not allow any operations on it until it is **thawed** (resumed). Let's start by creating a new cgroup hierarchy under the freezer cgroup:

```
root@ubuntu:~# cd /sys/fs/cgroup/freezer/
root@ubuntu:/sys/fs/cgroup/freezer# mkdir mycgroup
root@ubuntu:/sys/fs/cgroup/freezer# cd mycgroup/
```

```
root@ubuntu:/sys/fs/cgroup/freezer/mycgroup# ls
cgroup.clone_children  freezer.parent_freezing   freezer.state  tasks
cgroup.procs           freezer.self_freezing     notify_on_release
```

The new directory is populated by default upon its creation. The **tasks** file, initially empty, would otherwise hold PIDs of processes associated with the cgroup. Let's verify the empty file, then create a new process and associate it with our cgroup:

```
root@ubuntu:/sys/fs/cgroup/freezer/mycgroup# cat tasks
root@ubuntu:/sys/fs/cgroup/freezer/mycgroup#
```

Let's open a second terminal as a new **bash** process, and in the new terminal list its PID:

```
root@ubuntu:~# ps
  PID TTY          TIME CMD
20912 pts/1    00:00:00 sudo
20913 pts/1    00:00:00 bash
20943 pts/1    00:00:00 ps
```

Keep the second terminal running, and return to the first terminal to add the PID of the second terminal to the **tasks** file of the cgroup:

```
root@ubuntu:/sys/fs/cgroup/freezer/mycgroup# echo 20913 >> tasks
root@ubuntu:/sys/fs/cgroup/freezer/mycgroup# cat tasks
20913
```

Now we should be able to freeze the processes associated with our cgroup:

```
root@ubuntu:/sys/fs/cgroup/freezer/mycgroup# echo FROZEN > freezer.state
root@ubuntu:/sys/fs/cgroup/freezer/mycgroup# cat freezer.state
FROZEN
```

The state we just modified affects all the processes listed in the tasks file, that is the second terminal window. Return to the second terminal and try running the **date** command, for example. Nothing will be registered and displayed, as the process of the terminal is in a **frozen** (suspended) state.

Finally, let's **thaw** (resume) the second terminal process:

```
root@ubuntu:/sys/fs/cgroup/freezer/mycgroup# echo THAWED > freezer.state
root@ubuntu:/sys/fs/cgroup/freezer/mycgroup# cat freezer.state
THAWED
```

Once thawed (resumed), the second terminal will display the **date** command we ran earlier, while it was in a frozen state:

```
root@ubuntu:~# date
Sat Sep  7 10:39:01 UTC 2019
```

Lab 2.2 - Namespaces

Namespaces allow users to isolate mount points, PIDs, networks, IPCs, UTS (hostname) and user IDs. In this exercise, we will focus on the network virtualization feature of namespaces.

Let's list all namespaces for a particular process:

```
root@ubuntu:~# ls -l /proc/<PID>/ns/

root@ubuntu:~# ls -l /proc/1/ns/
total 0
lrwxrwxrwx 1 root root 0 Sep  7 10:53 cgroup -> 'cgroup:[4026531835]'
lrwxrwxrwx 1 root root 0 Sep  7 10:53 ipc -> 'ipc:[4026531839]'
lrwxrwxrwx 1 root root 0 Sep  7 10:53 mnt -> 'mnt:[4026531840]'
lrwxrwxrwx 1 root root 0 Sep  7 10:53 net -> 'net:[4026531992]'
lrwxrwxrwx 1 root root 0 Sep  7 10:53 pid -> 'pid:[4026531836]'
lrwxrwxrwx 1 root root 0 Sep  7 10:53 pid_for_children -> 'pid:[4026531836]'
lrwxrwxrwx 1 root root 0 Sep  7 10:53 user -> 'user:[4026531837]'
lrwxrwxrwx 1 root root 0 Sep  7 10:53 uts -> 'uts:[4026531838]'
```

Let's explore the network namespaces, by creating two separate namespaces enabled to communicate with each other through a virtual ethernet tunnel.

First, create two namespaces:

```
root@ubuntu:~# ip netns add namespace1
root@ubuntu:~# ip netns add namespace2
root@ubuntu:~# ip netns list
namespace2 (id: 1)
namespace1 (id: 0)
```

Then create a pair of interconnected virtual ethernet devices:

```
root@ubuntu:~# ip link add veth1 type veth peer name veth2
```

Link each device to a namespace respectively:

```
root@ubuntu:~# ip link set veth1 netns namespace1
root@ubuntu:~# ip link set veth2 netns namespace2
```

Bring up the devices while assigning them IP addresses:

```
root@ubuntu:~# ip netns exec namespace1 ip link set dev veth1 up
root@ubuntu:~# ip netns exec namespace2 ip link set dev veth2 up

root@ubuntu:~# ip netns exec namespace1 ifconfig veth1 192.168.1.1 up
root@ubuntu:~# ip netns exec namespace2 ifconfig veth2 192.168.1.2 up
```

And now verify the connectivity between the two namespaces as it is enabled by the veth pair tunnel. From the first namespace we should be able to ping the second one, and from the second namespace we should be able to ping the first one:

```
root@ubuntu:~# ip netns exec namespace1 ping 192.168.1.2
PING 192.168.1.2 (192.168.1.2) 56(84) bytes of data.
64 bytes from 192.168.1.2: icmp_seq=1 ttl=64 time=0.044 ms
64 bytes from 192.168.1.2: icmp_seq=2 ttl=64 time=0.038 ms
64 bytes from 192.168.1.2: icmp_seq=3 ttl=64 time=0.037 ms
64 bytes from 192.168.1.2: icmp_seq=4 ttl=64 time=0.039 ms
64 bytes from 192.168.1.2: icmp_seq=5 ttl=64 time=0.041 ms
^C
--- 192.168.1.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4089ms
rtt min/avg/max/mdev = 0.037/0.039/0.044/0.008 ms

root@ubuntu:~# ip netns exec namespace2 ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_seq=1 ttl=64 time=0.026 ms
64 bytes from 192.168.1.1: icmp_seq=2 ttl=64 time=0.046 ms
64 bytes from 192.168.1.1: icmp_seq=3 ttl=64 time=0.045 ms
64 bytes from 192.168.1.1: icmp_seq=4 ttl=64 time=0.045 ms
64 bytes from 192.168.1.1: icmp_seq=5 ttl=64 time=0.042 ms
^C
--- 192.168.1.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4073ms
rtt min/avg/max/mdev = 0.026/0.040/0.046/0.011 ms
```

After a successful verification, we may delete the two namespaces:

```
root@ubuntu:~# ip netns delete namespace1
root@ubuntu:~# ip netns delete namespace2
root@ubuntu:~# ip netns list
```

Lab 2.3 - UnionFS

UnionFS transparently overlays files and directories of separate filesystems, to create a unified seamless filesystem. Each participant directory is referred to as a branch and we may set priorities and access modes while mounting branches.

Let's first install the required tool:

```
root@ubuntu:~# apt install -y unionfs-fuse
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer
required:
  grub-pc-bin libnuma1
Use 'apt autoremove' to remove them.
The following NEW packages will be installed:
  unionfs-fuse
0 upgraded, 1 newly installed, 0 to remove and 33 not upgraded.
Need to get 48.7 kB of archives.
After this operation, 146 kB of additional disk space will be used.
Get:1 http://us-central1.gce.archive.ubuntu.com/ubuntu bionic/universe amd64
unionfs-fuse amd64 1.0-1ubuntu2 [48.7 kB]
Fetched 48.7 kB in 0s (302 kB/s)
Selecting previously unselected package unionfs-fuse.
(Reading database ... 90862 files and directories currently installed.)
Preparing to unpack .../unionfs-fuse_1.0-1ubuntu2_amd64.deb ...
Unpacking unionfs-fuse (1.0-1ubuntu2) ...
Setting up unionfs-fuse (1.0-1ubuntu2) ...
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
```

Let's create two separate directories (branches) with two files each respectively:

```
root@ubuntu:~# mkdir /root/dir1
```

```
root@ubuntu:~# touch /root/dir1/f1
root@ubuntu:~# touch /root/dir1/f2
root@ubuntu:~# mkdir /root/dir2
root@ubuntu:~# touch /root/dir2/f3
root@ubuntu:~# touch /root/dir2/f4
```

Let's create an empty directory, where the union filesystem will be mounted:

```
root@ubuntu:~# mkdir /root/union
```

Let's mount the two branches and verify the transparent overlay by listing all the files in the union:

```
root@ubuntu:~# unionfs /root/dir1/:/root/dir2/ /root/union/
root@ubuntu:~# ls /root/union/
f1  f2  f3  f4
```

Lab 3.1 - Chroot

In this exercise we will explore chroot's ability to change the apparent root directory for a process. The setup will require an installation of a guest system in a subdirectory of our Ubuntu host system. Although similar to the installation of a guest Operating System Virtual Machine, this scenario is missing the hypervisor component, allowing the guest OS to be installed directly on top of the host OS, while allowing the host kernel to manage the guest as well.

The tool required to install the Debian based guest on an existing running host is debootstrap. It installs a Debian based guest in a subdirectory of our Ubuntu system. Let's install debootstrap:

```
student@ubuntu:~$ sudo -i
root@ubuntu:~# apt install -y debootstrap
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer
required:
  grub-pc-bin libnuma1
Use 'apt autoremove' to remove them.
Suggested packages:
  ubuntu-archive-keyring
The following NEW packages will be installed:
  debootstrap
0 upgraded, 1 newly installed, 0 to remove and 33 not upgraded.
Need to get 35.6 kB of archives.
After this operation, 272 kB of additional disk space will be used.
Get:1 http://us-central1.gce.archive.ubuntu.com/ubuntu bionic-updates/main
amd64 debootstrap all 1.0.95ubuntu0.5 [35.6 kB]
Fetched 35.6 kB in 0s (385 kB/s)
Selecting previously unselected package debootstrap.
(Reading database ... 90876 files and directories currently installed.)
Preparing to unpack .../debootstrap_1.0.95ubuntu0.5_all.deb ...
Unpacking debootstrap (1.0.95ubuntu0.5) ...
Setting up debootstrap (1.0.95ubuntu0.5) ...
```

```
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
```

Now let's setup a subdirectory on our host system for the guest:

```
root@ubuntu:~# mkdir /mnt/chroot-ubuntu-xenial
```

Install an Ubuntu Xenial guest in the subdirectory created earlier:

```
root@ubuntu:~# debootstrap xenial /mnt/chroot-ubuntu-xenial/
http://archive.ubuntu.com/ubuntu/
I: Retrieving InRelease
I: Checking Release signature
I: Valid Release signature (key id 790BC7277767219C42C86F933B4FE6ACC0B21F32)
I: Retrieving Packages
I: Validating Packages
I: Resolving dependencies of required packages...
I: Resolving dependencies of base packages...
I: Checking component main on http://archive.ubuntu.com/ubuntu...
I: Retrieving adduser 3.113+nmu3ubuntu4
I: Validating adduser 3.113+nmu3ubuntu4
I: Retrieving apt 1.2.10ubuntu1
I: Validating apt 1.2.10ubuntu1
I: Retrieving apt-utils 1.2.10ubuntu1
I: Validating apt-utils 1.2.10ubuntu1
I: Retrieving base-files 9.4ubuntu4
I: Validating base-files 9.4ubuntu4
...
```

Before verifying the guest installation with chroot, let's verify the version of our Ubuntu host OS:

```
root@ubuntu:~# cat /etc/os-release
NAME="Ubuntu"
VERSION="18.04.3 LTS (Bionic Beaver)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 18.04.3 LTS"
VERSION_ID="18.04"
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-po
licy"
VERSION_CODENAME=bionic
UBUNTU_CODENAME=bionic
```

Now, with chroot, let's open a shell into the newly installed guest OS environment. Every subsequent command will run inside the chrooted environment as the new root of /mnt/chroot-ubuntu-xenial:

```
root@ubuntu:~# chroot /mnt/chroot-ubuntu-xenial/ /bin/bash

root@ubuntu:/# cat /etc/os-release
NAME="Ubuntu"
VERSION="16.04 LTS (Xenial Xerus)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 16.04 LTS"
VERSION_ID="16.04"
HOME_URL="http://www.ubuntu.com/"
SUPPORT_URL="http://help.ubuntu.com/"
BUG_REPORT_URL="http://bugs.launchpad.net/ubuntu/"
UBUNTU_CODENAME=xenial
```

We can safely exit the chrooted environment:

```
root@ubuntu:/# exit
exit
root@ubuntu:~#
```

Lab 3.2 - LXC

LXC is an interface for Linux kernel OS level virtualization features. It allows the creation of linux containers, both unprivileged and privileged. Let's explore both aspects in this exercise.

Let's install it first:

```
student@ubuntu:~$ sudo apt install -y lxc
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer
required:
  grub-pc-bin libnuma1
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  bridge-utils libpam-cgfs lxc-utils
Suggested packages:
  ifupdown lxc-templates lxctl
The following NEW packages will be installed:
  bridge-utils libpam-cgfs lxc lxc-utils
0 upgraded, 4 newly installed, 0 to remove and 0 not upgraded.
Need to get 420 kB of archives.
After this operation, 1418 kB of additional disk space will be used.
Get:1 http://us-central1.gce.archive.ubuntu.com/ubuntu bionic/main amd64
bridge-utils amd64 1.5-15ubuntu1 [30.1 kB]
Get:2 http://us-central1.gce.archive.ubuntu.com/ubuntu bionic-updates/universe
amd64 libpam-cgfs amd64 3.0.3-0ubuntu1~18.04.1 [29.8 kB]
Get:3 http://us-central1.gce.archive.ubuntu.com/ubuntu bionic-updates/universe
amd64 lxc-utils amd64 3.0.3-0ubuntu1~18.04.1 [357 kB]
Get:4 http://us-central1.gce.archive.ubuntu.com/ubuntu bionic-updates/universe
amd64 lxc all 3.0.3-0ubuntu1~18.04.1 [2968 B]
Fetched 420 kB in 0s (1140 kB/s)
Selecting previously unselected package bridge-utils.
(Reading database ... 60131 files and directories currently installed.)
```

```
Preparing to unpack .../bridge-utils_1.5-15ubuntu1_amd64.deb ...
Unpacking bridge-utils (1.5-15ubuntu1) ...
Selecting previously unselected package libpam-cgfs.
Preparing to unpack .../libpam-cgfs_3.0.3-0ubuntu1~18.04.1_amd64.deb ...
Unpacking libpam-cgfs (3.0.3-0ubuntu1~18.04.1) ...
Selecting previously unselected package lxc-utils.
Preparing to unpack .../lxc-utils_3.0.3-0ubuntu1~18.04.1_amd64.deb ...
Unpacking lxc-utils (3.0.3-0ubuntu1~18.04.1) ...
Selecting previously unselected package lxc.
Preparing to unpack .../lxc_3.0.3-0ubuntu1~18.04.1_all.deb ...
Unpacking lxc (3.0.3-0ubuntu1~18.04.1) ...
Setting up bridge-utils (1.5-15ubuntu1) ...
Setting up libpam-cgfs (3.0.3-0ubuntu1~18.04.1) ...
Setting up lxc-utils (3.0.3-0ubuntu1~18.04.1) ...
Created symlink /etc/systemd/system/multi-user.target.wants/lxc-net.service →
/lib/systemd/system/lxc-net.service.
Created symlink /etc/systemd/system/multi-user.target.wants/lxc.service →
/lib/systemd/system/lxc.service.
Setting up lxc dnsmasq configuration.
Setting up lxc (3.0.3-0ubuntu1~18.04.1) ...
Processing triggers for ureadahead (0.100.0-21) ...
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
```

Create an unprivileged container as the **student** user

An unprivileged container is the safest container to deploy in any environment. The UID of the container's root is mapped to a UID with few or no privileges on the host system. While an unprivileged container can be created by **root** also, we will explore the lxc configuration and the container creation steps as the **student** user. Prior to creating an unprivileged container, we have to set the configuration for UID and GID mapping, and a network device quota that allows the unprivileged user to create network devices on the host - which otherwise is not allowed by default.

Let's display the UID and GID map defined for the student user on the host (the values associated with the student user will be used later to customize the configuration):

```
student@ubuntu:~$ cat /etc/subuid
lxd:100000:65536
root:100000:65536
ubuntu:165536:65536
student:231072:65536

student@ubuntu:~$ cat /etc/subgid
lxd:100000:65536
root:100000:65536
```

```
ubuntu:165536:65536
student:231072:65536
```

Let's add the student user to a configuration file which allows the user to create network devices on the host, to be then used by the linux containers:

```
student@ubuntu:~$ sudo bash -c 'echo student veth lxcbr0 10  >>
/etc/lxc/lxc-usernet'

student@ubuntu:~$ cat /etc/lxc/lxc-usernet
# USERNAME TYPE BRIDGE COUNT
student veth lxcbr0 10
```

Let's setup the configuration file for lxc. First verify that it is not already setup:

```
student@ubuntu:~$ ls -a ~/ | grep config
```

Then continue by creating the necessary directories, copy over the default configuration file, and display its default content:

```
student@ubuntu:~$ mkdir -p ~/.config/lxc

student@ubuntu:~$ ls -a ~/.config/
.  ..  lxc

student@ubuntu:~$ cp /etc/lxc/default.conf ~/.config/lxc/default.conf

student@ubuntu:~$ cat ~/.config/lxc/default.conf
lxc.net.0.type = veth
lxc.net.0.link = lxcbr0
lxc.net.0.flags = up
lxc.net.0.hwaddr = 00:16:3e:xx:xx:xx
```

Now let's customize the configuration file with the UID and GID maps of the student user, displayed earlier:

```
student@ubuntu:~$ echo lxc.idmap = u 0 231072 65536 >>
~/.config/lxc/default.conf

student@ubuntu:~$ echo lxc.idmap = g 0 231072 65536 >>
~/.config/lxc/default.conf

student@ubuntu:~$ cat ~/.config/lxc/default.conf
lxc.net.0.type = veth
```

```
lxc.net.0.link = lxcbr0
lxc.net.0.flags = up
lxc.net.0.hwaddr = 00:16:3e:xx:xx:xx
lxc.idmap = u 0 231072 65536
lxc.idmap = g 0 231072 65536
```

Reboot your host, or log out and then log back in:

```
student@ubuntu:~$ sudo reboot
```

At this point, we are ready to create an unprivileged container. We will use the **download** template which will present us a list of all available images designed to work without privileges. Once the image index is displayed, the tool will expect three separate entries from the user at the CLI: **distribution**, **release** and **architecture**. For this example **ubuntu**, **xenial** and **amd64** have been entered respectively at the prompts:

```
student@ubuntu:~$ lxc-create -t download -n unpriv-cont-user
Setting up the GPG keyring
Downloading the image index

---
DIST  RELEASE     ARCH  VARIANT     BUILD
---
alpine3.10  amd64 default     20200308_13:00
alpine3.10  arm64 default     20200308_13:00
alpine3.10  armhf default     20200308_13:00
alpine3.10  i386  default     20200308_13:00
alpine3.10  ppc64el     default     20200308_13:00
alpine3.10  s390x default     20200308_13:00
alpine3.11  amd64 default     20200308_13:00
...
ubuntutrustyamd64 default     20200308_07:42
ubuntutrustyarm64 default     20200308_07:42
ubuntutrustyarmhf default     20200308_07:44
ubuntutrustyi386  default     20200308_07:42
ubuntutrustyppc64el     default     20200308_07:42
ubuntuxenialamd64 default     20200308_07:42
ubuntuxenialarm64 default     20200308_07:42
ubuntuxenialarmhf default     20200308_07:46
ubuntuxenialppc64el     default     20200308_07:42
ubuntuxenials390x default     20200308_07:42
voidlinux   current     amd64 default     20200308_17:10
voidlinux   current     arm64 default     20200308_17:10
voidlinux   current     armhf default     20200308_17:10
voidlinux   current     i386  default     20200308_17:10
```

```
---

Distribution:
ubuntu
Release:
xenial
Architecture:
amd64

Downloading the image index
Downloading the rootfs
Downloading the metadata
The image cache is now ready
Unpacking the rootfs

---
You just created an Ubuntu xenial amd64 (20200308_07:42) container.

To enable SSH, run: apt install openssh-server
No default root or user password are set by LXC.
```

With the container created, now we can start it:

```
student@ubuntu:~$ lxc-start -n unpriv-cont-user -d
```

We can list containers, and also display individual container details:

```
student@ubuntu:~$ lxc-ls -f
NAME             STATE   AUTOSTART GROUPS IPV4       IPV6 UNPRIVILEGED
unpriv-cont-user RUNNING 0         -      10.0.3.102 -    true

student@ubuntu:~$ lxc-info -n unpriv-cont-user
Name:           unpriv-cont-user
State:          RUNNING
PID:            1726
IP:             10.0.3.102
Memory use:     16.89 MiB
KMem use:       4.62 MiB
Link:           vethNJDBNT
 TX bytes:      1.89 KiB
 RX bytes:      2.84 KiB
 Total bytes:   4.73 KiB
```

We can log into the running container and interact with its environment. Let's display the container hostname, list of processes, and OS release, then exit the container:

```
student@ubuntu:~$ lxc-attach -n unpriv-cont-user

root@unpriv-cont-user:/# hostname
unpriv-cont-user

root@unpriv-cont-user:/# ps
  PID TTY          TIME CMD
  176 pts/2    00:00:00 agetty
  350 pts/2    00:00:00 bash
  354 pts/2    00:00:00 ps

root@unpriv-cont-user:/# cat /etc/os-release
NAME="Ubuntu"
VERSION="16.04.6 LTS (Xenial Xerus)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 16.04.6 LTS"
VERSION_ID="16.04"
HOME_URL="http://www.ubuntu.com/"
SUPPORT_URL="http://help.ubuntu.com/"
BUG_REPORT_URL="http://bugs.launchpad.net/ubuntu/"
VERSION_CODENAME=xenial
UBUNTU_CODENAME=xenial

root@unpriv-cont-user:/# exit
exit
student@ubuntu:~$
```

Stopping and removing a container is quite simple:

```
student@ubuntu:~$ lxc-stop -n unpriv-cont-user

student@ubuntu:~$ lxc-destroy -n unpriv-cont-user
lxc-destroy: unpriv-cont-user: tools/lxc_destroy.c: main: 271 Destroyed
container unpriv-cont-user
```

Create a privileged container (as **root**)

In the situation when a privileged container has to be created, the process is similar to an unprivileged container, with the only distinction that it is created by the root of the host. Again, we will be required to provide at the CLI prompt the desired distribution, release and architecture from the available image index:

```
root@ubuntu:~# lxc-create -t download -n priv-cont
Setting up the GPG keyring
Downloading the image index

---
DIST  RELEASE     ARCH  VARIANT     BUILD
---
alpine3.10  amd64 default     20200308_13:00
alpine3.10  arm64 default     20200308_13:00
alpine3.10  armhf default     20200308_13:00
alpine3.10  i386  default     20200308_13:00
...
ubuntuxenialamd64 default     20200308_07:42
ubuntuxenialarm64 default     20200308_07:42
ubuntuxenialarmhf default     20200308_07:46
ubuntuxenialppc64el     default     20200308_07:42
ubuntuxenials390x default     20200308_07:42
voidlinux   current     amd64 default     20200308_17:10
voidlinux   current     arm64 default     20200308_17:10
voidlinux   current     armhf default     20200308_17:10
voidlinux   current     i386  default     20200308_17:10
---

Distribution:
ubuntu
Release:
xenial
Architecture:
amd64

Downloading the image index
Downloading the rootfs
Downloading the metadata
The image cache is now ready
Unpacking the rootfs

---
You just created an Ubuntu xenial amd64 (20200308_07:42) container.

To enable SSH, run: apt install openssh-server
No default root or user password are set by LXC.
```

With the container created, now we can start it:

```
root@ubuntu:~# lxc-start -n priv-cont -d
```

We can list containers, and also display individual container details:

```
root@ubuntu:~# lxc-ls -f
NAME      STATE   AUTOSTART GROUPS IPV4 IPV6 UNPRIVILEGED
priv-cont RUNNING 0         -      -    -    false

root@ubuntu:~# lxc-info -n priv-cont
Name:           priv-cont
State:          RUNNING
PID:            5660
IP:             10.0.3.49
CPU use:        0.31 seconds
BlkIO use:      14.24 MiB
Memory use:     29.35 MiB
KMem use:       2.99 MiB
Link:           vethMJRD1G
 TX bytes:      1.35 KiB
 RX bytes:      1.54 KiB
 Total bytes:   2.89 KiB
```

We can log into the running container and interact with its environment. Let's display the container hostname, list of processes, and OS release, then exit the container:

```
root@ubuntu:~# lxc-attach -n priv-cont

root@priv-cont:~# hostname
priv-cont

root@priv-cont:~# ps
  PID TTY          TIME CMD
  165 pts/3    00:00:00 agetty
  175 pts/3    00:00:00 bash
  188 pts/3    00:00:00 ps

root@priv-cont:~# cat /etc/os-release
NAME="Ubuntu"
VERSION="16.04.6 LTS (Xenial Xerus)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 16.04.6 LTS"
VERSION_ID="16.04"
HOME_URL="http://www.ubuntu.com/"
SUPPORT_URL="http://help.ubuntu.com/"
BUG_REPORT_URL="http://bugs.launchpad.net/ubuntu/"
VERSION_CODENAME=xenial
```

```
UBUNTU_CODENAME=xenial
root@priv-cont:~#
root@priv-cont:~# exit
exit
root@ubuntu:~#
```

Stopping and removing a container is quite simple:

```
root@ubuntu:~# lxc-stop -n priv-cont

root@ubuntu:~# lxc-destroy -n priv-cont
lxc-destroy: priv-cont: tools/lxc_destroy.c: main: 271 Destroyed container
priv-cont
```

Lab 3.3 - Systemd-nspawn

Another method of creating an isolated virtual environment where an application or an entire Operating System could run is by creating a systemd-container. Subsequently, this container can be managed with systemd tools.

Let's install the container tool:

```
root@ubuntu:~# apt install -y systemd-container
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer
required:
  grub-pc-bin libnuma1
Use 'apt autoremove' to remove them.
The following additional packages will be installed:
  libnss-mymachines libnss-systemd libpam-systemd libsystemd0 systemd
The following NEW packages will be installed:
  libnss-mymachines systemd-container
The following packages will be upgraded:
  libnss-systemd libpam-systemd libsystemd0 systemd
...
```

Let's bootstrap a Debian base system in a target directory of the host:

```
root@ubuntu:~# debootstrap --arch=amd64 stable ~/DebianContainer
I: Keyring file not available at
/usr/share/keyrings/debian-archive-keyring.gpg; switching to https mirror
https://deb.debian.org/debian
I: Retrieving InRelease
I: Retrieving Packages
I: Validating Packages
I: Resolving dependencies of required packages...
```

```
I: Resolving dependencies of base packages...
I: Found additional required dependencies: adduser debian-archive-keyring
fdisk gcc-8-base gpgv libacl1 libapt-pkg5.0 libattr1 libaudit-common libaudit1
libblkid1 libbz2-1.0 libc6 libcap-ng0 libcom-err2 libdb5.3 libdebconfclient0
libext2fs2 libfdisk1 libffi6 libgcc1 libgcrypt20 libgmp10 libgnutls30
libgpg-error0 libhogweed4 libidn2-0 liblz4-1 liblzma5 libmount1 libncursesw6
libnettle6 libp11-kit0 libpam0g libpcre3 libseccomp2 libselinux1
libsemanage-common libsemanage1 libsepol1 libsmartcols1 libss2 libstdc++6
libsystemd0 libtasn1-6 libtinfo6 libudev1 libunistring2 libuuid1 libzstd1
zlib1g
I: Found additional base dependencies: dmsetup libapparmor1 libapt-inst2.0
libargon2-1 libbsd0 libcap2 libcap2-bin libcryptsetup12 libdevmapper1.02.1
libdns-export1104 libelf1 libestr0 libfastjson4 libidn11 libip4tc0 libip6tc0
libiptc0 libisc-export1100 libjson-c3 libkmod2 liblocale-gettext-perl
liblognorm5 libmnl0 libncurses6 libnetfilter-conntrack3 libnewt0.52
libnfnetlink0 libnftnl11 libpopt0 libprocps7 libslang2 libssl1.1
libtext-charwidth-perl libtext-iconv-perl libtext-wrapi18n-perl libxtables12
lsb-base openssl xxd
...
```

Now we can safely create a container from the directory where the Debian base system is running:

```
root@ubuntu:~# systemd-nspawn -bD ~/DebianContainer
Spawning container DebianContainer on /root/DebianContainer.
Press ^] three times within 1s to kill container.
Host and machine ids are equal (762b9a54c8bc38ec7c693a05acbef61c): refusing to
link journals
systemd 241 running in system mode. (+PAM +AUDIT +SELINUX +IMA +APPARMOR
+SMACK +SYSVINIT +UTMP +LIBCRYPTSETUP +GCRYPT +GNUTLS +ACL +XZ +LZ4 +SECCOMP
+BLKID +ELFUTILS +KMOD -IDN2 +IDN -PCRE2 default-hierarchy=hybrid)
Detected virtualization systemd-nspawn.
Detected architecture x86-64.

Welcome to Debian GNU/Linux 10 (buster)!

Set hostname to <ubuntu>.
File /lib/systemd/system/systemd-journald.service:12 configures an IP firewall
(IPAddressDeny=any), but the local system does not support BPF/cgroup based
firewalling.
Proceeding WITHOUT firewalling in effect! (This warning is only shown for the
first loaded unit using IP firewalling.)
[  OK  ] Reached target Slices.
[  OK  ] Listening on Journal Socket (/dev/log).
[  OK  ] Reached target Swap.
...
```

We can safely disregard the login prompt once the above command succeeds. Leave the current terminal open and open a second terminal on the same host VM. From the second terminal let's list containers and display container details:

```
root@ubuntu:~# machinectl list
MACHINE          CLASS      SERVICE         OS      VERSION  ADDRESSES
DebianContainer  container  systemd-nspawn  debian  10       -

1 machines listed.

root@ubuntu:~# machinectl status DebianContainer
DebianContainer(762b9a54c8bc38ec7c693a05acbef61c)
           Since: Sun 2020-03-08 09:07:11 UTC; 9min ago
          Leader: 15873 (systemd)
         Service: systemd-nspawn; class container
            Root: /root/DebianContainer
              OS: Debian GNU/Linux 10 (buster)
            Unit: machine-DebianContainer.scope
                  ├─init.scope
                  │ └─15873 /lib/systemd/systemd
                  └─system.slice
                    ├─console-getty.service
                    │ └─15946 /sbin/agetty -o -p -- \u --noclear --keep-ba...
                    ├─cron.service
                    │ └─15936 /usr/sbin/cron -f
                    ├─rsyslog.service
                    │ └─15937 /usr/sbin/rsyslogd -n -iNONE
                    └─systemd-journald.service
                      └─15912 /lib/systemd/systemd-journald

root@ubuntu:~# machinectl show DebianContainer
Name=DebianContainer
Id=762b9a54c8bc38ec7c693a05acbef61c
Timestamp=Sun 2020-03-08 09:07:11 UTC
TimestampMonotonic=84850036605
Service=systemd-nspawn
Unit=machine-DebianContainer.scope
Leader=15873
Class=container
RootDirectory=/root/DebianContainer
State=running
```

Now it is safe to terminate the container:

```
root@ubuntu:~# machinectl terminate DebianContainer

root@ubuntu:~# machinectl list
```

No machines.

Lab 4.1 - Install runc

There are various methods of installing the runc CLI tool. For Ubuntu it is available to install directly from the Ubuntu software package repository. So let's proceed with the installation of runc on Ubuntu. First, update the system and run all required upgrades:

```
student@ubuntu:~$ sudo apt update
Hit:1 http://us-central1.gce.archive.ubuntu.com/ubuntu bionic InRelease
Get:2 http://us-central1.gce.archive.ubuntu.com/ubuntu bionic-updates
InRelease [88.7 kB]
Get:3 http://us-central1.gce.archive.ubuntu.com/ubuntu bionic-backports
InRelease [74.6 kB]
Hit:4 http://archive.canonical.com/ubuntu bionic InRelease
Get:5 http://security.ubuntu.com/ubuntu bionic-security InRelease [88.7 kB]
Fetched 252 kB in 1s (426 kB/s)
Reading package lists... Done
Building dependency tree
...

student@ubuntu:~$ sudo apt upgrade -y
Reading package lists... Done
Building dependency tree
Reading state information... Done
Calculating upgrade... Done
...
```

Now it is safe to install the runc package:

```
student@ubuntu:~$ sudo apt install -y runc
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer
required:
```

```
  grub-pc-bin libnuma1
Use 'sudo apt autoremove' to remove them.
The following NEW packages will be installed:
  runc
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 1903 kB of archives.
After this operation, 8638 kB of additional disk space will be used.
Get:1 http://us-central1.gce.archive.ubuntu.com/ubuntu bionic-updates/universe
amd64 runc amd64 1.0.0~rc7+git20190403.029124da-0ubuntu1~18.04.2 [1903 kB]
Fetched 1903 kB in 0s (33.9 MB/s)
Selecting previously unselected package runc.
(Reading database ... 60322 files and directories currently installed.)
Preparing to unpack
.../runc_1.0.0~rc7+git20190403.029124da-0ubuntu1~18.04.2_amd64.deb ...
Unpacking runc (1.0.0~rc7+git20190403.029124da-0ubuntu1~18.04.2) ...
Setting up runc (1.0.0~rc7+git20190403.029124da-0ubuntu1~18.04.2) ...
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
```

Verify the version of the installed runc package:

```
student@ubuntu:~$ runc --version
runc version spec: 1.0.1-dev
```

Lab 4.2 - Install containerd

Although containerd is not intended to be used directly by users, we will still explore the installation method of the containerd package on Ubuntu.

First, update the system and run all required upgrades:

```
student@ubuntu:~$ sudo apt update
Hit:1 http://us-central1.gce.archive.ubuntu.com/ubuntu bionic InRelease
Get:2 http://us-central1.gce.archive.ubuntu.com/ubuntu bionic-updates
InRelease [88.7 kB]
Get:3 http://us-central1.gce.archive.ubuntu.com/ubuntu bionic-backports
InRelease [74.6 kB]
Hit:4 http://archive.canonical.com/ubuntu bionic InRelease
Get:5 http://security.ubuntu.com/ubuntu bionic-security InRelease [88.7 kB]
Fetched 252 kB in 1s (426 kB/s)
Reading package lists... Done
Building dependency tree
...

student@ubuntu:~$ sudo apt upgrade -y
Reading package lists... Done
Building dependency tree
Reading state information... Done
Calculating upgrade... Done
...
```

Now it is safe to install the containerd package:

```
student@ubuntu:~$ sudo apt install -y containerd
Reading package lists... Done
Building dependency tree
Reading state information... Done
```

```
The following packages were automatically installed and are no longer
required:
  grub-pc-bin libnuma1
Use 'sudo apt autoremove' to remove them.
The following NEW packages will be installed:
  containerd
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 21.7 MB of archives.
After this operation, 111 MB of additional disk space will be used.
Get:1 http://us-central1.gce.archive.ubuntu.com/ubuntu bionic-updates/universe
amd64 containerd amd64 1.3.3-0ubuntu1~18.04.1 [21.7 MB]
Fetched 21.7 MB in 1s (24.5 MB/s)
Selecting previously unselected package containerd.
(Reading database ... 60347 files and directories currently installed.)
Preparing to unpack .../containerd_1.3.3-0ubuntu1~18.04.1_amd64.deb ...
Unpacking containerd (1.3.3-0ubuntu1~18.04.1) ...
Setting up containerd (1.3.3-0ubuntu1~18.04.1) ...
Created symlink /etc/systemd/system/multi-user.target.wants/containerd.service
→ /lib/systemd/system/containerd.service.
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
```

Verify the version of the installed containerd package:

```
student@ubuntu:~$ containerd --version
containerd github.com/containerd/containerd 1.3.3-0ubuntu1~18.04.1
```

As containerd runs as a daemon on the Ubuntu host, we can manage it with systemd:

```
student@ubuntu:~$ sudo systemctl status containerd.service
● containerd.service - containerd container runtime
   Loaded: loaded (/lib/systemd/system/containerd.service; enabled; vendor
preset: enabled)
   Active: active (running) since Mon 2020-03-09 07:35:18 UTC; 13min ago
     Docs: https://containerd.io
  Process: 10624 ExecStartPre=/sbin/modprobe overlay (code=exited,
status=0/SUCCESS)
 Main PID: 10625 (containerd)
    Tasks: 9
   CGroup: /system.slice/containerd.service
           └─10625 /usr/bin/containerd

Mar 09 07:35:19 ubuntu containerd[10625]:
time="2020-03-09T07:35:19.274733216Z" level=error msg="Fai
Mar 09 07:35:19 ubuntu containerd[10625]:
time="2020-03-09T07:35:19.274935584Z" level=info msg="load
```

Lab 4.3 - Install Docker Engine

Docker Engine has many installation options. While for Mac and Windows there is the Docker Desktop, for Linux distributions there are repositories including software packages of the docker components. For Ubuntu, there are two installation options. First method installs the docker daemon with runc and containerd dependencies directly from the Ubuntu software package repository. The second method allows for the installation of the Docker Engine - Community edition from the official Docker repository.

The first installation method requires a single step:

```
student@ubuntu:~$ sudo apt install -y docker.io
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer
required:
  grub-pc-bin libnuma1
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  cgroupfs-mount pigz ubuntu-fan
Suggested packages:
  aufs-tools debootstrap docker-doc rinse zfs-fuse | zfsutils
The following NEW packages will be installed:
  cgroupfs-mount docker.io pigz ubuntu-fan
...
Processing triggers for systemd (237-3ubuntu10.39) ...
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
Processing triggers for ureadahead (0.100.0-21) ...
```

The second method requires several steps for system cleanup, repository setup, and for package installation. Let's remove all related previously installed packages - docker, containerd, runc:

```
student@ubuntu:~$ sudo apt remove docker docker-engine docker.io containerd
runc
```

```
Reading package lists... Done
Building dependency tree
Reading state information... Done
Package 'docker-engine' is not installed, so not removed
Package 'docker' is not installed, so not removed
The following packages were automatically installed and are no longer
required:
  cgroupfs-mount grub-pc-bin libnuma1 pigz ubuntu-fan
Use 'sudo apt autoremove' to remove them.
The following packages will be REMOVED:
  containerd docker.io runc
0 upgraded, 0 newly installed, 3 to remove and 0 not upgraded.
After this operation, 275 MB disk space will be freed.
Do you want to continue? [Y/n] Y
(Reading database ... 60618 files and directories currently installed.)
Removing docker.io (18.09.7-0ubuntu1~18.04.4) ...
'/usr/share/docker.io/contrib/nuke-graph-directory.sh' ->
'/var/lib/docker/nuke-graph-directory.sh'
Removing containerd (1.3.3-0ubuntu1~18.04.1) ...
Removing runc (1.0.0~rc7+git20190403.029124da-0ubuntu1~18.04.2) ...
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
```

Update packages:

```
student@ubuntu:~$ sudo apt update
Hit:1 http://us-central1.gce.archive.ubuntu.com/ubuntu bionic InRelease
Get:2 http://us-central1.gce.archive.ubuntu.com/ubuntu bionic-updates
InRelease [88.7 kB]
Get:3 http://us-central1.gce.archive.ubuntu.com/ubuntu bionic-backports
InRelease [74.6 kB]
Hit:4 http://archive.canonical.com/ubuntu bionic InRelease
Get:5 http://security.ubuntu.com/ubuntu bionic-security InRelease [88.7 kB]
Fetched 252 kB in 1s (413 kB/s)
Reading package lists... Done
Building dependency tree
Reading state information... Done
All packages are up to date.
```

Install packages required for HTTPS repository:

```
student@ubuntu:~$ sudo apt install -y apt-transport-https ca-certificates curl
gnupg-agent software-properties-common
Reading package lists... Done
Building dependency tree
Reading state information... Done
ca-certificates is already the newest version (20180409).
```

```
ca-certificates set to manually installed.
curl is already the newest version (7.58.0-2ubuntu3.8).
curl set to manually installed.
software-properties-common is already the newest version (0.96.24.32.12).
software-properties-common set to manually installed.
The following packages were automatically installed and are no longer
required:
  cgroupfs-mount grub-pc-bin libnuma1 pigz ubuntu-fan
Use 'sudo apt autoremove' to remove them.
The following NEW packages will be installed:
  apt-transport-https gnupg-agent
...
Unpacking gnupg-agent (2.2.4-1ubuntu1.2) ...
Setting up apt-transport-https (1.6.12) ...
Setting up gnupg-agent (2.2.4-1ubuntu1.2) ...
```

Add Docker GPG key:

```
student@ubuntu:~$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg |
sudo apt-key add -
OK
```

Verify key fingerprint:

```
student@ubuntu:~$ sudo apt-key fingerprint 0EBFCD88
pub   rsa4096 2017-02-22 [SCEA]
      9DC8 5822 9FC7 DD38 854A  E2D8 8D81 803C 0EBF CD88
uid           [ unknown] Docker Release (CE deb) <docker@docker.com>
sub   rsa4096 2017-02-22 [S]
```

Add the stable repository:

```
student@ubuntu:~$ sudo add-apt-repository "deb [arch=amd64]
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
Hit:1 http://us-central1.gce.archive.ubuntu.com/ubuntu bionic InRelease
Hit:2 http://us-central1.gce.archive.ubuntu.com/ubuntu bionic-updates
InRelease
Hit:3 http://us-central1.gce.archive.ubuntu.com/ubuntu bionic-backports
InRelease
Get:4 https://download.docker.com/linux/ubuntu bionic InRelease [64.4 kB]
Get:5 http://security.ubuntu.com/ubuntu bionic-security InRelease [88.7 kB]
Hit:6 http://archive.canonical.com/ubuntu bionic InRelease
Get:7 https://download.docker.com/linux/ubuntu bionic/stable amd64 Packages
[10.7 kB]
Fetched 164 kB in 1s (281 kB/s)
```

```
Reading package lists... Done
```

Update packages:

```
student@ubuntu:~$ sudo apt update
Hit:1 http://us-central1.gce.archive.ubuntu.com/ubuntu bionic InRelease
Hit:2 http://us-central1.gce.archive.ubuntu.com/ubuntu bionic-updates
InRelease
Hit:3 http://us-central1.gce.archive.ubuntu.com/ubuntu bionic-backports
InRelease
Hit:4 http://security.ubuntu.com/ubuntu bionic-security InRelease
Hit:5 https://download.docker.com/linux/ubuntu bionic InRelease
Hit:6 http://archive.canonical.com/ubuntu bionic InRelease
Reading package lists... Done
Building dependency tree
Reading state information... Done
All packages are up to date.
```

Install the latest version of the Docker Engine - Community edition

```
student@ubuntu:~$ sudo apt install -y docker-ce docker-ce-cli containerd.io
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer
required:
  grub-pc-bin libnuma1 ubuntu-fan
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  aufs-tools libltdl7
The following NEW packages will be installed:
  aufs-tools containerd.io docker-ce docker-ce-cli libltdl7
...
Setting up aufs-tools (1:4.9+20170918-1ubuntu1) ...
Setting up containerd.io (1.2.13-1) ...
Setting up libltdl7:amd64 (2.4.6-2) ...
Setting up docker-ce-cli (5:19.03.7~3-0~ubuntu-bionic) ...
Setting up docker-ce (5:19.03.7~3-0~ubuntu-bionic) ...
Installing new version of config file /etc/init.d/docker ...
Installing new version of config file /etc/init/docker.conf ...
Created symlink /etc/systemd/system/multi-user.target.wants/docker.service →
/lib/systemd/system/docker.service.
Processing triggers for libc-bin (2.27-3ubuntu1) ...
Processing triggers for systemd (237-3ubuntu10.39) ...
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
Processing triggers for ureadahead (0.100.0-21) ...
```

Display the version of the Docker Engine components:

```
student@ubuntu:~$ sudo docker version
Client: Docker Engine - Community
 Version:           19.03.7
 API version:       1.40
 Go version:        go1.12.17
 Git commit:        7141c199a2
 Built:             Wed Mar  4 01:22:36 2020
 OS/Arch:           linux/amd64
 Experimental:      false

Server: Docker Engine - Community
 Engine:
  Version:          19.03.7
  API version:      1.40 (minimum version 1.12)
  Go version:       go1.12.17
  Git commit:       7141c199a2
  Built:            Wed Mar  4 01:21:08 2020
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.2.13
  GitCommit:        7ad184331fa3e55e52b890ea95e65ba581ae3429
 runc:
  Version:          1.0.0-rc10
  GitCommit:        dc9208a3303feef5b3839f4323d9beb36df0a9dd
 docker-init:
  Version:          0.18.0
  GitCommit:        fec3683
```

Verify the installation by running a test image:

```
student@ubuntu:~$ sudo docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
1b930d010525: Pull complete
Digest:
sha256:fc6a51919cfeb2e6763f62b6d9e8815acbf7cd2e476ea353743570610737b752
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
```

```
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

We can manage the Docker daemon with systemd:

```
student@ubuntu:~$ sudo systemctl status docker.service
● docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset:
enabled)
   Active: active (running) since Mon 2020-03-09 10:10:02 UTC; 27min ago
     Docs: https://docs.docker.com
 Main PID: 16503 (dockerd)
    Tasks: 10
   CGroup: /system.slice/docker.service
           └─16503 /usr/bin/dockerd -H fd://
--containerd=/run/containerd/containerd.sock

Mar 09 10:10:01 ubuntu dockerd[16503]: time="2020-03-09T10:10:01.848543161Z"
Mar 09 10:10:01 ubuntu dockerd[16503]: time="2020-03-09T10:10:01.848663838Z"
Mar 09 10:10:01 ubuntu dockerd[16503]: time="2020-03-09T10:10:01.849145302Z"
...
```

Lab 4.4 - Install CRI-O

Although CRI-O is not intended to be used directly by users, we will still explore the installation method of the CRI-O package on Ubuntu.

First, update the system and run all required upgrades:

```
student@ubuntu:~$ sudo apt update
Hit:1 http://us-central1.gce.archive.ubuntu.com/ubuntu bionic InRelease
Get:2 http://us-central1.gce.archive.ubuntu.com/ubuntu bionic-updates
InRelease [88.7 kB]
Get:3 http://us-central1.gce.archive.ubuntu.com/ubuntu bionic-backports
InRelease [74.6 kB]
...
```

Install the package that allows software repository management:

```
student@ubuntu:~$ sudo apt install -y software-properties-common
Reading package lists... Done
Building dependency tree
Reading state information... Done
software-properties-common is already the newest version (0.96.24.32.12).
The following packages were automatically installed and are no longer
required:
  grub-pc-bin libnuma1 ubuntu-fan
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
```

Add new repository:

```
student@ubuntu:~$ sudo add-apt-repository ppa:projectatomic/ppa

 More info: https://launchpad.net/~projectatomic/+archive/ubuntu/ppa
```

```
Press [ENTER] to continue or Ctrl-c to cancel adding it.

Hit:1 http://us-central1.gce.archive.ubuntu.com/ubuntu bionic InRelease
Hit:2 http://us-central1.gce.archive.ubuntu.com/ubuntu bionic-updates
InRelease
Hit:3 http://us-central1.gce.archive.ubuntu.com/ubuntu bionic-backports
InRelease
Hit:4 https://download.docker.com/linux/ubuntu bionic InRelease
...
```

Update package index

```
student@ubuntu:~$ sudo apt update
Hit:1 http://us-central1.gce.archive.ubuntu.com/ubuntu bionic InRelease
Hit:2 http://us-central1.gce.archive.ubuntu.com/ubuntu bionic-updates
InRelease
Hit:3 http://us-central1.gce.archive.ubuntu.com/ubuntu bionic-backports
InRelease
...
```

Install CRI-O package:

```
student@ubuntu:~$ sudo apt install -y cri-o-1.15
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer
required:
  grub-pc-bin libnuma1 ubuntu-fan
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  conmon containers-common containers-golang containers-image cri-o-runc
libgpgme11
Suggested packages:
  containernetworking-plugins
The following NEW packages will be installed:
  conmon containers-common containers-golang containers-image cri-o-1.15
cri-o-runc libgpgme11
...
```

Display CRI-O version:

```
student@ubuntu:~$ crio --version
crio version 1.15.3-dev
commit: unknown
```

Lab 4.5 - Install rkt

Rkt runtime, an archived project as of the time of this writing, installs quite easily on Linux distributions from maintained software packages. For Ubuntu, a manual install of Debian packages is required.

Retrieve the gpg key:

```
student@ubuntu:~$ gpg --recv-key 18AD5014C99EF7E3BA5F6CE950BDD3E0FC8A365E
gpg: keybox '/home/student/.gnupg/pubring.kbx' created
gpg: key 50BDD3E0FC8A365E: 10 signatures not checked due to missing keys
gpg: /home/student/.gnupg/trustdb.gpg: trustdb created
gpg: key 50BDD3E0FC8A365E: public key "CoreOS Application Signing Key
<security@coreos.com>" imported
gpg: no ultimately trusted keys found
gpg: Total number processed: 1
gpg:               imported: 1
```

Download the package file together with its signature file:

```
student@ubuntu:~$ wget
https://github.com/rkt/rkt/releases/download/v1.30.0/rkt_1.30.0-1_amd64.deb
--2020-03-10 05:04:29--
https://github.com/rkt/rkt/releases/download/v1.30.0/rkt_1.30.0-1_amd64.deb
Resolving github.com (github.com)... 192.30.253.112
Connecting to github.com (github.com)|192.30.253.112|:443... connected.
HTTP request sent, awaiting response... 302 Found
...


student@ubuntu:~$ wget
https://github.com/rkt/rkt/releases/download/v1.30.0/rkt_1.30.0-1_amd64.deb.as
c
```

```
--2020-03-10 05:05:10--
https://github.com/rkt/rkt/releases/download/v1.30.0/rkt_1.30.0-1_amd64.deb.as
c
Resolving github.com (github.com)... 192.30.253.112
Connecting to github.com (github.com)|192.30.253.112|:443... connected.
HTTP request sent, awaiting response... 302 Found
...
```

Validate the signature and install the package:

```
student@ubuntu:~$ gpg --verify rkt_1.30.0-1_amd64.deb.asc
gpg: assuming signed data in 'rkt_1.30.0-1_amd64.deb'
gpg: Signature made Mon Apr 16 09:50:05 2018 UTC
gpg:                using RSA key 5B1053CE38EA2E0FEB956C0595BC5E3F3F1B2C87
gpg: Good signature from "CoreOS Application Signing Key
<security@coreos.com>" [unknown]
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: 18AD 5014 C99E F7E3 BA5F  6CE9 50BD D3E0 FC8A 365E
     Subkey fingerprint: 5B10 53CE 38EA 2E0F EB95  6C05 95BC 5E3F 3F1B 2C87

student@ubuntu:~$ sudo dpkg -i rkt_1.30.0-1_amd64.deb
Selecting previously unselected package rkt.
(Reading database ... 60731 files and directories currently installed.)
Preparing to unpack rkt_1.30.0-1_amd64.deb ...
Unpacking rkt (1.30.0-1) ...
Setting up rkt (1.30.0-1) ...
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
```

After the installation we can verify the version of the installed rkt package:

```
student@ubuntu:~$ rkt version
rkt Version: 1.30.0
appc Version: 0.8.11
Go Version: go1.8.3
Go OS/Arch: linux/amd64
Features: -TPM +SDJOURNAL
```

Lab 4.6 - Install LXD

We explored the LXC installation, an Operating System virtualization mechanism that uses the Linux system as a runtime for Linux Containers. LXD, another tool to manage Linux Containers, enhances the LXC experience by replacing the LXC set of tools with a single CLI tool based on a REST API that enables management of remote containers as well.

Let's start with a system update:

```
student@ubuntu:~$ sudo apt update
Hit:1 http://us-central1.gce.archive.ubuntu.com/ubuntu bionic InRelease
Get:2 http://us-central1.gce.archive.ubuntu.com/ubuntu bionic-updates
InRelease [88.7 kB]
Get:3 http://us-central1.gce.archive.ubuntu.com/ubuntu bionic-backports
InRelease [74.6 kB]
...
```

Let's install all libraries, tools and packages for LXD:

```
student@ubuntu:~$ sudo apt install -y acl autoconf dnsmasq-base git golang
libacl1-dev libcap-dev liblxc1 liblxc-dev libtool libudev-dev libuv1-dev make
pkg-config rsync
Reading package lists... Done
Building dependency tree
Reading state information... Done
acl is already the newest version (2.2.52-3build1).
acl set to manually installed.
dnsmasq-base is already the newest version (2.79-1).
dnsmasq-base set to manually installed.
git is already the newest version (1:2.17.1-1ubuntu0.5).
git set to manually installed.
liblxc1 is already the newest version (3.0.3-0ubuntu1~18.04.1).
liblxc1 set to manually installed.
rsync is already the newest version (3.1.2-2.1ubuntu1.1).
```

```
rsync set to manually installed.
The following packages were automatically installed and are no longer
required:
  grub-pc-bin libnuma1 ubuntu-fan
...
```

Once installed, we can verify the version of the LXD package:

```
student@ubuntu:~$ lxd version
3.0.3
```

Lab 4.7 - Install Podman

The Podman runtime for OCI and Docker containers, runs both privileged and unprivileged containers, wrapped in a container pod. The container pod is extensively used by Kubernetes and rkt. Podman is typically used in conjunction with Buildah, a tool to build both OCI and Docker images.

Let's install Podman. First source the host OS version information, to make it available as environment variables to subsequent commands:

```
student@ubuntu:~$ . /etc/os-release
```

Add the repository to the list of apt sources:

```
student@ubuntu:~$ sudo sh -c "echo 'deb
http://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable/
xUbuntu_${VERSION_ID}/ /' >
/etc/apt/sources.list.d/devel:kubic:libcontainers:stable.list"
```

Retrieve the specific release key:

```
wget -nv
https://download.opensuse.org/repositories/devel:kubic:libcontainers:stable/xU
buntu_${VERSION_ID}/Release.key -O- | sudo apt-key add -
2020-03-10 21:26:15
URL:https://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/st
able/xUbuntu_18.04/Release.key [1094/1094] -> "-" [1]
OK
```

Update package index:

```
student@ubuntu:~$ sudo apt update -qq
```

Install:

```
student@ubuntu:~$ sudo apt -qq -y install podman
The following packages were automatically installed and are no longer
required:
  grub-pc-bin libnuma1 ubuntu-fan
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  catatonit containernetworking-plugins containers-common containers-image
podman-plugins slirp4netns
Recommended packages:
  varlink
The following NEW packages will be installed:
  catatonit containernetworking-plugins podman podman-plugins slirp4netns
The following packages will be upgraded:
  containers-common containers-image
...
```

Once installed, let's verify the version:

```
student@ubuntu:~$ podman --version
podman version 1.8.0
```

Lab 5.1 - Image Operations with Docker

Docker commands are run as **root**. However, a more secure method is to run them with **sudo**. Search the Docker Hub registry for container images, using **nginx** as search term:

```
student@ubuntu:~$ sudo docker search nginx
NAME                                DESCRIPTION
STARS               OFFICIAL            AUTOMATED
nginx                               Official build of Nginx.
12788               [OK]
jwilder/nginx-proxy                 Automated Nginx reverse proxy for docker
con…   1750                                 [OK]
richarvey/nginx-php-fpm             Container running Nginx + PHP-FPM capable
of…    758                                [OK]
linuxserver/nginx                   An Nginx container, brought to you by
LinuxS…    95
bitnami/nginx                       Bitnami nginx Docker Image
77                                       [OK]
tiangolo/nginx-rtmp                 Docker image with Nginx using the
nginx-rtmp…    64                                      [OK]
jc21/nginx-proxy-manager            Docker container for managing Nginx proxy
ho…    45
nginxdemos/hello                    NGINX webserver that serves a simple page
co…    41                                 [OK]
nginx/unit                          NGINX Unit is a dynamic web and application
…    36
jlesage/nginx-proxy-manager         Docker container for Nginx Proxy Manager
35                                       [OK]
nginx/nginx-ingress                 NGINX Ingress Controller for Kubernetes
28
```

Pull an image from the Docker Hub:

```
student@ubuntu:~$ sudo docker image pull nginx
Using default tag: latest
latest: Pulling from library/nginx
68ced04f60ab: Pull complete
28252775b295: Pull complete
a616aa3b0bf2: Pull complete
Digest:
sha256:2539d4344dd18e1df02be842ffc435f8e1f699cfc55516e2cf2cb16b7a9aea0b
Status: Downloaded newer image for nginx:latest
docker.io/library/nginx:latest
```

Pull an image from a private registry, by specifying registry name, port number, image name and tag:

```
student@ubuntu:~$ sudo docker image pull myregistry.com:5000/alpine:latest
```

List images cached in the local repository:

```
student@ubuntu:~$ sudo docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
nginx               latest              6678c7c2e56c        6 days ago
127MB
hello-world         latest              fce289e99eb9        14 months ago
1.84kB
```

List images and their digests. A digest is a hash associated with the content of each layer part of the image. Digests ensure each layer's integrity. The image ID is another hash, derived from the JSON configuration file of the image:

```
student@ubuntu:~$ sudo docker image ls --digests
REPOSITORY          TAG                 DIGEST
IMAGE ID            CREATED             SIZE
nginx               latest
sha256:2539d4344dd18e1df02be842ffc435f8e1f699cfc55516e2cf2cb16b7a9aea0b
6678c7c2e56c        6 days ago          127MB
hello-world         latest
sha256:fc6a51919cfeb2e6763f62b6d9e8815acbf7cd2e476ea353743570610737b752
fce289e99eb9        14 months ago       1.84kB
```

Push an image to Docker Hub. Assuming the existence of a lfstudent account on Docker Hub, a user can login from the CLI and then push an image into the registry to be shared by other users:

```
student@ubuntu:~$ sudo docker login

Login with your Docker ID to push and pull images from Docker Hub. If you
don't
have a Docker ID, head over to https://hub.docker.com to create one.
Username: lfstudent
Password: ********
Login Succeeded

student@ubuntu:~$ sudo docker image push lfstudent/alpine:training
The push refers to a repository [docker.io/lfstudent/alpine]
724d404d96ef: Pushed
60ab55d3379d: Mounted from library/alpine
training: digest:
sha256:fcc29a8a772bed232fc3026f9ea5df745e57ea4c99b2306f997036510d4be0f9 size:
735
```

Push an image to a private registry:

```
student@ubuntu:~$ sudo docker image push myregistry.com:5000/alpine:training
```

Remove unused cached images from local repository

```
student@ubuntu:~$ sudo docker image prune
WARNING! This will remove all dangling images.
Are you sure you want to continue? [y/N]
y
Deleted Images:
deleted:
sha256:74688f28366cd966f203b54f175de233846e7b720eb2e0d09fae06a26a939779
Total reclaimed space: 0 B
```

Remove one or more cached images from the local repository

```
student@ubuntu:~$ sudo docker image rm -f alpine:latest nginx:latest
Untagged: alpine:latest
Untagged:
alpine@sha256:dfbd4a3a8ebca874ebd2474f044a0b33600d4523d03b0df76e5c5986cb02d7e8
Untagged: nginx:latest
Untagged:
nginx@sha256:f2d384a6ca8ada733df555be3edc427f2e5f285ebf468aae940843de8cf74645
Deleted:
sha256:cc1b614067128cd2f5cdafb258b0a4dd25760f14562bcce516c13f760c3b79c4
Deleted:
sha256:a92b2e17cabb27c7e920fba7d683b402e0ab99b4658faecd0d889cd98007193f
```

```
Deleted:
sha256:db2175a8ce095b094fcb38584f8fd47298de0720aede8b29125e9befb8e56912
Deleted:
sha256:a2ae92ffcd29f7ededa0320f4a4fd709a723beae9a4e681696874932db7aee2c
```

Display image details:

```
student@ubuntu:~$ sudo docker image inspect nginx
[
    {
        "Id":
"sha256:6678c7c2e56c970388f8d5a398aa30f2ab60e85f20165e101053c3d3a11e6663",
        "RepoTags": [
            "nginx:latest"
        ],
        "RepoDigests": [

"nginx@sha256:2539d4344dd18e1df02be842ffc435f8e1f699cfc55516e2cf2cb16b7a9aea0b
"
        ],
        "Parent": "",
        "Comment": "",
        "Created": "2020-03-04T17:31:55.614610625Z",
        "Container":
"c42767258d0cefc082164aa63f0f9c3a261076f9cec29dbc9e236ed38a89e866",
        "ContainerConfig": {
            "Hostname": "c42767258d0c",
            "Domainname": "",
            "User": "",
            "AttachStdin": false,
            "AttachStdout": false,
            "AttachStderr": false,
            "ExposedPorts": {
                "80/tcp": {}
            },
            "Tty": false,
            "OpenStdin": false,
            "StdinOnce": false,
            "Env": [

"PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
                "NGINX_VERSION=1.17.9",
                "NJS_VERSION=0.3.9",
                "PKG_RELEASE=1~buster"
            ],
            "Cmd": [
                "/bin/sh",
```

```
                "-c",
                "#(nop) ",
                "CMD [\"nginx\" \"-g\" \"daemon off;\"]"
            ],
            "ArgsEscaped": true,
            "Image":
"sha256:38b0ff683cb34665b59ef69009de267ccbc2154deb72ca0947858a22f5db42cd",
            "Volumes": null,
            "WorkingDir": "",
            "Entrypoint": null,
            "OnBuild": null,
            "Labels": {
                "maintainer": "NGINX Docker Maintainers
<docker-maint@nginx.com>"
            },
            "StopSignal": "SIGTERM"
        },
        "DockerVersion": "18.09.7",
        "Author": "",
        "Config": {
            "Hostname": "",
            "Domainname": "",
            "User": "",
            "AttachStdin": false,
            "AttachStdout": false,
            "AttachStderr": false,
            "ExposedPorts": {
                "80/tcp": {}
            },
            "Tty": false,
            "OpenStdin": false,
            "StdinOnce": false,
            "Env": [

"PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
                "NGINX_VERSION=1.17.9",
                "NJS_VERSION=0.3.9",
                "PKG_RELEASE=1~buster"
            ],
            "Cmd": [
                "nginx",
                "-g",
                "daemon off;"
            ],
            "ArgsEscaped": true,
            "Image":
"sha256:38b0ff683cb34665b59ef69009de267ccbc2154deb72ca0947858a22f5db42cd",
            "Volumes": null,
```

```
            "WorkingDir": "",
            "Entrypoint": null,
            "OnBuild": null,
            "Labels": {
                "maintainer": "NGINX Docker Maintainers
<docker-maint@nginx.com>"
            },
            "StopSignal": "SIGTERM"
        },
        "Architecture": "amd64",
        "Os": "linux",
        "Size": 126768999,
        "VirtualSize": 126768999,
        "GraphDriver": {
            "Data": {
                "LowerDir":
"/var/lib/docker/overlay2/17462c46bdbcefa3c0c76435085ab7061db3f59eedca30e10d8e
c40228fbb7f5/diff:/var/lib/docker/overlay2/0909f0faa6b4451f12dd340ff1fac11d505
2608bd74af0533b39e169b3355cd5/diff",
                "MergedDir":
"/var/lib/docker/overlay2/acd3c90885b61e04cc2704974ebf3dc5f95287e0b7a07cba128f
3dea09bb89c0/merged",
                "UpperDir":
"/var/lib/docker/overlay2/acd3c90885b61e04cc2704974ebf3dc5f95287e0b7a07cba128f
3dea09bb89c0/diff",
                "WorkDir":
"/var/lib/docker/overlay2/acd3c90885b61e04cc2704974ebf3dc5f95287e0b7a07cba128f
3dea09bb89c0/work"
            },
            "Name": "overlay2"
        },
        "RootFS": {
            "Type": "layers",
            "Layers": [

"sha256:f2cb0ecef392f2a630fa1205b874ab2e2aedf96de04d0b8838e4e728e28142da",

"sha256:71f2244bc14dacf7f73128b4b89b1318f41a9421dffc008c2ba91bb6dc2716f1",

"sha256:55a77731ed2630d9c092258490b03be3491d5f245fe13a1c6cb4e21babfb15b7"
            ]
        },
        "Metadata": {
            "LastTagTime": "0001-01-01T00:00:00Z"
        }
    }
]
```

Lab 5.2 - Image Operations with rkt

Rkt also is run as root, but a more secure method is to run it with sudo. Let's fetch an image from the registry to the local repository:

```
student@ubuntu:~$ sudo rkt fetch coreos.com/etcd:v3.1.7
image: keys already exist for prefix "coreos.com/etcd", not fetching again
Downloading signature: [=======================================] 490 B/490 B
Downloading ACI: [=============================================] 9.47 MB/9.47
MB
image: signature verified:
  CoreOS Application Signing Key <security@coreos.com>
sha512-e7a54697d04ddae899ed1bfd263235cb
```

Fetch a Docker image. One of rkt's advantages is its flexibility to retrieve images from its own registry and from Docker Hub registry as well. Unfortunately, it is not able to check the integrity of Docker images.

```
student@ubuntu:~$ sudo rkt fetch --insecure-options=image docker://nginx
Downloading sha256:a616aa3b0bf [==============================]     203 B / 203
B
Downloading sha256:68ced04f60a [==============================] 27.1 MB / 27.1
MB
Downloading sha256:28252775b29 [==============================] 23.9 MB / 23.9
MB
sha512-d00ac58834f14e71aadbc781a8d9bd30
```

List images cached in the local repository:

```
student@ubuntu:~$ sudo rkt image list
ID                NAME                                SIZE   IMPORT TIME LAST
USED
sha512-e7a54697d04d     coreos.com/etcd:v3.1.7        58MiB 8 minutes ago
8 minutes ago
```

```
sha512-d00ac58834f1     registry-1.docker.io/library/nginx:latest 244MiB59
seconds ago 56 seconds ago
sha512-402c24acf901     coreos.com/etcd:v3.1.1        58MiB 10 seconds ago
9 seconds ago
```

Remove an image from the local cached repository, then list images to verify successful removal:

```
student@ubuntu:~$ sudo rkt image rm coreos.com/etcd:v3.1.1
successfully removed aci for image:
"sha512-402c24acf901a7e4933c7fecd52ff1606cf3b12c221c521c321d2c9376b70c4f"
rm: 1 image(s) successfully removed

student@ubuntu:~$ sudo rkt image list
ID                NAME                                  SIZE  IMPORT TIME LAST
USED
sha512-e7a54697d04d     coreos.com/etcd:v3.1.7              58MiB 24 minutes
ago   24 minutes ago
sha512-d00ac58834f1     registry-1.docker.io/library/nginx:latest 244MiB17
minutes ago 17 minutes ago
```

Lab 5.3 - Image Operations with Podman

In this exercise we will focus on image operations allowed by Podman, while in a later chapter we will explore image building capabilities of both Buildah and Podman.

Podman operations closely resemble operations explored earlier with Docker. Let's take a closer look at a few Podman operations. Searching the registries for an image:

```
student@ubuntu:~$ podman search --filter=is-official nginx
INDEX      NAME                      DESCRIPTION               STARS  OFFICIAL
AUTOMATED
Docker.io  docker.io/library/nginx  Official build of Nginx.  12795  [OK]
```

Pulling an image from the registry to the local repository:

```
student@ubuntu:~$ podman image pull docker.io/library/nginx
Trying to pull docker.io/library/nginx...
Getting image source signatures
Copying blob 28252775b295 done
Copying blob a616aa3b0bf2 done
Copying blob 68ced04f60ab done
Copying config 6678c7c2e5 [======================================] 6.5KiB /
6.5KiB
Writing manifest to image destination
Storing signatures
6678c7c2e56c970388f8d5a398aa30f2ab60e85f20165e101053c3d3a11e6663
```

List images in the local repository:

```
student@ubuntu:~$ podman image list
REPOSITORY                TAG      IMAGE ID       CREATED      SIZE
docker.io/library/nginx   latest   6678c7c2e56c   7 days ago   131 MB
```

Display the details of an image from the local repository:

```
student@ubuntu:~$ podman image inspect nginx
[
    {
        "Id":
"6678c7c2e56c970388f8d5a398aa30f2ab60e85f20165e101053c3d3a11e6663",
        "Digest":
"sha256:2539d4344dd18e1df02be842ffc435f8e1f699cfc55516e2cf2cb16b7a9aea0b",
        "RepoTags": [
            "docker.io/library/nginx:latest"
        ],
        "RepoDigests": [

"docker.io/library/nginx@sha256:2539d4344dd18e1df02be842ffc435f8e1f699cfc55516
e2cf2cb16b7a9aea0b",

"docker.io/library/nginx@sha256:3936fb3946790d711a68c58be93628e43cbca72439079e
16d154b5db216b58da"
        ],
        "Parent": "",
        "Comment": "",
        "Created": "2020-03-04T17:31:55.614610625Z",
        "Config": {
            "ExposedPorts": {
                "80/tcp": {}
            },
            "Env": [

"PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
                "NGINX_VERSION=1.17.9",
                "NJS_VERSION=0.3.9",
                "PKG_RELEASE=1~buster"
            ],
            "Cmd": [
                "nginx",
                "-g",
                "daemon off;"
            ],
            "Labels": {
                "maintainer": "NGINX Docker Maintainers
<docker-maint@nginx.com>"
            },
            "StopSignal": "SIGTERM"
        },
        "Version": "18.09.7",
        "Author": "",
```

```
        "Architecture": "amd64",
        "Os": "linux",
        "Size": 130607350,
        "VirtualSize": 130607350,
        "GraphDriver": {
            "Name": "vfs",
            "Data": null
        },
        "RootFS": {
            "Type": "layers",
            "Layers": [

"sha256:f2cb0ecef392f2a630fa1205b874ab2e2aedf96de04d0b8838e4e728e28142da",

"sha256:71f2244bc14dacf7f73128b4b89b1318f41a9421dffc008c2ba91bb6dc2716f1",

"sha256:55a77731ed2630d9c092258490b03be3491d5f245fe13a1c6cb4e21babfb15b7"
            ]
        },
        "Labels": {
            "maintainer": "NGINX Docker Maintainers <docker-maint@nginx.com>"
        },
        "Annotations": {},
        "ManifestType":
"application/vnd.docker.distribution.manifest.v2+json",
        "User": "",
        "History": [
            {
                "created": "2020-02-26T00:37:39.301941924Z",
                "created_by": "/bin/sh -c #(nop) ADD
file:e5a364615e0f6961626089c7d658adbf8c8d95b3ae95a390a8bb33875317d434 in / "
            },
            {
                "created": "2020-02-26T00:37:39.539684396Z",
                "created_by": "/bin/sh -c #(nop)  CMD [\"bash\"]",
                "empty_layer": true
            },
            {
                "created": "2020-02-26T20:01:52.907016299Z",
                "created_by": "/bin/sh -c #(nop)  LABEL maintainer=NGINX
Docker Maintainers <docker-maint@nginx.com>",
                "empty_layer": true
            },
            {
                "created": "2020-03-04T17:31:33.533938237Z",
                "created_by": "/bin/sh -c #(nop)  ENV NGINX_VERSION=1.17.9",
                "empty_layer": true
            },
```

```
            {
                "created": "2020-03-04T17:31:33.742602249Z",
                "created_by": "/bin/sh -c #(nop)  ENV NJS_VERSION=0.3.9",
                "empty_layer": true
            },
            {
                "created": "2020-03-04T17:31:33.91932088Z",
                "created_by": "/bin/sh -c #(nop)  ENV PKG_RELEASE=1~buster",
                "empty_layer": true
            },
            {
                "created": "2020-03-04T17:31:54.307659955Z",
    ...
    }
]
```

Display the updates and changes history of an image:

```
student@ubuntu:~$ podman image history nginx
ID             CREATED       CREATED BY
SIZE      COMMENT
6678c7c2e56c   7 days ago    /bin/sh -c #(nop) CMD ["nginx" "-g" "daemo...
0B
<missing>      7 days ago    /bin/sh -c #(nop) STOPSIGNAL SIGTERM
0B
<missing>      7 days ago    /bin/sh -c #(nop) EXPOSE 80
0B
<missing>      7 days ago    /bin/sh -c ln -sf /dev/stdout /var/log/ngi...
3.584kB
<missing>      7 days ago    /bin/sh -c set -x && addgroup --system --g...
58.11MB
<missing>      7 days ago    /bin/sh -c #(nop) ENV PKG_RELEASE=1~buster
0B
<missing>      7 days ago    /bin/sh -c #(nop) ENV NJS_VERSION=0.3.9
0B
<missing>      7 days ago    /bin/sh -c #(nop) ENV NGINX_VERSION=1.17.9
0B
<missing>      2 weeks ago   /bin/sh -c #(nop) LABEL maintainer=NGINX D...
0B
<missing>      2 weeks ago   /bin/sh -c #(nop) CMD ["bash"]
0B
<missing>      2 weeks ago   /bin/sh -c #(nop) ADD file:e5a364615e0f696...
72.48MB
```

Remove an image from the local repository (image rm):

```
student@ubuntu:~$ podman image rm nginx
Untagged: docker.io/library/nginx:latest
Deleted: 6678c7c2e56c970388f8d5a398aa30f2ab60e85f20165e101053c3d3a11e6663
```

Remove an image from the local repository (rmi):

```
student@ubuntu:~$ podman rmi nginx
Untagged: docker.io/library/nginx:latest
Deleted: 6678c7c2e56c970388f8d5a398aa30f2ab60e85f20165e101053c3d3a11e6663
```

Remove all dangling images from local repository:

```
student@ubuntu:~$ sudo podman image prune

WARNING! This will remove all dangling images.
Are you sure you want to continue? [y/N] y
6678c7c2e56c970388f8d5a398aa30f2ab60e85f20165e101053c3d3a11e6663
```

Remove all unused images from local repository:

```
student@ubuntu:~$ sudo podman image prune -a -f
6678c7c2e56c970388f8d5a398aa30f2ab60e85f20165e101053c3d3a11e6663
```

Lab 6.1 - Container Operations with runc

Prior to being able to perform container operations with runc, we need to create a container in an OCI bundle format. We will use a busybox Docker container to export its filesystem in a tar archive, and use the extracted filesystem at as the rootfs for the runc container:

```
student@ubuntu:~$ mkdir -p runc-container/rootfs

student@ubuntu:~$ sudo docker container export \
  $(sudo docker container create busybox) \
  > busybox.tar
Unable to find image 'busybox:latest' locally
latest: Pulling from library/busybox
0669b0daf1fb: Pulling fs layer
0669b0daf1fb: Verifying Checksum
0669b0daf1fb: Download complete
0669b0daf1fb: Pull complete
Digest:
sha256:b26cd013274a657b86e706210ddd5cc1f82f50155791199d29b9e86e935ce135
Status: Downloaded newer image for busybox:latest

student@ubuntu:~$ tar -C runc-container/rootfs/ -xf busybox.tar

student@ubuntu:~$ cd runc-container/rootfs/

student@ubuntu:~/runc-container/rootfs$ ls
bin  dev  etc  home  proc  root  sys  tmp  usr  var
```

Aside from rootfs, runc requires a spec configuration file to start a container. Runc allows us to create a sample spec file:

```
student@ubuntu:~/runc-container/rootfs$ cd ..

student@ubuntu:~/runc-container$ runc spec
```

```
student@ubuntu:~/runc-container$ ls
config.json  rootfs
```

Display the content of the config.json file. Observe some of the sections in the file, such as **process**, **root**, and **namespaces**. The process section specifies a shell process that will run in a terminal as root (uid 0, gid 0). The root directory of the container is mapped, in a read-only mode, to the rootfs generated earlier. Namespaces specifies all the namespaces that the container needs to have for the isolation of pid, network, ipc, hostname, and mount.

```
student@ubuntu:~/runc-container$ cat config.json
{
	"ociVersion": "1.0.1-dev",
	"process": {
		"terminal": true,
		"user": {
			"uid": 0,
			"gid": 0
		},
		"args": [
			"sh"
		],
		"env": [

"PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
			"TERM=xterm"
		],
		"cwd": "/",
		"capabilities": {
			"bounding": [
				"CAP_AUDIT_WRITE",
				"CAP_KILL",
				"CAP_NET_BIND_SERVICE"
			],
			"effective": [
				"CAP_AUDIT_WRITE",
				"CAP_KILL",
				"CAP_NET_BIND_SERVICE"
			],
			"inheritable": [
				"CAP_AUDIT_WRITE",
				"CAP_KILL",
				"CAP_NET_BIND_SERVICE"
			],
			"permitted": [
				"CAP_AUDIT_WRITE",
				"CAP_KILL",
```

```
                        "CAP_NET_BIND_SERVICE"
                ],
                "ambient": [
                        "CAP_AUDIT_WRITE",
                        "CAP_KILL",
                        "CAP_NET_BIND_SERVICE"
                ]
        },
        "rlimits": [
                {
                        "type": "RLIMIT_NOFILE",
                        "hard": 1024,
                        "soft": 1024
                }
        ],
        "noNewPrivileges": true
},
"root": {
        "path": "rootfs",
        "readonly": true
},
"hostname": "runc",
"mounts": [
        {
                "destination": "/proc",
                "type": "proc",
                "source": "proc"
        },
        {
                "destination": "/dev",
                "type": "tmpfs",
                "source": "tmpfs",
                "options": [
                        "nosuid",
                        "strictatime",
                        "mode=755",
                        "size=65536k"
                ]
        },
        {
                "destination": "/dev/pts",
                "type": "devpts",
                "source": "devpts",
                "options": [
                        "nosuid",
                        "noexec",
                        "newinstance",
                        "ptmxmode=0666",
```

```
                "mode=0620",
                "gid=5"
            ]
        },
        {
            "destination": "/dev/shm",
            "type": "tmpfs",
            "source": "shm",
            "options": [
                "nosuid",
                "noexec",
                "nodev",
                "mode=1777",
                "size=65536k"
            ]
        },
        {
            "destination": "/dev/mqueue",
            "type": "mqueue",
            "source": "mqueue",
            "options": [
                "nosuid",
                "noexec",
                "nodev"
            ]
        },
        {
            "destination": "/sys",
            "type": "sysfs",
            "source": "sysfs",
            "options": [
                "nosuid",
                "noexec",
                "nodev",
                "ro"
            ]
        },
        {
            "destination": "/sys/fs/cgroup",
            "type": "cgroup",
            "source": "cgroup",
            "options": [
                "nosuid",
                "noexec",
                "nodev",
                "relatime",
                "ro"
            ]
```

```
        }
    ],
    "linux": {
        "resources": {
            "devices": [
                {
                    "allow": false,
                    "access": "rwm"
                }
            ]
        },
        "namespaces": [
            {
                "type": "pid"
            },
            {
                "type": "network"
            },
            {
                "type": "ipc"
            },
            {
                "type": "uts"
            },
            {
                "type": "mount"
            }
        ],
        "maskedPaths": [
            "/proc/acpi",
            "/proc/asound",
            "/proc/kcore",
            "/proc/keys",
            "/proc/latency_stats",
            "/proc/timer_list",
            "/proc/timer_stats",
            "/proc/sched_debug",
            "/sys/firmware",
            "/proc/scsi"
        ],
        "readonlyPaths": [
            "/proc/bus",
            "/proc/fs",
            "/proc/irq",
            "/proc/sys",
            "/proc/sysrq-trigger"
        ]
    }
```

```
}
```

Now we are ready to start the container. Make sure to leave this terminal window as it is, with the running shell from the busybox container:

```
student@ubuntu:~/runc-container$ sudo runc run busybox
/ #
```

Open a second terminal on you VM instance and list containers:

```
student@ubuntu:~$ sudo runc list
ID          PID    STATUS      BUNDLE
CREATED                          OWNER
busybox    361    running     /home/student/runc-container
2020-03-12T08:52:58.030568873Z   root
```

Also from the second terminal, list the processes running inside the busybox container:

```
student@ubuntu:~$ sudo runc ps busybox
UID         PID   PPID  C STIME TTY          TIME CMD
root        361    341  0 08:52 pts/0    00:00:00 sh
```

Also from the second terminal, list the events of the busybox container:

```
student@ubuntu:~$ sudo runc events busybox
{"type":"stats","id":"busybox","data":{"cpu":{"usage":{"total":20301701,"percp
u":[20301701],"kernel":10000000,"user":0},"throttling":{}},"memory":{"usage":{
"limit":9223372036854771712,"usage":495616,"max":5455872,"failcnt":0},"swap":{
"limit":0,"failcnt":0},"kernel":{"limit":9223372036854771712,"usage":442368,"m
ax":643072,"failcnt":0},"kernelTCP":{"limit":9223372036854771712,"failcnt":0},
"raw":{"active_anon":53248,"active_file":0,"cache":0,"dirty":0,"hierarchical_m
emory_limit":9223372036854771712,"inactive_anon":0,"inactive_file":0,"mapped_f
ile":0,"pgfault":2376,"pgmajfault":0,"pgpgin":2046,"pgpgout":2050,"rss":4096,"
rss_huge":0,"shmem":0,"total_active_anon":53248,"total_active_file":0,"total_c
ache":0,"total_dirty":0,"total_inactive_anon":0,"total_inactive_file":0,"total
_mapped_file":0,"total_pgfault":2376,"total_pgmajfault":0,"total_pgpgin":2046,
"total_pgpgout":2050,"total_rss":4096,"total_rss_huge":0,"total_shmem":0,"tota
l_unevictable":0,"total_writeback":0,"unevictable":0,"writeback":0}},"pids":{"
current":1},"blkio":{},"hugetlb":{"1GB":{"failcnt":0},"2MB":{"failcnt":0}},"in
tel_rdt":{},"network_interfaces":null}}
...
```

Runc allows for a container to be paused and then resumed. From the second terminal list containers and issue the pause command, then list again to confirm the paused status. Return to the first terminal and try to type a command at the shell prompt. No command will be registered or executed because the container is paused. Now return to the second terminal and resume the container, listing containers again to confirm running status. Return to the first terminal to see that the commands typed before are now displayed, together with the expected output.

```
student@ubuntu:~$ sudo runc list
ID          PID         STATUS      BUNDLE                          CREATED
OWNER
busybox     361         running     /home/student/runc-container
2020-03-12T08:52:58.030568873Z   root
student@ubuntu:~$ sudo runc pause busybox
student@ubuntu:~$ sudo runc list
ID          PID         STATUS      BUNDLE                          CREATED
OWNER
busybox     361         paused      /home/student/runc-container
2020-03-12T08:52:58.030568873Z   root

student@ubuntu:~$ sudo runc resume busybox
student@ubuntu:~$ sudo runc list
ID          PID         STATUS      BUNDLE                          CREATED
OWNER
busybox     361         running     /home/student/runc-container
2020-03-12T08:52:58.030568873Z   root
```

The container status can be displayed, again, from the second terminal window:

```
student@ubuntu:~$ sudo runc state busybox
{
  "ociVersion": "1.0.1-dev",
  "id": "busybox",
  "pid": 361,
  "status": "running",
  "bundle": "/home/student/runc-container",
  "rootfs": "/home/student/runc-container/rootfs",
  "created": "2020-03-12T08:52:58.030568873Z",
  "owner": ""
}
```

There are a few methods to delete this container. From the second terminal, because it is running we need to supply the -f option to force the delete command. If the container were stopped, then -f would not be necessary. Another method would be to exit out of the shell running in the first terminal window, which would terminate the shell process running in the busybox container, and as a result the busybox container will be removed as well:

```
student@ubuntu:~$ sudo runc delete -f busybox
student@ubuntu:~$ sudo runc list
ID          PID         STATUS      BUNDLE      CREATED     OWNER
```

Lab 6.2 - Container Operations with Docker

Pull an image from the Docker Hub registry to the local repository, to start exploring container operations supported by Docker:

```
student@ubuntu:~$ sudo docker image pull alpine
Using default tag: latest
latest: Pulling from library/alpine
0a8490d0dfd3: Pull complete
Digest:
sha256:dfbd4a3a8ebca874ebd2474f044a0b33600d4523d03b0df76e5c5986cb02d7e8
Status: Downloaded newer image for alpine:latest
```

By default, Docker pulls images from Docker Hub registry. We can pull images from a private registry as well:

```
student@ubuntu:~$ sudo docker image pull <private_registry>:<port>/image
```

List images available in the local repository:

```
student@ubuntu:~$ sudo docker image ls
REPOSITORY      TAG         IMAGE ID              CREATED         SIZE
alpine          latest       88e169ea8f46         4 weeks ago  3.98 MB
```

Create a container from the image available in the local repository. This command does not start the container, it only creates it:

```
student@ubuntu:~$ sudo docker container create -it alpine sh
6ac952f2c282216dd871d8f958e82550d4228a0fa24067589fe9eda48f730b69
```

Start the created container, using a partial container ID. The container will start the sh program, as we provided it as a COMMAND argument:

```
student@ubuntu:~$ sudo docker container start 6ac
6ac
```

List running containers:

```
student@ubuntu:~$ sudo docker container ls
CONTAINER ID     IMAGE     COMMAND     CREATED          STATUS
PORTS     NAMES
6ac952f2c282        alpine      "sh"                5 minutes ago     Up
About a minute              dreamy_golick
```

Running a container instead of creating & starting. The **-t** option allocates a pseudo-TTY, and the **-i** option keeps the STDIN open in interactive mode. Both **-i** and **-t** options can be combined into a **-it** or **-ti** notation, all with the same effect. The **--name** option allows the myalpine name to be assigned to the running container.

```
student@ubuntu:~$ sudo docker container run -it --name myalpine alpine sh
```

Detaching from a running container ensures the container remains running. By pressing the **Ctrl p + Ctrl q** key combination in the terminal of a running container:

```
student@ubuntu:~$ sudo docker container run -it --name myalpine alpine sh
/ # Ctrl p + Ctrl q
```

The container is being detached, yet still running. Other methods to close the shell process in the terminal window would terminate the container as well. We can confirm the detached/running container by listing the running containers:

```
student@ubuntu:~$ sudo docker container ls
CONTAINER ID   IMAGE    COMMAND  CREATED        STATUS    PORTS     NAMES
a9d171545e62   alpine   "sh"    2 minutes ago  Up 2 minutes    myalpine
6ac952f2c282   alpine   "sh"    20 minutes ago Up 20 minutes   dreamy_golick
```

We can attach a running container. As a result we receive a shell into the container::

```
student@ubuntu:~$ sudo docker container attach myalpine
/ #
```

We can run a container in the background, in which case we receive an output with the container ID:

```
student@ubuntu:~$ sudo docker container run -d alpine /bin/sh -c 'while [ 1 ];
do echo "hello world from container"; sleep 1; done'
be9f86d4d1df561210ac7df15fbaec1222f5e763f1a26b4ed5919e5825efaa7c
```

Display container logs:

```
student@ubuntu:~$ sudo docker container logs be9
hello world from container
hello world from container
hello world from container
hello world from container
...
```

Stop a running container:

```
student@ubuntu:~$ sudo docker container stop be9f86d4d1df
be9f86d4d1df
```

List all containers (running and stopped):

```
student@ubuntu:~$ sudo docker container ls -a
CONTAINER ID   IMAGE    COMMAND                  CREATED         STATUS
PORTS                  NAMES
be9f86d4d1df   alpine   "/bin/sh -c 'while..."   10 minutes ago  Exited (137)
53 seconds ago          elastic_yalow
...
```

Start a stopped container:

```
student@ubuntu:~$ sudo docker container start be9f86d4d1df
be9f86d4d1df

student@ubuntu:~$ sudo docker container ls -a
CONTAINER ID   IMAGE    COMMAND                  CREATED           STATUS
PORTS      NAMES
be9f86d4d1df   alpine   "/bin/sh -c 'while..."   14 minutes ago  Up About a
minute     elastic_yalow
...
```

Restarting a container. This command stops and then starts a container, but it does not change the container ID or its name:

```
student@ubuntu:~$ sudo docker container restart be9f86d4d1df
be9f86d4d1df
```

Pausing and resuming a container:

```
student@ubuntu:~$ sudo docker container pause be9f86d4d1df
be9f86d4d1df

student@ubuntu:~$ sudo docker container ls -a
CONTAINER ID   IMAGE    COMMAND                  CREATED           STATUS
PORTS      NAMES
be9f86d4d1df   alpine   "/bin/sh -c 'while..."   20 minutes ago  Up 3 minutes
(Paused)     elastic_yalow
...

student@ubuntu:~$ sudo docker container unpause be9f86d4d1df
be9f86d4d1df

student@ubuntu:~$ sudo docker container ls -a
CONTAINER ID   IMAGE    COMMAND                  CREATED           STATUS
PORTS      NAMES
be9f86d4d1df   alpine   "/bin/sh -c 'while..."   22 minutes ago  Up 5 minutes
elastic_yalow
...
```

Renaming a running container. Let's rename the be9f86d4d1df container, from elastic_yalow to hello_world_loop:

```
student@ubuntu:~$ sudo docker container rename elastic_yalow hello_world_loop

student@ubuntu:~$ sudo docker container ls -a
CONTAINER ID   IMAGE    COMMAND                  CREATED           STATUS
PORTS      NAMES
be9f86d4d1df   alpine   "/bin/sh -c 'while..."   26 minutes ago  Up 9 minutes
hello_world_loop
...
```

Deleting or removing a container. There are two separate options available to remove containers. The default command that removes stopped containers, and a force option to remove running containers:

```
student@ubuntu:~$ sudo docker container stop hello_world_loop
hello_world_loop
student@ubuntu:~$ sudo docker container rm hello_world_loop
hello_world_loop

student@ubuntu:~$ sudo docker container ls -a
CONTAINER ID   IMAGE    COMMAND  CREATED        STATUS    PORTS      NAMES
a9d171545e62   alpine   "sh"    45 minutes ago  Up 45 minutes      myalpine
...

student@ubuntu:~$ sudo docker container rm -f myalpine
myalpine

student@ubuntu:~$ sudo docker container ls -a
CONTAINER ID   IMAGE    COMMAND  CREATED        STATUS    PORTS      NAMES
6ac952f2c282   alpine   "sh"    50 minutes ago  Up 50 minutes
dreamy_golick
...
```

Automatically remove a container upon its exit. In order to automate the removal process, to avoid repetitive tasks to manually find terminated containers and manually remove them, we can automate the

removal process by passing an option when we run the container. As a result, we will no longer see terminated containers in their stopped state:

```
student@ubuntu:~$ sudo docker container run --rm --name auto_rm alpine ping -c
3 google.com
PING google.com (172.217.2.14): 56 data bytes
64 bytes from 172.217.2.14: seq=0 ttl=58 time=1.780 ms
64 bytes from 172.217.2.14: seq=1 ttl=58 time=1.682 ms
64 bytes from 172.217.2.14: seq=2 ttl=58 time=1.870 ms

--- google.com ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 1.682/1.777/1.870 ms

student@ubuntu:~$ sudo docker container ls -a
CONTAINER ID   IMAGE    COMMAND  CREATED        STATUS    PORTS      NAMES
6ac952f2c282   alpine   "sh"    50 minutes ago  Up 50 minutes
dreamy_golick
...
```

Setting the hostname of a container. Unless we explicitly set the hostname of a container, by default, at runtime, the container hostname is set to the container ID:

```
student@ubuntu:~$ sudo docker container run -h alpine-host -it --rm alpine sh
/ # hostname
alpine-host
/ # exit
```

Set the current working directory of a container:

```
student@ubuntu:~$ sudo docker container run -it -w /tmp/mypath --rm alpine sh
/tmp/mypath # pwd
/tmp/mypath
/tmp/mypath # exit
```

Set an environment variable of a container and assign a value to it. We are setting the WEB_HOST environment variable of the container, and assign an IP address to it:

```
student@ubuntu:~$ sudo docker container run -it --env "WEB_HOST=172.168.1.1"
--rm alpine sh
/ # env
HOSTNAME=9b9f7a458286
SHLVL=1
HOME=/root
TERM=xterm
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
PWD=/
WEB_HOST=172.168.1.1
/ # exit
```

Set the ulimit of a container. **Ulimit** is a command line tool to manage resource limits for users. It returns current limits for the user, but it can also set such resource limits. Let's display all the default limits on a new alpine container:

```
student@ubuntu:~$ sudo docker container run -it --rm alpine sh
/ # ulimit -a
core file size (blocks)         (-c) unlimited
data seg size (kb)              (-d) unlimited
scheduling priority             (-e) 0
file size (blocks)              (-f) unlimited
pending signals                 (-i) 6629
max locked memory (kb)          (-l) 16384
max memory size (kb)            (-m) unlimited
open files                      (-n) 1048576
POSIX message queues (bytes)    (-q) 819200
real-time priority              (-r) 0
stack size (kb)                 (-s) 8192
cpu time (seconds)              (-t) unlimited
max user processes              (-u) unlimited
virtual memory (kb)             (-v) unlimited
file locks                      (-x) unlimited
/ # exit
```

By default, user processes are unlimited. Let's try to limit the max user processes. By setting a limit we restrict the number of processes this container can create:

```
student@ubuntu:~$ sudo docker container run -it --ulimit nproc=10 --rm alpine
sh
/ # ulimit -a
core file size (blocks)         (-c) unlimited
data seg size (kb)              (-d) unlimited
scheduling priority             (-e) 0
file size (blocks)              (-f) unlimited
```

```
pending signals                 (-i) 6629
max locked memory (kb)          (-l) 16384
max memory size (kb)            (-m) unlimited
open files                      (-n) 1048576
POSIX message queues (bytes)    (-q) 819200
real-time priority              (-r) 0
stack size (kb)                 (-s) 8192
cpu time (seconds)              (-t) unlimited
max user processes              (-u) 10
virtual memory (kb)             (-v) unlimited
file locks                      (-x) unlimited
/ # exit
```

Display all the details of a container, such as hostname, IP address, attached volumes, image, and network configuration:

```
student@ubuntu:~$ sudo docker container ls
CONTAINER ID    IMAGE   COMMAND  CREATED       STATUS     PORTS   NAMES
b366d6cfcb74    alpine  "sh"     12 hours ago  Up 12 hours    dreamy_golick
...

student@ubuntu:~$ sudo docker container inspect b366d6cfcb74
[
    {
        "Id":
"b366d6cfcb742ed33acc4452f7c1bc91bde5314ef58bd5944f2e2660bd46d126",
        "Created": "2020-03-12T10:10:23.22934031Z",
        "Path": "sh",
        "Args": [],
        "State": {
            "Status": "running",
            "Running": true,
            "Paused": false,
            "Restarting": false,
            "OOMKilled": false,
            "Dead": false,
            "Pid": 1735,
            "ExitCode": 0,
            "Error": "",
            "StartedAt": "2020-03-12T10:10:24.034452971Z",
            "FinishedAt": "0001-01-01T00:00:00Z"
        },
        "Image":
"sha256:e7d92cdc71feacf90708cb59182d0df1b911f8ae022d29e8e95d75ca6a99776a",
```

```
        "ResolvConfPath":
"/var/lib/docker/containers/b366d6cfcb742ed33acc4452f7c1bc91bde5314ef58bd5944f
2e2660bd46d126/resolv.conf",
        "HostnamePath":
"/var/lib/docker/containers/b366d6cfcb742ed33acc4452f7c1bc91bde5314ef58bd5944f
2e2660bd46d126/hostname",
        "HostsPath":
"/var/lib/docker/containers/b366d6cfcb742ed33acc4452f7c1bc91bde5314ef58bd5944f
2e2660bd46d126/hosts",
        "LogPath":
"/var/lib/docker/containers/b366d6cfcb742ed33acc4452f7c1bc91bde5314ef58bd5944f
2e2660bd46d126/b366d6cfcb742ed33acc4452f7c1bc91bde5314ef58bd5944f2e2660bd46d12
6-json.log",
        "Name": "/dreamy_golick",
        "RestartCount": 0,
        "Driver": "overlay2",
        "Platform": "linux",
        "MountLabel": "",
        "ProcessLabel": "",
        "AppArmorProfile": "docker-default",
        "ExecIDs": null,
        "HostConfig": {
            "Binds": null,
            "ContainerIDFile": "",
            "LogConfig": {
                "Type": "json-file",
                "Config": {}
            },
            "NetworkMode": "default",
            "PortBindings": {},
            "RestartPolicy": {
                "Name": "no",
                "MaximumRetryCount": 0
            },
            "AutoRemove": false,
            "VolumeDriver": "",
            "VolumesFrom": null,
            "CapAdd": null,
            "CapDrop": null,
            "Capabilities": null,
            "Dns": [],
            "DnsOptions": [],
            "DnsSearch": [],
            "ExtraHosts": null,
            "GroupAdd": null,
            "IpcMode": "private",
            "Cgroup": "",
            "Links": null,
```

```
"OomScoreAdj": 0,
"PidMode": "",
"Privileged": false,
"PublishAllPorts": false,
"ReadonlyRootfs": false,
"SecurityOpt": null,
"UTSMode": "",
"UsernsMode": "",
"ShmSize": 67108864,
"Runtime": "runc",
"ConsoleSize": [
    0,
    0
],
"Isolation": "",
"CpuShares": 0,
"Memory": 0,
"NanoCpus": 0,
"CgroupParent": "",
"BlkioWeight": 0,
"BlkioWeightDevice": [],
"BlkioDeviceReadBps": null,
"BlkioDeviceWriteBps": null,
"BlkioDeviceReadIOps": null,
"BlkioDeviceWriteIOps": null,
"CpuPeriod": 0,
"CpuQuota": 0,
"CpuRealtimePeriod": 0,
"CpuRealtimeRuntime": 0,
"CpusetCpus": "",
"CpusetMems": "",
"Devices": [],
"DeviceCgroupRules": null,
"DeviceRequests": null,
"KernelMemory": 0,
"KernelMemoryTCP": 0,
"MemoryReservation": 0,
"MemorySwap": 0,
"MemorySwappiness": null,
"OomKillDisable": false,
"PidsLimit": null,
"Ulimits": null,
"CpuCount": 0,
"CpuPercent": 0,
"IOMaximumIOps": 0,
"IOMaximumBandwidth": 0,
"MaskedPaths": [
    "/proc/asound",
```

```
                "/proc/acpi",
                "/proc/kcore",
                "/proc/keys",
                "/proc/latency_stats",
                "/proc/timer_list",
                "/proc/timer_stats",
                "/proc/sched_debug",
                "/proc/scsi",
                "/sys/firmware"
            ],
            "ReadonlyPaths": [
                "/proc/bus",
                "/proc/fs",
                "/proc/irq",
                "/proc/sys",
                "/proc/sysrq-trigger"
            ]
        },
        "GraphDriver": {
            "Data": {
                "LowerDir":
"/var/lib/docker/overlay2/e86e35e0dee3c219b2467ac7423b462623788400ba2f05e903af
47eab8279ef4-init/diff:/var/lib/docker/overlay2/cb6f5330621065b710744c2ae1c4f9
8e0bd2c43766c1d7583f8c96f3f385f54a/diff",
                "MergedDir":
"/var/lib/docker/overlay2/e86e35e0dee3c219b2467ac7423b462623788400ba2f05e903af
47eab8279ef4/merged",
                "UpperDir":
"/var/lib/docker/overlay2/e86e35e0dee3c219b2467ac7423b462623788400ba2f05e903af
47eab8279ef4/diff",
                "WorkDir":
"/var/lib/docker/overlay2/e86e35e0dee3c219b2467ac7423b462623788400ba2f05e903af
47eab8279ef4/work"
            },
            "Name": "overlay2"
        },
        "Mounts": [],
        "Config": {
            "Hostname": "b366d6cfcb74",
            "Domainname": "",
            "User": "",
            "AttachStdin": true,
            "AttachStdout": true,
            "AttachStderr": true,
            "Tty": true,
            "OpenStdin": true,
            "StdinOnce": true,
            "Env": [
```

```
"PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
            ],
            "Cmd": [
                "sh"
            ],
            "Image": "alpine",
            "Volumes": null,
            "WorkingDir": "",
            "Entrypoint": null,
            "OnBuild": null,
            "Labels": {}
        },
        "NetworkSettings": {
            "Bridge": "",
            "SandboxID":
"304b557d89c291a6111621b868a74fb4032d18d63ecbec9640ab34f87a490676",
            "HairpinMode": false,
            "LinkLocalIPv6Address": "",
            "LinkLocalIPv6PrefixLen": 0,
            "Ports": {},
            "SandboxKey": "/var/run/docker/netns/304b557d89c2",
            "SecondaryIPAddresses": null,
            "SecondaryIPv6Addresses": null,
            "EndpointID":
"d639d01b06a33e2c74cfe30c3efe33abd8a3ad2a9bff13bfbbeaeea9c218758d",
            "Gateway": "172.17.0.1",
            "GlobalIPv6Address": "",
            "GlobalIPv6PrefixLen": 0,
            "IPAddress": "172.17.0.5",
            "IPPrefixLen": 16,
            "IPv6Gateway": "",
            "MacAddress": "02:42:ac:11:00:05",
            "Networks": {
                "bridge": {
                    "IPAMConfig": null,
                    "Links": null,
                    "Aliases": null,
                    "NetworkID":
"f673d7cb958cbb3a60ea7174accde412d3bdf6a3f0509c70286ae9fda9909641",
                    "EndpointID":
"d639d01b06a33e2c74cfe30c3efe33abd8a3ad2a9bff13bfbbeaeea9c218758d",
                    "Gateway": "172.17.0.1",
                    "IPAddress": "172.17.0.5",
                    "IPPrefixLen": 16,
                    "IPv6Gateway": "",
                    "GlobalIPv6Address": "",
                    "GlobalIPv6PrefixLen": 0,
```

```
                    "MacAddress": "02:42:ac:11:00:05",
                    "DriverOpts": null
                }
            }
        }
    }
]
```

Restrict the host CPU(s) that are allowed to execute a container. We can set a single CPU, or a range of CPUs that are allowed to execute the container. In the previous full output of the inspect command, there were no such restrictions. In this exercise let's restrict the container to be executed only by CPU "0":

```
student@ubuntu:~$ sudo docker container run -d --name cpu-set
--cpuset-cpus="0" alpine top
0504b3bb7ddf9943ee207396fdce92b48adc5d37e6f78d2a0903ddd32985783f
```

Let's verify the new setting by inspecting the container:

```
student@ubuntu:~$ sudo docker container inspect cpu-set | grep -i cpuset
            "CpusetCpus": "0",
            "CpusetMems": "",
```

Now it is safe to remove the container:

```
student@ubuntu:~$ sudo docker container rm -f cpu-set
cpu-set
```

We can also set the amount of memory of a container. By default, from the previous full output of the inspect command, the value is set to "0". Let's reset that value:

```
student@ubuntu:~$ sudo docker container run -d --name memory --memory "200m"
alpine top
WARNING: Your kernel does not support swap limit capabilities or the cgroup is
not mounted. Memory limited without swap.
86ef309433bba0aafe73fed5dcf97f483243da70fd5bb4a45f2c959a6a41bdfa

student@ubuntu:~$ sudo docker container inspect memory | grep -i mem
        "Name": "/memory",
            "Memory": 209715200,
            "CpusetMems": "",
```

```
...
```

Now let's remove the container:

```
student@ubuntu:~$ sudo docker container rm -f memory
memory
```

Create a new process inside a running container, a feature very useful for debugging. Let's execute a new process inside a container by running a command that retrieves and lists the IP address of the container. As soon as the command finishes, the newly forked process also gets terminated:

```
student@ubuntu:~$ sudo docker container ls
CONTAINER ID        IMAGE                         COMMAND             CREATED
STATUS              PORTS               NAMES
b366d6cfcb74        alpine                        "sh"                14 hours
ago        Up 14 hours                              dreamy_golick
...

student@ubuntu:~$ sudo docker container exec dreamy_golick ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
33: eth0@if34: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue
state UP
    link/ether 02:42:ac:11:00:05 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.5/16 brd 172.17.255.255 scope global eth0
       valid_lft forever preferred_lft forever
```

Set the restart policy of a container. An always restart policy restarts the container every time it fails. An on-failure policy, however, allows us to control the number of restarts of the container as a result of several failures - set to 3 in the example below:

```
student@ubuntu:~$ sudo docker container run -d --restart=always --name
web-always nginx
04ba201f0c62e1b14f71dab71c00879e7886fd4592bf5e889f5fae5da63f6343

student@ubuntu:~$ sudo docker container run -d --restart=on-failure:3 --name
web-on-failure nginx
8fd91ca8c73355f13eca1ed92bc3d96e3b7f522034e33fc4f015cf4443e5c976
```

We can copy files between the host system and a running container. This example will overwrite the index.html file of the nginx webserver running in a container, and the verification step will include the display of the container IP and finally a curl command to display the new web page served by the webserve:

```
student@ubuntu:~$ echo Welcome to Container Fundamentals! > host-file

student@ubuntu:~$ sudo docker container cp host-file
web-on-failure:/usr/share/nginx/html/index.html

student@ubuntu:~$ sudo docker container inspect --format='{{range
.NetworkSettings.Networks}}{{.IPAddress}}{{end}}' web-on-failure
172.17.0.7

student@ubuntu:~$ curl 172.17.0.7
Welcome to Container Fundamentals!
```

Labeling a container:

```
student@ubuntu:~$ sudo docker container run -d --label env=dev nginx
82e9bd095d83455c7cf9aa0166c703afbc9a57887e213f58d9a2c1610ad87c04

student@ubuntu:~$ sudo docker container ls
CONTAINER ID        IMAGE                          COMMAND
CREATED             STATUS              PORTS                NAMES
82e9bd095d83        nginx                          "nginx -g 'daemon of…"   12
seconds ago      Up 11 seconds       80/tcp               dazzling_northcutt
8fd91ca8c733        nginx                          "nginx -g 'daemon of…"   40
minutes ago      Up 14 minutes       80/tcp               web-on-failure
04ba201f0c62        nginx                          "nginx -g 'daemon of…"   41
minutes ago      Up 41 minutes       80/tcp               web-always
b366d6cfcb74        alpine                         "sh"                     15
hours ago        Up 15 hours                              dreamy_golick
...
```

Filtering container lists. We can filter containers by specifying conditions, to control and limit the output only to the desired objects. In this example let's use the label created previously as a filter:

```
student@ubuntu:~$ sudo docker container ls --filter label=env=dev
CONTAINER ID        IMAGE               COMMAND                  CREATED
STATUS              PORTS               NAMES
82e9bd095d83        nginx               "nginx -g 'daemon of…"   6 minutes ago
Up 6 minutes        80/tcp              dazzling_northcutt
```

Remove/delete all (running and stopped) containers with one command:

```
student@ubuntu:~$ sudo docker container ls -q
82e9bd095d83
8fd91ca8c733
04ba201f0c62
b366d6cfcb74
9e09b2cf6ff3
b1f1f19b1735
124b392d4bf2

student@ubuntu:~$ sudo docker container rm -f `sudo docker container ls -q`
82e9bd095d83
8fd91ca8c733
04ba201f0c62
b366d6cfcb74
9e09b2cf6ff3
b1f1f19b1735
124b392d4bf2
```

Change the default executable that runs at container startup. If defined, a default executable runs when a container starts. A nginx container starts with /usr/sbin/nginx -g daemon off. We can change it by passing the another command and possibly arguments when running the container. Let's start a container from the nginx image, but with a running shell instead:

```
student@ubuntu:~$ sudo docker container run -it nginx sh
/ #
```

Privileged containers. In privileged mode, containers gain permissions to access devices on the host. By default, it is disabled and containers run in un-privileged mode. Let's demonstrate privileges by running two containers in un-privileged and privileged mode respectively, while attempting to change host network settings, that is to create a simple alias to a network device:

```
student@ubuntu:~$ sudo docker container run -it --net=host alpine sh
/ # ifconfig ens4:0 192.168.2.1 up
ifconfig: SIOCSIFADDR: Operation not permitted
/ #

student@ubuntu:~$ sudo docker container run -it --net=host --privileged alpine
sh
/ # ifconfig ens4:0 192.168.2.1 up
/ # ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1460 qdisc fq_codel state UP
qlen 1000
    link/ether 42:01:0a:80:00:05 brd ff:ff:ff:ff:ff:ff
    inet 10.128.0.5/32 scope global dynamic ens4
       valid_lft 1844sec preferred_lft 1844sec
    inet 192.168.2.1/24 brd 192.168.2.255 scope global ens4:0
       valid_lft forever preferred_lft forever
    inet6 fe80::4001:aff:fe80:5/64 scope link
       valid_lft forever preferred_lft forever
3: lxcbr0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state
DOWN qlen 1000
    link/ether 00:16:3e:00:00:00 brd ff:ff:ff:ff:ff:ff
    inet 10.0.3.1/24 scope global lxcbr0
       valid_lft forever preferred_lft forever
    inet6 fe80::216:3eff:fe00:0/64 scope link
       valid_lft forever preferred_lft forever
8: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state
DOWN
    link/ether 02:42:93:77:3b:cd brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:93ff:fe77:3bcd/64 scope link
       valid_lft forever preferred_lft forever
/ #
```

Remove all stopped containers:

```
student@ubuntu:~$ sudo docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES

student@ubuntu:~$ sudo docker container ls -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS                      PORTS               NAMES
3f639a5fceae        alpine              "sh"                6 minutes ago
Exited (0) 11 seconds ago                       gracious_moser
9a050a479a9d        alpine              "sh"                7 minutes ago
Exited (1) 6 minutes ago                        serene_chatelet
fd26346e8a95        alpine              "sh"                17 hours ago
Exited (0) 17 hours ago                         sweet_booth
32f7643ef139        busybox             "sh"                19 hours ago
Created                                         recursing_ganguly
8a6c65920f97        hello-world         "/hello"            3 days ago
Exited (0) 3 days ago                           nice_clarke

student@ubuntu:~$ sudo docker container prune
WARNING! This will remove all stopped containers.
Are you sure you want to continue? [y/N] y
Deleted Containers:
3f639a5fceaeb8e58cb291b18985071db67920c1e21699440f0db0e4b9c9e14e
9a050a479a9db355e9b6759811b7248906618ef647f26be5a60572187747cc53
fd26346e8a9515d12d91e8127b4639e07f00f3b62ea03511ef264e263f2139b2
32f7643ef139852ef4dc8c20db526e8cd7eb12c499af51c3dbeae8ccd56e74ad
8a6c65920f9789e0e83a7185c735b0f94e7b18a45e2cf626a1a950647f17afb3

Total reclaimed space: 118B

student@ubuntu:~$ sudo docker container ls -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
student@ubuntu:~$
```

Lab 6.3 - Container and Pod Operations with rkt

Let's fetch an ACI image from the registry to the local repository, so we can explore application containers and pods operations supported by rkt:

```
student@ubuntu:~$ sudo rkt fetch coreos.com/etcd:v3.1.7
image: keys already exist for prefix "coreos.com/etcd", not fetching again
Downloading signature: [=======================================] 490 B/490 B
Downloading ACI: [=============================================] 9.47 MB/9.47
MB
image: signature verified:
  CoreOS Application Signing Key <security@coreos.com>
sha512-e7a54697d04ddae899ed1bfd263235cb
```

Fetch a Docker image. One of rkt's advantages is its flexibility to retrieve images from its own registry and from Docker Hub registry as well. Unfortunately, it is not able to check the integrity of Docker images:

```
student@ubuntu:~$ sudo rkt fetch --insecure-options=image docker://nginx
Downloading sha256:a616aa3b0bf [==============================]     203 B / 203
B
Downloading sha256:68ced04f60a [==============================] 27.1 MB / 27.1
MB
Downloading sha256:28252775b29 [==============================] 23.9 MB / 23.9
MB
sha512-d00ac58834f14e71aadbc781a8d9bd30
```

List images cached in the local repository:

```
student@ubuntu:~$ sudo rkt image list
ID                  NAME                                    SIZE   IMPORT TIME LAST
USED
```

```
sha512-e7a54697d04d     coreos.com/etcd:v3.1.7        58MiB 8 minutes ago
8 minutes ago
sha512-d00ac58834f1     registry-1.docker.io/library/nginx:latest 244MiB59
seconds ago 56 seconds ago
sha512-402c24acf901     coreos.com/etcd:v3.1.1        58MiB 10 seconds ago
9 seconds ago
```

Run an ACI from the local repository in a pod. Rkt can reference an ACI by name, hash, or URL when it attempts to run the application container in a pod. Let's run an ACI and a Docker image from the local repository. The pods will run in the foreground locking our terminals, for additional operations we need to open other terminals:

```
student@ubuntu:~$ sudo rkt run coreos.com/etcd:v3.1.7

student@ubuntu:~$ sudo rkt run registry-1.docker.io/library/nginx:latest
```

List pods in default format and in extended/full format:

```
student@ubuntu:~$ sudo rkt list
UUID        APP    IMAGE NAME                          STATE CREATED
STARTED           NETWORKS
9623b1ce    nginx registry-1.docker.io/library/nginx:latest running     9
minutes ago 9 minutes ago     default:ip4=172.16.28.2
c800baa5    etcd  coreos.com/etcd:v3.1.7                   running     3
minutes ago 3 minutes ago     default:ip4=172.16.28.3

student@ubuntu:~$ sudo rkt list --full
UUID                          APP    IMAGE NAME                             IMAGE
ID           STATE CREATED                                STARTED
             NETWORKS
9623b1ce-787a-421e-8390-52342a2f0751nginx
registry-1.docker.io/library/nginx:latest sha512-d00ac58834f1     running
2020-03-13 04:17:39.081 +0000 UTC   2020-03-13 04:17:39.205 +0000 UTC
default:ip4=172.16.28.2
c800baa5-42ae-4da8-a087-5526a5cd9a92etcd  coreos.com/etcd:v3.1.7
      sha512-e7a54697d04d     running     2020-03-13 04:23:31.623 +0000 UTC
2020-03-13 04:23:31.755 +0000 UTC   default:ip4=172.16.28.3
```

Display the status of individual pods, providing the pod UUID:

```
student@ubuntu:~$ sudo rkt status 9623b1ce
state=running
created=2020-03-13 04:17:39.081 +0000 UTC
started=2020-03-13 04:17:39.205 +0000 UTC
networks=default:ip4=172.16.28.2
pid=18396
exited=false

student@ubuntu:~$ sudo rkt status c800baa5
state=running
created=2020-03-13 04:23:31.623 +0000 UTC
started=2020-03-13 04:23:31.755 +0000 UTC
networks=default:ip4=172.16.28.3
pid=18572
exited=false
```

Run multiple application containers in the same pod, while assigning a custom name to the busybox app. Observe the distinct methods of referencing the two docker images. Both methods are acceptable to reference docker images:

```
student@ubuntu:~$ sudo rkt run docker://nginx
registry-1.docker.io/library/busybox:latest --name=bzbx
```

Enter an application running in a pod - that is opening a shell into the application. For single-app pods, the enter command does not require additional parameters, only the pod UUID. However, for multi-app pods the name of the app also has to be specified with the enter command. Also, specifying the command is recommended in both cases (we specify sh):

```
student@ubuntu:~$ sudo rkt enter --app=nginx 791ed80c sh
# ls /usr/share/nginx/html
50x.html  index.html
#
```

Stopping a pod requires the pod UUID:

```
student@ubuntu:~$ sudo rkt stop 569f5fb3
"569f5fb3-eb3a-42d9-8faa-49f144146792"
```

Removing a stopped pod also requires the pod UUID:

```
student@ubuntu:~$ sudo rkt rm 569f5fb3
"569f5fb3-eb3a-42d9-8faa-49f144146792"
```

Garbage Collection is a neat feature of rkt that may run periodically and on its first pass moves stopped pods to the garbage, while on the second pass cleans up those pods:

```
student@ubuntu:~$ sudo rkt gc --grace-period=5m0s
gc: moving pod "9623b1ce-787a-421e-8390-52342a2f0751" to garbage
gc: moving pod "c800baa5-42ae-4da8-a087-5526a5cd9a92" to garbage
gc: moving failed prepare "069205c9-ace7-4b9d-8ab8-256a7acaae82" to garbage
gc: pod "9623b1ce-787a-421e-8390-52342a2f0751" not removed: still within grace
period (5m0s)
gc: pod "c800baa5-42ae-4da8-a087-5526a5cd9a92" not removed: still within grace
period (5m0s)
Garbage collecting pod "069205c9-ace7-4b9d-8ab8-256a7acaae82"

student@ubuntu:~$ sudo rkt gc --grace-period=5m0s
gc: pod "9623b1ce-787a-421e-8390-52342a2f0751" not removed: still within grace
period (5m0s)
gc: pod "c800baa5-42ae-4da8-a087-5526a5cd9a92" not removed: still within grace
period (5m0s)
```

Cat-manifest is another neat feature, useful for troubleshooting. It allows for the configuration manifest of a pod to be exported. The manifest below is for the pod running both the nginx and the busybox applications:

```
student@ubuntu:~$ sudo rkt cat-manifest 791ed80c
{
      "acVersion": "1.30.0",
      "acKind": "PodManifest",
      "apps": [
            {
                  "name": "nginx",
                  "image": {
                        "name": "registry-1.docker.io/library/nginx",
                        "id":
"sha512-d00ac58834f14e71aadbc781a8d9bd307c0bdbecbfb64b33af0fb9e0300ef586",
```

```
                        "labels": [
                              {
                                    "name": "arch",
                                    "value": "amd64"
                              },
                              {
                                    "name": "os",
                                    "value": "linux"
                              },
                              {
                                    "name": "version",
                                    "value": "latest"
                              }
                        ]
                  },
                  "app": {
                        "exec": [
                              "nginx",
                              "-g",
                              "daemon off;"
                        ],
                        "user": "0",
                        "group": "0",
                        "environment": [
                              {
                                    "name": "PATH",
                                    "value":
"/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
                              },
                              {
                                    "name": "NGINX_VERSION",
                                    "value": "1.17.9"
                              },
                              {
                                    "name": "NJS_VERSION",
                                    "value": "0.3.9"
                              },
                              {
                                    "name": "PKG_RELEASE",
                                    "value": "1~buster"
                              }
                        ],
                        "ports": [
                              {
                                    "name": "80-tcp",
                                    "protocol": "tcp",
                                    "port": 80,
                                    "count": 1,
```

```
                                    "socketActivated": false
                              }
                        ]
                  }
            },
            {
                  "name": "bzbx",
                  "image": {
                        "name": "registry-1.docker.io/library/busybox",
                        "id":
"sha512-a749dac4b4e362dd3f5de56241b499da6a9a02c749f299e496ee35d6437dd770",
                        "labels": [
                              {
                                    "name": "arch",
                                    "value": "amd64"
                              },
                              {
                                    "name": "os",
                                    "value": "linux"
                              },
                              {
                                    "name": "version",
                                    "value": "latest"
                              }
                        ]
                  },
                  "app": {
                        "exec": [
                              "sh"
                        ],
                        "user": "0",
                        "group": "0",
                        "environment": [
                              {
                                    "name": "PATH",
                                    "value":
"/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
                              }
                        ]
                  }
            }
      ],
      "volumes": null,
      "isolators": null,
      "annotations": [
            {
                  "name": "coreos.com/rkt/stage1/mutable",
                  "value": "false"
```

```
            }
        ],
        "ports": []
}
```

Lab 6.4 - Container Operations with Podman

Before exploring some of the container operations supported by Podman, let's search the registries for an nginx image and pull it to the local repository:

```
student@ubuntu:~$ podman search --filter=is-official nginx
INDEX      NAME                      DESCRIPTION               STARS  OFFICIAL
AUTOMATED
Docker.io  docker.io/library/nginx  Official build of Nginx.  12795  [OK]

student@ubuntu:~$ podman image pull docker.io/library/nginx
Trying to pull docker.io/library/nginx...
Getting image source signatures
Copying blob 28252775b295 done
Copying blob a616aa3b0bf2 done
Copying blob 68ced04f60ab done
Copying config 6678c7c2e5 [======================================] 6.5KiB /
6.5KiB
Writing manifest to image destination
Storing signatures
6678c7c2e56c970388f8d5a398aa30f2ab60e85f20165e101053c3d3a11e6663

student@ubuntu:~$ podman image list
REPOSITORY                TAG      IMAGE ID       CREATED      SIZE
docker.io/library/nginx   latest   6678c7c2e56c   7 days ago   131 MB
```

Create a container. Once an image is available in the local repository, podman can create a container. This step only creates the container but it does not start it. A container is started with a separate command.

```
student@ubuntu:~$ podman container create nginx
ae0a5dc2d8d1620c39f1fdccf6b880a25df59e3734dd46f9ce3cc9930277147b
```

Listing all containers, to include non-running containers as well:

```
student@ubuntu:~$ podman container list -a
CONTAINER ID  IMAGE                            COMMAND               CREATED
STATUS    PORTS  NAMES
ae0a5dc2d8d1  docker.io/library/nginx:latest  nginx -g daemon o...  23 seconds
ago  Created         reverent_euler
```

Starting a container:

```
student@ubuntu:~$ podman container start reverent_euler
reverent_euler
```

List only running containers:

```
student@ubuntu:~$ podman container list
CONTAINER ID  IMAGE                            COMMAND               CREATED
STATUS            PORTS  NAMES
ae0a5dc2d8d1  docker.io/library/nginx:latest  nginx -g daemon o...  3 minutes
ago  Up 7 seconds ago         reverent_euler
```

Restarting a running container, performs the sequence of a stop and a start operations. However, the container ID does not change:

```
student@ubuntu:~$ podman container restart reverent_euler
ae0a5dc2d8d1620c39f1fdccf6b880a25df59e3734dd46f9ce3cc9930277147b

student@ubuntu:~$ podman container list
CONTAINER ID  IMAGE                            COMMAND               CREATED
STATUS            PORTS  NAMES
ae0a5dc2d8d1  docker.io/library/nginx:latest  nginx -g daemon o...  5 minutes
ago  Up 6 seconds ago         reverent_euler
```

Stopping a container is an easy task:

```
student@ubuntu:~$ podman container stop reverent_euler
ae0a5dc2d8d1620c39f1fdccf6b880a25df59e3734dd46f9ce3cc9930277147b
```

Removal of a stopped container is also easy. However, the removal of a running or a paused container requires the force option:

```
student@ubuntu:~$ podman container rm reverent_euler
ae0a5dc2d8d1620c39f1fdccf6b880a25df59e3734dd46f9ce3cc9930277147b
```

Podman's prune feature automates the removal of stopped containers based on a configurable filter:

```
student@ubuntu:~$ podman container list -a
CONTAINER ID  IMAGE                            COMMAND               CREATED
STATUS                     PORTS  NAMES
186c1702848e  docker.io/library/nginx:latest  nginx -g daemon o...  2 minutes
ago  Exited (0) 5 seconds ago         suspicious_hertz
f978690abc85  docker.io/library/nginx:latest  nginx -g daemon o...  2 hours
ago    Up 2 hours ago                  mystifying_cori

student@ubuntu:~$ podman container prune -f
186c1702848e98486625a72545e8eb7243f3e44cdfa806f1220af82f23fd1795

student@ubuntu:~$ podman container list -a
CONTAINER ID  IMAGE                            COMMAND               CREATED
STATUS          PORTS  NAMES
f978690abc85  docker.io/library/nginx:latest  nginx -g daemon o...  2 hours
ago  Up 2 hours ago         mystifying_cori
```

The top utility of podman is a helpful tool for displaying processes running in a container, together with their CPU utilization:

```
student@ubuntu:~$ podman container top mystifying_cori
USER    PID   PPID   %CPU    ELAPSED             TTY   TIME   COMMAND
root    1     0      0.000   1h0m38.778847959s   ?     0s     nginx: master
process nginx -g daemon off;
nginx   6     1      0.000   1h0m38.778919673s   ?     0s     nginx: worker
process
```

Exec allows for commands to be run inside a running container. By running a shell in the container allows users to directly interact with the container environment. We can also run installers, validators, display the environment or a set of permissions from the container:

```
student@ubuntu:~$ podman container exec -ti  mystifying_cori /bin/sh
# ls /usr/share/nginx/html
50x.html  index.html
# exit
```

We can copy content between the host system and a container running on the host:

```
student@ubuntu:~$ echo Welcome to Container Fundamentals! > host-file

student@ubuntu:~$ podman cp host-file
mystifying_cori:/usr/share/nginx/html/index.html

student@ubuntu:~$ podman container exec -ti  mystifying_cori /bin/sh
# cat /usr/share/nginx/html/index.html
Welcome to Container Fundamentals!
# exit
```

We can inspect a container, to output its entire configuration:

```
student@ubuntu:~$ podman container inspect mystifying_cori
[
    {
        "Id":
"f978690abc8517d00d6dc540820e25f98d88127079a29168ebe87ec1d9cd485e",
        "Created": "2020-03-13T06:27:00.112110101Z",
        "Path": "nginx",
        "Args": [
            "-g",
            "daemon off;"
        ],
        "State": {
            "OciVersion": "1.0.1-dev",
            "Status": "running",
            "Running": true,
            "Paused": false,
            "Restarting": false,
            "OOMKilled": false,
            "Dead": false,
            "Pid": 22340,
            "ConmonPid": 22318,
            "ExitCode": 0,
            "Error": "",
            "StartedAt": "2020-03-13T06:41:14.683981673Z",
            "FinishedAt": "2020-03-13T06:41:00.044570654Z",
            "Healthcheck": {
                "Status": "",
                "FailingStreak": 0,
                "Log": null
            }
        },
        "Image":
"6678c7c2e56c970388f8d5a398aa30f2ab60e85f20165e101053c3d3a11e6663",
        "ImageName": "docker.io/library/nginx:latest",
        "Rootfs": "",
        "Pod": "",
        "ResolvConfPath":
"/run/user/1001/vfs-containers/f978690abc8517d00d6dc540820e25f98d88127079a2916
8ebe87ec1d9cd485e/userdata/resolv.conf",
        "HostnamePath":
"/run/user/1001/vfs-containers/f978690abc8517d00d6dc540820e25f98d88127079a2916
8ebe87ec1d9cd485e/userdata/hostname",
        "HostsPath":
"/run/user/1001/vfs-containers/f978690abc8517d00d6dc540820e25f98d88127079a2916
8ebe87ec1d9cd485e/userdata/hosts",
```

```
        "StaticDir":
"/home/student/.local/share/containers/storage/vfs-containers/f978690abc8517d0
0d6dc540820e25f98d88127079a29168ebe87ec1d9cd485e/userdata",
        "OCIConfigPath":
"/home/student/.local/share/containers/storage/vfs-containers/f978690abc8517d0
0d6dc540820e25f98d88127079a29168ebe87ec1d9cd485e/userdata/config.json",
        "OCIRuntime": "runc",
        "LogPath":
"/home/student/.local/share/containers/storage/vfs-containers/f978690abc8517d0
0d6dc540820e25f98d88127079a29168ebe87ec1d9cd485e/userdata/ctr.log",
        "LogTag": "",
        "ConmonPidFile":
"/run/user/1001/vfs-containers/f978690abc8517d00d6dc540820e25f98d88127079a2916
8ebe87ec1d9cd485e/userdata/conmon.pid",
        "Name": "mystifying_cori",
        "RestartCount": 0,
        "Driver": "vfs",
        "MountLabel": "",
        "ProcessLabel": "",
        "AppArmorProfile": "",
        "EffectiveCaps": [
            "CAP_CHOWN",
            "CAP_DAC_OVERRIDE",
            "CAP_FSETID",
            "CAP_FOWNER",
            "CAP_MKNOD",
            "CAP_NET_RAW",
            "CAP_SETGID",
            "CAP_SETUID",
            "CAP_SETFCAP",
            "CAP_SETPCAP",
            "CAP_NET_BIND_SERVICE",
            "CAP_SYS_CHROOT",
            "CAP_KILL",
            "CAP_AUDIT_WRITE"
        ],
        "BoundingCaps": [
            "CAP_CHOWN",
            "CAP_DAC_OVERRIDE",
            "CAP_FSETID",
            "CAP_FOWNER",
            "CAP_MKNOD",
            "CAP_NET_RAW",
            "CAP_SETGID",
            "CAP_SETUID",
            "CAP_SETFCAP",
            "CAP_SETPCAP",
            "CAP_NET_BIND_SERVICE",
```

```
        "CAP_SYS_CHROOT",
        "CAP_KILL",
        "CAP_AUDIT_WRITE"
    ],
    "ExecIDs": [],
    "GraphDriver": {
        "Name": "vfs",
        "Data": null
    },
    "Mounts": [],
    "Dependencies": [],
    "NetworkSettings": {
        "Bridge": "",
        "SandboxID": "",
        "HairpinMode": false,
        "LinkLocalIPv6Address": "",
        "LinkLocalIPv6PrefixLen": 0,
        "Ports": [],
        "SandboxKey": "",
        "SecondaryIPAddresses": null,
        "SecondaryIPv6Addresses": null,
        "EndpointID": "",
        "Gateway": "",
        "GlobalIPv6Address": "",
        "GlobalIPv6PrefixLen": 0,
        "IPAddress": "",
        "IPPrefixLen": 0,
        "IPv6Gateway": "",
        "MacAddress": ""
    },
    "ExitCommand": [
        "/usr/bin/podman",
        "--root",
        "/home/student/.local/share/containers/storage",
        "--runroot",
        "/run/user/1001",
        "--log-level",
        "error",
        "--cgroup-manager",
        "cgroupfs",
        "--tmpdir",
        "/run/user/1001/libpod/tmp",
        "--runtime",
        "runc",
        "--storage-driver",
        "vfs",
        "--events-backend",
        "file",
```

```
            "container",
            "cleanup",
            "f978690abc8517d00d6dc540820e25f98d88127079a29168ebe87ec1d9cd485e"
        ],
        "Namespace": "",
        "IsInfra": false,
        "Config": {
            "Hostname": "f978690abc85",
            "Domainname": "",
            "User": "",
            "AttachStdin": false,
            "AttachStdout": false,
            "AttachStderr": false,
            "Tty": false,
            "OpenStdin": false,
            "StdinOnce": false,
            "Env": [
"PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
                "TERM=xterm",
                "container=podman",
                "NGINX_VERSION=1.17.9",
                "NJS_VERSION=0.3.9",
                "PKG_RELEASE=1~buster",
                "HOSTNAME=f978690abc85",
                "HOME=/root"
            ],
            "Cmd": [
                "nginx",
                "-g",
                "daemon off;"
            ],
            "Image": "docker.io/library/nginx:latest",
            "Volumes": null,
            "WorkingDir": "/",
            "Entrypoint": "",
            "OnBuild": null,
            "Labels": {
                "maintainer": "NGINX Docker Maintainers
<docker-maint@nginx.com>"
            },
            "Annotations": {
                "io.container.manager": "libpod",
                "io.kubernetes.cri-o.Created":
"2020-03-13T06:27:00.112110101Z",
                "io.kubernetes.cri-o.TTY": "false",
                "io.podman.annotations.autoremove": "FALSE",
                "io.podman.annotations.init": "FALSE",
```

```
            "io.podman.annotations.privileged": "FALSE",
            "io.podman.annotations.publish-all": "FALSE",
            "org.opencontainers.image.stopSignal": "15"
        },
        "StopSignal": 15,
        "CreateCommand": [
            "podman",
            "container",
            "create",
            "nginx"
        ]
    },
    "HostConfig": {
        "Binds": [],
        "ContainerIDFile": "",
        "LogConfig": {
            "Type": "k8s-file",
            "Config": null
        },
        "NetworkMode": "default",
        "PortBindings": {},
        "RestartPolicy": {
            "Name": "",
            "MaximumRetryCount": 0
        },
        "AutoRemove": false,
        "VolumeDriver": "",
        "VolumesFrom": null,
        "CapAdd": [],
        "CapDrop": [],
        "Dns": [],
        "DnsOptions": [],
        "DnsSearch": [],
        "ExtraHosts": [],
        "GroupAdd": [],
        "IpcMode": "",
        "Cgroup": "",
        "Cgroups": "default",
        "Links": null,
        "OomScoreAdj": 0,
        "PidMode": "",
        "Privileged": false,
        "PublishAllPorts": false,
        "ReadonlyRootfs": false,
        "SecurityOpt": [],
        "Tmpfs": {},
        "UTSMode": "",
        "UsernsMode": "",
```

```
            "ShmSize": 65536000,
            "Runtime": "oci",
            "ConsoleSize": [
                0,
                0
            ],
            "Isolation": "",
            "CpuShares": 1024,
            "Memory": 0,
            "NanoCpus": 0,
            "CgroupParent": "",
            "BlkioWeight": 0,
            "BlkioWeightDevice": null,
            "BlkioDeviceReadBps": null,
            "BlkioDeviceWriteBps": null,
            "BlkioDeviceReadIOps": null,
            "BlkioDeviceWriteIOps": null,
            "CpuPeriod": 0,
            "CpuQuota": 0,
            "CpuRealtimePeriod": 0,
            "CpuRealtimeRuntime": 0,
            "CpusetCpus": "",
            "CpusetMems": "",
            "Devices": [],
            "DiskQuota": 0,
            "KernelMemory": 0,
            "MemoryReservation": 0,
            "MemorySwap": 0,
            "MemorySwappiness": 0,
            "OomKillDisable": false,
            "PidsLimit": 0,
            "Ulimits": [
                {
                    "Name": "RLIMIT_NOFILE",
                    "Soft": 1024,
                    "Hard": 1024
                }
            ],
            "CpuCount": 0,
            "CpuPercent": 0,
            "IOMaximumIOps": 0,
            "IOMaximumBandwidth": 0
        }
    }
]
```

Lab 7.1 - Building Docker Images

Build an Image from a running Container

Let's run a container first, and apply some changes by saving data on its filesystem. After verifying the changes, the output of the date command, we can then create a new image out of the modified container:

```
student@ubuntu:~$ sudo docker container run -ti --name myalpine alpine sh
Unable to find image 'alpine:latest' locally
latest: Pulling from library/alpine
c9b1b535fdd9: Pull complete
Digest:
sha256:ab00606a42621fb68f2ed6ad3c88be54397f981a7b70a79db3d1172b11c4367d
Status: Downloaded newer image for alpine:latest
/ # date > /data
/ # cat /data
Wed Mar 11 07:24:31 UTC 2020
/ # <Ctrl p + Ctrl q>

student@ubuntu:~$ sudo docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
124b392d4bf2        alpine              "sh"                3 minutes ago
Up 2 minutes                            myalpine

student@ubuntu:~$ sudo docker container diff myalpine
A /data
C /root
A /root/.ash_history

student@ubuntu:~$ sudo docker container commit myalpine
lfstudent/alpine:training
sha256:14299dae172ddf26b218e8ddad5e764ef44df7a585175c1318aa9cd103aeb16e
```

```
student@ubuntu:~$ sudo docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
lfstudent/alpine    training            14299dae172d        23 seconds ago
5.59MB
nginx               latest              6678c7c2e56c        6 days ago
127MB
alpine              latest              e7d92cdc71fe        7 weeks ago
5.59MB
hello-world         latest              fce289e99eb9        14 months ago
1.84kB
```

For verification, we may now create a container out of the new image, and verify the existence of the data produced earlier by the date command:

```
student@ubuntu:~$ sudo docker container run -ti lfstudent/alpine:training
/ # cat /data
Wed Mar 11 07:24:31 UTC 2020
/ # <Ctrl p + Ctrl q>

student@ubuntu:~$ sudo docker container ls
CONTAINER ID        IMAGE                          COMMAND             CREATED
STATUS              PORTS               NAMES
b1f1f19b1735        lfstudent/alpine:training   "sh"                3 minutes
ago       Up 3 minutes                              crazy_clarke
124b392d4bf2        alpine                         "sh"                45 minutes
ago      Up 45 minutes                              myalpine
```

Export a Container filesystem to a tar archive

The filesystem of a running container can be exported as a tar archive. Subsequently, a new image can be created from the tar archive.

```
student@ubuntu:~$ sudo docker container ls
CONTAINER ID        IMAGE                          COMMAND             CREATED
STATUS              PORTS               NAMES
b1f1f19b1735        lfstudent/alpine:training   "sh"                3 minutes
ago       Up 3 minutes                              crazy_clarke
124b392d4bf2        alpine                         "sh"                45 minutes
ago      Up 45 minutes                              myalpine

student@ubuntu:~$ sudo docker container export b1f1f19b1735 >
lfstudent_alpine.tar
```

```
student@ubuntu:~$ ls lfstudent_alpine.tar
lfstudent_alpine.tar
```

Import filesystem from a tar archive into an Image

From an existing tar file, which archives a container filesystem, we can create a new image:

```
student@ubuntu:~$ ls lfstudent_alpine.tar
lfstudent_alpine.tar

student@ubuntu:~$ sudo docker image import lfstudent_alpine.tar
lfstudent/alpine:latest
sha256:465fb0347d9d3da56486bec4da8047b93e5a4693d44f94e27d57af99c037808d

student@ubuntu:~$ sudo docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
lfstudent/alpine    latest              465fb0347d9d        35 seconds ago
5.59MB
lfstudent/alpine    training            14299dae172d        About an hour ago
5.59MB
nginx               latest              6678c7c2e56c        6 days ago
127MB
alpine              latest              e7d92cdc71fe        7 weeks ago
5.59MB
hello-world         latest              fce289e99eb9        14 months ago
1.84kB
```

As a verification step we may run a container out of the new image, and verify the existence of the data produced earlier by the date command:

```
student@ubuntu:~$ sudo docker container run -ti lfstudent/alpine:latest sh
/ # cat /data
Wed Mar 11 07:24:31 UTC 2020
/ # <Ctrl p + Ctrl q>

student@ubuntu:~$ sudo docker container ls
CONTAINER ID        IMAGE                          COMMAND             CREATED
STATUS              PORTS               NAMES
9e09b2cf6ff3        lfstudent/alpine:latest     "sh"                2 minutes
ago       Up 2 minutes                              zen_nightingale
```

```
b1f1f19b1735        lfstudent/alpine:training   "sh"                About an
hour ago   Up About an hour                       crazy_clarke
124b392d4bf2        alpine                      "sh"                2 hours
ago         Up 2 hours                          myalpine
```

Push an image to Docker Hub

Assuming the existence of a **lfstudent** account on Docker Hub, a user is required to login from the CLI before attempting to push a container image to the registry:

```
student@ubuntu:~$ sudo docker login

Login with your Docker ID to push and pull images from Docker Hub. If you
don't
have a Docker ID, head over to https://hub.docker.com to create one.
Username: lfstudent
Password: ********
Login Succeeded
```

Once logged in to Docker Hub, a user may push an image into the registry to be shared by other users:

```
student@ubuntu:~$ sudo docker image push lfstudent/alpine:training
The push refers to a repository [docker.io/lfstudent/alpine]
724d404d96ef: Pushed
60ab55d3379d: Mounted from library/alpine
training: digest:
sha256:fcc29a8a772bed232fc3026f9ea5df745e57ea4c99b2306f997036510d4be0f9 size:
735
```

Push an image to a private registry

```
student@ubuntu:~$ sudo docker image push myregistry.com:5000/alpine:training
```

Remove unused cached images from local repository

```
student@ubuntu:~$ sudo docker image prune
WARNING! This will remove all dangling images.
Are you sure you want to continue? [y/N]
y
Deleted Images:
deleted:
sha256:74688f28366cd966f203b54f175de233846e7b720eb2e0d09fae06a26a939779
Total reclaimed space: 0 B
```

Remove one or more cached images from the local repository

```
student@ubuntu:~$ sudo docker image rm -f alpine:latest nginx:latest
Untagged: alpine:latest
Untagged:
alpine@sha256:dfbd4a3a8ebca874ebd2474f044a0b33600d4523d03b0df76e5c5986cb02d7e8
Untagged: nginx:latest
Untagged:
nginx@sha256:f2d384a6ca8ada733df555be3edc427f2e5f285ebf468aae940843de8cf74645
Deleted:
sha256:cc1b614067128cd2f5cdafb258b0a4dd25760f14562bcce516c13f760c3b79c4
...
```

Lab 7.2 - Building a Docker Image with Dockerfile

Another method to create and share a Docker containerized application is through a Dockerfile. For a while, this method was Docker specific. However, other runtimes and image building tools have adopted this method to build and distribute images. With Dockerfile, instead of sharing the container image through a registry, we share a script with steps to create the image. The Dockerfile represents a reproducible method allowing the creation of identical images on any platform supporting Docker.

The following is a redacted example of Dockerfile of the latest (as of the time of this writing) nginx container image from Docker Hub:

```
FROM debian:buster-slim

LABEL maintainer="NGINX Docker Maintainers <docker-maint@nginx.com>"

ENV NGINX_VERSION   1.17.9
ENV NJS_VERSION     0.3.9
ENV PKG_RELEASE     1~buster

RUN set -x \
# create nginx user/group first, to be consistent throughout docker variants
    && addgroup --system --gid 101 nginx \
    && adduser --system --disabled-login --ingroup nginx --no-create-home
--home /nonexistent --gecos "nginx user" --shell /bin/false --uid 101 nginx \
    && apt-get update \
    && apt-get install --no-install-recommends --no-install-suggests -y gnupg1
...

# forward request and error logs to docker log collector
RUN ln -sf /dev/stdout /var/log/nginx/access.log \
    && ln -sf /dev/stderr /var/log/nginx/error.log

EXPOSE 80

STOPSIGNAL SIGTERM
```

```
CMD ["nginx", "-g", "daemon off;"]
```

The file includes reserved keywords such as FROM, LABEL, ENV, RUN, EXPOSE, STOPSIGNAL, and CMD that are instructions followed by sets of arguments. These instructions are read by the Docker daemon when the docker build command is issued from the Docker client CLI, to build the container images as specified by the Dockerfile. By default, the build process looks for a file called Dockerfile inside the context folder. We can also use custom configuration files as long as the build process refers to them via the -f option.

Let's attempt to create our own Dockerfile in a custom application subdirectory, which would be treated as the context of the build:

```
student@ubuntu:~$ mkdir myapp
student@ubuntu:~$ cd myapp/
student@ubuntu:~/myapp$ vim Dockerfile
student@ubuntu:~/myapp$ cat Dockerfile
FROM alpine
RUN date > data
```

Let's build the image, based on the Dockerfile we had just created:

```
student@ubuntu:~/myapp$ sudo docker image build -t lfstudent/alpine:dockerfile .
Sending build context to Docker daemon  2.048kB
Step 1/2 : FROM alpine
 ---> e7d92cdc71fe
Step 2/2 : RUN date > data
 ---> Running in 5d9bb3a2a2e1
Removing intermediate container 5d9bb3a2a2e1
 ---> ff36e0b37ced
Successfully built ff36e0b37ced
Successfully tagged lfstudent/alpine:dockerfile
```

This build command instructs the docker daemon to use the current directory as context, and use the Dockerfile found at the top of the context, while tagging the image with the provided name and tag. The context is archived into a tarball and sent to the Docker daemon running on the Docker host (which could be local or remote). Our Dockerfile includes two instructions, each corresponding to a step in the build process. Step 1 instructs the daemon to use the alpine container image as a base image, which in subsequent step(s) is customized to alter its behavior. Step 2 instructs the daemon to run a command that alters the base image filesystem. Step 2 runs the date command and saves some data onto the writable layer of the filesystem. At the end of the build process, once both steps 1 and 2 have completed, the newly built image is a modified alpine image, altered with some additional data.

```
student@ubuntu:~/myapp$ sudo docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
```

```
lfstudent/alpine  dockerfile    ff36e0b37ced      11 minutes ago      5.59MB
```

The same build could have been achieved by running the build command with a reference to the myapp directory which includes the Dockerfile:

```
student@ubuntu:~$ pwd
/home/student
student@ubuntu:~$ sudo docker image build -t lfstudent/alpine:dockerfile myapp
```

Image caching and build times

By default, Docker caches prior build steps to achieve faster future builds from the same base image. Let's output the time required by an initial build:

```
student@ubuntu:~/myapp$ time sudo docker image build -t lfstudent/nginx:dockerfile .
Sending build context to Docker daemon   7.68kB
Step 1/10 : FROM debian:buster-slim
 ---> 2f14a0fb67b9
Step 2/10 : LABEL maintainer="NGINX Docker Maintainers
<docker-maint@nginx.com>"
 ---> Running in b80ef5cf315a
...
Removing intermediate container a56f2fd1e116
 ---> 31c7447f26b9
Successfully built 31c7447f26b9
Successfully tagged lfstudent/nginx:dockerfile

real  0m34.284s
user  0m0.068s
sys   0m0.085s
```

While the initial build took a little over 34 seconds, let's attempt another build from the same Dockerfile:

```
student@ubuntu:~/myapp$ time sudo docker image build -t lfstudent/nginx:cached .
Sending build context to Docker daemon   7.68kB
Step 1/10 : FROM debian:buster-slim
 ---> 2f14a0fb67b9
Step 2/10 : LABEL maintainer="NGINX Docker Maintainers
<docker-maint@nginx.com>"
 ---> Using cache
 ---> 49391ba47654
Step 3/10 : ENV NGINX_VERSION   1.17.9
```

```
 ---> Using cache
 ---> c6907874ade0
...
Step 8/10 : EXPOSE 80
 ---> Using cache
 ---> 59288a1a4ca7
Step 9/10 : STOPSIGNAL SIGTERM
 ---> Using cache
 ---> 153468fc22b0
Step 10/10 : CMD ["nginx", "-g", "daemon off;"]
 ---> Using cache
 ---> 31c7447f26b9
Successfully built 31c7447f26b9
Successfully tagged lfstudent/nginx:cached

real  0m0.200s
user  0m0.035s
sys   0m0.038s
```

The second build took 0.2 second, because every single build step was referenced from the cache. Let's modify one of the steps in the Dockerfile, and attempt another build. Edit the Dockerfile and add port number 443 to the EXPORT instruction:

```
student@ubuntu:~/myapp$ vim Dockerfile

...
# forward request and error logs to docker log collector
RUN ln -sf /dev/stdout /var/log/nginx/access.log \
    && ln -sf /dev/stderr /var/log/nginx/error.log

EXPOSE 80 443

STOPSIGNAL SIGTERM

CMD ["nginx", "-g", "daemon off;"]


student@ubuntu:~/myapp$ time sudo docker image build -t lfstudent/nginx:export .
Sending build context to Docker daemon   7.68kB
Step 1/10 : FROM debian:buster-slim
 ---> 2f14a0fb67b9
Step 2/10 : LABEL maintainer="NGINX Docker Maintainers
<docker-maint@nginx.com>"
 ---> Using cache
 ---> 49391ba47654
Step 3/10 : ENV NGINX_VERSION   1.17.9
 ---> Using cache
```

```
...
Step 8/10 : EXPOSE 80 443
 ---> Running in 0c928237fe71
Removing intermediate container 0c928237fe71
 ---> 2a6687756f54
Step 9/10 : STOPSIGNAL SIGTERM
 ---> Running in 94332e1e6d61
Removing intermediate container 94332e1e6d61
 ---> 46f77237d8c7
Step 10/10 : CMD ["nginx", "-g", "daemon off;"]
 ---> Running in 4365c70c3c6b
Removing intermediate container 4365c70c3c6b
 ---> 16cf5322f3dc
Successfully built 16cf5322f3dc
Successfully tagged lfstudent/nginx:export

real  0m0.708s
user  0m0.042s
sys   0m0.037s
```

This build took 0.7 seconds, about half a second longer than the previous build which used the cache for every single build step. The latest build took longer because starting with step 8, representing the EXPOSE instruction, the build steps had to be executed instead of referencing the cache.

Let's rerun the last build, instructing the build to avoid the cache:

```
student@ubuntu:~/myapp$ time sudo docker image build --no-cache -t
lfstudent/nginx:export .
Sending build context to Docker daemon   7.68kB
Step 1/10 : FROM debian:buster-slim
 ---> 2f14a0fb67b9
Step 2/10 : LABEL maintainer="NGINX Docker Maintainers
<docker-maint@nginx.com>"
 ---> Running in 8fe44dbce7ac
...
Step 8/10 : EXPOSE 80 443
 ---> Running in 15b177f17838
Removing intermediate container 15b177f17838
 ---> e8bd8671cfa1
Step 9/10 : STOPSIGNAL SIGTERM
 ---> Running in a1cc364aafb7
Removing intermediate container a1cc364aafb7
 ---> a25219ec3245
Step 10/10 : CMD ["nginx", "-g", "daemon off;"]
 ---> Running in eda94acf0953
Removing intermediate container eda94acf0953
 ---> 666cac18dc1c
```

```
Successfully built 666cac18dc1c
Successfully tagged lfstudent/nginx:export

real   0m33.727s
user   0m0.074s
sys    0m0.077s
```

This final build took 33 seconds, which is very close to the first timed build. From the output we see that each build step was executed and not referenced from the cache.

Listing the images we've previously built, we may notice something strange. The last image built is shown with its full repository name and tag (ID 666cac18dc1c), while the previous build (ID 16cf5322f3dc) no longer shows the repository and tag. With both builds using the same image name, only the latest image kept the desired name, while the previous image was stripped of its repository name and tag.

```
student@ubuntu:~/myapp$ sudo docker image ls
REPOSITORY          TAG             IMAGE ID         CREATED               SIZE
lfstudent/nginx     export          666cac18dc1c     About a minute ago    127MB
<none>              <none>          16cf5322f3dc     14 minutes ago        127MB
lfstudent/nginx     cached          31c7447f26b9     31 minutes ago        127MB
lfstudent/nginx     dockerfile      31c7447f26b9     31 minutes ago        127MB
```

Such images with <none> as repository name and tag, are known as dangling images. We can filter through the image list to display only such images, and ultimately remove them from the local repository:

```
student@ubuntu:~/myapp$ sudo docker image ls --quiet --filter=dangling=true
16cf5322f3dc

student@ubuntu:~/myapp$ sudo docker image ls --quiet --filter=dangling=true |
xargs --no-run-if-empty sudo docker image rm
Deleted:
sha256:16cf5322f3dcfff356cf20b19c67c39f55b4a94efe110acdb546a9ebb33d1b28
Deleted:
sha256:46f77237d8c7f6b10a30c97d855e8c6406df6363e1f70dab133b0b4cdeb7b8b6
Deleted:
sha256:2a6687756f543e57d8fe9d9f08604d7c00bb7c1b530f81a16f287a8eba53f7fc

student@ubuntu:~/myapp$ sudo docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
lfstudent/nginx     export              666cac18dc1c        4 minutes ago
127MB
lfstudent/nginx     cached              31c7447f26b9        35 minutes ago
127MB
lfstudent/nginx     dockerfile          31c7447f26b9        35 minutes ago
127MB
```

Lab 8.1 - Docker Networking

Working with Docker networks

1. List available networks

Before working with networks and container networking in Docker, let's quickly display the networks available in Docker running on our host system:

```
root@ubuntu:~# docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
0178f26dea86        bridge              bridge              local
812e1f9599d1        host                host                local
3cadc63d003f        none                null                local
```

Docker makes available three different networks for usage with containers: the default bridge network, and two additional networks used to start and connect a container directly to the host networking stack, or to start a container with no network devices.

2. Display network details

Displaying details of a network can be achieved by referencing the network by its network ID or by its name. Although using either the network ID **`0178f26dea86`** or the name **`bridge`** will produce the same output, we can take advantage of the autocomplete feature when typing out the command with the network name instead of the network ID. The output has been slightly edited for readability:

```
root@ubuntu:~# docker network inspect 0178f26dea86
[
    {
```

```
        "Name": "bridge",
        "Id": "0178f26dea86320062538f6fb22857adde9c602ea4edf8750c00c84f1...",
        "Created": "2020-01-10T18:53:02.313768719Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "172.17.0.0/16",
                    "Gateway": "172.17.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {},
        "Options": {
            "com.docker.network.bridge.default_bridge": "true",
            "com.docker.network.bridge.enable_icc": "true",
            "com.docker.network.bridge.enable_ip_masquerade": "true",
            "com.docker.network.bridge.host_binding_ipv4": "0.0.0.0",
            "com.docker.network.bridge.name": "docker0",
            "com.docker.network.driver.mtu": "1500"
        },
        "Labels": {}
    }
]
```


By default, the **`bridge`** network uses the **`bridge`** driver, and it creates the **`172.17.0.0/16`** subnet for the IP addresses to be assigned to containers running on this host when attached to the **`bridge`** network.

3. Create a user-defined network

Let's create a **`bridge`** network of our own, ensuring we can **attach** running containers to it as well as new containers. The new **`mynet`** network also uses the **`bridge`** driver, and it creates the **`172.18.0.0/16`** subnet for the IP addresses to be assigned to containers running on this host when attached to the **`mynet`** network:

```
root@ubuntu:~# docker network create -d bridge --attachable mynet
Bdee7801048d562fd3c4d3303ad2fca0f48ceae15d2fe2c96076219bf4a7cb8b
root@ubuntu:~# docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
0178f26dea86        bridge              bridge              local
812e1f9599d1        host                host                local
bdee7801048d        mynet               bridge              local
3cadc63d003f        none                null                local
root@ubuntu:~# docker network inspect mynet
[
    {
        "Name": "mynet",
        "Id": "bdee7801048d562fd3c4d3303ad2fca0f48ceae15d2fe2c96076219bf4...",
        "Created": "2020-01-10T19:59:01.756312Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.18.0.0/16",
                    "Gateway": "172.18.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": true,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {},
        "Options": {},
        "Labels": {}
    }
]
```

By contrast, the **`mynet`** network is attachable while the default **`bridge`** network is not.

4. List all containers attached to a particular network

Although there is not one command to list all containers connected to a particular network, this is a method of listing those containers:

```
root@ubuntu:~# docker network inspect <network-name>|grep Name|tail -n +2|cut
-d':' -f2|tr -d ' ,"'
```

5. Remove a particular network or all unused networks

Prior to removing a Docker network, you must ensure that all running containers have been disconnected from it. Let's create a **`testnet`** network first and then remove it, so that we save the **`mynet`** network for the following exercises.

```
root@ubuntu:~# docker network create testnet
fb46a2007c7d7de6084e8ae608a48ad3a9819a4b750fdf9294f1f9452ae8ee9a
root@ubuntu:~# docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
0178f26dea86        bridge              bridge              local
812e1f9599d1        host                host                local
bdee7801048d        mynet               bridge              local
3cadc63d003f        none                null                local
fb46a2007c7d        testnet             bridge              local
root@ubuntu:~# docker network rm testnet
testnet
root@ubuntu:~# docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
0178f26dea86        bridge              bridge              local
812e1f9599d1        host                host                local
bdee7801048d        mynet               bridge              local
3cadc63d003f        none                null                local
```

A network can be removed either by name or by network ID. Before removal we may verify that there are no containers attached to a particular network by displaying the network's details with **`docker network inspect <network-name>`** and observing the **`"Containers": {}`** field. The empty curly brackets confirm that no container is attached to this network. By listing the networks after the creation step and after the removal step we verify that the **`testnet`** network has been created and then removed successfully.

If we have many unused networks (and we confirmed that there are no containers attached to them), and we want to remove all of them at once, we may do so with the following command:

```
root@ubuntu:~# docker network prune
```

Working with container networking

1. Display the IP address of a running container

Let's run a container with the **`nginx`** container image without specifying a network this time, and then display the container details to observe the network it is attached to by default, the IP address assigned to it, MAC address, etc. The output has been slightly edited for readability:

```
root@ubuntu:~# docker container run -d --name web nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
8ec398bc0356: Pull complete
dfb2a46f8c2c: Pull complete
b65031b6a2a5: Pull complete
Digest:
sha256:8aa7f6a9585d908a63e5e418dc5d14ae7467d2e36e1ab4f0d8f9d059a3d071ce
Status: Downloaded newer image for nginx:latest
b0025c13ea35a1d0c2a21a1f9af397ab5e6ea7e8c7eca17501c8a316909123a9
root@ubuntu:~# docker container ls
CONTAINER ID  IMAGE  COMMAND      CREATED         STATUS        PORTS   NAMES
b0025c13ea35  nginx  "nginx -g …" 13 seconds ago Up 8 seconds 80/tcp  web
root@ubuntu:~# docker container inspect web | more
[
    {
        "Id": "b0025c13ea35a1d0c2a21a1f9af397ab5e6ea7e8c7eca17501c8a316...",
        "Created": "2020-01-10T20:43:36.381148157Z",
        "Path": "nginx",
        "Args": [
            "-g",
            "daemon off;"
        ],
        "State": {
            "Status": "running",
            "Running": true,
            "Paused": false,
            "Restarting": false,
            "OOMKilled": false,
            "Dead": false,
            "Pid": 10003,
            "ExitCode": 0,
```

```
            "Error": "",
            "StartedAt": "2020-01-10T20:43:41.202149787Z",
            "FinishedAt": "0001-01-01T00:00:00Z"
        },
...
        "NetworkSettings": {
            "Bridge": "",
            "SandboxID": "c7f17408ca95acf90ae334ced9b2cdfcf93dc3ce076c5...",
            "HairpinMode": false,
            "LinkLocalIPv6Address": "",
            "LinkLocalIPv6PrefixLen": 0,
            "Ports": {
                "80/tcp": null
            },
            "SandboxKey": "/var/run/docker/netns/c7f17408ca95",
            "SecondaryIPAddresses": null,
            "SecondaryIPv6Addresses": null,
            "EndpointID": "f8ad764b149489a08e6512ec2194ca95e75a3ff247a9...",
            "Gateway": "172.17.0.1",
            "GlobalIPv6Address": "",
            "GlobalIPv6PrefixLen": 0,
            "IPAddress": "172.17.0.2",
            "IPPrefixLen": 16,
            "IPv6Gateway": "",
            "MacAddress": "02:42:ac:11:00:02",
            "Networks": {
                "bridge": {
                    "IPAMConfig": null,
                    "Links": null,
                    "Aliases": null,
                    "NetworkID": "0178f26dea86320062538f6fb22857adde9c6...",
                    "EndpointID": "f8ad764b149489a08e6512ec2194ca95e75a...",
                    "Gateway": "172.17.0.1",
                    "IPAddress": "172.17.0.2",
                    "IPPrefixLen": 16,
                    "IPv6Gateway": "",
                    "GlobalIPv6Address": "",
                    "GlobalIPv6PrefixLen": 0,
                    "MacAddress": "02:42:ac:11:00:02",
                    "DriverOpts": null
                }
            }
        }
    }
]
```

To no surprise, the **web** container is running attached to the default **bridge** network (observe the **"NetworkID": "0178f26dea86…"** field) and it received the **172.17.0.2** IP address from the default **172.17.0.0/16** subnet.

When Docker daemon starts, it creates a **docker0** network bridge on the host system. By default, all the containers connect to the **docker0** network bridge. Docker creates a **veth** pair to attach a container to a bridge. One end of the **veth** pair is attached to the bridge while the other end to the container. The bridge side of the veth pair is **vethcbba2f9@if7** while the container end of the veth pair is **eth0** (not displayed here, but accessible from inside the running container):

```
root@ubuntu:~# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1460 qdisc fq_codel state UP
group default qlen 1000
    link/ether 42:01:0a:80:00:03 brd ff:ff:ff:ff:ff:ff
    inet 10.128.0.3/32 scope global dynamic ens4
       valid_lft 2693sec preferred_lft 2693sec
    inet6 fe80::4001:aff:fe80:3/64 scope link
       valid_lft forever preferred_lft forever
3: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
group default
    link/ether 02:42:92:80:94:51 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:92ff:fe80:9451/64 scope link
       valid_lft forever preferred_lft forever
4: br-bdee7801048d: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue
state DOWN group default
    link/ether 02:42:e0:44:d1:5e brd ff:ff:ff:ff:ff:ff
    inet 172.18.0.1/16 brd 172.18.255.255 scope global br-bdee7801048d
       valid_lft forever preferred_lft forever
8: vethcbba2f9@if7: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
master docker0 state UP group default
    link/ether c2:c9:21:01:c3:09 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet6 fe80::c0c9:21ff:fe01:c309/64 scope link
       valid_lft forever preferred_lft forever
```

2. Expose a container port on a specific port of the host

When running with default options, a container is not directly accessible from outside the host because for security and privacy reasons it is isolated and shielded from the outside world. However, we can expose a container to the outside world, for when a client needs access to a particular front-end service or a portal service running inside a container. We may publish a container port, or map a port on the host to a container port with a similar mapping notation: **<hostPort>:<containerPort>**.

```
root@ubuntu:~# docker container run -d --name web1 -p 80:80 nginx
a1a9916f9abba19deb21138c32e3d4c68f6f10e84168027f4acdc2392d07210f
root@ubuntu:~# docker container ls
CONTAINER ID    IMAGE   ...   STATUS          PORTS                   NAMES
a1a9916f9abb    nginx   ...   Up 8 seconds    0.0.0.0:80->80/tcp      web1
b0025c13ea35    nginx   ...   Up 2 hours      80/tcp                  web
```

Once the **web1** container port **80** has been published, we may access the **nginx** service running inside the container directly over the host IP (regardless of whether private or public IP of you host system):

```
root@ubuntu:~# curl 10.128.0.3
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
...
</html>
root@ubuntu:~# curl 35.239.38.42
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
...
</html>
```

Unfortunately, we may only publish one port of a single container through the port 80 of the host. Even if we attempt to publish another container port through the host port 80, we will be unsuccessful:

```
root@ubuntu:~# docker container run -d --name web2 -p 80:80 nginx
ab4e1b2d3234af2b0a26e042f2279e46c6f875eac6769e96a4dfb445368fdb31
docker: Error response from daemon: driver failed programming external
connectivity on endpoint web2
(1a078fbf0a0df30b7ee646c8758ab04aeae1274adf17e2b99808adb4b1f2e73b): Bind for
0.0.0.0:80 failed: port is already allocated.
```

However, we may publish container port 80 through another host port, such as host port 8080:

```
root@ubuntu:~# docker container run -d --name web3 -p 8080:80 nginx
13a805bb0b2e7ea93fc7ba7662c5c13acbbd2d74f92289b714506c08c6d8b789
root@ubuntu:~# docker container ls
CONTAINER ID     IMAGE   ...   STATUS          PORTS                   NAMES
13a805bb0b2e     nginx   ...   Up 6 seconds    0.0.0.0:8080->80/tcp    web3
```

```
a1a9916f9abb     nginx    ...    Up 21 minutes     0.0.0.0:80->80/tcp     web1
b0025c13ea35     nginx    ...    Up 2 hours        80/tcp                 web
```

Because **curl** targets port **80** by default, we now have to specify the host port **8080** with the **curl** command in order to access the **nginx** service running inside the **web3** container:

```
root@ubuntu:~# curl 10.128.0.3:8080
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
...
</html>
root@ubuntu:~# curl 35.239.38.42:8080
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
...
</html>
```

3. Expose a container port on a random port of the host

In the previous exercise we manually mapped container ports to host ports. When we publish a couple of container ports on the host it may not be such a challenging task. However, when running hundreds, or possibly more containers on one host, keeping track of mapped port numbers could become quite challenging, if not impossible. For such challenging cases, Docker comes to the rescue and offers a very simple solution - it maps all the ports of a container to random host ports picked from a pool of available host ports. Below the port **80** of **web4** container is mapped randomly to host port **32768**.

```
root@ubuntu:~# docker container run -d --name web4 -P nginx
d33bde5197c4a2a6d1c10752151c1a21a53ed234d2811cf7fd9a91cf5373410f
root@ubuntu:~# docker container ls
CONTAINER ID     IMAGE    ...    STATUS           PORTS                     NAMES
d33bde5197c4     nginx    ...    Up 8 seconds     0.0.0.0:32768->80/tcp     web4
13a805bb0b2e     nginx    ...    Up 16 minutes    0.0.0.0:8080->80/tcp      web3
a1a9916f9abb     nginx    ...    Up 37 minutes    0.0.0.0:80->80/tcp        web1
b0025c13ea35     nginx    ...    Up 2 hours       80/tcp                    web
```

Now we need to specify port **32768** with the **curl** command in order to access the **nginx** service running inside the **web4** container:

```
root@ubuntu:~# curl 10.128.0.3:32768
<!DOCTYPE html>
<html>
```

```
<head>
<title>Welcome to nginx!</title>
...
</html>
root@ubuntu:~# curl 35.239.38.42:32768
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
...
</html>
```

4. Run a container without a network interface

We may run a container without attaching it to a network, perhaps to attach it later to a particular network or not attach it at all. In the interactive terminal of the container running the **`alpine`** image we are able to confirm there are no networks the container is attached to:

```
root@ubuntu:~# docker container run -it --network=none alpine sh
Unable to find image 'alpine:latest' locally
latest: Pulling from library/alpine
e6b0cf9c0882: Pull complete
Digest:
sha256:21716586201556792 40babee0a7714f6509fae66898db422ad803b951257db78
Status: Downloaded newer image for alpine:latest
/ # ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
/ # exit
```

5. Share the host network namespace with a container

Similarly we may run a container that shares the host network namespace. In the interactive terminal of the container running the **`alpine`** image we are able to confirm that it shares the host network namespace by observing in the output the entire network configuration of the host system:

```
root@ubuntu:~# docker container run -it --network=host alpine sh
/ # ip a
```

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1460 qdisc fq_codel state UP
qlen 1000
    link/ether 42:01:0a:80:00:03 brd ff:ff:ff:ff:ff:ff
    inet 10.128.0.3/32 scope global dynamic ens4
       valid_lft 1858sec preferred_lft 1858sec
    inet6 fe80::4001:aff:fe80:3/64 scope link
       valid_lft forever preferred_lft forever
3: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
    link/ether 02:42:92:80:94:51 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:92ff:fe80:9451/64 scope link
       valid_lft forever preferred_lft forever
4: br-bdee7801048d: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue
state DOWN
    link/ether 02:42:e0:44:d1:5e brd ff:ff:ff:ff:ff:ff
    inet 172.18.0.1/16 brd 172.18.255.255 scope global br-bdee7801048d
       valid_lft forever preferred_lft forever
8: vethcbba2f9@if7: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc
noqueue master docker0 state UP
    link/ether c2:c9:21:01:c3:09 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::c0c9:21ff:fe01:c309/64 scope link
       valid_lft forever preferred_lft forever
10: veth2df7e3b@if9: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc
noqueue master docker0 state UP
    link/ether 06:5c:3b:41:c5:a8 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::45c:3bff:fe41:c5a8/64 scope link
       valid_lft forever preferred_lft forever
14: veth2c2a394@if13: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc
noqueue master docker0 state UP
    link/ether e2:f6:74:83:b7:18 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::e0f6:74ff:fe83:b718/64 scope link
       valid_lft forever preferred_lft forever
16: veth8008340@if15: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc
noqueue master docker0 state UP
    link/ether fe:cc:e6:87:66:8b brd ff:ff:ff:ff:ff:ff
    inet6 fe80::fccc:e6ff:fe87:668b/64 scope link
       valid_lft forever preferred_lft forever
/ # exit
```

In an attempt to verify this on the host system we may run the following command on the host system and compare the outputs. Surprisingly, or not, they are identical:

```
root@ubuntu:~# ip a
...
```

6. Share a network namespace among containers

Let's revisit the **web** container, which is still running the **nginx** container image. Let's keep in mind that a container image is minimal in size, therefore it packs a minimal amount of features and libraries required by the container environment to run the image. Having said that, let's try to find the **ip** package in the running container by starting an interactive terminal into the **web** container:

```
root@ubuntu:~# docker container exec -it web sh
# which ip
#
```

It was not found. However, we could really use the help of the **ip** package to display the IP address of the **web** container. So let's install it, display the container IP address, and then exit the container with the **Ctrl+p Ctrl+q** sequence:

```
# apt update
...
# apt install -y iproute2
...
# which ip
/sbin/ip
# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
7: eth0@if8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
group default
    link/ether 02:42:ac:11:00:02 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 172.17.0.2/16 brd 172.17.255.255 scope global eth0
       valid_lft forever preferred_lft forever
# ^p^q
```

Let's run a new container and attempt to share the network namespace of the **web** container with this new container. Subsequently we are able to confirm that the new alpine container shares the web container network namespace together with its IP address **172.17.0.2**:

```
root@ubuntu:~# docker container run -it --network=container:web alpine sh
/# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
7: eth0@if8: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue
state UP
    link/ether 02:42:ac:11:00:02 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.2/16 brd 172.17.255.255 scope global eth0
       valid_lft forever preferred_lft forever
/ # ^p^q
```

7. Run a container and attach it to an existing network

In a previous exercise we created a **`bridge`** network, **`mynet`**, which was configured with the **`172.18.0.0/16`** subnet for container IP addresses. Let's run a container and attach it to the mynet network. Then let's display the container IP address:

```
root@ubuntu:~# docker container run -it --network=mynet alpine sh
/ # ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
23: eth0@if24: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue
state UP
    link/ether 02:42:ac:12:00:02 brd ff:ff:ff:ff:ff:ff
    inet 172.18.0.2/16 brd 172.18.255.255 scope global eth0
       valid_lft forever preferred_lft forever
```

The new container was assigned the IP address **`172.18.0.2`** from the **`mynet`** network, confirming that this container was attached to the existing **`mynet`** network.

8. Attach a running container to an existing network

There may be times when an already running container needs to be attached to a network, such as when the network latency needs to be minimized between two communicating services, or to allow two services to talk to each other. Let's run a new container from the **`alpine`** container image without a network. Then

let's manually attach the new **alpine** container to the existing **mynet** network and verify that its IP address was assigned out of the **172.18.0.0/16** subnet of the **mynet** network:

```
root@ubuntu:~# docker container run -it --name alpine alpine sh
/ # ^p^q
root@ubuntu:~# docker network connect mynet alpine
root@ubuntu:~# docker container exec -it alpine sh
/ # ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
31: eth0@if32: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue
state UP
    link/ether 02:42:ac:11:00:06 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.6/16 brd 172.17.255.255 scope global eth0
       valid_lft forever preferred_lft forever
33: eth1@if34: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue
state UP
    link/ether 02:42:ac:12:00:02 brd ff:ff:ff:ff:ff:ff
    inet 172.18.0.2/16 brd 172.18.255.255 scope global eth1
       valid_lft forever preferred_lft forever
/ #
```

9. Detach a running container from a network

Detaching a running container from a network is quite simple. Let's display the details of the **alpine** container, showing that it is connected to two networks: the default **bridge** network and the user-defined **mynet** network:

```
root@ubuntu:~# docker container inspect alpine
...
       "NetworkSettings": {
            "Bridge": "",
            ...
            "Networks": {
                "bridge": {
                    "IPAMConfig": null,
                    "Links": null,
                    "Aliases": null,
                    "NetworkID": "0178f26dea86320062538f6fb22857...",
                    "EndpointID": "6020ed89d0846279f58f3b4b03098...",
                    "Gateway": "172.17.0.1",
                    "IPAddress": "172.17.0.6",
```

```
                    "IPPrefixLen": 16,
                    "IPv6Gateway": "",
                    "GlobalIPv6Address": "",
                    "GlobalIPv6PrefixLen": 0,
                    "MacAddress": "02:42:ac:11:00:06",
                    "DriverOpts": null
                },
                "mynet": {
                    "IPAMConfig": {},
                    "Links": null,
                    "Aliases": [
                        "fb81e99a7b7c"
                    ],
                    "NetworkID": "bdee7801048d562fd3c4d3303ad2fc...",
                    "EndpointID": "c9fcac5ee9584ade0b14df1f1f977...",
                    "Gateway": "172.18.0.1",
                    "IPAddress": "172.18.0.2",
                    "IPPrefixLen": 16,
                    "IPv6Gateway": "",
                    "GlobalIPv6Address": "",
                    "GlobalIPv6PrefixLen": 0,
                    "MacAddress": "02:42:ac:12:00:02",
                    "DriverOpts": null
                }
            }
        }
    }
]
```

Let's detach the **`alpine`** container from the **`mynet`** network, and confirm by displaying the container details again:

```
root@ubuntu:~# docker network disconnect mynet alpine
root@ubuntu:~# docker container inspect alpine
...
       "NetworkSettings": {
            "Bridge": "",
            ...
            "Networks": {
                "bridge": {
                    "IPAMConfig": null,
                    "Links": null,
                    "Aliases": null,
                    "NetworkID": "0178f26dea86320062538f6fb22857...",
                    "EndpointID": "6020ed89d0846279f58f3b4b03098...",
                    "Gateway": "172.17.0.1",
                    "IPAddress": "172.17.0.6",
                    "IPPrefixLen": 16,
```

```
                    "IPv6Gateway": "",
                    "GlobalIPv6Address": "",
                    "GlobalIPv6PrefixLen": 0,
                    "MacAddress": "02:42:ac:11:00:06",
                    "DriverOpts": null
                }
            }
        }
    }
]
```

Lab 8.2 - Rkt Networking

1. Run a pod with a loopback only interface

Also called the no networking method, it completely isolates a pod's network:

```
root@ubuntu:~/rkt-v1.30.0# ./rkt run --interactive --net=none
quay.io/coreos/alpine-sh:latest
pubkey: prefix: "quay.io/coreos/alpine-sh"
key: "https://quay.io/aci-signing-key"
gpg key fingerprint is: BFF3 13CD AA56 0B16 A898  7B8F 72AB F5F6 799D 33BC
        Quay.io ACI Converter (ACI conversion signing key) <support@quay.io>
Are you sure you want to trust this key (yes/no)?
yes
Trusting "https://quay.io/aci-signing-key" for prefix
"quay.io/coreos/alpine-sh" after fingerprint review.
Added key for prefix "quay.io/coreos/alpine-sh" at
"/etc/rkt/trustedkeys/prefix.d/quay.io/coreos/alpine-sh/bff313cdaa560b16a8987b
8f72abf5f6799d33bc"
Downloading signature: [=================================] 473 B/473 B
Downloading ACI: [=======================================] 2.65 MB/2.65 MB
image: signature verified:
  Quay.io ACI Converter (ACI conversion signing key) <support@quay.io>
networking: networking namespace with loopback only
/ # ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
/ # exit
```

2. Share the host network namespace with a pod

A pod can share the host network namespace, in which case it also shares the host IP as well:

```
root@ubuntu:~/rkt-v1.30.0# ./rkt run --interactive --net=host
quay.io/coreos/alpine-sh:latest
/ # ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1460 qdisc fq_codel state UP
qlen 1000
    link/ether 42:01:0a:80:00:04 brd ff:ff:ff:ff:ff:ff
    inet 10.128.0.4/32 scope global dynamic ens4
       valid_lft 2172sec preferred_lft 2172sec
    inet6 fe80::4001:aff:fe80:4/64 scope link
       valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state
DOWN
    link/ether 02:42:ed:e4:25:40 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
/ # exit
```

3. Display the IP address of a running pod

Once a pod is running on a network, we may display the pod's IP address either by listing the running pods, or by displaying the status of that particular pod:

```
root@ubuntu:~/rkt-v1.30.0# systemd-run --slice=machine ./rkt run
--insecure-options=image docker://nginx
Running as unit: run-rdb584374ab5f4074949b6af4508d55f2.service
root@ubuntu:~/rkt-v1.30.0# ./rkt list
UUID     APP       IMAGE NAME                          NETWORKS
1b3c5722 alpine-sh quay.io/coreos/alpine-sh:latest
7c759fbe nginx     registry-1.docker.io/library/ng...  default:ip4=172.16.28.2
cff89b89 alpine-sh quay.io/coreos/alpine-sh:latest
```

```
root@ubuntu:~/rkt-v1.30.0# ./rkt status 7c759fbe
state=running
created=2020-01-11 05:38:26.537 +0000 UTC
started=2020-01-11 05:38:26.661 +0000 UTC
networks=default:ip4=172.16.28.2
pid=4713
exited=false
```

4. Expose a pod port on a specific port of the host

When running with default options, a pod is not directly accessible from outside the host because for security and privacy reasons it is isolated and shielded from the outside world. However, we can expose a pod to the outside world, for when a client needs access to a particular front-end service or a portal service running inside a pod. We may map a port on the host to a pod port with a similar mapping notation: **<podPort-protocol>:<hostPort>**.

```
root@ubuntu:~/rkt-v1.30.0# systemd-run --slice=machine ./rkt run
--port=80-tcp:8080 --insecure-options=image docker://nginx
Running as unit: run-r952ba996152e4a51a8cc5a2df7c8fad2.service
root@ubuntu:~/rkt-v1.30.0# ./rkt list
UUID     APP       IMAGE NAME                          NETWORKS
1b3c5722 alpine-sh quay.io/coreos/alpine-sh:latest
7c759fbe nginx     registry-1.docker.io/library/ng...  default:ip4=172.16.28.2
9055f8fe nginx     registry-1.docker.io/library/ng...  default:ip4=172.16.28.3
cff89b89 alpine-sh quay.io/coreos/alpine-sh:latest
root@ubuntu:~/rkt-v1.30.0# curl 146.148.74.29:8080
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
...
</html>
```

5. Create a user-defined bridge network

Let's create a **bridge** network of our own, named **containers**, with the **172.19.0.0/16** subnet for the IP addresses to be assigned to pods running on this host when attached to the **containers** network. For verification, a pod running the **alpine** image, attached to the newly created **containers** network displays the **172.19.0.2** IP address, which is assigned from the **containers** network subnet:

```
root@ubuntu:~/rkt-v1.30.0# mkdir -p /etc/rkt/net.d
root@ubuntu:~/rkt-v1.30.0# vim /etc/rkt/net.d/10-containers.conf
{
      "name": "containers",
      "type": "bridge",
      "bridge": "rktbr1",
      "ipam":{
            "type": "host-local",
            "subnet": "172.19.0.0/16"
      }
}
root@ubuntu:~/rkt-v1.30.0# ./rkt run --interactive --net=containers
quay.io/coreos/alpine-sh
/ # ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
3: eth0@if6: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue
state UP
    link/ether 92:5a:f7:22:4a:97 brd ff:ff:ff:ff:ff:ff
    inet 172.19.0.2/16 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::905a:f7ff:fe22:4a97/64 scope link
       valid_lft forever preferred_lft forever
/ # exit
```

After completing these lab exercises we should have a pretty good understanding of how networking is handled by both **`Docker`** and **`rkt`**.

Lab 9.1 - Manage Storage with Docker

1. Run a container with a mounted volume

To manage volumes for containers, we may use the **`--mount`** or the **`-v`** options. Let's run the **`alpine`** container image in a **`cvol`** container while mounting a volume under the container's **`/data`** mount point. Passing the **`-i`** and **`-t`** options and the **`sh`** command allow us to interact with the container's environment from a terminal:

```
root@ubuntu:~# docker container run -i -t --mount target=/data --name cvol
alpine sh
Unable to find image 'alpine:latest' locally
latest: Pulling from library/alpine
e6b0cf9c0882: Pull complete
Digest:
sha256:21716586201556792 40babee0a7714f6509fae66898db422ad803b951257db78
Status: Downloaded newer image for alpine:latest
/ # cd /data/
/data # ls
/data # touch file1
/data # ls
file1
```

As a verification step, we created a **`file1`** file inside the **`/data`** directory of the **`cvol`** container.

Similarly, the volume could have been mounted inside the container with the **`-v`** option:

```
root@ubuntu:~# docker container run -i -t -v /data --name cvol alpine sh
```

2. Run a container with a mounted named volume (named volume created on the fly)

Let's run the same **alpine** container image in a **cmountvol** container while mounting a volume named **mountvol** under the container's **/data** mount point:

```
root@ubuntu:~# docker container run -i -t --mount source=mountvol,target=/data
--name cmountvol alpine sh
/ # cd /data/
/data # ls
/data # touch mount-file1
/data # ls
mount-file1
```

As a verification step, we created a **mount-file1** file inside the **/data** directory of the **cmountvol** container.

Similarly, the volume could have been mounted inside the container with the **-v** option:

```
root@ubuntu:~# docker container run -i -t -v mountvol:/data --name cmountvol
alpine sh
```

3. Display container details to view mounted volume properties

By displaying the container's details, inside the **Mounts** section we find detailed information about mounted volumes, such as source, destination, and access mode. The output below has been redacted for readability:

```
root@ubuntu:~# docker container ls
CONTAINER ID   IMAGE    COMMAND CREATED         STATUS     PORTS   NAMES
f360a5ff317a   alpine   "sh"    2 minutes ago   Up 2 minutes       cmountvol
aead1c30cd3d   alpine   "sh"    4 minutes ago   Up 4 minutes       cvol
root@ubuntu:~# docker container inspect cvol | more
...
        "Mounts": [
            {
                "Type": "volume",
                "Name": "50875bf11582aaacb98e29063...",
                "Source": "/var/lib/docker/volumes/50875bf1.../_data",
                "Destination": "/data",
                "Driver": "local",
                "Mode": "",
                "RW": true,
```

```
                "Propagation": ""
            }
        ],
...
```

The full volume name is **50875bf11582aaacb98e290630421ad9e71657b4957717f63848082e1cdc3624**, the full path to the source volume on the host system is **/var/lib/docker/volumes/50875bf11582aaacb98e290630421ad9e71657b4957717f63848082e1cdc3624/_data**, and the target mount point on the container is **/data**. This volume is mounted in Read-Write mode **("RW":true)**. By navigating to the source on the host system, we are able to find the verification file **file1** created inside the container's **/data** directory:

```
root@ubuntu:~# ls /var/lib/docker/volumes/50875bf11582aaac.../_data/
file1
```

By default, all the local volumes are saved under **/var/lib/docker/volumes** directory of the host system.

4. List volumes

Let's list the volumes created and mounted so far:

```
root@ubuntu:~# docker volume ls
DRIVER          VOLUME NAME
local           50875bf11582aaacb98e290630421ad9e71657b4957717f63...
local           mountvol
```

5. Create a named volume

Previously we allowed Docker to create volumes on our behalf at container runtime. However, Docker provides us with the capability to create our own volume, and then decide whether we want to mount it in a container.

Let's create a volume and then list the available volumes:

```
root@ubuntu:~# docker volume create --name myvol
myvol
root@ubuntu:~# docker volume ls
DRIVER          VOLUME NAME
local           50875bf11582aaacb98e290630421ad9e71657b4957717f63...
local           mountvol
```

```
local           myvol
```

6. Run a container with a mounted named volume (pre-created volume)

With the **myvol** volume available, we decide to mount it in a container and then create a file and add some content to it:

```
root@ubuntu:~# docker container run -i -t --mount source=myvol,target=/data
--name cmyvol alpine sh
/ # cd /data/
/data # echo "Docker volumes" > learning.txt
/data # ls
learning.txt
/data # cat learning.txt
Docker volumes
/data # Ctrl+p Ctrl+q
root@ubuntu:~# cat /var/lib/docker/volumes/myvol/_data/learning.txt
Docker volumes
```

As a verification step, not only that we created a **learning.txt** file with some content on the mounted volume in the container, but also we verified the existence of the file together with its content from the host system by navigating to the source path of the volume **/var/lib/docker/volumes/myvol/_data/**.

7. Remove a volume

If the volume is created with the container, such as the case of the **cvol** container, we can remove the volume together with the container:

```
root@ubuntu:~# docker container rm -f -v cvol
```

The **-v** option instructs Docker to remove the volume together with the container.

Volumes may be removed separately as well, provided they have been unmounted/released from the container that used them. The first attempt to remove the **myvol** volume fails, with the error message displaying the ID of the container still mounting the volume. We then have to stop and remove the container to unmount/release the volume we intend to remove:

```
root@ubuntu:~# docker volume ls
DRIVER          VOLUME NAME
```

```
local           mountvol
local           myvol
root@ubuntu:~# docker volume rm myvol
Error response from daemon: remove myvol: volume is in use -
[62870ebe162299620a92d7fd280d46f54f9337b420f8c072d8ce892f86ff2ff2]
root@ubuntu:~# docker container ls
CONTAINER ID   IMAGE   COMMAND CREATED          STATUS      PORTS     NAMES
62870ebe1622   alpine  "sh"    9 minutes ago    Up 8 minutes          cmyvol
f360a5ff317a   alpine  "sh"    23 minutes ago   Up 23 minutes         cmountvol
root@ubuntu:~# docker container rm -f cmyvol
cmyvol
root@ubuntu:~# docker volume rm myvol
myvol
root@ubuntu:~# docker container ls
CONTAINER ID   IMAGE   COMMAND CREATED          STATUS      PORTS     NAMES
f360a5ff317a   alpine  "sh"    24 minutes ago   Up 24 minutes         cmountvol
root@ubuntu:~# docker volume ls
DRIVER          VOLUME NAME
local           mountvol
```

8. Mount a host directory inside a container in bind mode

We may use a host directory as a shared storage location between the host system and a container, by mounting a host directory inside the container. This can be achieved by specifying an additional parameter to declare it a bind type of mount. A bind type of mount, however, does not produce a new volume entry in the volumes list, and no source path in the default **`/var/lib/docker/volumes/`** directory either.

NOTE: After the **`touch bind-file`** command we want to detach from the container while also keeping it running so we can return to it. Use the **`Ctrl+p Ctrl+q`** keys combination to detach from a running container without terminating it at the same time (presented as **`^p^q`** below).

```
root@ubuntu:~# mkdir /mnt/shared
root@ubuntu:~# docker container run -i -t --mount
type=bind,source=/mnt/shared,target=/data --name csharedvol alpine sh
/ # cd /data/
/data # ls
/data # touch bind-file
/data # ^p^q
root@ubuntu:~#
root@ubuntu:~# cd /mnt/shared/
root@ubuntu:/mnt/shared# ls
bind-file
root@ubuntu:/mnt/shared# echo "text from host" > bind-file
root@ubuntu:/mnt/shared# docker attach csharedvol
```

```
/data # ls
bind-file
/data # cat bind-file
text from host
```

After our verification step where we created the **`bind-file`** file from the container in the mounted directory, we are able to navigate to it and add content to the shared file from the host system, and finally return to the running container with the **`attach`** command (after ensuring we did not terminate the container by detaching from it earlier with **`Ctrl+p Ctrl+q`** keys combination) to read the shared content from inside the container.

9. Mount a Read-Only host directory inside a container in bind mode

In this scenario we provide an additional parameter for the Read-Only option to the bind mount type. During the verification steps, we are not allowed to create new content on the mounted volume from the container, although we attempt to create a new file and to add extra content to the existing file. In both cases we are reminded that the container has Read-Only access to the shared mounted volume, which allows the container to list files in the directory and read the contents of the file:

```
root@ubuntu:~# mkdir /tmp/ro
root@ubuntu:~# echo "host file" > /tmp/ro/host-ro-file
root@ubuntu:~# docker container run -i -t --mount
type=bind,source=/tmp/ro,target=/data,readonly --name crovol alpine sh
/ # cd /data/
/data # ls
host-ro-file
/data # touch container-file
touch: container-file: Read-only file system
/data # ls
host-ro-file
/data # cat host-ro-file
host file
/data # echo "hello from container" >> host-ro-file
sh: can't create host-ro-file: Read-only file system
/data # ls
host-ro-file
/data # cat host-ro-file
host file
```

10. Display host system disk usage by Docker

In order to store container image data, running container data, and data from volumes and other types of storage layer data, Docker makes use of host system's storage. We may list the availability of free storage resources of the host system:

```
root@ubuntu:~# docker system df
TYPE                TOTAL      ACTIVE      SIZE          RECLAIMABLE
Images              1          1           5.591MB       0B (0%)
Containers          4          4           73B           0B (0%)
Local Volumes       3          3           0B            0B
Build Cache         0          0           0B            0B
```

11. How to remove all unused volumes?

We may remove all the unused/unmounted/released volumes from docker, all with one command:

```
root@ubuntu:~# docker volume prune
WARNING! This will remove all local volumes not used by at least one
container.
Are you sure you want to continue? [y/N] y
Deleted Volumes:
mountvol
20e639412902d69f40c5be3bc722b26ad3ebaaba3f35488a5799b024f930f22b
a0fab9c54ae01dc2d08c5ea0c2ba4de654fcb512c9526404c380cb2dca623b6a

Total reclaimed space: 0B
root@ubuntu:~# docker volume ls
DRIVER              VOLUME NAME
root@ubuntu:~#
```

Lab 9.2 - Manage Storage with rkt

Volumes are defined via the --volume flag, the volume is then mounted into each app running in the pod based on information defined in the ACI manifest.

1. Mount a host volume in Read-Write mode

Let's explore rkt's host volume mounting options. The host volume mounting allows rkt pods to map a persistent storage space of the host system to its applications to mount it and use it to share data among themselves.

Let's begin by creating the host directory with an empty file in it. Since rkt runs Docker container images as well, we will run a rkt pod with an **`nginx`** application running from an **`nginx`** container image pulled from the Docker Hub. We will ensure the pod runs in interactive mode and that at runtime it will return a shell for us to validate the volume mount specified as part of the **`rkt run`** command. We will also share data from the pod with the host, and once we exit the pod we will confirm that both systems were able to equally save data in the shared directory, because we enabled a Read-Write mount of the host volume inside the pod, which is the default access mode.

```
root@ubuntu:~# mkdir /tmp/rktvol
root@ubuntu:~# touch /tmp/rktvol/host-file
root@ubuntu:~# ./rkt-v1.30.0/rkt run --interactive --insecure-options=image
--volume=rkthostvol,kind=host,source=/tmp/rktvol --mount
volume=rkthostvol,target=/data/app1 docker://nginx --exec /bin/sh
Downloading sha256:f473f9fd0a8 [======================]     204 B / 204 B
Downloading sha256:8ec398bc035 [======================] 27.1 MB / 27.1 MB
Downloading sha256:465560073b6 [======================] 23.7 MB / 23.7 MB
# ls
bin  boot  data  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run
sbin  srv  sys  tmp  usr  var
# cd data
# ls
app1
```

```
# cd app1
# ls
host-file
# echo pod data > host-file
# cat host-file
pod data
# touch pod-file
# exit
root@ubuntu:~# cat /tmp/rktvol/host-file
pod data
root@ubuntu:~# ls /tmp/rktvol/
host-file pod-file
root@ubuntu:~#
```

2. Mount a host volume in Read-Only mode

In this scenario we will mount a similar host volume, but we will enforce a Read-Only access mode on the mounted file system.

Let's begin by creating the host directory and store some content into a file on it. We will then run a rkt pod with an **`nginx`** application running from an **`nginx`** container image. We will ensure the pod runs in interactive mode and that at runtime it will return a shell for us to validate the volume mount specified as part of the **`rkt run`** command. We will attempt to share data from the pod with the host by trying to write to the existing file and by trying to create a new file in the shared directory. As expected, the pod will not be allowed to store any data on the shared file system because the volume was declared as **`readOnly`** in the **`rkt run`** command. Once we exit the pod we will confirm that the pod did not alter the content of the shared volume.

```
root@ubuntu:~# mkdir /tmp/rktrovol
root@ubuntu:~# echo host read-only data > /tmp/rktrovol/host-file-ro
root@ubuntu:~# ./rkt-v1.30.0/rkt run --interactive --insecure-options=image
--volume=rkthostrovol,kind=host,source=/tmp/rktrovol,readOnly=true --mount
volume=rkthostrovol,target=/data/app2 docker://nginx --exec /bin/sh
# ls
bin  boot  data  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run
sbin  srv  sys  tmp  usr  var
# cd data
# ls
app2
# cd app2
# ls
host-file-ro
# echo pod data >> host-file-ro
/bin/sh: 18: cannot create host-file-ro: Read-only file system
# cat host-file-ro
```

```
host read-only data
# touch pod-file
touch: cannot touch 'pod-file': Read-only file system
# ls
host-file-ro
# exit
root@ubuntu:~# cat /tmp/rktrovol/host-file-ro
host read-only data
root@ubuntu:~# ls /tmp/rktrovol/
host-file-ro
root@ubuntu:~#
```

After completing these lab exercises we should have a pretty good understanding of how storage and volume mounts are handled by both **`Docker`** and **`rkt`**.