Creating an application first using node js

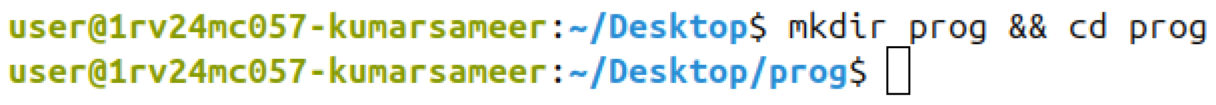
Check the version of node

node -v

npm -v

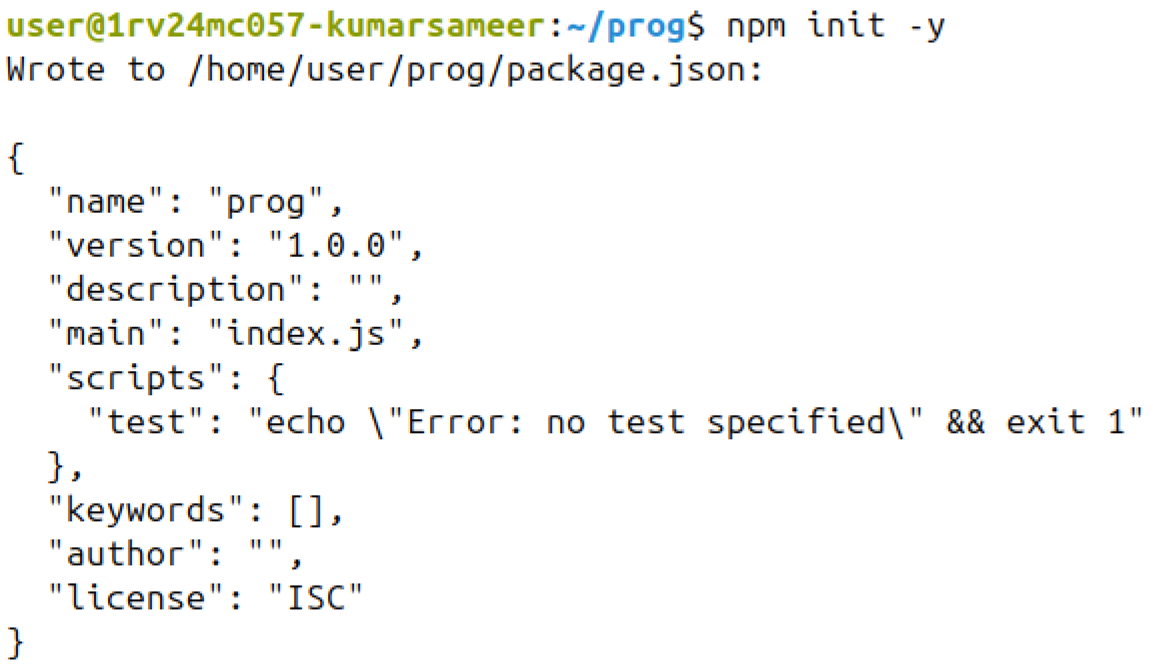
Project Setup

Step 1: Create a new folder for your app and move inside it



```
user@1rv24mc057-kumarsameer:~/Desktop$ mkdir prog && cd prog
user@1rv24mc057-kumarsameer:~/Desktop/prog$ 
```

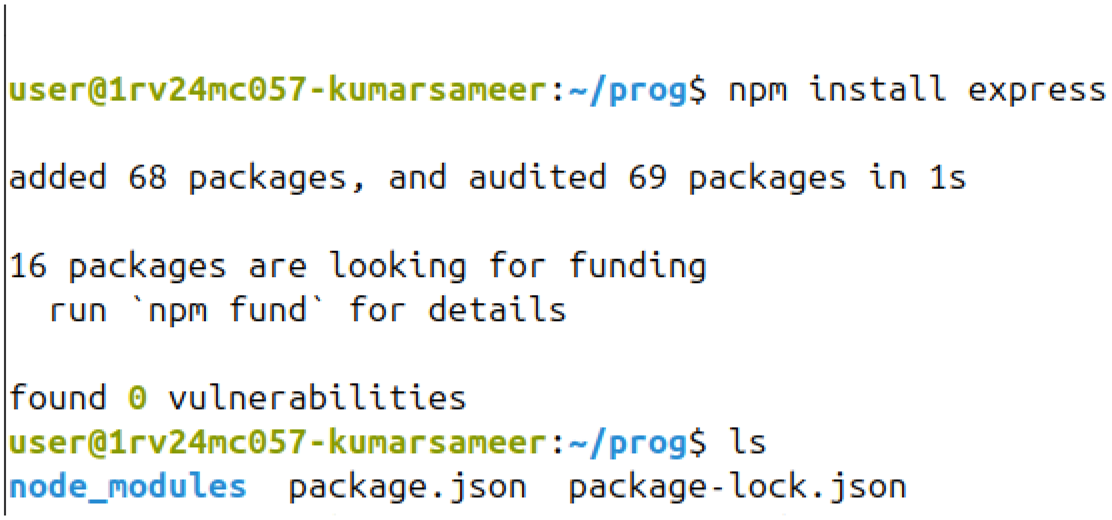
Step 2: Initialize a Node.js project



```
user@1rv24mc057-kumarsameer:~/prog$ npm init -y
Wrote to /home/user/prog/package.json:

{
  "name": "prog",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

This command generates a default package.json file which stores your project's metadata and dependencies.

Step 3: Install Express



```
user@1rv24mc057-kumarsameer:~/prog$ npm install express

added 68 packages, and audited 69 packages in 1s

16 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
user@1rv24mc057-kumarsameer:~/prog$ ls
node_modules  package.json  package-lock.json
```

Express is a lightweight web framework for Node.js — it helps to easily handle web requests and responses.

After installing, we can see a node_modules/ folder and package-lock.json file created automatically.

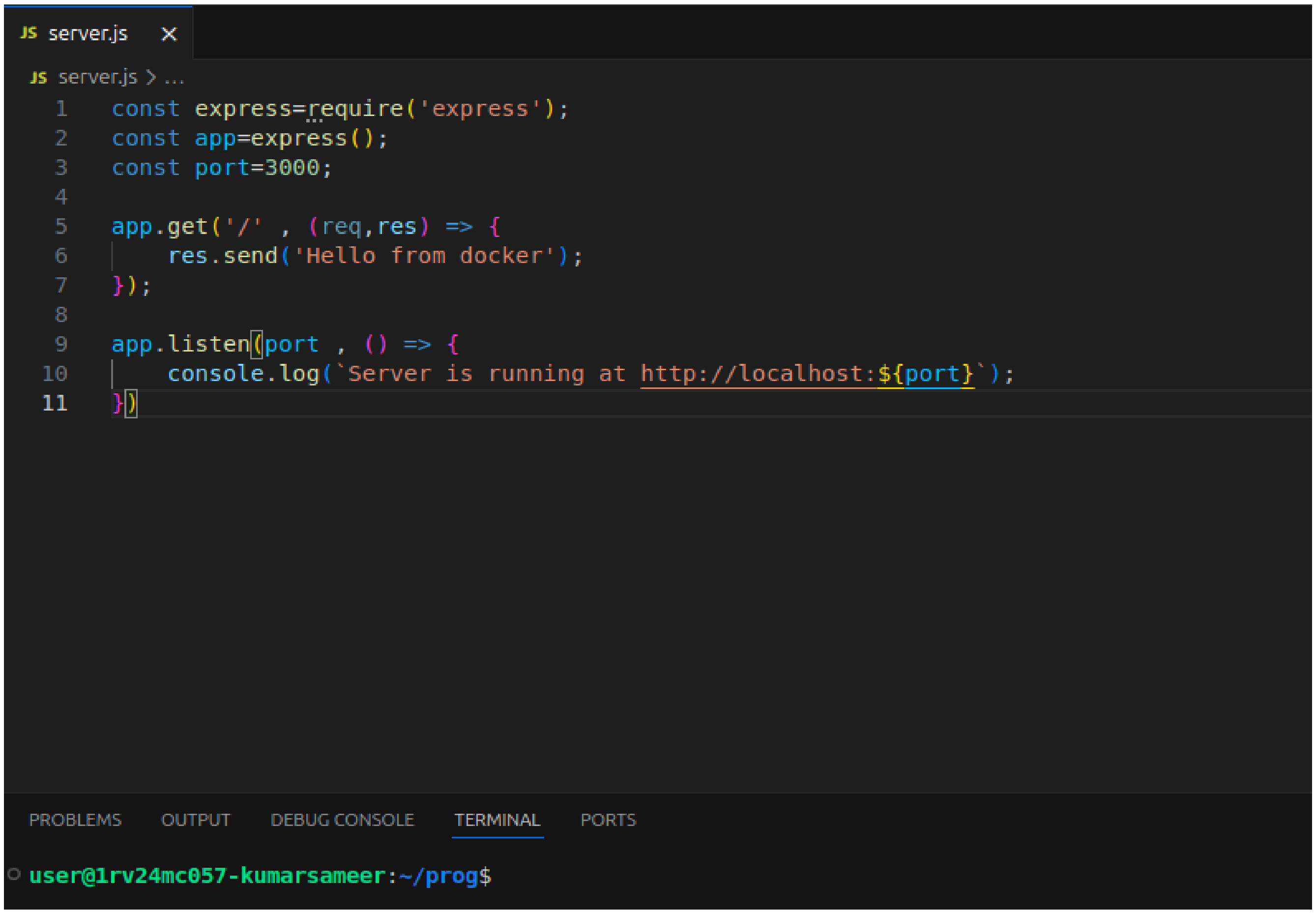
<u>Create the Application Code</u>

Step 1: Create a file named server.js

```
nano server.js
```

(If you're on Windows and don't have touch, just create the file manually or run echo. > server.js.)

Step 2: Add the following code to server.js



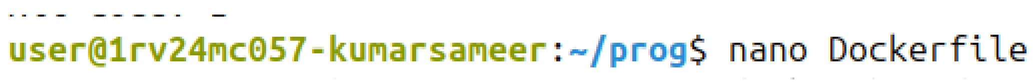
```javascript
const express=require('express');
const app=express();
const port=3000;

app.get('/' , (req,res) => {
    res.send('Hello from docker');
});

app.listen(port , () => {
    console.log(`Server is running at http://localhost:${port}`);
})
```

```
user@1rv24mc057-kumarsameer:~/prog$
```

Creates a web server on port 3000

Sends a simple “Hello from Docker” message when you visit /

Create a dockerfile inside the project directory



```
user@1rv24mc057-kumarsameer:~/prog$ nano Dockerfile
```

Dockerfile commands

1.`FROM node:latest`

This line tells Docker to start building the image using the latest version of Node.js as the base image.

2.`WORKDIR /app`

This sets the working directory inside the container to `/app`.
Any following commands (like `COPY`, `RUN`, or `CMD`) will happen inside this `/app` folder.
If `/app` doesn't exist, Docker will create it automatically.

3.`LABEL author=sam`

This adds a label (a piece of metadata) to the image.
It's not required for the container to run but is useful for documentation or identifying who built the image.

4. `ARG TYPE=PROD`

This defines a build-time variable called `TYPE`, and gives it a default value of `PROD`.
You can override this when building the image, like so:

```
docker build --build-arg TYPE=DEV -t myapp .
```

5. `ENV TYPE=${TYPE}`

This creates an environment variable inside the container, using the value of the build time argument TYPE.
This means the app running inside the container will have access to an environment variable called `TYPE` (for example, `PROD` or `DEV`).

6. `COPY package.json ./`

This copies the `package.json` file from your computer into the container's `/app` folder.
This file lists your project's dependencies.

7. `COPY package-lock.json ./`

This copies the `package-lock.json` file into the container too.
Having both `package.json` and `package-lock.json` ensures that npm installs the exact same versions of dependencies each time.

8. `RUN npm install`

This command installs all the dependencies listed in `package.json` inside the container.
After this step, your container will have all the Node.js packages your app needs to run.

9. `COPY . .`

This copies the rest of your project files (like your source code, config files, etc.) from your computer into the container's `/app` directory.

10. `EXPOSE 3000`

This tells Docker that the application inside the container will use port 3000.
It doesn't actually open the port; it's more of a hint to anyone running the container which port to connect to.
You still need to map the port when running the container, like:

```
docker run -p 3000:3000 myapp
```

---

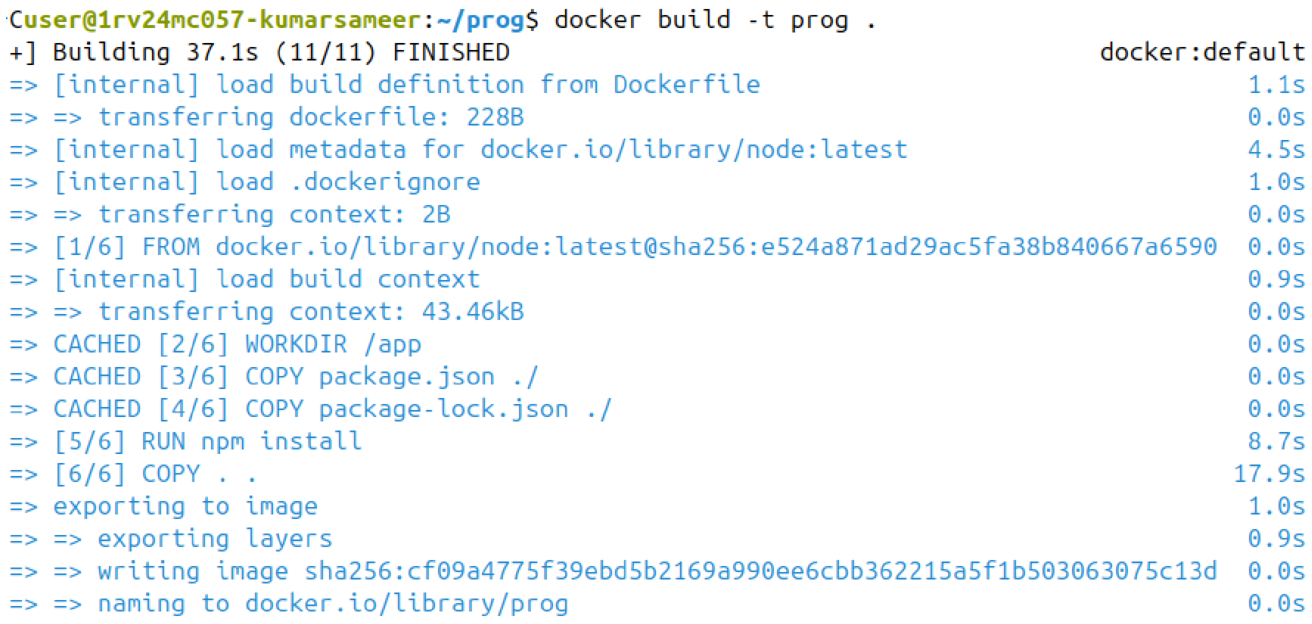
11. `CMD ["node", "server.js"]`

This defines the default command that runs when the container starts.
Here, it runs your Node.js application by executing:

```
node server.js
```

Now building the image using docker commands in the terminal

```
docker build -t name .
```

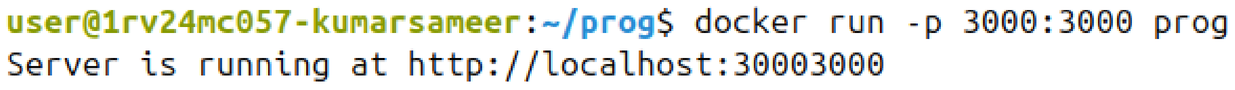


```
Cuser@1rv24mc057-kumarsameer:~/prog$ docker build -t prog .
+] Building 37.1s (11/11) FINISHED                                          docker:default
=> [internal] load build definition from Dockerfile                                   1.1s
=> => transferring dockerfile: 228B                                                   0.0s
=> [internal] load metadata for docker.io/library/node:latest                         4.5s
=> [internal] load .dockerignore                                                      1.0s
=> => transferring context: 2B                                                        0.0s
=> [1/6] FROM docker.io/library/node:latest@sha256:e524a871ad29ac5fa38b840667a6590   0.0s
=> [internal] load build context                                                      0.9s
=> => transferring context: 43.46kB                                                   0.0s
=> CACHED [2/6] WORKDIR /app                                                          0.0s
=> CACHED [3/6] COPY package.json ./                                                  0.0s
=> CACHED [4/6] COPY package-lock.json ./                                             0.0s
=> [5/6] RUN npm install                                                              8.7s
=> [6/6] COPY . .                                                                    17.9s
=> exporting to image                                                                 1.0s
=> => exporting layers                                                                0.9s
=> => writing image sha256:cf09a4775f39ebd5b2169a990ee6cbb362215a5f1b503063075c13d   0.0s
=> => naming to docker.io/library/prog                                                0.0s
```

This runs through each line of the Dockerfile and creates an image that is ready to run.

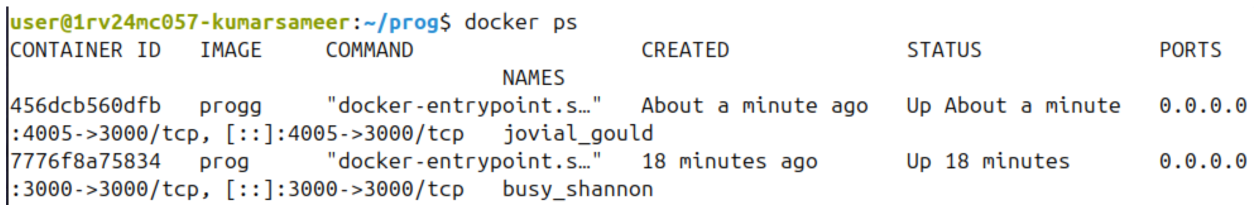
Now run the container using

```
docker run -p 3000:3000 myapp
```



```
user@1rv24mc057-kumarsameer:~/prog$ docker run -p 3000:3000 prog
Server is running at http://localhost:30003000
```

This starts the app in a container and makes it accessible at `http://localhost:3000` as we've defined it in the command.



```
user@1rv24mc057-kumarsameer:~/prog$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED              STATUS              PORTS
                                                  NAMES
456dcb560dfb   progg     "docker-entrypoint.s…"   About a minute ago   Up About a minute   0.0.0.0
:4005->3000/tcp, [::]:4005->3000/tcp   jovial_gould
7776f8a75834   prog      "docker-entrypoint.s…"   18 minutes ago       Up 18 minutes       0.0.0.0
:3000->3000/tcp, [::]:3000->3000/tcp   busy_shannon
```