

# Lab 3 Program: Dockerfile for a Node.js App

This guide explains the complete procedure for setting up a Node.js project and containerizing it using a multistage Dockerfile, based on the provided lab files.

## 1. Final Folder Structure

This is the complete folder structure you will have after setting up the project and running the build script.

```
Program-2/
├── src/
│   └── index.js      # Your Node.js application source code
├── dist/
│   └── index.js      # The "built" application code (copied from src)
├── node_modules/
│   └── ...            # All installed dependencies (like 'express')
├── Dockerfile        # The multistage build instructions
├── package.json       # Project definition, scripts, and dependencies
└── package-lock.json  # Records exact dependency versions
```

## Explanation of Files & Folders

- **src/index.js:** This is where you write your application logic.
- **package.json:** This file is the "manifest" for your Node.js project. It defines:
  - **dependencies:** The libraries your app needs to run (e.g., express).
  - **scripts:** Commands you can run. In your case:
    - "build": A script that creates the dist folder and copies the source code into it. This simulates a real-world build step (like compiling TypeScript).
    - "start": A script that runs the final application from the dist folder.
- **Dockerfile:** This is the instruction manual for Docker to build your image. It's special because it has two FROM instructions, creating a "multistage" build.
- **dist/:** This "distribution" folder holds the code that is ready to be run in production. It's generated by the npm run build command.
- **node\_modules/:** This folder is created by npm install and contains all the external libraries (dependencies) your project needs.

## 2. File Contents

Here is the exact code you need for each file.

## **src/index.js**

(This is your simple Express web server)

```
const express = require('express');
const app = express();
const PORT = 3000;

app.get('/', (req, res) => {
  res.send('Hello from Multi-stage Docker!');
});

app.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
});
```

## **package.json**

(This defines your project and its scripts)

```
{
  "name": "program-2",
  "version": "1.0.0",
  "description": "A simple Node.js app with multi-stage dockerfile",
  "main": "dist/index.js",
  "scripts": {
    "start": "node dist/index.js",
    "build": "mkdir -p dist && cp -r src/* dist/"
  },
  "author": "Sahana",
  "license": "MIT",
  "dependencies": {
    "express": "4.18.2"
  }
}
```

## **Dockerfile**

(This is the core of the lab: the multistage instructions)

```
# Stage 1: Build Stage
# We use a full Node.js image and name this stage 'builder'
FROM node:18-alpine AS builder
```

```
# Set the working directory inside the container
WORKDIR /app

# Copy package files first to leverage Docker's layer cache
COPY package.json package-lock.json ./

# Install all dependencies (including devDependencies, if any)
RUN npm install

# Copy the rest of the application source code
COPY ..

# Run the build script defined in package.json
# This will create the /app/dist folder inside this stage
RUN npm run build

# ---

# Stage 2: Production Stage
# We start from a fresh, lightweight alpine image
FROM node:18-alpine

# Set the working directory
WORKDIR /app

# Copy ONLY the necessary files from the 'builder' stage
COPY --from=builder /app/package.json .
COPY --from=builder /app/package-lock.json .
COPY --from=builder /app/dist ./dist
COPY --from=builder /app/node_modules ./node_modules

# Expose the port the application runs on
EXPOSE 3000

# The command to start the application
CMD [ "node", "dist/index.js" ]
```

### 3. Detailed Step-by-Step Procedure

Follow these commands in your terminal to create and run the project.

## **Step 1: Set Up the Project Directory**

1. Create the main folder for your project.  
`mkdir Program-2`

2. Move into the new directory.  
`cd Program-2`

## **Step 2: Create the Source Code**

1. Create the src folder.

```
mkdir src
```

2. Create the index.js file inside src.

```
# You can use a text editor like 'nano' or 'vim', or just 'touch'
```

```
# For example, with nano:
```

```
nano src/index.js
```

3. Copy and paste the src/index.js content from Section 2 into this file and save it.

## **Step 3: Create the package.json File**

1. Run `npm init -y` (this creates the basic `package.json`).
2. Manually edit the `package.json` file to add the "build" script under the "scripts" section.

## **Step 4: Install Dependencies**

1. Run the npm install command. This will read your package.json, download express, and create the node\_modules and package-lock.json files.  
`>npm install`

## **Step 5: Create the Dockerfile**

1. In the Program-2 root, create the Dockerfile.  
`nano Dockerfile`

2. Copy and paste the Dockerfile content from Section 2 into this file and save it.

## **Step 6: Build Your Docker Image**

1. Now, you'll build the image. This command tells Docker to read the Dockerfile in the current directory(.) and build an image, tagging it (-t) with the name my-node-app.  
`docker build -t my-node-app .`

2. You will see Docker execute each step from your file. Notice how it runs Stage 1 first, then starts fresh for Stage 2 and only copies files over.

```
1RV24MC089_SAHPANA_H_J@sahana:~/DevOps_Automation/pr$ docker build -t lab3 .
[+] Building 236.5s (10/10) FINISHED                                            docker:default
=> [internal] load build definition from Dockerfile                           0.0s
=> => transferring dockerfile: 157B                                         0.0s
=> [internal] load metadata for docker.io/library/node:latest                6.8s
=> [internal] load .dockerrcignore                                           0.0s
=> => transferring context: 2B                                              0.0s
=> [1/5] FROM docker.io/library/node:latest@sha256:0b1fad950f54a1d6f9e8e580205c157b43315d2c4231c3a  226.8s
=> => resolve docker.io/library/node:latest@sha256:0b1fad950f54a1d6f9e8e580205c157b43315d2c4231c3a0b  0.0s
=> => sha256:5cdcc8ed4f59b12fd00bf3cf863adeb51841e9c215f3d2b03842d77f750d9bb914 6.75kB / 6.75kB  0.0s
=> => sha256:5d93aea697980315f27f81c68582d14f63dd3579c23a27dc495a588279eda20 48.48MB / 48.48MB  81.5s
=> => sha256:bb445e472b1bad54f5a28edd51b11aec79eca8513394866a261891be9da6a343 24.03MB / 24.03MB  28.0s
=> => sha256:2123190679e81d983648d92f1bb9ddc74383512edb00ad64f93d24d00d8807a 64.40MB / 64.40MB  88.1s
=> => sha256:0b1fad950f54a1d6f9e8e580205c157b43315d2c4231c3a0b78137d87fb92fa 5.14kB / 5.14kB  0.0s
=> => sha256:260d5fc808df32b294d6eb34f31fef285b2e7fdc0986989517607b37aa40495f 2.49kB / 2.49kB  0.0s
=> => sha256:32885a2b0a589e832bf6b250bd35a528b268360f166af2cd7094d3a14993fcc1 211.45MB / 211.45MB  219.6s
=> => sha256:d12117b46e66d25e585fad6bb9cb05ab0cedfff0a9cf634b7d0bb4a6fe0f0848 3.32kB / 3.32kB  82.7s
=> => extracting sha256:5d93aea697980315f27f81c68582d14f63dd3579c23a27dc495a588279eda20 1.5s
=> => sha256:1c036b4ac03eae628080e4b172cbddc4f169887aeb81b1d979fcca796cb79c1 56.16MB / 56.16MB  140.7s
=> => extracting sha256:bb445e472b1bad54f5a28edd51b11aec79eca8513394866a261891be9da6a343 0.5s
=> => extracting sha256:2123190679e81d983648da92f1bb9ddc74383512edb00ad64f93d24d00d8807a 2.1s
=> => sha256:a9521666e9d84e13efe850468f5c3be3e068522ff9e8b9ba2158707f695221bf 1.25MB / 1.25MB  90.7s
=> => sha256:d56bd99507f882da74ece6b9f7c4aeee969118df49d83e47a3de443998cf3ed4 446B / 446B  91.9s
=> => extracting sha256:32885a2b0a589e832bf6b250bd35a528b268360f166af2cd7094d3a14993fcc1 4.8s
=> => extracting sha256:d12117b46e66d25e585fad6bb9cb05ab0cedfff0a9cf634b7d0bb4a6fe0f0848 0.0s
=> => extracting sha256:1c036b4ac03eae628080e4b172cbddc4f169887aeb81b1d979fcca796cb79c1 2.0s
=> => extracting sha256:a9521666e9d84e13efe850468f5c3be3e068522ff9e8b9ba2158707f695221bf 0.0s
=> => extracting sha256:d56bd99507f882da74ece6b9f7c4aeee969118df49d83e47a3de443998cf3ed4 0.0s
=> [internal] load build context                                               0.1s
=> => transferring context: 2.31MB                                           0.1s
=> [2/5] WORKDIR /app                                                       0.5s
=> [3/5] COPY package*.json ./                                              0.0s
=> [4/5] RUN npm install express                                            1.9s
=> [5/5] COPY . . .                                                       0.1s
=> => exporting to image                                                    0.2s
=> => exporting layers                                                       0.2s
=> => writing image sha256:ce22c728b0ece058843bee56a4c52a7f9528ac9b7e90aa3716d287c9eade0ff7 0.0s
=> => naming to docker.io/library/lab3                                     0.0s
```

## Step 7: Run Your Docker Container

- Once the build is complete, run your new image as a container.

docker run -p 3000:3000 -d my-node-app

- docker run: The command to run a container.
- -p 3000:3000: Maps port 3000 on your computer (host) to port 3000 inside the container.
- -d: Runs the container in "detached" mode (in the background).
- my-node-app: The name of the image you want to run.

```
Node.js v25.1.0
1RV24MC089_SAHPANA_H_J@sahana:~/DevOps_Automation/pr$ docker run -p 3001:3000 lab3b
App listening at http://localhost:3000
```

## Step 8: Verify the Application

- Your container is now running! You can verify it in two ways:
- **Browser:** Open your web browser and navigate to <http://localhost:3000>.
  - **cURL:** Open a new terminal and run:  
curl <http://localhost:3000>

2. In both cases, you should see the message: **Hello from Multi-stage Docker!**



Hello World! This is Sahana from DevOps Automation Team.