Mean Stack Deployment To Ubuntu In Aws

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

`

Now, when you have already learned how to deploy LAMP, LEMP and MERN Web stacks - it is time to get yourself familiar with MEAN stack and deploy it to Ubuntu server.

MEAN Stack is a combination of following components:

MongoDB (Document database) - Stores and allows to retrieve data.

Express (Back-end application framework) - Makes requests to Database for Reads and Writes.

Angular (Front-end application framework) - Handles Client and Server Requests

Node.js (JavaScript runtime environment) - Accepts requests and displays results to end user

All together it's called a stack, just like when you have heard some developers describe themselves as Mern stack developers.

Tldr;

#Video link

Prerequisites:

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

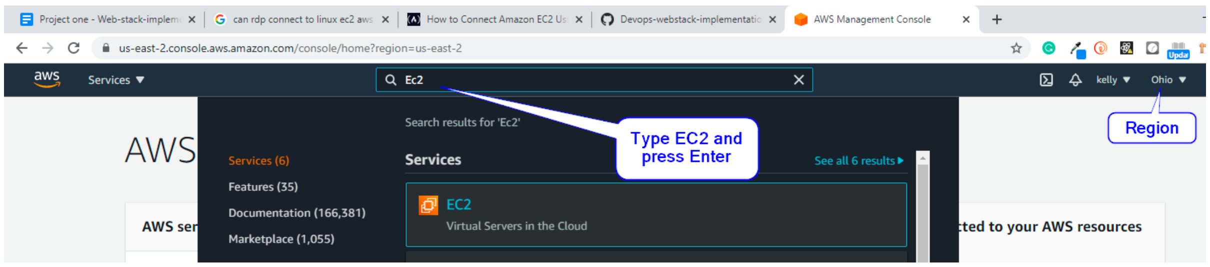
Task

We are going to implement a simple Book Register web form using MEAN stack on an AWS EC2 Ubuntu Server.

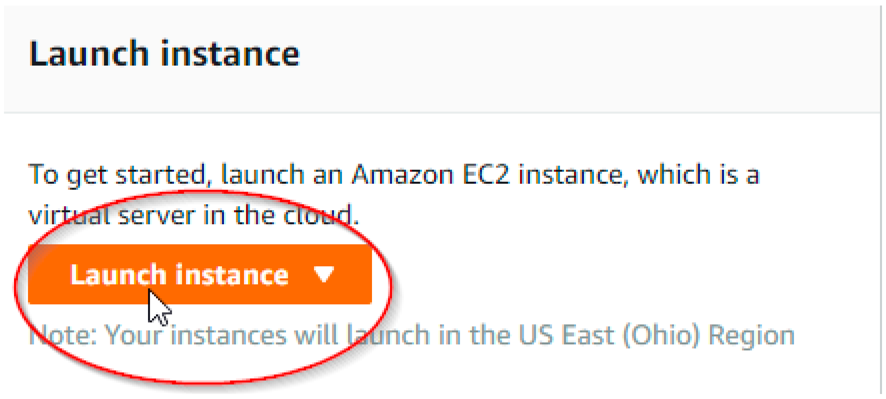
Implementation

- 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.



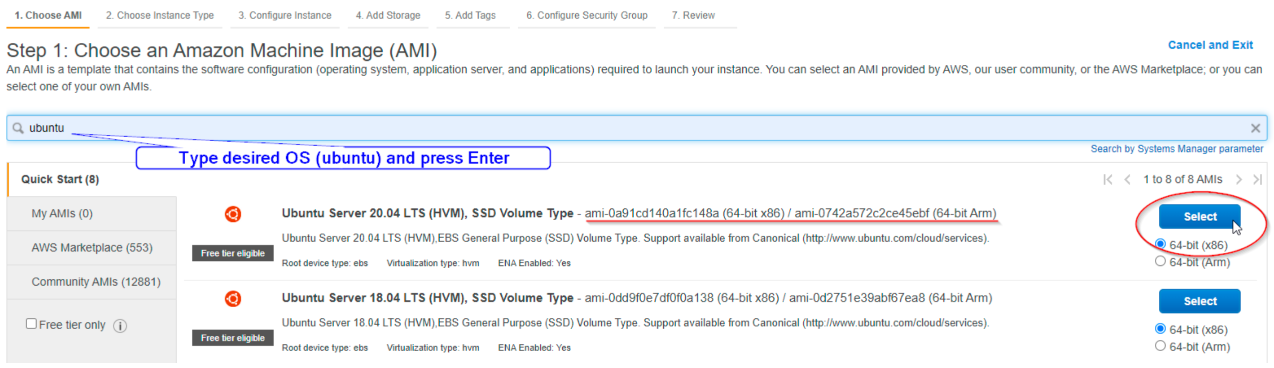
Launch instance

To get started, launch an Amazon EC2 instance, which is a virtual server in the cloud.

Note: Your instances will launch in the US East (Ohio) Region

- 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



Step 1: Choose an Amazon Machine Image (AMI)

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. You can select an AMI provided by AWS, our user community, or the AWS Marketplace; or you can select one of your own AMIs.

ubuntu

Type desired OS (ubuntu) and press Enter

Ubuntu Server 20.04 LTS (HVM), SSD Volume Type - ami-0a91cd140a1fc148a (64-bit x86) / ami-0742a572c2ce45ebf (64-bit Arm)

Ubuntu Server 20.04 LTS (HVM),EBS General Purpose (SSD) Volume Type. Support available from Canonical (http://www.ubuntu.com/cloud/services).

Free tier eligible

Root device type: ebs Virtualization type: hvm ENA Enabled: Yes

Select [✓ 64-bit (x86)] 64-bit (Arm)

Ubuntu Server 18.04 LTS (HVM), SSD Volume Type - ami-0dd9f0e7df0f0a138 (64-bit x86) / ami-0d2751e39abf67ea8 (64-bit Arm)

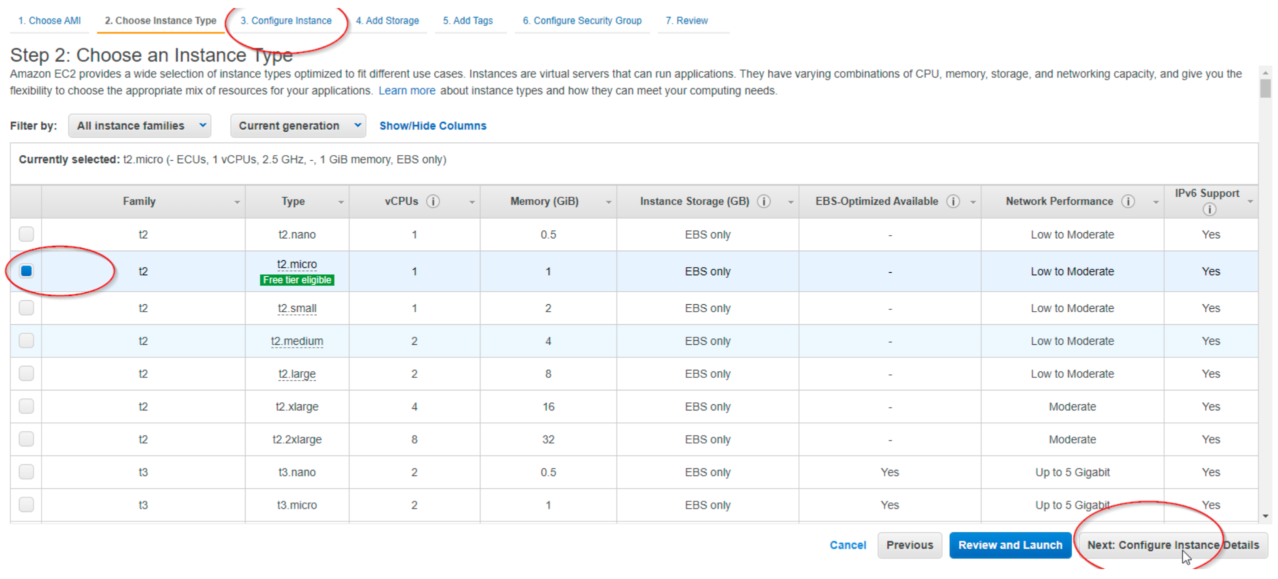
Ubuntu Server 18.04 LTS (HVM),EBS General Purpose (SSD) Volume Type. Support available from Canonical (http://www.ubuntu.com/cloud/services).

Free tier eligible

Root device type: ebs Virtualization type: hvm ENA Enabled: Yes

Select [✓ 64-bit (x86)] 64-bit (Arm)

- 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.



Step 2: Choose an Instance Type

Amazon EC2 provides a wide selection of instance types optimized to fit different use cases. Instances are virtual servers that can run applications. They have varying combinations of CPU, memory, storage, and networking capacity, and give you the flexibility to choose the appropriate mix of resources for your applications. Learn more about instance types and how they can meet your computing needs.

Filter by: All instance families | Current generation | Show/Hide Columns

Currently selected: t2.micro (- ECUs, 1 vCPUs, 2.5 GHz, -, 1 GiB memory, EBS only)

	Family	Type	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance	IPv6 Support
☐	t2	t2.nano	1	0.5	EBS only	-	Low to Moderate	Yes
☑	t2	t2.micro Free tier eligible	1	1	EBS only	-	Low to Moderate	Yes
☐	t2	t2.small	1	2	EBS only	-	Low to Moderate	Yes
☐	t2	t2.medium	2	4	EBS only	-	Low to Moderate	Yes
☐	t2	t2.large	2	8	EBS only	-	Low to Moderate	Yes
☐	t2	t2.xlarge	4	16	EBS only	-	Moderate	Yes
☐	t2	t2.2xlarge	8	32	EBS only	-	Moderate	Yes
☐	t3	t3.nano	2	0.5	EBS only	Yes	Up to 5 Gigabit	Yes
☐	t3	t3.micro	2	1	EBS only	Yes	Up to 5 Gigabit	Yes

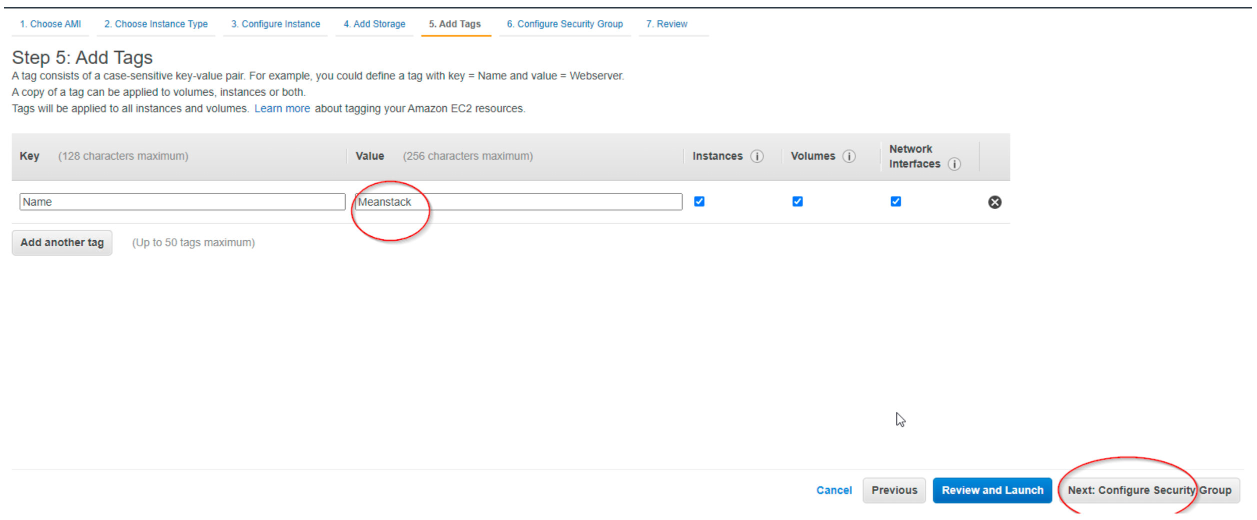
Move to next step

- 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 -y
```

- 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.



Step 5: Add Tags
A tag consists of a case-sensitive key-value pair. For example, you could define a tag with key = Name and value = Webserver.
A copy of a tag can be applied to volumes, instances or both.
Tags will be applied to all instances and volumes. Learn more about tagging your Amazon EC2 resources.

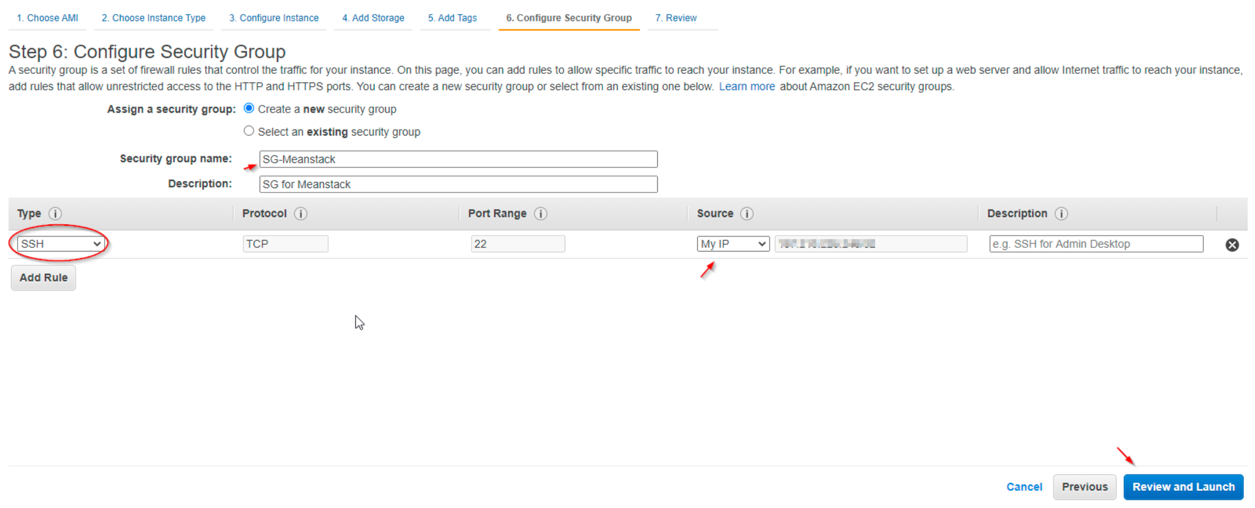
Key (128 characters maximum)	Value (256 characters maximum)	Instances	Volumes	Network Interfaces
Name	Meanstack	✓	✓	✓

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,

You may add descriptions on the last column.


1. Choose AMI | 2. Choose Instance Type | 3. Configure Instance | 4. Add Storage | 5. Add Tags | 6. Configure Security Group | 7. Review

Step 6: Configure Security Group

A security group is a set of firewall rules that control the traffic for your instance. On this page, you can add rules to allow specific traffic to reach your instance. For example, if you want to set up a web server and allow Internet traffic to reach your instance, add rules that allow unrestricted access to the HTTP and HTTPS ports. You can create a new security group or select from an existing one below. Learn more about Amazon EC2 security groups.

Assign a security group: [✓ Create a **new** security group] Select an **existing** security group

Security group name: SG-Meanstack

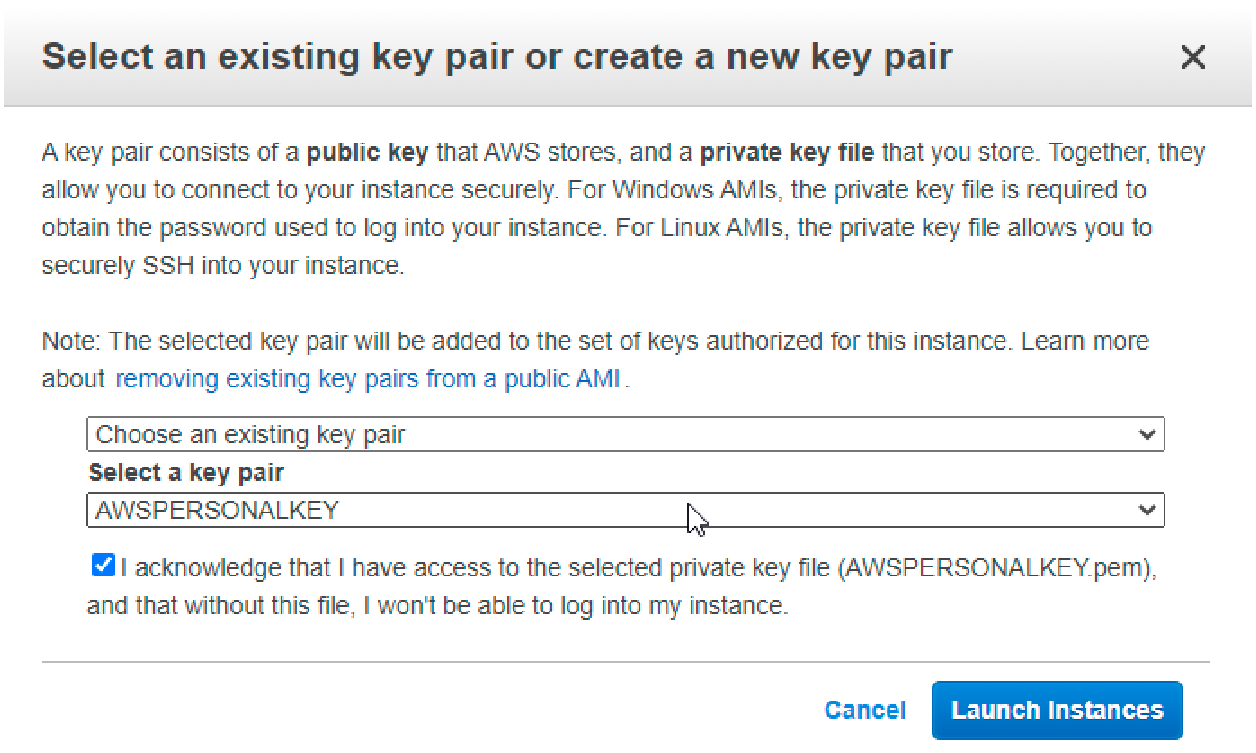
Description: SG for Meanstack

Type	Protocol	Port Range	Source	Description
SSH	TCP	22	My IP [redacted]	e.g. SSH for Admin Desktop

- 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.

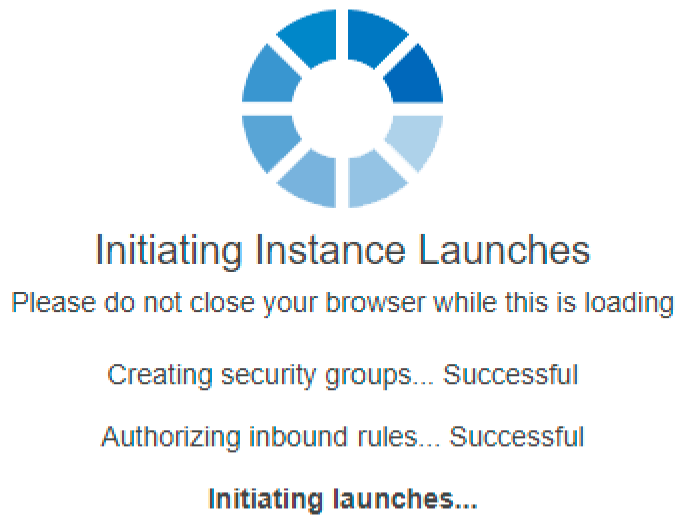
Note: The selected key pair will be added to the set of keys authorized for this instance. Learn more about removing existing key pairs from a public AMI .

Choose an existing key pair

Select a key pair

AWSPERSONALKEY

✓ I acknowledge that I have access to the selected private key file (AWSPERSONALKEY.pem), and that without this file, I won't be able to log into my instance.


Initiating Instance Launches

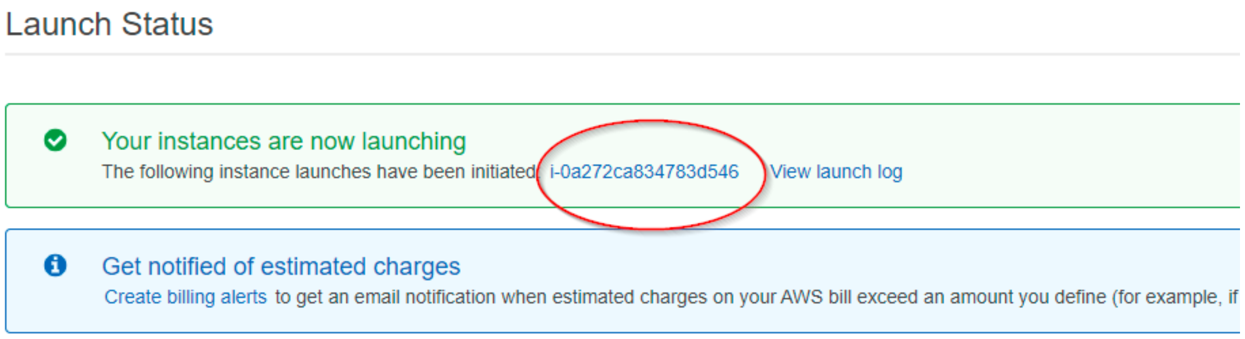
Please do not close your browser while this is loading

Creating security groups... Successful

Authorizing inbound rules... Successful

Initiating launches...

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

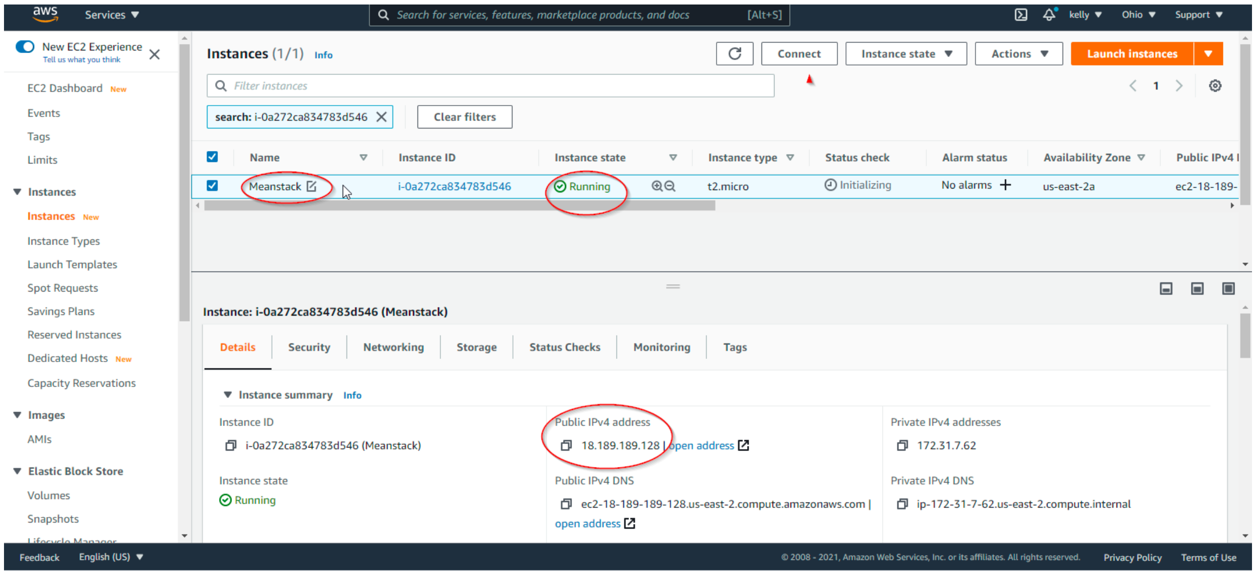


Launch Status

Your instances are now launching
The following instance launches have been initiated: i-0a272ca834783d546 View launch log

Get notified of estimated charges
Create billing alerts to get an email notification when estimated charges on your AWS bill exceed an amount you define (for example, if

Welcome to the EC2 Dashboard



Instances (1/1) Info

search: i-0a272ca834783d546

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4
Meanstack	i-0a272ca834783d546	Running	t2.micro	Initializing	No alarms	us-east-2a	ec2-18-189-

Instance: i-0a272ca834783d546 (Meanstack)

Instance summary Info

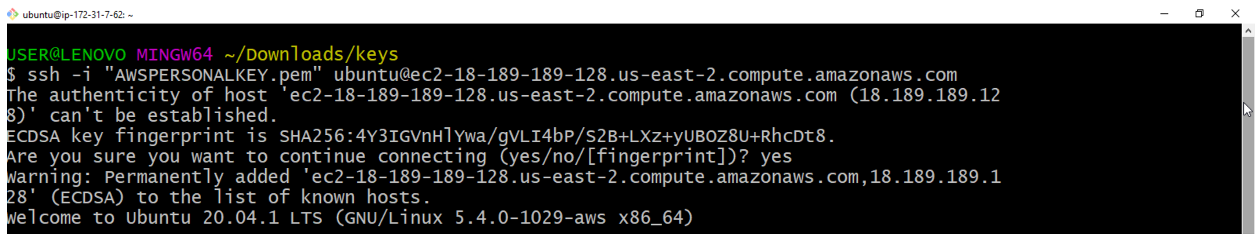
Instance ID	Public IPv4 address	Private IPv4 addresses
i-0a272ca834783d546 (Meanstack)	18.189.189.128 | open address	172.31.7.62
Instance state	**Public IPv4 DNS**	**Private IPv4 DNS**
Running	ec2-18-189-189-128.us-east-2.compute.amazonaws.com | open address	ip-172-31-7-62.us-east-2.compute.internal

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

Yay!!!

Open git bash or putty or mobaxterm, whichever console is suitable, else download.

We are using git bash here:



```
USER@LENOVO MINGW64 ~/Downloads/keys
$ ssh -i "AWSPERSONALKEY.pem" ubuntu@ec2-18-189-189-128.us-east-2.compute.amazonaws.com
The authenticity of host 'ec2-18-189-189-128.us-east-2.compute.amazonaws.com (18.189.189.12
8)' can't be established.
ECDSA key fingerprint is SHA256:4Y3IGVnHlYwa/gVLI4bP/S2B+LXz+yUBOZ8U+RhcDt8.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-18-189-189-128.us-east-2.compute.amazonaws.com,18.189.189.1
28' (ECDSA) to the list of known hosts.
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-1029-aws x86_64)
```

Type YES, to connect.

You have now connected to the EC2 instance via SSH

Type clear, to have a neat console and proceed, You could run below command to get your public IP also.

Type: curl http://169.254.169.254/latest/user-data

By default EC2 user is given sudo privilege

Great!

Now we have Nodejs installed, which can be confirmed by running **(node -v),** So let's continue with other installations.

Step 2: Install MongoDB

MongoDB stores data in flexible, JSON-like documents. Fields in a database can vary from document to document and data structure can be changed over time. For our example application, we are adding book records to MongoDB that contain book name, isbn number, author, and number of pages.

Run this code:

```
$  sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv
0C49F3730359A14518585931BC711F9BA15703C6


$  echo "deb [ arch=amd64 ] https://repo.mongodb.org/apt/ubuntu trusty/mongodb-
org/3.4 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-3.4.list
```

```
$  sudo apt install -y mongodb

$  sudo service mongodb start

$  sudo systemctl status mongodb
```

Next we will need to install a Node package manager to help install a Body-parser package to help with our app.

```
$  sudo apt install -y npm

$  sudo npm install body-parser
```

Next, we need to Create a folder which will be our app name - ‘Books’

.

Run this code:

```
$ mkdir Books && cd Books

$ vi server.js
```

Type below code into the server.js file created with vi or vim editor:

```
var express = require('express');
var bodyParser = require('body-parser');
var app = express();
app.use(express.static(__dirname + '/public'));
app.use(bodyParser.json());
require('./apps/routes')(app);
app.set('port', 3300);
app.listen(app.get('port'), function() {
    console.log('Server up: http://localhost:' +
app.get('port'));
});
```

Step 3: Install Express and set up routes to the server

Express is a minimal and flexible Node.js web application framework that provides features for web and mobile applications. We will use Express in to pass book information to and from our MongoDB database.

We also will use Mongoose package which provides a straight-forward, schema-based solution to model your application data. We will use Mongoose to establish a schema for the database to store data of our book register.

Type exit to below code to install the requirements:

```
$  sudo npm install express mongoose

Then

$  mkdir apps && cd apps

$  vi routes.js
```

From the new Apps directory created above, we will add below content in the routes.js file

```
var Book = require('./models/book');
module.exports = function(app) {
  app.get('/book', function(req, res) {
    Book.find({}, function(err, result) {
      if ( err ) throw err;
      res.json(result);
    });
  });
  app.post('/book', function(req, res) {
    var book = new Book( {
      name:req.body.name,
      isbn:req.body.isbn,
      author:req.body.author,
      pages:req.body.pages
    });
    book.save(function(err, result) {
      if ( err ) throw err;
      res.json( {
        message:"Successfully added book",
        book:result
      });
    });
  });
  app.delete("/book/:isbn", function(req, res) {
    Book.findOneAndRemove(req.query, function(err, result) {
      if ( err ) throw err;
      res.json( {
        message: "Successfully deleted the book",
        book: result
      });
    });
  });
  var path = require('path');
  app.get('*', function(req, res) {
    res.sendfile(path.join(__dirname + '/public', 'index.html'));
  });
};
```

In the ‘apps’ folder, create a folder named **models**

```
$  mkdir models && cd models

$  vi book.js
```

Add below scripts to the **book.js** file created

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

var dbHost = 'mongodb://localhost:27017/test';

mongoose.connect(dbHost);

mongoose.connection;

mongoose.set('debug', true);

var bookSchema = mongoose.Schema( {

  name: String,

  isbn: {type: String, index: true},

  author: String,

  pages: Number

});

var Book = mongoose.model('Book', bookSchema);

module.exports = mongoose.model('Book', bookSchema);
```

Step 4 - Access the routes with AngularJS

AngularJS provides a web framework for creating dynamic views in your web applications. In this tutorial, we use AngularJS to connect our web page with Express and perform actions on our book register.

Navigate back to the ‘Books’ directory and create a new folder, then add a file - **script.js**

Use this Command

```
$  cd ../..

$  mkdir public && cd public

$  vim script.js
```

In the Vim editor, type "i" to enter the insert mode.

Enter this code below:

```
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope, $http) {
  $http( {
    method: 'GET',
    url: '/book'
  }).then(function successCallback(response) {
    $scope.books = response.data;
  }, function errorCallback(response) {
    console.log('Error: ' + response);
  });
  $scope.del_book = function(book) {
    $http( {
      method: 'DELETE',
      url: '/book/:isbn',
      params: {'isbn': book.isbn}
    }).then(function successCallback(response) {
      console.log(response);
    }, function errorCallback(response) {
      console.log('Error: ' + response);
    });
  };
  $scope.add_book = function() {
    var body = '{ "name": "' + $scope.Name +
    '", "isbn": "' + $scope.Isbn +
    '", "author": "' + $scope.Author +
    '", "pages": "' + $scope.Pages + '" }';
    $http({
      method: 'POST',
      url: '/book',
      data: body
    }).then(function successCallback(response) {
      console.log(response);
    }, function errorCallback(response) {
      console.log('Error: ' + response);
    });
  };
});
```

Then press ESC and exit with " :wq " command

Next, In the 'public' folder, create a file named `index.html`

```
$  vi index.html
```

Then add below code into the new index file create.

```
<!doctype html>
<html ng-app="myApp" ng-controller="myCtrl">
  <head>
    <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.4/angular.min.js"></script>
    <script src="script.js"></script>
  </head>
  <body>
    <div>
      <table>
        <tr>
          <td>Name:</td>
          <td><input type="text" ng-model="Name"></td>
        </tr>
        <tr>
          <td>Isbn:</td>
          <td><input type="text" ng-model="Isbn"></td>
        </tr>
        <tr>
          <td>Author:</td>
          <td><input type="text" ng-model="Author"></td>
        </tr>
        <tr>
          <td>Pages:</td>
          <td><input type="number" ng-model="Pages"></td>
        </tr>
      </table>
      <button ng-click="add_book()">Add</button>
    </div>
    <hr>
    <div>
      <table>
        <tr>
          <th>Name</th>
          <th>Isbn</th>
          <th>Author</th>
          <th>Pages</th>

        </tr>
        <tr ng-repeat="book in books">
          <td>{{book.name}}</td>
          <td>{{book.isbn}}</td>
          <td>{{book.author}}</td>
          <td>{{book.pages}}</td>

          <td><input type="button" value="Delete" data-ng-click="del_book(book)"></td>
        </tr>
      </table>
    </div>
  </body>
</html>
```

Good job, Now navigate back to "Books" folder.

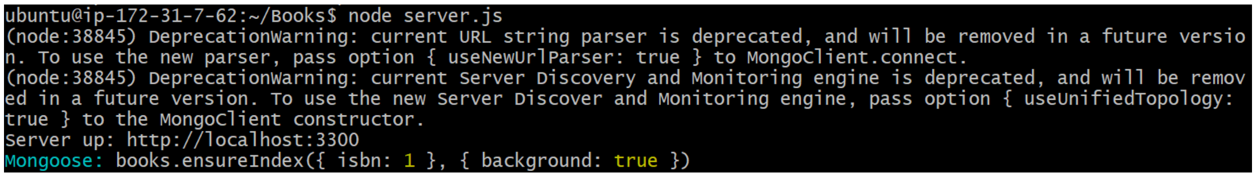
```
$ cd ..
```

Now what right?

We are done! let's ask nodejs to serve us some good content.

```
$ node server.js
```

The server is now up and running, we can connect it via port 3300.

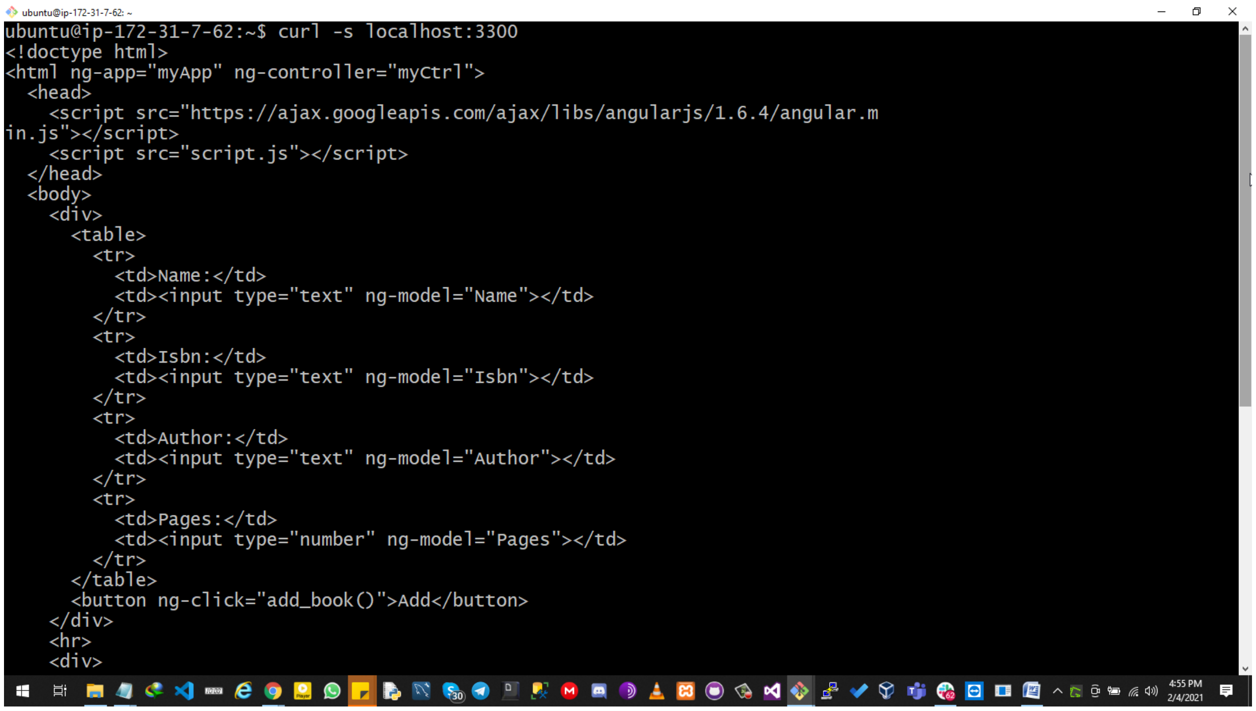


```
ubuntu@ip-172-31-7-62:~/Books$ node server.js
(node:38845) DeprecationWarning: current URL string parser is deprecated, and will be removed in a future versio
n. To use the new parser, pass option { useNewUrlParser: true } to MongoClient.connect.
(node:38845) DeprecationWarning: current Server Discovery and Monitoring engine is deprecated, and will be remov
ed in a future version. To use the new Server Discover and Monitoring engine, pass option { useUnifiedTopology:
true } to the MongoClient constructor.
Server up: http://localhost:3300
Mongoose: books.ensureIndex({ isbn: 1 }, { background: true })
```

Test locally with this:

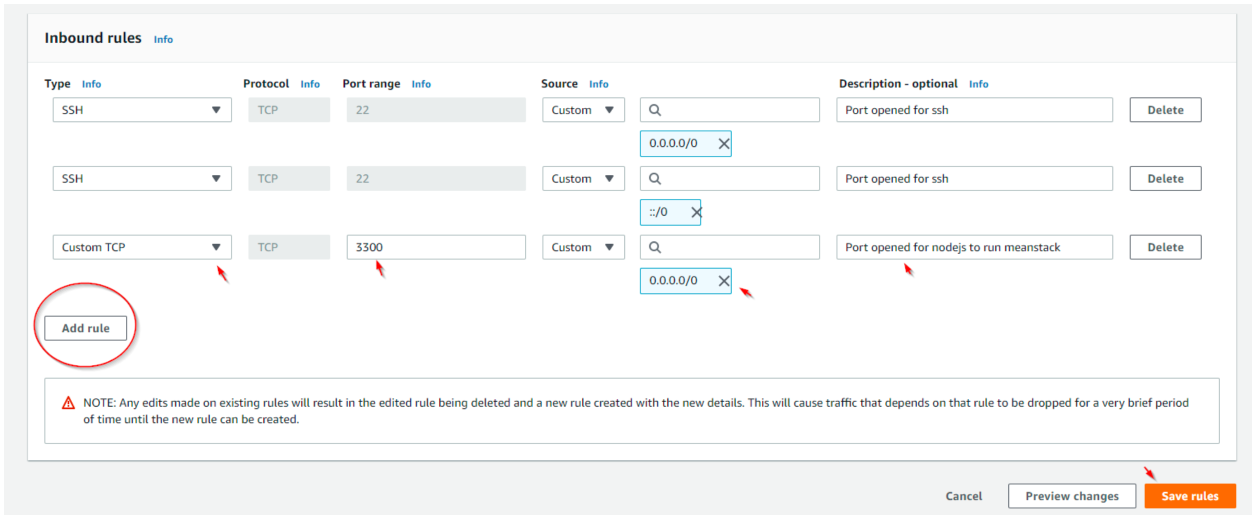
```
$ curl -s http://localhost:3300
```

I guess you like it.



```
ubuntu@ip-172-31-7-62:~$ curl -s localhost:3300
<!doctype html>
<html ng-app="myApp" ng-controller="myCtrl">
  <head>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.4/angular.m
in.js"></script>
    <script src="script.js"></script>
  </head>
  <body>
    <div>
      <table>
        <tr>
          <td>Name:</td>
          <td><input type="text" ng-model="Name"></td>
        </tr>
        <tr>
          <td>Isbn:</td>
          <td><input type="text" ng-model="Isbn"></td>
        </tr>
        <tr>
          <td>Author:</td>
          <td><input type="text" ng-model="Author"></td>
        </tr>
        <tr>
          <td>Pages:</td>
          <td><input type="number" ng-model="Pages"></td>
        </tr>
      </table>
      <button ng-click="add_book()">Add</button>
    </div>
    <hr>
    <div>
```

So to display the result via the browser we need AWS to allow us to access that port through the Security Group.

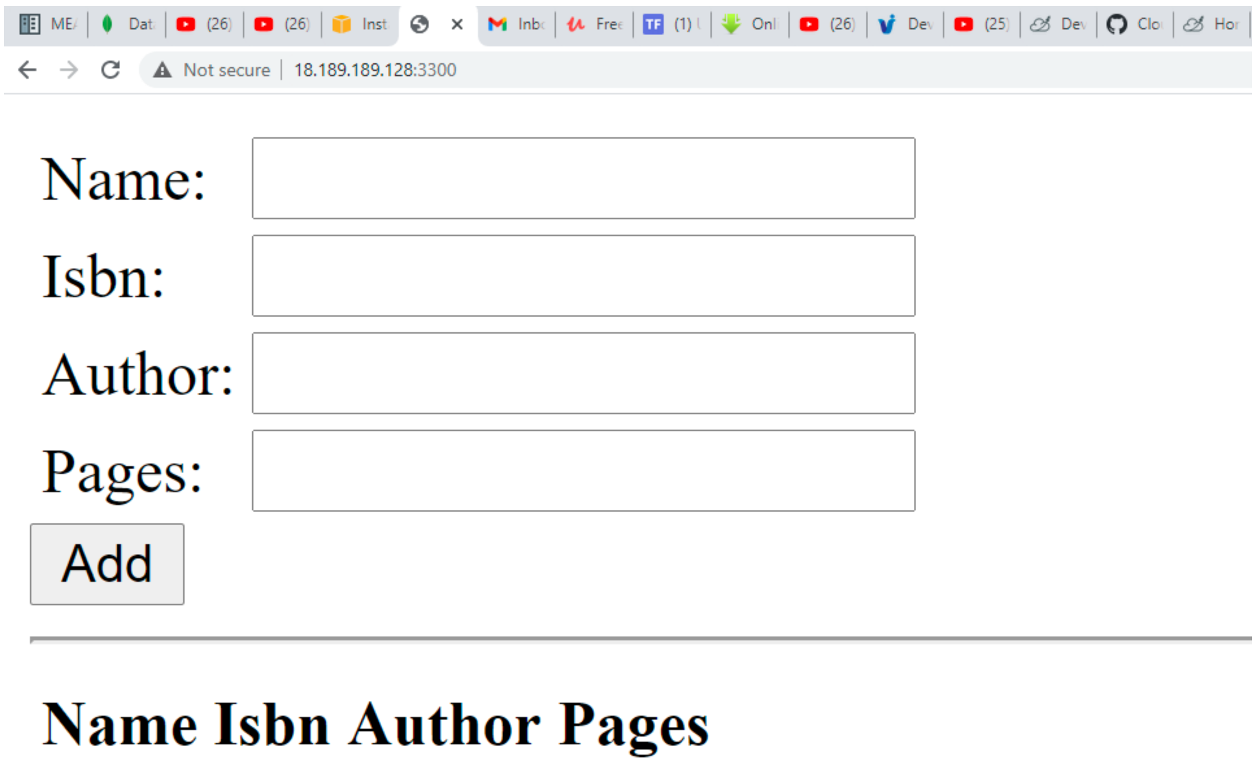


Inbound rules Info

Type	Protocol	Port range	Source		Description - optional
SSH	TCP	22	Custom	0.0.0.0/0	Port opened for ssh
SSH	TCP	22	Custom	::/0	Port opened for ssh
Custom TCP	TCP	3300	Custom	0.0.0.0/0	Port opened for nodejs to run meanstack

NOTE: Any edits made on existing rules will result in the edited rule being deleted and a new rule created with the new details. This will cause traffic that depends on that rule to be dropped for a very brief period of time until the new rule can be created.

Good, Now you can access our Book Register web application from the Internet with a browser using Public IP address or Public DNS name.


Not secure | 18.189.189.128:3300

Name:

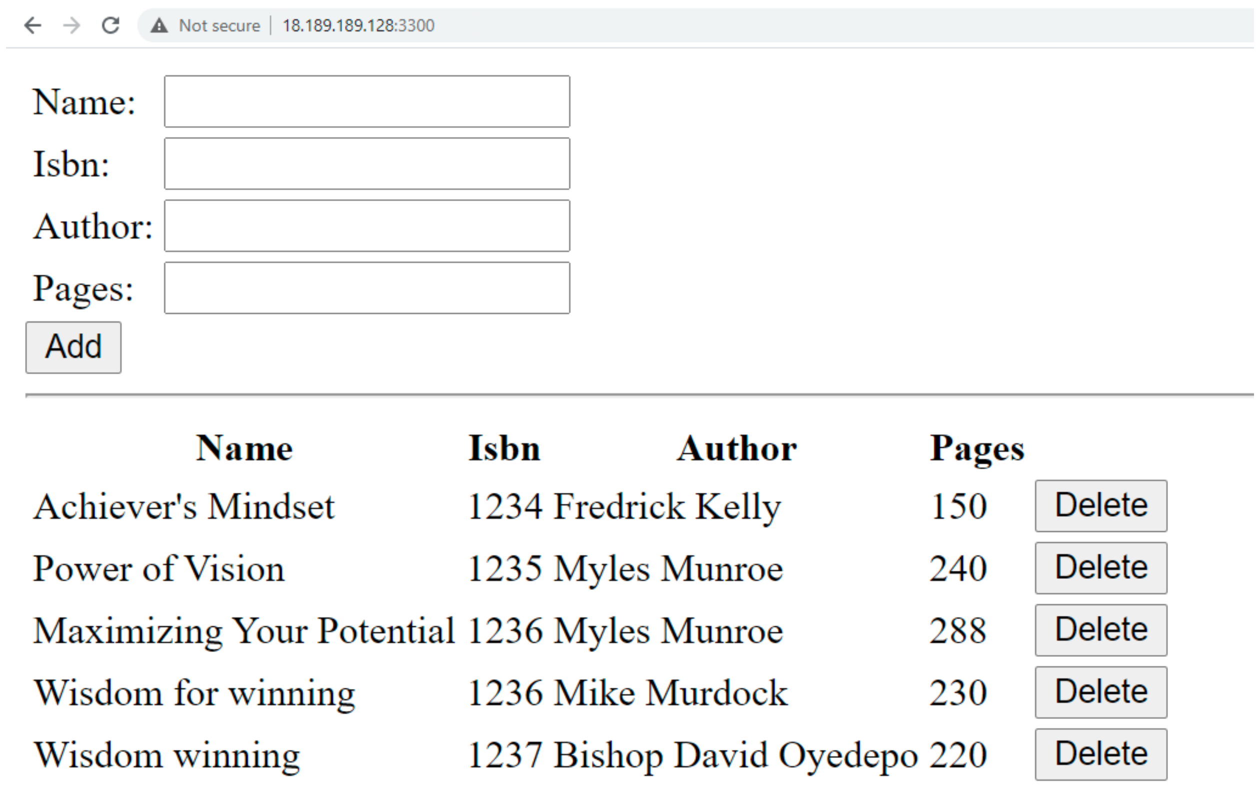
Isbn:

Author:

Pages:

Name Isbn Author Pages

When you add dummy data to the input fields, and hit Add it should populate in columns, as shown below



Name	Isbn	Author	Pages	
Achiever's Mindset	1234	Fredrick Kelly	150	Delete
Power of Vision	1235	Myles Munroe	240	Delete
Maximizing Your Potential	1236	Myles Munroe	288	Delete
Wisdom for winning	1236	Mike Murdock	230	Delete
Wisdom winning	1237	Bishop David Oyedepo	220	Delete

The Hola, it's done. Our Mean Stack App is ready to be shipped across the continent.

Thank you, this is the minimum requirement to set up an AWS instance with Mean Stack for a web project.

Hope this was informative.

PS: Remember to terminate your EC2 instance.