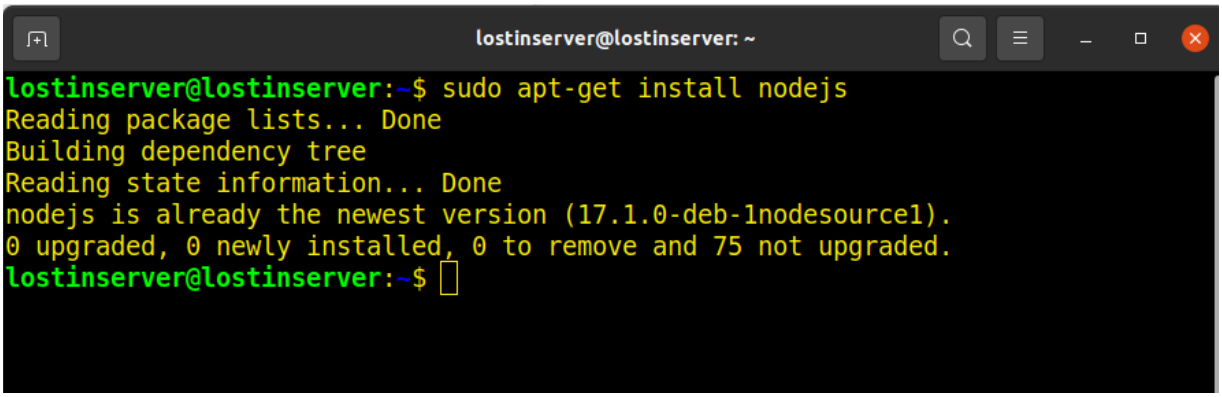
Node JS

Node.js is an open-source, cross-platform, back-end JavaScript runtime environment that runs on the V8 engine and executes JavaScript code outside a web browser.

Installing Node.js in our system

We curl to the latest node from the Node.js website and install it as;

$ curl -sL https://deb.nodesource.com/setup_17.x | sudo -E bash
$ sudo apt-get install -y nodejs



As we can see, we have our newest version installed on our system.

To check the version we can do,
$ nodejs --version

For more native addons for our system,
$ sudo apt-get install -y build-essential

Creating 2 API's running on ports 6080 and 7080

We start initially by creating a directory for our project. Let's create it as **mynodeproject** inside **~/Documents/NodeJs** directory and ,

$ sudo mkdir mynodeproject
$ cd mynodeproject
$ sudo npm init

```
lostinserver@lostinserver: ~/Documents/NodeJs/mynodeproject
license: (ISC)
About to write to /home/lostinserver/Documents/NodeJs/mynodeproject/package.json
:

{
  "name": "mynodeproject",
  "version": "1.0.0",
  "description": "First node project",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Prerit Bhandari",
  "license": "ISC"
}


Is this OK? (yes) yes
npm notice
npm notice New patch version of npm available! 8.1.0 -> 8.1.3
npm notice Changelog: https://github.com/npm/cli/releases/tag/v8.1.3
npm notice Run npm install -g npm@8.1.3 to update!
npm notice
lostinserver@lostinserver:~/Documents/NodeJs/mynodeproject$
```

Thus after few answers given, we have successfully initialized our npm.

Creating api1 running on port 6080

We give our api name as **api1.js** and we create it as,

a) Create a js file first,

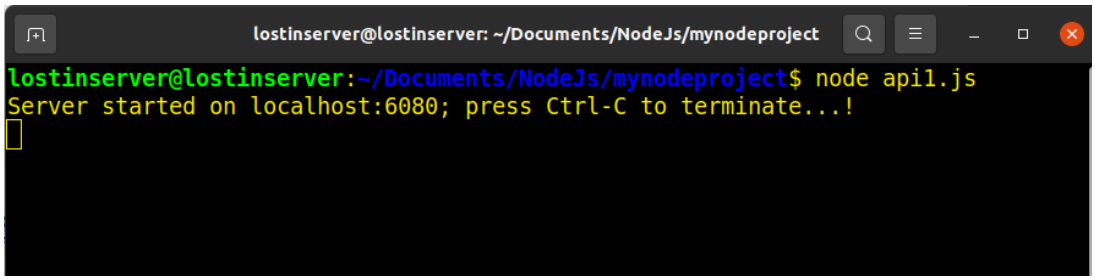
$ sudo nano api1.js

b) Put the code in it,

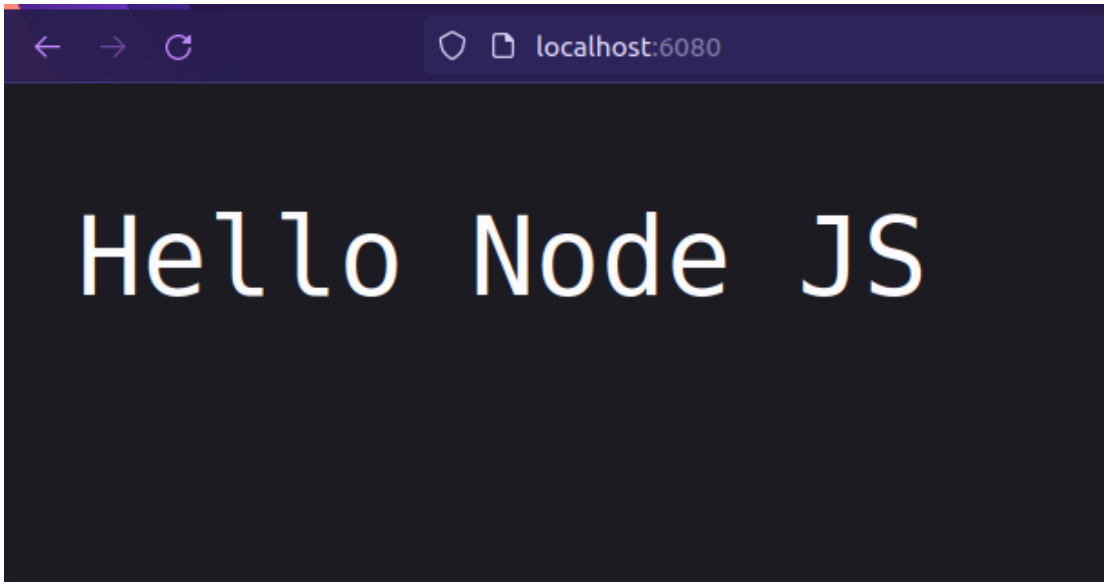
```
var http = require('http');
http.createServer(function(req,res){
res.writeHead(200, { 'Content-Type': 'text/plain' });
res.end('Hello Node JS');
}).listen(6080);
console.log('Server started on localhost:6080; press Ctrl-C to terminate...!');
```

c) Start the application **ap1** to run on port 6080,

$ node api1.js



As we can see our server has started. Let's check it in the browser:



Creating api2 running on port 6080

Similarly as above, we give our api name as **api2.js** and we create it as,

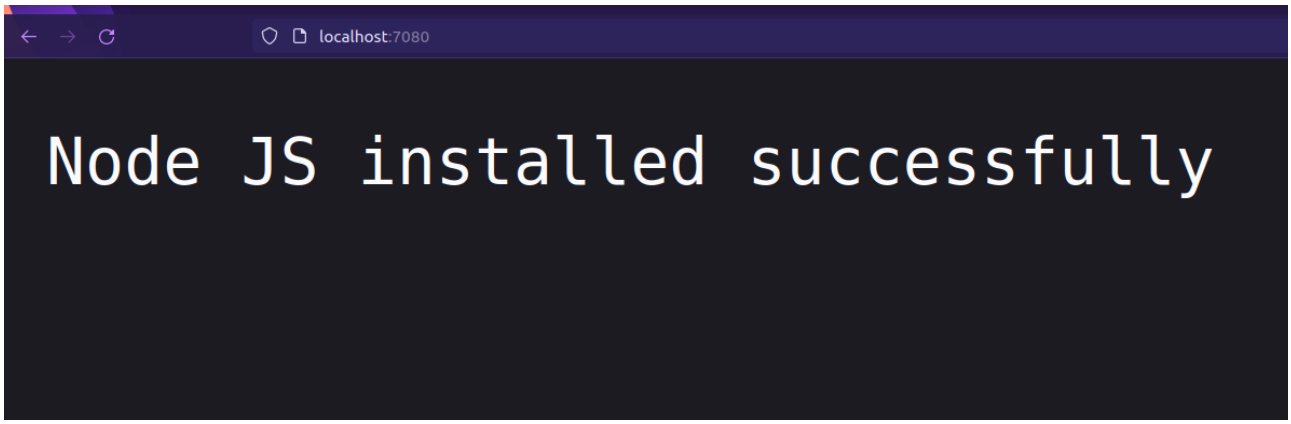
a) Create a js file first,

$ sudo nano api2.js

b) Put the code in it,

```
var http = require('http');
http.createServer(function(req,res){
res.writeHead(200, { 'Content-Type': 'text/plain' });
res.end('Node JS installed successfully');
}).listen(7080);
console.log('Server started on localhost:7080; press Ctrl-C to terminate...!');
```

c) Start the application **ap2** to run on port 7080,
$ node api2.js



Installing pm2 tool and Creating 4 clusters of both nodes

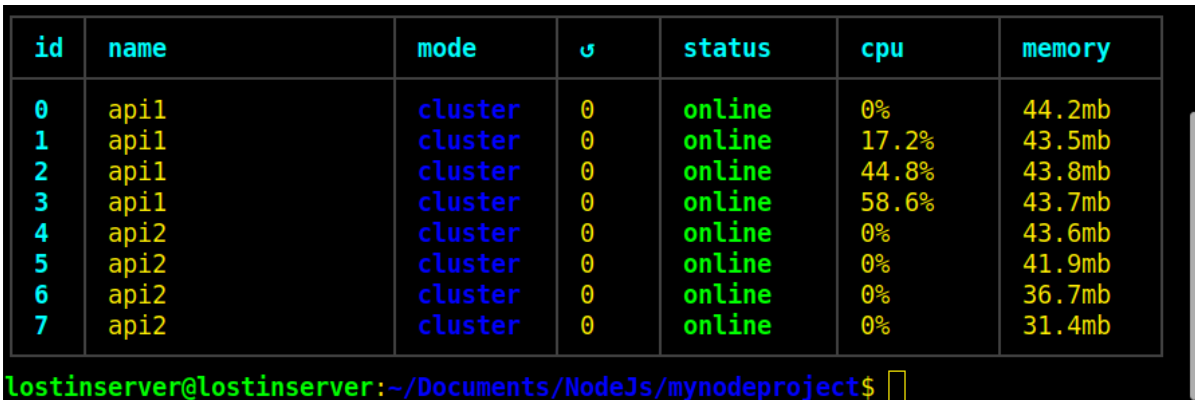
PM2 is a free open source, advanced, efficient and cross-platform production-level process manager for Node.js with a built-in load balancer.

For installation,

$ sudo npm i -g pm2

Now to create 4 clusters for both we can do as,

$ sudo pm2 start api1.js api2.js -i 4 **// for api1.js and api2.js**



id	name	mode	↺	status	cpu	memory
0	api1	cluster	0	online	0%	44.2mb
1	api1	cluster	0	online	17.2%	43.5mb
2	api1	cluster	0	online	44.8%	43.8mb
3	api1	cluster	0	online	58.6%	43.7mb
4	api2	cluster	0	online	0%	43.6mb
5	api2	cluster	0	online	0%	41.9mb
6	api2	cluster	0	online	0%	36.7mb
7	api2	cluster	0	online	0%	31.4mb

lostinserver@lostinserver:~/Documents/NodeJs/mynodeproject$

Deleting all 4 cluster

For deletion of cluster first we need to stop the running cluster by,

 # sudo pm2 stop api1.js api2.js

id	name	mode	↺	status	cpu	memory
0	api1	cluster	0	stopped	0%	0b
1	api1	cluster	0	stopped	0%	0b
2	api1	cluster	0	stopped	0%	0b
3	api1	cluster	0	stopped	0%	0b
4	api2	cluster	0	stopped	0%	0b
5	api2	cluster	0	stopped	0%	0b
6	api2	cluster	0	stopped	0%	0b
7	api2	cluster	0	stopped	0%	0b

```
lostinserver@lostinserver:~/Documents/NodeJs/mynodeproject$
```

As we can see now all the clusters are stopped. Now for deletion, we use the cluster **id** to do the deletion one-by-one as,

 # sudo pm2 delete 0
 # sudo pm2 delete 1
 # sudo pm2 delete 2
 # sudo pm2 delete 3
 # sudo pm2 delete 4
 # sudo pm2 delete 5
 # sudo pm2 delete 6
 # sudo pm2 delete 7

```
lostinserver@lostinserver:~/Documents/NodeJs/mynodeproject$ sudo pm2 delete 5
[PM2] Applying action deleteProcessId on app [5](ids: [ '5' ])
[PM2] [api2](5) ✓
```

id	name	mode	↺	status	cpu	memory
6	api2	cluster	0	stopped	0%	0b
7	api2	cluster	0	stopped	0%	0b

```
lostinserver@lostinserver:~/Documents/NodeJs/mynodeproject$ sudo pm2 delete 6
[PM2] Applying action deleteProcessId on app [6](ids: [ '6' ])
[PM2] [api2](6) ✓
```

id	name	mode	↺	status	cpu	memory
7	api2	cluster	0	stopped	0%	0b

```
lostinserver@lostinserver:~/Documents/NodeJs/mynodeproject$ sudo pm2 delete 7
[PM2] Applying action deleteProcessId on app [7](ids: [ '7' ])
[PM2] [api2](7) ✓
```

id	name	mode	↺	status	cpu	memory

All the clusters have been successfully deleted.

React JS

React JS is a free and open-source front-end JavaScript library for building user interfaces or UI components. It is developed by Facebook (Meta) in 2011.

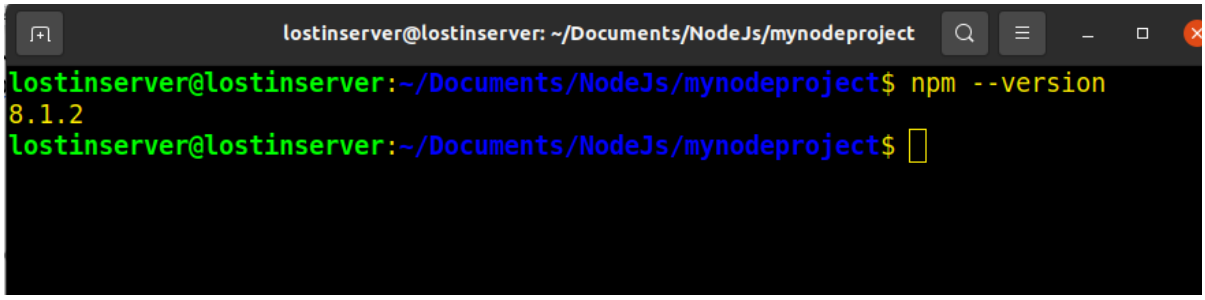
Installing react js in our system

Firstly we need to install **npm** package manager in our system so that we can move ahead.

“ ***npm*** *is a package manager for the JavaScript programming language maintained by npm.*”

$ sudo apt install npm **// installing npm**

$ npm --version **// checking npm version**

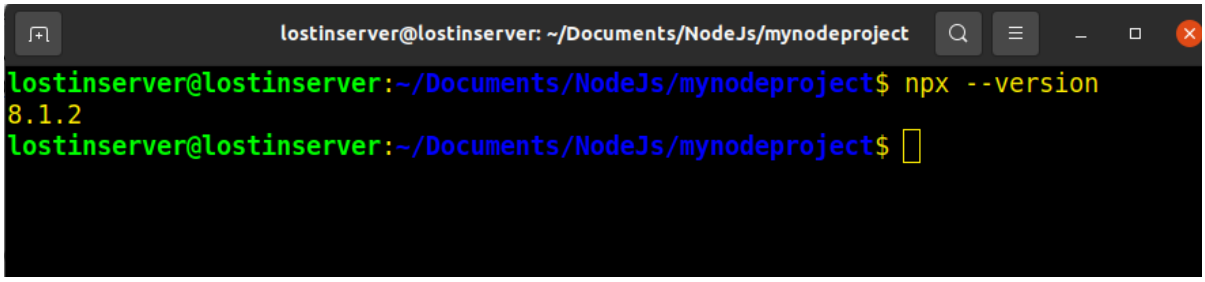


npm has the **npx** package installed by default. If it is not install using,

$ npm install -g npx **// installing npx package**

$ npx --version **// checking npx version**

npx is a package runner tool that comes with npm 5.2+ and helps to create a react app. We use this as through this we always can create a react app with updated packages and also the npx package is a lot smaller than the **create-react-app** utility.



Now, we can create a new react app using,

 # npx create-react-app myreactapp

```
lostinserver@lostinserver: ~/Documents/React

Success! Created myreactapp at /home/lostinserver/Documents/React/myreactapp
Inside that directory, you can run several commands:

  npm start
    Starts the development server.

  npm run build
    Bundles the app into static files for production.

  npm test
    Starts the test runner.

  npm run eject
    Removes this tool and copies build dependencies, configuration files
    and scripts into the app directory. If you do this, you can’t go back!

We suggest that you begin by typing:

  cd myreactapp
  npm start

Happy hacking!
lostinserver@lostinserver:~/Documents/React$
```

As we can see our react app has been successfully created.

Now, in order to print **“Hello React.js”** message we simply edit our **App.js** file located in our myreactapp/src as below,

```
JS App.js  M
src > JS App.js > App
1   import './App.css';
2
3   function App() {
4     return (
5       <div className="App">
6         Hello React.js
7       </div>
8     );
9   }
10
11  export default App;
12
```

(Optional) Let's also modify some css to make it a bit visible. So, let us modify App.css as,

```
JS App.js M    # App.css M ×
src > # App.css > ...
1   .App {
2     text-align: center;
3     line-height: 30vh;
4     font-weight: bold;
5     font-size: 40px;
6   }
7
```

Default port for React is **3000**. We now change it to **3001** by simply using,

$ export PORT=3001

** This should be done inside the project's directory i.e. myreactapp (here)*

After this, we can run npm to view our site,

$ npm start

```
PROBLEMS    OUTPUT    TERMINAL    DEBUG CONSOLE

Compiled successfully!

You can now view myreactapp in the browser.

  Local:            http://localhost:3001
  On Your Network:  http://192.168.254.36:3001

Note that the development build is not optimized.
To create a production build, use npm run build.
```

localhost:3001

Hello React.js