1. **FROM maven:3.8.6-amazoncorretto-17 AS myapp-build**:
   * This sets the base image to **maven:3.8.6-amazoncorretto-17** which is an image containing Maven and Amazon Corretto 17 JDK.
   * It also assigns the alias **myapp-build** to this stage for reference later in the Dockerfile.
2. **ENV MYAPP\_HOME /opt/myapp**:
   * Sets an environment variable **MYAPP\_HOME** with the value **/opt/myapp**. This will be the working directory for the application.
3. **WORKDIR $MYAPP\_HOME**:
   * Changes the working directory inside the container to **$MYAPP\_HOME**.
4. **COPY pom.xml .**:
   * Copies the **pom.xml** file from the local directory (where the Dockerfile is located) into the container's working directory.
5. **COPY src ./src**:
   * Copies the entire **src** directory (source code) from the local directory into the container's working directory.
6. **RUN mvn package -DskipTests**:
   * Runs the Maven command to package the application. **-DskipTests** skips the execution of tests during the build process.
7. **FROM amazoncorretto:17**:
   * Starts a new stage with Amazon Corretto 17 as the base image.
8. **ENV MYAPP\_HOME /opt/myapp**:
   * Sets the same environment variable **MYAPP\_HOME** with the value **/opt/myapp**. This ensures consistency with the previous stage.
9. **WORKDIR $MYAPP\_HOME**:
   * Changes the working directory inside the container to **$MYAPP\_HOME**.
10. **COPY --from=myapp-build $MYAPP\_HOME/target/\*.jar $MYAPP\_HOME/myapp.jar**:

* Copies the JAR file (result of the Maven build) from the previous stage (**myapp-build**) into the current stage.

1. **ENTRYPOINT java -jar myapp.jar**:

* Specifies the command that should be executed when a container is started based on this image. In this case, it runs the Java application with the **myapp.jar** file as the main executable.

A multistage build in Docker allows you to use multiple **FROM** statements in a single Dockerfile, resulting in a more efficient and lightweight final image. This technique is particularly useful when you need to build complex applications or services with various dependencies and build tools.

In a multistage build, you have two or more stages, each with its own **FROM** statement. Each stage is a separate phase of the build process, and the artifacts (files, binaries, etc.) generated in one stage can be copied into subsequent stages.