Docker file with multistage:

```
# Use an official Node.js image as a base
FROM node:18-alpine AS builder
# Set the working directory inside the container
WORKDIR /app
# Copy package.json and package-lock.json (if available) first
COPY package.json ./
COPY package-lock.json ./
# Install dependencies
RUN npm install --frozen-lockfile
# Copy the entire project (except files in .dockerignore)
COPY . .
# Build the React app
RUN npm run build
# Use a lightweight web server to serve the built files
FROM nginx:alpine
# Copy build output from previous stage to Nginx's public directory
COPY --from=builder /app/build /usr/share/nginx/html
# Expose port 80 to be used by the container
EXPOSE 80
# Start Nginx server
CMD ["nginx", "-g", "daemon off;"]
```

```
ARG NODE_ENV=production

RUN if [ "$NODE_ENV" = "development" ]; \

   then npm install; \

   else npm install --omit=dev; \

fi
```

Cache Backend	Command	Use Case
Inline Cache (Default)	`sh docker build --build-arg BUILDKIT_INLINE_CACHE=1 -t myapp . docker push myapp`	Stores cache inside the built image and pushes it to a registry.
Use Image Cache in Build	`sh docker build --cache-from=myapp -t myapp:latest .`	Reuses cache from a previously pushed image in a registry.
Local Disk Cache	`sh docker build --cache-to=type=local,dest=/tmp/cache --cache-from=type=local,src=/tmp/cache -t myapp .`	Saves cache to a local directory, useful for local builds.
Registry Cache (Docker Hub, ECR, GCR, etc.)	`sh docker build --cache-to=type=registry,ref=myrepo/mycache:latest`	Stores and retrieves cache from a Docker registry for CI/CD.

	`--cache-from=type=registry,ref=myrepo/mycache:latest -t myapp .`	
Remote Cache (Experimental)	`sh docker build --cache-to=type=remote,url=https://my-cache-server/cache --cache-from=type=remote,url=https://my-cache-server/cache -t myapp .`	Uses an HTTP-based remote cache (e.g., Amazon S3, Google Cloud Storage) for centralized caching.

CI

docker build --progress=plain . ->enable annotations

Actions workflow:

```
name: CI/CD Pipeline

on:
  push:
        branches:
        - main
  pull_request:
        branches:
        - main

jobs:
  build_and_test:
        name: Build, Test, and Push to Docker Hub
        runs-on: ubuntu-latest

        env:
        REGISTRY: docker.io
        IMAGE_NAME: ${{ secrets.DOCKER_USERNAME }}/myapp

        steps:
        # 🔹 Step 1: Checkout repository
        - name: Checkout repository
        uses: actions/checkout@v4

        # 🔹 Step 2: Set up Docker Buildx for multi-platform builds
        - name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v3

        # 🔹 Step 3: Cache dependencies (Node.js & Docker layers)
        - name: Cache Node.js dependencies
        uses: actions/cache@v4
        with:
        path: ~/.npm
        key: node-modules-${{ runner.os }}-${{ hashFiles('**/package-lock.json') }}
        restore-keys: |
        node-modules-${{ runner.os }}-

        - name: Cache Docker layers
        uses: actions/cache@v4
        with:
```

```
path: /tmp/.buildx-cache
key: docker-cache-${{ github.sha }}
restore-keys: |
docker-cache-

# 🔹 Step 4: Install dependencies
- name: Install dependencies
run: npm ci

# 🔹 Step 5: Run ESLint for code quality check
- name: Lint code
run: npm run lint

# 🔹 Step 6: Run unit tests
- name: Run unit tests
run: npm test

# 🔹 Step 7: Build the Docker Compose services
- name: Build Docker Compose
run: docker compose build

# 🔹 Step 9: Log in to Docker Hub using secrets
- name: Log in to Docker Hub
uses: docker/login-action@v3
with:
username: ${{ secrets.DOCKER_USERNAME }}
password: ${{ secrets.DOCKER_PASSWORD }}

# 🔹 Step 10: Push image to Docker Hub with cache support
- name: Push Docker Image
run: |
docker buildx build \
--platform linux/amd64,linux/arm64 \
--cache-from=type=local,src=/tmp/.buildx-cache \
--cache-to=type=local,dest=/tmp/.buildx-cache \
--secret id=db_password,src=./secrets/db_password.txt \
-t ${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}:latest \
--push .

# 🔹 Step 11: Clean up Docker cache
- name: Clean up Docker cache
run: |
docker builder prune -af
```

Basic Docker Compose:

```
version: '3.8'

services:
  backend:
      build:
      context: ./backend  # Backend build context
      dockerfile: Dockerfile  # Backend Dockerfile
      args:
      - NODE_ENV=${NODE_ENV}  # Use environment variable for flexibility
      container_name: backend
      restart: always
```

```
      ports:
      - "5000:5000"
      environment:
      - MONGOURL=${MONGO_URL}  # Read MongoDB URL from .env file
      volumes:
      - ./backend:/app  # Sync backend code for live reload
      - backend_node_modules:/app/node_modules  # Use named volume for
dependencies
      - ./logs/backend:/app/logs  # Persist backend logs
      command: npm run dev  # Consider `node server.js` in production
      depends_on:
      mongodb:
      condition: service_healthy  # Ensures MongoDB is ready before backend starts
      networks:
      - app_network

  frontend:
      build:
      context: ./client  # Frontend build context
      dockerfile: Dockerfile  # Frontend Dockerfile
      container_name: frontend
      restart: always
      ports:
      - "3000:3000"
      volumes:
      - ./client:/app  # Sync frontend code for hot reload
      - frontend_node_modules:/app/node_modules  # Use named volume for
dependencies
      - ./logs/frontend:/app/logs  # Persist frontend logs
      environment:
      - CHOKIDAR_USEPOLLING=true  # Ensures file change detection in Docker
      depends_on:
      - backend
      networks:
      - app_network

  mongodb:
      image: mongo:6.0
      container_name: mongo_db
      restart: always
      ports:
      - "27017:27017"
      volumes:
      - mongodb_data:/data/db  # Persistent MongoDB storage
      environment:
      - MONGO_INITDB_DATABASE=Ecommerce_Database  # Set initial database
      healthcheck:
      test: ["CMD", "mongosh", "--eval", "db.runCommand({ping:1})"]
      interval: 10s
      retries: 5
      start_period: 30s
      networks:
      - app_network

volumes:
  mongodb_data:
  backend_node_modules:
  frontend_node_modules:
```

```
networks:
  app_network:
       driver: bridge
```

Bash:

```
#!/bin/bash

# 1. Variables
name="Abdullah"
echo "Hello, $name!"

# Read User Input
read -p "Enter your name: " user_name
echo "Welcome, $user_name!"

file="test.txt"
if [ -f "$file" ]; then
  echo "File exists"
else
  echo "File does not exist"
fi

#Loops
for i in {1..5}; do
  echo "Number: $i"
done

count=1
while [ $count -le 5 ]; do
  echo "Count: $count"
  ((count++))
done

# Functions
greet() {
  echo "Hello, $1!"
}
greet "Abdullah"

# Working with Files & Directories
mkdir -p new_folder  # Create a directory
ls -l  # List files

echo "Reading file line by line:"
while read line; do
  echo "$line"
done < file.txt

# 8. Error Handling
command_that_fails || echo "Command failed"

# 9. Useful Shortcuts
echo "PWD: $(pwd)"
echo "Home Directory: $HOME"
echo "Exit Status of Last Command: $?"
```

```
# 10. Next.js Project Testing, Running, and Building
echo "Managing Next.js Project"
npm install  # Install dependencies
npm run dev  # Run development server
npm run build  # Build the project
npm start  # Start the production server
```