Docker Hands-On Assignment

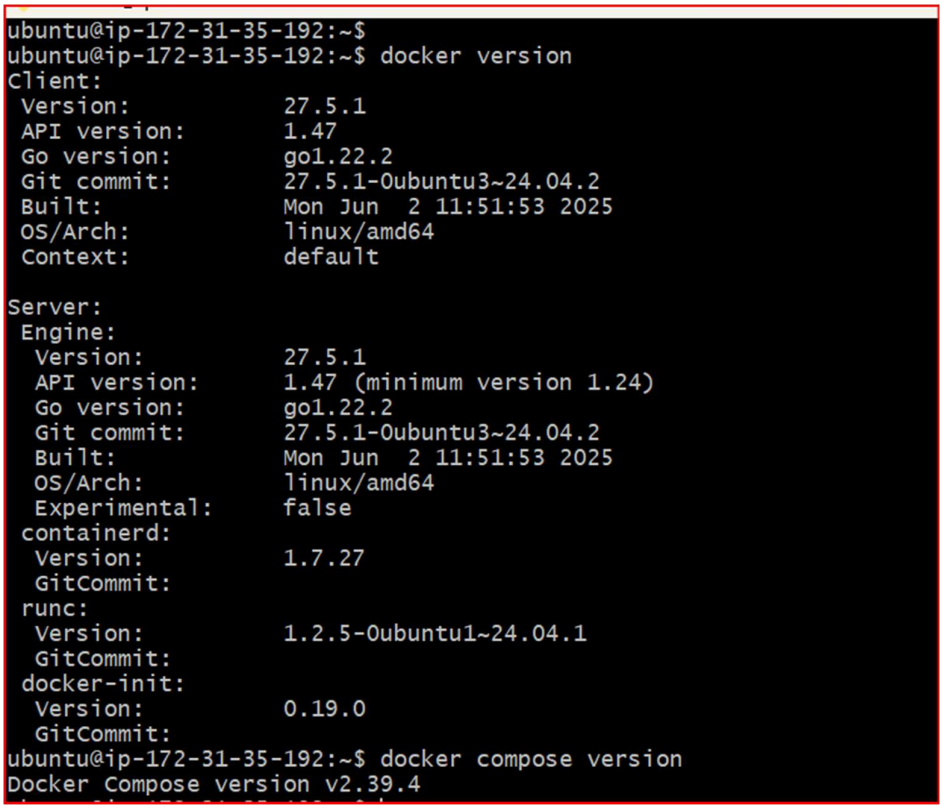
Everything You Need to Know About Docker

Objective

Master Docker fundamentals by building, running, and managing containers. Learn Docker networking, volumes, and image creation through hands-on practice.

Prerequisites

- Docker installed on your machine (Docker Desktop for Mac/Windows or Docker Engine for Linux)
- Basic command line knowledge
- Text editor of your choice



Part 1: Docker Basics (30 minutes)

Task 1.1: Running Your First Containers

1. **Pull and run basic containers:**

```
docker run -d -t --name my-alpine alpine
docker run -d -t --name my-busybox busybox
```

```
ubuntu@ip-172-31-35-192:~$ docker run -d -t --name my-alpine alpine
440d89e9c8a3e93c732c720b04d73ac0c2a1f236f15bfa8a89285ca9a0dcffae
ubuntu@ip-172-31-35-192:~$ docker run -d -t --name my-busybox busybox
ba7dafab9048980c0634badfad10b18f565718152c8f26a941ea4450314ad0e3
ubuntu@ip-172-31-35-192:~$
```

2. **List all containers:**

```
docker ps
docker ps -a
```

```
ubuntu@ip-172-31-35-192:~$ docker ps
CONTAINER ID   IMAGE     COMMAND     CREATED          STATUS          PORTS     NAMES
ba7dafab9048   busybox   "sh"        26 minutes ago   Up 26 minutes             my-busybox
440d89e9c8a3   alpine    "/bin/sh"   26 minutes ago   Up 26 minutes             my-alpine
ubuntu@ip-172-31-35-192:~$ docker ps -a
CONTAINER ID   IMAGE     COMMAND     CREATED          STATUS          PORTS     NAMES
ba7dafab9048   busybox   "sh"        26 minutes ago   Up 26 minutes             my-busybox
440d89e9c8a3   alpine    "/bin/sh"   26 minutes ago   Up 26 minutes             my-alpine
ubuntu@ip-172-31-35-192:~$
```

3. **Check downloaded images:**

```
docker image ls
```

```
ubuntu@ip-172-31-35-192:~$ docker image ls
REPOSITORY    TAG       IMAGE ID       CREATED         SIZE
nginx         latest    41f689c20910   6 weeks ago     192MB
hello-world   latest    1b44b5a3e06a   6 weeks ago     10.1kB
alpine        latest    9234e8fb04c4   2 months ago    8.31MB
busybox       latest    0ed463b26dae   12 months ago   4.43MB
ubuntu@ip-172-31-35-192:~$
```

Questions to Answer:

Q: What's the difference between docker ps and docker ps -a?

docker ps shows only running containers.

docker ps -a shows all running as well as exited containers.

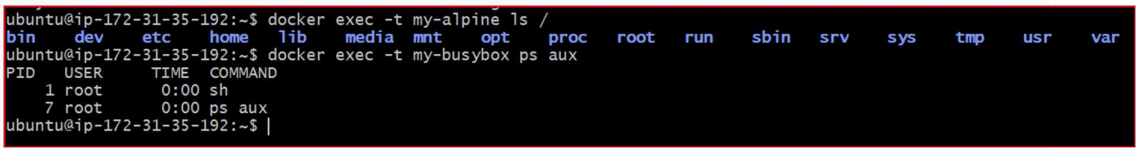
Q: Why are Alpine and BusyBox images so small?

They contain only the required binaries and libraries, without extra tools or packages.

Task 1.2: Container Interaction

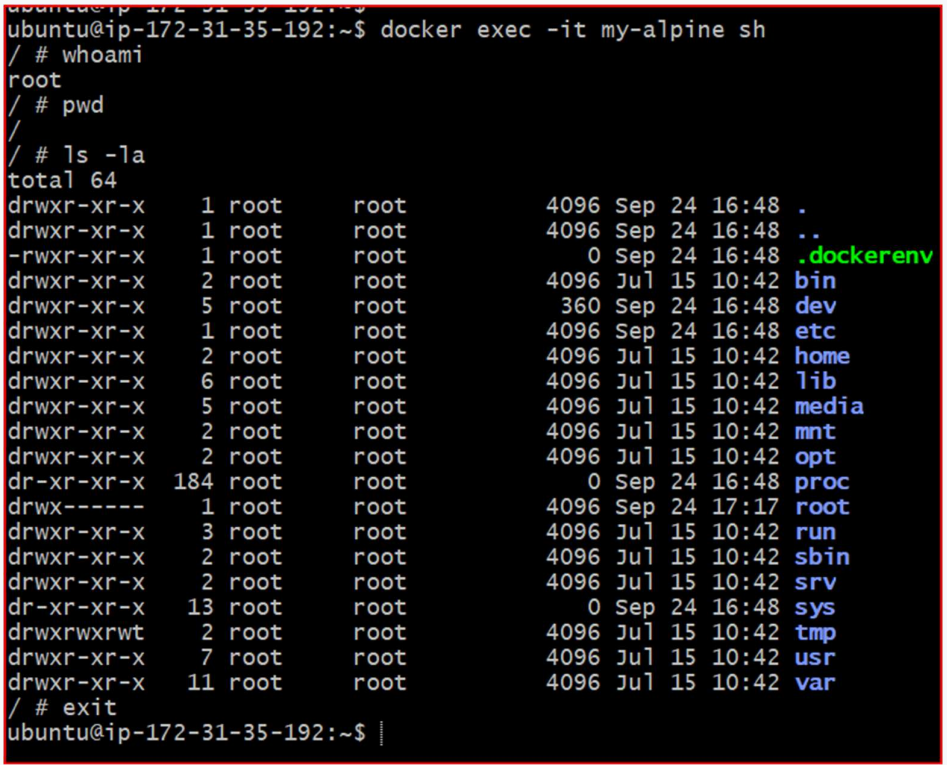
1. **Execute commands in running containers:**

```
docker exec -t my-alpine ls /
docker exec -t my-busybox ps aux
```



2. **Open interactive shell sessions:**

```
docker exec -it my-alpine sh
# Inside container: run `whoami`, `pwd`, `ls -la`
# Type `exit` to leave
```



3. **Container lifecycle management:**

```
docker stop my-alpine
docker start my-alpine
docker rm -f my-busybox
```

```
ubuntu@ip-172-31-35-192:~$
ubuntu@ip-172-31-35-192:~$ docker stop my-alpine
my-alpine
ubuntu@ip-172-31-35-192:~$ docker start my-alpine
my-alpine
ubuntu@ip-172-31-35-192:~$ docker rm -f my-busybox
my-busybox
ubuntu@ip-172-31-35-192:~$ docker ps -a
CONTAINER ID   IMAGE     COMMAND     CREATED          STATUS          PORTS     NAMES
440d89e9c8a3   alpine    "/bin/sh"   30 minutes ago   Up 18 seconds             my-alpine
ubuntu@ip-172-31-35-192:~$
```

Deliverable: Take screenshots of your container interactions and note the differences between Alpine and BusyBox.

Part 2: Docker Networking (45 minutes)

Task 2.1: Default Bridge Network

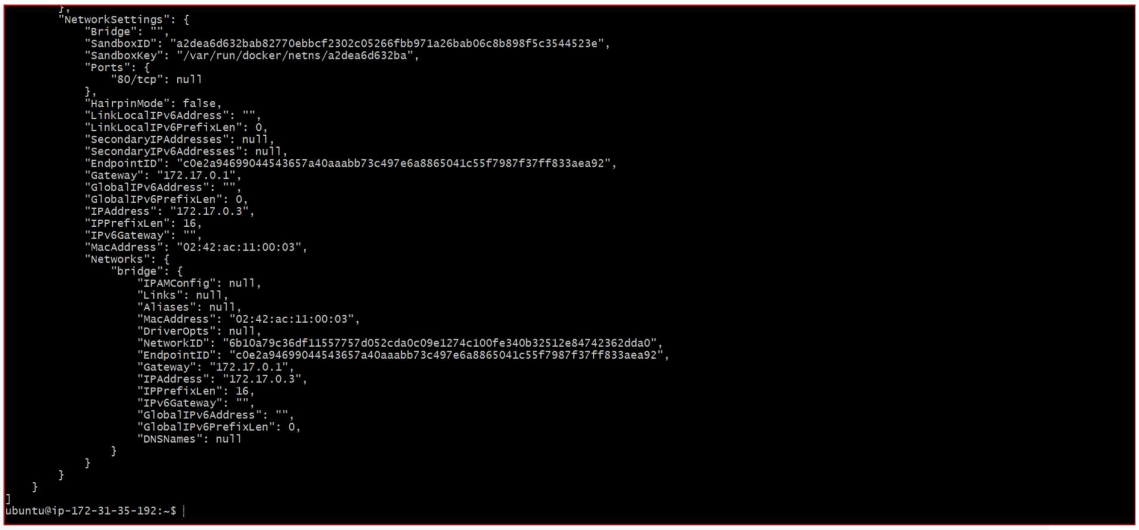
1. **Run Nginx container:**

```
docker run -d --name nginx-default nginx:latest
```

2. **Inspect the container:**

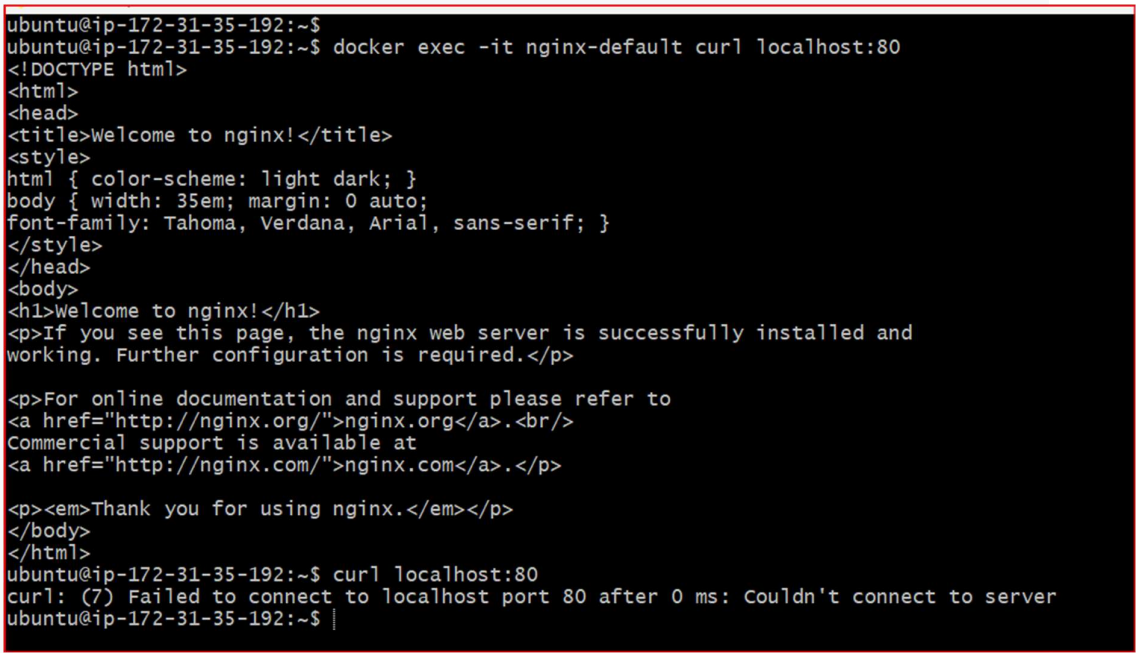
```
docker inspect nginx-default
# Look for NetworkSettings section
```

```
ubuntu@ip-172-31-35-192:~$
ubuntu@ip-172-31-35-192:~$ docker run -d --name nginx-default nginx:latest
60b0c74f33f2d2ce1e06d9c7413b59ab292d5cceae15a9669d53e6d0b424abdd
ubuntu@ip-172-31-35-192:~$ docker inspect nginx-default
[
    {
        "Id": "60b0c74f33f2d2ce1e06d9c7413b59ab292d5cceae15a9669d53e6d0b424abdd",
        "Created": "2025-09-24T17:19:40.662350486Z",
        "Path": "/docker-entrypoint.sh",
        "Args": [
            "nginx",
            "-g",
            "daemon off;"
        ],
        "State": {
            "Status": "running",
            "Running": true,
            "Paused": false,
            "Restarting": false,
            "OOMKilled": false,
            "Dead": false,
            "Pid": 6222,
            "ExitCode": 0,
            "Error": "",
            "StartedAt": "2025-09-24T17:19:40.748503417Z",
            "FinishedAt": "0001-01-01T00:00:00Z"
        },
        "Image": "sha256:41f689c209100e6cadf3ce7fdd02035e90dbd1d586716bf8fc6ea55c365b2d81",
        "ResolvConfPath": "/var/lib/docker/containers/60b0c74f33f2d2ce1e06d9c7413b59ab292d5cceae15a9669d53e6d0b424abdd/resolv.conf",
        "HostnamePath": "/var/lib/docker/containers/60b0c74f33f2d2ce1e06d9c7413b59ab292d5cceae15a9669d53e6d0b424abdd/hostname",
        "HostsPath": "/var/lib/docker/containers/60b0c74f33f2d2ce1e06d9c7413b59ab292d5cceae15a9669d53e6d0b424abdd/hosts",
        "LogPath": "/var/lib/docker/containers/60b0c74f33f2d2ce1e06d9c7413b59ab292d5cceae15a9669d53e6d0b424abdd/60b0c74f33f2d2ce1e06d9c7413b59ab292d5cceae15a9669d53e6d0b424
abdd-json.log",
        "Name": "/nginx-default",
        "RestartCount": 0,
        "Driver": "overlay2",
        "Platform": "linux",
        "MountLabel": "",
        "ProcessLabel": "",
        "AppArmorProfile": "docker-default",
        "ExecIDs": null,
        "HostConfig": {
            "Binds": null,
            "ContainerIDFile": "",
            "LogConfig": {
                "Type": "json-file",
                "Config": {}
            },
```



3. **Test connectivity:**

```
docker exec -it nginx-default curl localhost:80
# Try accessing from host (this should fail):
curl localhost:80
```



Task 2.2: Port Forwarding

1. **Run Nginx with port mapping:**

```
docker rm -f nginx-default
```

```
docker run -d -p 8080:80 --name nginx-exposed nginx:latest
```

```
ubuntu@ip-172-31-35-192:~$ docker rm -f nginx-default
nginx-default
ubuntu@ip-172-31-35-192:~$ docker run -d -p 8080:80 --name nginx-exposed nginx:latest
405a0e53e595b7015751b356ca1d424e0992f5d4a4c34a87d19918832eb33ac6
ubuntu@ip-172-31-35-192:~$ docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS          PORTS                                       NAMES
405a0e53e595   nginx:latest   "/docker-entrypoint.…"   41 seconds ago   Up 40 seconds   0.0.0.0:8080->80/tcp, [::]:8080->80/tcp     nginx-exposed
440d89e9c8a3   alpine         "/bin/sh"                34 minutes ago   Up 4 minutes                                                my-alpine
ubuntu@ip-172-31-35-192:~$
```

2. **Test access:**

```
# From your host machine:
curl localhost:8080
```

```
ubuntu@ip-172-31-35-192:~$ curl localhost:8080
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
ubuntu@ip-172-31-35-192:~$
```

Task 2.3: Custom Bridge Network

1. **Create custom network:**

```
docker network create my-network
docker network ls
```

```
ubuntu@ip-172-31-35-192:~$ docker network create my-network
4d3ba47532230a93364a3158a81e11cf7aedefdab4ebca6de4bf9bd81f1c6b87
ubuntu@ip-172-31-35-192:~$ docker network ls
NETWORK ID     NAME         DRIVER    SCOPE
6b10a79c36df   bridge       bridge    local
b163633679fd   host         host      local
4d3ba4753223   my-network   bridge    local
acf4669decbe   none         null      local
ubuntu@ip-172-31-35-192:~$
```

2. **Run containers in custom network:**

```
docker run -d --network my-network --name web-server nginx:latest
docker run -it --network my-network --name client alpine sh
```

Test name resolution:

```
# Inside the Alpine container:
ping web-server
wget -qO- http://web-server
```

```
ubuntu@ip-172-31-35-192:~$ docker run -d --network my-network --name web-server nginx:latest
01a4a001a6f609b6b7bbd3af2148dbc010fec741fd62a17eaecec6d4f43bc5d9
ubuntu@ip-172-31-35-192:~$ docker run -it --network my-network --name client alpine sh
/ # ping web-server
PING web-server (172.18.0.2): 56 data bytes
64 bytes from 172.18.0.2: seq=0 ttl=64 time=0.086 ms
64 bytes from 172.18.0.2: seq=1 ttl=64 time=0.067 ms
^Z[1]+  Stopped                    ping web-server
/ # wget -qO- http://web-server
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
/ #
```

Questions to Answer:

Q: Why can containers ping each other by name in custom networks but not in the default bridge?

Default bridge → No name resolution (must use container IP).

Custom bridge → Built-in DNS allows containers to ping each other by name.

Q: What happens when you try to access the web server from your host machine in the custom network?

We cannot reach container because port is not published. Inorder to reach the container we have to publish the port using -p.

Part 3: Docker Volumes (30 minutes)

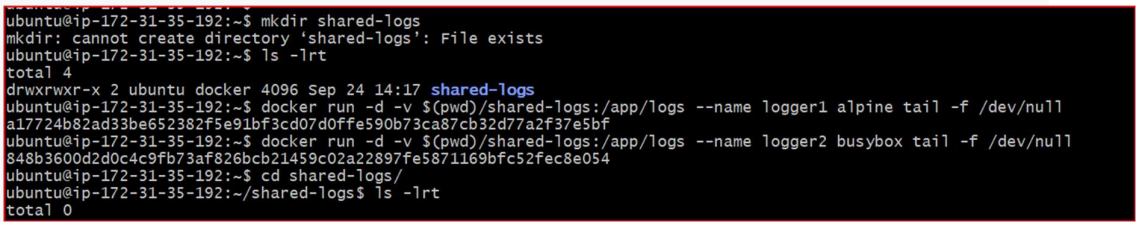
Task 3.1: Bind Mounts

1. **Create a directory on your host:**

```
mkdir shared-logs
```

2. **Run containers with bind mount:**

```
docker run -d -v $(pwd)/shared-logs:/app/logs --name logger1 alpine tail -f
/dev/null
docker run -d -v $(pwd)/shared-logs:/app/logs --name logger2 busybox tail -
f /dev/null
```

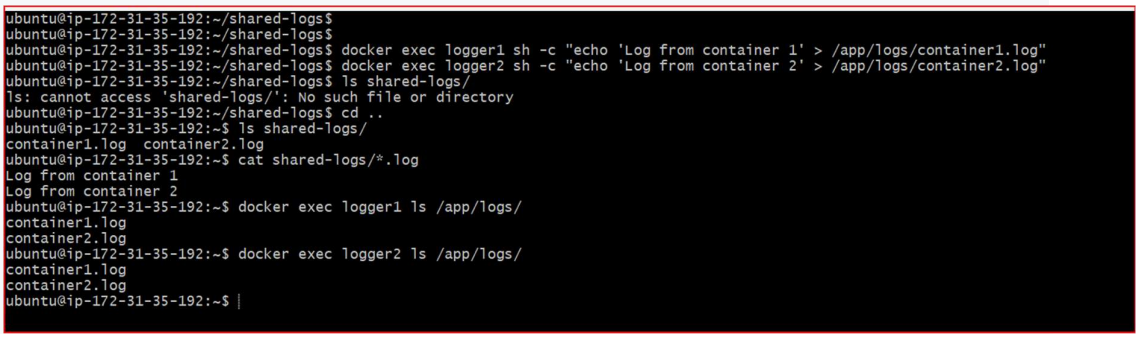


3. **Create files from containers:**

```
docker exec logger1 sh -c "echo 'Log from container 1' >
/app/logs/container1.log"
docker exec logger2 sh -c "echo 'Log from container 2' >
/app/logs/container2.log"
```

4. **Verify file sharing:**

```
# Check from host:
ls shared-logs/
cat shared-logs/*.log
# Check from containers:
docker exec logger1 ls /app/logs/
docker exec logger2 ls /app/logs/
```



Task 3.2: Docker Volumes

1. **Create and use Docker volume:**

```
docker volume create app-data
docker volume ls
docker volume inspect app-data
```

```
ubuntu@ip-172-31-35-192:~$
ubuntu@ip-172-31-35-192:~$ docker volume create app-data
app-data
ubuntu@ip-172-31-35-192:~$ docker volume ls
DRIVER    VOLUME NAME
local     app-data
ubuntu@ip-172-31-35-192:~$ docker volume inspect app-data
[
    {
        "CreatedAt": "2025-09-24T17:35:06Z",
        "Driver": "local",
        "Labels": null,
        "Mountpoint": "/var/lib/docker/volumes/app-data/_data",
        "Name": "app-data",
        "Options": null,
        "Scope": "local"
    }
]
ubuntu@ip-172-31-35-192:~$
```

2. **Mount volume in containers:**

```
docker run -d --mount source=app-data,target=/data --name data1 alpine tail
-f /dev/null
docker run -d --mount source=app-data,target=/data --name data2
nginx:latest
```

```
ubuntu@ip-172-31-35-192:~$
ubuntu@ip-172-31-35-192:~$ docker run -d --mount source=app-data,target=/data --name data1 alpine tail -f /dev/null
cb39e42f6702d9a72919c1ea9f3a4f85d08b9fcb5b7ec530f67ba6d92673cf96
ubuntu@ip-172-31-35-192:~$ docker run -d --mount source=app-data,target=/data --name data2 nginx:latest
4b8d844d0c5a8d110340de533da122dde673ab6ffbe7d86ca5768103d05c9f06
ubuntu@ip-172-31-35-192:~$
```

3. **Test data persistence:**

```
docker exec data1 sh -c "echo 'Persistent data' > /data/test.txt"
docker exec data2 cat /data/test.txt
```

```
ubuntu@ip-172-31-35-192:~$ docker exec data1 sh -c "echo 'Persistent data' > /data/test.txt"
ubuntu@ip-172-31-35-192:~$ docker exec data2 cat /data/test.txt
Persistent data
```

Data Persistence: When would you choose bind mounts over Docker volumes and vice versa?

Ans:

- Bind mounts are used for development and direct file access; Docker volumes used for production and managed persistence.
- Bind mounts need specific hostfile path. Docker volumes means docker will manage the storage life cycle

Part 4: Building Docker Images (60 minutes)

Task 4.1: Create a Flask Application

1. **Create project directory:**

```
mkdir flask-docker-app
cd flask-docker-app
```

2. **Create application files:**

   app.py:

```
from flask import Flask, jsonify
import os

app = Flask(__name__)

@app.route('/')
def hello():
    return jsonify({
        "message": "Hello from Docker!",
        "container_id": os.environ.get('HOSTNAME', 'unknown')
    })

@app.route('/health')
def health():
    return jsonify({"status": "healthy"})

if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0', port=5000)
```

   requirements.txt:

```
Flask==2.3.3
```

   Dockerfile:

```
FROM python:3.11-slim

WORKDIR /app

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

COPY app.py .

EXPOSE 5000

CMD ["python", "app.py"]
```

Task 4.2: Build and Test Image

1. **Build the image:**

```
docker build -t my-flask-app:v1.0 .
```

```
ubuntu@ip-172-31-35-192:~/flask-docker-app$ docker build -t my-flask-app:v1.0 .
DEPRECATED: The legacy builder is deprecated and will be removed in a future release.
            Install the buildx component to build images with BuildKit:
            https://docs.docker.com/go/buildx/

Sending build context to Docker daemon  4.096kB
Step 1/7 : FROM python:3.11-slim
3.11-slim: Pulling from library/python
ce1261c6d567: Pull complete
11b89692b208: Pull complete
764e05fe66b6: Pull complete
a4aefcec16c5: Pull complete
Digest: sha256:a0939570b38cddeb861b8e75d20b1c8218b21562b18f301171904b544e8cf228
Status: Downloaded newer image for python:3.11-slim
 ---> c4640ec0986f
Step 2/7 : WORKDIR /app
 ---> Running in a0ccb4c86e02
 ---> Removed intermediate container a0ccb4c86e02
 ---> 7a64fca8ca65
Step 3/7 : COPY requirements.txt .
 ---> 70f4ce0d7941
Step 4/7 : RUN pip install --no-cache-dir -r requirements.txt
 ---> Running in e3f194d60ee6
Collecting Flask==2.3.3 (from -r requirements.txt (line 1))
  Downloading flask-2.3.3-py3-none-any.whl.metadata (3.6 kB)
Collecting Werkzeug>=2.3.7 (from Flask==2.3.3->-r requirements.txt (line 1))
  Downloading werkzeug-3.1.3-py3-none-any.whl.metadata (3.7 kB)
Collecting Jinja2>=3.1.2 (from Flask==2.3.3->-r requirements.txt (line 1))
  Downloading jinja2-3.1.6-py3-none-any.whl.metadata (2.9 kB)
Collecting itsdangerous>=2.1.2 (from Flask==2.3.3->-r requirements.txt (line 1))
  Downloading itsdangerous-2.2.0-py3-none-any.whl.metadata (1.9 kB)
Collecting click>=8.1.3 (from Flask==2.3.3->-r requirements.txt (line 1))
  Downloading click-8.3.0-py3-none-any.whl.metadata (2.6 kB)
Collecting blinker>=1.6.2 (from Flask==2.3.3->-r requirements.txt (line 1))
  Downloading blinker-1.9.0-py3-none-any.whl.metadata (1.6 kB)
Collecting MarkupSafe>=2.0 (from Jinja2>=3.1.2->Flask==2.3.3->-r requirements.txt (line 1))
  Downloading MarkupSafe-3.0.2-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (4.0 kB)
Downloading flask-2.3.3-py3-none-any.whl (96 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 96.1/96.1 kB 24.7 MB/s eta 0:00:00
Downloading blinker-1.9.0-py3-none-any.whl (8.5 kB)
Downloading click-8.3.0-py3-none-any.whl (107 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 107.3/107.3 kB 274.5 MB/s eta 0:00:00
Downloading itsdangerous-2.2.0-py3-none-any.whl (16 kB)
Downloading jinja2-3.1.6-py3-none-any.whl (134 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 134.9/134.9 kB 309.1 MB/s eta 0:00:00
Downloading werkzeug-3.1.3-py3-none-any.whl (224 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 224.5/224.5 kB 333.3 MB/s eta 0:00:00
Downloading MarkupSafe-3.0.2-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (23 kB)
Installing collected packages: MarkupSafe, itsdangerous, click, blinker, Werkzeug, Jinja2, Flask
Successfully installed Flask-2.3.3 Jinja2-3.1.6 MarkupSafe-3.0.2 Werkzeug-3.1.3 blinker-1.9.0 click-8.3.0 itsdangerous-2.2.0
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is recommended to use a virtual envi
onment instead: https://pip.pypa.io/warnings/venv

[notice] A new release of pip is available: 24.0 -> 25.2
[notice] To update, run: pip install --upgrade pip
 ---> Removed intermediate container e3f194d60ee6
 ---> a465ac3e0428
Step 5/7 : COPY app.py .
 ---> d2544b69a2cd
Step 6/7 : EXPOSE 5000
 ---> Running in 63c56a9be509
 ---> Removed intermediate container 63c56a9be509
 ---> eac834e6ce10
Step 7/7 : CMD ["python", "app.py"]
 ---> Running in 7086dcf15e1f
 ---> Removed intermediate container 7086dcf15e1f
 ---> b8266b4d2c1c
Successfully built b8266b4d2c1c
Successfully tagged my-flask-app:v1.0
ubuntu@ip-172-31-35-192:~/flask-docker-app$
```

```
ubuntu@ip-172-31-35-192:~/flask-docker-app$ docker image ls
REPOSITORY     TAG         IMAGE ID       CREATED              SIZE
my-flask-app   v1.0        b8266b4d2c1c   About a minute ago   140MB
nginx          latest      41f689c20910   6 weeks ago          192MB
hello-world    latest      1b44b5a3e06a   6 weeks ago          10.1kB
python         3.11-slim   c4640ec0986f   6 weeks ago          125MB
alpine         latest      9234e8fb04c4   2 months ago         8.31MB
busybox        latest      0ed463b26dae   12 months ago        4.43MB
ubuntu@ip-172-31-35-192:~/flask-docker-app$
```

2. **Run container:**

```
docker run -d -p 5000:5000 --name flask-app my-flask-app:v1.0
```

3. **Test the application:**

```
curl localhost:5000
curl localhost:5000/health
```

```
ubuntu@ip-172-31-35-192:~/flask-docker-app$
ubuntu@ip-172-31-35-192:~/flask-docker-app$ docker run -d -p 5000:5000 --name flask-app my-flask-app:v1.0
b65310ab29e7e0ec673e434298a0769ac17bafa6f4daea829ece1ffb68747b59
ubuntu@ip-172-31-35-192:~/flask-docker-app$ curl localhost:5000
{
  "container_id": "b65310ab29e7",
  "message": "Hello from Docker!"
}
ubuntu@ip-172-31-35-192:~/flask-docker-app$ curl localhost:5000/health
{
  "status": "healthy"
}
ubuntu@ip-172-31-35-192:~/flask-docker-app$
```

Task 4.3: Multi-container Application

1. **Create docker-compose.yml:**

```yaml
version: '3.8'
services:
  web:
    build: .
    ports:
      - "5000:5000"
    volumes:
      - app-logs:/app/logs
    networks:
      - app-network

  redis:
    image: redis:alpine
    networks:
      - app-network

volumes:
  app-logs:

networks:
    app-network:
```

2. **Run with docker-compose:**

```
docker-compose up -d
docker-compose ps
```

```
ubuntu@ip-172-31-35-192:~/flask-docker-app$ docker-compose up -d
Starting flask-docker-app_web_1 ...
Starting flask-docker-app_web_1 ... done
ubuntu@ip-172-31-35-192:~/flask-docker-app$ docker-compose ps
          Name                        Command               State                    Ports
-------------------------------------------------------------------------------------------------------
flask-docker-app_redis_1   docker-entrypoint.sh redis ...   Up      6379/tcp
flask-docker-app_web_1     python app.py                    Up      0.0.0.0:5000->5000/tcp,:::5000->5000/tcp
ubuntu@ip-172-31-35-192:~/flask-docker-app$
```

Part 5: Image Registry (30 minutes)

Task 5.1: Push to Docker Hub

1. **Create Docker Hub account** (if you don't have one)
2. **Login to Docker Hub:**

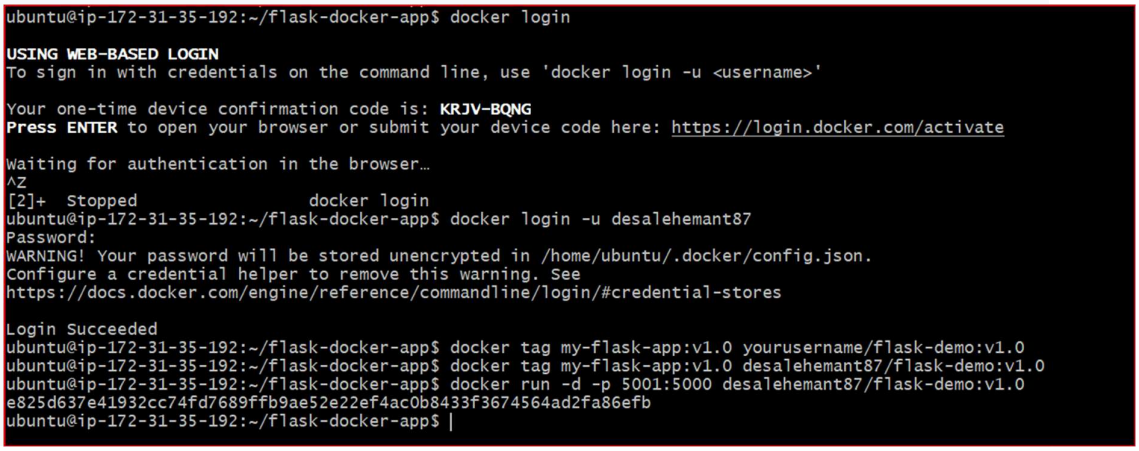
```
docker login
```

3. **Tag and push image:**

```
docker tag my-flask-app:v1.0 yourusername/flask-demo:v1.0
docker push yourusername/flask-demo:v1.0
```

4. **Test pulling image:**

```
docker rmi my-flask-app:v1.0 yourusername/flask-demo:v1.0
docker run -d -p 5001:5000 yourusername/flask-demo:v1.0
```



Assignment Deliverables

1. Lab Report (Submit as PDF)

Create a document containing:

- Screenshots of each major task completion
- Answers to all questions posed in the tasks
- Docker commands used and their outputs
- Explanation of networking concepts learned

2. Code Repository

Create a Git repository with:

- Flask application code
- Dockerfile
- docker-compose.yml
- README.md with setup instructions

3. Reflection Questions (Answer in your report)

Container vs VM: Explain the key differences between Docker containers and virtual machines.

Ans:

- Containers are faster and lighter but less isolated; VMs are more secure but resource-intensive.
- Containers share the host OS kernel, VMs have their own complete OS
- Containers are more portable across environments, VMs are platform-specific

Networking: Why do containers in custom bridge networks have DNS resolution while default bridge network containers don't?

Ans:

- Custom bridge networks have an embedded DNS server, default bridge doesn't have dns.
- Custom networks automatically register container names as hostnames for DNS lookup
- Custom networks resolve container names to IP addresses automatically

Data Persistence: When would you choose bind mounts over Docker volumes and vice versa?

Ans:

- Bind mounts are used for development and direct file access; Docker volumes used for production and managed persistence.
- Bind mounts need specific hostfile path. Docker volumes means docker will manage the storage life cycle

Image Optimization: What strategies could you use to reduce Docker image size?

Ans:

- Multi-Stage build. That means use separate stages for build and runtime.
- Use distroless image. That means start with minimal base image instead of full OS Image.
- Minimize the docker layers.
- Only copy the required files. Do not copy the entire project

Security: What are three security best practices when building Docker images?

Ans:

- Run as non-root
- Scan for vulnerabilities
- Keep images minimal to reduce security risks.

Production Readiness: What additional considerations would you need for running containers in production?

Ans:

Production requires:

- Container orchestration
- Monitoring & Logging
- Security hardening
- High availability.

Bonus Challenges

Challenge 1: Multi-stage Build

Create a Dockerfile using multi-stage builds to reduce the final image size.

Challenge 2: Health Checks

Add health checks to your containers and demonstrate how they work.

Challenge 3: Container Monitoring

Set up basic monitoring for your containers using docker stats and logging.

Challenge 4: Environment Variables

Modify your Flask app to use environment variables for configuration and demonstrate different ways to pass them to containers.

Evaluation Criteria

Basic Tasks (70 points):

- Container management and interaction
- Networking configuration and testing
- Volume usage and data persistence
- Image building and running

Documentation (20 points):

- Clear explanations of concepts
- Well-documented commands and outputs
- Thoughtful answers to reflection questions

Code Quality (10 points):

- Clean, working application code
- Proper Dockerfile best practices
- Clear repository organization

Total: 100 points

Time Estimate: 3-4 hours

Resources

- Official Docker Documentation
- Docker Hub
- Dockerfile Best Practices

Submission

Submit your lab report PDF and provide the link to your Git repository by the due date specified by your instructor.