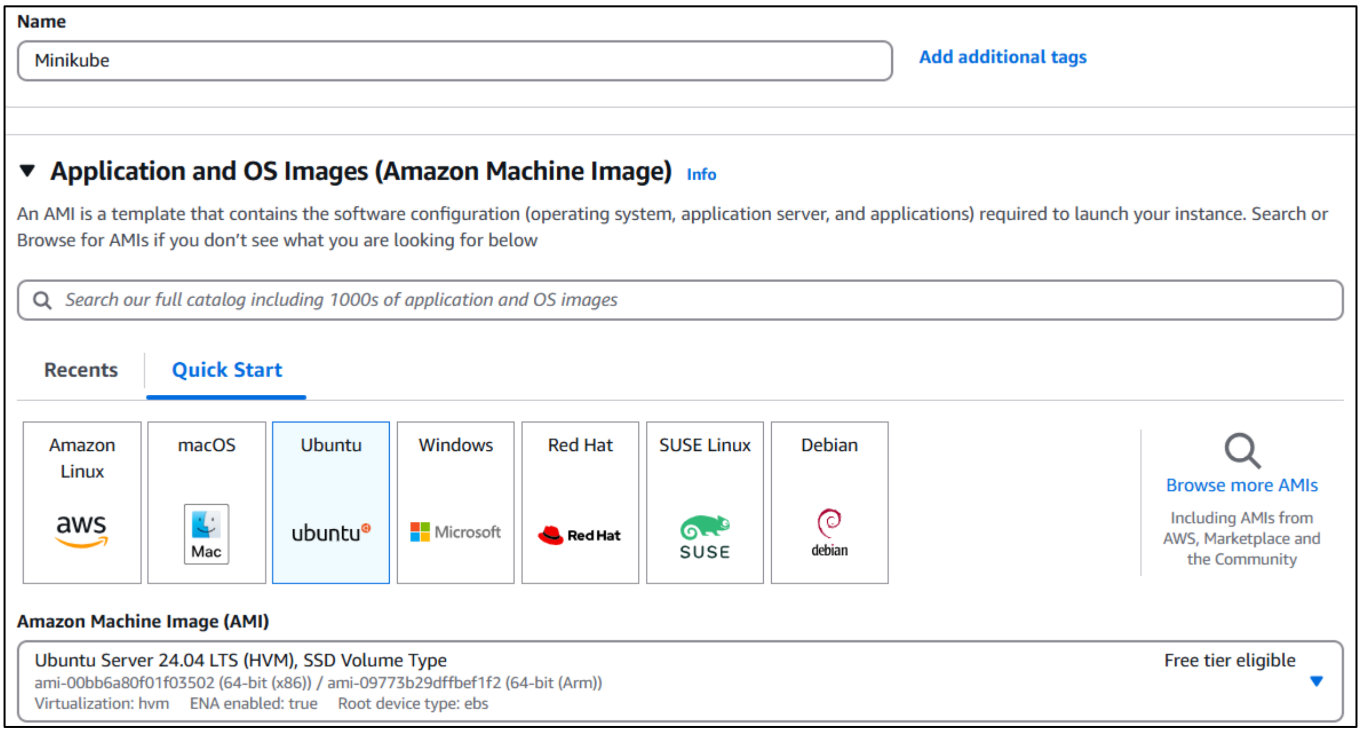
Execution of Docker Compose Concepts and Networking Concepts.

- Launch an EC2 Instance
    - **Enter the Name of the Instance**, eg: DockerCompose
    - Choose **Ubuntu Server 24.04 LTS (HVM)** under **Amazon Machine Image(AMI)**

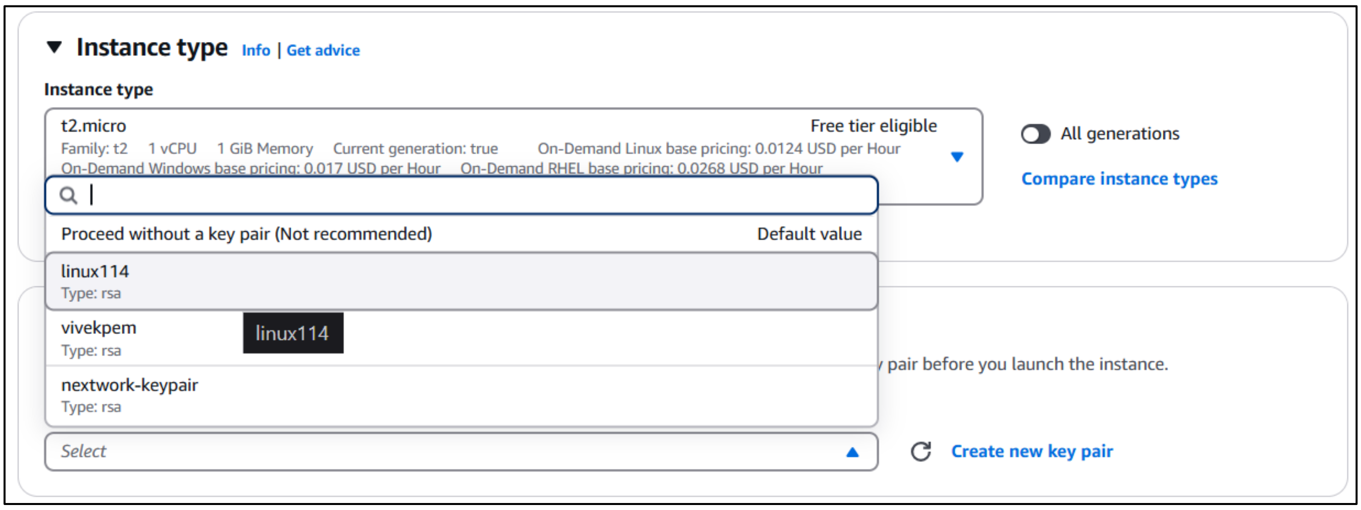

Name
Add additional tags
Application and OS Images (Amazon Machine Image)
Info
Search our full catalog including 1000s of application and OS images
Recents
Quick Start
Amazon Linux
macOS
Ubuntu
Windows
Red Hat
SUSE Linux
Debian
Browse more AMIs
Including AMIs from AWS, Marketplace and the Community
Amazon Machine Image (AMI)
Free tier eligible

Name: Minikube

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. Search or Browse for AMIs if you don't see what you are looking for below

Ubuntu Server 24.04 LTS (HVM), SSD Volume Type
ami-00bb6a80f01f03502 (64-bit (x86)) / ami-09773b29dffbef1f2 (64-bit (Arm))
Virtualization: hvm   ENA enabled: true   Root device type: ebs

-
    - Choose **t2.micro** under **Instance type**.
    - Under **Key pair (login)**, give your key pair name eg:linux114 is my keypair.

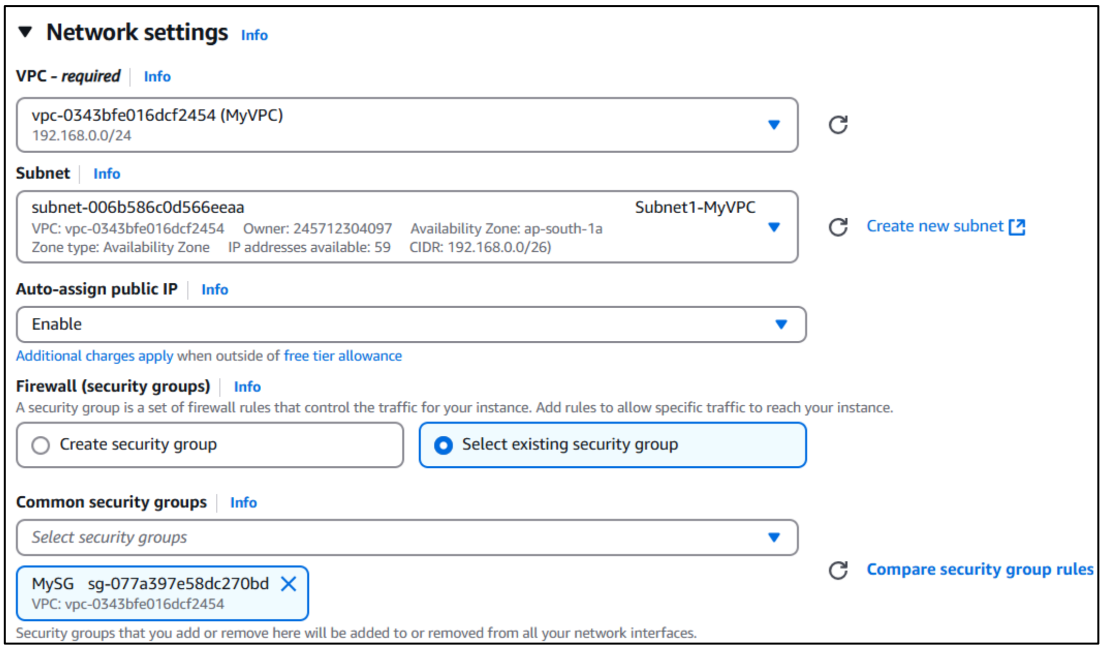


Instance type: t2.micro — Free tier eligible
Family: t2   1 vCPU   1 GiB Memory   Current generation: true   On-Demand Linux base pricing: 0.0124 USD per Hour
On-Demand Windows base pricing: 0.017 USD per Hour   On-Demand RHEL base pricing: 0.0268 USD per Hour

Proceed without a key pair (Not recommended) — Default value
linux114 — Type: rsa
vivekpem — Type: rsa
nextwork-keypair — Type: rsa

pair before you launch the instance.

- Back to our EC2 instance setup, head to the **Network settings** section and click **Edit.**
- Give Your customised VPC and Public Subnet, **Enable** Auto Assign Public IP and Select your own customised Security Group.



Network settings Info

VPC - *required* Info
vpc-0343bfe016dcf2454 (MyVPC)
192.168.0.0/24

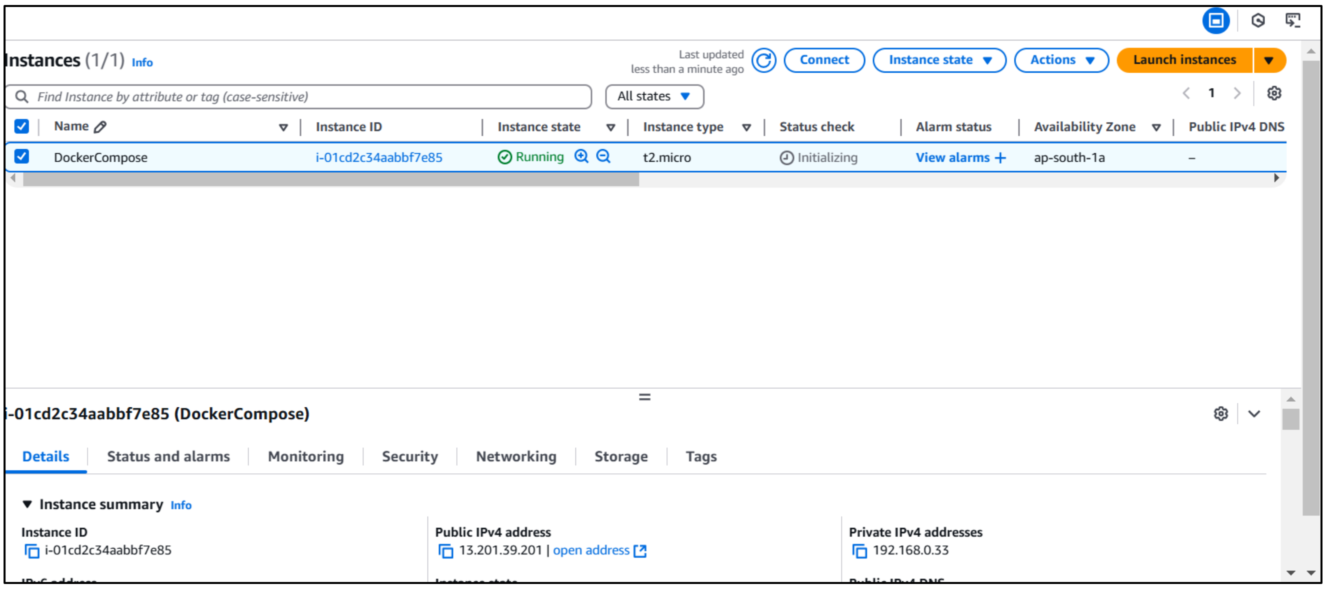
Subnet Info
subnet-006b586c0d566eeaa Subnet1-MyVPC
VPC: vpc-0343bfe016dcf2454 Owner: 245712304097 Availability Zone: ap-south-1a
Zone type: Availability Zone IP addresses available: 59 CIDR: 192.168.0.0/26)

Auto-assign public IP Info
Enable
Additional charges apply when outside of free tier allowance

Firewall (security groups) Info
A security group is a set of firewall rules that control the traffic for your instance. Add rules to allow specific traffic to reach your instance.
Create security group [✓ Select existing security group]

Common security groups Info
Select security groups
MySG sg-077a397e58dc270bd
VPC: vpc-0343bfe016dcf2454
Security groups that you add or remove here will be added to or removed from all your network interfaces.

- choose **Launch instance**
- Connect to Instance via Putty (**note: login as : ubuntu**)



Instances (1/1) Info

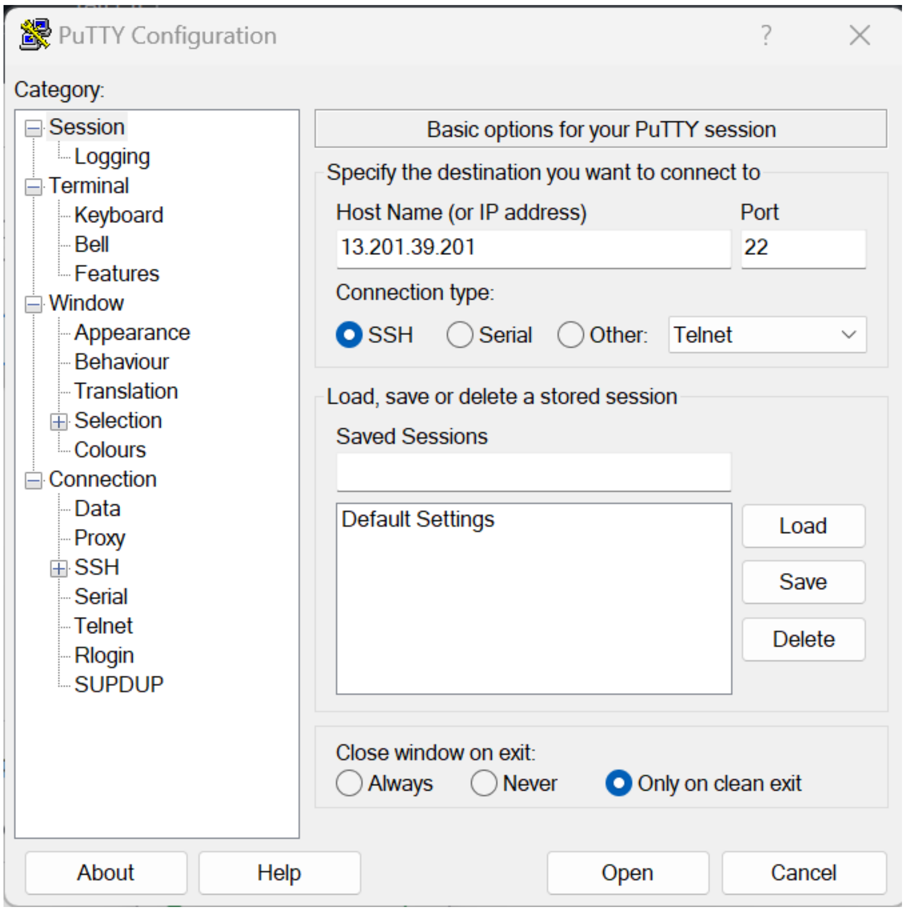
Last updated less than a minute ago

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS
DockerCompose	i-01cd2c34aabbf7e85	Running	t2.micro	Initializing	View alarms +	ap-south-1a	–

i-01cd2c34aabbf7e85 (DockerCompose)

Instance summary Info

Instance ID: i-01cd2c34aabbf7e85
Public IPv4 address: 13.201.39.201 | open address
Private IPv4 addresses: 192.168.0.33


PuTTY Configuration

Basic options for your PuTTY session

Specify the destination you want to connect to

Host Name (or IP address): 13.201.39.201

Port: 22

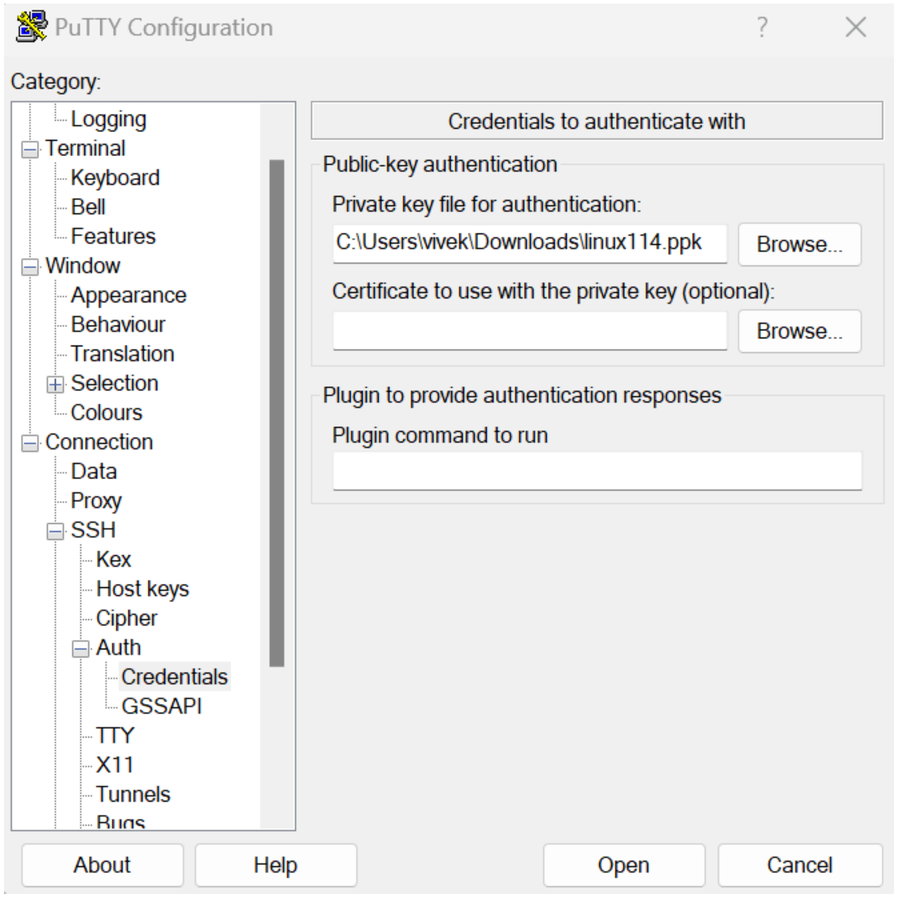
Connection type: [✓ SSH] Serial Other: Telnet

Load, save or delete a stored session

Saved Sessions

Default Settings

Close window on exit: Always Never [✓ Only on clean exit]


PuTTY Configuration

Credentials to authenticate with

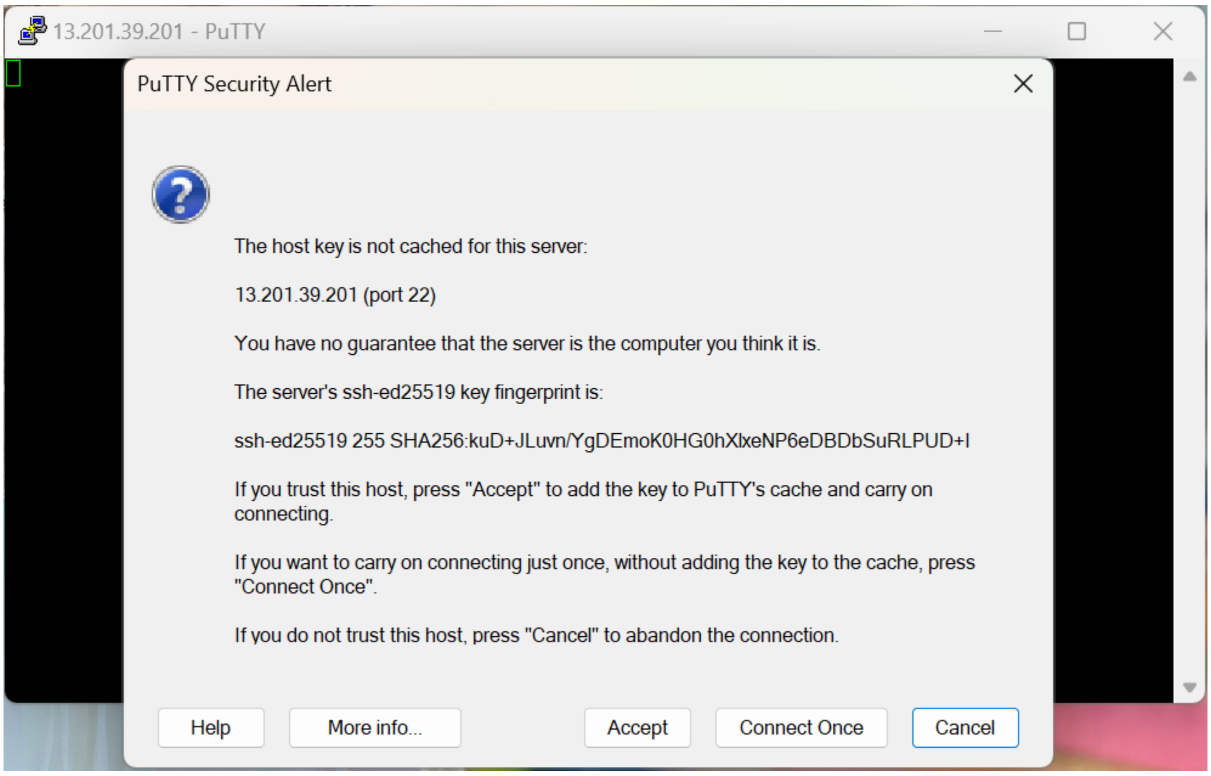
Public-key authentication

Private key file for authentication: C:\Users\vivek\Downloads\linux114.ppk

Certificate to use with the private key (optional):

Plugin to provide authentication responses

Plugin command to run



The host key is not cached for this server:

13.201.39.201 (port 22)

You have no guarantee that the server is the computer you think it is.

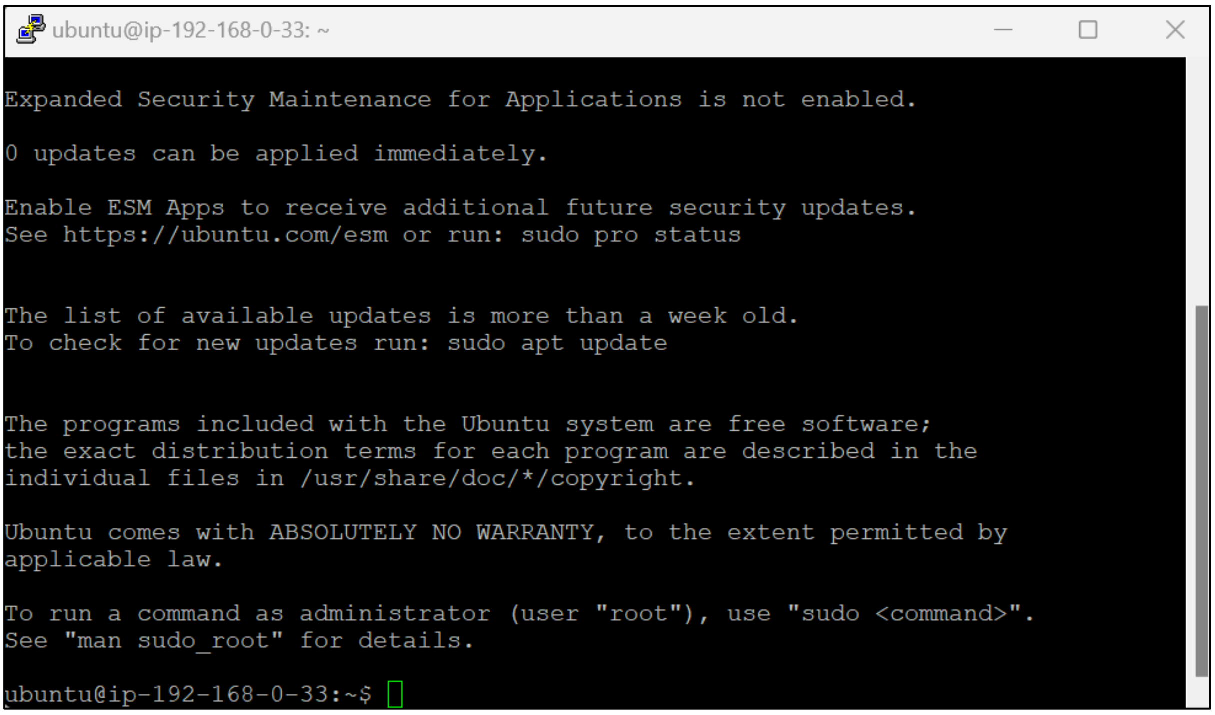
The server's ssh-ed25519 key fingerprint is:

ssh-ed25519 255 SHA256:kuD+JLuvn/YgDEmoK0HG0hXlxeNP6eDBDbSuRLPUD+I

If you trust this host, press "Accept" to add the key to PuTTY's cache and carry on connecting.

If you want to carry on connecting just once, without adding the key to the cache, press "Connect Once".

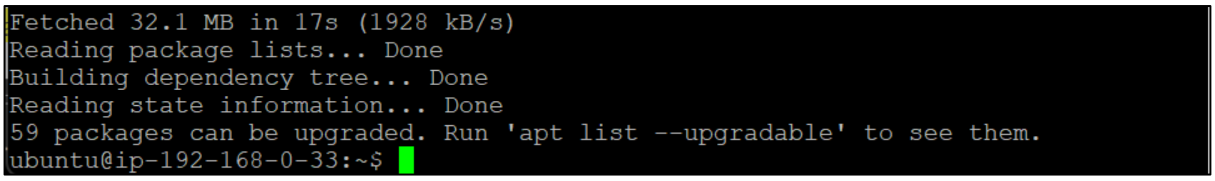
If you do not trust this host, press "Cancel" to abandon the connection.



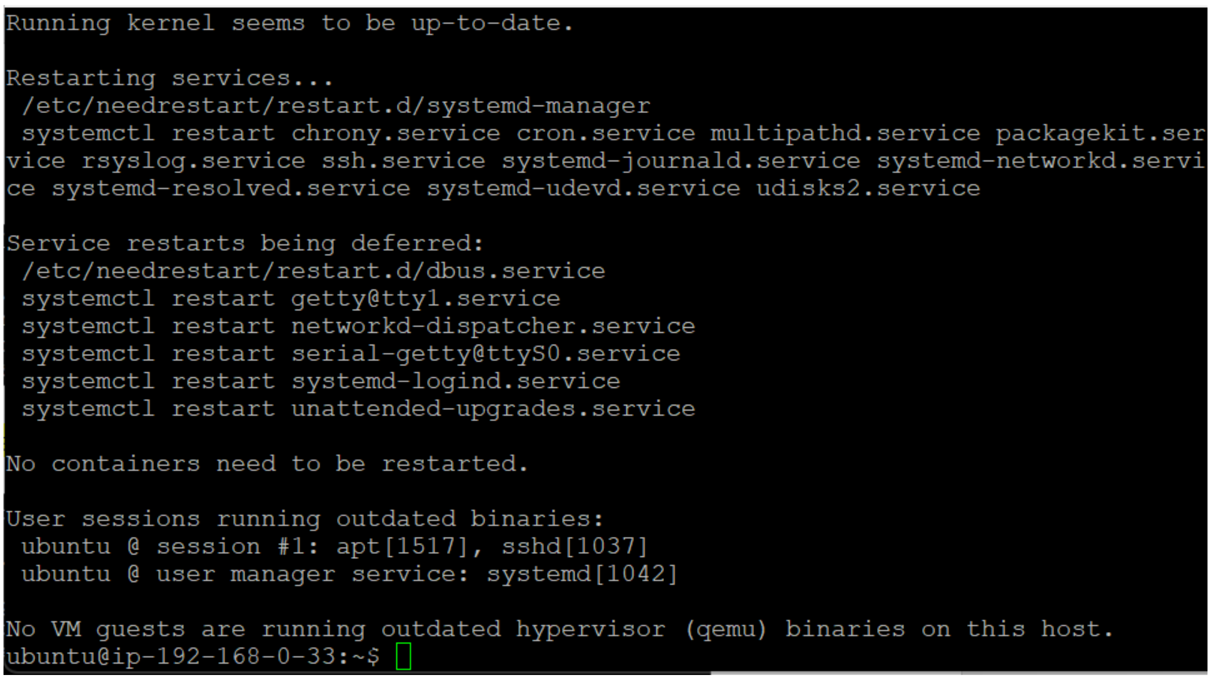
```
Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status


The list of available updates is more than a week old.
To check for new updates run: sudo apt update


The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ip-192-168-0-33:~$
```

- Now that we have connected to our EC2 Instance, lets install docker in the EC2.
- The commands for installation of docker are as follows:

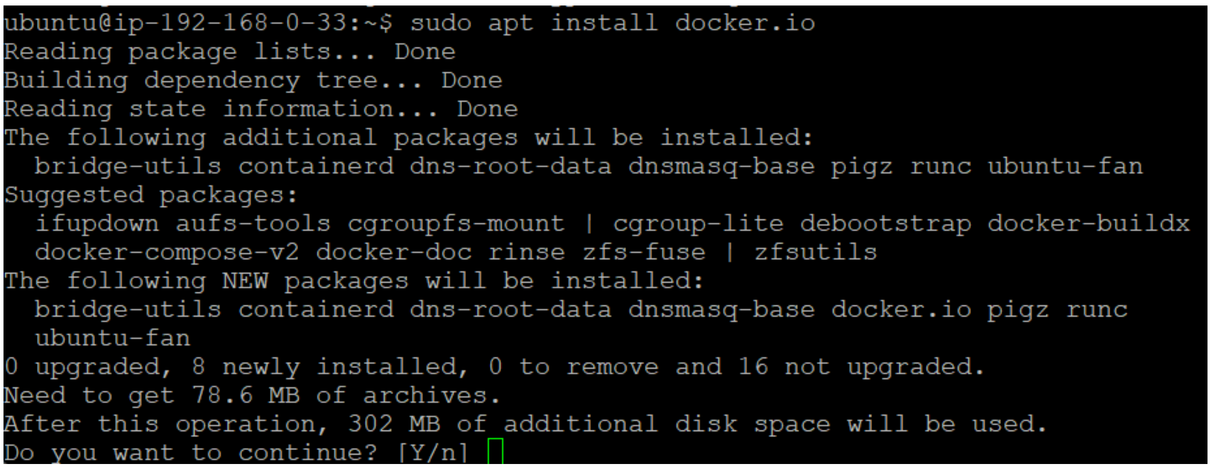
  sudo apt update


```
Fetched 32.1 MB in 17s (1928 kB/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
59 packages can be upgraded. Run 'apt list --upgradable' to see them.
ubuntu@ip-192-168-0-33:~$
```

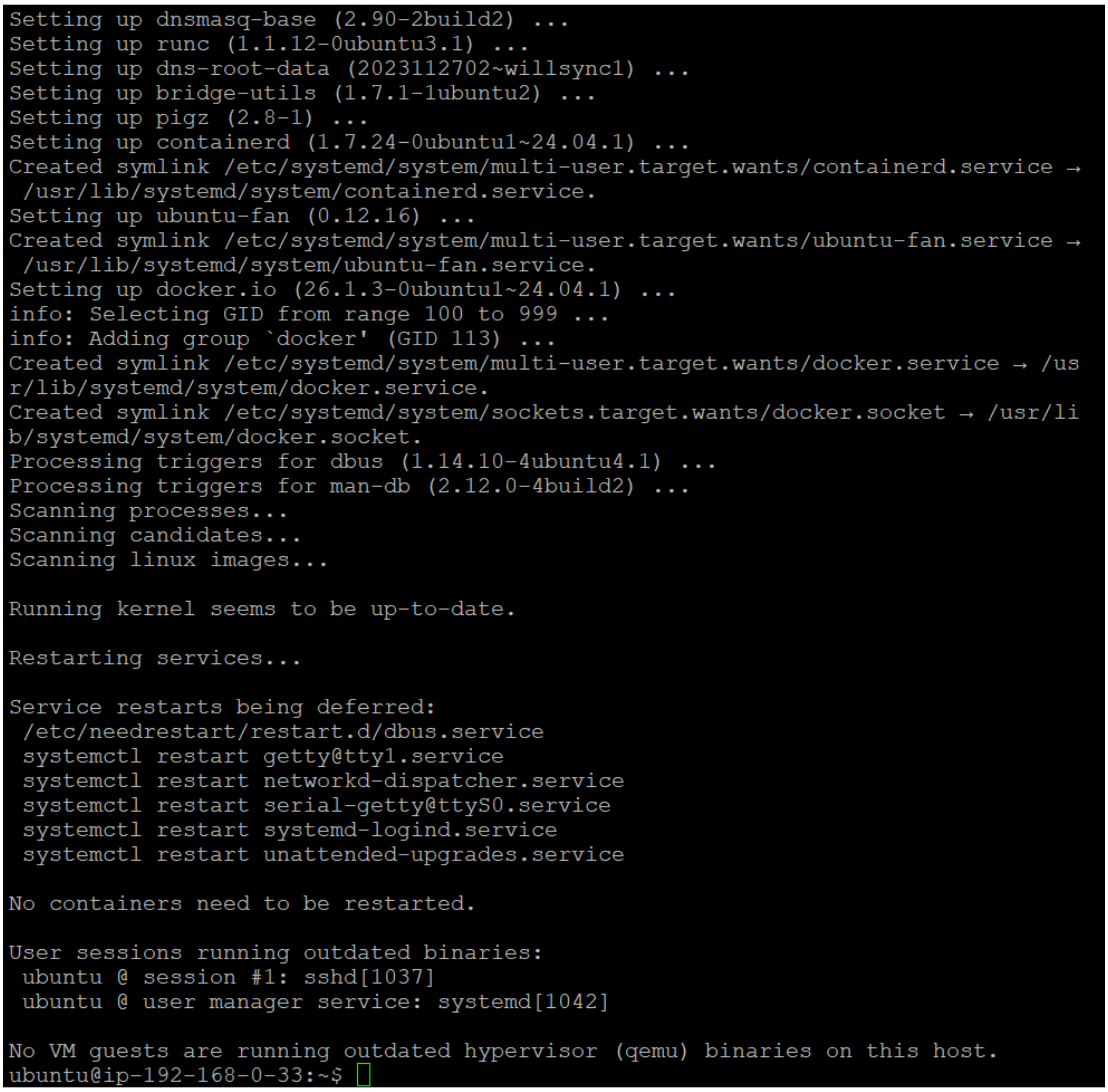
sudo apt upgrade -y


```
Running kernel seems to be up-to-date.

Restarting services...
 /etc/needrestart/restart.d/systemd-manager
 systemctl restart chrony.service cron.service multipathd.service packagekit.ser
vice rsyslog.service ssh.service systemd-journald.service systemd-networkd.servi
ce systemd-resolved.service systemd-udevd.service udisks2.service

Service restarts being deferred:
 /etc/needrestart/restart.d/dbus.service
 systemctl restart getty@tty1.service
 systemctl restart networkd-dispatcher.service
 systemctl restart serial-getty@ttyS0.service
 systemctl restart systemd-logind.service
 systemctl restart unattended-upgrades.service

No containers need to be restarted.

User sessions running outdated binaries:
 ubuntu @ session #1: apt[1517], sshd[1037]
 ubuntu @ user manager service: systemd[1042]

No VM guests are running outdated hypervisor (qemu) binaries on this host.
ubuntu@ip-192-168-0-33:~$
```

sudo apt install docker.io

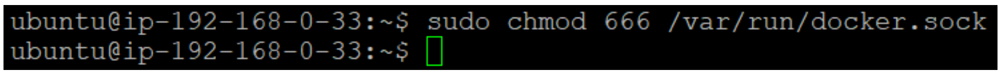

```
ubuntu@ip-192-168-0-33:~$ sudo apt install docker.io
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  bridge-utils containerd dns-root-data dnsmasq-base pigz runc ubuntu-fan
Suggested packages:
  ifupdown aufs-tools cgroupfs-mount | cgroup-lite debootstrap docker-buildx
  docker-compose-v2 docker-doc rinse zfs-fuse | zfsutils
The following NEW packages will be installed:
  bridge-utils containerd dns-root-data dnsmasq-base docker.io pigz runc
  ubuntu-fan
0 upgraded, 8 newly installed, 0 to remove and 16 not upgraded.
Need to get 78.6 MB of archives.
After this operation, 302 MB of additional disk space will be used.
Do you want to continue? [Y/n]
```

Type Y



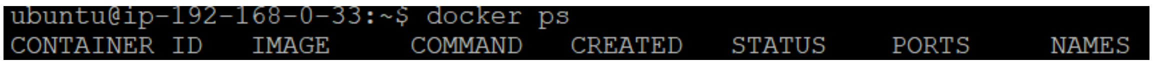
```
Setting up dnsmasq-base (2.90-2build2) ...
Setting up runc (1.1.12-0ubuntu3.1) ...
Setting up dns-root-data (2023112702~willsync1) ...
Setting up bridge-utils (1.7.1-1ubuntu2) ...
Setting up pigz (2.8-1) ...
Setting up containerd (1.7.24-0ubuntu1~24.04.1) ...
Created symlink /etc/systemd/system/multi-user.target.wants/containerd.service →
 /usr/lib/systemd/system/containerd.service.
Setting up ubuntu-fan (0.12.16) ...
Created symlink /etc/systemd/system/multi-user.target.wants/ubuntu-fan.service →
 /usr/lib/systemd/system/ubuntu-fan.service.
Setting up docker.io (26.1.3-0ubuntu1~24.04.1) ...
info: Selecting GID from range 100 to 999 ...
info: Adding group `docker' (GID 113) ...
Created symlink /etc/systemd/system/multi-user.target.wants/docker.service → /us
r/lib/systemd/system/docker.service.
Created symlink /etc/systemd/system/sockets.target.wants/docker.socket → /usr/li
b/systemd/system/docker.socket.
Processing triggers for dbus (1.14.10-4ubuntu4.1) ...
Processing triggers for man-db (2.12.0-4build2) ...
Scanning processes...
Scanning candidates...
Scanning linux images...

Running kernel seems to be up-to-date.

Restarting services...

Service restarts being deferred:
 /etc/needrestart/restart.d/dbus.service
 systemctl restart getty@tty1.service
 systemctl restart networkd-dispatcher.service
 systemctl restart serial-getty@ttyS0.service
 systemctl restart systemd-logind.service
 systemctl restart unattended-upgrades.service

No containers need to be restarted.

User sessions running outdated binaries:
 ubuntu @ session #1: sshd[1037]
 ubuntu @ user manager service: systemd[1042]

No VM guests are running outdated hypervisor (qemu) binaries on this host.
ubuntu@ip-192-168-0-33:~$
```

sudo chmod 666 /var/run/docker.sock



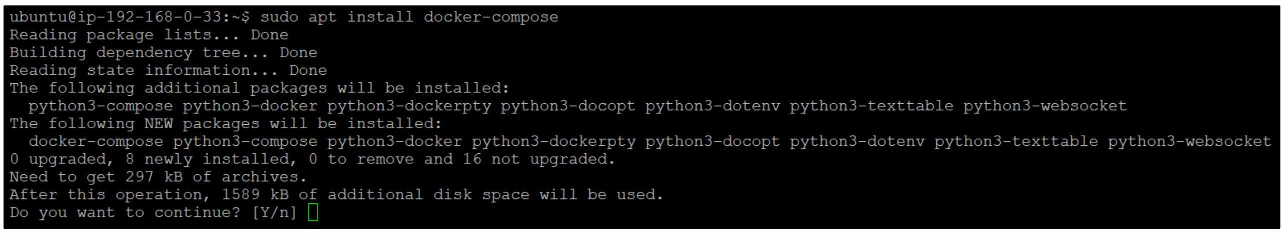
```
ubuntu@ip-192-168-0-33:~$ sudo chmod 666 /var/run/docker.sock
ubuntu@ip-192-168-0-33:~$
```

- Run docker ps command to see the status.



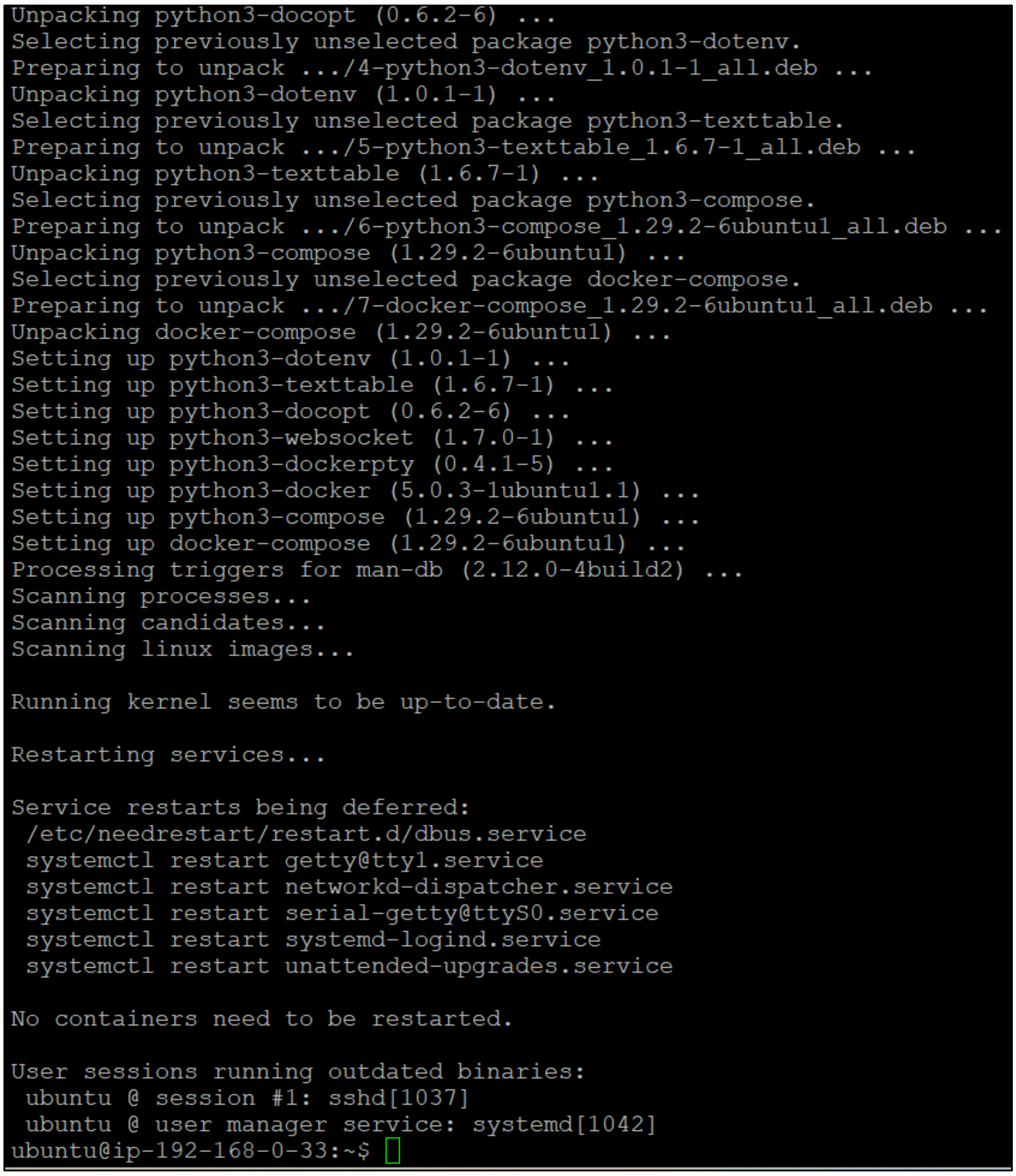
```
ubuntu@ip-192-168-0-33:~$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
```

- Run sudo apt install docker-compose to install Docker compose.



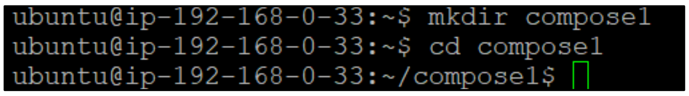
```
ubuntu@ip-192-168-0-33:~$ sudo apt install docker-compose
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  python3-compose python3-docker python3-dockerpty python3-docopt python3-dotenv python3-texttable python3-websocket
The following NEW packages will be installed:
  docker-compose python3-compose python3-docker python3-dockerpty python3-docopt python3-dotenv python3-texttable python3-websocket
0 upgraded, 8 newly installed, 0 to remove and 16 not upgraded.
Need to get 297 kB of archives.
After this operation, 1589 kB of additional disk space will be used.
Do you want to continue? [Y/n]
```

Type Y



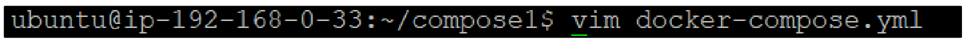
```
Unpacking python3-docopt (0.6.2-6) ...
Selecting previously unselected package python3-dotenv.
Preparing to unpack .../4-python3-dotenv_1.0.1-1_all.deb ...
Unpacking python3-dotenv (1.0.1-1) ...
Selecting previously unselected package python3-texttable.
Preparing to unpack .../5-python3-texttable_1.6.7-1_all.deb ...
Unpacking python3-texttable (1.6.7-1) ...
Selecting previously unselected package python3-compose.
Preparing to unpack .../6-python3-compose_1.29.2-6ubuntu1_all.deb ...
Unpacking python3-compose (1.29.2-6ubuntu1) ...
Selecting previously unselected package docker-compose.
Preparing to unpack .../7-docker-compose_1.29.2-6ubuntu1_all.deb ...
Unpacking docker-compose (1.29.2-6ubuntu1) ...
Setting up python3-dotenv (1.0.1-1) ...
Setting up python3-texttable (1.6.7-1) ...
Setting up python3-docopt (0.6.2-6) ...
Setting up python3-websocket (1.7.0-1) ...
Setting up python3-dockerpty (0.4.1-5) ...
Setting up python3-docker (5.0.3-1ubuntu1.1) ...
Setting up python3-compose (1.29.2-6ubuntu1) ...
Setting up docker-compose (1.29.2-6ubuntu1) ...
Processing triggers for man-db (2.12.0-4build2) ...
Scanning processes...
Scanning candidates...
Scanning linux images...

Running kernel seems to be up-to-date.

Restarting services...

Service restarts being deferred:
 /etc/needrestart/restart.d/dbus.service
 systemctl restart getty@tty1.service
 systemctl restart networkd-dispatcher.service
 systemctl restart serial-getty@ttyS0.service
 systemctl restart systemd-logind.service
 systemctl restart unattended-upgrades.service

No containers need to be restarted.

User sessions running outdated binaries:
 ubuntu @ session #1: sshd[1037]
 ubuntu @ user manager service: systemd[1042]
ubuntu@ip-192-168-0-33:~$
```

- Create a folder named **`compose1`** by running the command `mkdir compose1`
- Enter into the folder with the command `cd compose1`

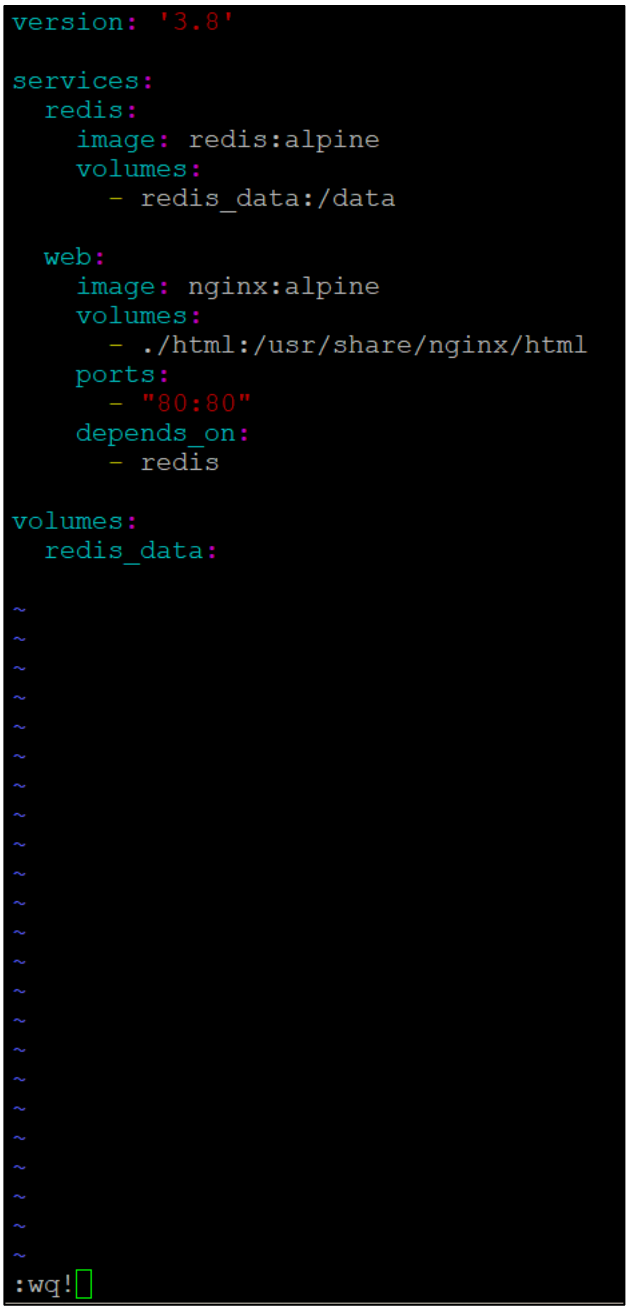


```
ubuntu@ip-192-168-0-33:~$ mkdir compose1
ubuntu@ip-192-168-0-33:~$ cd compose1
ubuntu@ip-192-168-0-33:~/compose1$
```

- Create a file named **`docker-compose.yml`** with edit access by running `vim docker-compose.yml`, then paste the available Docker Compose YAML script into the file as save the file with `:wq!` command



```
ubuntu@ip-192-168-0-33:~/compose1$ vim docker-compose.yml
```



```
version: '3.8'

services:
  redis:
    image: redis:alpine
    volumes:
      - redis_data:/data

  web:
    image: nginx:alpine
    volumes:
      - ./html:/usr/share/nginx/html
    ports:
      - "80:80"
    depends_on:
      - redis

volumes:
  redis_data:

~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
:wq!
```

The explanation for the yaml script is as follows:

1. **`version: '3.8'`**
   Specifies the version of the Docker Compose file format. In this case, it's version 3.8, which allows access to features available in that version.

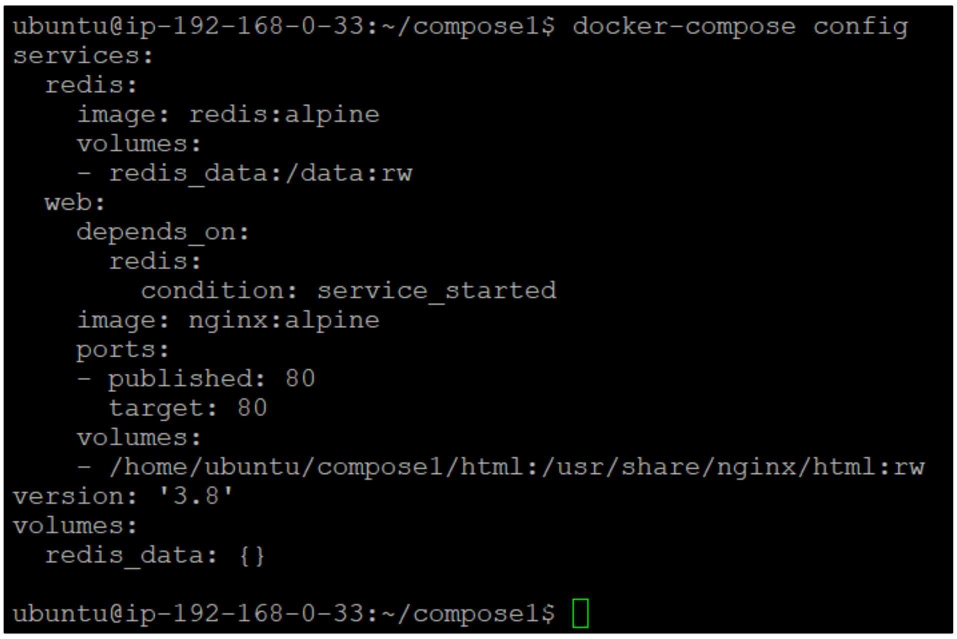
2. **`services:`**
   Defines the individual containers (services) that will be part of this multi-container setup.
3. **`redis:`**
   This is the name of the first service, which runs the Redis container.
4. **`image: redis:alpine`**
   The Docker image to use for the Redis service. The `alpine` variant is a smaller, lightweight version of Redis built on Alpine Linux.

5. **`volumes:`**
   Declares that the Redis data should be stored in a persistent volume. This mounts the `redis_data` volume to the `/data` directory inside the Redis container.
6. **`web:`**
   This is the second service, which runs the Nginx web server.
7. **`image: nginx:alpine`**
   Specifies the Docker image to use for the Nginx web server. The `alpine` variant of Nginx is also lightweight and efficient.

8. **`volumes:`**
   Mounts the local `./html` directory to `/usr/share/nginx/html` inside the container. This allows you to serve static files stored in the local `html` folder through Nginx.
9. **`ports:`**
   Maps port 80 on the host machine to port 80 inside the container. This allows external access to the web server via the host's IP on port 80.
10. **`depends_on:`**
    Ensures that the **web** service starts only after the **redis** service is up and running. This is useful if your web service depends on Redis.
11. **`volumes:`**
    Declares a named volume `redis_data` to persist Redis data. This ensures data isn't lost when the container is removed or restarted.

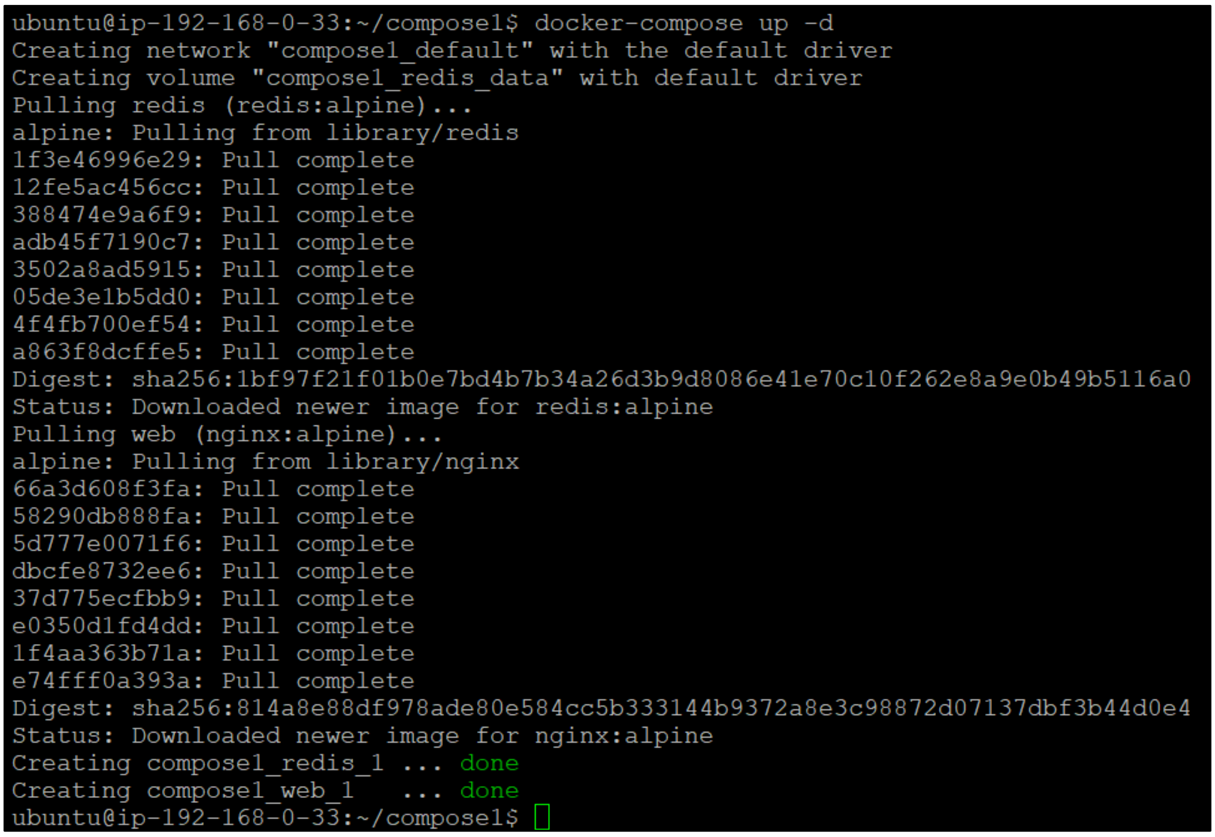
This configuration sets up a multi-container application with a **Redis** service for data persistence and an **Nginx** service to serve static files, with persistent storage for Redis and web content.

- To validate your `docker-compose.yml` file, run the command `docker-compose config`



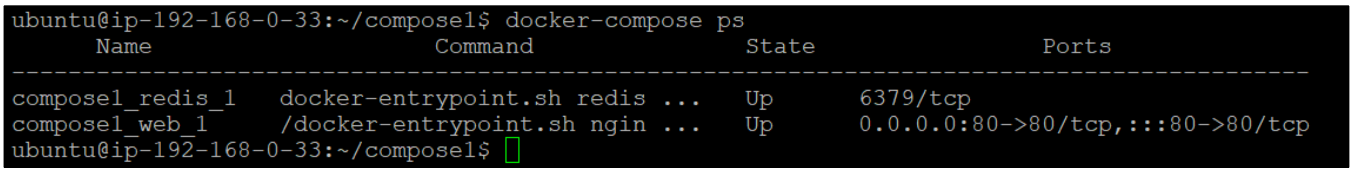
```
ubuntu@ip-192-168-0-33:~/compose1$ docker-compose config
services:
  redis:
    image: redis:alpine
    volumes:
    - redis_data:/data:rw
  web:
    depends_on:
      redis:
        condition: service_started
    image: nginx:alpine
    ports:
    - published: 80
      target: 80
    volumes:
    - /home/ubuntu/compose1/html:/usr/share/nginx/html:rw
version: '3.8'
volumes:
  redis_data: {}

ubuntu@ip-192-168-0-33:~/compose1$
```

- Run the command `docker-compose up -d` to build, create, and start the services defined in the `docker-compose.yml` file in detached mode, allowing them to run in the background while keeping the terminal available for other tasks.



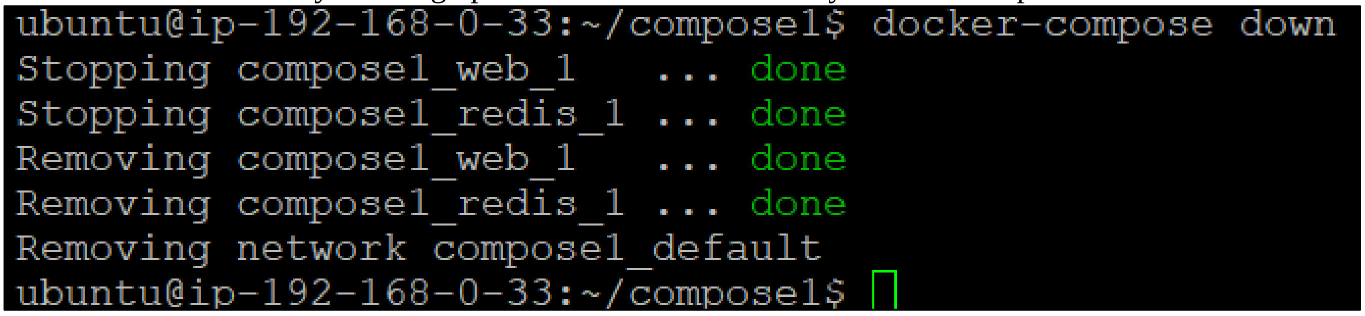
```
ubuntu@ip-192-168-0-33:~/compose1$ docker-compose up -d
Creating network "compose1_default" with the default driver
Creating volume "compose1_redis_data" with default driver
Pulling redis (redis:alpine)...
alpine: Pulling from library/redis
1f3e46996e29: Pull complete
12fe5ac456cc: Pull complete
388474e9a6f9: Pull complete
adb45f7190c7: Pull complete
3502a8ad5915: Pull complete
05de3e1b5dd0: Pull complete
4f4fb700ef54: Pull complete
a863f8dcffe5: Pull complete
Digest: sha256:1bf97f21f01b0e7bd4b7b34a26d3b9d8086e41e70c10f262e8a9e0b49b5116a0
Status: Downloaded newer image for redis:alpine
Pulling web (nginx:alpine)...
alpine: Pulling from library/nginx
66a3d608f3fa: Pull complete
58290db888fa: Pull complete
5d777e0071f6: Pull complete
dbcfe8732ee6: Pull complete
37d775ecfbb9: Pull complete
e0350d1fd4dd: Pull complete
1f4aa363b71a: Pull complete
e74fff0a393a: Pull complete
Digest: sha256:814a8e88df978ade80e584cc5b333144b9372a8e3c98872d07137dbf3b44d0e4
Status: Downloaded newer image for nginx:alpine
Creating compose1_redis_1 ... done
Creating compose1_web_1   ... done
ubuntu@ip-192-168-0-33:~/compose1$
```

- Now execute the command `docker-compose ps` to display the status of the containers for the services defined in the `docker-compose.yml` file, including container names, service associations, current states, and exposed ports.



```
ubuntu@ip-192-168-0-33:~/compose1$ docker-compose ps
      Name                    Command               State                  Ports
-------------------------------------------------------------------------------------------------
compose1_redis_1   docker-entrypoint.sh redis ...   Up      6379/tcp
compose1_web_1     /docker-entrypoint.sh ngin ...   Up      0.0.0.0:80->80/tcp,:::80->80/tcp
ubuntu@ip-192-168-0-33:~/compose1$
```

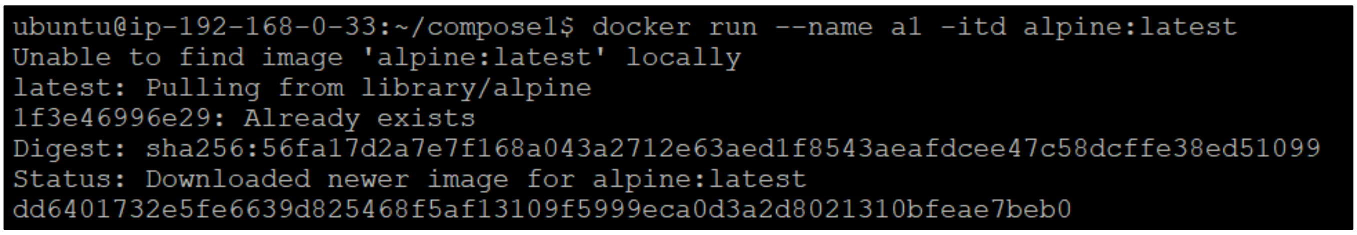
- Now execute the command `docker-compose down` to stop and remove the containers, networks, and volumes defined in the `docker-compose.yml` file, effectively cleaning up the environment created by Docker Compose.



```
ubuntu@ip-192-168-0-33:~/compose1$ docker-compose down
Stopping compose1_web_1   ... done
Stopping compose1_redis_1 ... done
Removing compose1_web_1   ... done
Removing compose1_redis_1 ... done
Removing network compose1_default
ubuntu@ip-192-168-0-33:~/compose1$
```

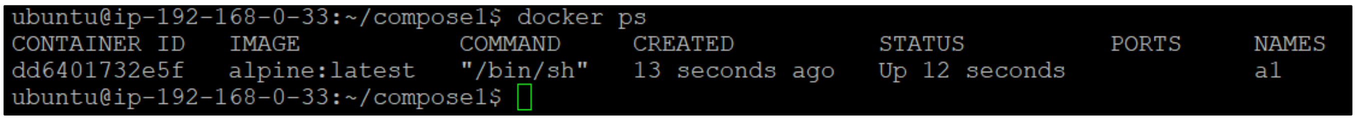
Networking Concepts:

- Execute the command `docker run --name a1 -itd alpine:latest`, this will create and run a new container named **a1** using the **Alpine Linux** image (`alpine:latest`). The `-it` flags allow for interaction with the container through an interactive terminal, while the `-d` flag runs the container in **detached mode**, meaning it will run in the background without taking up the terminal. This setup initializes a lightweight Alpine Linux container, which is useful for running simple tasks or creating a testing environment.



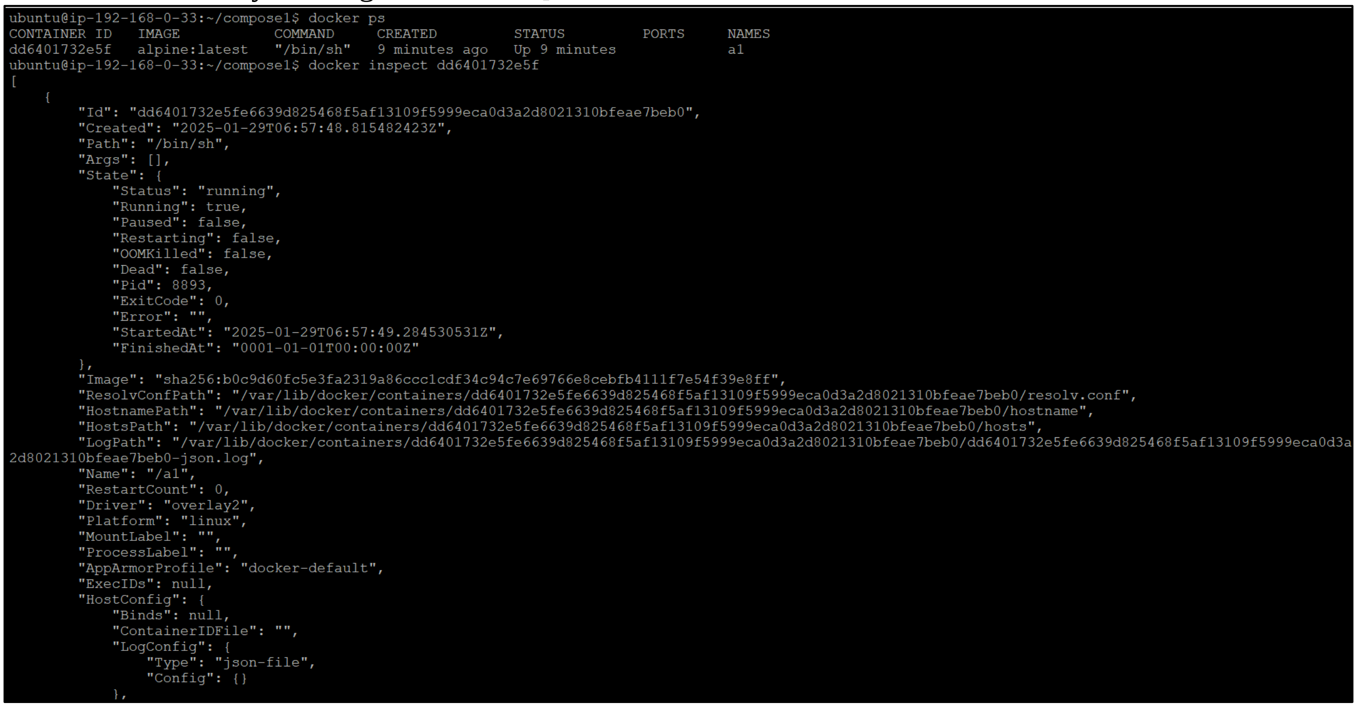
```
ubuntu@ip-192-168-0-33:~/compose1$ docker run --name a1 -itd alpine:latest
Unable to find image 'alpine:latest' locally
latest: Pulling from library/alpine
1f3e46996e29: Already exists
Digest: sha256:56fa17d2a7e7f168a043a2712e63aed1f8543aeafdcee47c58dcffe38ed51099
Status: Downloaded newer image for alpine:latest
dd6401732e5fe6639d825468f5af13109f5999eca0d3a2d8021310bfeae7beb0
```

- Now execute the command `docker ps` to check the newly created container from the previous command and see its current status.

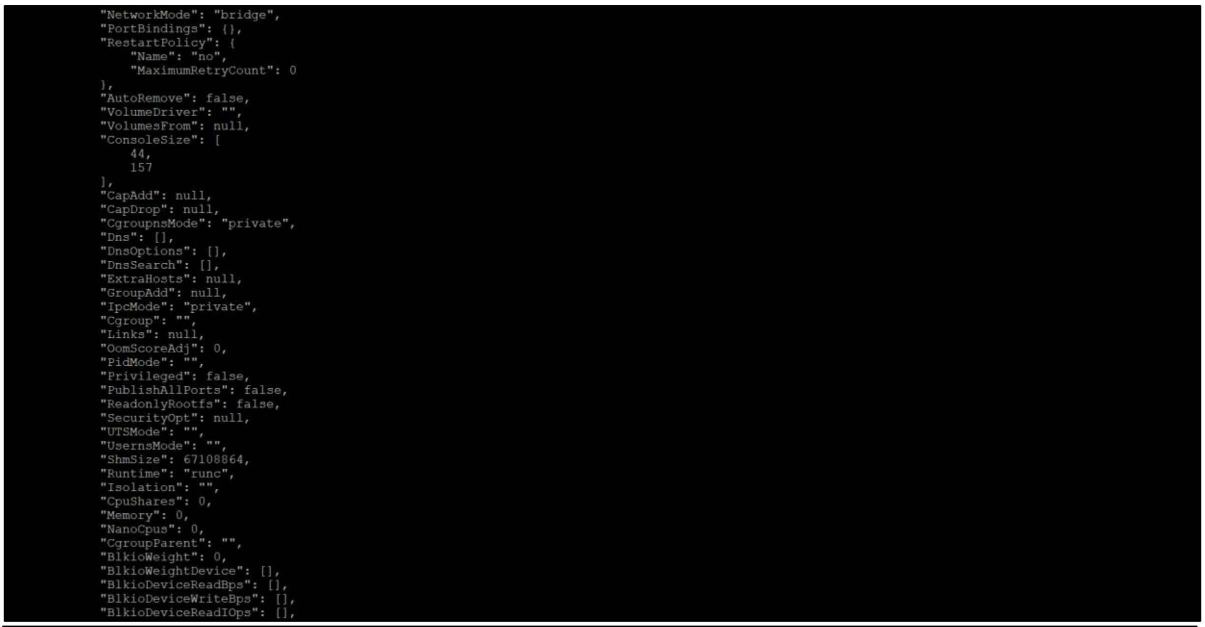


```
ubuntu@ip-192-168-0-33:~/compose1$ docker ps
CONTAINER ID   IMAGE           COMMAND     CREATED          STATUS         PORTS     NAMES
dd6401732e5f   alpine:latest   "/bin/sh"   13 seconds ago   Up 12 seconds            a1
ubuntu@ip-192-168-0-33:~/compose1$ 
```

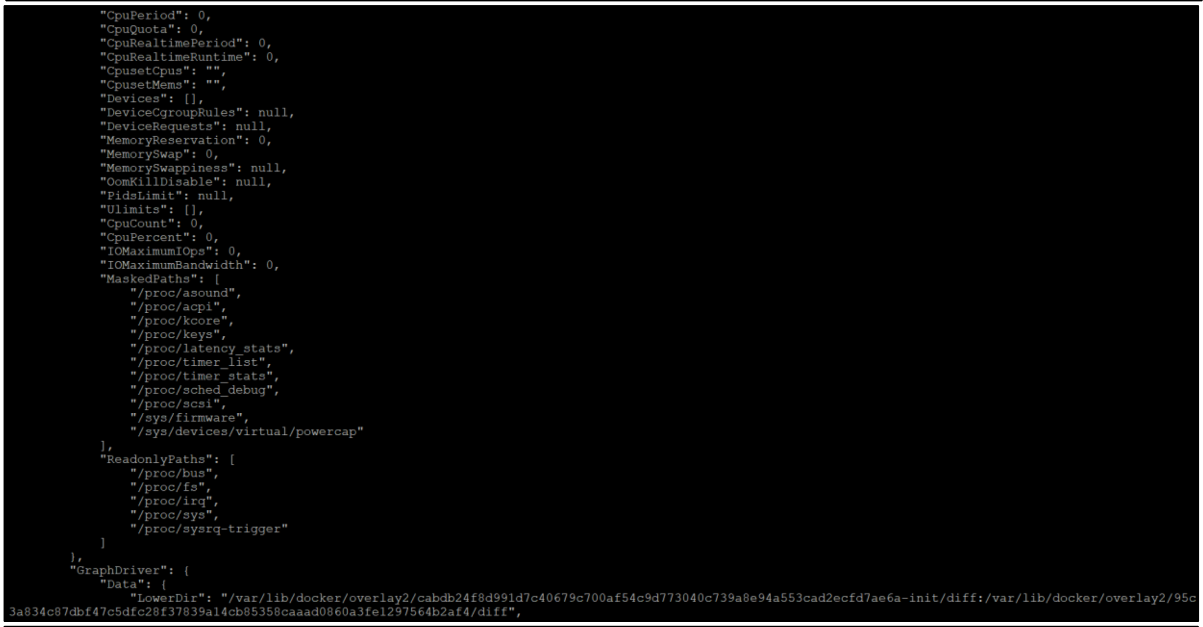
- Now run the command `docker inspect <containerID>` `in my case` `docker inspect dd6401732e5f` followed by the container ID of the **a1** container to obtain detailed information about the container. You can obtain the container ID by running the `docker ps` command.



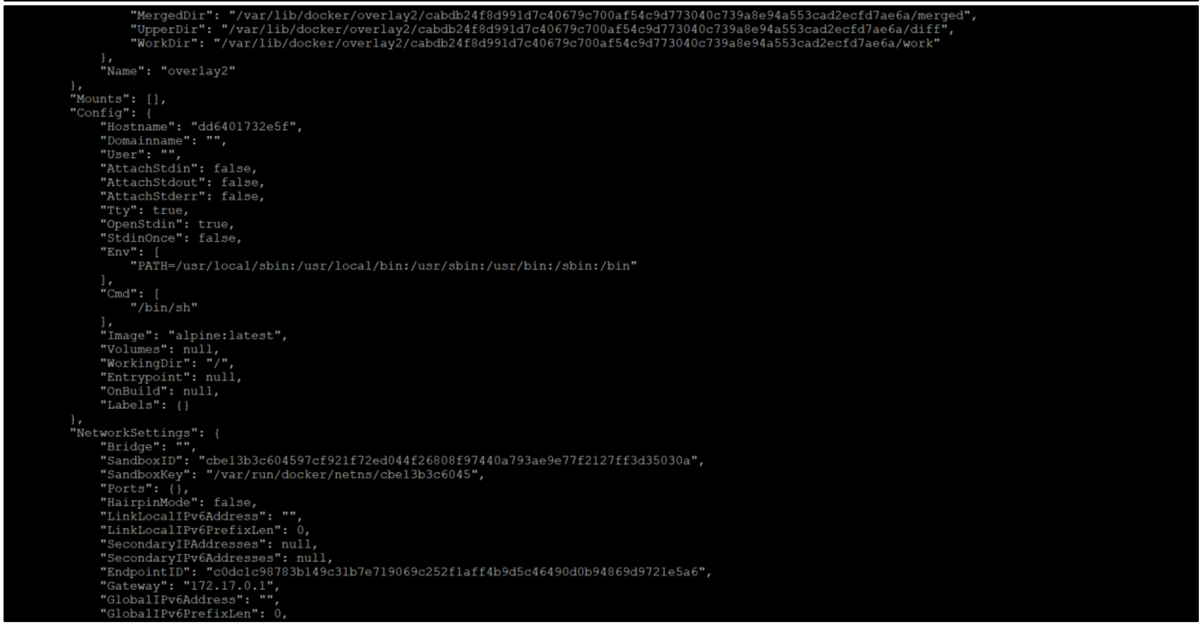
```
ubuntu@ip-192-168-0-33:~/compose1$ docker ps
CONTAINER ID   IMAGE           COMMAND     CREATED         STATUS         PORTS     NAMES
dd6401732e5f   alpine:latest   "/bin/sh"   9 minutes ago   Up 9 minutes             a1
ubuntu@ip-192-168-0-33:~/compose1$ docker inspect dd6401732e5f
[
    {
        "Id": "dd6401732e5fe6639d825468f5af13109f5999eca0d3a2d8021310bfeae7beb0",
        "Created": "2025-01-29T06:57:48.815482423Z",
        "Path": "/bin/sh",
        "Args": [],
        "State": {
            "Status": "running",
            "Running": true,
            "Paused": false,
            "Restarting": false,
            "OOMKilled": false,
            "Dead": false,
            "Pid": 8893,
            "ExitCode": 0,
            "Error": "",
            "StartedAt": "2025-01-29T06:57:49.284530531Z",
            "FinishedAt": "0001-01-01T00:00:00Z"
        },
        "Image": "sha256:b0c9d60fc5e3fa2319a86ccc1cdf34c94c7e69766e8cebfb4111f7e54f39e8ff",
        "ResolvConfPath": "/var/lib/docker/containers/dd6401732e5fe6639d825468f5af13109f5999eca0d3a2d8021310bfeae7beb0/resolv.conf",
        "HostnamePath": "/var/lib/docker/containers/dd6401732e5fe6639d825468f5af13109f5999eca0d3a2d8021310bfeae7beb0/hostname",
        "HostsPath": "/var/lib/docker/containers/dd6401732e5fe6639d825468f5af13109f5999eca0d3a2d8021310bfeae7beb0/hosts",
        "LogPath": "/var/lib/docker/containers/dd6401732e5fe6639d825468f5af13109f5999eca0d3a2d8021310bfeae7beb0/dd6401732e5fe6639d825468f5af13109f5999eca0d3a
2d8021310bfeae7beb0-json.log",
        "Name": "/a1",
        "RestartCount": 0,
        "Driver": "overlay2",
        "Platform": "linux",
        "MountLabel": "",
        "ProcessLabel": "",
        "AppArmorProfile": "docker-default",
        "ExecIDs": null,
        "HostConfig": {
            "Binds": null,
            "ContainerIDFile": "",
            "LogConfig": {
                "Type": "json-file",
                "Config": {}
            },
```



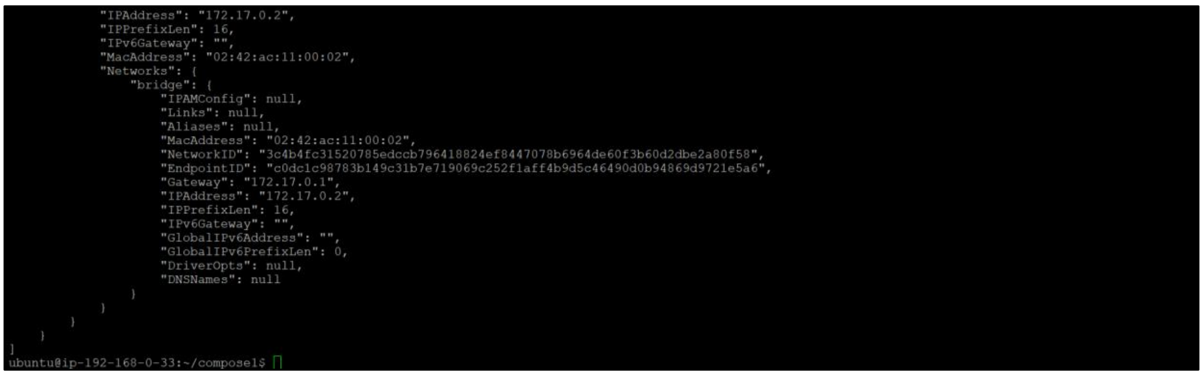
```
            "NetworkMode": "bridge",
            "PortBindings": {},
            "RestartPolicy": {
                "Name": "no",
                "MaximumRetryCount": 0
            },
            "AutoRemove": false,
            "VolumeDriver": "",
            "VolumesFrom": null,
            "ConsoleSize": [
                44,
                157
            ],
            "CapAdd": null,
            "CapDrop": null,
            "CgroupnsMode": "private",
            "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",
            "Isolation": "",
            "CpuShares": 0,
            "Memory": 0,
            "NanoCpus": 0,
            "CgroupParent": "",
            "BlkioWeight": 0,
            "BlkioWeightDevice": [],
            "BlkioDeviceReadBps": [],
            "BlkioDeviceWriteBps": [],
            "BlkioDeviceReadIOps": [],
```



```
            "CpuPeriod": 0,
            "CpuQuota": 0,
            "CpuRealtimePeriod": 0,
            "CpuRealtimeRuntime": 0,
            "CpusetCpus": "",
            "CpusetMems": "",
            "Devices": [],
            "DeviceCgroupRules": null,
            "DeviceRequests": null,
            "MemoryReservation": 0,
            "MemorySwap": 0,
            "MemorySwappiness": null,
            "OomKillDisable": null,
            "PidsLimit": null,
            "Ulimits": [],
            "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",
                "/sys/devices/virtual/powercap"
            ],
            "ReadonlyPaths": [
                "/proc/bus",
                "/proc/fs",
                "/proc/irq",
                "/proc/sys",
                "/proc/sysrq-trigger"
            ]
        },
        "GraphDriver": {
            "Data": {
                "LowerDir": "/var/lib/docker/overlay2/cabdb24f8d991d7c40679c700af54c9d773040c739a8e94a553cad2ecfd7ae6a-init/diff:/var/lib/docker/overlay2/95c
3a834c87dbf47c5dfc28f37839a14cb85358caaad0860a3fe1297564b2af4/diff",
```



```
                "MergedDir": "/var/lib/docker/overlay2/cabdb24f8d991d7c40679c700af54c9d773040c739a8e94a553cad2ecfd7ae6a/merged",
                "UpperDir": "/var/lib/docker/overlay2/cabdb24f8d991d7c40679c700af54c9d773040c739a8e94a553cad2ecfd7ae6a/diff",
                "WorkDir": "/var/lib/docker/overlay2/cabdb24f8d991d7c40679c700af54c9d773040c739a8e94a553cad2ecfd7ae6a/work"
            },
            "Name": "overlay2"
        },
        "Mounts": [],
        "Config": {
            "Hostname": "dd6401732e5f",
            "Domainname": "",
            "User": "",
            "AttachStdin": false,
            "AttachStdout": false,
            "AttachStderr": false,
            "Tty": true,
            "OpenStdin": true,
            "StdinOnce": false,
            "Env": [
                "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
            ],
            "Cmd": [
                "/bin/sh"
            ],
            "Image": "alpine:latest",
            "Volumes": null,
            "WorkingDir": "/",
            "Entrypoint": null,
            "OnBuild": null,
            "Labels": {}
        },
        "NetworkSettings": {
            "Bridge": "",
            "SandboxID": "cbe13b3c604597cf921f72ed044f26808f97440a793ae9e77f2127ff3d35030a",
            "SandboxKey": "/var/run/docker/netns/cbe13b3c6045",
            "Ports": {},
            "HairpinMode": false,
            "LinkLocalIPv6Address": "",
            "LinkLocalIPv6PrefixLen": 0,
            "SecondaryIPAddresses": null,
            "SecondaryIPv6Addresses": null,
            "EndpointID": "c0dc1c98783b149c31b7e719069c252f1aff4b9d5c46490d0b94869d9721e5a6",
            "Gateway": "172.17.0.1",
            "GlobalIPv6Address": "",
            "GlobalIPv6PrefixLen": 0,
```

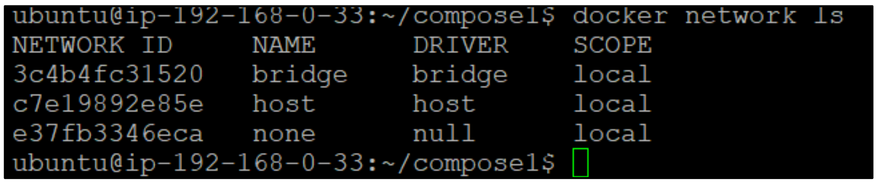


```
            "IPAddress": "172.17.0.2",
            "IPPrefixLen": 16,
            "IPv6Gateway": "",
            "MacAddress": "02:42:ac:11:00:02",
            "Networks": {
                "bridge": {
                    "IPAMConfig": null,
                    "Links": null,
                    "Aliases": null,
                    "MacAddress": "02:42:ac:11:00:02",
                    "NetworkID": "3c4b4fc31520785edccb796418824ef8447078b6964de60f3b60d2dbe2a80f58",
                    "EndpointID": "c0dc1c98783b149c31b7e719069c252f1aff4b9d5c46490d0b94869d9721e5a6",
                    "Gateway": "172.17.0.1",
                    "IPAddress": "172.17.0.2",
                    "IPPrefixLen": 16,
                    "IPv6Gateway": "",
                    "GlobalIPv6Address": "",
                    "GlobalIPv6PrefixLen": 0,
                    "DriverOpts": null,
                    "DNSNames": null
                }
            }
        }
    }
]
ubuntu@ip-192-168-0-33:~/compose1$ 
```

We can observe the container’s network settings from the `docker inspect` command we used to obtain detailed information about the container as follows:

- **IPAddress**: `172.17.0.2`
- **Gateway**: `172.17.0.1`
- **Network Settings**:
    - **Network Mode**: `bridge`
    - **Network ID**: `3c4b4fc31520785edccb796418824ef8447078b6964de60f3b60d2dbe2a80f58`
    - **Endpoint ID**: `c0dc1c98783b149c31b7e719069c252f1aff4b9d5c46490d0b94869d9721e5a6`
    - **Link Local IPv6 Address**: N/A
    - **Link Local IPv6 Prefix Length**: `0`
    - **Secondary IP Addresses**: N/A
    - **Secondary IPv6 Addresses**: N/A
    - **Global IPv6 Address**: N/A
    - **Global IPv6 Prefix Length**: `0`
    - **DNS Names**: N/A

- From the above data, we can observe that the **a1** container has a network mode set to **bridge**.
- We can obtain the available networks in Docker by running the command `docker network ls`.



```
ubuntu@ip-192-168-0-33:~/compose1$ docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
3c4b4fc31520   bridge    bridge    local
c7e19892e85e   host      host      local
e37fb3346eca   none      null      local
ubuntu@ip-192-168-0-33:~/compose1$ 
```

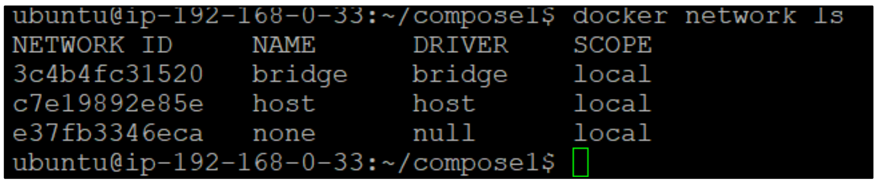
In Docker, there are three main types of networks: **bridge**, **host**, and **none**.

- The **bridge** network is the default network for containers when none is specified. It creates a private internal network on your host system, and containers on this network can communicate with each other.

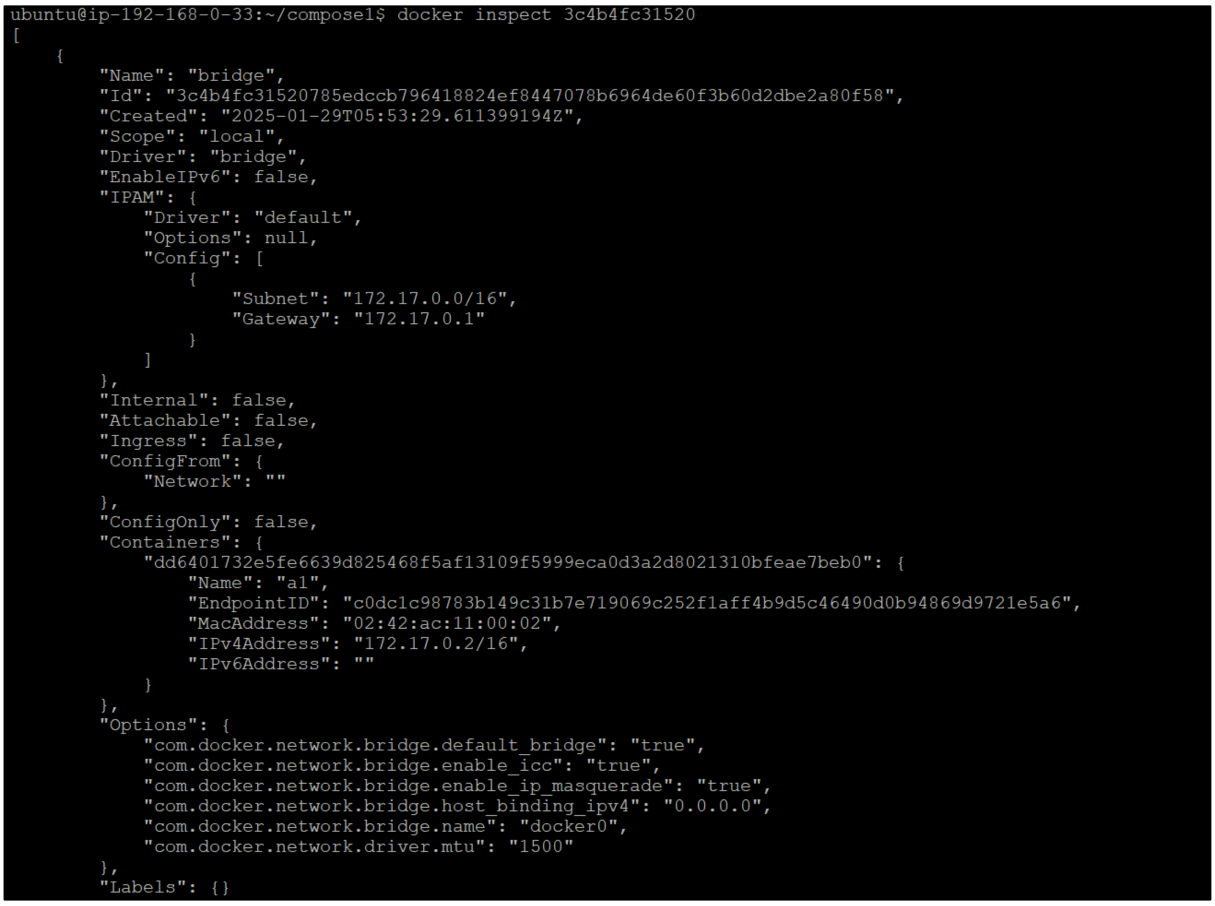
- The **host** network allows containers to share the host's network stack, which means the container uses the host's IP address and can directly communicate with the external network.
- The **none** network isolates the container from any network, meaning it cannot communicate with other containers or the outside world.

By default, Docker containers are connected to the **bridge** network unless otherwise specified.

- We can inspect the networks using the command `docker inspect <bridge network-id>`.
  In my case scenario, to inspect the details of each network, I can use the `docker inspect` command followed by the network ID. For instance, **to inspect the Bridge network**, I would run `docker inspect 3c4b4fc31520`, **for the Host network**, I would run `docker inspect c7e19892e85e`, and **for the None network**, I would run `docker inspect e37fb3346eca`. These commands will provide detailed information about the respective networks, including network settings, connected containers, and other relevant configurations.

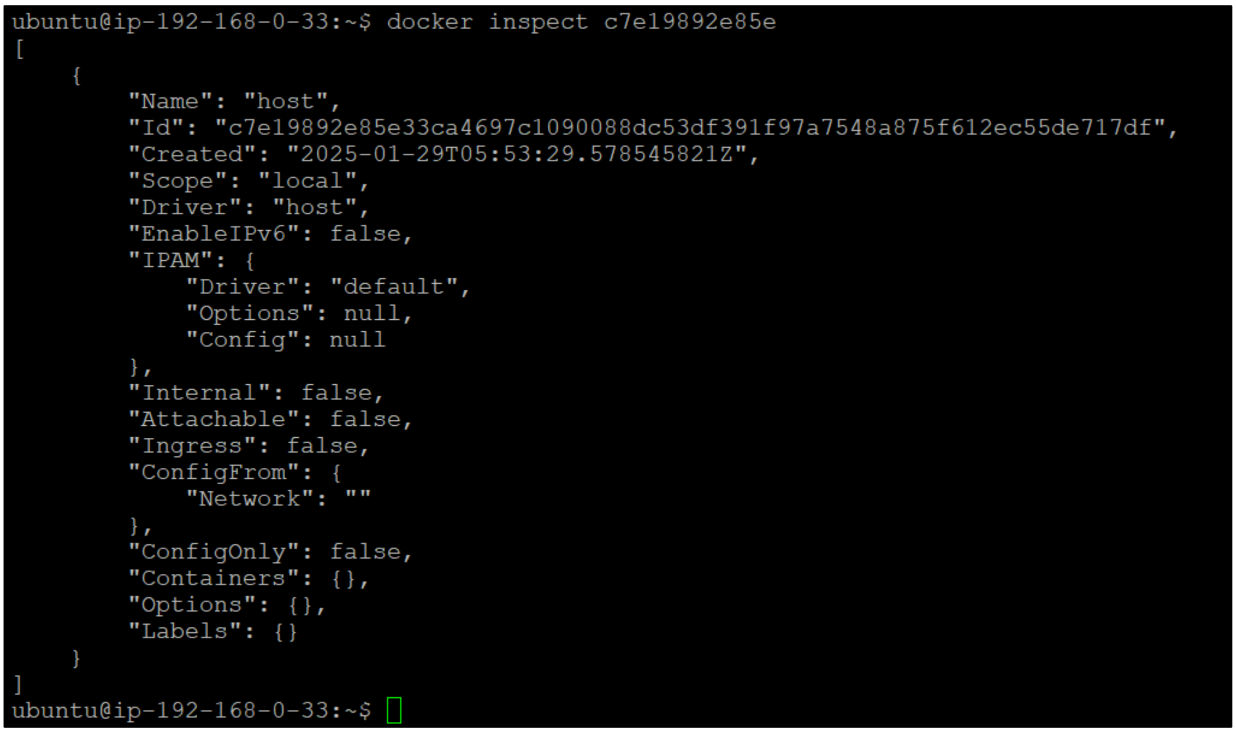


```
ubuntu@ip-192-168-0-33:~/compose1$ docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
3c4b4fc31520   bridge    bridge    local
c7e19892e85e   host      host      local
e37fb3346eca   none      null      local
ubuntu@ip-192-168-0-33:~/compose1$
```

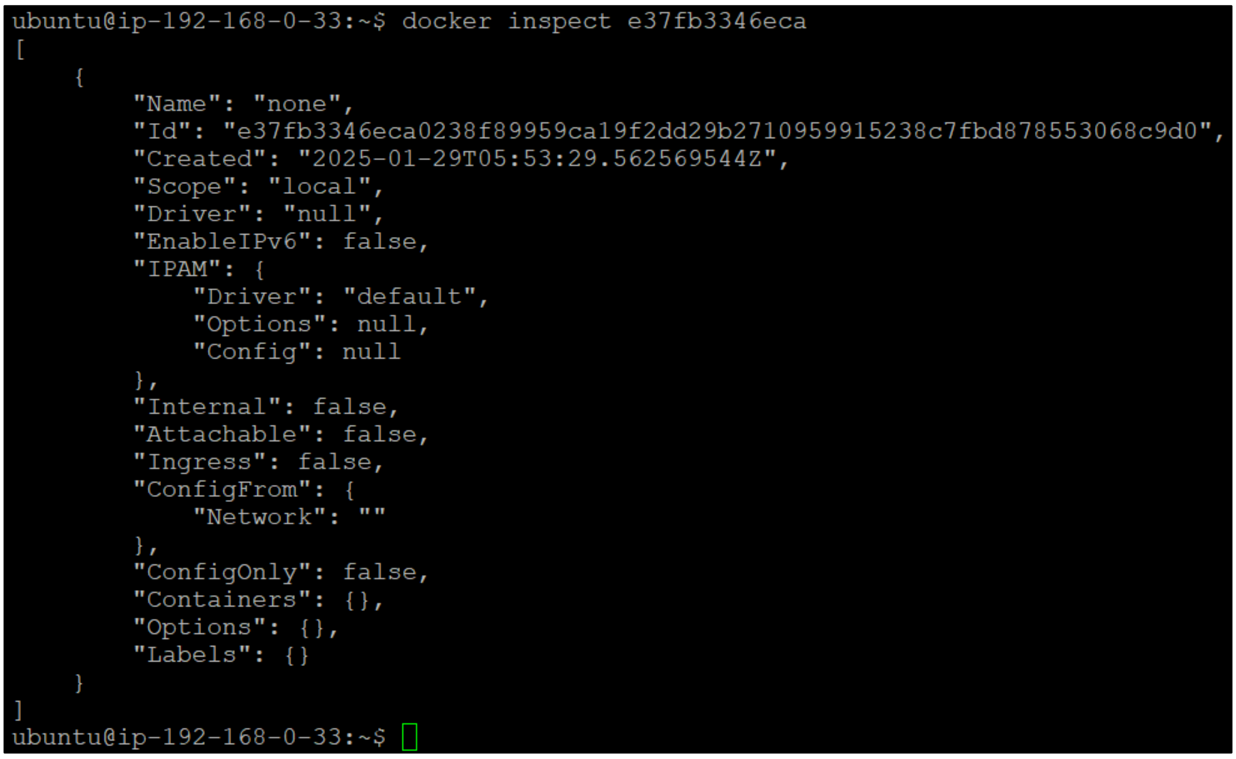


```
ubuntu@ip-192-168-0-33:~/compose1$ docker inspect 3c4b4fc31520
[
    {
        "Name": "bridge",
        "Id": "3c4b4fc31520785edccb796418824ef8447078b6964de60f3b60d2dbe2a80f58",
        "Created": "2025-01-29T05:53:29.611399194Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": 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": {
            "dd6401732e5fe6639d825468f5af13109f5999eca0d3a2d8021310bfeae7beb0": {
                "Name": "a1",
                "EndpointID": "c0dc1c98783b149c31b7e719069c252f1aff4b9d5c46490d0b94869d9721e5a6",
                "MacAddress": "02:42:ac:11:00:02",
                "IPv4Address": "172.17.0.2/16",
                "IPv6Address": ""
            }
        },
        "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, any container created will be added to the Bridge network. In this case, we can see from the output that the container "a1" we created is listed under the "Containers" section of the Bridge network. This indicates that when containers are created, they are allocated to the Bridge network by default. The container "a1" has an assigned IP address of 172.17.0.2/16 within the Bridge network, confirming that it is part of this network and able to communicate with other containers on the same network.

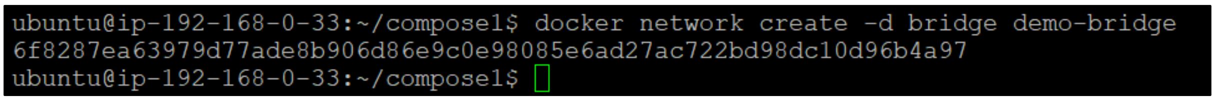


```
ubuntu@ip-192-168-0-33:~$ docker inspect c7e19892e85e
[
    {
        "Name": "host",
        "Id": "c7e19892e85e33ca4697c1090088dc53df391f97a7548a875f612ec55de717df",
        "Created": "2025-01-29T05:53:29.578545821Z",
        "Scope": "local",
        "Driver": "host",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": null
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {},
        "Options": {},
        "Labels": {}
    }
]
ubuntu@ip-192-168-0-33:~$
```

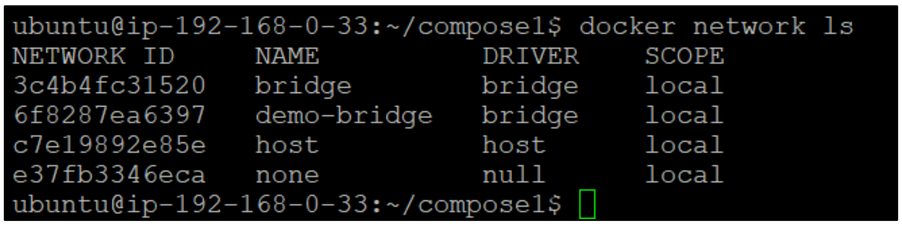


```
ubuntu@ip-192-168-0-33:~$ docker inspect e37fb3346eca
[
    {
        "Name": "none",
        "Id": "e37fb3346eca0238f89959ca19f2dd29b2710959915238c7fbd878553068c9d0",
        "Created": "2025-01-29T05:53:29.562569544Z",
        "Scope": "local",
        "Driver": "null",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": null
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {},
        "Options": {},
        "Labels": {}
    }
]
ubuntu@ip-192-168-0-33:~$
```

- Containers created on the same bridge network can communicate with each other using their IP addresses, which simplifies the process of setting up multi-container applications. To create a new network of the desired type, you can use the command `docker network create -d bridge <networkname>`. For example, to create a network named **demo-bridge**, the command would be `docker network create -d bridge demo-bridge`.



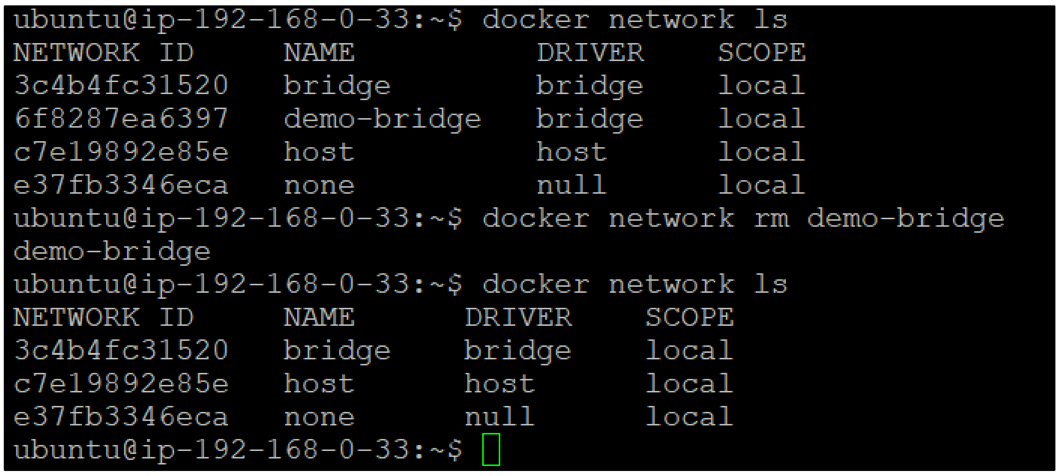
```
ubuntu@ip-192-168-0-33:~/compose1$ docker network create -d bridge demo-bridge
6f8287ea63979d77ade8b906d86e9c0e98085e6ad27ac722bd98dc10d96b4a97
ubuntu@ip-192-168-0-33:~/compose1$
```



```
ubuntu@ip-192-168-0-33:~/compose1$ docker network ls
NETWORK ID     NAME          DRIVER    SCOPE
3c4b4fc31520   bridge        bridge    local
6f8287ea6397   demo-bridge   bridge    local
c7e19892e85e   host          host      local
e37fb3346eca   none          null      local
ubuntu@ip-192-168-0-33:~/compose1$
```

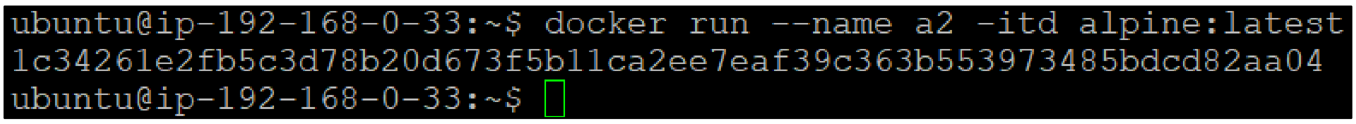
A new bridge network demo-bridge has been successfully created.

- To delete a network in Docker, use the command `docker network rm <network_name_or_id>`, for example, `docker network rm demo-bridge` to **remove the `demo-bridge` network.**

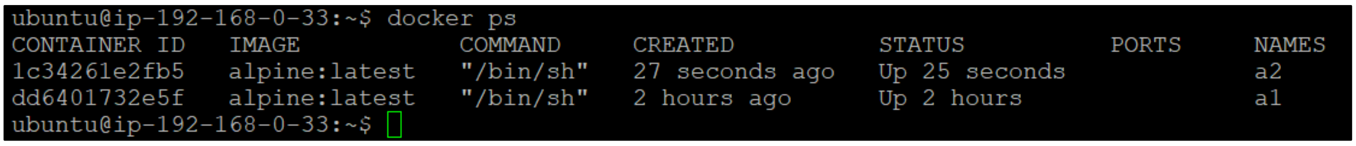


```
ubuntu@ip-192-168-0-33:~$ docker network ls
NETWORK ID     NAME          DRIVER    SCOPE
3c4b4fc31520   bridge        bridge    local
6f8287ea6397   demo-bridge   bridge    local
c7e19892e85e   host          host      local
e37fb3346eca   none          null      local
ubuntu@ip-192-168-0-33:~$ docker network rm demo-bridge
demo-bridge
ubuntu@ip-192-168-0-33:~$ docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
3c4b4fc31520   bridge    bridge    local
c7e19892e85e   host      host      local
e37fb3346eca   none      null      local
ubuntu@ip-192-168-0-33:~$
```

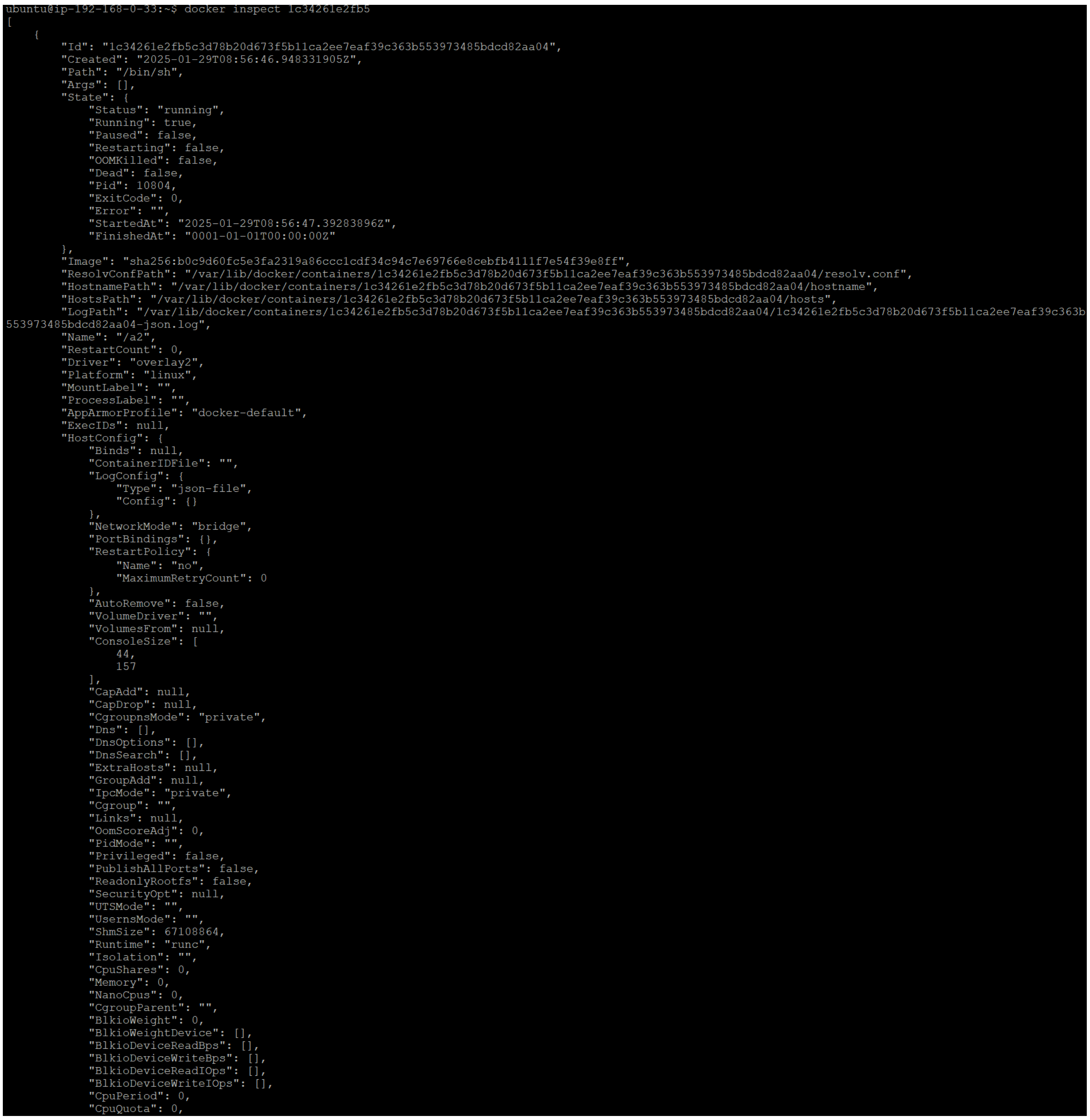
- Now, let's **create another container named `a2`** by executing the command `docker run --name a2 -itd alpine:latest`. After that, check the networks of both containers by running the `docker inspect` commands using their container IDs (which you can obtain from `docker ps`), like so: `docker inspect <container_id_of_a1>` and `docker inspect <container_id_of_a2>`.



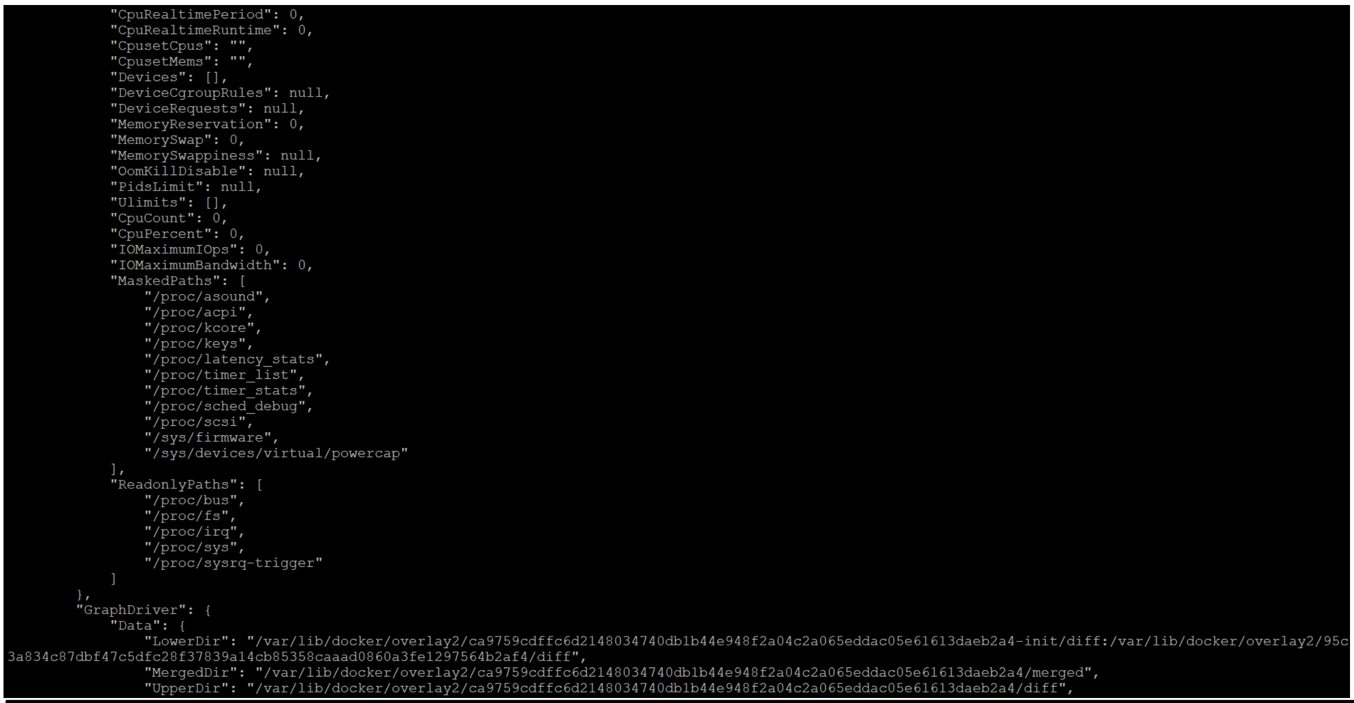
```
ubuntu@ip-192-168-0-33:~$ docker run --name a2 -itd alpine:latest
1c34261e2fb5c3d78b20d673f5b11ca2ee7eaf39c363b553973485bdcd82aa04
ubuntu@ip-192-168-0-33:~$
```



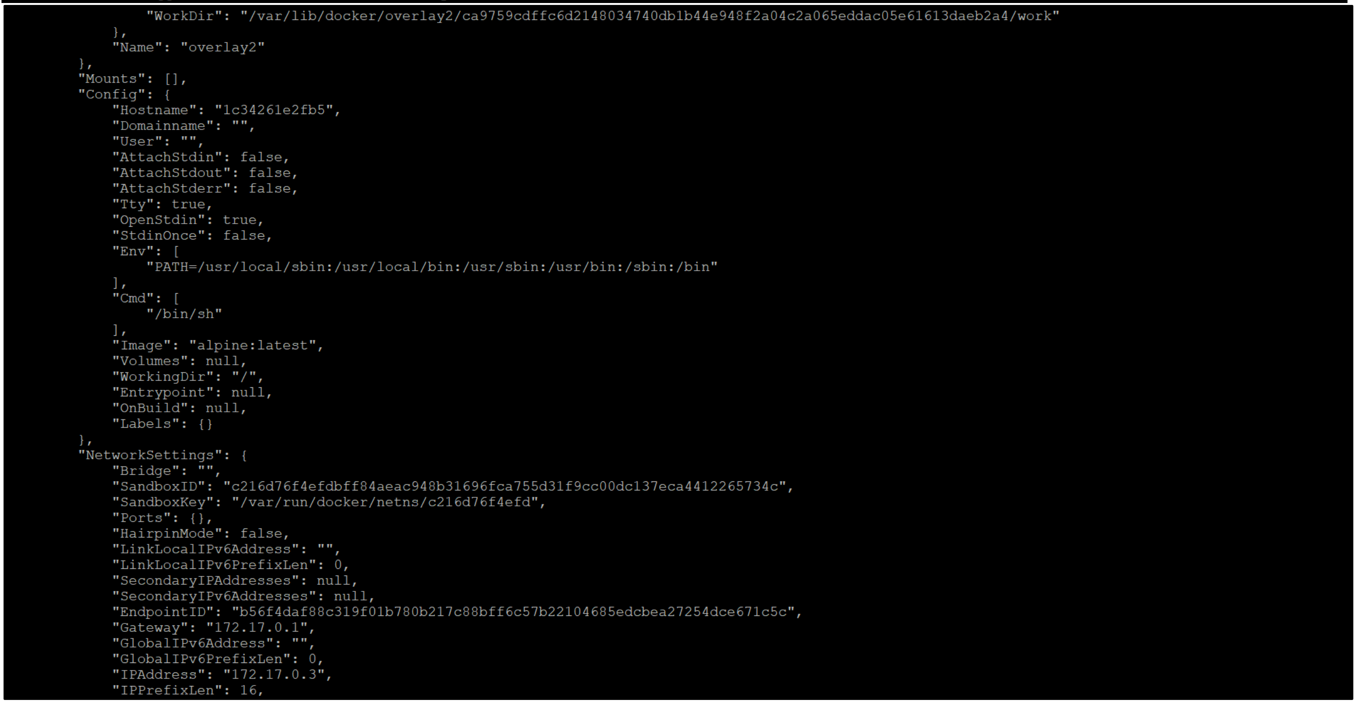
```
ubuntu@ip-192-168-0-33:~$ docker ps
CONTAINER ID   IMAGE           COMMAND     CREATED          STATUS          PORTS     NAMES
1c34261e2fb5   alpine:latest   "/bin/sh"   27 seconds ago   Up 25 seconds             a2
dd6401732e5f   alpine:latest   "/bin/sh"   2 hours ago      Up 2 hours                a1
ubuntu@ip-192-168-0-33:~$
```



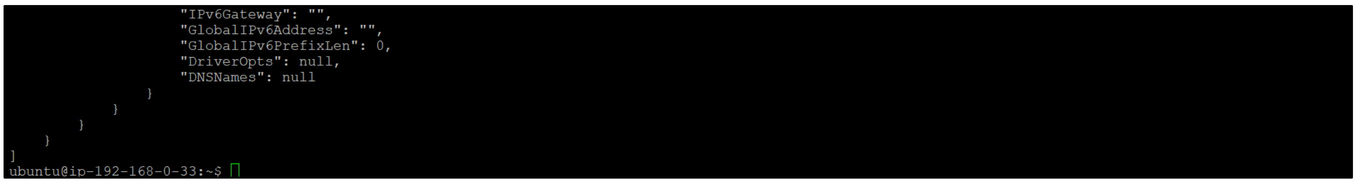
```
ubuntu@ip-192-168-0-33:~$ docker inspect 1c34261e2fb5
[
    {
        "Id": "1c34261e2fb5c3d78b20d673f5b11ca2ee7eaf39c363b553973485bdcd82aa04",
        "Created": "2025-01-29T08:56:46.948331905Z",
        "Path": "/bin/sh",
        "Args": [],
        "State": {
            "Status": "running",
            "Running": true,
            "Paused": false,
            "Restarting": false,
            "OOMKilled": false,
            "Dead": false,
            "Pid": 10804,
            "ExitCode": 0,
            "Error": "",
            "StartedAt": "2025-01-29T08:56:47.39283896Z",
            "FinishedAt": "0001-01-01T00:00:00Z"
        },
        "Image": "sha256:b0c9d60fc5e3fa2319a86ccc1cdf34c94c7e69766e8cebfb4111f7e54f39e8ff",
        "ResolvConfPath": "/var/lib/docker/containers/1c34261e2fb5c3d78b20d673f5b11ca2ee7eaf39c363b553973485bdcd82aa04/resolv.conf",
        "HostnamePath": "/var/lib/docker/containers/1c34261e2fb5c3d78b20d673f5b11ca2ee7eaf39c363b553973485bdcd82aa04/hostname",
        "HostsPath": "/var/lib/docker/containers/1c34261e2fb5c3d78b20d673f5b11ca2ee7eaf39c363b553973485bdcd82aa04/hosts",
        "LogPath": "/var/lib/docker/containers/1c34261e2fb5c3d78b20d673f5b11ca2ee7eaf39c363b553973485bdcd82aa04/1c34261e2fb5c3d78b20d673f5b11ca2ee7eaf39c363b
553973485bdcd82aa04-json.log",
        "Name": "/a2",
        "RestartCount": 0,
        "Driver": "overlay2",
        "Platform": "linux",
        "MountLabel": "",
        "ProcessLabel": "",
        "AppArmorProfile": "docker-default",
        "ExecIDs": null,
        "HostConfig": {
            "Binds": null,
            "ContainerIDFile": "",
            "LogConfig": {
                "Type": "json-file",
                "Config": {}
            },
            "NetworkMode": "bridge",
            "PortBindings": {},
            "RestartPolicy": {
                "Name": "no",
                "MaximumRetryCount": 0
            },
            "AutoRemove": false,
            "VolumeDriver": "",
            "VolumesFrom": null,
            "ConsoleSize": [
                44,
                157
            ],
            "CapAdd": null,
            "CapDrop": null,
            "CgroupnsMode": "private",
            "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",
            "Isolation": "",
            "CpuShares": 0,
            "Memory": 0,
            "NanoCpus": 0,
            "CgroupParent": "",
            "BlkioWeight": 0,
            "BlkioWeightDevice": [],
            "BlkioDeviceReadBps": [],
            "BlkioDeviceWriteBps": [],
            "BlkioDeviceReadIOps": [],
            "BlkioDeviceWriteIOps": [],
            "CpuPeriod": 0,
            "CpuQuota": 0,
```



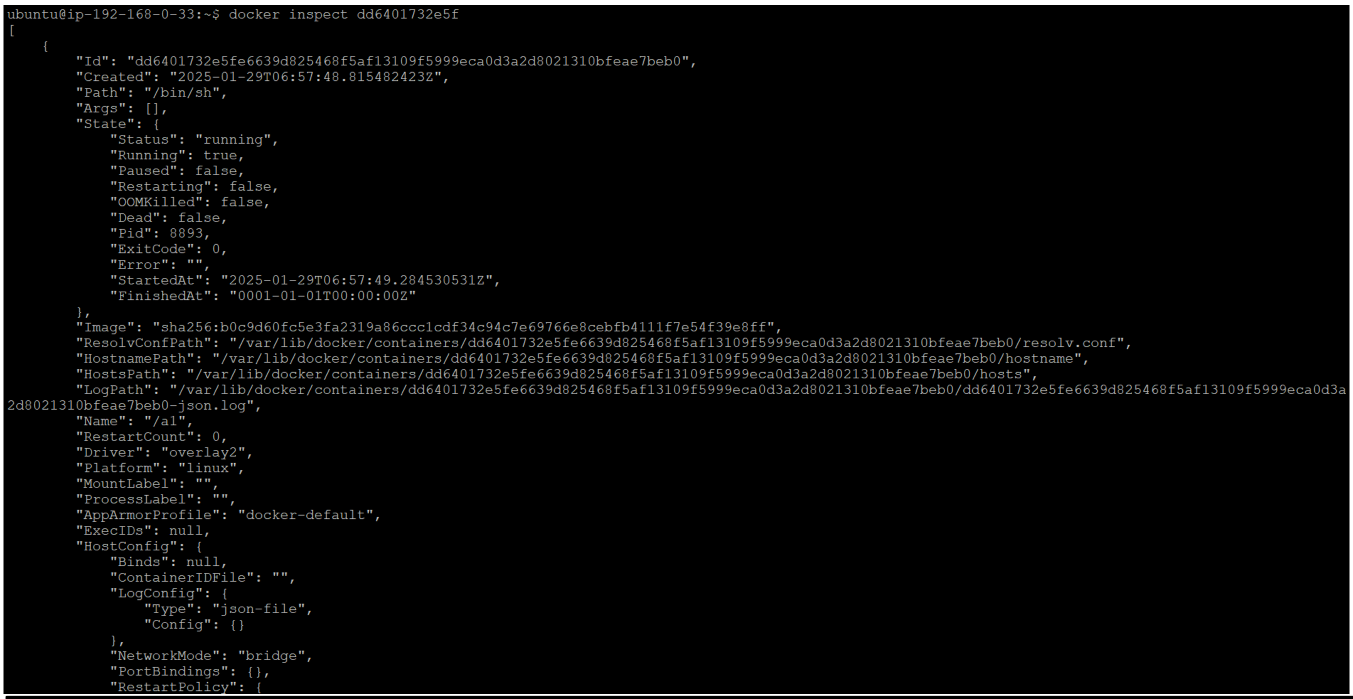
```
            "CpuRealtimePeriod": 0,
            "CpuRealtimeRuntime": 0,
            "CpusetCpus": "",
            "CpusetMems": "",
            "Devices": [],
            "DeviceCgroupRules": null,
            "DeviceRequests": null,
            "MemoryReservation": 0,
            "MemorySwap": 0,
            "MemorySwappiness": null,
            "OomKillDisable": null,
            "PidsLimit": null,
            "Ulimits": [],
            "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",
                "/sys/devices/virtual/powercap"
            ],
            "ReadonlyPaths": [
                "/proc/bus",
                "/proc/fs",
                "/proc/irq",
                "/proc/sys",
                "/proc/sysrq-trigger"
            ]
        },
        "GraphDriver": {
            "Data": {
                "LowerDir": "/var/lib/docker/overlay2/ca9759cdffc6d2148034740db1b44e948f2a04c2a065eddac05e61613daeb2a4-init/diff:/var/lib/docker/overlay2/95c
3a834c87dbf47c5dfc28f37839a14cb85358caaad0860a3fe1297564b2af4/diff",
                "MergedDir": "/var/lib/docker/overlay2/ca9759cdffc6d2148034740db1b44e948f2a04c2a065eddac05e61613daeb2a4/merged",
                "UpperDir": "/var/lib/docker/overlay2/ca9759cdffc6d2148034740db1b44e948f2a04c2a065eddac05e61613daeb2a4/diff",
```



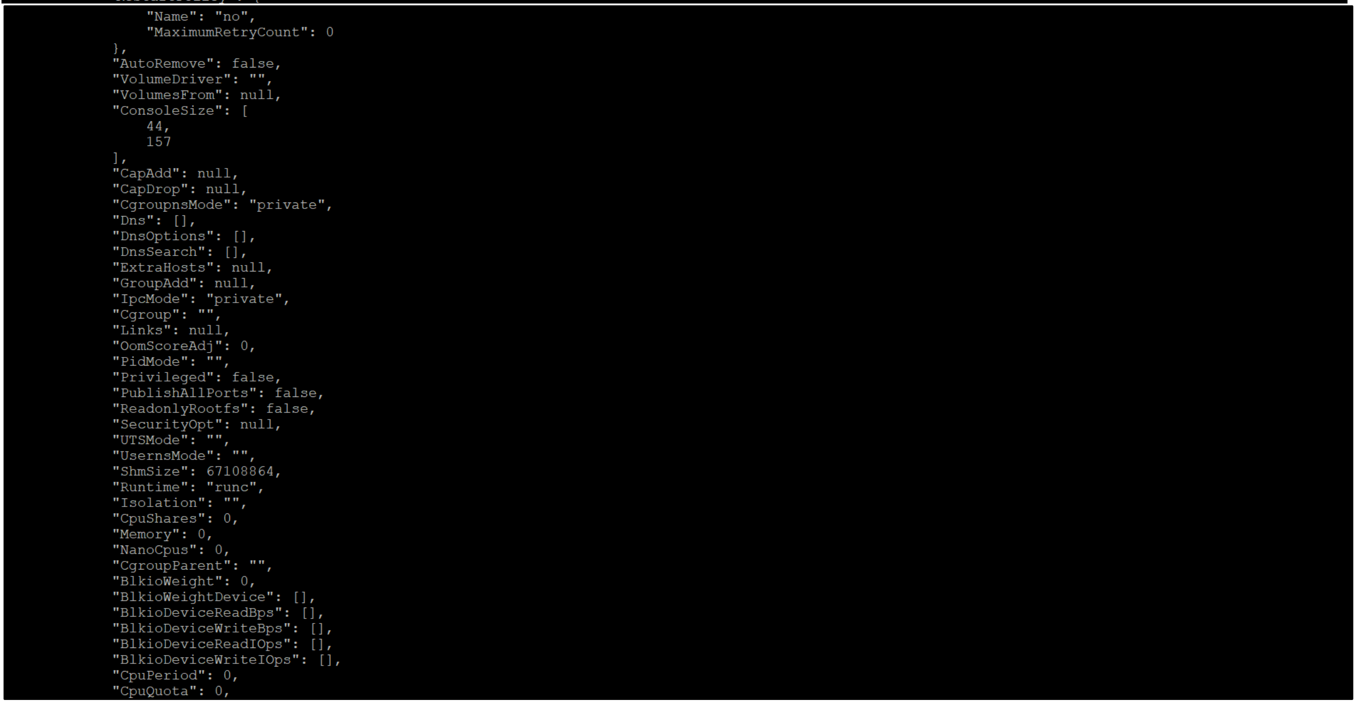
```
                "WorkDir": "/var/lib/docker/overlay2/ca9759cdffc6d2148034740db1b44e948f2a04c2a065eddac05e61613daeb2a4/work"
            },
            "Name": "overlay2"
        },
        "Mounts": [],
        "Config": {
            "Hostname": "1c34261e2fb5",
            "Domainname": "",
            "User": "",
            "AttachStdin": false,
            "AttachStdout": false,
            "AttachStderr": false,
            "Tty": true,
            "OpenStdin": true,
            "StdinOnce": false,
            "Env": [
                "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
            ],
            "Cmd": [
                "/bin/sh"
            ],
            "Image": "alpine:latest",
            "Volumes": null,
            "WorkingDir": "/",
            "Entrypoint": null,
            "OnBuild": null,
            "Labels": {}
        },
        "NetworkSettings": {
            "Bridge": "",
            "SandboxID": "c216d76f4efdbff84aeac948b31696fca755d31f9cc00dc137eca4412265734c",
            "SandboxKey": "/var/run/docker/netns/c216d76f4efd",
            "Ports": {},
            "HairpinMode": false,
            "LinkLocalIPv6Address": "",
            "LinkLocalIPv6PrefixLen": 0,
            "SecondaryIPAddresses": null,
            "SecondaryIPv6Addresses": null,
            "EndpointID": "b56f4daf88c319f01b780b217c88bff6c57b22104685edcbea27254dce671c5c",
            "Gateway": "172.17.0.1",
            "GlobalIPv6Address": "",
            "GlobalIPv6PrefixLen": 0,
            "IPAddress": "172.17.0.3",
            "IPPrefixLen": 16,
```



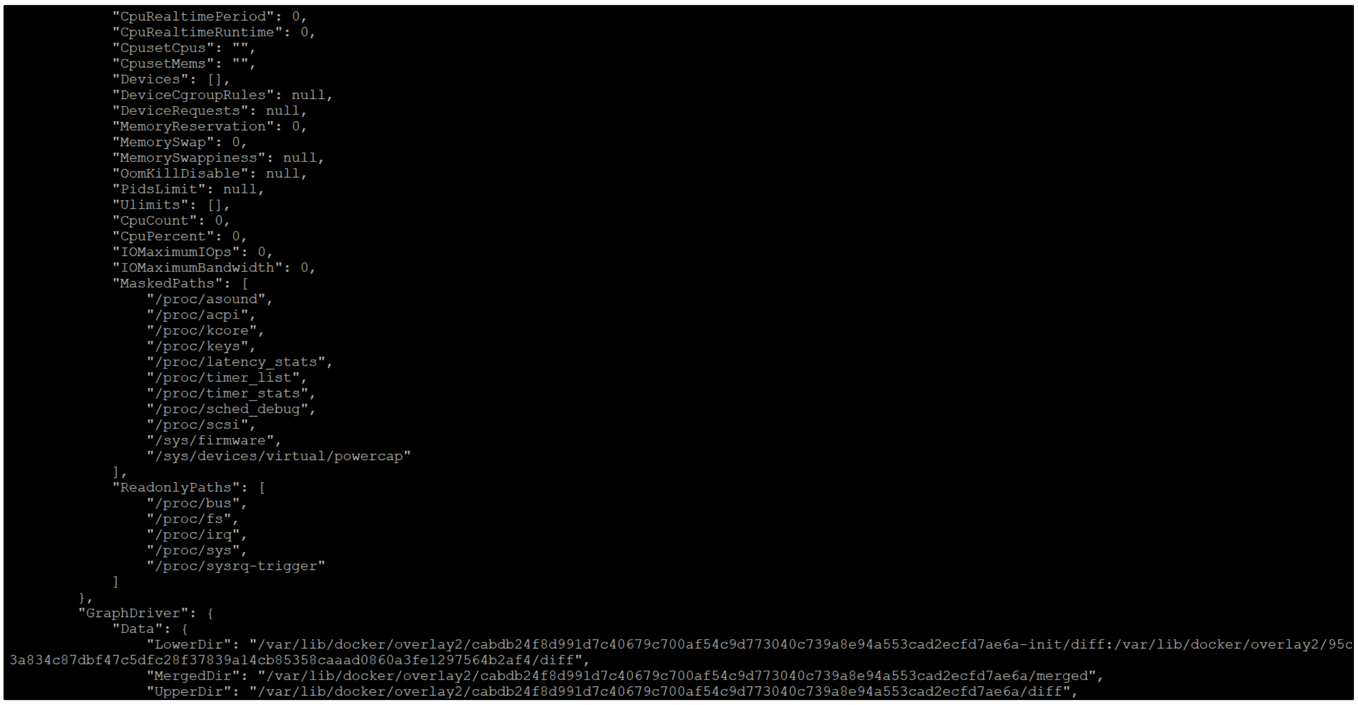
```
                    "IPv6Gateway": "",
                    "GlobalIPv6Address": "",
                    "GlobalIPv6PrefixLen": 0,
                    "DriverOpts": null,
                    "DNSNames": null
                }
            }
        }
    }
]
ubuntu@ip-192-168-0-33:~$ 
```



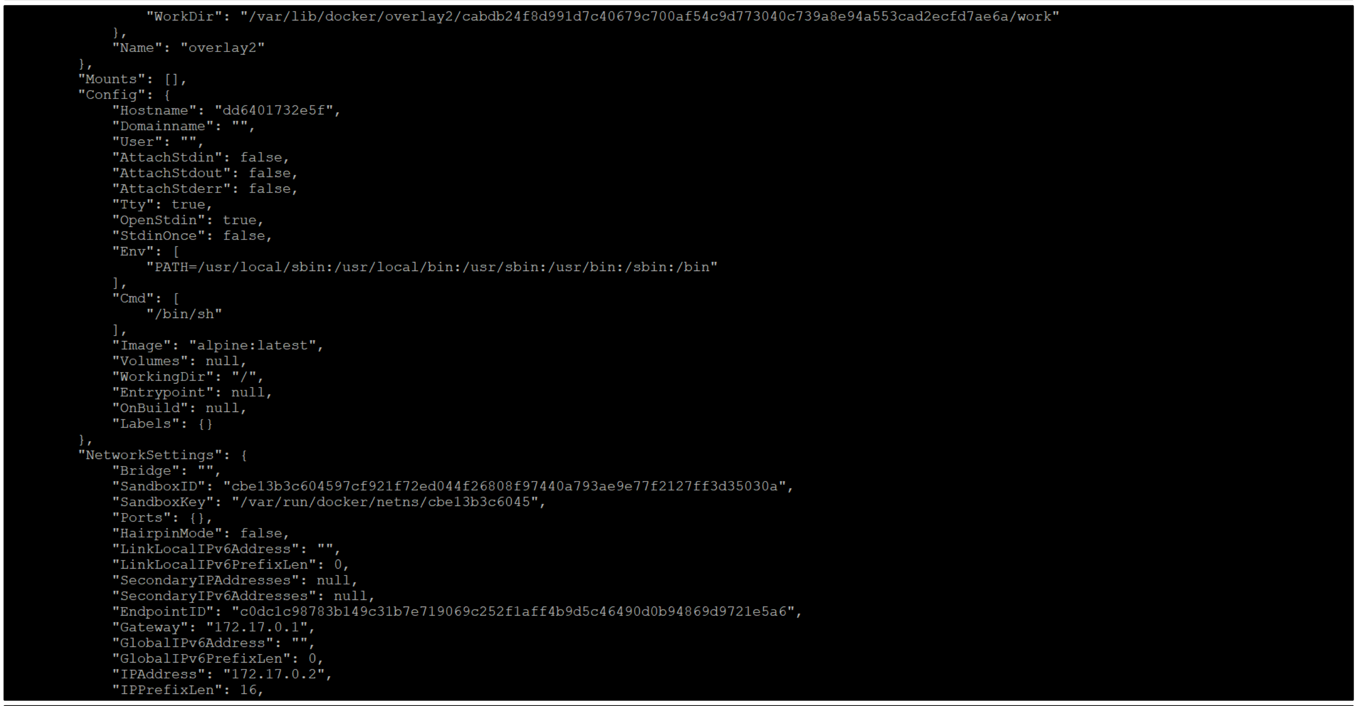
```
ubuntu@ip-192-168-0-33:~$ docker inspect dd6401732e5f
[
    {
        "Id": "dd6401732e5fe6639d825468f5af13109f5999eca0d3a2d8021310bfeae7beb0",
        "Created": "2025-01-29T06:57:48.815482423Z",
        "Path": "/bin/sh",
        "Args": [],
        "State": {
            "Status": "running",
            "Running": true,
            "Paused": false,
            "Restarting": false,
            "OOMKilled": false,
            "Dead": false,
            "Pid": 8893,
            "ExitCode": 0,
            "Error": "",
            "StartedAt": "2025-01-29T06:57:49.284530531Z",
            "FinishedAt": "0001-01-01T00:00:00Z"
        },
        "Image": "sha256:b0c9d60fc5e3fa2319a86ccc1cdf34c94c7e69766e8cebfb4111f7e54f39e8ff",
        "ResolvConfPath": "/var/lib/docker/containers/dd6401732e5fe6639d825468f5af13109f5999eca0d3a2d8021310bfeae7beb0/resolv.conf",
        "HostnamePath": "/var/lib/docker/containers/dd6401732e5fe6639d825468f5af13109f5999eca0d3a2d8021310bfeae7beb0/hostname",
        "HostsPath": "/var/lib/docker/containers/dd6401732e5fe6639d825468f5af13109f5999eca0d3a2d8021310bfeae7beb0/hosts",
        "LogPath": "/var/lib/docker/containers/dd6401732e5fe6639d825468f5af13109f5999eca0d3a2d8021310bfeae7beb0/dd6401732e5fe6639d825468f5af13109f5999eca0d3a
2d8021310bfeae7beb0-json.log",
        "Name": "/a1",
        "RestartCount": 0,
        "Driver": "overlay2",
        "Platform": "linux",
        "MountLabel": "",
        "ProcessLabel": "",
        "AppArmorProfile": "docker-default",
        "ExecIDs": null,
        "HostConfig": {
            "Binds": null,
            "ContainerIDFile": "",
            "LogConfig": {
                "Type": "json-file",
                "Config": {}
            },
            "NetworkMode": "bridge",
            "PortBindings": {},
            "RestartPolicy": {
```



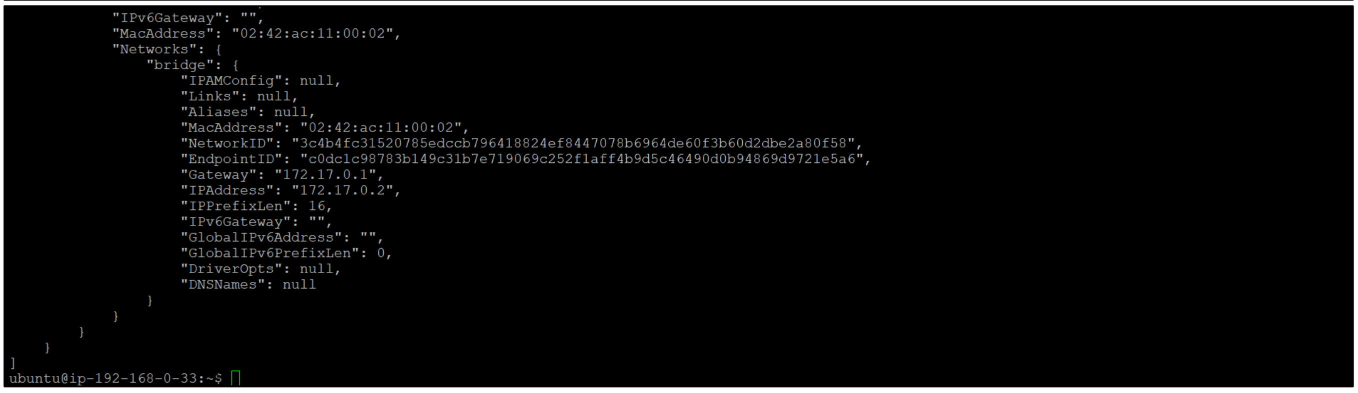
```
                "Name": "no",
                "MaximumRetryCount": 0
            },
            "AutoRemove": false,
            "VolumeDriver": "",
            "VolumesFrom": null,
            "ConsoleSize": [
                44,
                157
            ],
            "CapAdd": null,
            "CapDrop": null,
            "CgroupnsMode": "private",
            "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",
            "Isolation": "",
            "CpuShares": 0,
            "Memory": 0,
            "NanoCpus": 0,
            "CgroupParent": "",
            "BlkioWeight": 0,
            "BlkioWeightDevice": [],
            "BlkioDeviceReadBps": [],
            "BlkioDeviceWriteBps": [],
            "BlkioDeviceReadIOps": [],
            "BlkioDeviceWriteIOps": [],
            "CpuPeriod": 0,
            "CpuQuota": 0,
```



```
            "CpuRealtimePeriod": 0,
            "CpuRealtimeRuntime": 0,
            "CpusetCpus": "",
            "CpusetMems": "",
            "Devices": [],
            "DeviceCgroupRules": null,
            "DeviceRequests": null,
            "MemoryReservation": 0,
            "MemorySwap": 0,
            "MemorySwappiness": null,
            "OomKillDisable": null,
            "PidsLimit": null,
            "Ulimits": [],
            "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",
                "/sys/devices/virtual/powercap"
            ],
            "ReadonlyPaths": [
                "/proc/bus",
                "/proc/fs",
                "/proc/irq",
                "/proc/sys",
                "/proc/sysrq-trigger"
            ]
        },
        "GraphDriver": {
            "Data": {
                "LowerDir": "/var/lib/docker/overlay2/cabdb24f8d991d7c40679c700af54c9d773040c739a8e94a553cad2ecfd7ae6a-init/diff:/var/lib/docker/overlay2/95c
3a834c87dbf47c5dfc28f37839a14cb85358caaad0860a3fe1297564b2af4/diff",
                "MergedDir": "/var/lib/docker/overlay2/cabdb24f8d991d7c40679c700af54c9d773040c739a8e94a553cad2ecfd7ae6a/merged",
                "UpperDir": "/var/lib/docker/overlay2/cabdb24f8d991d7c40679c700af54c9d773040c739a8e94a553cad2ecfd7ae6a/diff",
```



```
                "WorkDir": "/var/lib/docker/overlay2/cabdb24f8d991d7c40679c700af54c9d773040c739a8e94a553cad2ecfd7ae6a/work"
            },
            "Name": "overlay2"
        },
        "Mounts": [],
        "Config": {
            "Hostname": "dd6401732e5f",
            "Domainname": "",
            "User": "",
            "AttachStdin": false,
            "AttachStdout": false,
            "AttachStderr": false,
            "Tty": true,
            "OpenStdin": true,
            "StdinOnce": false,
            "Env": [
                "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
            ],
            "Cmd": [
                "/bin/sh"
            ],
            "Image": "alpine:latest",
            "Volumes": null,
            "WorkingDir": "/",
            "Entrypoint": null,
            "OnBuild": null,
            "Labels": {}
        },
        "NetworkSettings": {
            "Bridge": "",
            "SandboxID": "cbe13b3c604597cf921f72ed044f26808f97440a793ae9e77f2127ff3d35030a",
            "SandboxKey": "/var/run/docker/netns/cbe13b3c6045",
            "Ports": {},
            "HairpinMode": false,
            "LinkLocalIPv6Address": "",
            "LinkLocalIPv6PrefixLen": 0,
            "SecondaryIPAddresses": null,
            "SecondaryIPv6Addresses": null,
            "EndpointID": "c0dc1c98783b149c31b7e719069c252f1aff4b9d5c46490d0b94869d9721e5a6",
            "Gateway": "172.17.0.1",
            "GlobalIPv6Address": "",
            "GlobalIPv6PrefixLen": 0,
            "IPAddress": "172.17.0.2",
            "IPPrefixLen": 16,
```

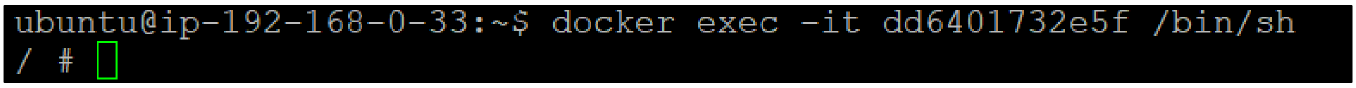


```
            "IPv6Gateway": "",
            "MacAddress": "02:42:ac:11:00:02",
            "Networks": {
                "bridge": {
                    "IPAMConfig": null,
                    "Links": null,
                    "Aliases": null,
                    "MacAddress": "02:42:ac:11:00:02",
                    "NetworkID": "3c4b4fc31520785edccb796418824ef8447078b6964de60f3b60d2dbe2a80f58",
                    "EndpointID": "c0dc1c98783b149c31b7e719069c252f1aff4b9d5c46490d0b94869d9721e5a6",
                    "Gateway": "172.17.0.1",
                    "IPAddress": "172.17.0.2",
                    "IPPrefixLen": 16,
                    "IPv6Gateway": "",
                    "GlobalIPv6Address": "",
                    "GlobalIPv6PrefixLen": 0,
                    "DriverOpts": null,
                    "DNSNames": null
                }
            }
        }
    }
]
ubuntu@ip-192-168-0-33:~$
```

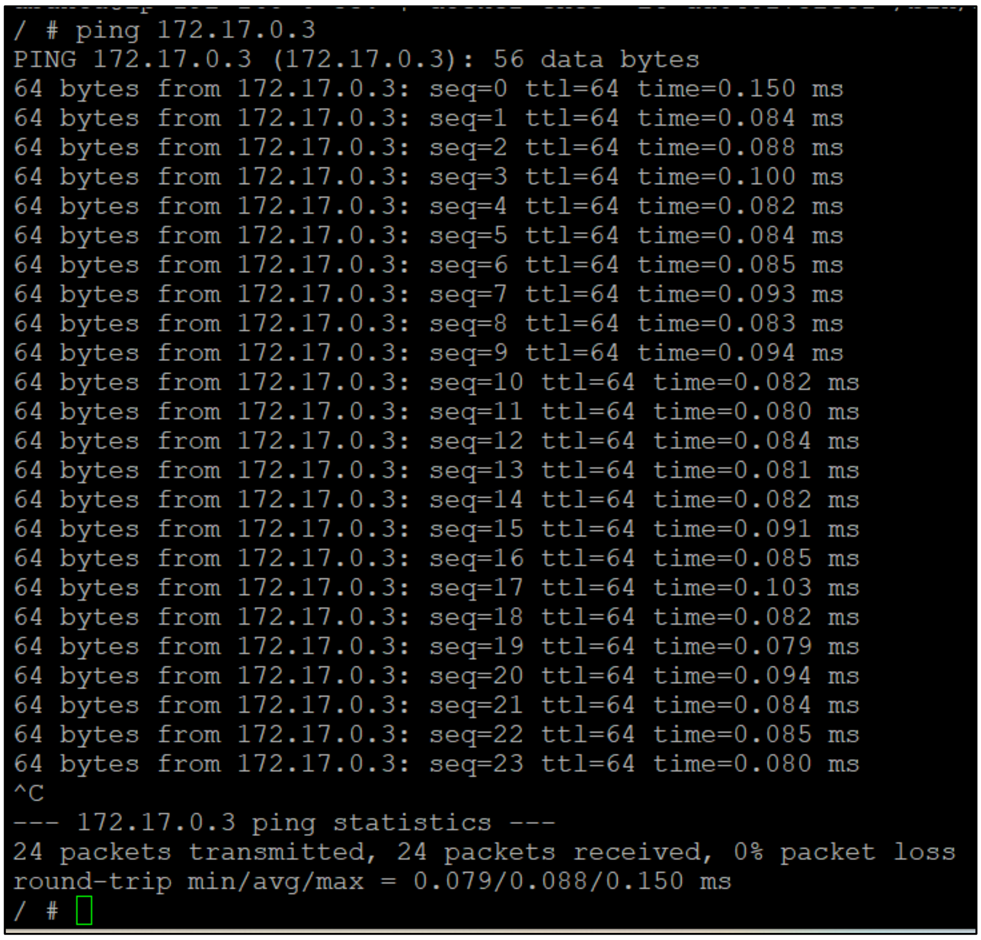
Upon inspecting both containers, we find that both are present inside the Bridge network by default.

Upon inspecting both containers, we observed that the IP addresses of the containers are as follows: for **container `a1`**, **the IP address is `172.17.0.2`**, and for **container `a2`, the IP address is `172.17.0.3`.**

- Now, **enter** the **`a1` container by executing** the command `docker exec -it <container_id_of_a1> /bin/sh`. **This command allows you to open an interactive shell (`/bin/sh`) inside the `a1` container**, enabling you to run commands directly within the container's environment.
- Once **inside the `a1` container**, try to **ping the IP address of container `a2`** by using the `ping 172.17.0.3` command. **Since both containers are on the same `bridge` network**, **the `a1` container will be able to reach the `a2` container**, and **the ping request will succeed, confirming the network connectivity between the two containers**.



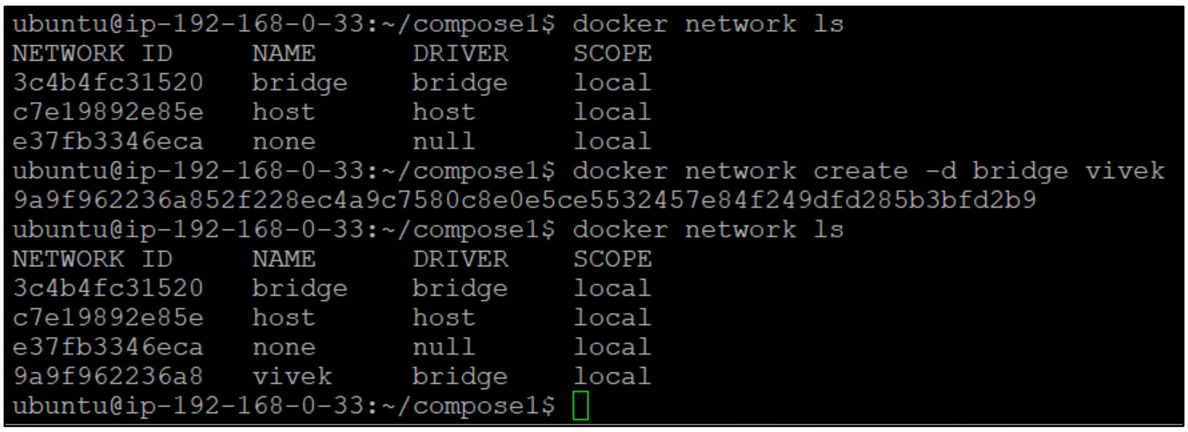
```
ubuntu@ip-192-168-0-33:~$ docker exec -it dd6401732e5f /bin/sh
/ #
```



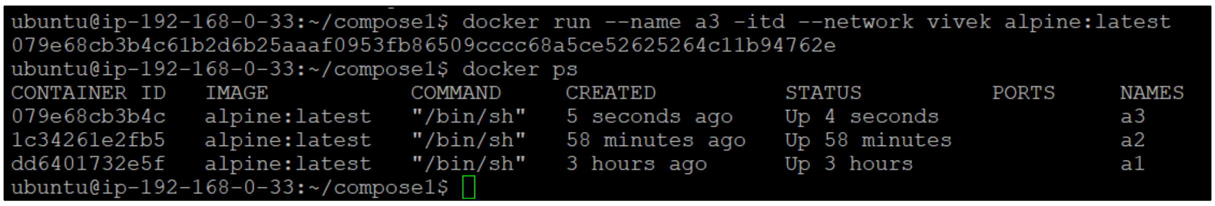
```
/ # ping 172.17.0.3
PING 172.17.0.3 (172.17.0.3): 56 data bytes
64 bytes from 172.17.0.3: seq=0 ttl=64 time=0.150 ms
64 bytes from 172.17.0.3: seq=1 ttl=64 time=0.084 ms
64 bytes from 172.17.0.3: seq=2 ttl=64 time=0.088 ms
64 bytes from 172.17.0.3: seq=3 ttl=64 time=0.100 ms
64 bytes from 172.17.0.3: seq=4 ttl=64 time=0.082 ms
64 bytes from 172.17.0.3: seq=5 ttl=64 time=0.084 ms
64 bytes from 172.17.0.3: seq=6 ttl=64 time=0.085 ms
64 bytes from 172.17.0.3: seq=7 ttl=64 time=0.093 ms
64 bytes from 172.17.0.3: seq=8 ttl=64 time=0.083 ms
64 bytes from 172.17.0.3: seq=9 ttl=64 time=0.094 ms
64 bytes from 172.17.0.3: seq=10 ttl=64 time=0.082 ms
64 bytes from 172.17.0.3: seq=11 ttl=64 time=0.080 ms
64 bytes from 172.17.0.3: seq=12 ttl=64 time=0.084 ms
64 bytes from 172.17.0.3: seq=13 ttl=64 time=0.081 ms
64 bytes from 172.17.0.3: seq=14 ttl=64 time=0.082 ms
64 bytes from 172.17.0.3: seq=15 ttl=64 time=0.091 ms
64 bytes from 172.17.0.3: seq=16 ttl=64 time=0.085 ms
64 bytes from 172.17.0.3: seq=17 ttl=64 time=0.103 ms
64 bytes from 172.17.0.3: seq=18 ttl=64 time=0.082 ms
64 bytes from 172.17.0.3: seq=19 ttl=64 time=0.079 ms
64 bytes from 172.17.0.3: seq=20 ttl=64 time=0.094 ms
64 bytes from 172.17.0.3: seq=21 ttl=64 time=0.084 ms
64 bytes from 172.17.0.3: seq=22 ttl=64 time=0.085 ms
64 bytes from 172.17.0.3: seq=23 ttl=64 time=0.080 ms
^C
--- 172.17.0.3 ping statistics ---
24 packets transmitted, 24 packets received, 0% packet loss
round-trip min/avg/max = 0.079/0.088/0.150 ms
/ #
```

- Now, let's **create a new bridge network called `vivek`** by executing the command `docker network create -d bridge vivek`. **Once the network is created**, we will **launch a new container `a3`** with the following command: `docker run --name a3 -`

`itd --network vivek alpine:latest`. This ensures that the container `a3` is connected to the newly created `vivek` network.



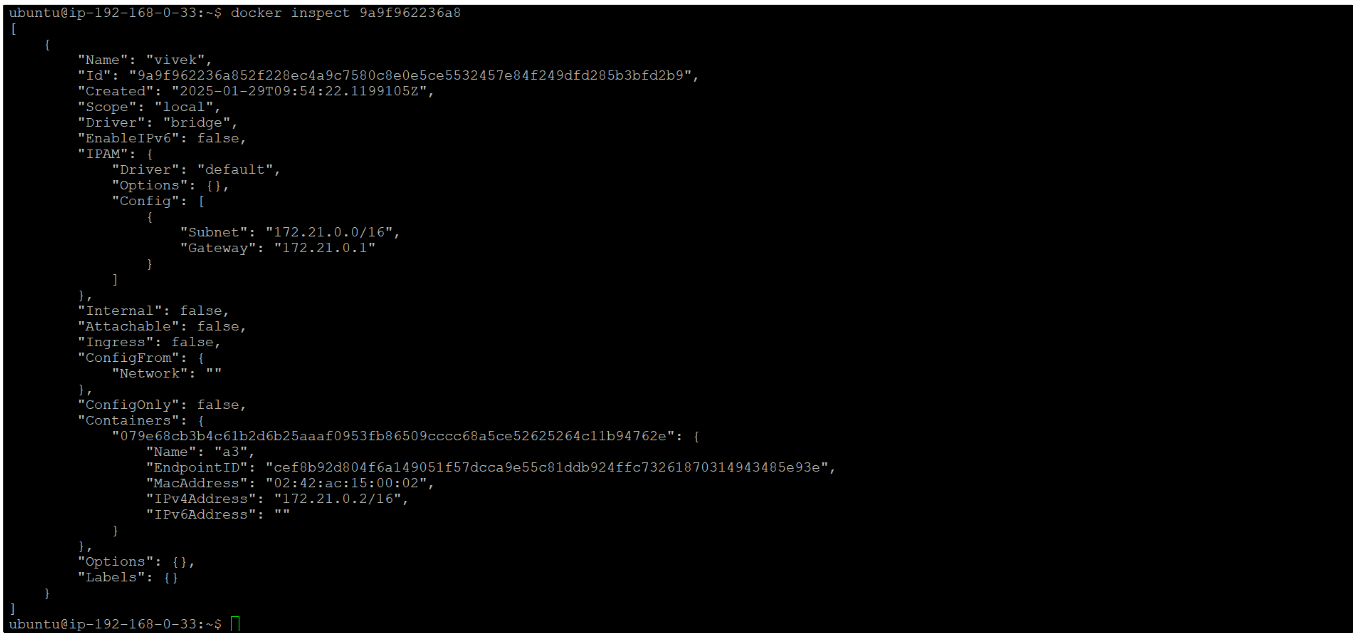
```
ubuntu@ip-192-168-0-33:~/compose1$ docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
3c4b4fc31520   bridge    bridge    local
c7e19892e85e   host      host      local
e37fb3346eca   none      null      local
ubuntu@ip-192-168-0-33:~/compose1$ docker network create -d bridge vivek
9a9f962236a852f228ec4a9c7580c8e0e5ce5532457e84f249dfd285b3bfd2b9
ubuntu@ip-192-168-0-33:~/compose1$ docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
3c4b4fc31520   bridge    bridge    local
c7e19892e85e   host      host      local
e37fb3346eca   none      null      local
9a9f962236a8   vivek     bridge    local
ubuntu@ip-192-168-0-33:~/compose1$
```



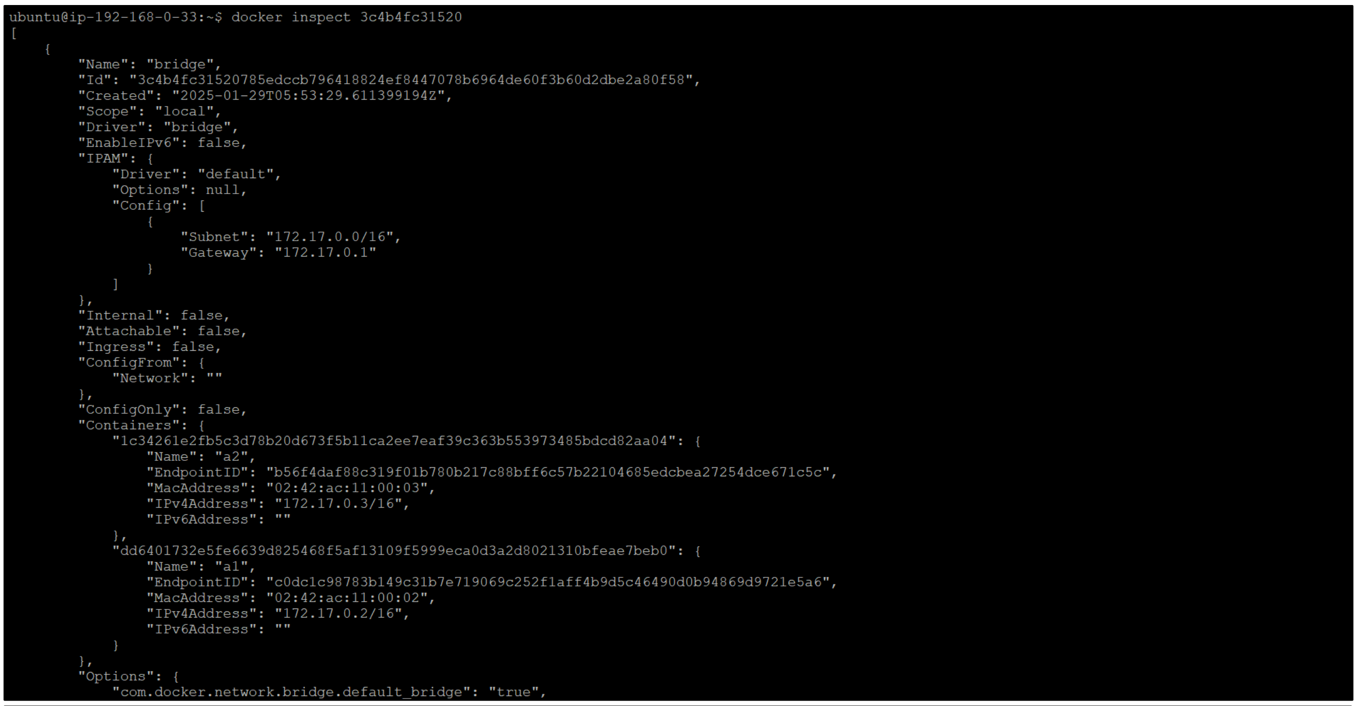
```
ubuntu@ip-192-168-0-33:~/compose1$ docker run --name a3 -itd --network vivek alpine:latest
079e68cb3b4c61b2d6b25aaaf0953fb86509cccc68a5ce52625264c11b94762e
ubuntu@ip-192-168-0-33:~/compose1$ docker ps
CONTAINER ID   IMAGE           COMMAND     CREATED          STATUS          PORTS     NAMES
079e68cb3b4c   alpine:latest   "/bin/sh"   5 seconds ago    Up 4 seconds              a3
1c34261e2fb5   alpine:latest   "/bin/sh"   58 minutes ago   Up 58 minutes             a2
dd6401732e5f   alpine:latest   "/bin/sh"   3 hours ago      Up 3 hours                a1
ubuntu@ip-192-168-0-33:~/compose1$
```

- Now, let's inspect both the `bridge` and `vivek` networks.

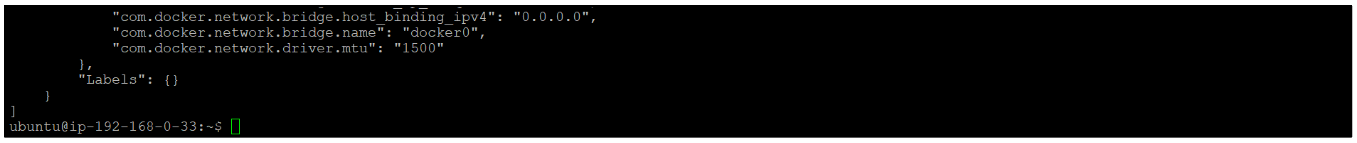
  To inspect the `bridge` and `vivek` networks using their network IDs, first, you can list all available networks using the command `docker network ls` to get the network IDs. Once you have the network IDs, you can inspect the `bridge` network with the command `docker inspect 3c4b4fc31520` and the `vivek` network with the command `docker inspect 9a9f962236a8`. This will provide detailed information about each network.



```
ubuntu@ip-192-168-0-33:~$ docker inspect 9a9f962236a8
[
    {
        "Name": "vivek",
        "Id": "9a9f962236a852f228ec4a9c7580c8e0e5ce5532457e84f249dfd285b3bfd2b9",
        "Created": "2025-01-29T09:54:22.1199105Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.21.0.0/16",
                    "Gateway": "172.21.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "079e68cb3b4c61b2d6b25aaaf0953fb86509cccc68a5ce52625264c11b94762e": {
                "Name": "a3",
                "EndpointID": "cef8b92d804f6a149051f57dcca9e55c81ddb924ffc732618703149434485e93e",
                "MacAddress": "02:42:ac:15:00:02",
                "IPv4Address": "172.21.0.2/16",
                "IPv6Address": ""
            }
        },
        "Options": {},
        "Labels": {}
    }
]
ubuntu@ip-192-168-0-33:~$
```



```
ubuntu@ip-192-168-0-33:~$ docker inspect 3c4b4fc31520
[
    {
        "Name": "bridge",
        "Id": "3c4b4fc31520785edccb796418824ef8447078b6964de60f3b60d2dbe2a80f58",
        "Created": "2025-01-29T05:53:29.611399194Z",
        "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": {
            "1c34261e2fb5c3d78b20d673f5b11ca2ee7eaf39c363b553973485bdcd82aa04": {
                "Name": "a2",
                "EndpointID": "b56f4daf88c319f01b780b217c88bff6c57b22104685edcbea27254dce671c5c",
                "MacAddress": "02:42:ac:11:00:03",
                "IPv4Address": "172.17.0.3/16",
                "IPv6Address": ""
            },
            "dd6401732e5fe6639d825468f5af13109f5999eca0d3a2d8021310bfeae7beb0": {
                "Name": "a1",
                "EndpointID": "c0dc1c98783b149c31b7e719069c252f1aff4b9d5c46490d0b94869d9721e5a6",
                "MacAddress": "02:42:ac:11:00:02",
                "IPv4Address": "172.17.0.2/16",
                "IPv6Address": ""
            }
        },
        "Options": {
            "com.docker.network.bridge.default_bridge": "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": {}
    }
]
ubuntu@ip-192-168-0-33:~$ 
```

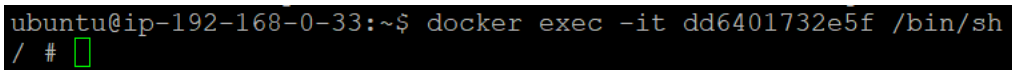
Upon inspection, we will observe that the `bridge` network contains containers `a1` and `a2`, while the `vivek` network contains the newly created container `a3`.

Here's a table displaying the IP addresses of the containers in their respective networks:

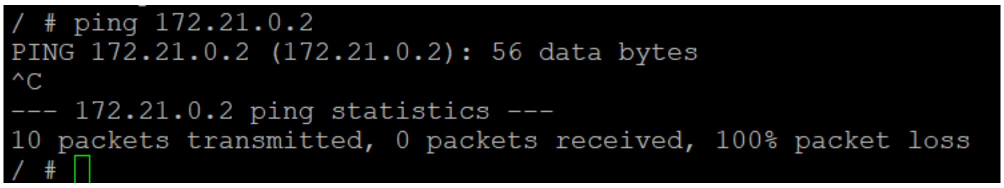
Container	Network	IP Address
a1	bridge	172.17.0.2
a2		172.17.0.3
a3	vivek	172.21.0.2

This table shows that containers `a1` and `a2` are in the `bridge` network with IPs `172.17.0.2` and `172.17.0.3`, respectively, while container `a3` is in the `vivek` network with the IP `172.21.0.2`.

- Now, let's enter the shell of the `a1` container and attempt to ping the IP address of the `a3` container. First, execute the command `docker exec -it dd6401732e5f /bin/sh` to access the shell of the `a1` container. Once inside the shell, run `ping 172.21.0.2` to send a ping request to the IP address of the `a3` container. Since `a1` and `a3` are in different networks (`bridge` and `vivek`), the ping request will likely fail because containers in separate networks are isolated from each other unless specific routing or network configurations are in place to allow communication between them.



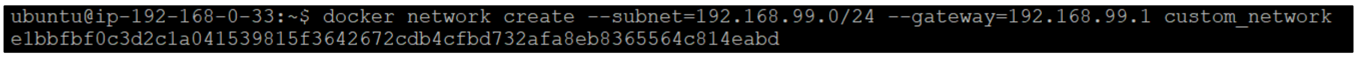
```
ubuntu@ip-192-168-0-33:~$ docker exec -it dd6401732e5f /bin/sh
/ # 
```



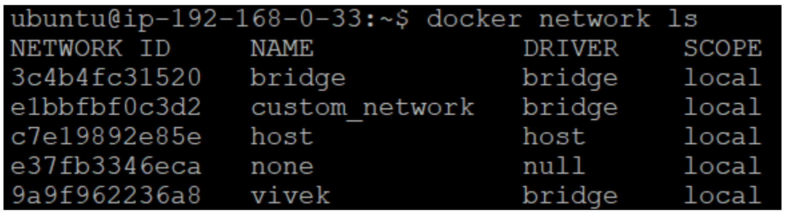
```
/ # ping 172.21.0.2
PING 172.21.0.2 (172.21.0.2): 56 data bytes
^C
--- 172.21.0.2 ping statistics ---
10 packets transmitted, 0 packets received, 100% packet loss
/ #
```

- Now, let's create a new Docker network with a custom subnet and gateway. To do this, execute the following command: `docker network create --subnet=192.168.99.0/24 --gateway=192.168.99.1 custom_network`

This command **creates a new network named `custom_network` with a subnet of `192.168.99.0/24` and a gateway of `192.168.99.1`**. By specifying the `--subnet` and `--gateway` options, we configure the network to use these custom settings, ensuring that the containers connected to this network will use IP addresses within the defined range.

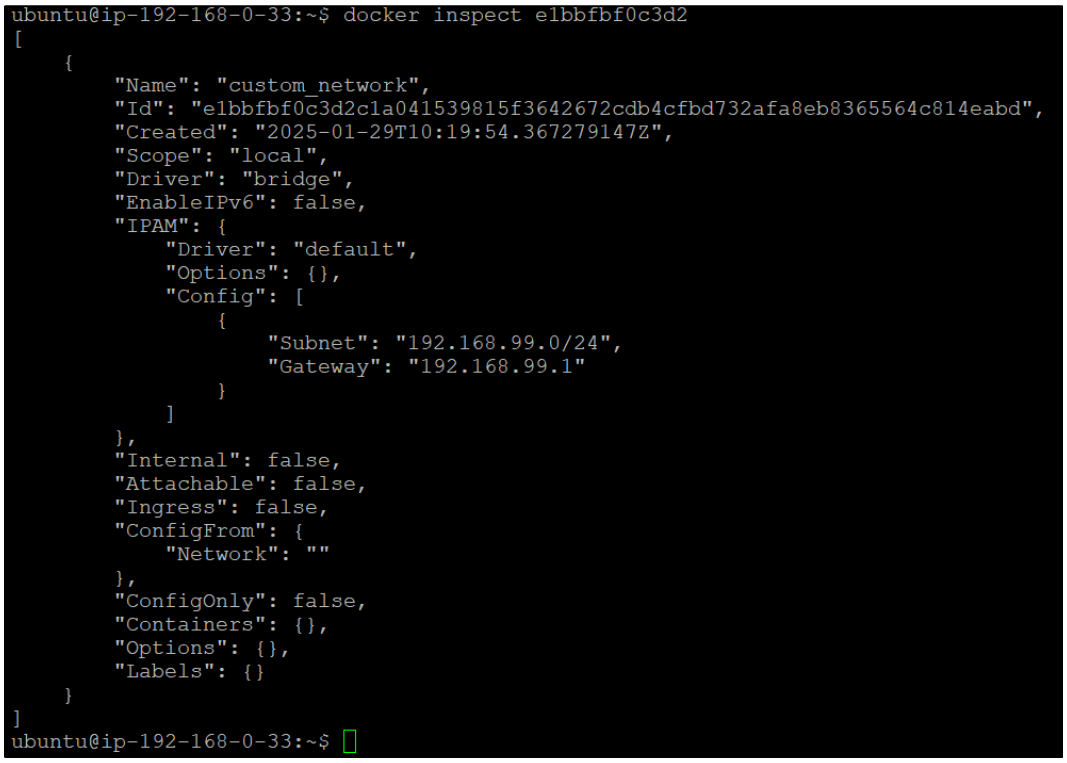


```
ubuntu@ip-192-168-0-33:~$ docker network create --subnet=192.168.99.0/24 --gateway=192.168.99.1 custom_network
e1bbfbf0c3d2c1a041539815f3642672cdb4cfbd732afa8eb8365564c814eabd
```



```
ubuntu@ip-192-168-0-33:~$ docker network ls
NETWORK ID     NAME             DRIVER    SCOPE
3c4b4fc31520   bridge           bridge    local
e1bbfbf0c3d2   custom_network   bridge    local
c7e19892e85e   host             host      local
e37fb3346eca   none             null      local
9a9f962236a8   vivek            bridge    local
```

upon inspecting the new custom_network using its network ID, by executing the command `docker inspect e1bbfbf0c3d2` we get the detailed information about the network, and upon inspection, you will observe that it has the same subnet (`192.168.99.0/24`) and gateway (`192.168.99.1`) that were allocated during the creation of the network. This confirms that the network was set up correctly with the specified configuration.



```
ubuntu@ip-192-168-0-33:~$ docker inspect e1bbfbf0c3d2
[
    {
        "Name": "custom_network",
        "Id": "e1bbfbf0c3d2c1a041539815f3642672cdb4cfbd732afa8eb8365564c814eabd",
        "Created": "2025-01-29T10:19:54.367279147Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "192.168.99.0/24",
                    "Gateway": "192.168.99.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {},
        "Options": {},
        "Labels": {}
    }
]
ubuntu@ip-192-168-0-33:~$
```