徽标

描述已自动生成

**IT.3503 - Architecture Virtualisée**

TP 1: Linux Containers in Practice: a Docker flavor

GUO Xiaofan

YIN Chenghao

CONTENT

[1. Environment Setup 3](#_Toc180889950)

[1. Linux Namespaces, Cgroups & Docker 3](#_Toc180889951)

[2. Install Docker Engine 4](#_Toc180889952)

[3. Install Docker Compose 5](#_Toc180889953)

[4. Docker CLI 5](#_Toc180889954)

[2. What is a container? 7](#_Toc180889955)

[1. Containers & Processes 7](#_Toc180889956)

[2. Containers & Namespaces 10](#_Toc180889957)

[3. Containers & Linux Capabilities 12](#_Toc180889958)

[4. Linux Kernel 13](#_Toc180889959)

[5. Inspecting a container 14](#_Toc180889960)

[6. Publishing ports 15](#_Toc180889961)

[3. Docker Images 16](#_Toc180889962)

[1. Dockerfile 16](#_Toc180889963)

[2. Build the image 18](#_Toc180889964)

[4. Docker Compose 19](#_Toc180889965)

# 1. Environment Setup

## 1. Linux Namespaces, Cgroups & Docker

1. **What is Docker?**

Docker is an open platform for developing, shipping, and running applications. Docker enables us to separate our applications from our infrastructure so we can deliver software quickly. It packages applications along with their dependencies into containers, enabling them to run consistently across different environments.

1. **What are the main components of Docker?**

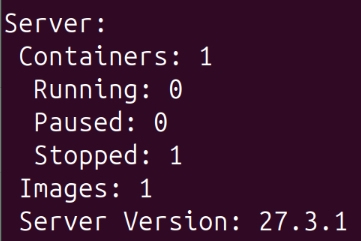
* Docker Engine: The core engine that runs Docker containers, including the client (CLI) and the server (daemon).
* Docker CLI: The command line interface for users to interact with Docker.
* Docker Daemon: A service that runs in the background, responsible for managing containers, images, networks, etc.
* Docker Images: Read-only templates for containers, containing everything needed to run an application.
* Docker Containers: Running instances of an image, including a running application and the environment it requires.
* Docker Registry: A place to store and distribute Docker images, with Docker Hub being the most common public registry.

1. **What are the technologies that Docker uses under the hood?**

* Linux Namespaces: Provide isolation of resources such as processes, networking, and file systems, ensuring that containers are isolated from each other.
* Cgroups (Control Groups): Manage and limit resources (like CPU, memory, disk I/O) for containers, ensuring proper allocation and isolation.
* Union File Systems (e.g., OverlayFS): Support layered filesystems, allowing containers to share base layers, which optimizes storage and performance.
* Container Runtime (e.g., runc): Docker uses a standard container runtime to create and manage containers.

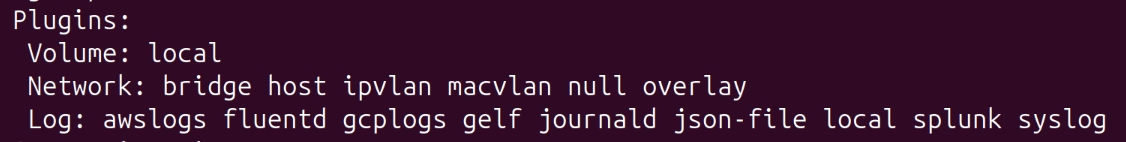
## 2. Install Docker Engine

1. **What is the Docker server (daemon) version?**



The version of Docker server is 27.3.1

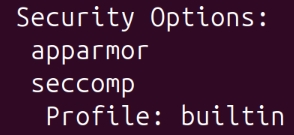
1. **What are the supported networking plugins?**



It supports the bridge, host, ipvlan, macvlan, null, overlay plugins.

* bridge: The default networking mode, which creates an internal private network for containers to communicate with each other.
* host: Shares the host’s network stack with the container, allowing it to access the same network interfaces.
* ipvlan: Provides Layer 2 (L2) or Layer 3 (L3) network isolation for containers, suitable for more complex network topologies.
* macvlan: Allows containers to have their own MAC addresses, useful for scenarios requiring high network isolation.
* null: Disable container networking
* overlay: Used for cross-host container networks, often used in multi-host cluster environments.

1. **Does Docker use SELinux? If not, what are the supported tools?**



Docker does not use SELinux.

Instead, it uses apparmor and seccomp as security options.

## 3. Install Docker Compose

**1. What is Docker Compose?**



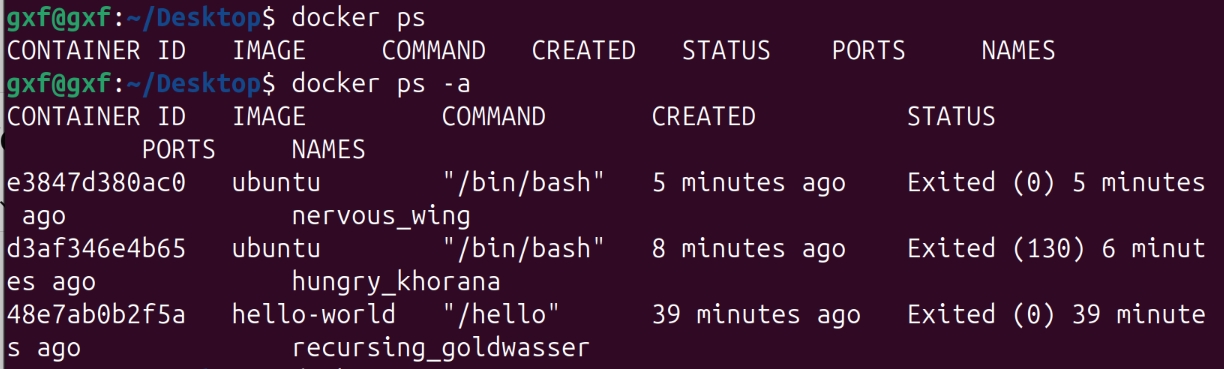
Docker Compose is a tool for defining and running multi-container Docker applications using a YAML file (docker-compose.yml).

It allows users to describe services, networks, and volumes in a single file, making it easy to orchestrate and manage complex applications with just a few commands.

## 4. Docker CLI

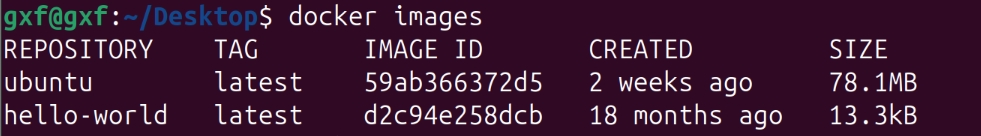
1. **What are the CLI commands that can give you:**
   1. **the list of the running containers**

*docker*



* 1. **the list of available container images**

*docker images*



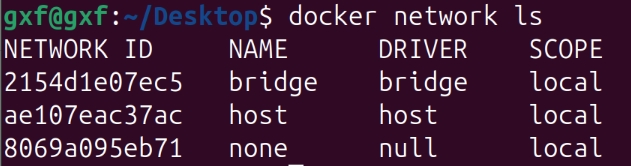
* 1. **some container statistics (CPU, RAM, I/O, etc.)**

*docker stats*



* 1. **the list of networks created by default**

*docker network ls*



1. **What is the command that can let you execute a command inside a running container?**

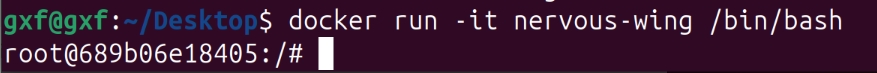
*docker exec -it <container\_name\_or\_id> <command>*

For example, open an interactive bash shell inside the container named nervous-wing and exit:

*docker exec -it nervous-wing bash*

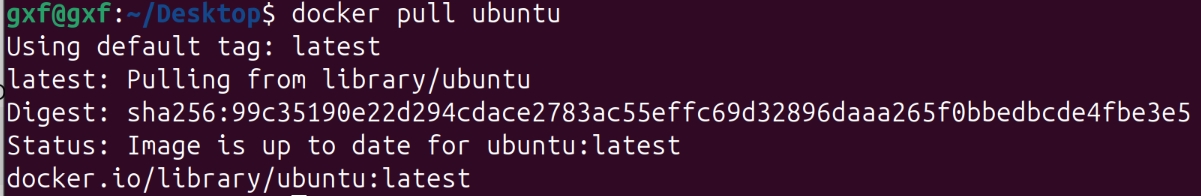


If can’t use *exec* to open, can use *run*:



1. **What is the command that can let you download a container image?**

*docker pull <image\_name>*



# 2. What is a container?

## 1. Containers & Processes

文本

描述已自动生成

**1. What is the result of *ps -aef |grep httpd* now?**

Multiple httpd processes are running:

* PID 8931, 8947, 8949, 8951 are shown as httpd processes.
* Each httpd process is running in the foreground and is shown as -DFOREGROUND.

**2. What is the PID and PPID of the parent httpd process?**

* + - PID: 8931
    - PPID: 8911

**3. What can you notice about both outputs?**

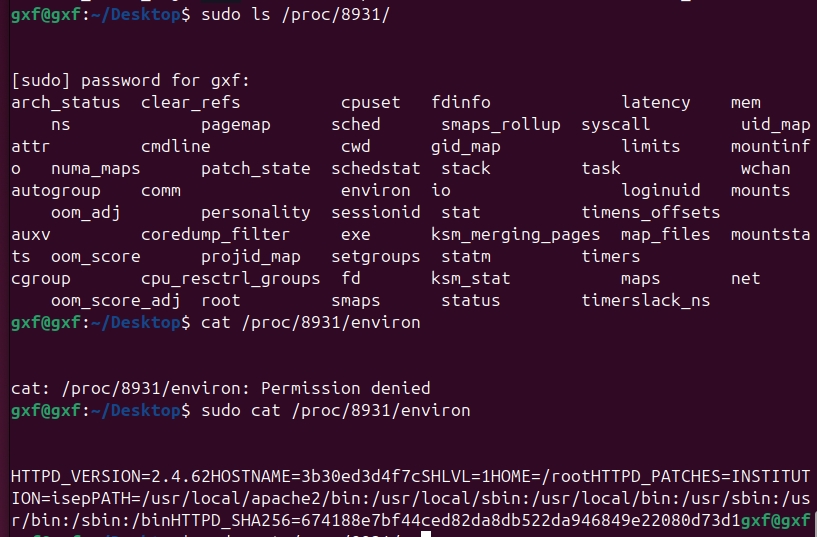
文本

中度可信度描述已自动生成

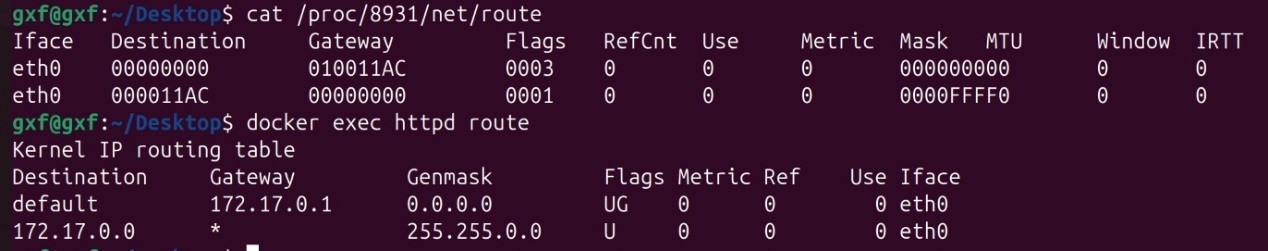
* *ps -aef*  shows all processes running on the host, including processes in containers. The PIDs of these processes are at the host system level.
* *docker top* shows the processes inside the container, which have different PIDs and PPIDs. Inside the container, the PID of the httpd process is 8931, which corresponds to the process ID inside the container.
* The processes inside the container shown by the *docker top* command are consistent with the processes shown by *ps -aef*, and the PID of the main process and the hierarchical structure of the child processes are the same.
* The processes inside the container have different PID and parent process relationships on the host, because Docker containers use independent PID namespaces.

**4. What do you notice?**





* Through the above instructions, many detailed information related to the httpd process can be seen, but some information cannot be accessed due to permission issues.
* Permission Denied: "*cat /porc/8931/environ*" displays a "Permission Denied" error. These directories represent the current working directory, root directory, and executable file path of the process. These symbolic links cannot be accessed as normal access rights. If you need to access this information, you can use sudo to elevate permissions to execute the command.



## 2. Containers & Namespaces

**1. What cinf is used for?**

*cinf* is a tool used to inspect the namespaces and other isolation features of running containers.

It allows to easily check which namespaces (such as PID, network, and mount) are being used by a container.

**2. What namespaces are used by httpd container? How many?**

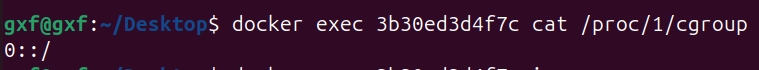
文本

描述已自动生成

A total of 6 namespaces are used:

* pid (process namespace): PID namespace is used to isolate process IDs, so that the process ID inside the container is different from that of the host.
* user (user namespace): User namespace provides independent user and group IDs for different containers.
* mnt (mount namespace): used to isolate file system mount points, so that each container has an independent mount point view.
* net (network namespace): isolates network interfaces, so that the network inside the container is isolated from the host.
* ipc (inter-process communication namespace): used to isolate inter-process communication resources of containers, such as semaphores.
* uts (host name and domain namespace): isolates host names and domain names.

**3. What is the version of cgroups used by this container? Justify whether it's v1 or v2.**



The container is using cgroups v1, confirmed by the output of /proc/1/cgroup.

## 3. Containers & Linux Capabilities

**1. How process capabilities can be listed?**

The capabilities of a process can be listed by reading the */proc/<PID>/status* file.

For example:



**2. What are the permitted capabilities of the httpd container?**

文本

描述已自动生成



## 4. Linux Kernel

**1. What is the Linux kernel's version of the httpd container?**

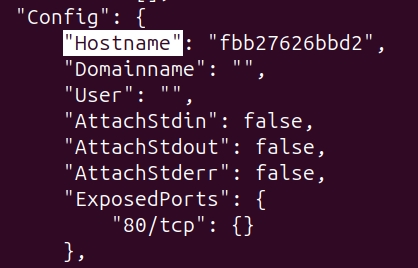


**2. What can you say about it?**

This is a relatively new Linux kernel version, especially used on Ubuntu 24.04, which brings better hardware compatibility and security enhancements.

## 5. Inspecting a container

**1. What is the Hostname of the container?**

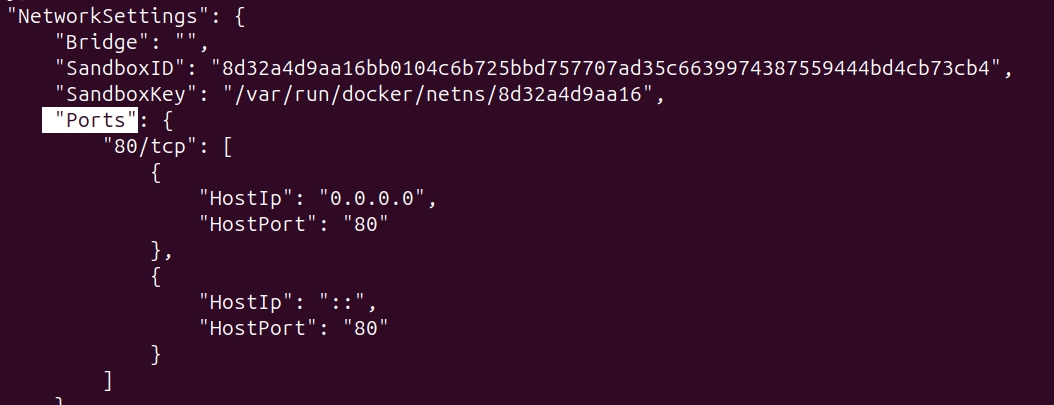


**2. What is the IP address of the container?**

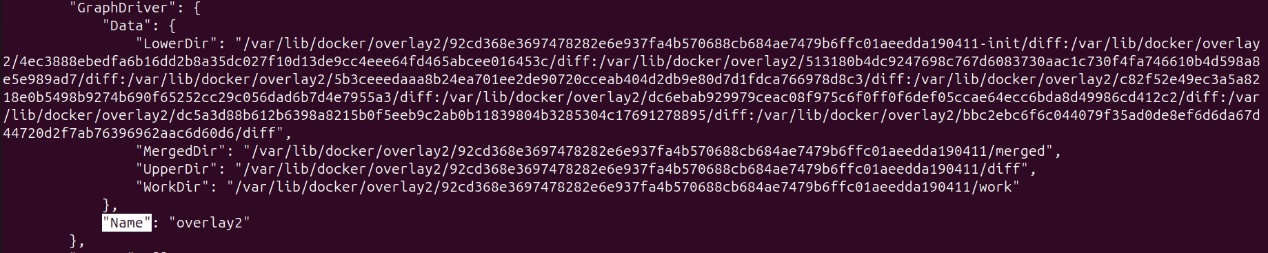


**3. Does the container open any ports? If yes, which ones?**

Yes, the open port is 80/TCP.

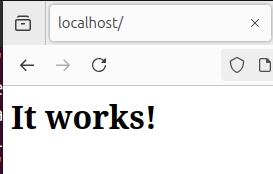


**4. What storage driver the containers uses?**



## 6. Publishing ports

**1. What is the result of the test?**



# 3. Docker Images

## 1. Dockerfile

1. **What is the role of the FROM instruction?**

The *FROM* instruction specifies the base image for creating a new Docker image. It serves as the foundation upon which subsequent layers and instructions will build.

Every Dockerfile starts with a *FROM* statement, which tells Docker which base image to use (e.g., alpine:latest, ubuntu, etc.).

1. **What is an image layer?**

An image layer is a read-only file system that adds to the base image each time a Dockerfile instruction (such as RUN, COPY, ADD) is executed. Layers are stacked on top of each other, with each new instruction adding a new layer. These layers make up the final Docker image.

1. **What is the difference between a container layer and an image layer?**

* Image layer: These are read-only layers that make up a Docker image. Each layer corresponds to an instruction in the Dockerfile and forms part of the immutable file system.
* Container layer: When a container is created from an image, a writable layer is added on top of the image layers. This writable container layer allows changes (e.g., creating files, modifying configurations) during the container's runtime, but these changes are lost when the container is destroyed unless saved explicitly.

1. **Is there any alternatives for Docker doemon to build a Docker image?**

Yes.

* Podman allows building Docker images without requiring a Docker daemon.
* Buildah can build OCI (Open Container Initiative) images without needing a running Docker daemon.

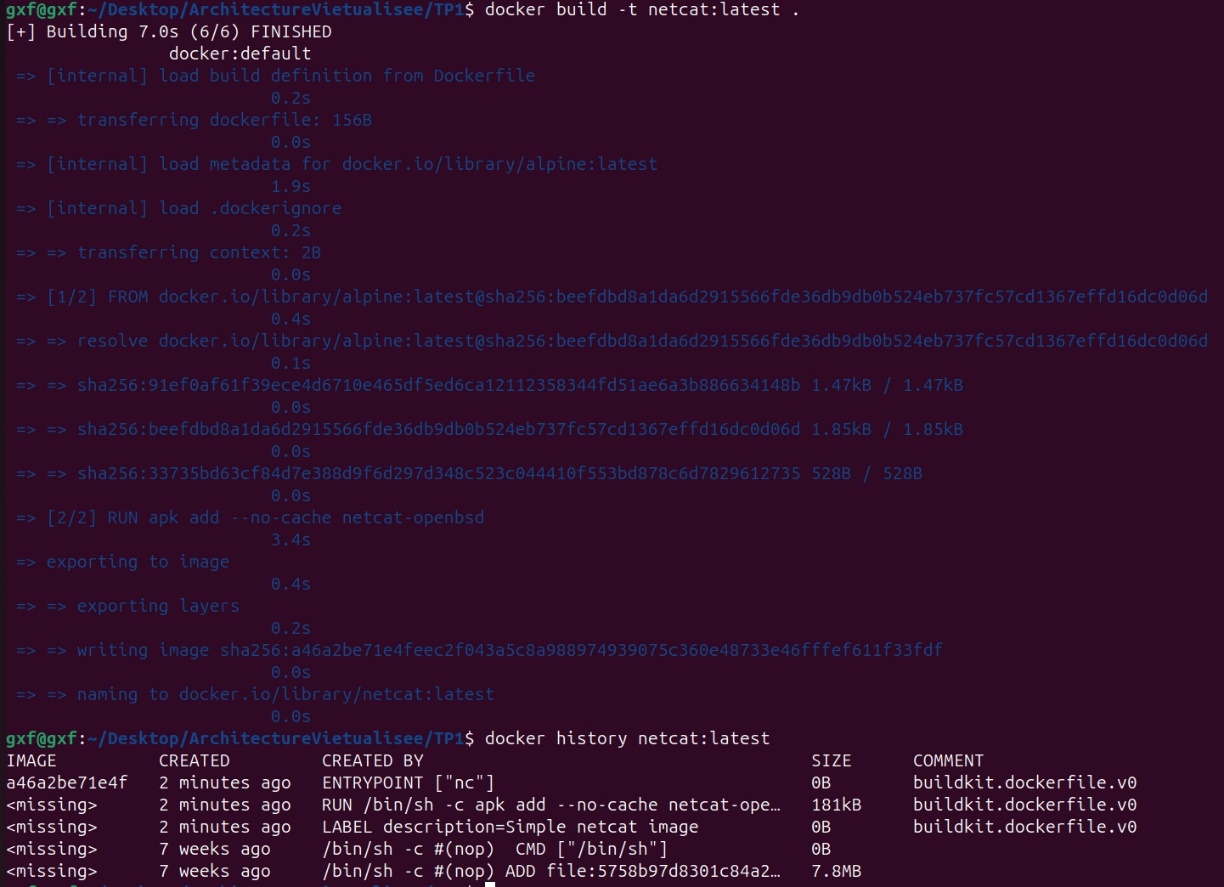
1. **What ENTRYPOINT is used for?**

The *ENTRYPOINT* instruction defines the main command that will run when a container starts. *ENTRYPOINT* cannot be overridden during the container's runtime, making it more suitable for defining the main application that should always run in the container.

In this case, the *ENTRYPOINT [ "nc" ]* ensures that *netcat* is always executed when the container starts.

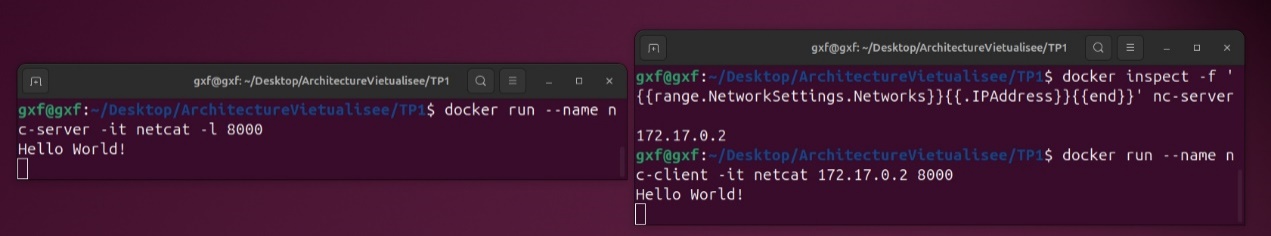
## 2. Build the image

**1. How many layers your netcat:latest image contains ? Explain why?**



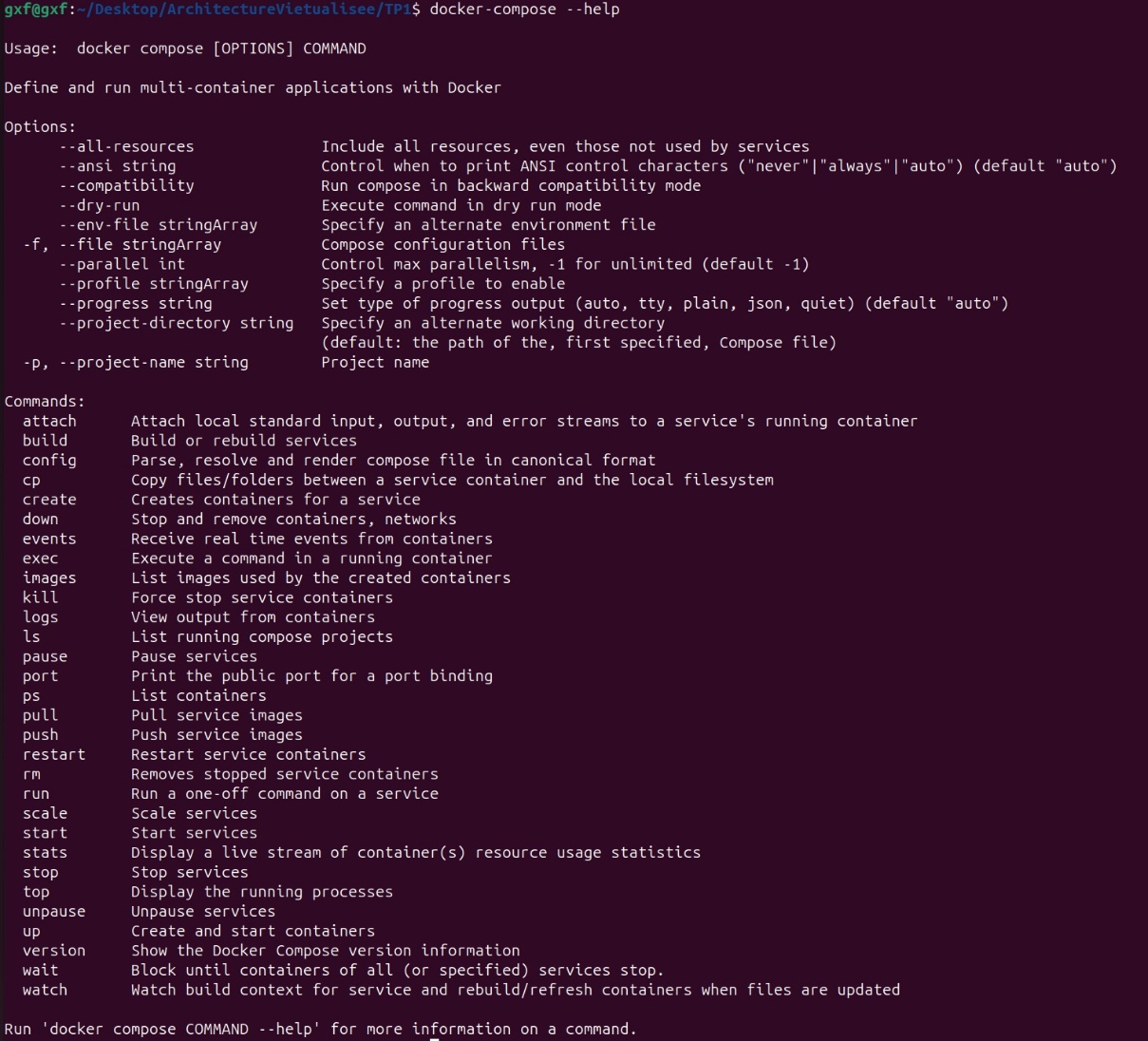
* The *netcat:latest* image has 4 layers in total.
* Each layer is generated based on the instructions in the *Dockerfile*. In the Dockerfile, instructions such as *FROM, RUN, LABEL, ENTRYPOINT* will generate new image layers.

2. **Why nc-client was able to connect to nc-server?**

****

* The nc-client was able to connect to nc-server because both containers are part of the same Docker network.
* By default, Docker creates a bridge network that allows containers within that network to communicate with each other using their internal IP addresses.

# 4. Docker Compose

****

**1. Which command can be used to run a service?**

*docker-compose up*

**2. Which command can be used to teardown a service?**

*docker-compose down*

**3. What does this file contain?**

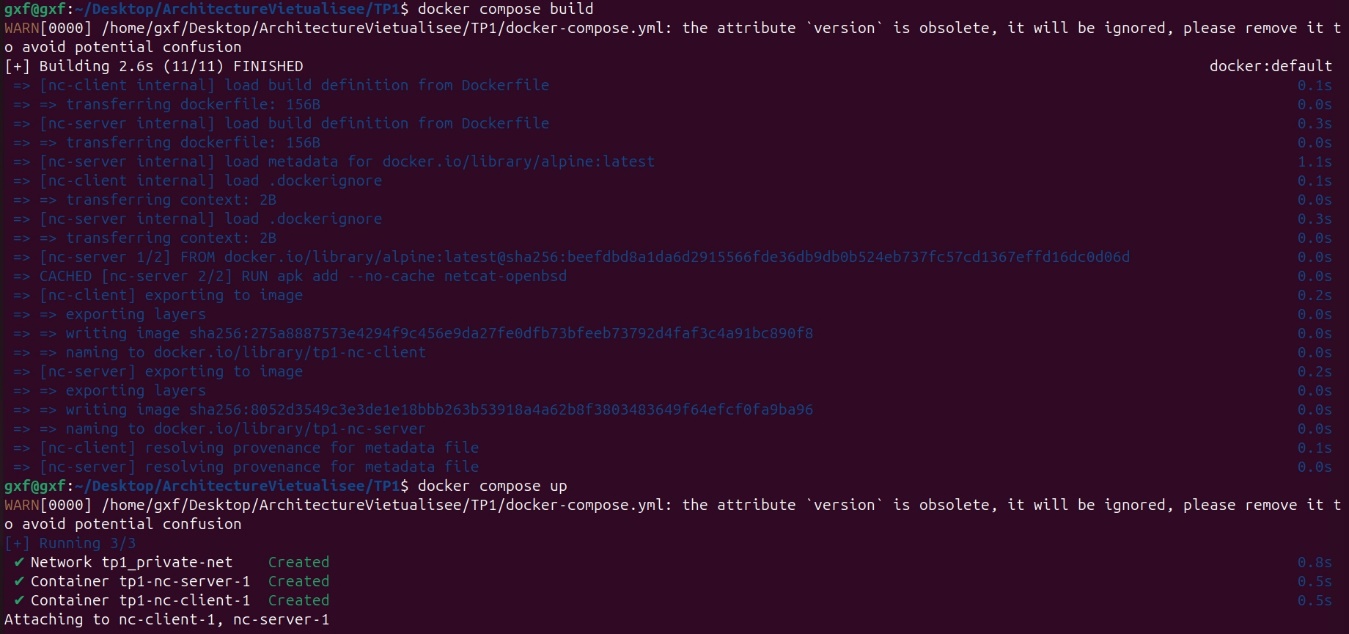
****

It defines two services (*nc-server* and *nc-client*) that will be part of a private network called *private-net.* Both services use a custom build with an entrypoint of *sleep 60*, meaning the containers will pause for 60 seconds upon starting.

**4. What part of the default image is overriden ?**

The *entrypoint* is overridden. Instead of running the default command in the image, the containers are instructed to run *sleep 60*, which temporarily pauses their execution for 60 seconds.

**5. What are the containers that are created by this compose file in the running containers' list?**

****

* *nc-server* & *nc-client*