**Project Description**

This is a very simple JavaScript application created specifically for Docker demonstration purposes. The application uses basic HTML, and in this project, we will be writing a Dockerfile to build an image from it. We'll then run a container with a persistent volume and map it to a port.

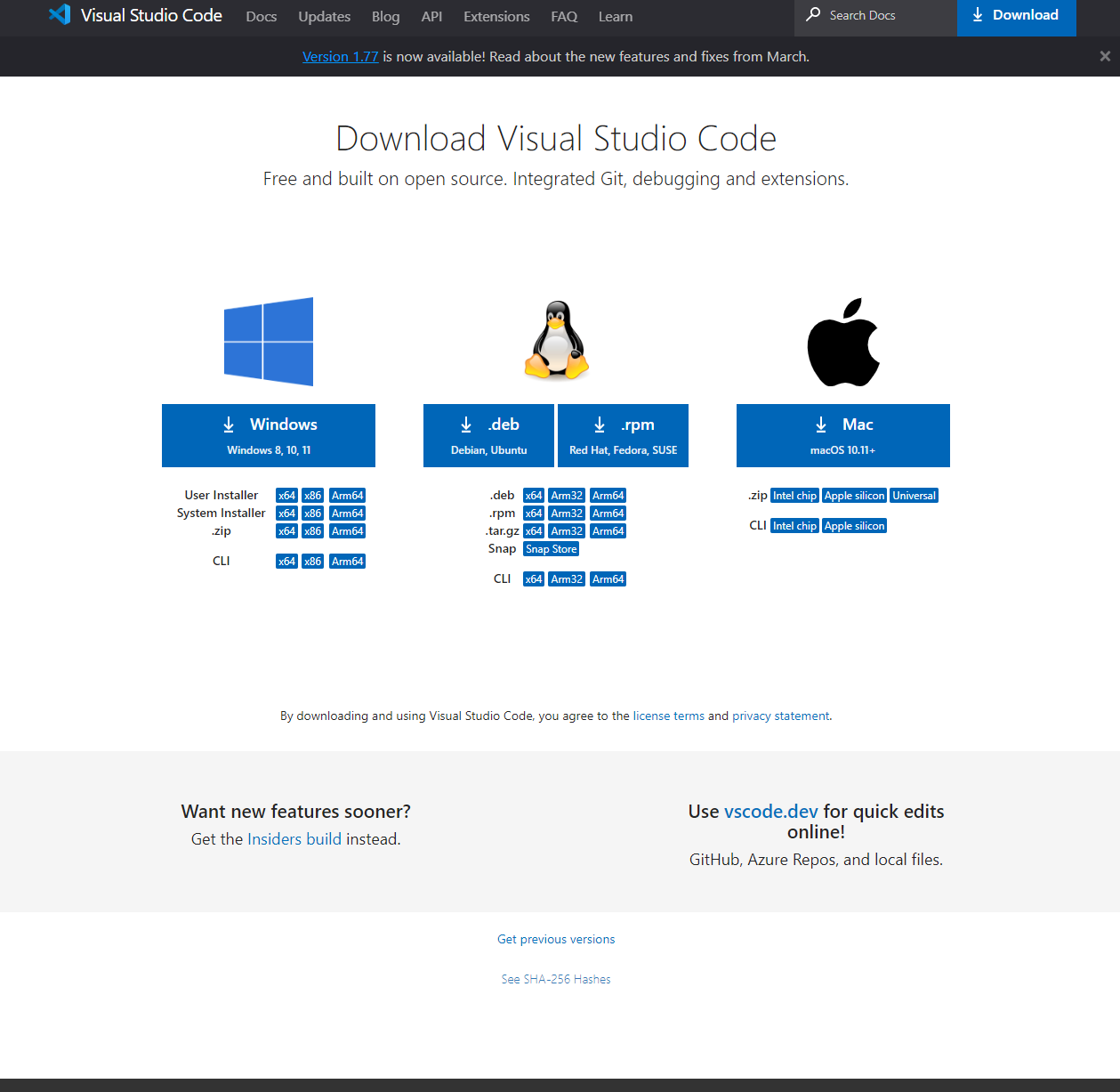
Nodemon is used for automatic restarts of the Node.js application during development.

Part 1

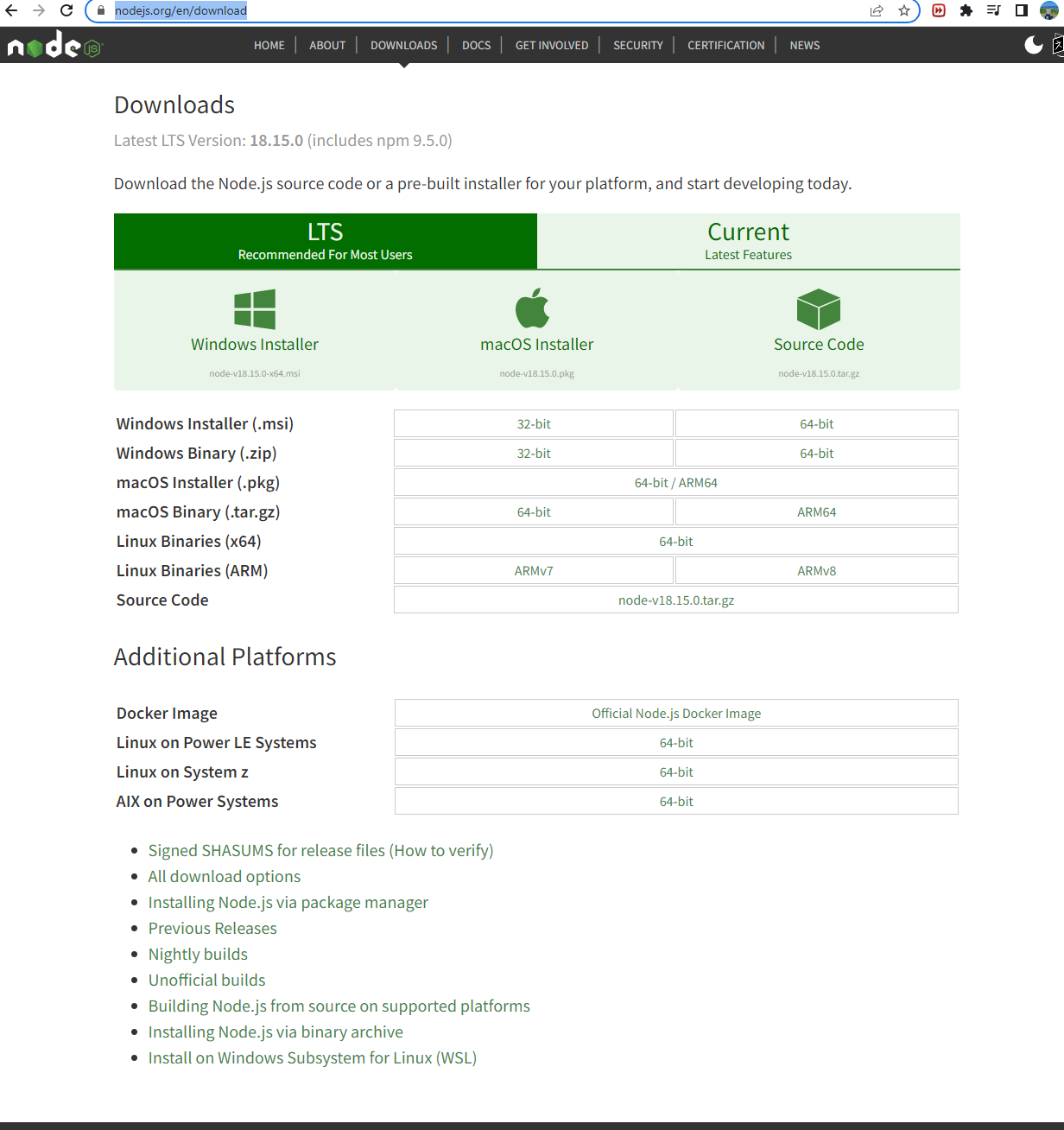
**Prerequisites**

**Step 1 :**

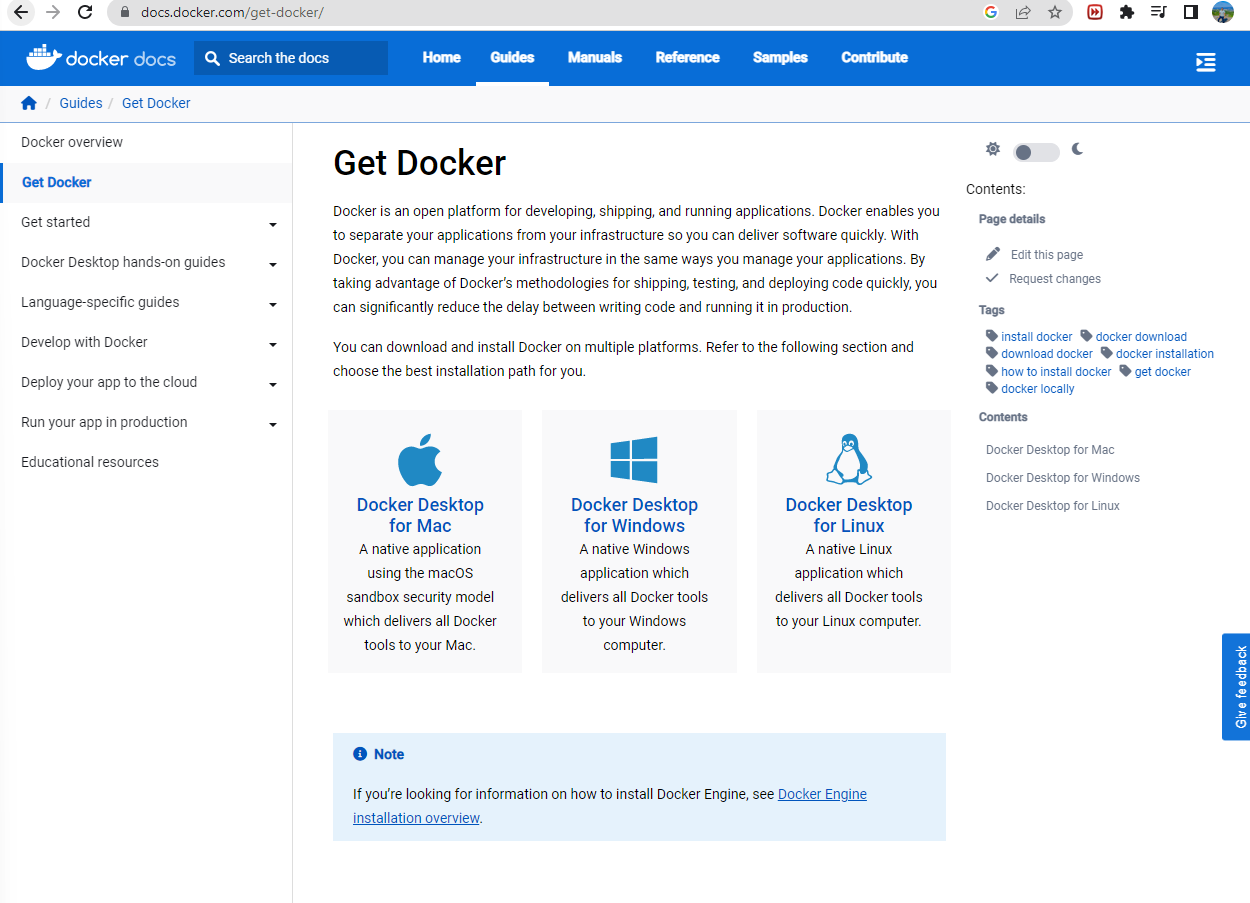
Before running the project, you will need to have Visual code , nodejs , Docker installed on your local machine.

1- To install Visual code Go to [**https://code.visualstudio.com/download**](https://code.visualstudio.com/download*)

2- To install Nodejs, First download Nodejs from this link **https://nodejs.org/en/download**



3- To install Docker using this link[**https://docs.docker.com/get-docker/**](https://docs.docker.com/get-docker/)

****

**Step 2:**

Once you have installed the prerequisite applications, follow these steps to set up the app:

1. Open the terminal and create a directory for the app using the command mkdir node-app. You can choose any location on your local host for this directory.
2. Change to the newly created directory using the command cd node-app.
3. To open Visual Studio with the directory set to the app's directory, use the command code in the terminal.

**Step 3:**

1. Open the Visual Studio terminal.
2. Type the command "node-app npm init" in the terminal and hit enter.
3. Once the command is executed, you will find a new file called "package.json" created in the directory.
4. Start building the Node.js application by adding an "index.js" file that will contain the code for the app.
5. Note that as this is a backend application, we need to create an Express server that can receive requests.
6. Install the necessary dependency for Express by running the command "npm i express" in the terminal.
7. After installation, a new folder called "node\_modules" will be created in your app directory.
8. Write the application code, which can be copied and pasted into the "index.js" file.

const express = require ('express');

// init app

const PORT = 4000;

const app = express();

app.get('/', (req, res) => res.send ( '<h1> Hello ' ));

app.listen(PORT, () => console.log(`app is up and running on port: ${PORT} `));

To ensure that the application is running on the local host, you can execute the command "node index.js". Running this command should generate the following output.



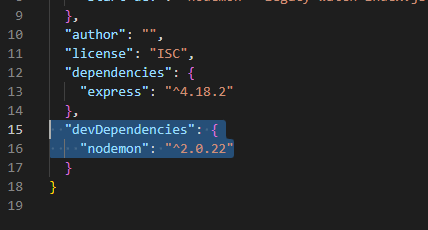
Open your browser and type in "localhost:4000" to view the HTML display of "Hello".



**Step 4 :**

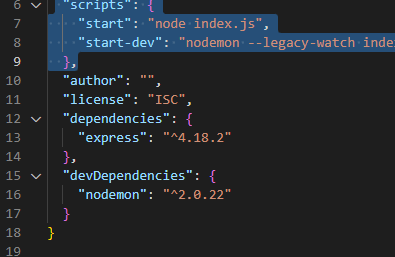
To enable automatic refreshing of your app when changes are made, you will have to install the nodemon package, which will allow your app to listen for any modifications.

To install nodemon as a dependency and enable automatic refreshing of your app upon changes, you can use the command "npm i nodemon --save-dev". This will install nodemon and add it to your package.json file as a dev dependency.

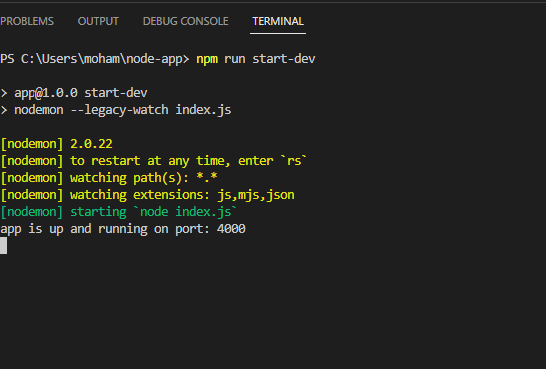


To run the server and enable it to listen for changes, we need to add two scripts to the package.json file. The first script will run the server, while the second script will enable nodemon to watch for any changes in the code.

To do this, we will modify the existing script in the package.json file as follows:



Now if you run “npm run start-dev”



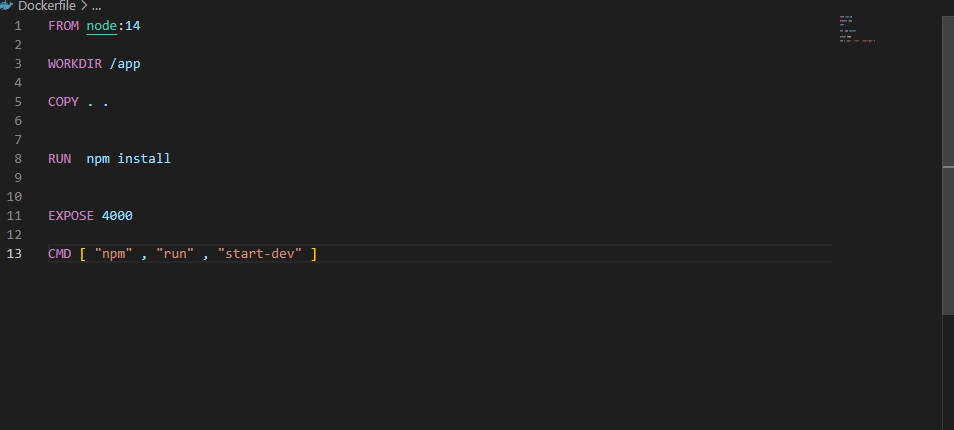
Once nodemon is up and running, it will continuously monitor your code for any changes. If you modify the HTML code and save the file, you can see the changes reflected immediately in the browser without having to manually exit and restart the app.

Up until now, we have not discussed anything related to Docker. We have simply been running the application locally in preparation for the next step, which is to run the same application using a Docker container. In this next step, we will create a Dockerfile that contains an image with Node.js pre-installed. We will copy all the necessary files to the app directory and take the necessary steps to run the Docker container. Please proceed to part 2 for further details.

Part 2

**Dockerfile**

The Dockerfile is used to build the Docker image for the application. Here's the Dockerfile used for this project:



The FROM keyword specifies the base image to use, which is Node.js version 14.

The WORKDIR sets the working directory for the container, and COPY copies all the files in the current directory to the /app directory in the container.

The RUN command install all necessary dependencies, while EXPOSE exposes port 4000 for the container.

Finally, CMD specifies the command to run when the container starts.

**Building the Docker Image**

To build the Docker image, run the following command in the project directory:

Copy code

**docker build -t express-node .**

This command uses the Dockerfile to build an image tagged as express-node.

If you're currently in the same directory as the Dockerfile, you don't need to specify the path. Just add (.)

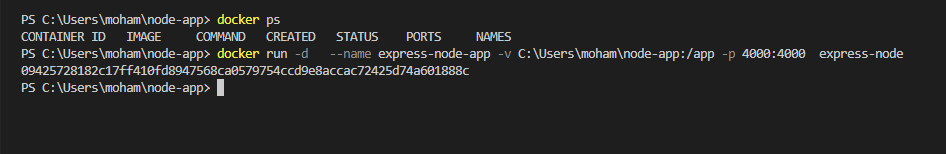
However, if you're in a different directory, you'll need to specify the path to the directory containing the Dockerfile.

Running the Docker Container

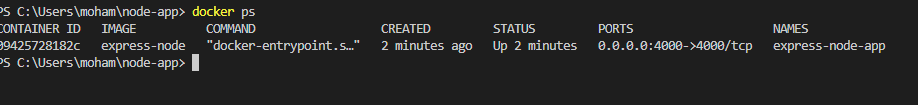
To run the Docker container, use the following command:

Copy code

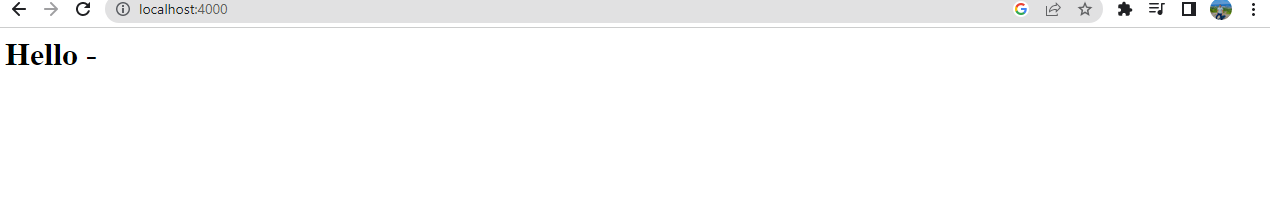
**docker run -d --name express-node-app -v <local\_dir>:/app:ro -p 4000:4000 express-node**

****

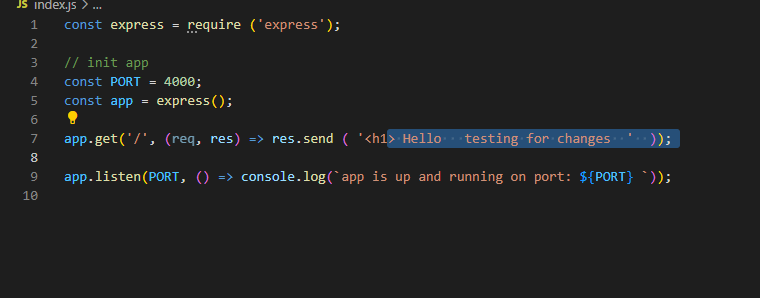
The Docker container should now be up and running, with our app listening on port 80 and nodemon continuously monitoring for changes. To confirm that the container is running, you can use the command "docker ps". This will display a list of all running containers, and you should see that the only running container is "express-node-app"



To view the HTML display from the container, open your browser and type "localhost:4000" in the address bar. You should be able to see the HTML display from the container.



To confirm whether changes made on the local host will be automatically synced to the app running on the container, open the "index.js" file and add a testing message for changes. Make sure to save the changes once you've added the testing message.



Head back to the browser and refresh the page you should see the new text update

****