DOCKERFILE'S BASED ON DIFFERENT SCENARIOS

Scenario 1: Multi-Stage Build for Optimized Image Size

Link - [scenario_based_learnings/Dockerfiles/Dockerfile1.txt at main · praveen1994dec/scenario_based_learnings (github.com)]

Purpose: To reduce the final image size by separating the build and runtime stages.

The purpose of using multi-stage builds is to separate build dependencies from the runtime environment. This approach helps in reducing the final image size by eliminating unnecessary build artifacts and dependencies, thus improving security and efficiency.

Benefits:

- **Efficiency**: Separates build dependencies from the runtime environment, reducing the final Docker image size.
- **Security**: Eliminates unnecessary build artifacts, minimizing potential attack surfaces.
- **Performance**: Optimizes Docker image build times by focusing on required dependencies only

Scenario 2: Using Non-Root User

Link - scenario_based_learnings/Dockerfiles/Dockerfile2.txt at main · praveen1994dec/scenario_based_learnings (github.com)

Purpose: Enhance security by running the application as a non-root user.

The purpose of using a non-root user (`javauser`) is to enhance security within the Docker container. Running applications with non-root privileges minimizes the potential impact of security vulnerabilities by limiting the scope of access an attacker could gain if they exploit the application.

```
# Stage 1: Build environment
FROM maven:3.8.4-openjdk-11-slim AS build

WORKDIR /app

# Copy only the pom.xml to cache dependencies
COPY pom.xml .

# Fetch dependencies
RUN mvn dependency:go-offline

# Copy the source code and build the application
COPY src ./src
RUN mvn clean package -DskipTests

# Stage 2: Runtime environment
FROM tomcat:9.0.54-jdk11-openjdk-slim

# Create a group and user
RUN groupadd -r mygroup && useradd -r -g mygroup myuser

# Set the working directory and change the ownership to the new user
WORKDIR /usr/local/tomcat
RUN chown -R myuser:mygroup /usr/local/tomcat

# Switch to the non-root user
USER myuser

# Copy the built WAR file from the build stage to the webapps directory of Tomcat
COPY --from=build /app/target/*.war /usr/local/tomcat/webapps/

# Expose the default port for Tomcat
EXPOSE 8080

# Start Tomcat server
CMD ["catalina.sh", "run"]
```

```
ec2-user@ip-172-31-40-46:~/Docker/java-hello-world-webapp
 => [internal] load metadata for docker.io/library/maven:3.8.4-openjdk-11-slim                                                0.7s
 => [internal] load .dockerignore                                                                                             0.0s
 => => transferring context: 2B                                                                                               0.0s
 => [build 1/6] FROM docker.io/library/maven:3.8.4-openjdk-11-slim@sha256:04f8e5ba4a6a74fb7f97940bc75ac7340520728d2fb051ecc5c9e  0.0s
 => CACHED [stage-1 1/5] FROM docker.io/library/tomcat:9.0.54-jdk11-openjdk-slim@sha256:2a9ee77b2aef3761804ab300e9d3c54f451ae45  0.0s
 => [internal] load build context                                                                                             0.0s
 => => transferring context: 666B                                                                                             0.0s
 => [stage-1 2/5] RUN groupadd -r mygroup && useradd -r -g mygroup myuser                                                     0.3s
 => CACHED [build 2/6] WORKDIR /app                                                                                           0.0s
 => CACHED [build 3/6] COPY pom.xml .                                                                                         0.0s
 => CACHED [build 4/6] RUN mvn dependency:go-offline                                                                          0.0s
 => CACHED [build 5/6] COPY src ./src                                                                                         0.0s
 => CACHED [build 6/6] RUN mvn clean package -DskipTests                                                                      0.0s
 => [stage-1 3/5] WORKDIR /usr/local/tomcat                                                                                   0.1s
 => [stage-1 4/5] RUN chown -R myuser:mygroup /usr/local/tomcat                                                               0.9s
 => [stage-1 5/5] COPY --from=build /app/target/*.war /usr/local/tomcat/webapps/                                              0.0s
 => exporting to image                                                                                                        0.5s
 => => exporting layers                                                                                                       0.4s
 => => writing image sha256:50479fa5b3e6c93170f508267b29ff5a65bd436ebfcf5ba47b7dc9a986d3714b                                  0.0s
 => => naming to docker.io/library/spring2                                                                                    0.0s
[ec2-user@ip-172-31-40-46 java-hello-world-webapp]$ docker ps
CONTAINER ID   IMAGE     COMMAND             CREATED         STATUS         PORTS                                     NAMES
de8d71b936f5   spring1   "catalina.sh run"   3 minutes ago   Up 3 minutes   0.0.0.0:80->8080/tcp, :::80->8080/tcp     spring1
[ec2-user@ip-172-31-40-46 java-hello-world-webapp]$ docker rm -f de8d71b936f5
de8d71b936f5
[ec2-user@ip-172-31-40-46 java-hello-world-webapp]$ docker run -dit --name spring2 -p 80:8080 spring2
51e3edd2bc3e9e319fee86e9271a9c08d6161a21fc27a463bd1f73d2d862a52f
[ec2-user@ip-172-31-40-46 java-hello-world-webapp]$ docker ps
CONTAINER ID   IMAGE     COMMAND             CREATED         STATUS         PORTS                                     NAMES
51e3edd2bc3e   spring2   "catalina.sh run"   4 seconds ago   Up 3 seconds   0.0.0.0:80->8080/tcp, :::80->8080/tcp     spring2
[ec2-user@ip-172-31-40-46 java-hello-world-webapp]$ curl http://3.110.162.128/java-hello-world/
<html>
<body>
<h2>Hello World 7</h2>
</body>
</html>
[ec2-user@ip-172-31-40-46 java-hello-world-webapp]$ docker images
REPOSITORY   TAG       IMAGE ID       CREATED              SIZE
spring2      latest    50479fa5b3e6   About a minute ago   468MB
spring1      latest    a69070a00a09   12 minutes ago       449MB
[ec2-user@ip-172-31-40-46 java-hello-world-webapp]$
```

Benefits:

- **Security**: Reduces the impact of security vulnerabilities by limiting container privileges.

Scenario 3: Health Checks

Link- [scenario_based_learnings/Dockerfiles/Dockerfile3.txt at main · praveen1994dec/scenario_based_learnings (github.com)](https://github.com)

Purpose: Ensure the application is running correctly by adding health checks.

The purpose of implementing health checks is to monitor the application's health status within the Docker container. This allows Docker to automatically restart containers that fail health checks, ensuring continuous availability of the application.

```
[ec2-user@ip-172-31-40-46 java-hello-world-webapp]$ docker build -t spring3 .
[+] Building 1.4s (7/7) FINISHED                                                              docker:default
 => [internal] load build definition from Dockerfile                                                     0.0s
 => => transferring dockerfile: 543B                                                                     0.0s
 => [internal] load metadata for docker.io/library/tomcat:9.0.54-jdk11-openjdk-slim                      1.2s
 => [internal] load .dockerignore                                                                        0.0s
 => => transferring context: 2B                                                                          0.0s
 => [internal] load build context                                                                        0.0s
 => => transferring context: 2.03kB                                                                      0.0s
 => [1/2] FROM docker.io/library/tomcat:9.0.54-jdk11-openjdk-slim@sha256:2a9ee77b2aef3761804ab300e9d3c54f451ae45b6f8b7b758b800c  0.0s
 => CACHED [2/2] COPY target/java-hello-world.war /usr/local/tomcat/webapps/                             0.0s
 => exporting to image                                                                                   0.0s
 => => exporting layers                                                                                  0.0s
 => => writing image sha256:c4e22e65d8073f011ff1b5f7d836091aaec39c2c66383b7807f72d724f48cd86             0.0s
 => => naming to docker.io/library/spring3                                                               0.0s
[ec2-user@ip-172-31-40-46 java-hello-world-webapp]$ docker ps
CONTAINER ID   IMAGE     COMMAND             CREATED         STATUS         PORTS                                   NAMES
51e3edd2bc3e   spring2   "catalina.sh run"   4 minutes ago   Up 4 minutes   0.0.0.0:80->8080/tcp, :::80->8080/tcp   spring2
[ec2-user@ip-172-31-40-46 java-hello-world-webapp]$ docker rm -f 51e3edd2bc3e
51e3edd2bc3e
[ec2-user@ip-172-31-40-46 java-hello-world-webapp]$ docker run -dit --name spring3 -p 80:8080 spring3
afb52f117cc51f1069d913ea2ff219619847c5eae34fdde1453d027fb90244e0
[ec2-user@ip-172-31-40-46 java-hello-world-webapp]$ curl http://3.110.162.128/java-hello-world/
<html>
<body>
<h2>Hello World ?</h2>
</body>
</html>
[ec2-user@ip-172-31-40-46 java-hello-world-webapp]$
```

ec2-user@ip-172-31-40-46:~/Docker/java-hello-world-webapp

```
# Use the official Tomcat image from the Docker Hub
FROM tomcat:9.0.54-jdk11-openjdk-slim

# Copy the built WAR file to the webapps directory of Tomcat
COPY target/java-hello-world.war /usr/local/tomcat/webapps/

# Expose the default port for Tomcat
EXPOSE 8080

# Health check configuration
HEALTHCHECK --interval=30s --timeout=10s --retries=3 \
  CMD curl -f http://localhost:8080/ || exit 1

# Start Tomcat server
CMD ["catalina.sh", "run"]
```

Benefits:

- **Reliability**: Ensures continuous monitoring of application health, automatically restarting containers that fail health checks.
- **Availability**: Improves service availability by quickly detecting and responding to application failures.

Scenario 4: Minimal Base Image

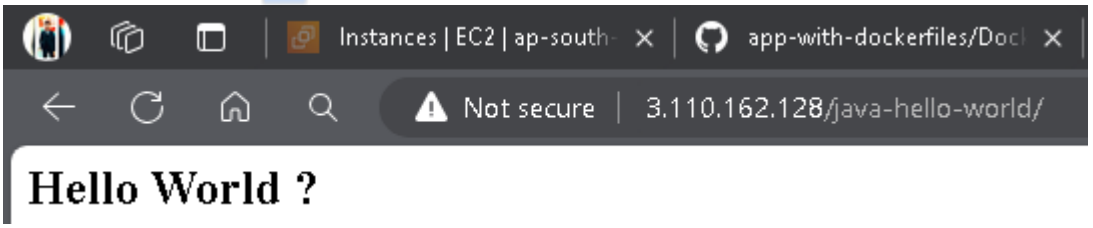
Link - [scenario_based_learnings/Dockerfiles/Dockerfile4.txt at main · praveen1994dec/scenario_based_learnings (github.com)]

Purpose: Use a minimal base image to reduce attack surface and image size.

The purpose of using a minimal base image (`openjdk:11-jre-slim`) is to reduce the Docker image size. Slim images exclude unnecessary packages and utilities, reducing the attack surface and optimizing runtime performance of the application.

```
[ec2-user@ip-172-31-40-46 java-hello-world-webapp]$ docker build -t spring4 .
[+] Building 0.8s (15/15) FINISHED                                                                                  docker:default
 => [internal] load build definition from Dockerfile                                                                         0.0s
 => => transferring dockerfile: 618B                                                                                         0.0s
 => [internal] load metadata for docker.io/library/adoptopenjdk:11-jre-hotspot-focal                                         0.7s
 => [internal] load metadata for docker.io/library/maven:3.8.4-openjdk-11-slim                                               0.6s
 => [internal] load .dockerignore                                                                                            0.0s
 => => transferring context: 2B                                                                                              0.0s
 => [build 1/6] FROM docker.io/library/maven:3.8.4-openjdk-11-slim@sha256:04f8e5ba4a6a74fb7f97940bc75ac7340520728d2fb051ecc5c9e  0.0s
 => [runtime 1/3] FROM docker.io/library/adoptopenjdk:11-jre-hotspot-focal@sha256:18e723434158eae3aca90870ea3df82605b1e61afaa77  0.0s
 => [internal] load build context                                                                                            0.0s
 => => transferring context: 666B                                                                                            0.0s
 => CACHED [runtime 2/3] WORKDIR /app                                                                                        0.0s
 => CACHED [build 2/6] WORKDIR /app                                                                                          0.0s
 => CACHED [build 3/6] COPY pom.xml .                                                                                        0.0s
 => CACHED [build 4/6] RUN mvn dependency:go-offline                                                                         0.0s
 => CACHED [build 5/6] COPY src ./src                                                                                        0.0s
 => CACHED [build 6/6] RUN mvn clean package -DskipTests                                                                     0.0s
 => CACHED [runtime 3/3] COPY --from=build /app/target/java-hello-world.war ./app.war                                        0.0s
 => exporting to image                                                                                                       0.0s
 => => exporting layers                                                                                                      0.0s
 => => writing image sha256:9507ee0e9594ee6d4eeb7d5a173f9560c7182693a0c47c5240a919649c3950ab                                 0.0s
 => => naming to docker.io/library/spring4                                                                                   0.0s
[ec2-user@ip-172-31-40-46 java-hello-world-webapp]$ docker run -dit --name spring4 -p 80:8080 spring4
5d33c9b1c3eb474a78aaac5c6021b0263fca02d7e907a6bd675b02151360037c
```

```
[ec2-user@ip-172-31-40-46 java-hello-world-webapp]$ docker images
REPOSITORY   TAG       IMAGE ID       CREATED          SIZE
spring4      latest    9507ee0e9594   12 minutes ago   244MB
spring2      latest    50479fa5b3e6   19 minutes ago   468MB
spring1      latest    a69070a00a09   31 minutes ago   449MB
spring3      latest    c4e22e65d807   46 minutes ago   449MB
```



Benefits:

- **Reduced Attack Surface**: Minimizes the number of installed packages and dependencies, reducing potential vulnerabilities.
- **Improved Performance**: Enhances Docker container startup time and runtime performance due to fewer resources being utilized

Scenario 5: Environment Variables and Secrets

Link - [scenario_based_learnings/Dockerfiles/Dockerfile5.txt at main · praveen1994dec/scenario_based_learnings (github.com)]

Purpose: Use environment variables to manage configuration and secrets.

The purpose of setting environment variables like (`DATABASE_URL`, `DATABASE_USERNAME`, `DATABASE_PASSWORD`) and using Docker secrets is to securely configure sensitive information required by the application at runtime. This approach ensures that sensitive data is not exposed in Dockerfiles or version-controlled files.

```
[ec2-user@ip-172-31-40-46 java-hello-world-webapp]$ ls
application.properties  config  Dockerfile  init.sh  pom.xml  README.md  run  src  target
[ec2-user@ip-172-31-40-46 java-hello-world-webapp]$ docker images
REPOSITORY   TAG       IMAGE ID       CREATED              SIZE
spring5      latest    cc9d81bbdd81   About a minute ago   444MB
spring4      latest    9507ee0e9594   24 minutes ago       244MB
spring2      latest    50479fa5b3e6   32 minutes ago       468MB
spring1      latest    a69070a00a09   44 minutes ago       449MB
spring3      latest    c4e22e65d807   59 minutes ago       449MB
[ec2-user@ip-172-31-40-46 java-hello-world-webapp]$ docker ps
CONTAINER ID   IMAGE     COMMAND             CREATED              STATUS              PORTS                                   NAMES
b5be0c246c92   spring5   "catalina.sh run"   About a minute ago   Up About a minute   0.0.0.0:80->8080/tcp, :::80->8080/tcp   spring5
```

```
ec2-user@ip-172-31-40-46:~/Docker/java-hello-world-webapp
b5be0c246c92   spring5   "catalina.sh run"   About a minute ago   Up About a minute   0.0.0.0:80->8080/tcp, :::80->8080/tcp   spring5
[ec2-user@ip-172-31-40-46 java-hello-world-webapp]$ cat Dockerfile
# Stage 1: Build environment
FROM maven:3.8.4-openjdk-11-slim AS build

WORKDIR /app

# Copy only the pom.xml to cache dependencies
COPY pom.xml .

# Fetch dependencies
RUN mvn dependency:go-offline

# Copy the source code and build application
COPY src ./src
RUN mvn package -DskipTests

# Stage 2: Runtime environment with Tomcat
FROM tomcat:9.0-jdk11-openjdk-slim

# Remove the default webapps provided by Tomcat
RUN rm -rf /usr/local/tomcat/webapps/*

# Set environment variables
ENV DATABASE_URL="jdbc:mysql://localhost:3306/mydb"
ENV DATABASE_USERNAME="user"
ENV DATABASE_PASSWORD="password"

# Create a non-root user
RUN adduser --system --group javauser
USER javauser

# Copy the built WAR file from the build stage into Tomcat's webapps directory
COPY --from=build /app/target/java-hello-world.war /usr/local/tomcat/webapps/ROOT.war

# Expose Tomcat port
EXPOSE 8080

# Command to run Tomcat
CMD ["catalina.sh", "run"]
```

```
[ec2-user@ip-172-31-40-46 java-hello-world-webapp]$ docker exec -it b5be0c246c92 /bin/bash
javauser@b5be0c246c92:/usr/local/tomcat$ echo $DATABASE_USERNAME
user
javauser@b5be0c246c92:/usr/local/tomcat$ echo $DATABASE_PASSWORD
password
```

Benefits:

- **Security**: Safely manages sensitive information such as database credentials and API keys without exposing them in Dockerfiles or version-controlled files.
- **Flexibility**: Allows dynamic configuration of application environments during runtime, supporting different deployment environments (development, testing, production)

Scenario 6: Using External Configuration Files

Link -[scenario_based_learnings/Dockerfiles/Dockerfile6.txt at main · praveen1994dec/scenario_based_learnings (github.com)]

Purpose: Separate configuration from the application code.

The purpose of copying an external configuration file (`application.properties`) into the Docker image is to provide flexibility in configuring the application without modifying the Dockerfile. External configuration files can be mounted as volumes during container runtime, allowing dynamic configuration updates.

```
# Stage 1: Build environment with Maven
FROM maven:3.8.4-openjdk-11-slim AS build

WORKDIR /app

# Copy only the pom.xml to cache dependencies
COPY pom.xml .

# Fetch dependencies
RUN mvn dependency:go-offline

# Copy the source code
COPY src ./src

# Build the application
RUN mvn package -DskipTests

# Stage 2: Runtime environment with Tomcat
FROM tomcat:9.0.59-jdk11-openjdk-slim

# Create necessary directories
RUN mkdir -p /usr/local/tomcat/conf/myapp

# Copy external configuration files
COPY config/application.properties /app/application.properties

# Copy the built WAR file into Tomcat webapps directory
COPY --from=build /app/target/java-hello-world.war /usr/local/tomcat/webapps/

# Expose default Tomcat port
EXPOSE 8080

# Start Tomcat
CMD ["catalina.sh", "run"]
```

```
[ec2-user@ip-172-31-40-46 java-hello-world-webapp]$ docker run -dit --name spring6 -p 80:8080 spring6
cfdd351918bbb189a0c64cada71ad8dfa0c8a65d9e0d1b381677316c5831b626
[ec2-user@ip-172-31-40-46 java-hello-world-webapp]$ docker exec -it spring6 /bin/bash
root@cfdd351918bb:/usr/local/tomcat# ls
BUILDING.txt     LICENSE  README.md      RUNNING.txt  conf  logs            temp     webapps.dist
CONTRIBUTING.md  NOTICE   RELEASE-NOTES  bin          lib   native-jni-lib  webapps  work
root@cfdd351918bb:/usr/local/tomcat# app
bash: app: command not found
root@cfdd351918bb:/usr/local/tomcat# cd ..
root@cfdd351918bb:/usr/local# ls
bin  etc  games  include  lib  man  openjdk-11  sbin  share  src  tomcat
root@cfdd351918bb:/usr/local# cd ..
root@cfdd351918bb:/usr# ls
bin  games  include  lib  libexec  local  sbin  share  src
root@cfdd351918bb:/usr# cd ..
root@cfdd351918bb:/# cd app/
root@cfdd351918bb:/app# ls
application.properties
root@cfdd351918bb:/app#
```

Benefits:

- **Configuration Management**: Separates configuration settings from the Docker image, facilitating easier configuration updates without rebuilding images.

Scenario 7: Customizing Base Image

Link -[scenario_based_learnings/Dockerfiles/Dockerfile7.txt at main · praveen1994dec/scenario_based_learnings (github.com)](https://github.com/praveen1994dec/scenario_based_learnings/blob/main/Dockerfiles/Dockerfile7.txt)

Purpose: Use a specific base image that includes only necessary dependencies.

The purpose of customizing the base image by installing additional packages (`curl`, `wget`) is to meet specific application dependencies or operational requirements. This approach enhances the functionality of the Docker container by including necessary tools or utilities.

```
root@c53e819d3e1d:/usr/local/tomcat/conf# printenv
HOSTNAME=c53e819d3e1d
JAVA_HOME=/usr/local/openjdk-11
GPG_KEYS=48F8E69F6390C9F25CFEDCD268248959359E722B A9C5DF4D22E99998D9875A5110C01C5A2F6059E7 DCFD35E0BF8CA7344752DE8B6FB21E8933C60243
PWD=/usr/local/tomcat/conf
TOMCAT_SHA512=74902b522abda04afb2be24d7410d4d93966d20fd07dde8f03bb281cdc714866f648babe1ff1ae85d663774779235f1cb9d701d5ce8884052f1f5efc
a7b62c68
TOMCAT_MAJOR=9
HOME=/root
LANG=C.UTF-8
MY_CUSTOM_VAR=custom_value
TOMCAT_NATIVE_LIBDIR=/usr/local/tomcat/native-jni-lib
TERM=xterm
CATALINA_HOME=/usr/local/tomcat
SHLVL=1
LD_LIBRARY_PATH=/usr/local/tomcat/native-jni-lib
PATH=/usr/local/tomcat/bin:/usr/local/openjdk-11/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
TOMCAT_VERSION=9.0.59
JAVA_VERSION=11.0.14.1
_=/usr/bin/printenv
OLDPWD=/usr/local/tomcat
root@c53e819d3e1d:/usr/local/tomcat/conf# echo $MY_CUSTOM_VAR
custom_value
root@c53e819d3e1d:/usr/local/tomcat/conf# ls
Catalina                catalina.policy     context.xml          jaspic-providers.xsd  server.xml        tomcat-users.xsd
application.properties  catalina.properties  jaspic-providers.xml  logging.properties    tomcat-users.xml  web.xml
root@c53e819d3e1d:/usr/local/tomcat/conf# cat application.properties
"key" = "value"

root@c53e819d3e1d:/usr/local/tomcat/conf#
```

```
# Use Tomcat base image with JDK 11
FROM tomcat:9.0.59-jdk11-openjdk-slim

LABEL maintainer="yourname@example.com"

# Set environment variables if needed
ENV MY_CUSTOM_VAR="custom_value"

# Copy your application WAR file into Tomcat's webapps directory
COPY target/java-hello-world.war /usr/local/tomcat/webapps/

# Copy external configuration files if needed
COPY config/application.properties /usr/local/tomcat/conf/

# Optionally, install additional packages or perform other customizations
# For example, installing curl
RUN apt-get update && apt-get install -y \
    curl \
    && rm -rf /var/lib/apt/lists/*

# Expose default Tomcat port
EXPOSE 8080

# Start Tomcat
CMD ["catalina.sh", "run"]
```

Benefits:

- **Application Dependencies**: Installs additional tools or libraries required by the application, ensuring compatibility and functionality.

Scenario 8: Including External Dependencies

Link - [scenario_based_learnings/Dockerfiles/Dockerfile8.txt at main · praveen1994dec/scenario_based_learnings (github.com)]

Purpose: Install additional system dependencies required by your application.

The purpose of installing external dependencies (`curl`) during the Docker image build process is to ensure that the Docker container has all necessary tools or libraries required by the application. This approach facilitates seamless integration with external services or APIs.

```
[ec2-user@ip-172-31-40-46 java-hello-world-webapp]$ docker images
REPOSITORY   TAG       IMAGE ID       CREATED              SIZE
spring8      latest    f55fb9c2b9bd   4 seconds ago        294MB
spring7      latest    bb0d0889d86e   9 minutes ago        454MB
spring6      latest    e1c9cc3de997   15 minutes ago       450MB
spring5      latest    cc9d81bbdd81   28 minutes ago       444MB
spring4      latest    9507ee0e9594   50 minutes ago       244MB
spring2      latest    50479fa5b3e6   58 minutes ago       468MB
spring1      latest    a69070a00a09   About an hour ago    449MB
spring3      latest    c4e22e65d807   About an hour ago    449MB
[ec2-user@ip-172-31-40-46 java-hello-world-webapp]$ docker run -dit --name spring8 -p 80:8080 spring8
29b9709baf05a0a95c422f55e3dafb6364323e3075d198b9c9e85d9866907f7f
[ec2-user@ip-172-31-40-46 java-hello-world-webapp]$
```

```
[ec2-user@ip-172-31-40-46 java-hello-world-webapp]$ docker ps
CONTAINER ID   IMAGE     COMMAND             CREATED          STATUS          PORTS                                       NAMES
29b9709baf05   spring8   "catalina.sh run"   30 seconds ago   Up 29 seconds   0.0.0.0:80->8080/tcp, :::80->8080/tcp   spring8
[ec2-user@ip-172-31-40-46 java-hello-world-webapp]$ docker exec -it 29b9709baf05 /bin/bash
root@29b9709baf05:/usr/local/tomcat# ls
BUILDING.txt     LICENSE  README.md      RUNNING.txt  conf     lib   native-jni-lib  webapps
CONTRIBUTING.md  NOTICE   RELEASE-NOTES  bin          include  logs  temp            work
root@29b9709baf05:/usr/local/tomcat# cd conf/
root@29b9709baf05:/usr/local/tomcat/conf# cat application.properties
"key" = "value"
```

Benefits:

- **Integration**: Integrates with external services or APIs required by the application, enabling seamless communication and data exchange.
- **Compatibility**: Ensures compatibility with third-party systems or services by including necessary dependencies in Docker images.
- **Operational Efficiency**: Simplifies deployment and management of applications with external dependencies, reducing configuration overhead

Scenario 9: Using Build Arguments

Link - [scenario_based_learnings/Dockerfiles/Dockerfile9.txt at main · praveen1994dec/scenario_based_learnings (github.com)]

Purpose: Use build arguments to customize the Docker build process.

The purpose of using build arguments (`MAVEN_VERSION`, `JAVA_VERSION`) is to parameterize the Docker image build process. Build arguments allow customization of dependencies or runtime environments without modifying the Dockerfile directly, improving flexibility and maintainability.

Benefits:

- **Configuration Flexibility**: Allows customization of Docker image builds based on environment-specific variables or user-defined parameters.
- **Version Control**: Supports versioning and tracking of build configurations, improving reproducibility and consistency across deployments.
- **Automation**: Facilitates automated builds and deployments using CI/CD pipelines by parameterizing build processes with configurable arguments.

Scenario 10: Handling Timezones

Link - [scenario_based_learnings/Dockerfiles/Dockerfile10.txt at main · praveen1994dec/scenario_based_learnings (github.com)]

Purpose: The purpose of setting the timezone (`Asia/Kolkata`) in the Docker container environment is to ensure that date and time-related operations within the application reflect the local timezone. This approach helps in maintaining accurate timestamps and scheduling tasks based on local time.

```
# Use Tomcat 9 image with OpenJDK 11
FROM tomcat:9-jre11-openjdk-slim

# Install timezone data package (if not already included)
RUN apt-get update && apt-get install -y tzdata

# Set the timezone (example: Asia/Kolkata)
ENV TZ=Asia/Kolkata

# Remove existing applications in Tomcat webapps directory
RUN rm -rf /usr/local/tomcat/webapps/*

# Copy the WAR file from local filesystem into the container at the Tomcat webapps directory
COPY ./target/java-hello-world.war /usr/local/tomcat/webapps/ROOT.war

# Expose default Tomcat port
EXPOSE 8080

# Command to run Tomcat
CMD ["catalina.sh", "run"]
```

```
[ec2-user@ip-172-31-40-46 java-hello-world-webapp]$ docker run -dit --name spring10 -p 80:8080 spring10
0aa05a91c0c0ffe2661d6d0e9def5e2e810594550e65e302eec5a8fadc4dee18
[ec2-user@ip-172-31-40-46 java-hello-world-webapp]$ docker ps
CONTAINER ID   IMAGE      COMMAND             CREATED         STATUS         PORTS                                   NAMES
0aa05a91c0c0   spring10   "catalina.sh run"   4 seconds ago   Up 3 seconds   0.0.0.0:80->8080/tcp, :::80->8080/tcp   spring10
[ec2-user@ip-172-31-40-46 java-hello-world-webapp]$ docker exec -it 0aa05a91c0c0 /bin/bash
root@0aa05a91c0c0:/usr/local/tomcat# date
Sun Jul 14 01:44:04 IST 2024
root@0aa05a91c0c0:/usr/local/tomcat# cat /etc/timezone
Etc/UTC
root@0aa05a91c0c0:/usr/local/tomcat# exit
exit
```

Benefits:

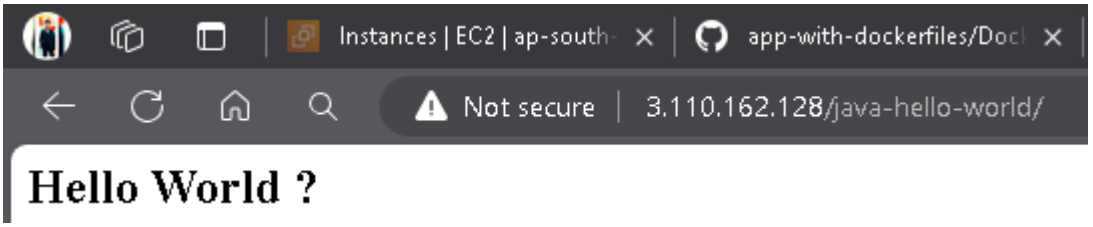
- **Localization**: Ensures accurate date and time handling within Docker containers based on specific geographic regions or user preferences.
- **Application Consistency**: Maintains consistency in timestamp records and scheduled tasks across distributed systems or global deployments.
- **User Experience**: Improves usability by presenting time-related information in local timezones, enhancing user experience and application functionality.

Scenario 11. Using Multi-Stage Builds for Different Environments

Link - scenario_based_learnings/Dockerfiles/Dockerfile11.txt at main · praveen1994dec/scenario_based_learnings (github.com)

Purpose: The purpose of using multi-stage builds for different environments (development and production) is to optimize the Docker image for each stage. Development stages may include additional tools or dependencies for debugging and testing, while production stages focus on minimizing image size and enhancing performance.

```
ec2-user@ip-172-31-40-46:~/Docker/java-hello-world-webapp
[ec2-user@ip-172-31-40-46 java-hello-world-webapp]$
[ec2-user@ip-172-31-40-46 java-hello-world-webapp]$ docker images
REPOSITORY   TAG       IMAGE ID       CREATED             SIZE
spring11     latest    a5dbd0703e87   2 minutes ago       294MB
spring10     latest    13ae31bce5e3   6 minutes ago       266MB
spring9      latest    5e1209abf613   17 minutes ago      294MB
spring8      latest    f55fb9c2b9bd   24 minutes ago      294MB
spring7      latest    bb0d0889d86e   34 minutes ago      454MB
spring6      latest    e1c9cc3de997   39 minutes ago      450MB
spring5      latest    cc9d81bbdd81   52 minutes ago      444MB
spring4      latest    9507ee0e9594   About an hour ago   244MB
spring2      latest    50479fa5b3e6   About an hour ago   468MB
spring1      latest    a69070a00a09   2 hours ago         449MB
spring3      latest    c4e22e65d807   2 hours ago         449MB
[ec2-user@ip-172-31-40-46 java-hello-world-webapp]$ docker run -dit --name spring11 -p 80:8080 spring11
4a8bad5a48eead793862b47b66bca520d68f1bf012e0a7fc10839b6c5df7f286
[ec2-user@ip-172-31-40-46 java-hello-world-webapp]$ docker ps
CONTAINER ID   IMAGE      COMMAND             CREATED          STATUS          PORTS                                   NAMES
4a8bad5a48ee   spring11   "catalina.sh run"   2 seconds ago    Up 2 seconds    0.0.0.0:80->8080/tcp, :::80->8080/tcp   spring11
[ec2-user@ip-172-31-40-46 java-hello-world-webapp]$ cat Dockerfile
# Stage 1: Build environment
FROM maven:3.8.4-openjdk-11-slim AS build

WORKDIR /app

# Copy only the pom.xml to cache dependencies
COPY pom.xml .

# Fetch dependencies
RUN mvn dependency:go-offline

# Copy the source code and build application
COPY src ./src
RUN mvn package -DskipTests

# Stage 2: Deploy to Tomcat environment
FROM tomcat:9-jre11-slim

# Remove existing applications in Tomcat webapps directory
RUN rm -rf /usr/local/tomcat/webapps/*

# Copy the WAR file built in the previous stage into the Tomcat webapps directory
COPY --from=build /app/target/java-hello-world.war /usr/local/tomcat/webapps/ROOT.war

# Expose Tomcat port
EXPOSE 8080

# Start Tomcat server
CMD ["catalina.sh", "run"]
```



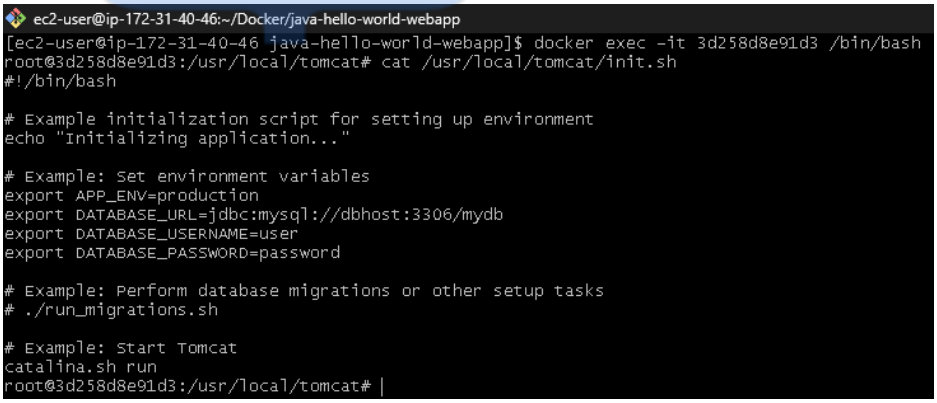
Benefits:

- **Development Efficiency**: Separates development dependencies from production environments, optimizing developer productivity and build times.
- **Production Optimization**: Creates lean and optimized Docker images for production deployments, minimizing image size and reducing attack surfaces.
- **Environment Consistency**: Ensures consistent application behavior across development, testing, and production environments by using environment-specific build stages.

Scenario 12. Running Initialization Scripts

Link - scenario_based_learnings/Dockerfiles/Dockerfile12.txt at main · praveen1994dec/scenario_based_learnings (github.com)

Purpose: The purpose of executing initialization scripts (`init.sh`) during container startup is to perform pre-deployment tasks such as environment setup, database migrations, or application configuration. Initialization scripts automate routine tasks, ensuring consistent and reliable container initialization.



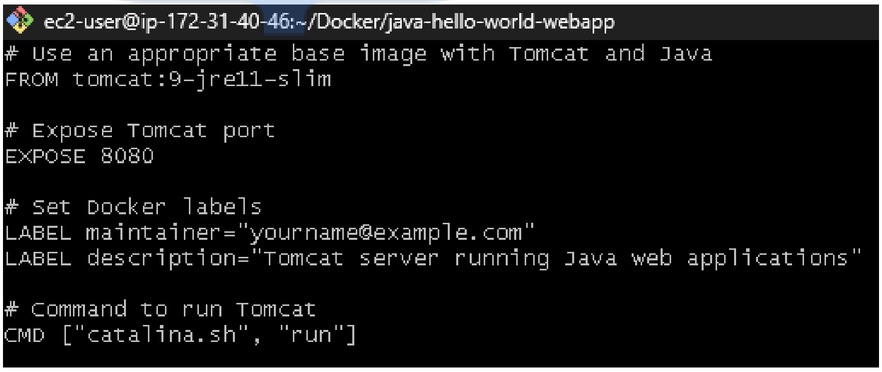
Benefits:

- **Automation**: Automates pre-deployment tasks such as environment setup, database schema migrations, or application configuration.
- **Consistency**: Ensures consistent and reproducible container initialization across different deployment environments.

Scenario 13. Using Docker Labels

Link - [scenario_based_learnings/Dockerfiles/Dockerfile13.txt at main · praveen1994dec/scenario_based_learnings (github.com)]

Purpose: The purpose of adding Docker labels (`maintainer`, `description`) to the Docker image is to provide metadata that describes the image's purpose, ownership, and other relevant information. Docker labels enhance image documentation, tracking, and management across development, testing, and production environments.



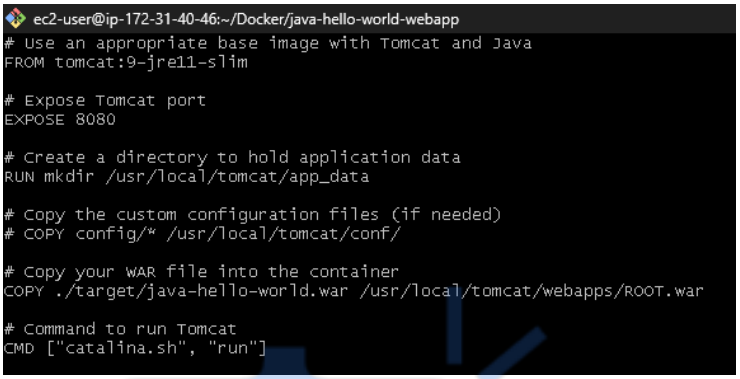
Benefits:

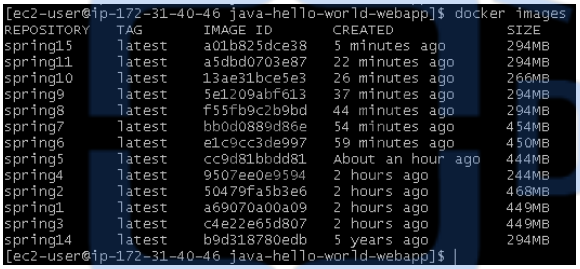
- **Documentation**: Provides metadata and descriptive information about Docker images, enhancing visibility and understanding of image purpose and ownership.
- **Tracking**: Supports image management and tracking throughout its lifecycle, including versioning, updates, and deployment history.
- **Compliance and Governance**: Facilitates compliance with organizational policies and governance requirements by labeling Docker images with relevant information.

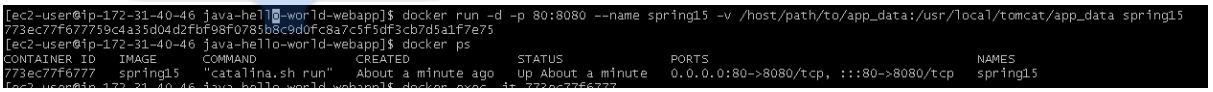
Scenario 14: Using Docker Volumes for Persistent Data

Link - scenario_based_learnings/Dockerfiles/Dockerfile14.txt at main · praveen1994dec/scenario_based_learnings (github.com)

Purpose: The purpose of using Docker volumes in a Dockerfile is to manage and persist application data outside the container filesystem. This approach ensures data persistence across container restarts and allows for data sharing between containers or with the host system.







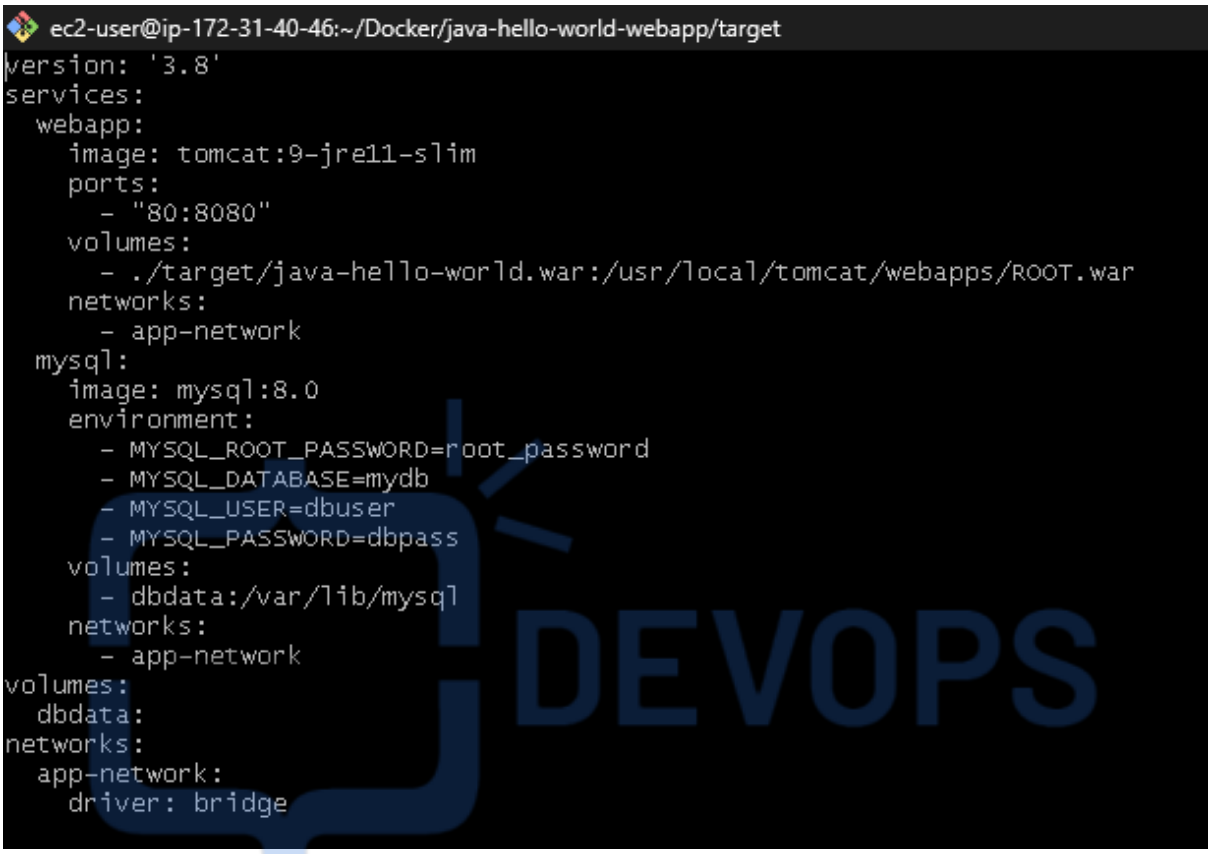
Benefits:

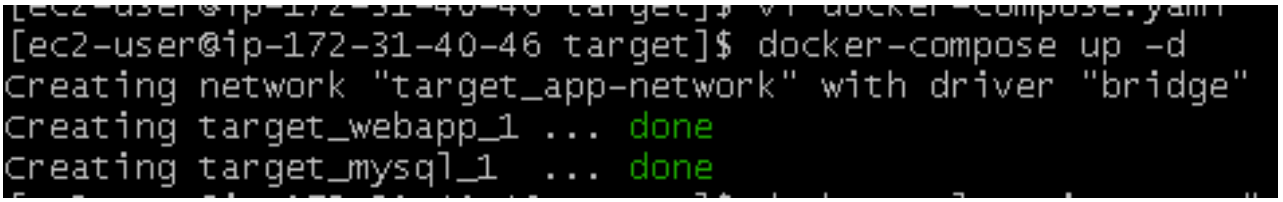
- **Persistence**: Data stored in Docker volumes persists even if the container is stopped or removed.
- **Separation of Concerns**: Separating data from the container filesystem simplifies backup, restoration, and migration of application data.
- **Flexibility**: Docker volumes can be easily shared between containers, enabling data exchange and collaboration in multi-container applications.

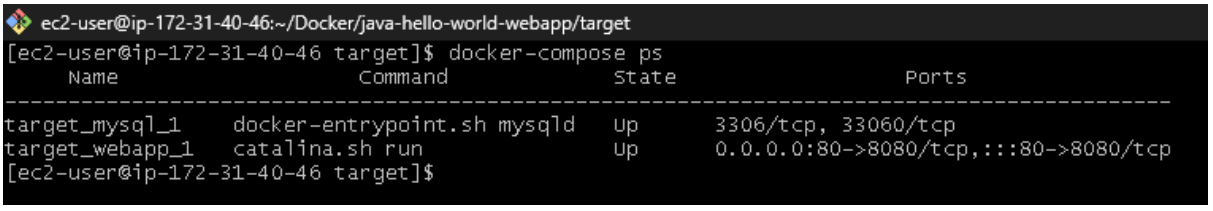
DOCKER COMPOSE SCENARIOS

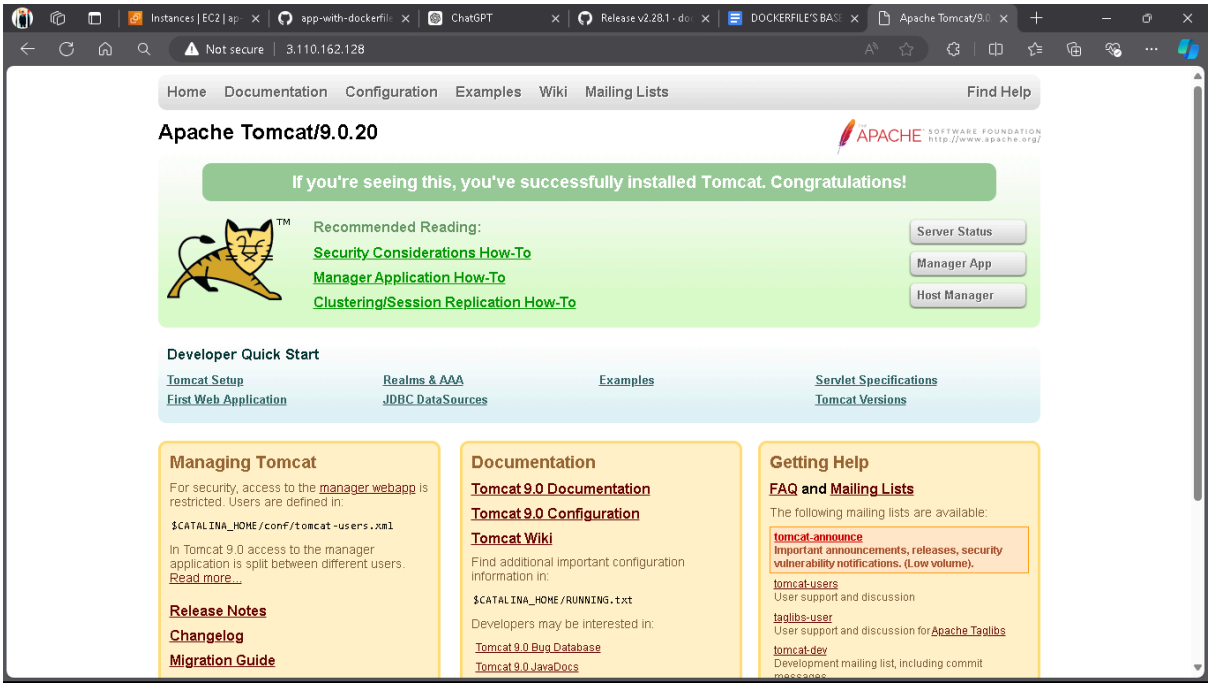
Scenario 1: Development Environment Setup

Objective: Setting up a development environment for the Java web application using Docker Compose.





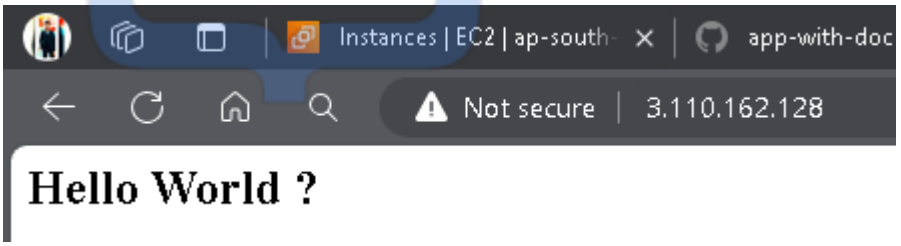




Scenario 4: Docker Compose for Java Web Application Deployment

Objective: Build and deploy a Java web application (`java-hello-world.war`) using Docker Compose.

```
[ec2-user@ip-172-31-40-46 java-hello-world-webapp]$ cat Dockerfile
# Dockerfile for building Tomcat deployment image
FROM tomcat:9-jre11-slim

# Remove existing ROOT application
RUN rm -rf /usr/local/tomcat/webapps/ROOT

# Copy WAR file to webapps directory
COPY ./target/java-hello-world.war /usr/local/tomcat/webapps/ROOT.war

# Expose Tomcat port
EXPOSE 8080

# Start Tomcat
CMD ["catalina.sh", "run"]

[ec2-user@ip-172-31-40-46 java-hello-world-webapp]$ cat docker-compose.yaml
version: '3.8'
services:
  webapp:
    build:
      context: .
      dockerfile: Dockerfile
    ports:
      - "80:8080"
    volumes:
      - ./target/java-hello-world.war:/usr/local/tomcat/webapps/ROOT.war

[ec2-user@ip-172-31-40-46 java-hello-world-webapp]$
```




```
[ec2-user@ip-172-31-40-46 java-hello-world-webapp]$ docker-compose up -d --build
Creating network "java-hello-world-webapp_default" with the default driver
Building webapp
[+] Building 0.5s (8/8) FINISHED                                                                                    docker:default
 => [internal] load build definition from Dockerfile                                                                        0.0s
 => => transferring dockerfile: 437B                                                                                        0.0s
 => [internal] load metadata for docker.io/library/tomcat:9-jre11-slim                                                      0.0s
 => [internal] load .dockerignore                                                                                           0.0s
 => => transferring context: 2B                                                                                             0.0s
 => CACHED [1/3] FROM docker.io/library/tomcat:9-jre11-slim                                                                 0.0s
 => [internal] load build context                                                                                           0.0s
 => => transferring context: 193B                                                                                           0.0s
 => [2/3] RUN rm -rf /usr/local/tomcat/webapps/ROOT                                                                         0.3s
 => [3/3] COPY ./target/java-hello-world.war /usr/local/tomcat/webapps/ROOT.war                                             0.0s
 => exporting to image                                                                                                      0.1s
 => => exporting layers                                                                                                     0.0s
 => => writing image sha256:3a16c033529f0d3202ea53eda1f1350af4c6af9848b95bbd9f1f035042c1a6be                                0.0s
 => => naming to docker.io/library/java-hello-world-webapp_webapp                                                           0.0s
Creating java-hello-world-webapp_webapp_1 ... done
[ec2-user@ip-172-31-40-46 java-hello-world-webapp]$ docker-compose ps
              Name                     Command       State                 Ports
-------------------------------------------------------------------------------------------
java-hello-world-webapp_webapp_1   catalina.sh run   Up      0.0.0.0:80->8080/tcp,:::80->8080/tcp
[ec2-user@ip-172-31-40-46 java-hello-world-webapp]$
```