Scenario

You’re given a tiny TypeScript/Express API that **must** be containerized twice:

1. A naïve **single-stage** image (works, but big).
2. A hardened **multi-stage** image (works, small, secure).

Your job is to meet the requirements.

Starter app (provided)

`package.json`

```json
{
  "name": "ts-api-demo",
  "version": "1.0.0",
  "type": "module",
  "main": "dist/index.js",
  "scripts": {
    "dev": "tsx watch src/index.ts",
    "build": "tsc -p tsconfig.json",
    "start": "node dist/index.js"
  },
  "dependencies": {
    "express": "^4.19.2"
  },
  "devDependencies": {
    "@types/express": "^4.17.21",
    "tsx": "^4.15.7",
    "typescript": "^5.5.4"
  }
}
```

`tsconfig.json`
```json
{
  "compilerOptions": {
    "target": "ES2022",
    "module": "ES2022",
    "moduleResolution": "bundler",
    "outDir": "dist",
    "rootDir": "src",
    "esModuleInterop": true,
    "strict": true
  },
  "include": ["src"]
}
```

`src/index.ts`
```ts
import express from "express";
const app = express();
const PORT = Number(process.env.PORT || 3000);
const APP_NAME = process.env.APP_NAME || "TS API";

app.get("/health", (_req, res) => {
  res.json({ ok: true, service: APP_NAME });
});

app.get("/whoami", (_req, res) => {
  res.json({ uid: process.getuid?.(), gid: process.getgid?.() });
});

app.listen(PORT, "0.0.0.0", () => {
  console.log(`[${APP_NAME}] listening on ${PORT}`);
});
```

Part A — Single-stage image (intentionally sloppy)

Create **`Dockerfile.single`** that:

- uses `node:20`
- installs dependencies
- builds the app
- starts with `npm run start`
- **must work** at `http://localhost:3000/health`

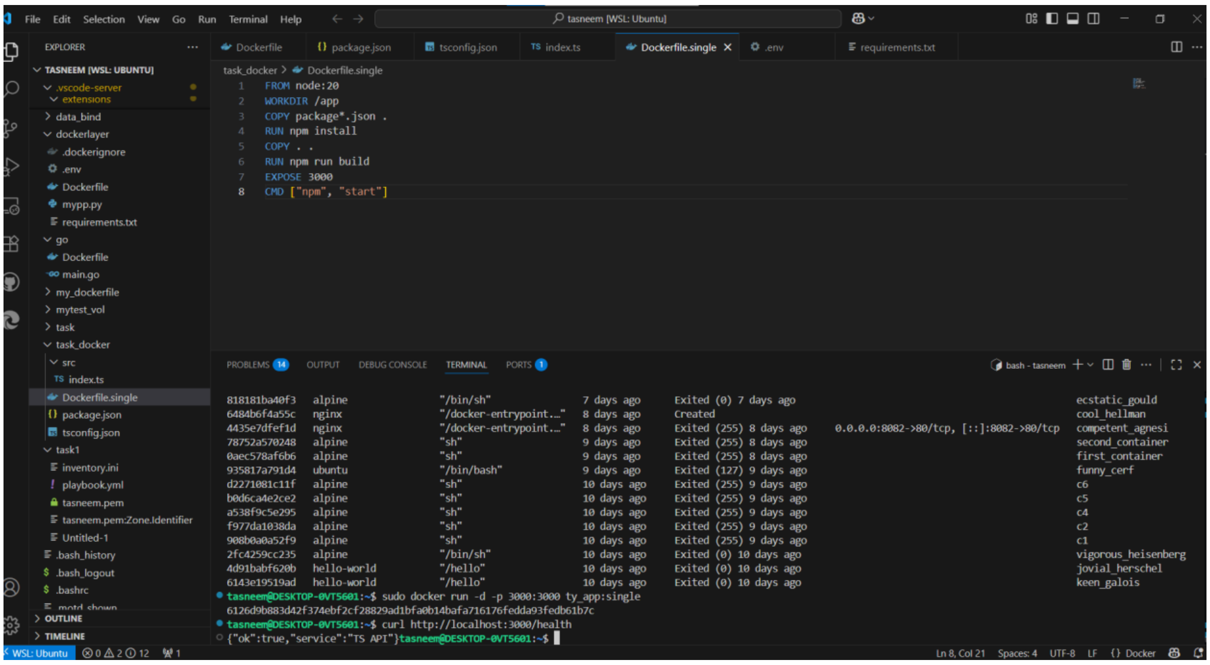
> This is the “baseline” (large image). It just needs to run.

CMD:

```
sudo docker build -f Dockerfile -t ty_app:single  .
sudo docker run -d -p 3000:3000  ty_app:single
```

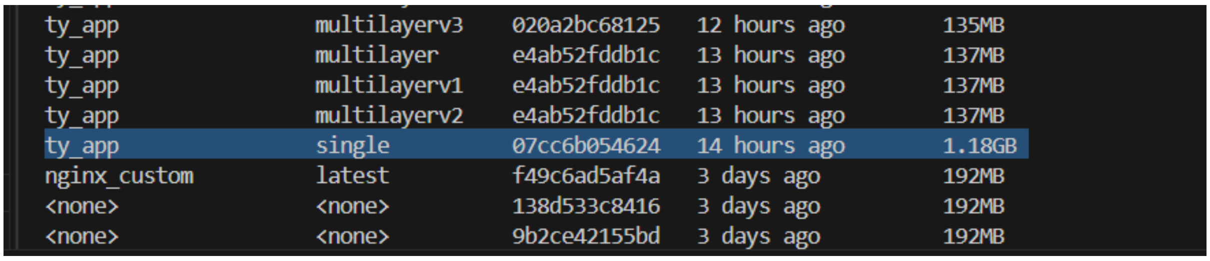
curl http://localhost:3000/health

Sudo docker images



```
FROM node:20
WORKDIR /app
COPY package*.json .
RUN npm install
COPY . .
RUN npm run build
EXPOSE 3000
CMD ["npm", "start"]
```

```
818181ba40f3   alpine        "/bin/sh"                 7 days ago    Exited (0) 7 days ago                                                  ecstatic_gould
6484b6f4a55c   nginx         "/docker-entrypoint.…"    8 days ago    Created                                                                cool_hellman
4435e7dfef1d   nginx         "/docker-entrypoint.…"    8 days ago    Exited (255) 8 days ago     0.0.0.0:8082->80/tcp, [::]:8082->80/tcp   competent_agnesi
78752a570248   alpine        "sh"                      9 days ago    Exited (255) 8 days ago                                                second_container
0aec578af6b6   alpine        "sh"                      9 days ago    Exited (255) 8 days ago                                                first_container
935817a791d4   ubuntu        "/bin/bash"               9 days ago    Exited (127) 9 days ago                                                funny_cerf
d2271081c11f   alpine        "sh"                      10 days ago   Exited (255) 9 days ago                                                c6
b0d6ca4e2ce2   alpine        "sh"                      10 days ago   Exited (255) 9 days ago                                                c5
a538f9c5e295   alpine        "sh"                      10 days ago   Exited (255) 9 days ago                                                c4
f977da1038da   alpine        "sh"                      10 days ago   Exited (255) 9 days ago                                                c2
908b0a0a52f9   alpine        "sh"                      10 days ago   Exited (255) 9 days ago                                                c1
2fc4259cc235   alpine        "/bin/sh"                 10 days ago   Exited (0) 10 days ago                                                 vigorous_heisenberg
4d91babf620b   hello-world   "/hello"                  10 days ago   Exited (0) 10 days ago                                                 jovial_herschel
6143e19519ad   hello-world   "/hello"                  10 days ago   Exited (0) 10 days ago                                                 keen_galois
tasneem@DESKTOP-0VT5601:~$ sudo docker run -d -p 3000:3000 ty_app:single
6126d9b883d42f374ebf2cf28829ad1bfa0b14bafa716176fedda93fedb61b7c
tasneem@DESKTOP-0VT5601:~$ curl http://localhost:3000/health
{"ok":true,"service":"TS API"}tasneem@DESKTOP-0VT5601:~$
```



```
ty_app        multilayerv3   020a2bc68125   12 hours ago   135MB
ty_app        multilayer     e4ab52fddb1c   13 hours ago   137MB
ty_app        multilayerv1   e4ab52fddb1c   13 hours ago   137MB
ty_app        multilayerv2   e4ab52fddb1c   13 hours ago   137MB
ty_app        single         07cc6b054624   14 hours ago   1.18GB
nginx_custom  latest         f49c6ad5af4a   3 days ago     192MB
<none>        <none>         138d533c8416   3 days ago     192MB
<none>        <none>         9b2ce42155bd   3 days ago     192MB
```

Part B — Multi-stage, optimized & secure

Create **`Dockerfile`** (the default) that:

1. **Builder stage**

  - uses `node:20`

  - installs *all* deps (including dev)

  - builds TypeScript (`npm run build`)

  - removes dev deps: `npm prune --omit=dev`

2. **Runtime stage**

  - uses `node:20-alpine`

  - copies only required files

  - **runs as non-root** user

  - add a `HEALTHCHECK` hitting `http://127.0.0.1:3000/health`

4. **No secrets** baked into image

5. **No npm** *required* at runtime (direct `node` entrypoint)

Targets to hit

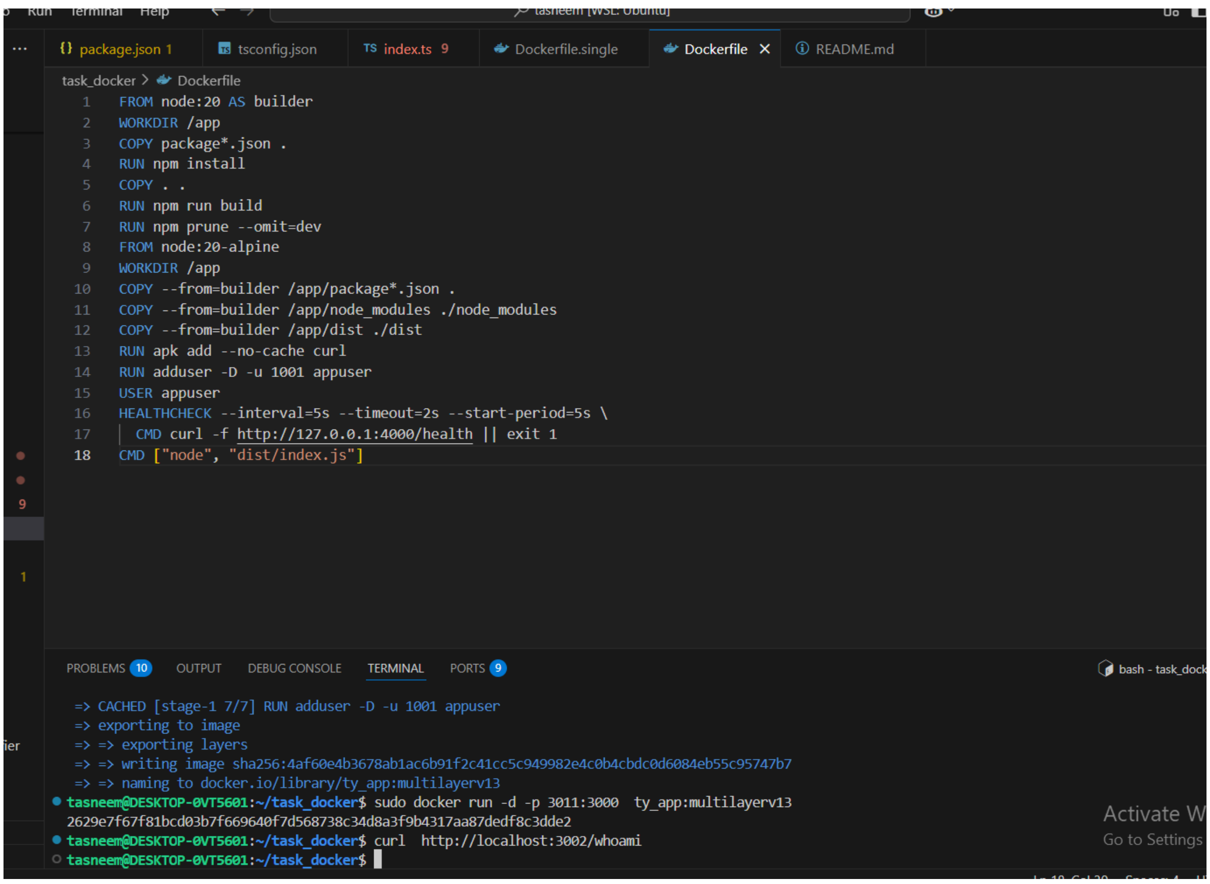
- Final runtime image size **≤ 140 MB**

- `/whoami` endpoint should show **non-root** uid/gid (neither 0 nor null)

CMD:

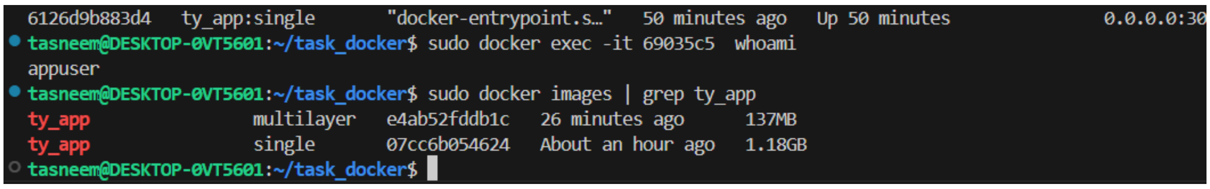
```
sudo docker build -f Dockerfile -t ty_app:multilayer  .

sudo docker run -d -p 3000:3000  ty_app:multilayer
```

curl http://localhost:3005/whoami



```
task_docker > Dockerfile
FROM node:20 AS builder
WORKDIR /app
COPY package*.json .
RUN npm install
COPY . .
RUN npm run build
RUN npm prune --omit=dev
FROM node:20-alpine
WORKDIR /app
COPY --from=builder /app/package*.json .
COPY --from=builder /app/node_modules ./node_modules
COPY --from=builder /app/dist ./dist
RUN apk add --no-cache curl
RUN adduser -D -u 1001 appuser
USER appuser
HEALTHCHECK --interval=5s --timeout=2s --start-period=5s \
  CMD curl -f http://127.0.0.1:4000/health || exit 1
CMD ["node", "dist/index.js"]
```

```
 => CACHED [stage-1 7/7] RUN adduser -D -u 1001 appuser
 => exporting to image
 => => exporting layers
 => => writing image sha256:4af60e4b3678ab1ac6b91f2c41cc5c949982e4c0b4cbdc0d6084eb55c95747b7
 => => naming to docker.io/library/ty_app:multilayerv13
tasneem@DESKTOP-0VT5601:~/task_docker$ sudo docker run -d -p 3011:3000  ty_app:multilayerv13
2629e7f67f81bcd03b7f669640f7d568738c34d8a3f9b4317aa87dedf8c3dde2
tasneem@DESKTOP-0VT5601:~/task_docker$ curl  http://localhost:3002/whoami
tasneem@DESKTOP-0VT5601:~/task_docker$ 
```



```
6126d9b883d4   ty_app:single       "docker-entrypoint.s…"   50 minutes ago   Up 50 minutes              0.0.0.0:30
tasneem@DESKTOP-0VT5601:~/task_docker$ sudo docker exec -it 69035c5  whoami
appuser
tasneem@DESKTOP-0VT5601:~/task_docker$ sudo docker images | grep ty_app
ty_app                   multilayer    e4ab52fddb1c   26 minutes ago      137MB
ty_app                   single        07cc6b054624   About an hour ago   1.18GB
tasneem@DESKTOP-0VT5601:~/task_docker$ 
```

Part C — Tricky “what ifs” to catch them out

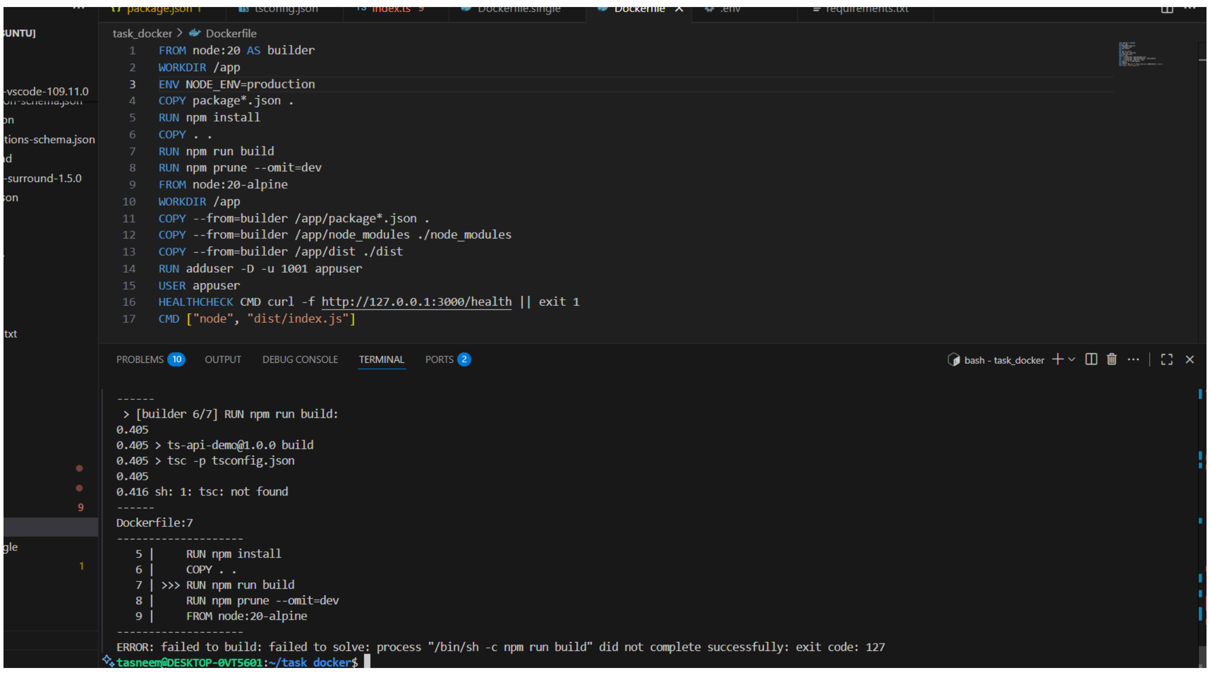
1. **NODE_ENV trap**

  - If set in builder before installing, build fails (no TypeScript).

CMD:

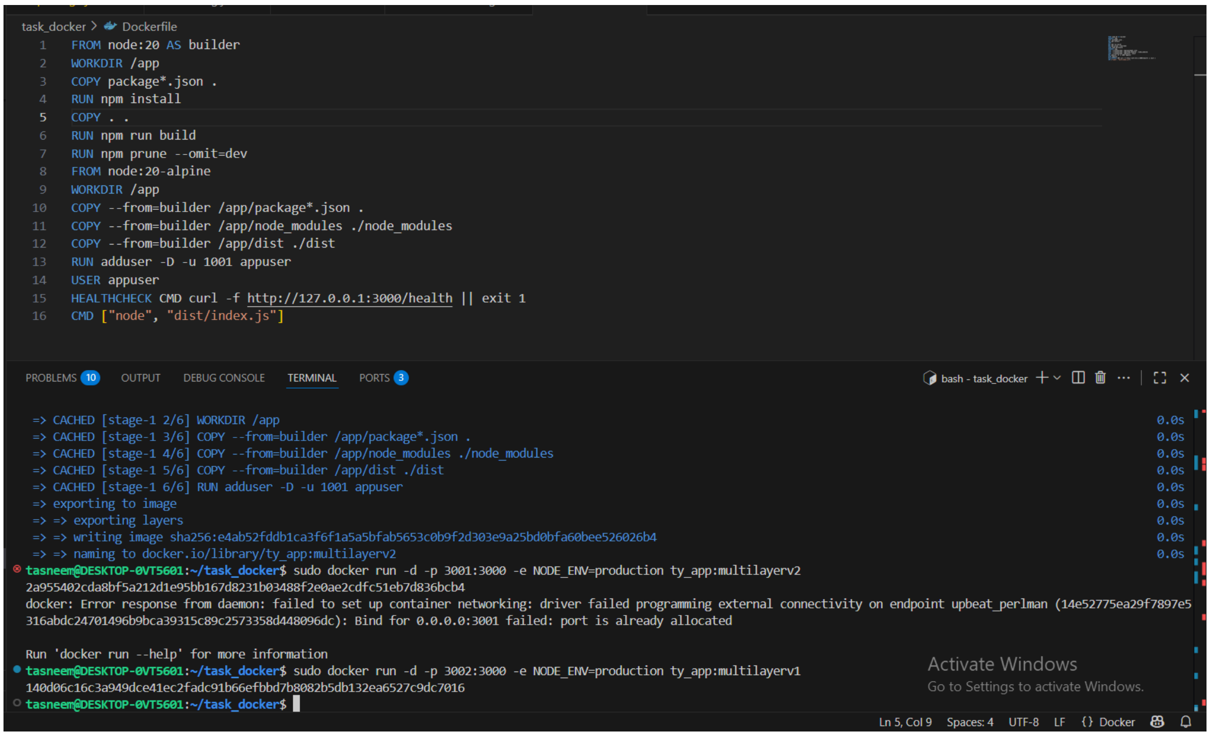
sudo docker build -f Dockerfile -t ty_app:multilayerv1 .

Solution:
Keep NODE_ENV unset or development in the builder. You can pass it at runtime instead



```
task_docker > Dockerfile
FROM node:20 AS builder
WORKDIR /app
ENV NODE_ENV=production
COPY package*.json .
RUN npm install
COPY . .
RUN npm run build
RUN npm prune --omit=dev
FROM node:20-alpine
WORKDIR /app
COPY --from=builder /app/package*.json .
COPY --from=builder /app/node_modules ./node_modules
COPY --from=builder /app/dist ./dist
RUN adduser -D -u 1001 appuser
USER appuser
HEALTHCHECK CMD curl -f http://127.0.0.1:3000/health || exit 1
CMD ["node", "dist/index.js"]
```

```
------
 > [builder 6/7] RUN npm run build:
0.405
0.405 > ts-api-demo@1.0.0 build
0.405 > tsc -p tsconfig.json
0.405
0.416 sh: 1: tsc: not found
------
Dockerfile:7
--------------------
   5 |     RUN npm install
   6 |     COPY . .
   7 | >>> RUN npm run build
   8 |     RUN npm prune --omit=dev
   9 |     FROM node:20-alpine
--------------------
ERROR: failed to build: failed to solve: process "/bin/sh -c npm run build" did not complete successfully: exit code: 127
tasneem@DESKTOP-0VT5601:~/task docker$
```



```
task_docker > Dockerfile
FROM node:20 AS builder
WORKDIR /app
COPY package*.json .
RUN npm install
COPY . .
RUN npm run build
RUN npm prune --omit=dev
FROM node:20-alpine
WORKDIR /app
COPY --from=builder /app/package*.json .
COPY --from=builder /app/node_modules ./node_modules
COPY --from=builder /app/dist ./dist
RUN adduser -D -u 1001 appuser
USER appuser
HEALTHCHECK CMD curl -f http://127.0.0.1:3000/health || exit 1
CMD ["node", "dist/index.js"]
```

```
 => CACHED [stage-1 2/6] WORKDIR /app                                          0.0s
 => CACHED [stage-1 3/6] COPY --from=builder /app/package*.json .              0.0s
 => CACHED [stage-1 4/6] COPY --from=builder /app/node_modules ./node_modules  0.0s
 => CACHED [stage-1 5/6] COPY --from=builder /app/dist ./dist                  0.0s
 => CACHED [stage-1 6/6] RUN adduser -D -u 1001 appuser                        0.0s
 => exporting to image                                                         0.0s
 => => exporting layers                                                        0.0s
 => => writing image sha256:e4ab52fddb1ca3f6f1a5a5bfab5653c0b9f2d303e9a25bd0bfa60bee526026b4  0.0s
 => => naming to docker.io/library/ty_app:multilayerv2                         0.0s
tasneem@DESKTOP-0VT5601:~/task_docker$ sudo docker run -d -p 3001:3000 -e NODE_ENV=production ty_app:multilayerv2
2a955402cda8bf5a212d1e95bb167d8231b03488f2e0ae2cdfc51eb7d836bcb4
docker: Error response from daemon: failed to set up container networking: driver failed programming external connectivity on endpoint upbeat_perlman (14e52775ea29f7897e5
316abdc24701496b9bca39315c89c2573358d448096dc): Bind for 0.0.0.0:3001 failed: port is already allocated

Run 'docker run --help' for more information
tasneem@DESKTOP-0VT5601:~/task_docker$ sudo docker run -d -p 3002:3000 -e NODE_ENV=production ty_app:multilayerv1
140d06c16c3a949dce41ec2fadc91b66efbbd7b8082b5db132ea6527c9dc7016
tasneem@DESKTOP-0VT5601:~/task_docker$
```

2. **Express in devDependencies trap**

- If `express` mistakenly in devDeps → runtime crash.

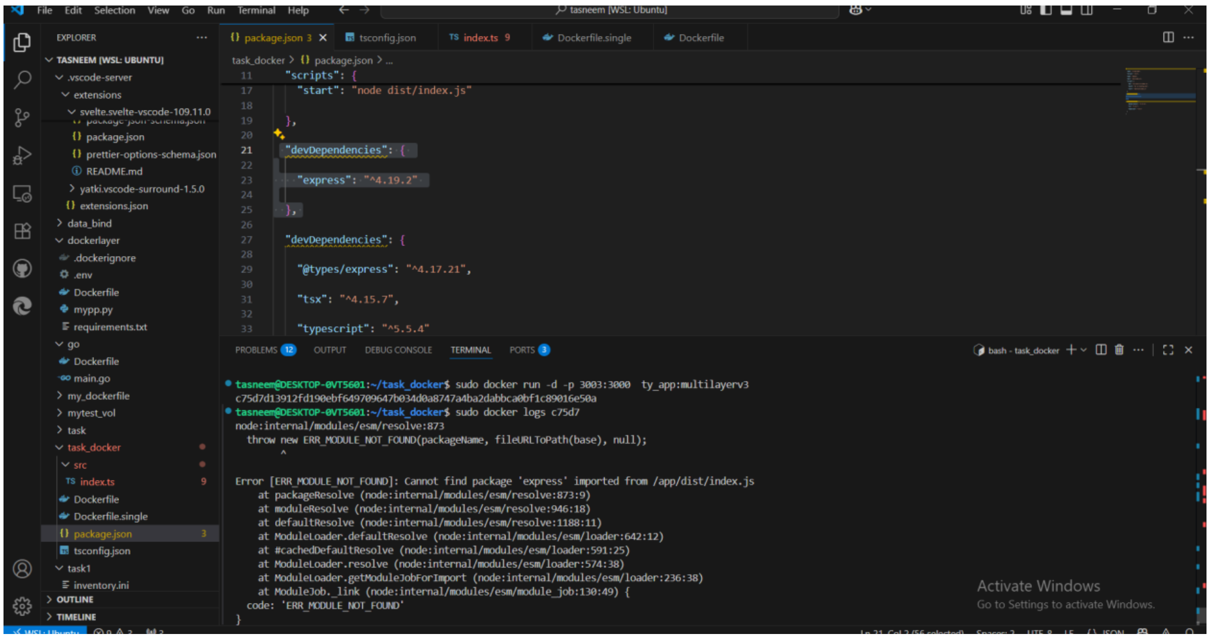
CMD:

sudo docker build -f Dockerfile -t ty_app:multilayerv15 .
sudo docker run -d -p 3000:3000 ty_app:multilayerv15

sudo docker logs c75d7

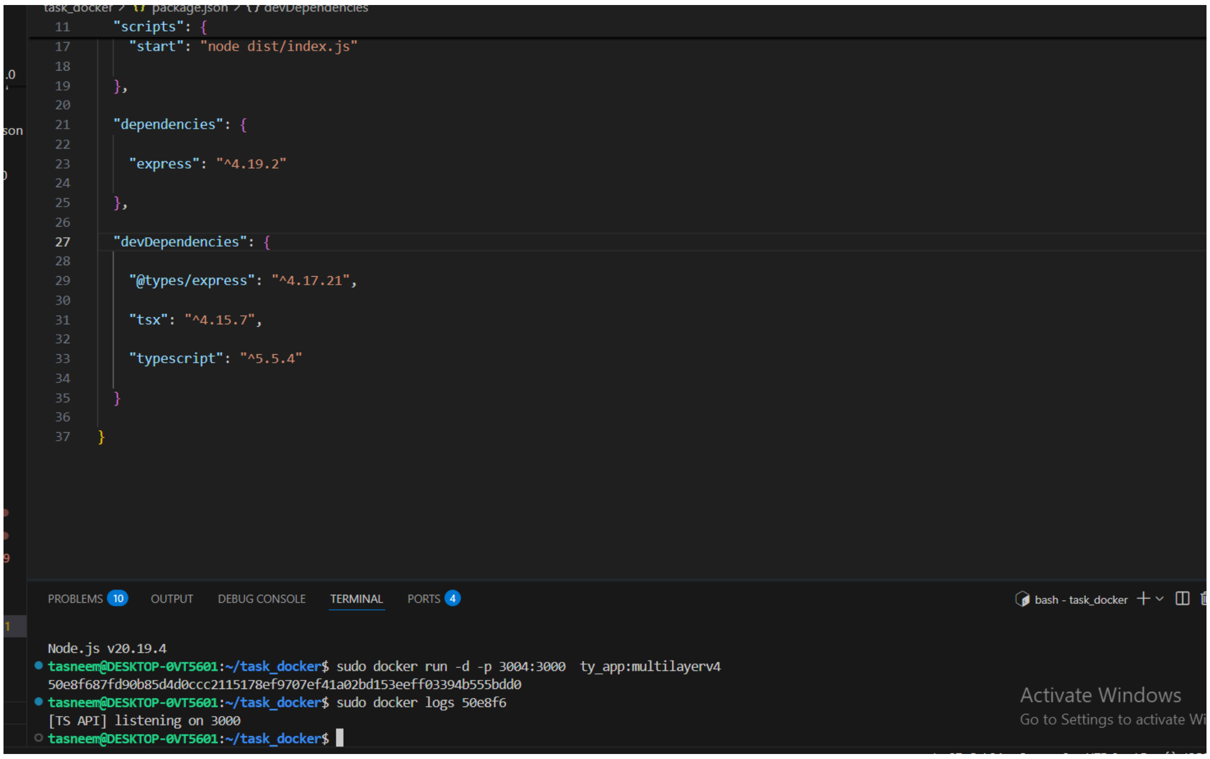
Solution:

All libraries required for running the app must be in dependencies



```
"scripts": {
  "start": "node dist/index.js"

},

"devDependencies": {

  "express": "^4.19.2"

},

"devDependencies": {

  "@types/express": "^4.17.21",

  "tsx": "^4.15.7",

  "typescript": "^5.5.4"
```

```
tasneem@DESKTOP-0VT5601:~/task_docker$ sudo docker run -d -p 3003:3000  ty_app:multilayerv3
c75d7d13912fd190ebf649709647b034d0a8747a4ba2dabbca0bf1c89016e50a
tasneem@DESKTOP-0VT5601:~/task_docker$ sudo docker logs c75d7
node:internal/modules/esm/resolve:873
  throw new ERR_MODULE_NOT_FOUND(packageName, fileURLToPath(base), null);
        ^

Error [ERR_MODULE_NOT_FOUND]: Cannot find package 'express' imported from /app/dist/index.js
    at packageResolve (node:internal/modules/esm/resolve:873:9)
    at moduleResolve (node:internal/modules/esm/resolve:946:18)
    at defaultResolve (node:internal/modules/esm/resolve:1188:11)
    at ModuleLoader.defaultResolve (node:internal/modules/esm/loader:642:12)
    at #cachedDefaultResolve (node:internal/modules/esm/loader:591:25)
    at ModuleLoader.resolve (node:internal/modules/esm/loader:574:38)
    at ModuleLoader.getModuleJobForImport (node:internal/modules/esm/loader:236:38)
    at ModuleJob._link (node:internal/modules/esm/module_job:130:49) {
  code: 'ERR_MODULE_NOT_FOUND'
}
```



```
"scripts": {
  "start": "node dist/index.js"

},

"dependencies": {

  "express": "^4.19.2"

},

"devDependencies": {

  "@types/express": "^4.17.21",

  "tsx": "^4.15.7",

  "typescript": "^5.5.4"

}
}
```

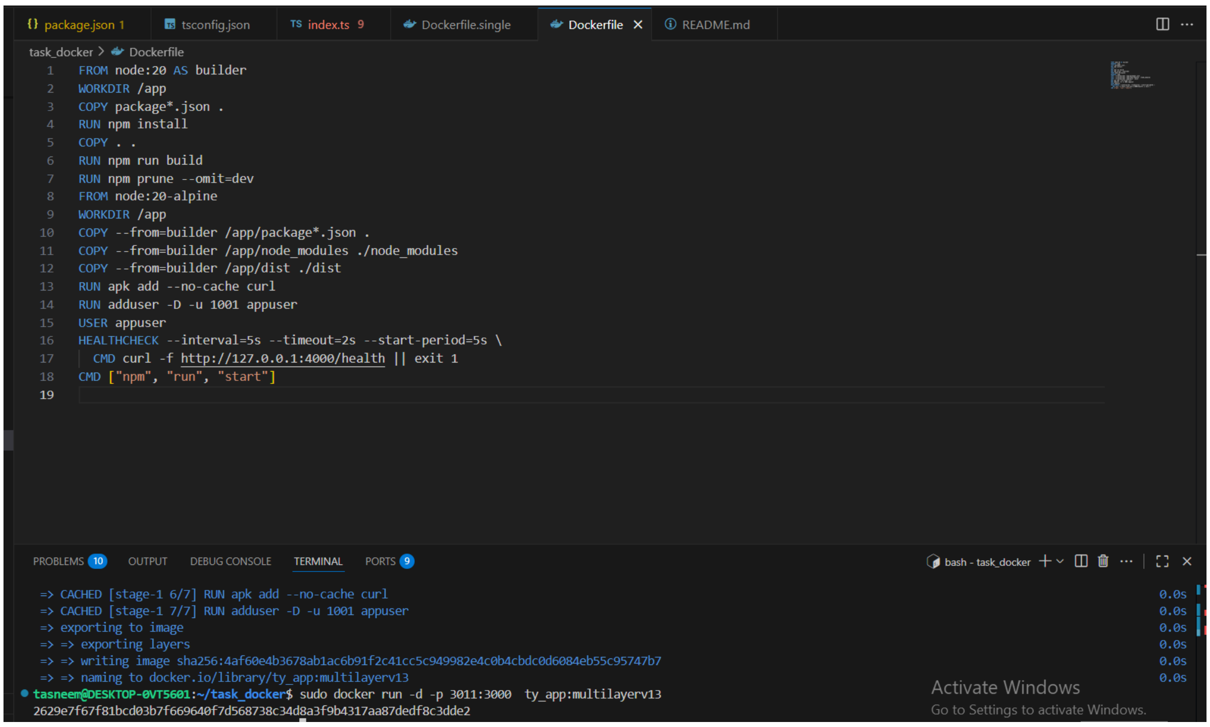
```
Node.js v20.19.4
tasneem@DESKTOP-0VT5601:~/task_docker$ sudo docker run -d -p 3004:3000  ty_app:multilayerv4
50e8f687fd90b85d4d0ccc2115178ef9707ef41a02bd153eeff03394b555bdd0
tasneem@DESKTOP-0VT5601:~/task_docker$ sudo docker logs 50e8f6
[TS API] listening on 3000
tasneem@DESKTOP-0VT5601:~/task_docker$
```

3. **Wrong CMD**

 - Using `npm run start` in runtime adds bloat. Must be direct `node`.

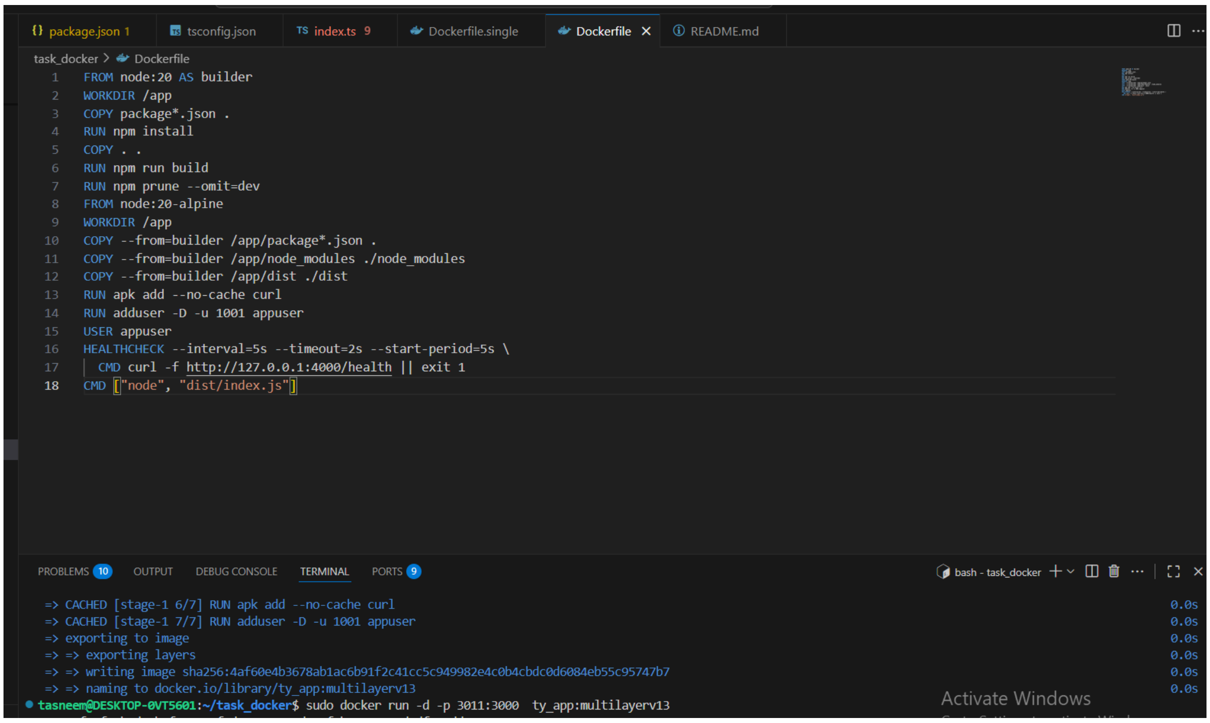
requires npm in the image → increases image size and bloat.

Solution: Use direct Node command



```
FROM node:20 AS builder
WORKDIR /app
COPY package*.json .
RUN npm install
COPY . .
RUN npm run build
RUN npm prune --omit=dev
FROM node:20-alpine
WORKDIR /app
COPY --from=builder /app/package*.json .
COPY --from=builder /app/node_modules ./node_modules
COPY --from=builder /app/dist ./dist
RUN apk add --no-cache curl
RUN adduser -D -u 1001 appuser
USER appuser
HEALTHCHECK --interval=5s --timeout=2s --start-period=5s \
  CMD curl -f http://127.0.0.1:4000/health || exit 1
CMD ["npm", "run", "start"]
```

```
 => CACHED [stage-1 6/7] RUN apk add --no-cache curl                                           0.0s
 => CACHED [stage-1 7/7] RUN adduser -D -u 1001 appuser                                        0.0s
 => exporting to image                                                                         0.0s
 => => exporting layers                                                                        0.0s
 => => writing image sha256:4af60e4b3678ab1ac6b91f2c41cc5c949982e4c0b4cbdc0d6084eb55c95747b7   0.0s
 => => naming to docker.io/library/ty_app:multilayerv13                                        0.0s
tasneem@DESKTOP-0VT5601:~/task_docker$ sudo docker run -d -p 3011:3000  ty_app:multilayerv13
2629e7f67f81bcd03b7f669640f7d568738c34d8a3f9b4317aa87dedf8c3dde2
```



```
FROM node:20 AS builder
WORKDIR /app
COPY package*.json .
RUN npm install
COPY . .
RUN npm run build
RUN npm prune --omit=dev
FROM node:20-alpine
WORKDIR /app
COPY --from=builder /app/package*.json .
COPY --from=builder /app/node_modules ./node_modules
COPY --from=builder /app/dist ./dist
RUN apk add --no-cache curl
RUN adduser -D -u 1001 appuser
USER appuser
HEALTHCHECK --interval=5s --timeout=2s --start-period=5s \
  CMD curl -f http://127.0.0.1:4000/health || exit 1
CMD ["node", "dist/index.js"]
```

```
 => CACHED [stage-1 6/7] RUN apk add --no-cache curl                                           0.0s
 => CACHED [stage-1 7/7] RUN adduser -D -u 1001 appuser                                        0.0s
 => exporting to image                                                                         0.0s
 => => exporting layers                                                                        0.0s
 => => writing image sha256:4af60e4b3678ab1ac6b91f2c41cc5c949982e4c0b4cbdc0d6084eb55c95747b7   0.0s
 => => naming to docker.io/library/ty_app:multilayerv13                                        0.0s
tasneem@DESKTOP-0VT5601:~/task_docker$ sudo docker run -d -p 3011:3000  ty_app:multilayerv13
```

4. **Root user**

  - `/whoami` returns `uid:0` → fail.

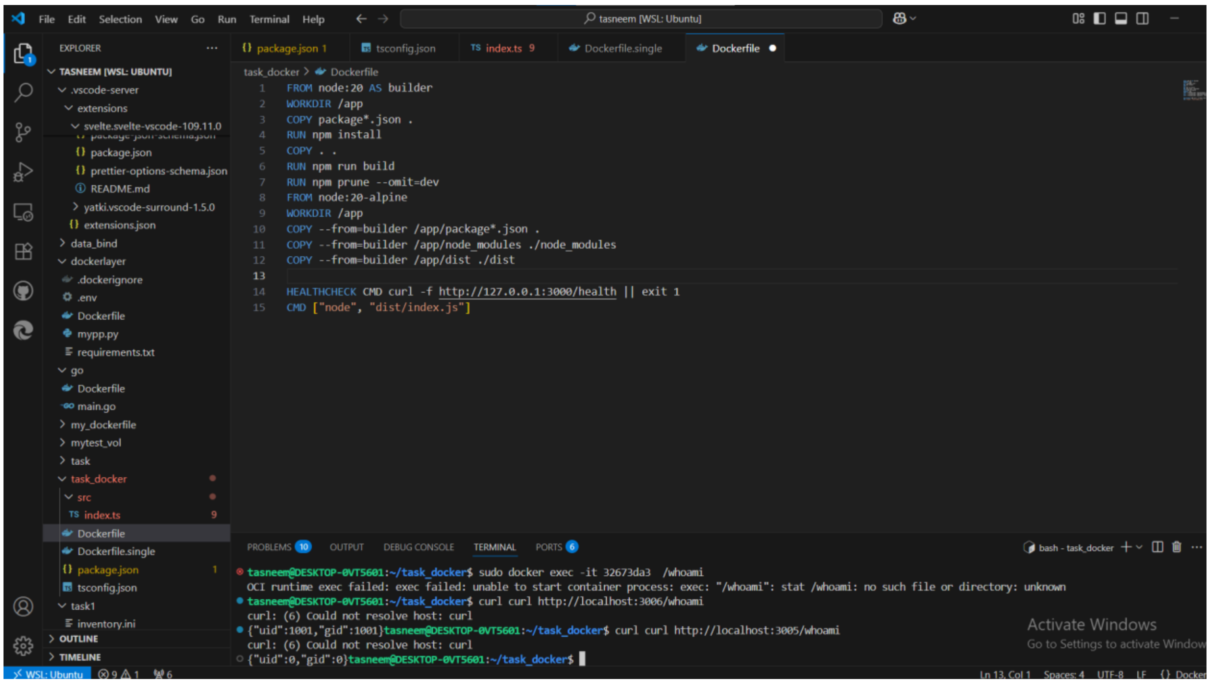
CMD:

sudo docker build -f Dockerfile -t ty_app:multilayerv12 .

sudo docker run -d -p 3000:3000 ty_app:multilayerv12
curl curl http://localhost:3005/whoami

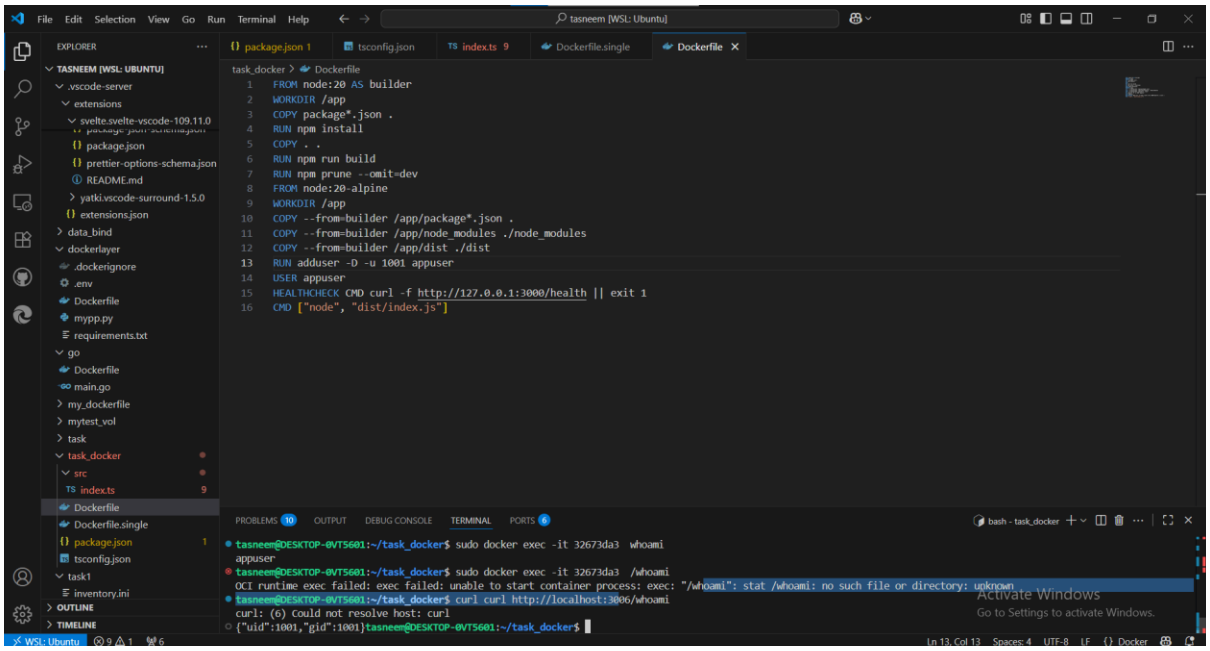
If you don't create a non-root user → /whoami returns uid:0 → fail.

Solution: Create a non-root user



```
FROM node:20 AS builder
WORKDIR /app
COPY package*.json .
RUN npm install
COPY . .
RUN npm run build
RUN npm prune --omit=dev
FROM node:20-alpine
WORKDIR /app
COPY --from=builder /app/package*.json .
COPY --from=builder /app/node_modules ./node_modules
COPY --from=builder /app/dist ./dist

HEALTHCHECK CMD curl -f http://127.0.0.1:3000/health || exit 1
CMD ["node", "dist/index.js"]
```

```
tasneem@DESKTOP-0VT5601:~/task_docker$ sudo docker exec -it 32673da3  /whoami
OCI runtime exec failed: exec failed: unable to start container process: exec: "/whoami": stat /whoami: no such file or directory: unknown
tasneem@DESKTOP-0VT5601:~/task_docker$ curl curl http://localhost:3006/whoami
curl: (6) Could not resolve host: curl
{"uid":1001,"gid":1001}tasneem@DESKTOP-0VT5601:~/task_docker$ curl curl http://localhost:3005/whoami
curl: (6) Could not resolve host: curl
{"uid":0,"gid":0}tasneem@DESKTOP-0VT5601:~/task_docker$
```



```
FROM node:20 AS builder
WORKDIR /app
COPY package*.json .
RUN npm install
COPY . .
RUN npm run build
RUN npm prune --omit=dev
FROM node:20-alpine
WORKDIR /app
COPY --from=builder /app/package*.json .
COPY --from=builder /app/node_modules ./node_modules
COPY --from=builder /app/dist ./dist
RUN adduser -D -u 1001 appuser
USER appuser
HEALTHCHECK CMD curl -f http://127.0.0.1:3000/health || exit 1
CMD ["node", "dist/index.js"]
```

```
tasneem@DESKTOP-0VT5601:~/task_docker$ sudo docker exec -it 32673da3  whoami
appuser
tasneem@DESKTOP-0VT5601:~/task_docker$ sudo docker exec -it 32673da3  /whoami
OCI runtime exec failed: exec failed: unable to start container process: exec: "/whoami": stat /whoami: no such file or directory: unknown
tasneem@DESKTOP-0VT5601:~/task_docker$ curl curl http://localhost:3006/whoami
curl: (6) Could not resolve host: curl
{"uid":1001,"gid":1001}tasneem@DESKTOP-0VT5601:~/task_docker$
```

5. **Cache misuse**

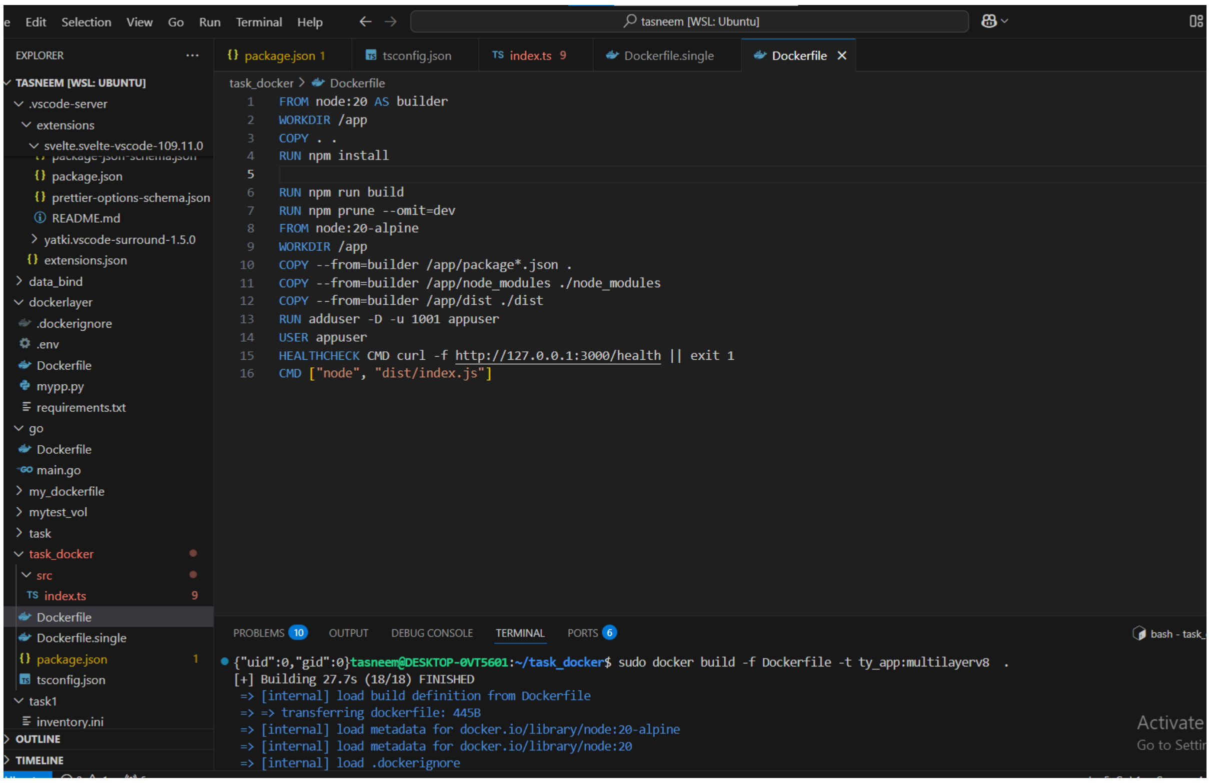
 - Copying code before `npm install` → breaks cache.

CMD:

```
sudo docker build -f Dockerfile -t ty_app:multilayerv12  .
```

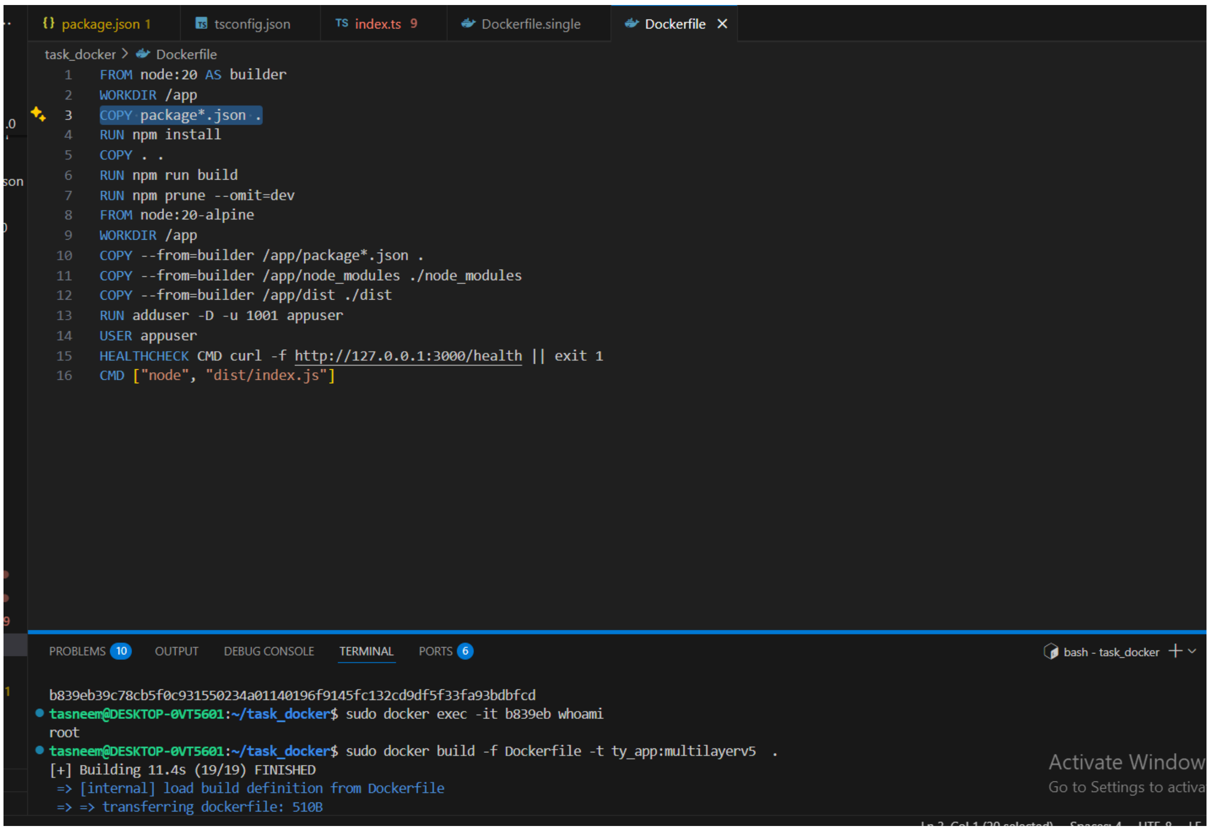
npm reinstall runs every build → slow.

Solution: Copy only manifests first



```
FROM node:20 AS builder
WORKDIR /app
COPY . .
RUN npm install

RUN npm run build
RUN npm prune --omit=dev
FROM node:20-alpine
WORKDIR /app
COPY --from=builder /app/package*.json .
COPY --from=builder /app/node_modules ./node_modules
COPY --from=builder /app/dist ./dist
RUN adduser -D -u 1001 appuser
USER appuser
HEALTHCHECK CMD curl -f http://127.0.0.1:3000/health || exit 1
CMD ["node", "dist/index.js"]
```

```
{"uid":0,"gid":0}tasneem@DESKTOP-0VT5601:~/task_docker$ sudo docker build -f Dockerfile -t ty_app:multilayerv8  .
[+] Building 27.7s (18/18) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 445B
 => [internal] load metadata for docker.io/library/node:20-alpine
 => [internal] load metadata for docker.io/library/node:20
 => [internal] load .dockerignore
```



```
FROM node:20 AS builder
WORKDIR /app
COPY package*.json .
RUN npm install
COPY . .
RUN npm run build
RUN npm prune --omit=dev
FROM node:20-alpine
WORKDIR /app
COPY --from=builder /app/package*.json .
COPY --from=builder /app/node_modules ./node_modules
COPY --from=builder /app/dist ./dist
RUN adduser -D -u 1001 appuser
USER appuser
HEALTHCHECK CMD curl -f http://127.0.0.1:3000/health || exit 1
CMD ["node", "dist/index.js"]
```

```
b839eb39c78cb5f0c931550234a01140196f9145fc132cd9df5f33fa93bdbfcd
tasneem@DESKTOP-0VT5601:~/task_docker$ sudo docker exec -it b839eb whoami
root
tasneem@DESKTOP-0VT5601:~/task_docker$ sudo docker build -f Dockerfile -t ty_app:multilayerv5  .
[+] Building 11.4s (19/19) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 510B
```

6. **Healthcheck**

- Wrong command/host → container unhealthy.

HEALTHCHECK CMD curl -f http://127.0.0.1:4000/health || exit 1

If the app runs on port 3000 → container always unhealthy.

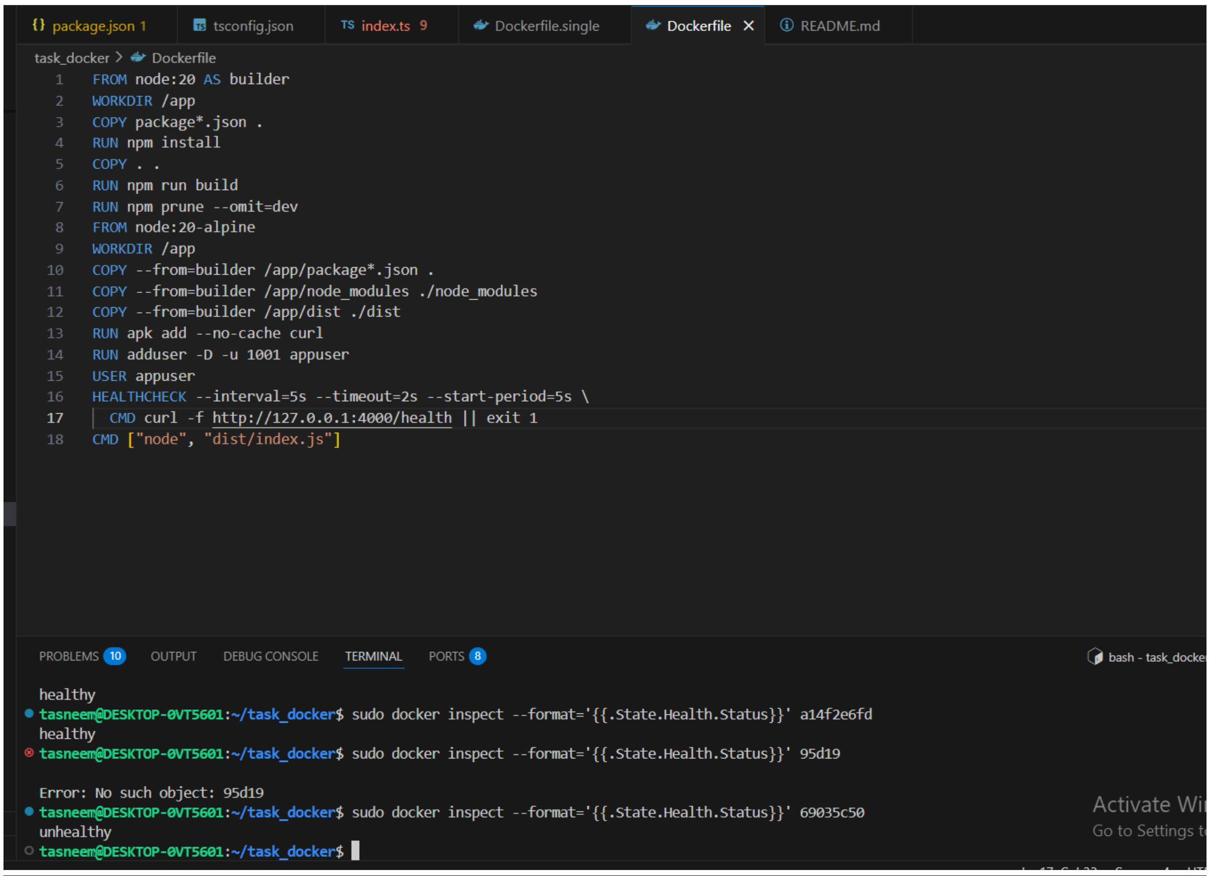
CMD:

sudo docker build -f Dockerfile -t ty_app:multilayerv15 .

sudo docker run -d -p 3000:3000 ty_app:multilayerv12

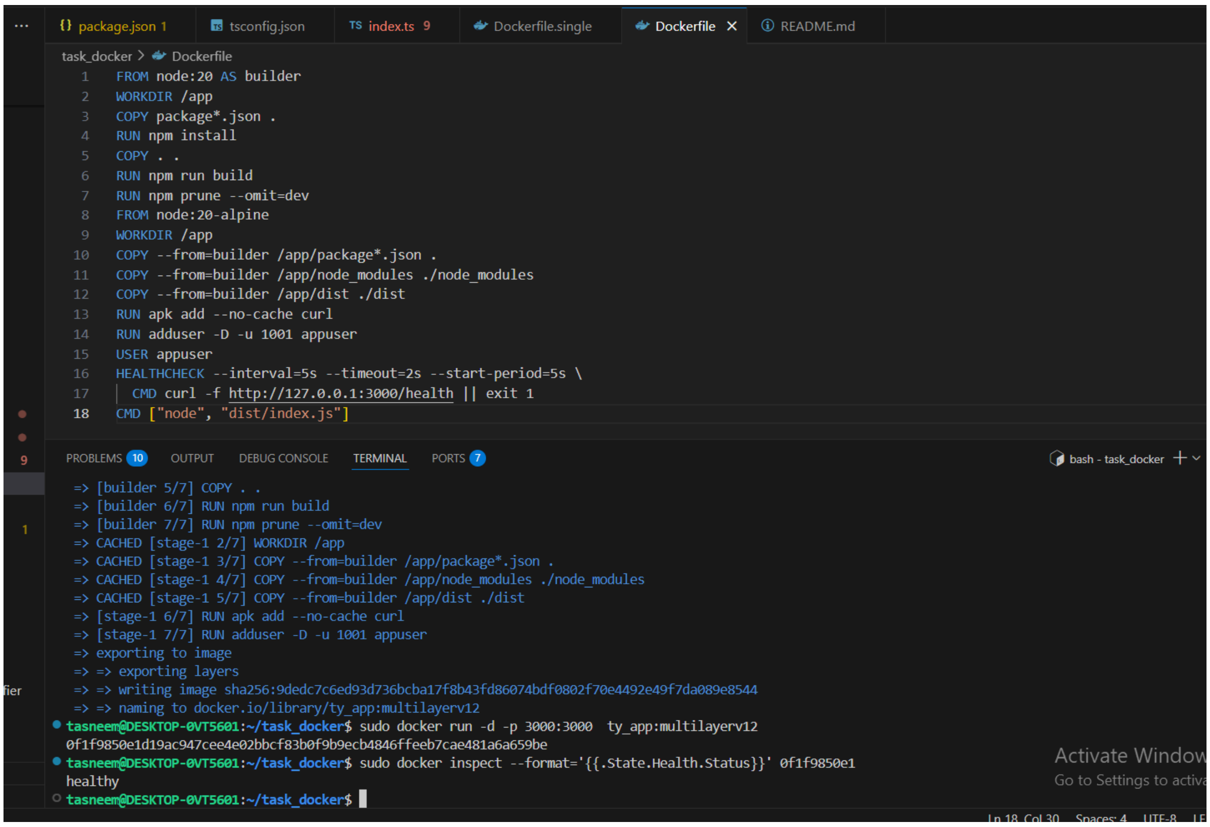
sudo docker inspect --format='{{.State.Health.Status}}' 69035c50

Solution: Correct port and endpoint



```
FROM node:20 AS builder
WORKDIR /app
COPY package*.json .
RUN npm install
COPY . .
RUN npm run build
RUN npm prune --omit=dev
FROM node:20-alpine
WORKDIR /app
COPY --from=builder /app/package*.json .
COPY --from=builder /app/node_modules ./node_modules
COPY --from=builder /app/dist ./dist
RUN apk add --no-cache curl
RUN adduser -D -u 1001 appuser
USER appuser
HEALTHCHECK --interval=5s --timeout=2s --start-period=5s \
  CMD curl -f http://127.0.0.1:4000/health || exit 1
CMD ["node", "dist/index.js"]
```

```
healthy
tasneem@DESKTOP-0VT5601:~/task_docker$ sudo docker inspect --format='{{.State.Health.Status}}' a14f2e6fd
healthy
tasneem@DESKTOP-0VT5601:~/task_docker$ sudo docker inspect --format='{{.State.Health.Status}}' 95d19

Error: No such object: 95d19
tasneem@DESKTOP-0VT5601:~/task_docker$ sudo docker inspect --format='{{.State.Health.Status}}' 69035c50
unhealthy
tasneem@DESKTOP-0VT5601:~/task_docker$
```



```
FROM node:20 AS builder
WORKDIR /app
COPY package*.json .
RUN npm install
COPY . .
RUN npm run build
RUN npm prune --omit=dev
FROM node:20-alpine
WORKDIR /app
COPY --from=builder /app/package*.json .
COPY --from=builder /app/node_modules ./node_modules
COPY --from=builder /app/dist ./dist
RUN apk add --no-cache curl
RUN adduser -D -u 1001 appuser
USER appuser
HEALTHCHECK --interval=5s --timeout=2s --start-period=5s \
  CMD curl -f http://127.0.0.1:3000/health || exit 1
CMD ["node", "dist/index.js"]
```

```
=> [builder 5/7] COPY . .
=> [builder 6/7] RUN npm run build
=> [builder 7/7] RUN npm prune --omit=dev
=> CACHED [stage-1 2/7] WORKDIR /app
=> CACHED [stage-1 3/7] COPY --from=builder /app/package*.json .
=> CACHED [stage-1 4/7] COPY --from=builder /app/node_modules ./node_modules
=> CACHED [stage-1 5/7] COPY --from=builder /app/dist ./dist
=> [stage-1 6/7] RUN apk add --no-cache curl
=> [stage-1 7/7] RUN adduser -D -u 1001 appuser
=> exporting to image
=> => exporting layers
=> => writing image sha256:9dedc7c6ed93d736bcba17f8b43fd86074bdf0802f70e4492e49f7da089e8544
=> => naming to docker.io/library/ty_app:multilayerv12
tasneem@DESKTOP-0VT5601:~/task_docker$ sudo docker run -d -p 3000:3000  ty_app:multilayerv12
0f1f9850e1d19ac947cee4e02bbcf83b0f9b9ecb4846ffeeb7cae481a6a659be
tasneem@DESKTOP-0VT5601:~/task_docker$ sudo docker inspect --format='{{.State.Health.Status}}' 0f1f9850e1
healthy
tasneem@DESKTOP-0VT5601:~/task_docker$
```

Deliverables (students submit)

- `Dockerfile.single`

- `Optmized Dockerfile`

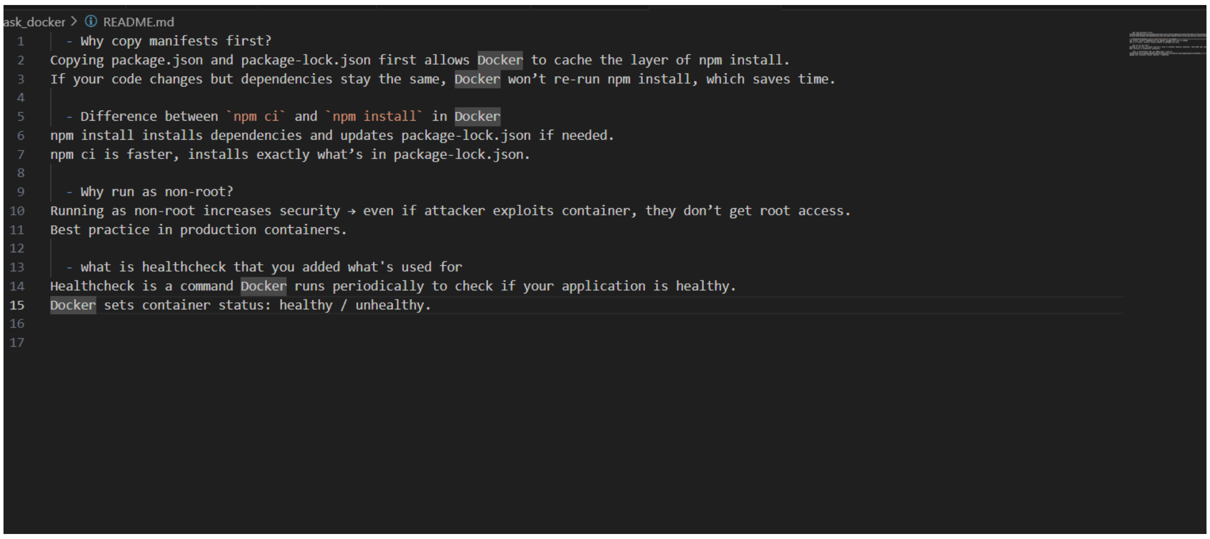
- **README.md** answering:

  - Why copy manifests first?

  - Difference between `npm ci` and `npm install` in Docker

  - Why run as non-root?

  - what is healthcheck that you added what's used for



ask_docker > README.md

```
  - Why copy manifests first?
Copying package.json and package-lock.json first allows Docker to cache the layer of npm install.
If your code changes but dependencies stay the same, Docker won't re-run npm install, which saves time.

  - Difference between `npm ci` and `npm install` in Docker
npm install installs dependencies and updates package-lock.json if needed.
npm ci is faster, installs exactly what's in package-lock.json.

  - Why run as non-root?
Running as non-root increases security → even if attacker exploits container, they don't get root access.
Best practice in production containers.

  - what is healthcheck that you added what's used for
Healthcheck is a command Docker runs periodically to check if your application is healthy.
Docker sets container status: healthy / unhealthy.
```

- The exact **build and run** commands used
