Part 1: Docker Basics

```
administrator@Ameet:~$ docker run -d -t --name my-alpine alpine
docker run -d -t --name my-busybox busybox
85eb7824b95fddc4aa6767af91f01dda8f02be5cb999a8d536992e99990d49d8
223a9c74ddf67282854e63ecd01e591addb67bee93f68718ba58ffb1cbb1c4fd
administrator@Ameet:~$ docker ps
docker ps -a
CONTAINER ID   IMAGE     COMMAND     CREATED         STATUS         PORTS     NAMES
223a9c74ddf6   busybox   "sh"        6 seconds ago   Up 5 seconds             my-busybox
85eb7824b95f   alpine    "/bin/sh"   6 seconds ago   Up 6 seconds             my-alpine
CONTAINER ID   IMAGE     COMMAND     CREATED         STATUS         PORTS     NAMES
223a9c74ddf6   busybox   "sh"        7 seconds ago   Up 6 seconds             my-busybox
85eb7824b95f   alpine    "/bin/sh"   7 seconds ago   Up 6 seconds             my-alpine
administrator@Ameet:~$ docker image ls
REPOSITORY              TAG       IMAGE ID       CREATED         SIZE
nginx                   latest    d5f28ef21aab   4 weeks ago     279MB
alpine                  latest    4bcff63911fc   2 months ago    12.8MB
busybox                 latest    ab33eacc8251   11 months ago   6.78MB
yauritux/busybox-curl   latest    dc6fd0cc4de3   3 years ago     13.4MB
administrator@Ameet:~$
```

```
administrator@Ameet:~$ docker exec -t my-alpine ls /
docker exec -t my-busybox ps aux
bin    dev    etc    home   lib    media  mnt    opt    proc   root   run    sbin   srv    sys    tmp    usr    var
PID   USER     TIME  COMMAND
    1 root      0:00 sh
    7 root      0:00 ps aux
administrator@Ameet:~$ docker exec -it my-alpine sh
/ # whoami
root
/ # pwd
/
/ # ls -a
.           .dockerenv  dev         home        media       opt         root        sbin        sys         usr
..          bin         etc         lib         mnt         proc        run         srv         tmp         var
/ # exit
administrator@Ameet:~$ docker stop my-alpine
my-alpine
administrator@Ameet:~$ docker ps -a
CONTAINER ID   IMAGE     COMMAND     CREATED         STATUS                      PORTS     NAMES
223a9c74ddf6   busybox   "sh"        6 minutes ago   Up 6 minutes                          my-busybox
85eb7824b95f   alpine    "/bin/sh"   6 minutes ago   Exited (137) 6 seconds ago            my-alpine
administrator@Ameet:~$ docker start my-alpine
my-alpine
administrator@Ameet:~$ docker ps -a
CONTAINER ID   IMAGE     COMMAND     CREATED         STATUS         PORTS     NAMES
223a9c74ddf6   busybox   "sh"        6 minutes ago   Up 6 minutes             my-busybox
85eb7824b95f   alpine    "/bin/sh"   6 minutes ago   Up 2 seconds             my-alpine
administrator@Ameet:~$ docker rm -f my-busybox
my-busybox
administrator@Ameet:~$ docker ps -a
CONTAINER ID   IMAGE     COMMAND     CREATED         STATUS          PORTS     NAMES
85eb7824b95f   alpine    "/bin/sh"   6 minutes ago   Up 13 seconds             my-alpine
administrator@Ameet:~$
```

Difference between `docker ps` and `docker ps -a`

While working with Docker, I noticed that the output of the command changes depending on whether I use the `-a` flag or not.

- `docker ps` → This shows only the **containers that are currently running**.
- `docker ps -a` → This lists **all containers**, including the ones that are running, stopped, or have already exited.

This helped me differentiate between actively running services and containers that were previously used but have already stopped.

Why Alpine and BusyBox images are so small

During the assignment, I explored the Alpine and BusyBox base images and found that both are extremely lightweight compared to other images. The reasons are:

- **Alpine Linux**
  - Very minimal distribution (around 5 MB in size).
  - Uses `musl libc` and `BusyBox` for essential functionality.
  - Ships only with the bare minimum packages, and any extras must be installed manually.
- **BusyBox**
  - Even smaller (around 1 MB).
  - Combines many common UNIX utilities into a **single binary**.
  - Initially designed for embedded systems, which explains its very small footprint.

In short, both images are small because they strip out everything non-essential, making them fast, efficient, and ideal as base images for Docker containers.

Difference between Alpine and BusyBox

While working with lightweight Docker images, I compared Alpine and BusyBox. Both are minimal, but they serve slightly different purposes.

- **Alpine Linux**
  - A **complete minimal Linux distribution** (around 5 MB).
  - Has its own package manager `apk`, so additional software can be easily installed.
  - Suitable as a base image for running real applications in production (e.g., Python, Node, Java apps).
  - Slightly larger than BusyBox, but still extremely lightweight.
- **BusyBox**
  - Not a full distribution, but a **single binary** that provides many UNIX utilities.
  - Extremely small (around 1 MB).
  - Does **not** come with a package manager. Adding extra tools is difficult.
  - Best suited for very small tasks, testing, or embedded systems.

👉 In summary, **Alpine is a minimal OS you can build upon**, while **BusyBox is a compact toolbox of utilities**, even smaller but less flexible for real application deployment.

Part 2: Docker Networking

```
administrator@Ameet:~$ docker ps -a
CONTAINER ID   IMAGE     COMMAND     CREATED          STATUS         PORTS     NAMES
85eb7824b95f   alpine    "/bin/sh"   10 minutes ago   Up 3 minutes             my-alpine
administrator@Ameet:~$ docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
c0fae1ce536d   bridge    bridge    local
ec3540add52a   host      host      local
0ffc5d598f75   none      null      local
administrator@Ameet:~$ docker run -d --name nginx-default nginx:latest
c252f713c454b2a5fb732c78f1e542aa1ab029b9761fa822f6f59dcf0950b050
administrator@Ameet:~$ docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS         PORTS     NAMES
c252f713c454   nginx:latest   "/docker-entrypoint.…"   3 seconds ago    Up 3 seconds   80/tcp    nginx-default
85eb7824b95f   alpine         "/bin/sh"                10 minutes ago   Up 3 minutes              my-alpine
administrator@Ameet:~$ docker inspect --format='{{json .NetworkSettings.Networks}}' nginx-default
{"bridge":{"IPAMConfig":null,"Links":null,"Aliases":null,"MacAddress":"ce:cc:ef:31:44:ca","DriverOpts":null,"GwPriority":0,"NetworkID":"c0fae1ce536d681d8dd75b3bb00b4b40728515a480a48639
1806f60fc74eac2f","EndpointID":"e483219ad423ec34d0cbb9180ebb062b5995b63f1e6aa6e70e775f29835959b1","Gateway":"172.17.0.1","IPAddress":"172.17.0.3","IPPrefixLen":16,"IPv6Gateway":"","Glo
balIPv6Address":"","GlobalIPv6PrefixLen":0,"DNSNames":null}}
administrator@Ameet:~$ docker exec -it nginx-default curl localhost:80
<!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>
administrator@Ameet:~$
```

```
administrator@Ameet:~$ docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS         PORTS     NAMES
c252f713c454   nginx:latest   "/docker-entrypoint.…"   5 minutes ago    Up 5 minutes   80/tcp    nginx-default
85eb7824b95f   alpine         "/bin/sh"                15 minutes ago   Up 9 minutes             my-alpine
administrator@Ameet:~$ docker rm -f nginx-default
nginx-default
administrator@Ameet:~$ docker ps -a
CONTAINER ID   IMAGE     COMMAND     CREATED          STATUS         PORTS     NAMES
85eb7824b95f   alpine    "/bin/sh"   15 minutes ago   Up 9 minutes             my-alpine
administrator@Ameet:~$ docker run -d -p 8080:80 --name nginx-exposed nginx:latest
79322938ec437942197476206867941d314e2e68968e854d77135a10dc83dbac
administrator@Ameet:~$ 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>
administrator@Ameet:~$
```

```
administrator@Ameet:~$ docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
c0fae1ce536d   bridge    bridge    local
ec3540add52a   host      host      local
0ffc5d598f75   none      null      local
administrator@Ameet:~$ docker network create my-network
d66e8cf51e67ebf269b142a166d1f2aea81dbc9056e9747e55182c9887762f35
administrator@Ameet:~$ docker network ls
NETWORK ID     NAME         DRIVER    SCOPE
c0fae1ce536d   bridge       bridge    local
ec3540add52a   host         host      local
d66e8cf51e67   my-network   bridge    local
0ffc5d598f75   none         null      local
administrator@Ameet:~$ docker run -d --network my-network --name web-server nginx:latest
4bb9881c049697738ca3d8ea54117845f3ad150d285ebd9ef9c5c9a453644cd6
administrator@Ameet:~$ 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=0.153 ms
64 bytes from 172.18.0.2: seq=1 ttl=64 time=0.145 ms
64 bytes from 172.18.0.2: seq=2 ttl=64 time=0.189 ms
^C
--- web-server ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.145/0.162/0.189 ms
/ # 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>
/ #
```

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

While experimenting with Docker networking, I observed that:

- In the **default bridge network**, containers can only communicate using **IP addresses**. Name resolution does not work by default.
- In a **custom bridge network** (created with `docker network create`), Docker automatically sets up an **embedded DNS server**. This allows containers to resolve each other by **container name** or **hostname**.

👉 That is why containers in a custom network can ping each other by name, but in the default bridge, name-based resolution fails.

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

When I tested a web server container inside a custom network:

- By default, the container is **isolated** inside the custom network and not directly accessible from the host machine.
- To make it accessible, I had to **publish the container's port** using the `-p` (or `--publish`) option.
  - Example: `docker run -d -p 8080:80 nginx`
- Once the port was published, I could successfully access the web server from my host machine using `http://localhost:8080`.

👉 In short, containers in a custom network are not reachable from the host unless their ports are explicitly published.

Part 3: Docker Volumes

```
administrator@Ameet:~$ mkdir shared-logs
administrator@Ameet:~$ docker volume ls
DRIVER    VOLUME NAME
administrator@Ameet:~$ docker run -d -v $(pwd)/shared-logs:/app/logs --name logger1 alpine
6783342ff83d9c32868e9b69a5c8f55f0ce611fbb12cfb74beaf0a4e42b64063
administrator@Ameet:~$ docker run -d -v $(pwd)/shared-logs:/app/logs --name logger2 busybox
baf18697614e0a18be785122cae5c7c052e2ec9b9597684d3e2cb6f6ca87e6da
administrator@Ameet:~$ docker exec logger1 sh -c "echo 'Log from container 1' > /app/logs/container1.log"
Error response from daemon: container 6783342ff83d9c32868e9b69a5c8f55f0ce611fbb12cfb74beaf0a4e42b64063 is not running
administrator@Ameet:~$ docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS                      PORTS                                     NAMES
baf18697614e   busybox        "sh"                     39 seconds ago   Exited (0) 38 seconds ago                                             logger2
6783342ff83d   alpine         "/bin/sh"                48 seconds ago   Exited (0) 47 seconds ago                                             logger1
e5e13840f86c   alpine         "sh"                     34 minutes ago   Exited (0) 33 minutes ago                                             client
4bb9881c0496   nginx:latest   "/docker-entrypoint.…"   35 minutes ago   Up 35 minutes               80/tcp                                    web-server
79322938ec43   nginx:latest   "/docker-entrypoint.…"   36 minutes ago   Up 36 minutes               0.0.0.0:8080->80/tcp, [::]:8080->80/tcp   nginx-exposed
85eb7824b95f   alpine         "/bin/sh"                52 minutes ago   Up 45 minutes                                                         my-alpine
administrator@Ameet:~$ docker rm logger1
logger1
administrator@Ameet:~$ docker rm logger2
logger2
administrator@Ameet:~$ docker run -d -v $(pwd)/shared-logs:/app/logs --name logger1 alpine tail -f /dev/null
3a73bdcdac730f4ce950c373b8f0ee90c1a0cea85f789cf958da97acdd3f3d71
administrator@Ameet:~$ docker run -d -v $(pwd)/shared-logs:/app/logs --name logger2 busybox tail -f /dev/null
7d9c6a2c67a18ba3983863b1370a0984123b17a59c9a2a11afda8f3127e09649
administrator@Ameet:~$ docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS                      PORTS                                     NAMES
7d9c6a2c67a1   busybox        "tail -f /dev/null"      6 seconds ago    Up 5 seconds                                                          logger2
3a73bdcdac73   alpine         "tail -f /dev/null"      13 seconds ago   Up 12 seconds                                                         logger1
e5e13840f86c   alpine         "sh"                     35 minutes ago   Exited (0) 34 minutes ago                                             client
4bb9881c0496   nginx:latest   "/docker-entrypoint.…"   35 minutes ago   Up 35 minutes               80/tcp                                    web-server
79322938ec43   nginx:latest   "/docker-entrypoint.…"   37 minutes ago   Up 37 minutes               0.0.0.0:8080->80/tcp, [::]:8080->80/tcp   nginx-exposed
85eb7824b95f   alpine         "/bin/sh"                53 minutes ago   Up 46 minutes                                                         my-alpine
administrator@Ameet:~$ docker exec logger1 sh -c "echo 'Log from container 1' > /app/logs/container1.log"
administrator@Ameet:~$ docker exec logger2 sh -c "echo 'Log from container 2' > /app/logs/container2.log"
administrator@Ameet:~$ ls shared-logs/
container1.log  container2.log
administrator@Ameet:~$ cat shared-logs/*.log
Log from container 1
Log from container 2
administrator@Ameet:~$ docker exec logger1 ls /app/logs/
container1.log
container2.log
administrator@Ameet:~$ docker exec logger2 ls /app/logs/
container1.log
container2.log
administrator@Ameet:~$
```

```
administrator@Ameet:~$ docker volume ls
DRIVER    VOLUME NAME
administrator@Ameet:~$ docker volume create app-data
app-data
administrator@Ameet:~$ docker volume ls
DRIVER    VOLUME NAME
local     app-data
administrator@Ameet:~$ docker volume inspect app-data
[
    {
        "CreatedAt": "2025-09-16T10:53:30Z",
        "Driver": "local",
        "Labels": null,
        "Mountpoint": "/var/lib/docker/volumes/app-data/_data",
        "Name": "app-data",
        "Options": null,
        "Scope": "local"
    }
]
administrator@Ameet:~$ docker run -d --mount source=app-data,target=/data --name data1 alpine tail -f /dev/null
669d63b8d1ed87a36d6f9be99135265fc9c4fc7d582d2b8c6aa41c987b8e7788
administrator@Ameet:~$ docker run -d --mount source=app-data,target=/data --name data2 nginx:latest
54e7fb165a261de4b709675d065b2f5158344a001a76c0bdf7e7568a66be7d25
administrator@Ameet:~$ docker volume inspect app-data
[
    {
        "CreatedAt": "2025-09-16T10:53:30Z",
        "Driver": "local",
        "Labels": null,
        "Mountpoint": "/var/lib/docker/volumes/app-data/_data",
        "Name": "app-data",
        "Options": null,
        "Scope": "local"
    }
]
administrator@Ameet:~$ docker exec data1 sh -c "echo 'Persistent data' > /data/test.txt"
administrator@Ameet:~$ docker exec data2 cat /data/test.txt
Persistent data
administrator@Ameet:~$
```

Difference between Bind Mounts and Docker Volumes

While working with persistent storage in Docker, I explored both **bind mounts** and **volumes**. They look similar at first but are used in different scenarios.

- **Bind Mounts**
  - Maps a **specific directory or file from the host machine** into the container.
  - The container directly reads/writes to the host's filesystem.
  - Very useful during development when I want to **edit code on my machine** and instantly see changes inside the container.
  - Less portable, because it depends on the exact host file paths.
- **Docker Volumes**
  - Managed completely by Docker, stored under Docker's directory (e.g., `/var/lib/docker/volumes/`).
  - Independent of the host's directory structure.
  - Preferred for **production use**, since volumes are easier to back up, migrate, and share between containers.
  - More secure and portable compared to bind mounts.

👉 In short, I use **bind mounts for local development**, and **volumes for production workloads** where portability and data management are important.

Part 4: Building Docker Images

springboot-mysql-docker / dockerfile

akhemani Add files via upload

Code Blame 35 lines (23 loc) · 912 Bytes

```
# ---- Builder ----
FROM eclipse-temurin:17-jdk-jammy AS builder

WORKDIR /workspace

COPY . .

RUN ./mvnw -q -DskipTests package

# ---- Runtime ----
FROM eclipse-temurin:17-jre-jammy

# Non-root user
RUN useradd -u 10001 -r -s /bin/false appuser

WORKDIR /app

# Copy runnable artifact only
COPY --from=builder /workspace/target/*.jar /app/app.jar

# OCI labels (edit values)
LABEL org.opencontainers.image.title="visit-tracker" \
      org.opencontainers.image.description="Spring Boot + MySQL demo (prod-hardened)" \
      org.opencontainers.image.version="1.0.0" \
      org.opencontainers.image.source="https://github.com/akhemani/springboot-mysql-docker/tree/version/v2-production" \
      org.opencontainers.image.licenses="Apache-2.0"

USER appuser

EXPOSE 8081

# Container-aware JVM
ENV JAVA_TOOL_OPTIONS="-XX:MaxRAMPercentage=75.0 -XX:+UseContainerSupport"

ENTRYPOINT ["java","-jar","/app/app.jar"]
```

springboot-mysql-docker / docker-compose.yml

akhemani Add files via upload

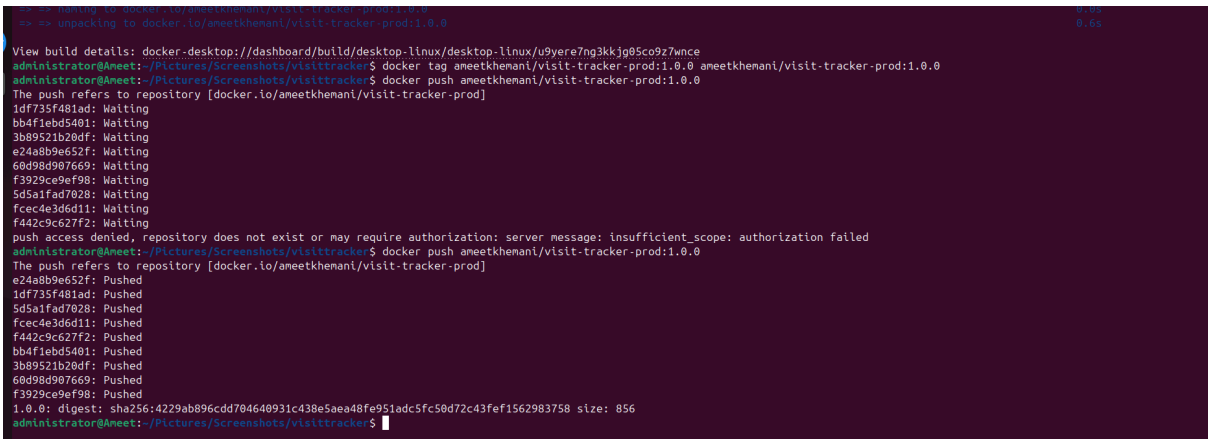
Code Blame 55 lines (51 loc) · 1.26 KB

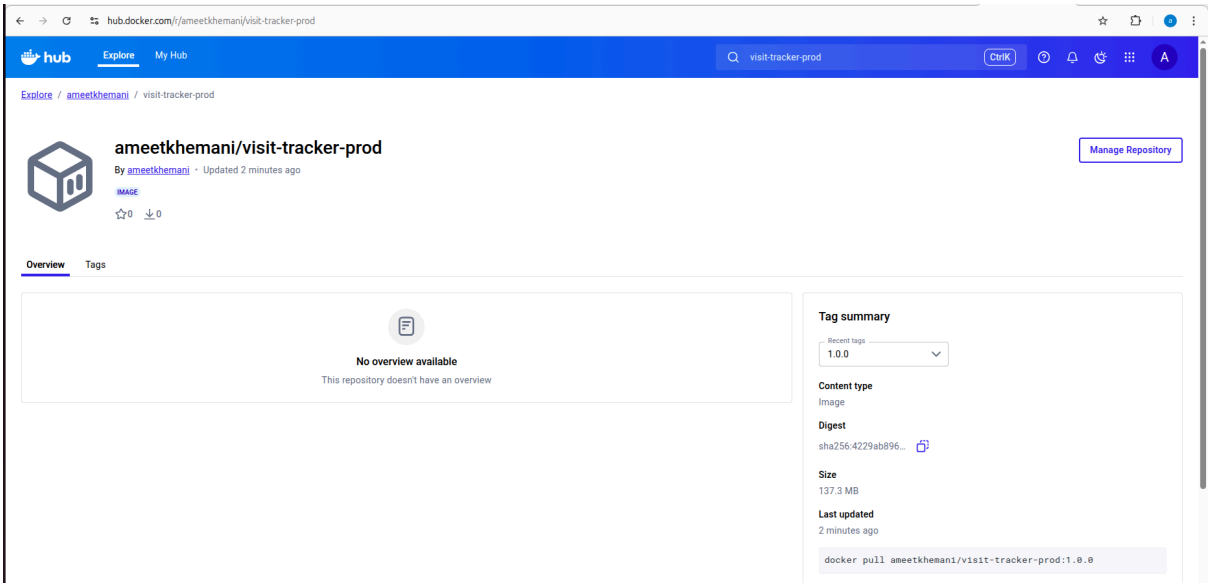
```yaml
version: "3.8"

networks:
  appnet:

volumes:
  mysql_data:

services:
  db:
    image: mysql:8.0.36
    command: >
      --default-authentication-plugin=mysql_native_password
      --character-set-server=utf8mb4
      --collation-server=utf8mb4_unicode_ci
    environment:
      MYSQL_ROOT_PASSWORD: ${DB_ROOT_PASS}
      MYSQL_DATABASE: ${DB_NAME}
      MYSQL_USER: ${DB_USER}
      MYSQL_PASSWORD: ${DB_PASS}
    volumes:
      - mysql_data:/var/lib/mysql
    healthcheck:
      test: ["CMD-SHELL", "mysqladmin ping -h 127.0.0.1 -u${DB_USER} -p${DB_PASS} || exit 1"]
      interval: 5s
      timeout: 3s
      retries: 20
      start_period: 20s
    networks:
      - appnet
    restart: unless-stopped
    # IMPORTANT: no ports published → DB not exposed to host

  app:
    build: .
    image: visit-tracker:2.0.0
    env_file: .env
    depends_on:
      db:
        condition: service_healthy
    ports:
      - "8081:8081"    # expose app only
    healthcheck:
      test: ["CMD-SHELL", "wget -qO- http://127.0.0.1:8081/actuator/health/readiness | grep -q '\"status\":\"UP\"'"]
      interval: 5s
      timeout: 3s
      retries: 12
      start_period: 20s
    read_only: true
    tmpfs:
      - /tmp
    user: "10001:10001"
    networks:
      - appnet
    restart: unless-stopped
```

Part 5: Image Registry

```
administrator@Ameet: ~/Pictures/Screenshots/test
administrator@Ameet:~/Pictures/Screenshots/test$ docker pull ameetkhemani/visit-tracker-prod:1.0.0
1.0.0: Pulling from ameetkhemani/visit-tracker-prod
3b89521b20df: Pull complete
fcec4e3d6d11: Pull complete
f442c9c627f2: Pull complete
Digest: sha256:4229ab896cdd704640931c438e5aea48fe951adc5fc50d72c43fef1562983758
Status: Downloaded newer image for ameetkhemani/visit-tracker-prod:1.0.0
docker.io/ameetkhemani/visit-tracker-prod:1.0.0
administrator@Ameet:~/Pictures/Screenshots/test$ dockerps
dockerps: command not found
administrator@Ameet:~/Pictures/Screenshots/test$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
administrator@Ameet:~/Pictures/Screenshots/test$ docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
administrator@Ameet:~/Pictures/Screenshots/test$ docker image ls
REPOSITORY                        TAG       IMAGE ID       CREATED          SIZE
ameetkhemani/visit-tracker-prod   1.0.0     4229ab896cdd   56 minutes ago   484MB
administrator@Ameet:~/Pictures/Screenshots/test$ docker run --name prod-tracker ameetkhemani/visit-tracker-prod:1.0.0
Picked up JAVA_TOOL_OPTIONS: -XX:MaxRAMPercentage=75.0 -XX:+UseContainerSupport

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/

 :: Spring Boot ::                (v3.5.5)

2025-09-16T15:52:32.516Z  INFO 1 --- [           main] c.visittracker.VisitTrackerApplication   : Starting VisitTrackerApplication v0.0.1-SNAPSHOT using Java 17.0.16 with PID 1 (/app/a
pp.jar started by appuser in /app)
2025-09-16T15:52:32.519Z  INFO 1 --- [           main] c.visittracker.VisitTrackerApplication   : No active profile set, falling back to 1 default profile: "default"
2025-09-16T15:52:33.910Z  INFO 1 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2025-09-16T15:52:33.972Z  INFO 1 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 51 ms. Found 1 JPA repository interface.
2025-09-16T15:52:34.718Z  INFO 1 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port 8081 (http)
2025-09-16T15:52:34.738Z  INFO 1 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2025-09-16T15:52:34.739Z  INFO 1 --- [           main] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache Tomcat/10.1.44]
2025-09-16T15:52:34.783Z  INFO 1 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2025-09-16T15:52:34.787Z  INFO 1 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 2183 ms
2025-09-16T15:52:35.300Z  INFO 1 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Starting...
2025-09-16T15:52:41.967Z  WARN 1 --- [           main] ConfigServletWebServerApplicationContext : Exception encountered during context initialization - cancelling refresh attempt: org.
springframework.beans.factory.BeanCreationException: Error creating bean with name 'entityManagerFactory' defined in class path resource [org/springframework/boot/autoconfigure/orm/jpa
/HibernateJpaConfiguration.class]: Failed to initialize dependency 'flywayInitializer' of LoadTimeWeaverAware bean 'entityManagerFactory': Error creating bean with name 'flywayInitiali
zer' defined in class path resource [org/springframework/boot/autoconfigure/flyway/FlywayAutoConfiguration$FlywayConfiguration.class]: Unable to obtain connection from database: Commun
ications link failure

The last packet sent successfully to the server was 0 milliseconds ago. The driver has not received any packets from the server.
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
```

```
administrator@Ameet:~/Pictures/Screenshots/visittracker$ docker login

USING WEB-BASED LOGIN

i Info → To sign in with credentials on the command line, use 'docker login -u <username>'


Your one-time device confirmation code is: QSWT-SXSN
Press ENTER to open your browser or submit your device code here: https://login.docker.com/activate

Waiting for authentication in the browser…

2025/09/16 21:09:02 notifying Desktop of credentials store update: Post "http://ipc/registry/credstore-updated": context deadline exceeded
Login Succeeded
administrator@Ameet:~/Pictures/Screenshots/visittracker$ docker build -t ameetkhemani/visit-tracker-prod:1.0.0 .
[+] Building 3.5s (14/14) FINISHED                                                                                    docker:desktop-linux
 => [internal] load build definition from dockerfile                                                                                   0.0s
 => => transferring dockerfile: 951B                                                                                                   0.0s
 => [internal] load metadata for docker.io/library/eclipse-temurin:17-jre-jammy                                                       2.3s
 => [internal] load metadata for docker.io/library/eclipse-temurin:17-jdk-jammy                                                       2.3s
 => [internal] load .dockerignore                                                                                                      0.0s
 => => transferring context: 2B                                                                                                        0.0s
 => [builder 1/4] FROM docker.io/library/eclipse-temurin:17-jdk-jammy@sha256:ac8a552a417551699694a711119e1c410a4b65ed79cb98b16584bc91c0d491b6  0.1s
 => => resolve docker.io/library/eclipse-temurin:17-jdk-jammy@sha256:ac8a552a417551699694a711119e1c410a4b65ed79cb98b16584bc91c0d491b6  0.1s
 => [internal] load build context                                                                                                      0.1s
 => => transferring context: 4.74kB                                                                                                    0.0s
 => [stage-1 1/4] FROM docker.io/library/eclipse-temurin:17-jre-jammy@sha256:93db0dba1dacb225fed826c2ae720adae9a71205f2b199b8af7e4df5be010cb7  0.1s
 => => resolve docker.io/library/eclipse-temurin:17-jre-jammy@sha256:93db0dba1dacb225fed826c2ae720adae9a71205f2b199b8af7e4df5be010cb7  0.1s
 => CACHED [stage-1 2/4] RUN useradd -u 10001 -r -s /bin/false appuser                                                                 0.0s
 => CACHED [stage-1 3/4] WORKDIR /app                                                                                                  0.0s
 => CACHED [builder 2/4] WORKDIR /workspace                                                                                            0.0s
 => CACHED [builder 3/4] COPY . .                                                                                                      0.0s
 => CACHED [builder 4/4] RUN ./mvnw -q -DskipTests package                                                                             0.0s
 => CACHED [stage-1 4/4] COPY --from=builder /workspace/target/*.jar /app/app.jar                                                      0.0s
 => exporting to image                                                                                                                 0.8s
 => => exporting layers                                                                                                                0.0s
 => => exporting manifest sha256:98eb4eef75de9ba8f47aaf7c12dc1f98dcaae7144bd03c4cbb60ef2e408088b9                                      0.0s
 => => exporting config sha256:72a5b00461589a3902eb0a0e7859fb2bc5452e1d85a31011339a394102428c15                                        0.0s
 => => exporting attestation manifest sha256:c9e218c3e216e300951b12dd2f151bd77bfacee60b28e2e80cb0cbe91a905e75                          0.1s
 => => exporting manifest list sha256:4229ab896cdd704640931c438e5aea48fe951adc5fc50d72c43fef1562983758                                 0.0s
 => => naming to docker.io/ameetkhemani/visit-tracker-prod:1.0.0                                                                       0.0s
 => => unpacking to docker.io/ameetkhemani/visit-tracker-prod:1.0.0                                                                    0.6s
```





https://github.com/akhemani/docker-fundamentals-practice

Container vs VM: Key Differences

While studying containerization, I compared Docker containers with traditional virtual machines (VMs).

- **Containers**
    - Share the host operating system kernel.
    - Lightweight, fast to start, and use fewer resources.
    - Ideal for microservices and deploying many small workloads.

- **Virtual Machines (VMs)**
    - Each VM includes a full guest operating system with its own kernel.
    - Heavier and slower to boot, with higher resource usage.
    - Useful when strong isolation is required or when running different OS types.

👉 In short, containers are more efficient, while VMs provide stronger isolation and flexibility across OS types.

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

During my networking experiments, I observed:

- **Default bridge network**
    - Does not have an integrated DNS for name resolution.
    - Containers can only reach each other via IP addresses.
- **Custom bridge network**
    - Docker provides an **embedded DNS server**.
    - Containers can communicate using **names/hostnames** instead of IPs.

👉 That's why DNS resolution works automatically in custom bridge networks but not in the default bridge.

Data Persistence: Choosing Bind Mounts vs Docker Volumes

I compared two methods of persisting data in Docker:

- **Bind Mounts**
  - Link a specific host directory/file to a container.
  - Best for **development** when I want changes on the host to instantly reflect inside the container.
- **Docker Volumes**
  - Managed by Docker and stored under Docker's data directory.
  - More secure, portable, and easier to back up.
  - Best for **production** environments.

👉 I use bind mounts for development convenience and volumes for production reliability.

Image Optimization: Strategies to Reduce Docker Image Size

While working with images, I noted several optimization techniques:

- Use **minimal base images** like Alpine.
- Remove unnecessary packages and dependencies.
- Combine multiple `RUN` commands into fewer layers.
- Use `.dockerignore` to avoid copying unnecessary files.
- Multi-stage builds: build in one stage, copy only required artifacts to the final stage.

👉 These strategies keep images small, faster to pull, and more secure.

Security: Best Practices for Building Docker Images

To improve image security, I followed these practices:

1. **Use official or trusted base images** instead of unknown sources.
2. **Run as a non-root user** inside containers to minimize risk.
3. **Keep images updated** by applying security patches and rebuilding regularly.

👉 These practices help reduce vulnerabilities and improve container security.

Production Readiness: Considerations for Running Containers

When preparing containers for production, I identified the following key considerations:

- **Logging & Monitoring**: Centralized logs and metrics (e.g., Prometheus, ELK, Grafana).
- **Scaling & Orchestration**: Use orchestration tools like Kubernetes or Docker Swarm.
- **Networking & Security**: Secure networks, TLS, secrets management.
- **Resource Limits**: Define CPU and memory limits for containers.
- **High Availability**: Run multiple replicas and use load balancing.

👉 These considerations ensure reliability, scalability, and security for production workloads.