Lab 6:

Virtualization with   
containers

Datacenter Virtualization

2024-2025

© Daan Pareit

CONTENT

[1 Introduction 3](#_Toc183713533)

[1.1 Learning goals 3](#_Toc183713534)

[1.1.1 Knowledge 3](#_Toc183713535)

[1.1.2 Skills 3](#_Toc183713536)

[1.2 Prerequisites 3](#_Toc183713537)

[1.3 The history of containers 3](#_Toc183713538)

[2 Installing a Linux virtual machine 5](#_Toc183713539)

[3 Docker recap 5](#_Toc183713540)

[4 podman 12](#_Toc183713541)

[5 LXC / LXD 15](#_Toc183713542)

# 

# Introduction

When talking about containers people usually talk about Docker. It is important to note that Docker is not a synonym for a container, and it is perfectly possible to use containers without Docker. To make things even more complicated there is a difference between **application containers** and **system containers.** No matter the type, containerization (aka. operating-system-level virtualization) has become an important element when talking about virtualization and even more so when talking about deploying and putting systems online.

In this lab we will explore some different tools to create and manage containers.

## Learning goals

### Knowledge

* Know what containers are
* Know the difference between containers and virtual machines
* Know the difference between an application container and a system container

### Skills

* Be able to launch and use containers with docker
* Be able to launch and use containers with lxc/lxd

## Prerequisites

This lab will require a virtualization solution on your laptop, VMware Workstation is recommended but others may work too.

We’ll use a debian VM to explore different container technologies

* You can get the latest debian iso from:

<https://cdimage.debian.org/debian-cd/current/amd64/iso-cd/>

## The history of containers

People often (mistakenly) think containers are something new. There are mainly three reasons why containers have gained popularity in the last few years:

* The microservices design pattern has gained in popularity and is a perfect match for containers.
* Docker became a popular tool (interface) to “easily” manage containers.
* Kubernetes has become very popular to deploy/ship/scale applications and it uses containers.

When talking about containers people usually talk about Docker. Not every container however is managed by Docker! A small overview is given in the links below:

* <https://blog.aquasec.com/a-brief-history-of-containers-from-1970s-chroot-to-docker-2016>
* <https://www.redhat.com/en/blog/history-containers>

# Installing a Linux virtual machine

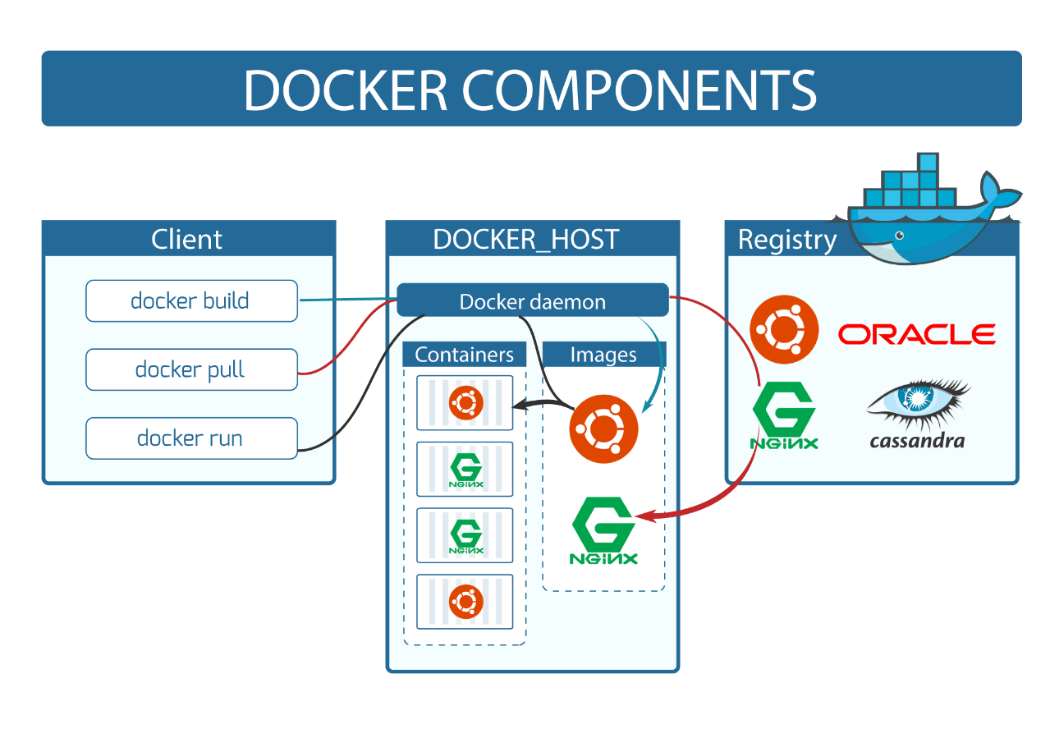
We will use and create a virtual machine that on its turn will act as the “host” for the containers.

Install and configure a new Debian virtual machine with the following tweaks:

* Give at least 4GB of memory.
* Make the hard disk slightly bigger: for example, 50GB
* NAT network is fine
* No graphical desktop environment needed
* Enter ***your first name and last name*** as full name for the new user.
* Enter ***your*** **first name** as username for your account.
* Choose ***debian-<firstname>-<lastname>*** as hostname
* Install the ‘sudo’ package and make sure your regular user can perform sudo commands by adding him/her to the sudo group.
* Take a snapshot of your cleanly installed VM (preferably when powered off)

As we’ll explore different container technologies during this lab, **it is recommended to take snapshots and return to a clean state when switching container runtimes** to avoid side effects of mixing different environments.

# Docker recap



In this section you are asked to rehearse existing Docker knowledge. Remember to take a snapshot of your VM to revert to later. Start by installing docker. We recommend following the instructions written here: <https://docs.docker.com/engine/install/debian/> .

Success A screenshot of a computer

Description automatically generated

Created a snapshot with docker installed.

Try to answer the following questions carefully!

1. Is your regular user able to run docker commands out-of-the-box?

Nope, it can not.

A screen shot of a computer

Description automatically generated

1. What is the easy way to make it possible for your regular user to run a container without requiring sudo?

Check that docker group exists

`getent group docker`

Then add user to it

`sudo usermod -aG docker serafim`

A screenshot of a computer screen

Description automatically generated

1. Issue the command: docker run hello-world and answer the following questions:
   1. What is the image name?

**hello-world:latest**

As far as I can read.

A screen shot of a computer code

Description automatically generated

* 1. Where did docker pull this from (in other words what is the registry?)

**library/hello-world**

**A screen shot of a computer

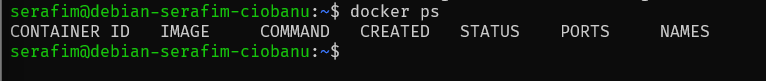
Description automatically generated**

* 1. Is the container still running after the “hello world” text was printed? How can you check this?

**docker ps**

**or docker ps -a**

No, it does not run.



* 1. What command was run?

**/hello**

**A screenshot of a computer program

Description automatically generated**

* 1. What is the difference between the name of a container and the container ID?

Container ID – it is used by docker to uniquely identify a container.

Name – it is more like a friendly usage for humans, because then you can have custom names and understand yourself what is running in there.

* 1. Issue docker inspect <image\_id> on the hello-world container. Is the hostname written in this json-output? Is the IP address of this container written in this output?



Hostname is (it is the ID)

A computer screen with white text

Description automatically generated

IP Address is not

A computer screen shot of white text

Description automatically generated

1. Try to run an instance of the **httpd image** in the background **(use the correct flag)** and answer the questions of the previous bullet again. Do you notice a difference concerning the IP address in the docker inspect?

**docker run -d httpd** (like a detached mode).

A screen shot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated

Now we can finally see some Network Settings (which makes sense)

* 1. What is the ‘gateway’ IP address for the httpd container mentioned in the docker inspect?

A computer screen shot of white text

Description automatically generated

Gateway is **172.17.0.1**

* 1. On your debian host, check the network interfaces. What docker related interface(s) do you now have? Is there a difference if the container is running or not (docker start & docker stop)? Can you find an IP address in the debian network interfaces you also saw in the docker inspect?

There is a new interface `docker0` which has the same settings for network as the docker inspect.

A screenshot of a computer screen

Description automatically generated

After docker stop

A screen shot of a computer

Description automatically generated

It does change.

* 1. Check the routing table on your debian host. Does it include the network for your docker containers?

A screen shot of a computer program

Description automatically generated

Yes it does include the route.

* 1. Verify you can access the httpd webserver in your container with curl.

curl 172.17.0.2:80

A black background with green and white text

Description automatically generated

* 1. Can you access the httpd server when surfing from your laptop to the debian VM?

Nope, you will not be able to do it. Because you are going to the machine itself, and it does not know that you want to access the container itself.

A screenshot of a computer screen

Description automatically generated

* 1. Stop the current httpd container and run a new httpd container in the background, but now with publishing port 80 of your httpd container on port 80 of your debian VM. Can you now access the httpd server when surfing from your laptop to the debian VM?

docker run -d -p 8080:80 httpd

And it works!

A screenshot of a computer

Description automatically generated

A screen shot of a computer screen

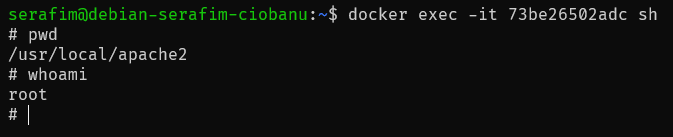
Description automatically generated

Hint: <https://docs.docker.com/get-started/docker-concepts/running-containers/publishing-ports/>

1. Figure out a way to “enter” the httpd container. In other words, retrieve a shell. Who is the user and what is the location (pwd) you’re entering the container? For example: is it a home folder from a user or /etc/httpd or /usr/bin or …?

Hint: <https://docs.docker.com/reference/cli/docker/container/exec/>

docker exec -it 73be26502adc sh (-it for interactive shell, the **sh** is the command itself, in this case it is a shell)



Pwd = /usr/local/apache2

User = root

1. Now, exit the shell on your container and go back to your debian host. Make sure you still have at least 1 container running if you issue the command docker ps.

A black background with white text

Description automatically generated

On your host now run: ps -ef --forest

A screen shot of a computer

Description automatically generated

A black screen with white text

Description automatically generated

* 1. Can you spot the process that is associated with the container? Search for the **container id**. What command/binary is used for this?

/usr/bin/containerd-shim-runc-v2

I am not sure, but I do not see anything related to even remotely to an ID in here.

* 1. **As which user is this process running?**

**root**

* 1. Is this command a child from the docker daemon?

It does not look like it.

1. Verify you now have the systemd units ‘docker.service’ and ‘containerd.service’ .

* Use systemctl status to view their status.
* Who’s using who? Use systemctl cat to view the systemd unit configuration files and pay attention to the keywords for dependencies

Docker.service is using the containerd.service because it waits for it to launch, and also uses it to launch the binary.

A screen shot of a computer

Description automatically generated

1. Should you want to use the httpd container with more options, different tags etc, explore: <https://hub.docker.com/_/httpd>

# podman

Podman is an open-source container management tool designed to simplify the creation, deployment, and management of containers. It provides a daemonless architecture, meaning it does not require a background service to manage containers, unlike Docker. Podman is part of the container tools ecosystem supported by Red Hat and is compatible with the Open Container Initiative (OCI) standards, ensuring interoperability with container images and runtimes. It also natively supports running containers as a non-root user, enhancing security.

1. Take a snapshot and return to your clean installation of the debian virtual machine. Install podman with apt install podman.
2. Run the default hello world with podman run hello-world

A screenshot of a computer program

Description automatically generated

* 1. Did you need sudo?

Nope, I did not need it.

* 1. Where did it pull the image from?

docker.io/library/hello-world:latest

from docker.io libraries.

1. Let’s try with the ‘httpd’ container, as you did before with docker. Does the analogous command “podman run -d httpd” work? What is the correct way to start the httpd container with podman?

podman run -dt -p 8080:80/tcp docker.io/library/httpd

A computer screen with white text

Description automatically generated

A black screen with white text

Description automatically generated

Hint: <https://podman.io/docs>

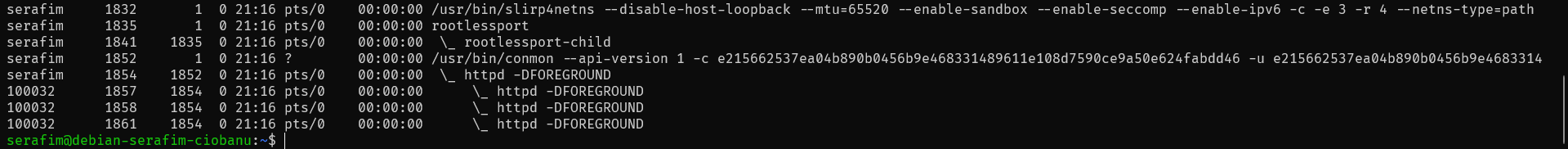
1. Why did podman run hello-world succeed without specifying a registry prefix? Check this config file: /etc/containers/registries.conf.d/shortnames.conf

A screenshot of a computer program

Description automatically generated

It has a bunch of aliases (like localhost, kind of), that allow it to fully get new images.

1. Remember the ps -ef --forest from before? Try running it again. What do you notice? As which user is the container running? What’s the difference with docker?



This time it looks like the container takes less effort to be managed, and hence in this case it even runs with my name, and not root.

# LXC / LXD

LXC (Linux Containers) and LXD (Linux Daemon) are container management technologies focused on providing lightweight, operating system-level virtualization. Unlike Docker and Podman, which are designed to manage application containers, LXC/LXD are better suited for managing system containers—containers that behave like full Linux systems, capable of running multiple processes and services.

* LXC: The lower-level container management library and tools for creating and managing system containers.
* LXD: A higher-level manager built on LXC, offering a user-friendly interface, enhanced networking, and clustering capabilities. It can be seen as a "next-generation" LXC.

Let’s explore LXC/LXD hands-on:

1. Return to the clean state (no docker/podman installed) by using snapshots.
2. The preferred way to install LXD is still with snap (apt install snapd): snap install lxd
3. Figure out where the lxd binary resides on your filesystem. Your $PATH should be updated automatically so logging out and back in (or sourcing) should make these changes effective. However, when performing ‘sudo’, weird behavior can occur. To fix this, perform a sudo visudo and add the $PATH of the lxd binary to the secure\_paths variable in the sudoers file.

A black screen with white text

Description automatically generated

A screen shot of a computer program

Description automatically generated

A screenshot of a computer screen

Description automatically generated

A black background with green and white text

Description automatically generated

1. Once this is done, we need to tell LXD some information about how we will manage the network and storage. Run: sudo lxd init

You can use all default suggestions for the different initialization options.

A screen shot of a computer screen

Description automatically generated

1. It might be contra-intuitive, but most commands will now start with lxc instead of lxd. You can however run lxd.lxc if that would make more sense. For example:

Afbeelding met tekst, schermopname, Lettertype, Elektrisch blauw

Automatisch gegenereerde beschrijving

1. Before we continue, there is an important step we mustn’t forget. Access control for LXD is based on group membership. The root user as well as members of the “lxd” group can interact with the local daemon. Make sure your user is part of the “lxd” group.

**Q: What commando do you use?**

**sudo usermod -aG lxd $USER**

**A black screen with white text

Description automatically generated**

1. Logout and login to your shell to ensure group membership is effective for your session. Now, run the following command: lxc list

A screen shot of a computer program

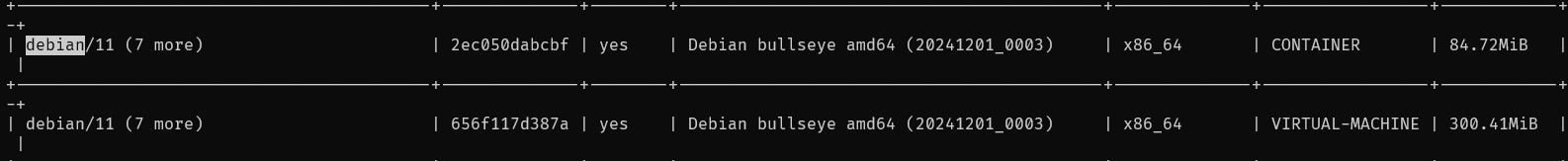
Description automatically generated

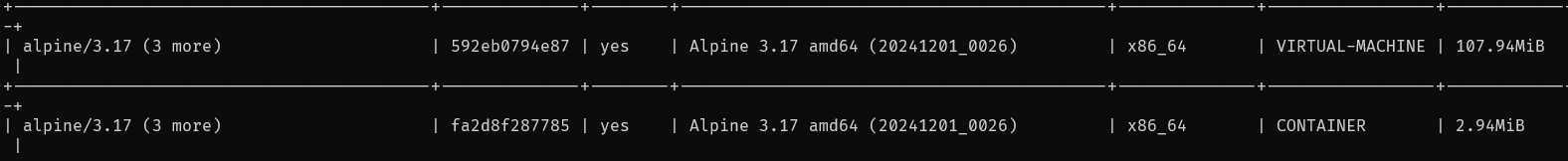
1. Let’s create our first container. First, we need to find an appropriate image. Run the following command to get an overview of the images on the LXD store: lxc image list images:

A screenshot of a computer

Description automatically generated

1. **Q: In this list, compare the size of the alpine container image with a Debian container image. Now compare this to a Debian virtual machine in this list.**

****

****

**Alpine is indeed very lightweight, especially comparing the VM image , which is 3 times as lighter than a debian VM.**

1. You can see the specific images which have an alias name for e.g. alpine with lxc image list images:alpine .
2. Let’s create our first container, which we’ll give -not very creative- the name “first”!

Execute: lxc launch images:alpine/3.20 first

**Q: How long did this take?**

**Is that just a question to be like “Hey look how blazingly fast this is!” or do I need to time it?**

**A black screen with white text

Description automatically generated**

**About this long**

1. Use lxc list to see your container(s) and some basic network information. Use lxc info first to have some more info and statistics about your container.

**A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated**

1. On your debian host, check the network interfaces. What lxc related interface(s) do you now have?

**A screenshot of a computer screen

Description automatically generated**

**lxdbr0**

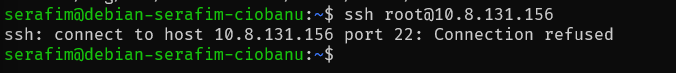
1. **Q: Can you ping to your ‘first’ container? Why or why not? Can you ssh? Why or why not?**

**A screen shot of a computer code

Description automatically generated**

**I can ping**

**And I do not think I can ssh to it, because I do not have a user, and loging in with root via ssh is always disabled?**

****

1. You can pull or push files to a container with:

lxc file pull first/etc/hosts .

A screen shot of a computer

Description automatically generated

Here I “Downloaded” the file from the container, from the /etc/hosts directory.

lxc file push hosts first/tmp/



Here I pushed or “uploaded” the file from my host, onto my container.

What did you do when executing the commands above?

1. Let’s look inside the container. As with docker and podman, the ‘exec’ option allows us to execute a task. As such, you can ‘enter’ the container by starting a shell:

lxc exec first -- /bin/sh

A screenshot of a computer screen

Description automatically generated

**Q: Does /bin/bash work? Why or why not?**

**Because Alpine does not have bash shell installed by default, it uses some other type (which I do not remember by heart)**

1. Of course, if you want to install bash or anything else, you can do this with Alpine’s package manager ‘apk’. E.g., by running the following inside your alpine container:

apk update  
apk upgrade  
apk add bash

A screenshot of a computer program

Description automatically generated

1. Let’s install curl by running: apk add curl
2. Let’s create a second container: lxc launch images:debian/12 second

Had to stop and delete my other container)

Lxc stop second

Lxc delete second

1. **Q: Try to find the command to list your local images:**

**A screenshot of a computer program

Description automatically generated**

1. Now go in your second container and install a webserver (for example apache2). Make sure to create a file test.html at the root of your web folder. Put some text in this file.

Apt update

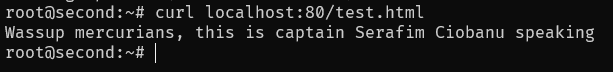
Apt install -y apache2

Service apache2 start

echo "Wassup mercurians, this is captain Serafim Ciobanu speaking" > /var/www/html/test.html

apt install curl

1. **Q: Now verify it works from within your debian container with a ‘curl localhost:80/test.html’ . Afterwards do a ‘curl localhost:80/test.html’ from your alpine container and your debian host. Do these also work?**

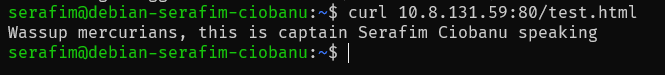
****

**Just so I do not forget about it.**

**A screen shot of a computer

Description automatically generated**

1. **Q: Can you curl the test.html of the debian second container from within the alpine first container if you use the internal private IP addresses?**



From host It works.

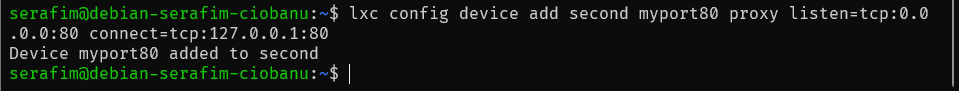
A black screen with white text

Description automatically generated

From alpine it also works, wow.

1. If you’d like to make the Apache web server in the container available on a port on your debian host (similar to port mapping in docker), you’ll have to create a lxc proxy device on the container. This can be done with:

lxc config device add second myport80 proxy listen=tcp:0.0.0.0:80 connect=tcp:127.0.0.1:80



1. Verify that browsing from your laptop host to your debian VM is working.

A screenshot of a computer

Description automatically generated

Magnificent.

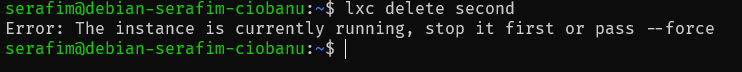
1. Once more, inspect ps -ef --forest and look at the processes associated with the containers. As which users are these processes running?

A black screen with white text

Description automatically generated

This ones are running as root, and then like child processes.

1. Try deleting a running container, you will notice you will have to stop them first. Stop both containers.



A screenshot of a computer screen

Description automatically generated

Incus is a community fork of LXD which will be used in Debian 13 (trixie) instead. It’s not available in Debian 12 in the standard repositories. However, it is available via ‘backports’. Backports are recompiled packages from testing (mostly) and unstable (in a few cases only, e.g. security updates), so they will run without new libraries (wherever it is possible) on a stable Debian distribution. This allows you to install new software which hasn’t made it into Debian stable yet (as Debian prefers stability over early adoption).

Let’s try incus:

1. Return to the clean state (no docker/podman/lxd installed) by using snapshots.
2. In /etc/apt/sources.list, add the following line:

deb http://deb.debian.org/debian bookworm-backports main

1. Now, perform an apt update to load/process the changes for your package manager
2. We can now install incus by explicitly specifying the backports repository:

apt install -t bookworm-backports incus

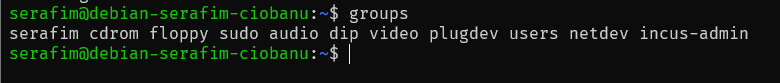
1. As with LXD, perform an initial configuration: incus admin init

A computer screen shot of a computer program

Description automatically generated

1. To allow your regular user to execute incus commands, add it to the incus-admin group. Logout and login to your shell to ensure group membership is effective for your session.

sudo usermod -aG incus-admin $USER



1. Now verify you can to everything with incus as you could with lxd, e.g.
   1. incus image list images:
   2. incus list
   3. run the same alpine and debian container as before

incus start <name>

* 1. …

A screenshot of a computer program

Description automatically generated