**Structre:-**

my-docker-optimization-project/

│

├── app/

│ ├── Dockerfile # Your Dockerfile

│ ├── requirements.txt # Ensure this file exists

│ └── app.py # Your application code

├── .dockerignore

├── optimize-build.sh

└── docker-compose.yml

Code for Dockerfile:-

# Stage 1: Build Stage (install dependencies)

FROM python:3.9-alpine AS builder

WORKDIR /app

# Copy the requirements file first to optimize caching

COPY requirements.txt .

# Install the dependencies

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

# Stage 2: Final Stage (minimal runtime image)

FROM python:3.9-alpine

WORKDIR /app

# Copy only the necessary files from the build stage

COPY --from=builder /app /app

# Copy the rest of the application code into the container

COPY . /app

# Set the default command

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

Code for requirements.txt:-

flask

requests

code for app.py:-

# app.py

print("Hello, Docker Optimization!")

dockerignore:--

\_\_pycache\_\_

\*.pyc

.git

node\_modules/

\*.log

.env

Code for optimize build.sh:-

#!/bin/bash

echo "Optimizing Docker Build Process"

# Start tracking time

START\_TIME=$(date +%s)

# Build the optimized Docker image

docker build --no-cache -t optimized-app .

# Measure build time

END\_TIME=$(date +%s)

BUILD\_TIME=$((END\_TIME - START\_TIME))

echo "Build Time: ${BUILD\_TIME} seconds"

# Optionally, track resource usage during the build process

# You could use tools like `docker stats` or integrate with a monitoring service here.

# Clean up any unused Docker images

docker system prune -f

echo "Docker build completed successfully"

code for docker compose.yml:-

version: '3.8'

services:

app:

build: ./app

container\_name: optimized-app-container

ports:

- "5000:5000"

depends\_on:

- db

db:

image: postgres:alpine

container\_name: postgres-container

environment:

POSTGRES\_USER: user

POSTGRES\_PASSWORD: password

POSTGRES\_DB: appdb

volumes:

- db\_data:/var/lib/postgresql/data

volumes:

db\_data:







