

AWS & DevOps Report

THE NATIONAL INSTITUTE OF ENGINEERING, MYSURU

(An Autonomous Institute under VTU, Belagavi)



ESTD : 1946

In partial fulfilment of the requirements for the award of degree of

**Bachelor of Engineering in
Computer Science and Engineering**

Submitted by

Aayushi Mishra (4NI19CS002)

Under the guidance of

Thamotharan N
DevOps Lead Architect
Solvendo

B R Vatsala
Assistant Professor
Dept. of CS&E
NIE, Mysuru



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
THE NATIONAL INSTITUTE OF ENGINEERING**

Mysore-570 008
2022-2023

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
THE NATIONAL INSTITUTE OF ENGINEERING



CERTIFICATE

This is to certify that the work carried out by **Aayushi Mishra (4NI19CS002)** in partial fulfilment of the requirements for the completion of the course “AWS & DevOps” in the semester VII, Department of Computer Science and Engineering, as per the academic regulations of The National Institute of Engineering, Mysuru, during the academic year 2022-23. It is certified that all corrections and suggestions indicated for the Internal Assessment have been incorporated in the report deposited in the department library.

Signature of the guide

B R Vatsala
Assistant Professor
Dept of CS&E
NIE, Mysuru

Table of Contents

Sl. No.	Content	Page No.
1	Introduction	1
2	Deploying a Node.js Application on AWS EC2 Server	2
3	Screenshot of Output	7

List of Figures

Figure No.	Description	Page No.
1	AWS Services	2
2	EC2 Dashboard	2
3	Launch an instance	3
4	Contents of node-hello directory	4
5	Starting the node.js app	4
6	Browser connection timeout	5
7	Security Groups	5
8	List of running applications	6
9	Output of deployed node.js web application	7

Introduction

AWS:

AWS stands for **Amazon Web Services**. The AWS service is provided by the Amazon that uses distributed IT infrastructure to provide different IT resources available on demand. It provides different services such as infrastructure as a service (IaaS), platform as a service (PaaS) and packaged software as a service (SaaS). Amazon launched AWS, a cloud computing platform to allow the different organizations to take advantage of reliable IT infrastructure. A small manufacturing organization uses their expertise to expand their business by leaving their IT management to the AWS. AWS provides services to customers when required without any prior commitment or upfront investment. Pay-As-You-Go enables the customers to procure services from AWS.

EC2:

EC2 stands for Elastic Compute Cloud. EC2 is on-demand computing service on the AWS cloud platform. Under computing, it includes all the services a computing device can offer to you along with the flexibility of a virtual environment. It also allows the user to configure their instances as per their requirements i.e., allocate the RAM, ROM, and storage according to the need of the current task. Even the user can dismantle the virtual device once its task is completed and it is no more required. For providing, all these scalable resources AWS charges some bill amount at the end of every month, bill amount is entirely dependent on your usage. EC2 provides you to rent virtual computers. The provision of servers on AWS Cloud is one of the easiest way in EC2. EC2 has resizable capacity. EC2 offers security, reliability, high-performance and cost-effective infrastructure so as to meet the demanding business needs.

Node JS:

NodeJS is a runtime for JavaScript, that is built on top of google chrome V8 engine which is writing in C++. NodeJS is an open source, cross-platform runtime environment for developing server-side and networking applications. NodeJS applications are written in JavaScript, and can be run within the NodeJS runtime on OS X, Microsoft Windows, and Linux.

NodeJS also provides a rich library of various JavaScript modules which simplifies the development of web applications using NodeJS to a great extent.

Following are the areas where NodeJS is proving itself as a perfect technology partner.

- I/O bound Applications

- Data Streaming Applications
- Data Intensive Real-time Applications (DIRT)
- JSON APIs based Applications
- Single Page Applications

Deploying a Node.js application on AWS EC2 server

Steps to be followed:

1. Create account on AWS:

Go to “<https://www.aws.amazon.com>” and create a new account on AWS. If you have already registered on AWS then go to AWS console and sign-in as root user and then enter root email and password.

2. Launch an EC2 instance:

Once you are logged in you can see all the aws services under the **Services** section as shown in Figure 1.

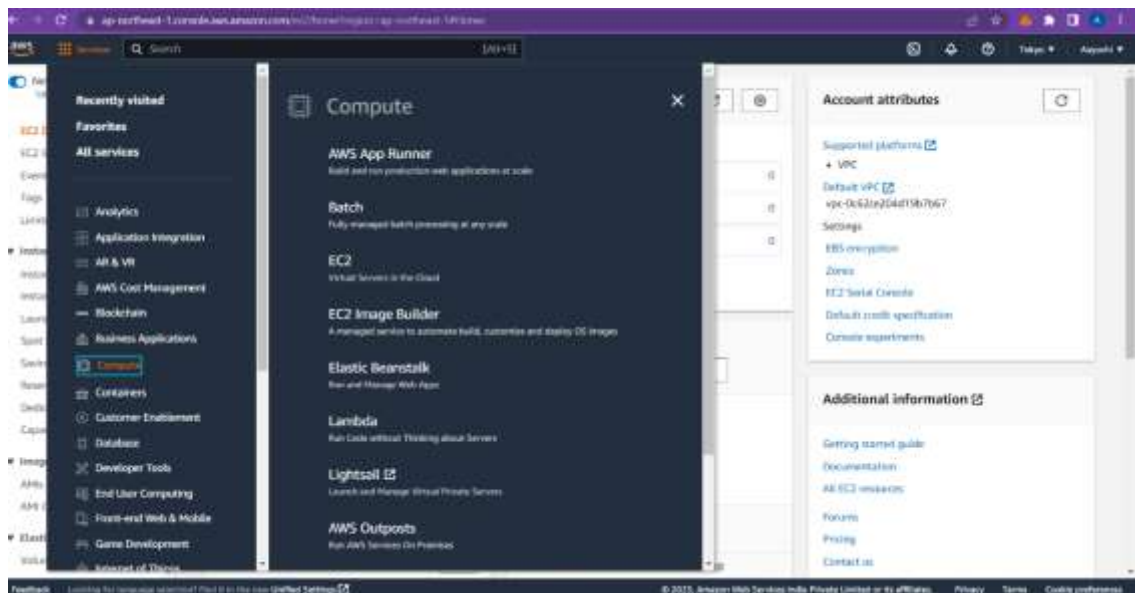


Figure 1: AWS Services

Click on the **EC2** Service, it will show us EC2 Dashboard page, as shown in Figure 2, click on **Launch Instance**.

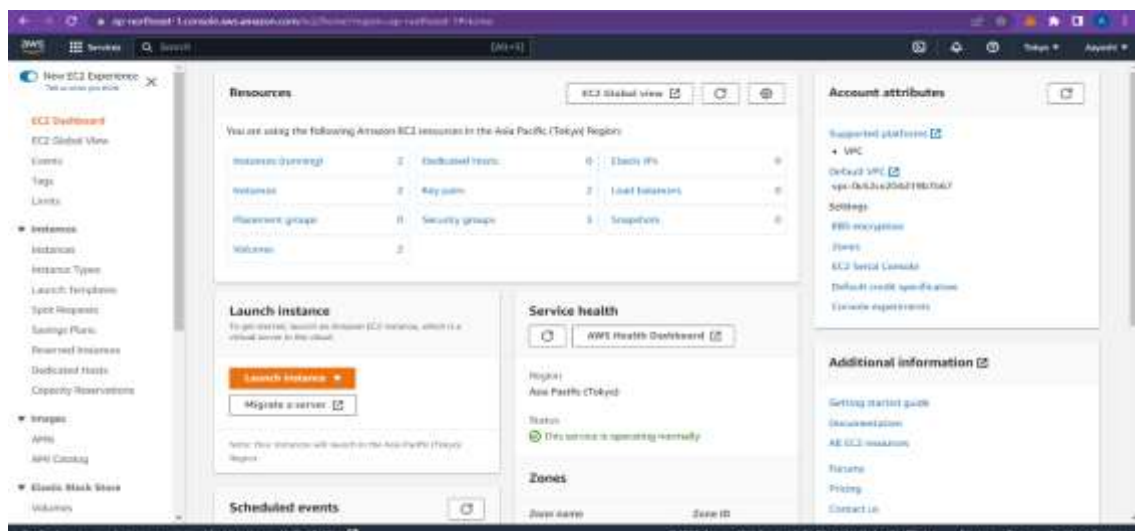


Figure 2: EC2 Dashboard

In **Launch an instance** page, as shown in Figure 3, enter name as “aws tutorial” under **Name and tags**. We have to choose **Amazon Machine Image** for launching our instance. Amazon provides various types of images.

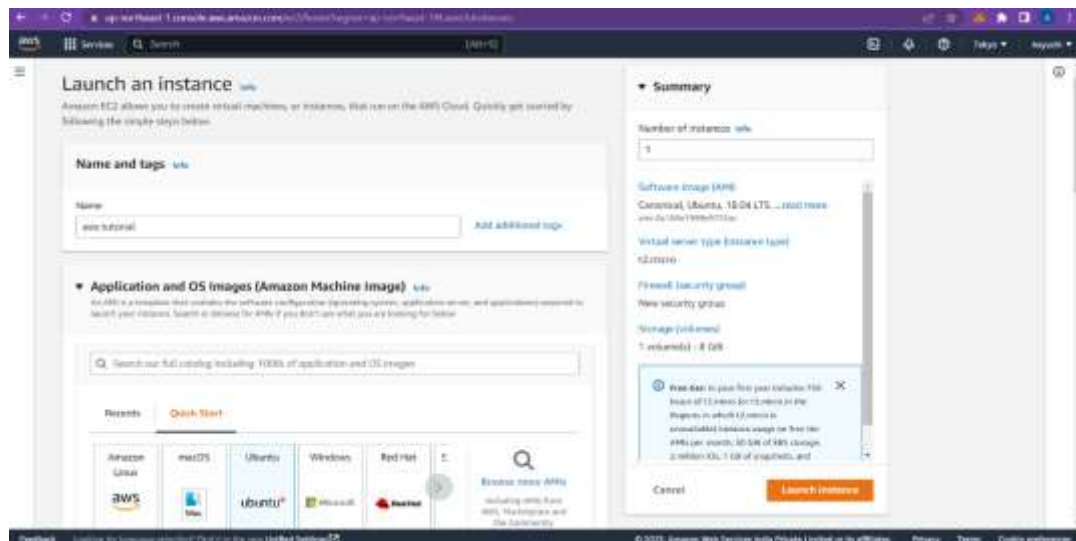


Figure 3: Launch an instance

Create a new key pair and then click on **Launch Instance** button.

3. SSH into your instance:

So, we have successfully launched our EC2 server. Now we will access the server from our local machine using putty on Windows. If you are using Windows you need to change your .pem key to .ppk format.

To convert .pem file to .ppk file, download puttygen.exe. Once downloaded open the PuTTY Key Generator and load the downloaded the .pem file and then click on save private key. Now open the putty and enter the public IP of the instance created. Select SSH, then click on Auth and then browse the saved .ppk file and click on open. Login as ec2-user.

4. Install Node.js:

Install Node.js on putty using the following commands-

- `sudo -i`
- `yum install gcc-c++ make`
- `curl -sL https://rpm.nodesource.com/setup_16.x | sudo -E bash -`
- `yum install nodejs`

5. Install Git and clone repository from GitHub:

Use the following commands to install git and for cloning repository from GitHub-

- `yum install git`
- `git clone https://github.com/johnpapa/node-hello.git`

After cloning the repository, run `cd node-hello` and see all the content of the folder using the `ls` command as shown in Figure 4.

```
[root@ip-172-31-44-226 ~]# cd node-hello
[root@ip-172-31-44-226 node-hello]# ls
index.js  package.json  package-lock.json  README.md
```

Figure 4: Contents of node-hello directory

6. Start the node.js app:

We have successfully cloned the app on our server. Run the commands as shown in Figure 5, inside our project directory:

- `npm install`

It will install all the required packages for the app. There are several ways to start our app but we will use simple command:

- `node index.js`

```
[root@ip-172-31-44-226 node-hello]# npm install
npm WARN ancient lockfile
npm WARN ancient lockfile The package-lock.json file was created with an old ver
sion of npm,
npm WARN ancient lockfile so supplemental metadata must be fetched from the regi
stry.
npm WARN ancient lockfile
npm WARN ancient lockfile This is a one-time fix-up, please be patient...
npm WARN ancient lockfile

up to date, audited 1 package in 215ms

found 0 vulnerabilities
[root@ip-172-31-44-226 node-hello]# node index.js
Server running on http://localhost:3000/
```

Figure 5: Starting the node.js app

Copy public DNS from your instances page –

“<http://ec2-54-199-74-186.ap-northeast-1.compute.amazonaws.com>”, paste it on your web browser, and append port :3000 at the end of the DNS address. As you will see this will not work and there is nothing to show on browser and the browser connection will timeout as shown in Figure 6.

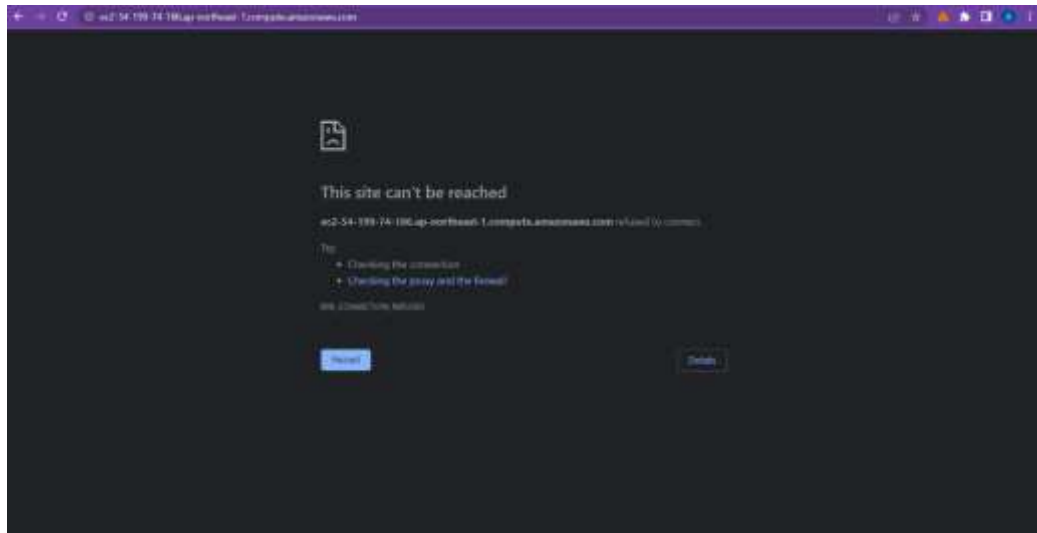


Figure 6: Browser connection timeout

This is where the Security Group comes in. In the AWS management console, go to **Security Groups**, select the one not named 'default' (launch-wizard-1 or your security group name) and edit the **Inbound** rules as shown in Figure 7.

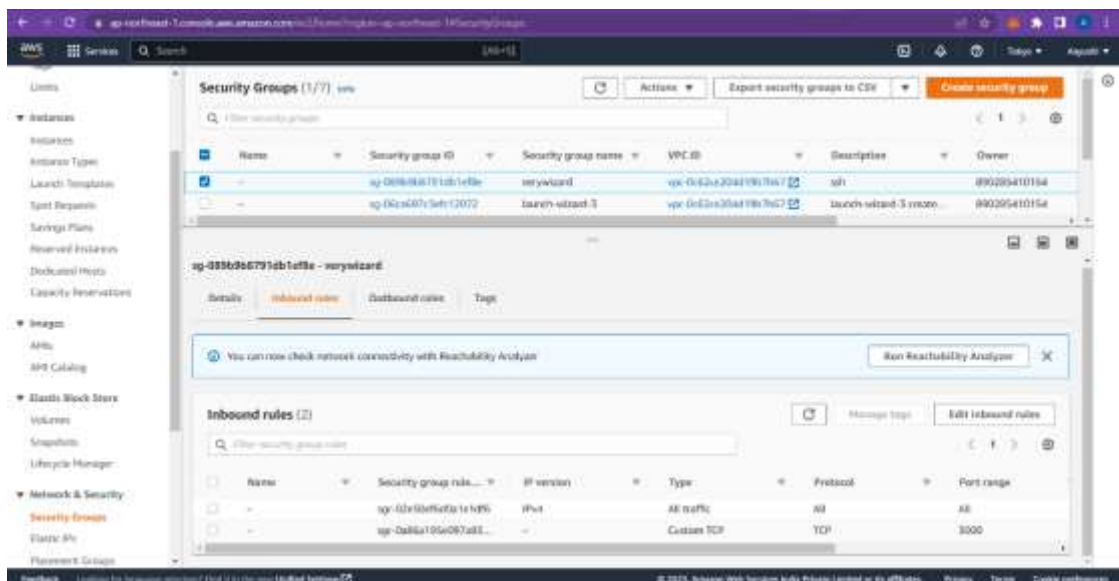


Figure 7: Security Groups

Access the URL again and you should be able to see the app content now.

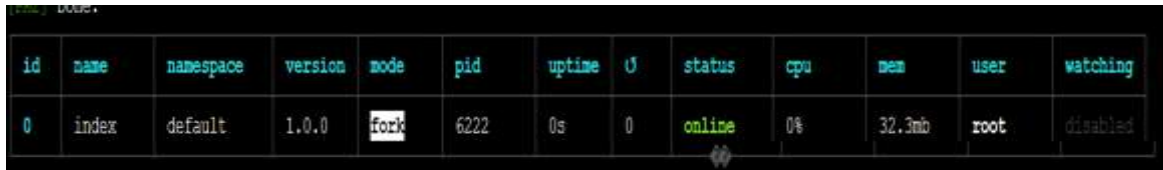
7. Keep App running using Pm2:

The app is running as soon as you open the terminal and it will terminate as you will close the terminal. We will install PM2 (Production manager 2) to keep live our app after closing our terminal or disconnect from remote server. Run the following command:

- `npm install pm2 -g`

It will install PM2 package globally on server. Switch to our app directory and run:

- `sudo pm2 start index.js`



id	name	namespace	version	mode	pid	uptime	U	status	cpu	mem	user	watching
0	index	default	1.0.0	fork	6222	0s	0	online	0%	32.3mb	root	disabled

Figure 8: List of running applications

Now even if you close the terminal and check the url in the browser, the app will be running. For automatically running PM2 when the server restarts, issue the following command:

- `sudo pm2 startup`

Now you can reboot the instance using `sudo reboot` and connect after 1 min. You can still see the app is running on port 3000 using:

- `pm2 list`
- `pm2 show app`

Source file code: index.js

```
const http = require('http');
const port = process.env.PORT || 3000;
const server = http.createServer((req, res) => {
  res.statusCode = 200;
  const msg = 'Hello Node!\n'
  res.end(msg);
});
server.listen(port, () => {
  console.log(`Server running on http://localhost:${port}^`);
});
```

Output: Hello Node!

Screenshot of output

Enter “<http://ec2-54-199-74-186.ap-northeast-1.compute.amazonaws.com:3000/>” in the web browser and the output is as shown in Figure – 9.

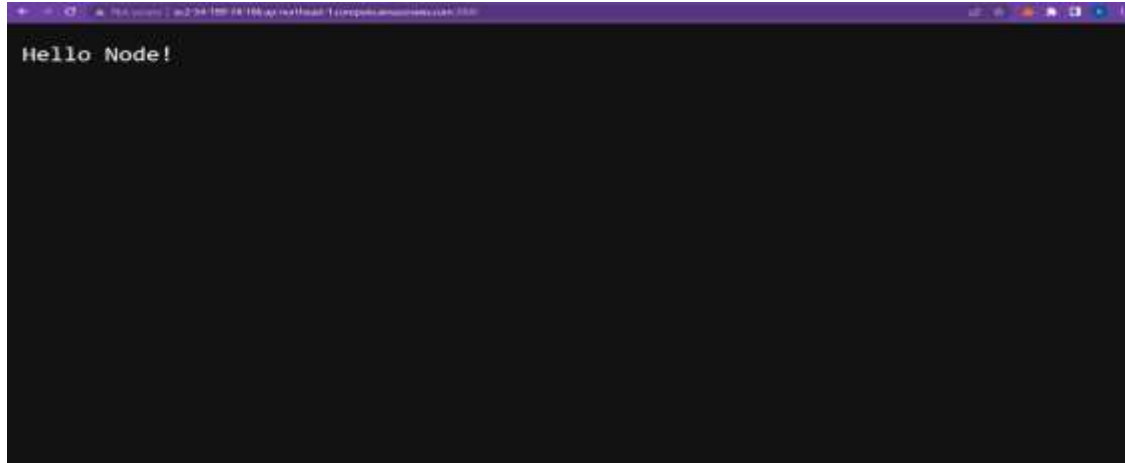


Figure 9: Output of deployed node.js web application