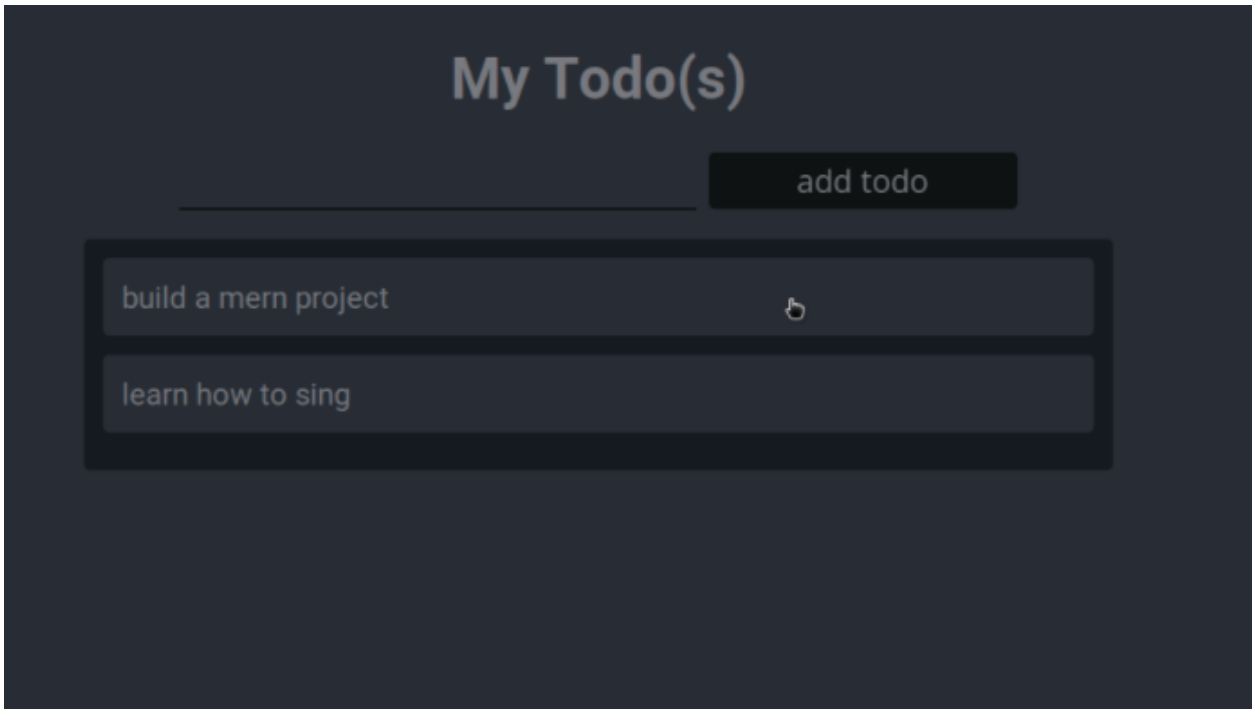
Simple To-Do application on MERN Web Stack

The goal of this project is to describe the concepts of Continuous Integration, Continuous Delivery / Deployment and DevOps on a Lamp web stack.

Never heard about Mern stack? No - Okay.

Mern => mongodb, express, reactjs and nodejs



Tldr;

#Video link

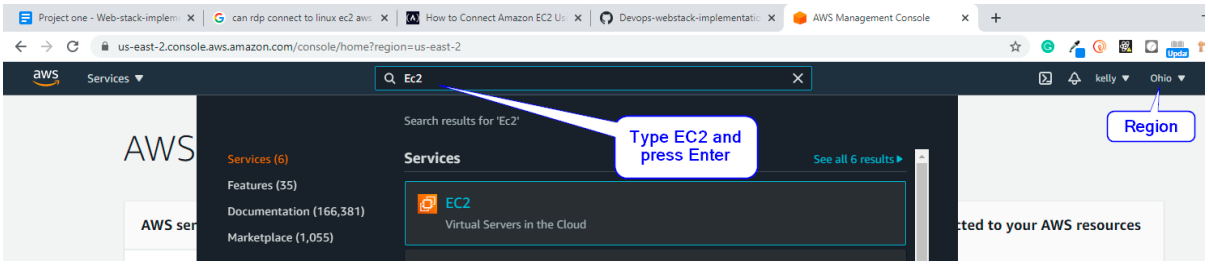
Prerequisites:

- Aws account running an EC2 instance
- Internet connection
- Fundamental Knowledge of downloading and installing
- Basics Linux skills

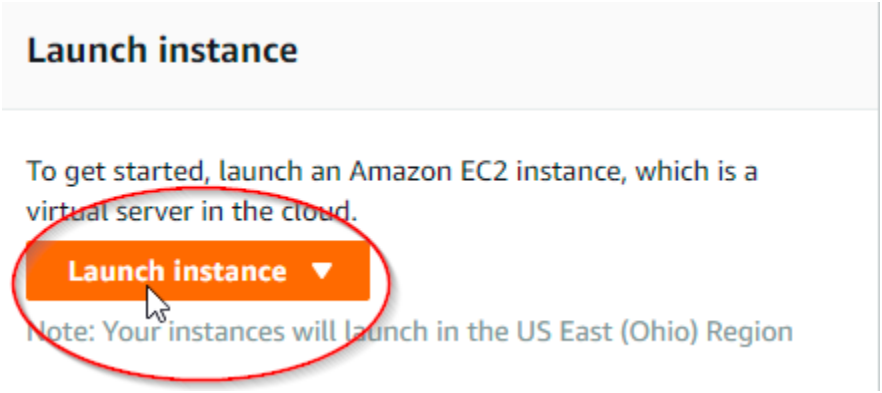
Implementation

- Open your PC browser and login to https://aws.amazon.com/

- A region is selected by default (change if necessary), from the search bar type EC2 and click.

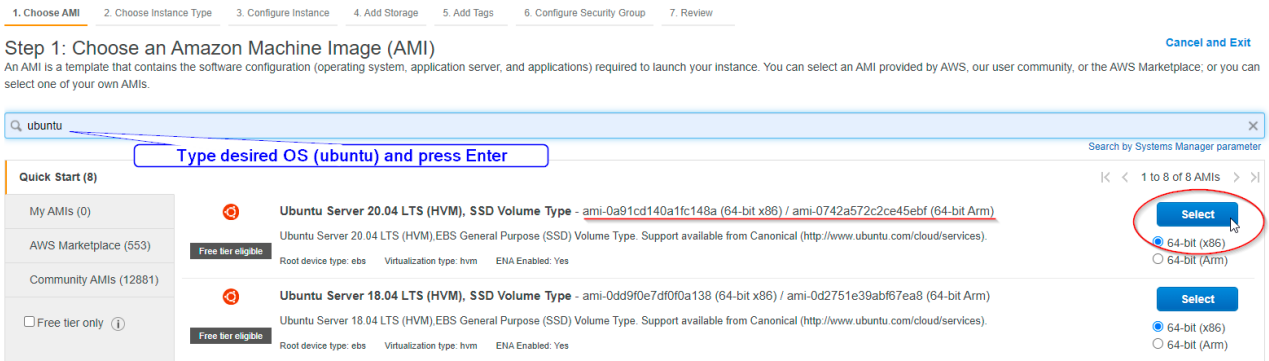


- From the Ec2 dashboard, click on the button “Launch instance” to start using a virtual server.

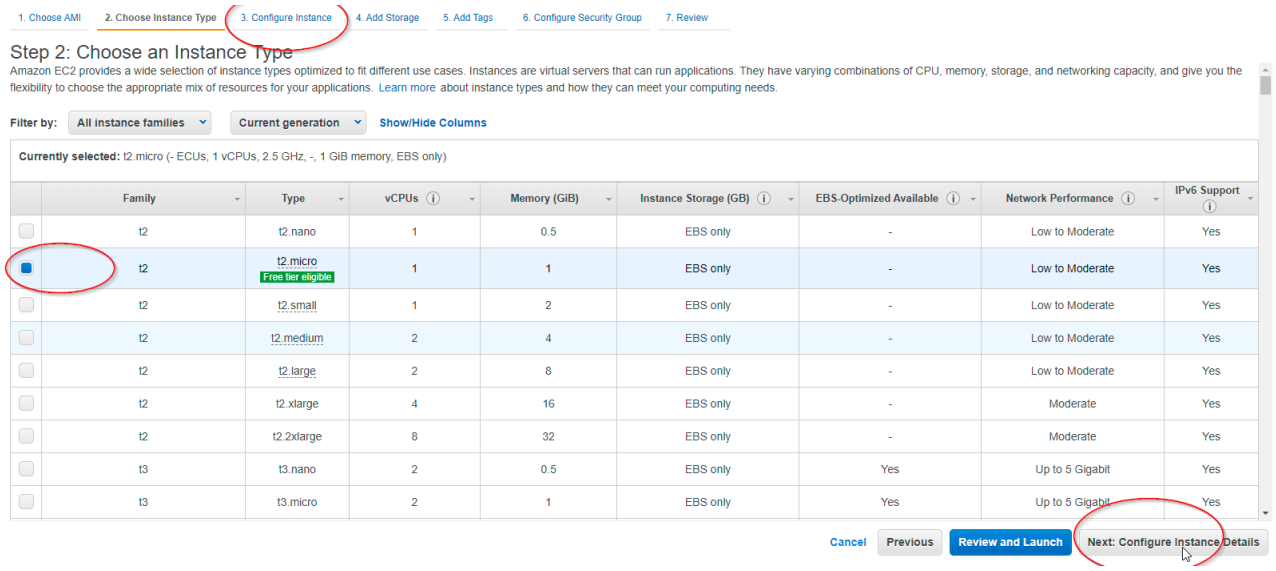


- An AMI window displays, type “Ubuntu” on the search bar and hit enter, or scroll down to select “Ubuntu Server 20.04 LTS (HVM), SSD Volume Type” based on your system architecture.

Note: the AMI (Amazon machine image) is always different from user to user



- The next step of configuring our EC2 is to select the instance type, preferably a **t2 micro - Free tier**. Then click (3) configure instance showing at the top or click next configuration details at the bottom.

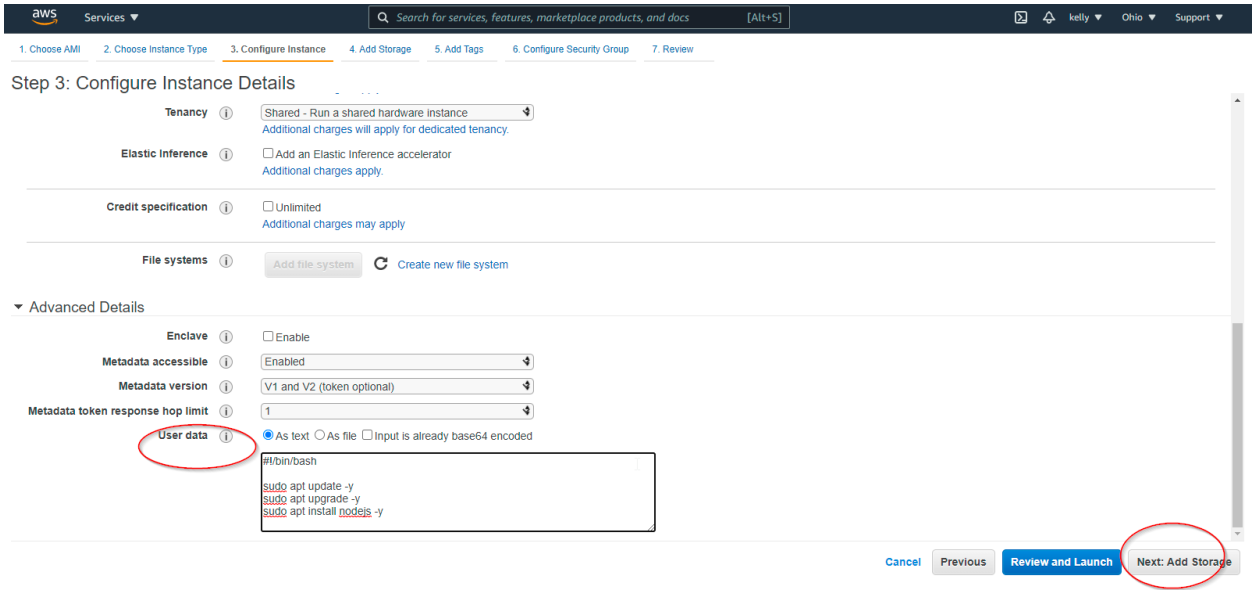


Move to next step

- To configure the instance, we will leave all default but scroll to the bottom and on the advanced details section, in the user data column add below script as shown on the screenshot.

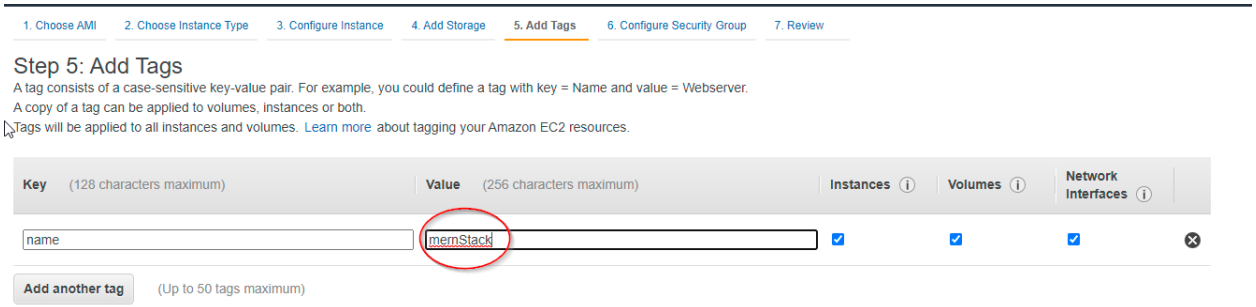
```
#!/bin/bash

sudo apt update -y

sudo apt upgrade -y

sudo apt install nodejs npm -y
```

Move to next step



- Move to tab 5 to Add tags to our EC2 instance, I have deliberately skipped tab 4 to choose the default storage volume given by AWS.

Tags are key-value paired fields and help to categorize your AWS resource, now click ADD TAG to assign a unique name and move to next.

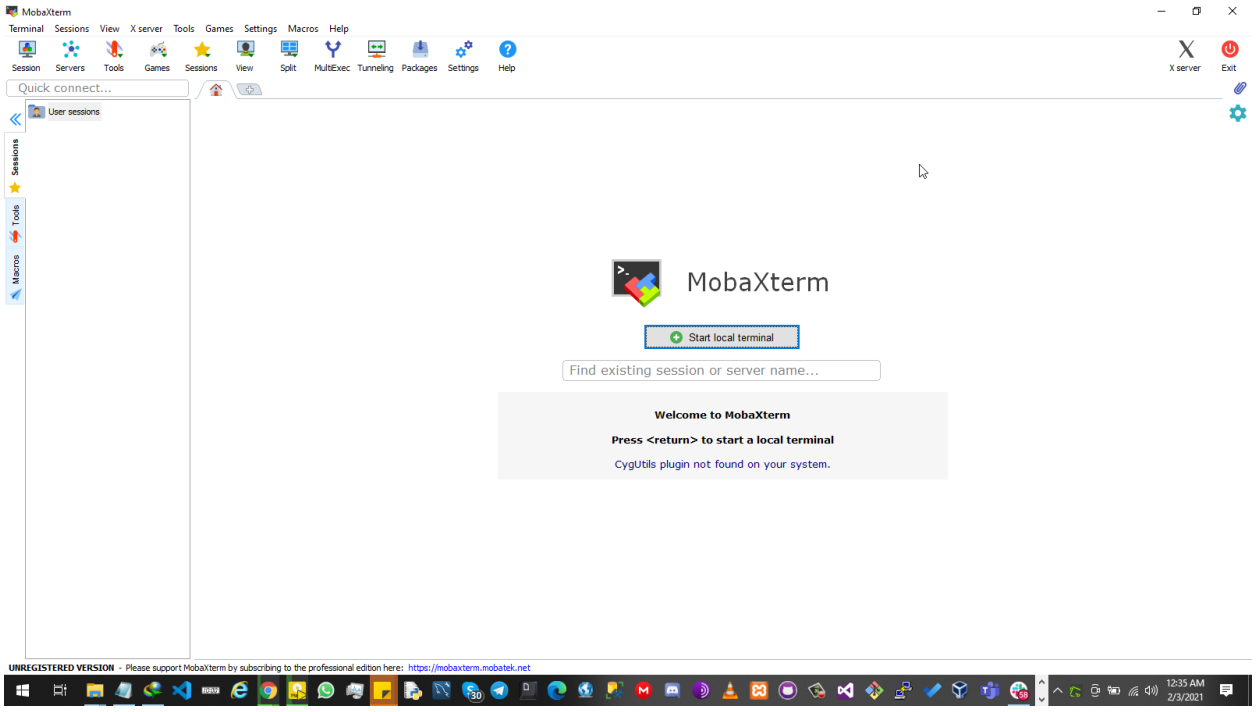


Move to Next

- We will not modify the default security group, but go with the default console access via ssh on **port 22.**

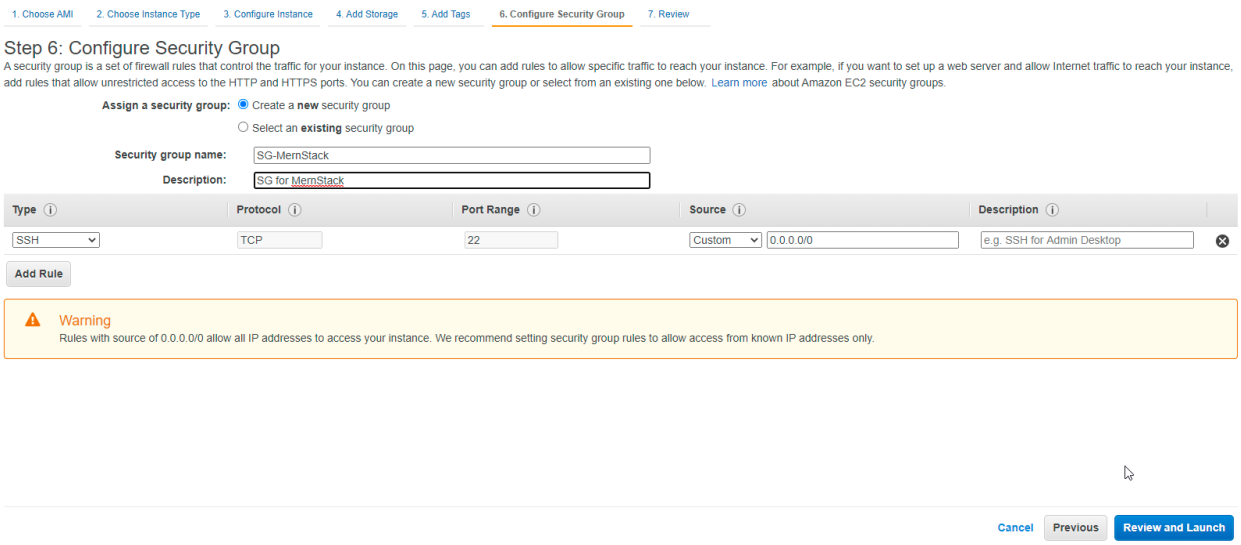
Reason: The security Group are set of firewall rules which denies and grant access to our EC2 instance,

To access the EC2 instance with a console, we use mobaxterm as shown below



Move to Next

Concluding our Ec2 setup,



- Click review and launch

You will get a Prompt to Create a Private Key File, feel free to choose an existing one, if it already exists on the same PC.

Download the key file to a good location, to be used later, Then Launch.

Select an existing key pair or create a new key pair

A key pair consists of a **public key** that AWS stores, and a **private key file** that you store. Together, they allow you to connect to your instance securely. For Windows AMIs, the private key file is required to obtain the password used to log into your instance. For Linux AMIs, the private key file allows you to securely SSH into your instance.

Note: The selected key pair will be added to the set of keys authorized for this instance. Learn more about removing existing key pairs from a public AMI.

Create a new key pair

Key pair name

mernstack

Download Key Pair

You have to download the **private key file** (*.pem file) before you can continue. **Store it in a secure and accessible location.** You will not be able to download the file again after it's created.

Cancel Launch Instances

Initiating Instance Launches

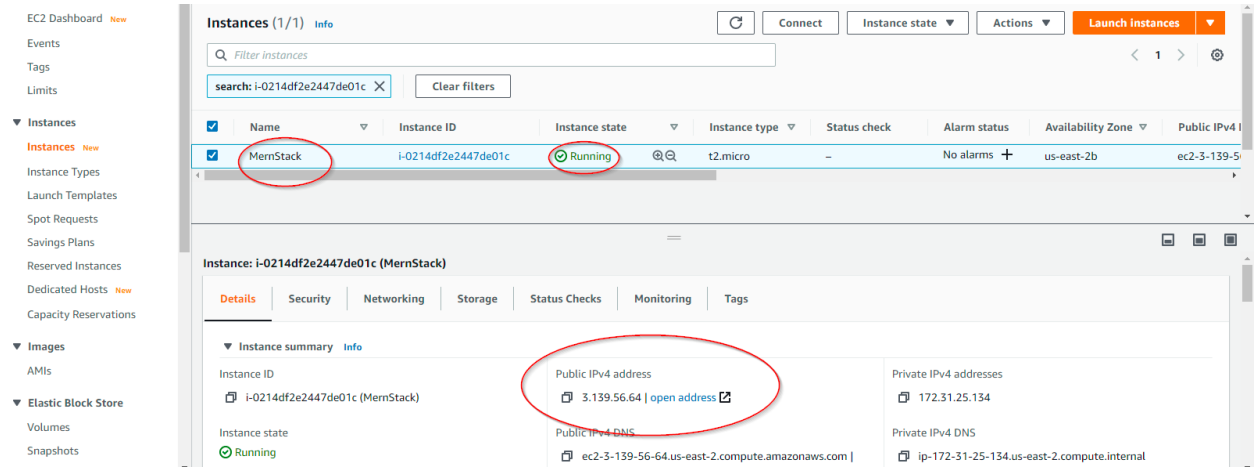
Please do not close your browser while this is loading

Creating security groups... Successful

Authorizing inbound rules... Successful

Initiating launches...

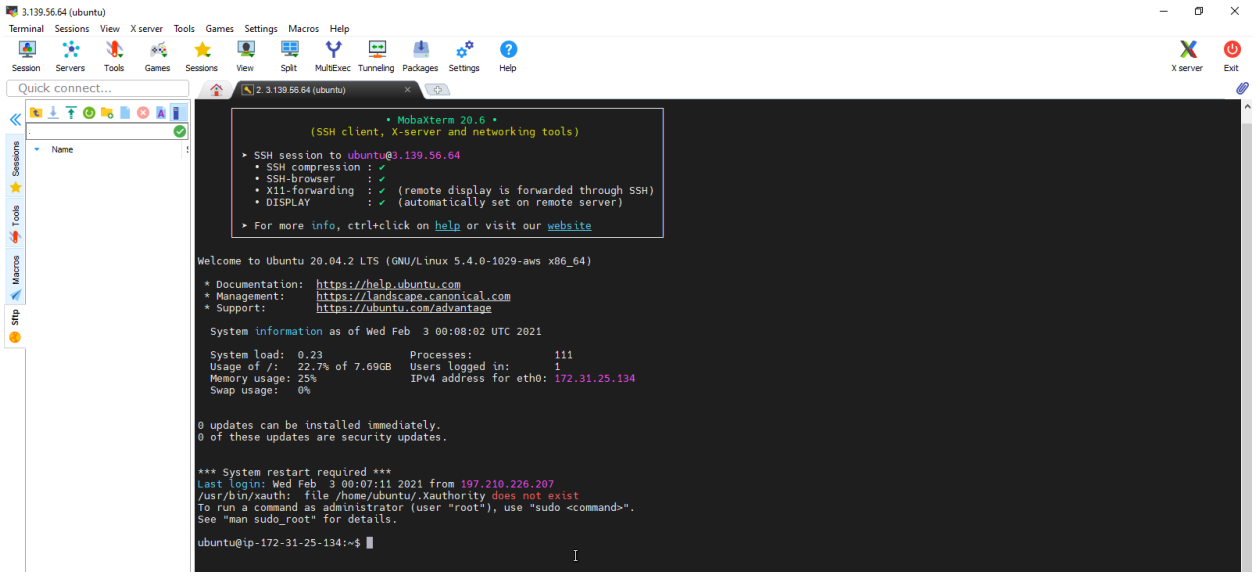
Done? Good Job, let's get to business now.



Copy your own Public IP as shown on the above screenshot, nowit's time to use the console

Yay!!!

Now, Open mobaxterm to connect the Ec2 instance with the public ip shown above,



You have now connected to the EC2 instance via SSH

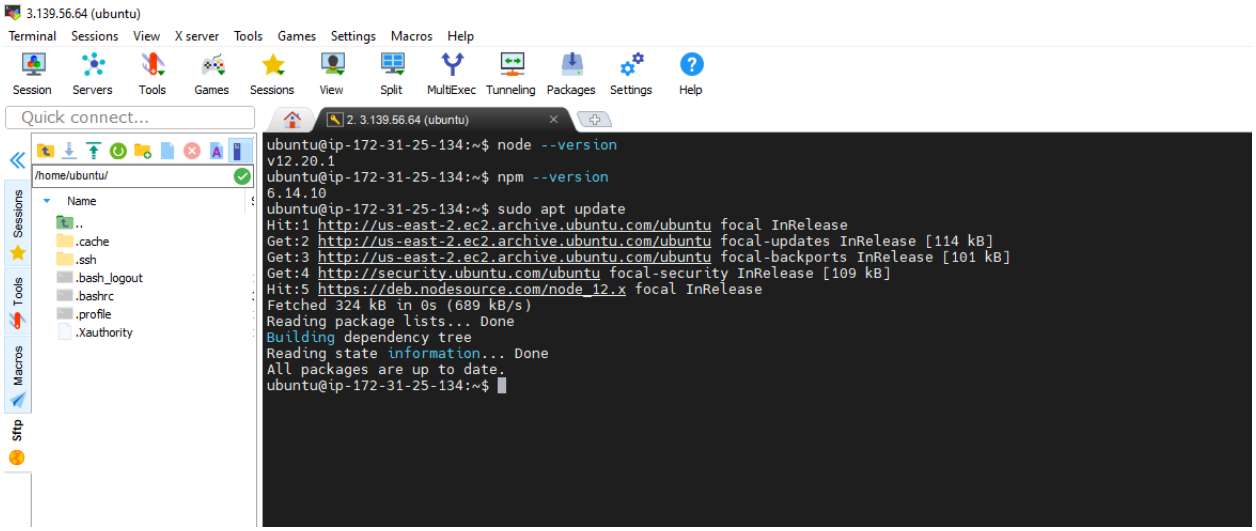
Type clear, to have a neat console and proceed.

We will now check if our userdata scripts were loaded.

Type:

node --version

npm --version



By default EC2 user is given sudo privilege

Great!!!

Application Code Setup

We have our linux instance up and nodejs installed, let's setup the code for our project.

Run this code:

$ mkdir todo && cd todo

```
ubuntu@ip-172-31-25-134:~$ mkdir todo && cd todo
ubuntu@ip-172-31-25-134:~/todo$ npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help init` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (todo)
version: (1.0.0)
description: A todo app on AWS
entry point: (index.js) app.js
test command:
git repository:
keywords:
author: Fredrick kelly
license: (ISC) MIT
About to write to /home/ubuntu/todo/package.json:

{
  "name": "todo",
  "version": "1.0.0",
  "description": "A todo app on AWS",
  "main": "app.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Fredrick kelly",
  "license": "MIT"
}


Is this OK? (yes)
ubuntu@ip-172-31-25-134:~/todo$ ls
package.json
ubuntu@ip-172-31-25-134:~/todo$
```

Install ExpressJs

Express is a framework for Node.js, which helps to define routes of your application based on HTTP methods and URLs.

To use express, we need to install it using npm as shown below:

Run this code:

$ sudo npm install express

```
ubuntu@ip-172-31-25-134:~/todo$ sudo npm install express
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN todo@1.0.0 No repository field.

+ express@4.17.1
added 50 packages from 37 contributors and audited 50 packages in 3.028s
found 0 vulnerabilities

ubuntu@ip-172-31-25-134:~/todo$
```

Great MYSQL is installed and configured for use, we could test by running below code:

$ touch app.js && ls

```
ubuntu@ip-172-31-25-134:~/todo$ touch app.js && ls
app.js  node_modules  package-lock.json  package.json
ubuntu@ip-172-31-25-134:~/todo$
```

Now we have create the file, let's Install the dotenv module to store our environmental variables

$ npm install dotenv

```
ubuntu@ip-172-31-25-134:~/todo$ npm install dotenv
npm WARN todo@1.0.0 No repository field.

+ dotenv@8.2.0
added 1 package and audited 51 packages in 0.835s
found 0 vulnerabilities

ubuntu@ip-172-31-25-134:~/todo$
```

Now let's add some code for our project into app.js file created earlier

```
$ vim app.js
```

Copy and paste below code:

```
// import express

const express = require('express');

// setting the env. config file

require('dotenv').config();

// create app server from the express module

const app = express();

// create and assign port

const port = process.env.PORT || 5500;

// calling the middleware function and allowing cors via http headers from any origin

app.use((req, res, next) => {

res.header("Access-Control-Allow-Origin", "\*");

res.header("Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type, Accept");

next();

});

// the next middleware function, sends text

app.use((req, res, next) => {

res.send('Welcome to my Express Buddy!');

});

// app server listens on env port or given port 5500

app.listen(port, () => {

console.log('\n*************************************')

console.log(`Server running on port http://localhost:${port}`)

});
```

Remember to press save and exit - " :wq "

3. 3.139.56.64 (ubuntu)

```
// import express

const express = require('express');

// setting the env. config file

require('dotenv').config();

// create app server from the express module
const app = express();

// create and assign port

const port = process.env.PORT || 5500;


// calling the middleware function and allowing cors via http headers from any origin

app.use((req, res, next) => {
res.header("Access-Control-Allow-Origin", "\*");
res.header("Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type, Accept");
next();
});



// the next middleware function, sends text
app.use((req, res, next) => {
res.send('Welcome to my Express Buddy!');
});



// app server listens on env port or given port 5500


app.listen(port, () => {
console.log('\n**********************************')
console.log(`Server running on port ${port}`)
});
```

Notice that we have specified to use port **5500** in the code. This will be required later when we go on the browser.

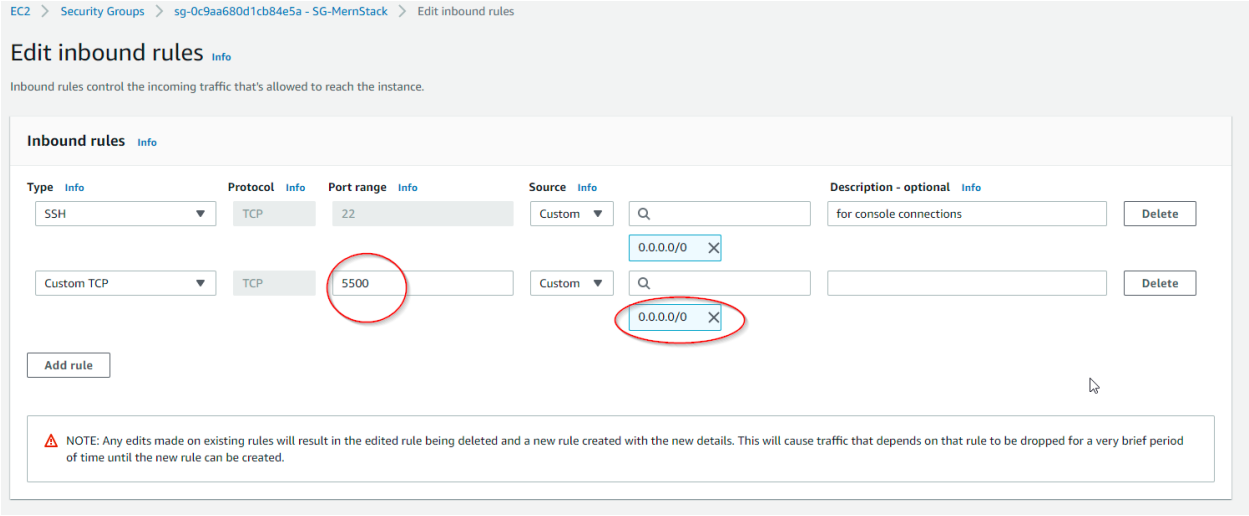
Result to confirm above script, run

```
node app.js
```

```
ubuntu@ip-172-31-25-134:~/todo$ node app.js

**********************************
Server running on port 5500
```

Excellent!!!

Now we need to open this port on our Security Group by editing the inbound rule as shown below:



Let's check the browser now and confirm it runs on port 5500 using the ip or DNS name.

Routes

There are three actions that our To-Do application needs to be able to do:

- Create a new task
- Display list of all tasks
- Delete a completed task

Each task will be associated with some particular endpoint and will use different standard HTTP request methods or sometimes called HTTP verbs :

POST (Create task),

GET (Read tasks by id),

DELETE (Remove task by id)

For each task, we need to create routes that will define various endpoints that the To-do app will depend on. A simple route can be represented as " /route"

So let us create a folder, navigate to it and create a file to store all routes

```
$ mkdir routes && cd routes

$ touch api.route.js

$ vim api.route.js
```

ubuntu@ip-172-31-25-134: ~/todo/routes

```
ubuntu@ip-172-31-25-134:~/todo$ mkdir routes && cd routes
ubuntu@ip-172-31-25-134:~/todo/routes$ touch api.route.js
vim api.route.js
ubuntu@ip-172-31-25-134:~/todo/routes$ vim api.route.js
```

```
// import express and router function

const express = require ('express');
const router = express.Router();

// using the router, create an endpoint on a get method, then move to next middleware function
router.get('/todos', (req, res, next) => {

});

// using the router, create an endpoint on a post method, then move to next middleware function
router.post('/todos', (req, res, next) => {

});


// using the router, create an endpoint by ID on a delete method.
router.delete('/todos/:id', (req, res, next) => {

})


// export the router module, to be called in app.js
module.exports = router;
~
~
~
~
~
~
-- INSERT --                                                    25,25          All
```

Models

Now comes the interesting part, since the app is going to make use of Mongodb which is a NoSQL database, we need to create a model.

A model is at the heart of JavaScript based applications, and it is what makes it interactive.

We will also use models to define the database schema . This is important so that we will be able to define the fields stored in each Mongodb document.

To create a Schema and a model, we will install mongoose which is a Node.js package that makes working with mongodb easier.

Change directory back Todo folder with **cd ..** and install Mongoose

```
$ cd ..

$ npm install mongoose
```

```
ubuntu@ip-172-31-25-134:~/todo$ npm install mongoose -y
npm WARN todo@1.0.0 No repository field.

+ mongoose@5.11.14
added 31 packages from 96 contributors and audited 82 packages in 2.976s

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

found 0 vulnerabilities
```

Run below command:

```
$ mkdir models && cd models && touch todo.model.js
```

Type "ls" without the quotes to confirm the file was created

```
$ vim todo.model.js
```

```
const mongoose = require('mongoose');

const Schema = mongoose.Schema;


//create schema for todo

const TodoSchema = new Schema({

action: {

type: String,

required: [true, 'The todo text field is required']

}

})


//create model for todo

const Todo = mongoose.model('todo', TodoSchema);


module.exports = Todo;
```

Now we need to update our routes from the file **api.route.js** in 'routes' directory to make use of the new model.

In routes directory, delete the code inside with **:%d** command and paste there code below into it then save and exit

```
const express = require ('express');
const router = express.Router();
const Todo = require('../models/todo.model');
router.get('/todos', (req, res, next) => {
//this will return all the data, exposing only the id and action field to the client
Todo.find({}, 'action')
.then(data => res.json(data))
.catch(next)
});
router.post('/todos', (req, res, next) => {
if(req.body.action){
Todo.create(req.body)
.then(data => res.json(data))
.catch(next)
}else {
res.json({
error: "The input field is empty"
})}
});
router.delete('/todos/:id', (req, res, next) => {
Todo.findOneAndDelete({"_id": req.params.id})
.then(data => res.json(data))
.catch(next)
})
module.exports = router;
```

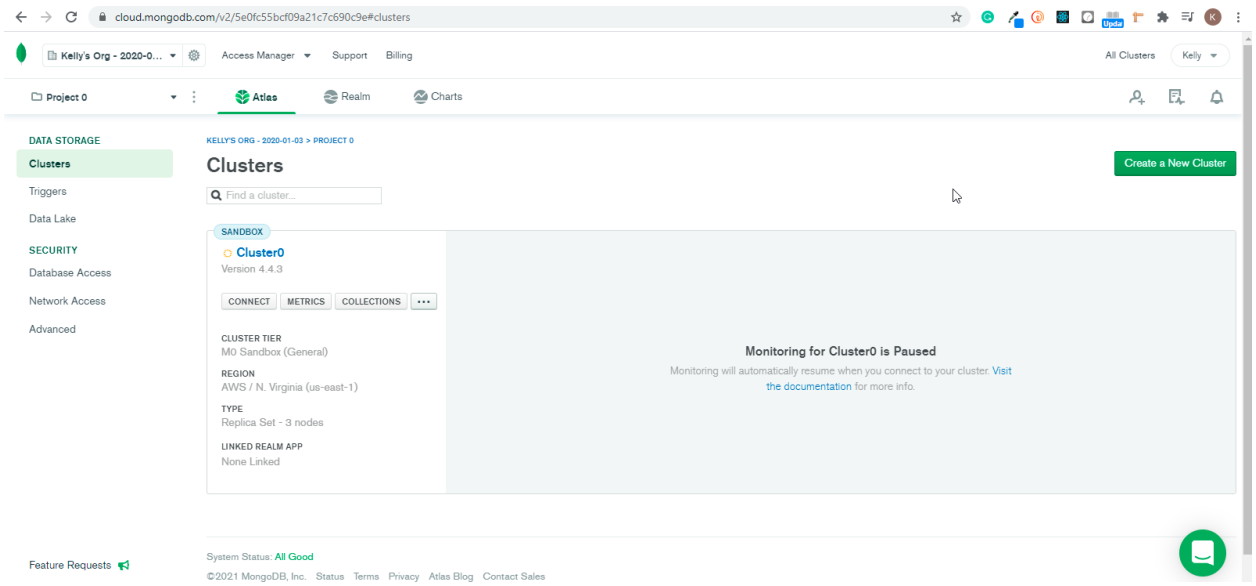
```
ubuntu@ip-172-31-25-134: ~/todo/routes
const express = require ('express');
const router = express.Router();
const Todo = require('../models/todo');

router.get('/todos', (req, res, next) => {

//this will return all the data, exposing only the id and action field to the client
Todo.find({}, 'action')
.then(data => res.json(data))
.catch(next)
});

router.post('/todos', (req, res, next) => {
if(req.body.action){
Todo.create(req.body)
.then(data => res.json(data))
.catch(next)
}else {
res.json({
error: "The input field is empty"
})
}
});

router.delete('/todos/:id', (req, res, next) => {
Todo.findOneAndDelete({"_id": req.params.id})
.then(data => res.json(data))
.catch(next)
})

module.exports = router;
                                                                                    31,24          All
```

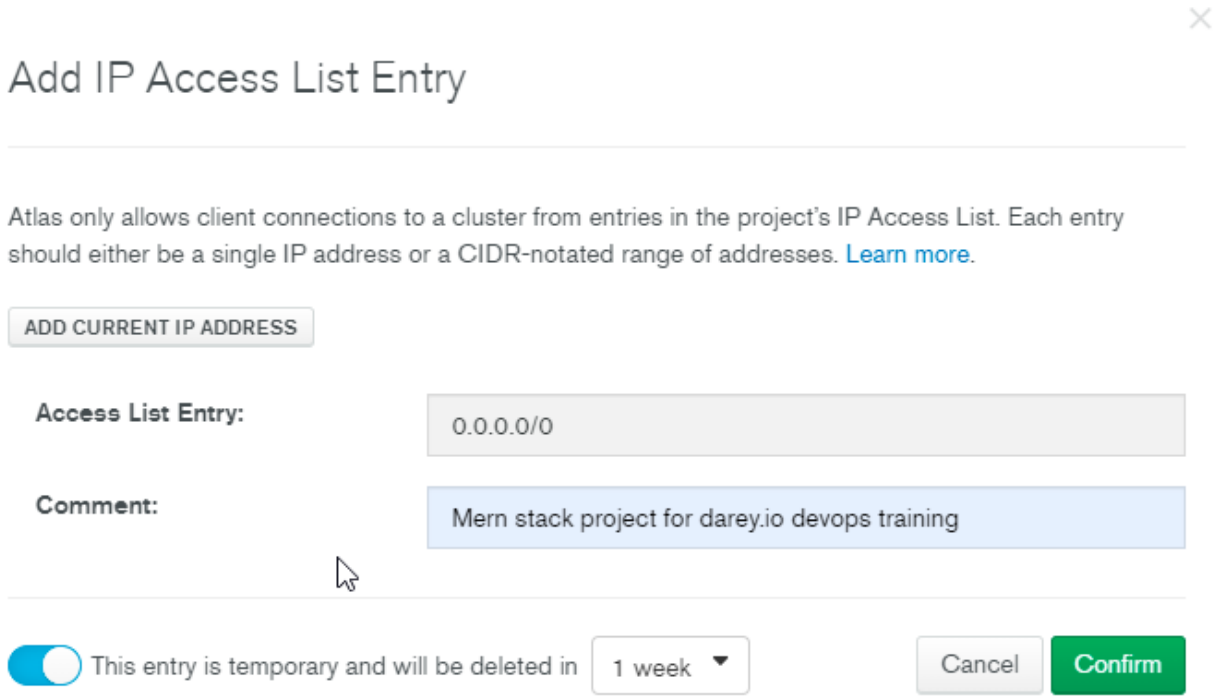
Good Job!

MongoDB Database

We need a database where we will store our data. For this we will make use of mLab. mLab provides MongoDB database as a service solution (DBaaS), so to make life easy, you will need to sign up for a shared clusters free account, which is ideal for our use case. Sign up here. Follow the sign up process, select AWS as the cloud provider, and choose a region near you.

We need to tell apache to enable this new directory to serve our site and disable the default site directory:

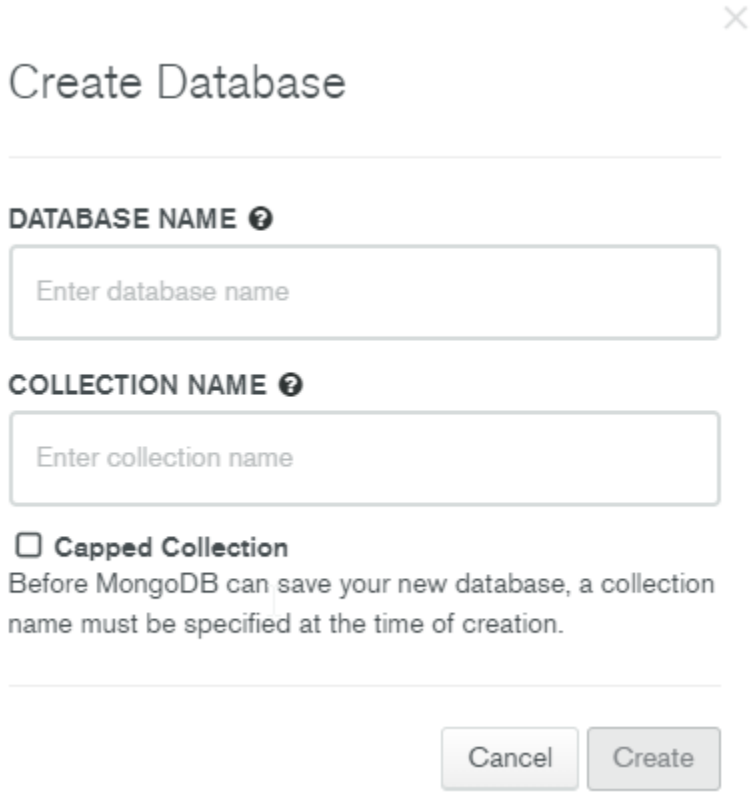




Run this code to store our environment variable file

```
$ touch  .env

$ vim  .env

DB = 'mongodb+srv://<username>:<password>@<network-address>/<dbname>?retryWrites=true&w=majority'
```

Remember to get your dbname, username and password to update your **.env** file



Check the .env file using this command as shown **" ls -lra "**

```
ubuntu@ip-172-31-25-134:~/todo$ ls -lra
total 68
drwxrwxr-x  2 ubuntu ubuntu  4096 Feb  3 08:45 routes
-rw-rw-r--  1 ubuntu ubuntu   331 Feb  3 08:25 package.json
-rw-r--r--  1 ubuntu ubuntu 23742 Feb  3 08:25 package-lock.json
drwxr-xr-x 79 ubuntu ubuntu  4096 Feb  3 08:25 node_modules
drwxrwxr-x  2 ubuntu ubuntu  4096 Feb  3 08:31 models
-rw-rw-r--  1 ubuntu ubuntu   836 Feb  3 07:44 app.js
-rw-rw-r--  1 ubuntu ubuntu    93 Feb  3 12:41 .env
-rw-r--r--  1 ubuntu ubuntu 12288 Feb  3 07:29 .app.js.swp
drwxr-xr-x  7 ubuntu ubuntu  4096 Feb  3 12:41 ..
drwxrwxr-x  5 ubuntu ubuntu  4096 Feb  3 12:41 .
```

Now we need to update the index.js to reflect the use of **.env** so that Node.js can connect to the database.

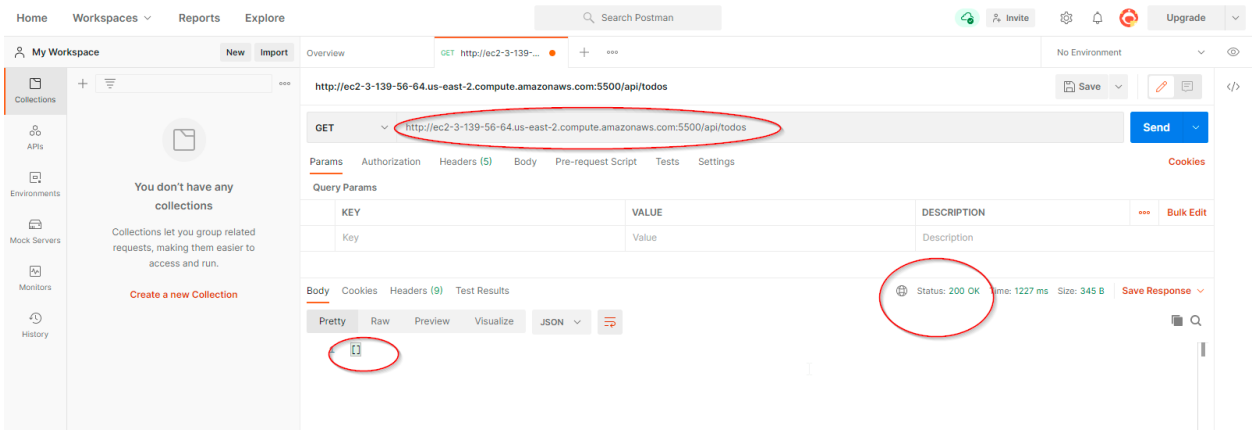
```
const express = require('express');

const bodyParser = require('body-parser');

const mongoose = require('mongoose');

const routes = require('./routes/api.route');

const path = require('path');

require('dotenv').config({ path: '.env'});

const app = express();

const port = process.env.PORT || 5500;

//connect to the database

mongoose.connect(process.env.DB,{ useUnifiedTopology: true }, { useNewUrlParser: true })

.then(() => console.log(`Hey, smile now your database connected successfully!`))

.catch(err => console.log(err));

//since mongoose promise is depreciated, we overide it with node's promise

mongoose.Promise = global.Promise;

app.use((req, res, next) => {

res.header("Access-Control-Allow-Origin", "\*");

res.header("Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type, Accept");

next();

});

app.use(bodyParser.json());

app.use('/api', routes);

app.use((err, req, res, next) => {

console.log(err);

next();

});

app.listen(port, () => {

console.log(`Server running on port ${port}`)

});
```

Simply delete existing content in the app.js file, and update it with the entire code above, and the result is below.

```
ubuntu@ip-172-31-25-134:~/todo$ node app.js
(node:43553) DeprecationWarning: current URL string parser is deprecated, and will be removed in a
n. To use the new parser, pass option { useNewUrlParser: true } to MongoClient.connect.
Server running on port 5500
Hey, smile now your database connected successfully!
```

Testing Backend Code without Frontend using RESTful API

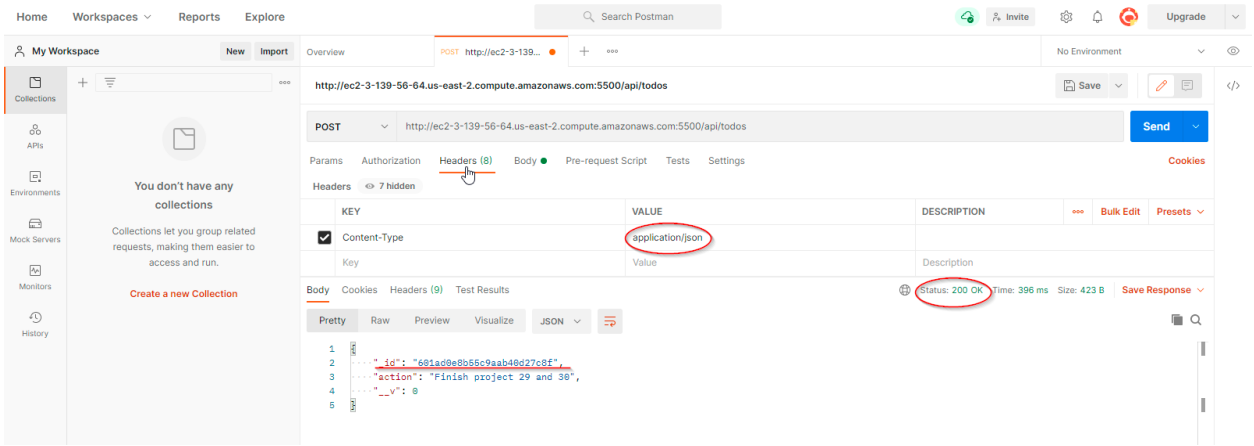
We will be using Postman to test all the API endpoints and make sure they are working. For the endpoints that require body, we will return JSON back with the necessary fields.



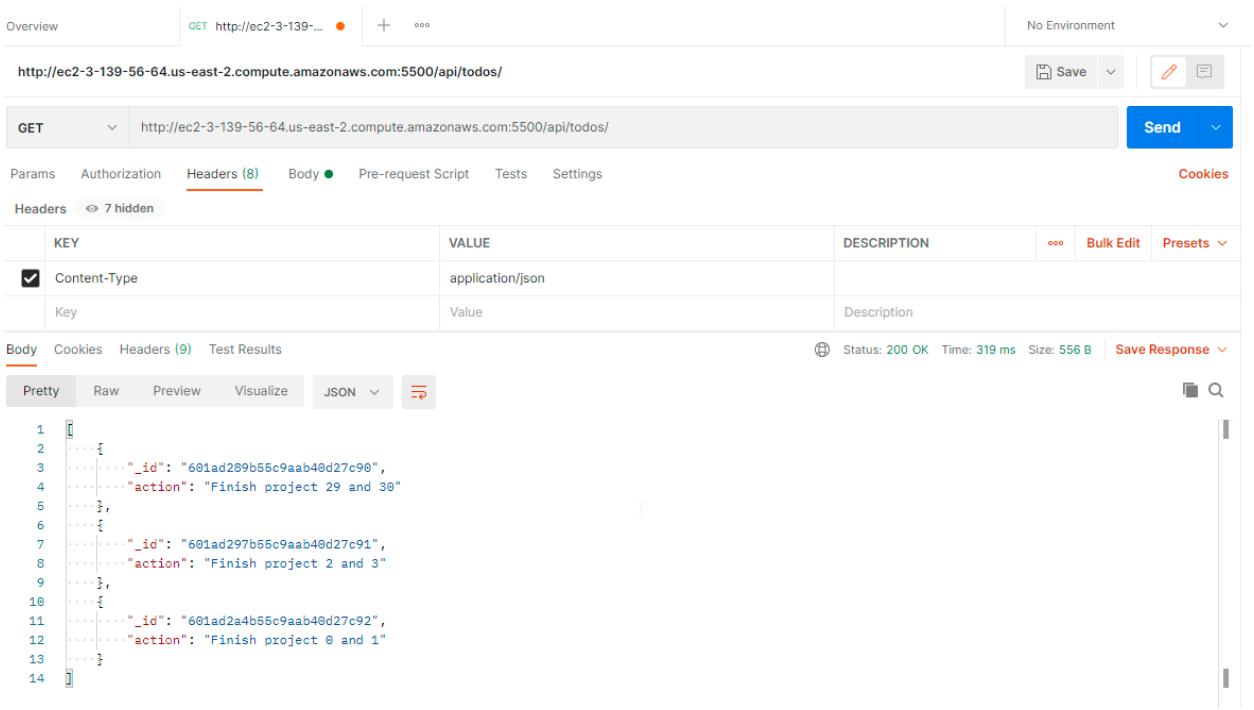
We will add a raw content to the body, with the help of the body parser to generate a new todo.

See below output:

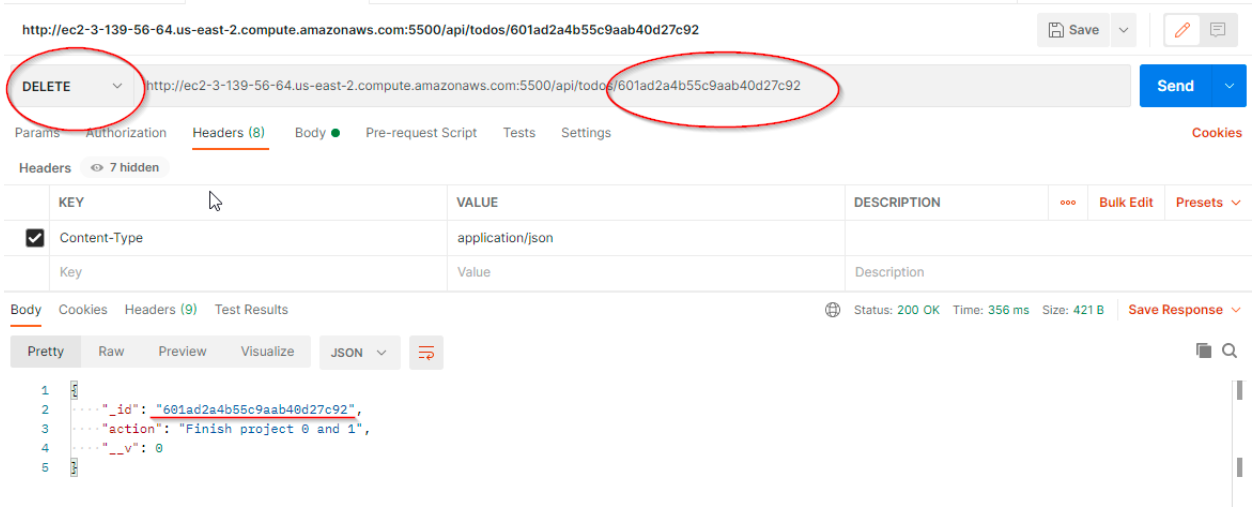
POST METHOD



GET METHOD



DELETE METHOD



Frontend creation

we are done with the functionality we want from our backend and API, it is time to create a user interface for a Web client (browser) to interact with the application via API. To start out with the frontend of the To-do app, we will use the create-react-app command to scaffold our app.

In the same root directory as your backend code, which is the Todo directory, run:

```
$ npx create-react-app client
```

This will create a new folder in your Todo directory called client, where you will add all the react code.

```
Success! Created client at /home/ubuntu/todo/client
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 client
  npm start

Happy hacking!
```

Next we run below code:

```
$ npm install concurrently --save-dev
$ npm install nodemon --save-dev
```

Now edit the package.json file with below:

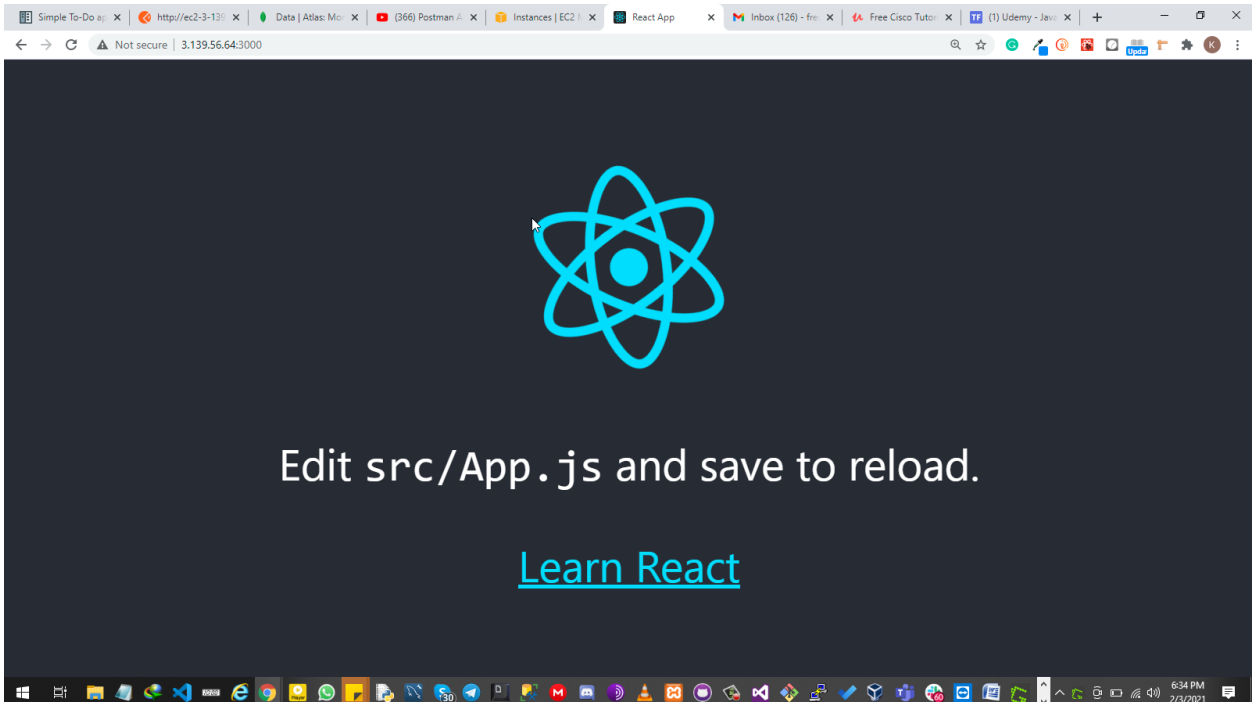
```
"scripts": {

"start": "node index.js",

"start-watch": "nodemon index.js",

"dev": "concurrently \"npm run start-watch\" \"cd client && npm start\""

},
```

Configure Proxy

```
$ cd client

$ vim package.json
```

Add the key value pair in the package.json file `"proxy": "http://localhost:5500"`

`Check the browser`



Creating your React Components

```
$ cd client

$ cd src && mkdir components && cd components

$ touch Input.js ListTodo.js Todo.js
```

Inside input.js file

```
import React, { Component } from 'react';
import axios from 'axios';

class Input extends Component {

state = {
action: ""
}

addTodo = () => {
const task = {action: this.state.action}

    if(task.action && task.action.length > 0){
      axios.post('/api/todos', task)
        .then(res => {
          if(res.data){
            this.props.getTodos();
            this.setState({action: ""})
          }
        })
        .catch(err => console.log(err))
    }else {
      console.log('input field required')
    }

}

handleChange = (e) => {
this.setState({
action: e.target.value
})
}

render() {
let { action } = this.state;
return (
<div>
<input type="text" onChange={this.handleChange}
value={action} />
<button onClick={this.addTodo}>add todo</button>
</div>
)
```

To make use of Axios, which is a Promise based HTTP client for the browser and node.js, you need to cd into your client from your terminal and run yarn add axios or npm install axios.

```
$ cd ..

$ cd ..

$ npm install axios

$ cd src/components

$ vi ListTodo.js
```

```
import React from 'react';

const ListTodo = ({ todos, deleteTodo }) => {

return (
<ul>
{
todos &&
todos.length > 0 ?
(
todos.map(todo => {
return (
<li key={todo._id} onClick={() =>
deleteTodo(todo._id)}>{todo.action}</li>
)
})
)
:
(
<li>No todo(s) left</li>
)
}
</ul>
)
}

export default ListTodo
```

```
import React, {Component} from 'react';
import axios from 'axios';

import Input from './Input';
import ListTodo from './ListTodo';

class Todo extends Component {

state = {
todos: []
}

componentDidMount(){
this.getTodos();
}

getTodos = () => {
axios.get('/api/todos')
.then(res => {
if(res.data){
this.setState({
todos: res.data
})
}
})
.catch(err => console.log(err))
}

deleteTodo = (id) => {

    axios.delete(`/api/todos/${id}`)
      .then(res => {
        if(res.data){
          this.getTodos()
        }
      })
      .catch(err => console.log(err))

}

render() {
let { todos } = this.state;

    return(
      <div>
        <h1>My Todo(s)</h1>
        <Input getTodos={this.getTodos}/>
        <ListTodo todos={todos} deleteTodo={this.deleteTodo}/>
      </div>
    )

}
}

export default Todo;
```

Delete the logo and adjust our `App.js` to look like this.

Move to the src folder

```
$ cd ..

$ vi app.js
```

```
import React from 'react';

import Todo from './components/Todo';
import './App.css';

const App = () => {
return (
<div className="App">
<Todo />
</div>
);
}

export default App;
```

In the src directory open the `App.css`

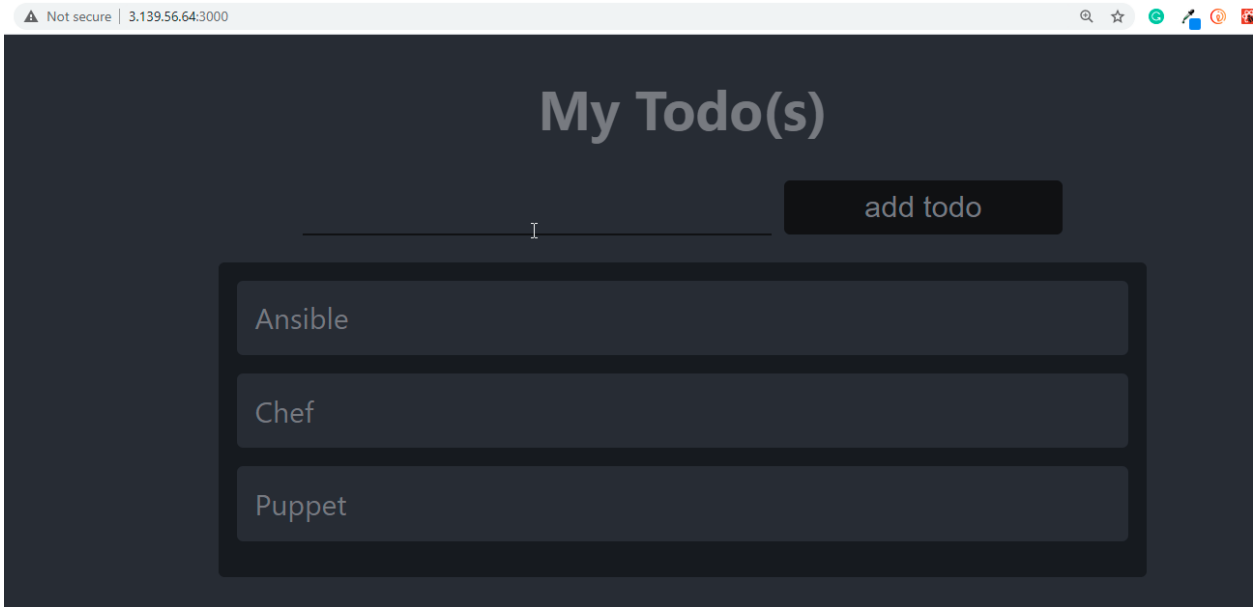
```
$ vi app.css
```

```
.App {
text-align: center;
font-size: calc(10px + 2vmin);
width: 60%;
margin-left: auto;
margin-right: auto;
}

input {
height: 40px;
width: 50%;
border: none;
border-bottom: 2px #101113 solid;
background: none;
font-size: 1.5rem;
color: #787a80;
}

input:focus {
outline: none;
}

button {
width: 25%;
height: 45px;
border: none;
margin-left: 10px;
font-size: 25px;
background: #101113;
border-radius: 5px;
color: #787a80;
cursor: pointer;
}

button:focus {
outline: none;
}

ul {
list-style: none;
text-align: left;
padding: 15px;
background: #171a1f;
border-radius: 5px;
}

li {
padding: 15px;
font-size: 1.5rem;
margin-bottom: 15px;
background: #282c34;
border-radius: 5px;
overflow-wrap: break-word;
cursor: pointer;
}

@media only screen and (min-width: 300px) {
.App {
width: 80%;
}

input {
width: 100%
}

button {
width: 100%;
margin-top: 15px;
margin-left: 0;
}
}

@media only screen and (min-width: 640px) {
.App {
width: 60%;
```

```
body {
margin: 0;
padding: 0;
font-family: -apple-system, BlinkMacSystemFont, "Segoe UI", "Roboto", "Oxygen",
"Ubuntu", "Cantarell", "Fira Sans", "Droid Sans", "Helvetica Neue",
sans-serif;
-webkit-font-smoothing: antialiased;
-moz-osx-font-smoothing: grayscale;
box-sizing: border-box;
background-color: #282c34;
color: #787a80;
}

code {
font-family: source-code-pro, Menlo, Monaco, Consolas, "Courier New",
monospace;
}
```

Edit index.css and add above code, next navigate to the todo directory and run

```
$ npm run dev
```



Then refresh and check your browser.

You should get above output when you visit the browser - simple To-Do and deployed to MERN stack

Hope this was informative.

PS: Remember to terminate your EC2 instance.