A **multi-stage Dockerfile** is a technique in Docker that allows you to use multiple FROM statements in a single Dockerfile to build and optimize your container images. It helps reduce the final image size by keeping only the necessary artifacts and dependencies while discarding unnecessary build dependencies.

**Why Use Multi-Stage Builds?**

* **Smaller final image size**: Unnecessary build dependencies are removed, reducing the attack surface and resource usage.
* **Improved security**: Since intermediate build artifacts are discarded, there are fewer vulnerabilities in the final image.
* **Better maintainability**: Keeps build and runtime environments separate, making the Dockerfile cleaner and more manageable.

**Example of a Multi-Stage Dockerfile**

Here’s an example of a **Golang application** using a multi-stage build:

dockerfile

CopyEdit

# Stage 1: Build the Go application

FROM golang:1.20 AS builder

WORKDIR /app

COPY . .

RUN go mod tidy && go build -o myapp

# Stage 2: Create a minimal final image

FROM alpine:latest

WORKDIR /root/

COPY --from=builder /app/myapp .

CMD ["./myapp"]

**How It Works**

1. **First Stage (builder)**
   * Uses golang:1.20 as the base image.
   * Copies the source code and builds the application.
   * Produces a compiled binary (myapp).
2. **Second Stage (Final Image)**
   * Uses a minimal alpine:latest image to keep the final image small.
   * Copies the compiled binary (myapp) from the builder stage.
   * Runs the application with a minimal footprint.

**Use Cases**

* **Building and running compiled languages** (e.g., Go, C++, Java, etc.).
* **Reducing image size** by removing unnecessary build tools.
* **Keeping different dependencies for build and runtime** environments.
* **Optimizing frontend applications** (e.g., compiling a React app and serving it with Nginx).