PROBLEM STATEMENT

OBJECTIVE:

Dockerize a simple Node.js app and deploy it on an AWS EC2 instance, this application will interact with a MySQL database hosted on AWS RDS, and set up a CI/CD pipeline with either AWS CodePipeline or Jenkins. Provision the infrastructure on AWS using terraform.

REQUIREMENTS:

- ➔ Dockerize a Simple Node.js App
- ➔ Set Up AWS Resources
- ➔ Deploy Node.js App on EC2
- ➔ CI/CD Pipeline
  - ❖ Option 1: AWS CodePipeline:
  - ❖ Option 2: Jenkins
- ➔ Monitoring
- ➔ Set Up above AWS Resources with Terraform

SUBMISSIONS DETAILS:

- ➔ Submit task in github repository link
- ➔ GitHub repo containing:
  - ❖ Application code.
  - ❖ Dockerfile for containerization.
  - ❖ Jenkinsfile for CI/CD pipeline.
  - ❖ Terraform template for infrastructure provisioning.
  - ❖ README file with setup and run instructions.
  - ❖ Brief overview of design decisions with workflow diagram.
  - ❖ Description of any challenges faced during implementation.

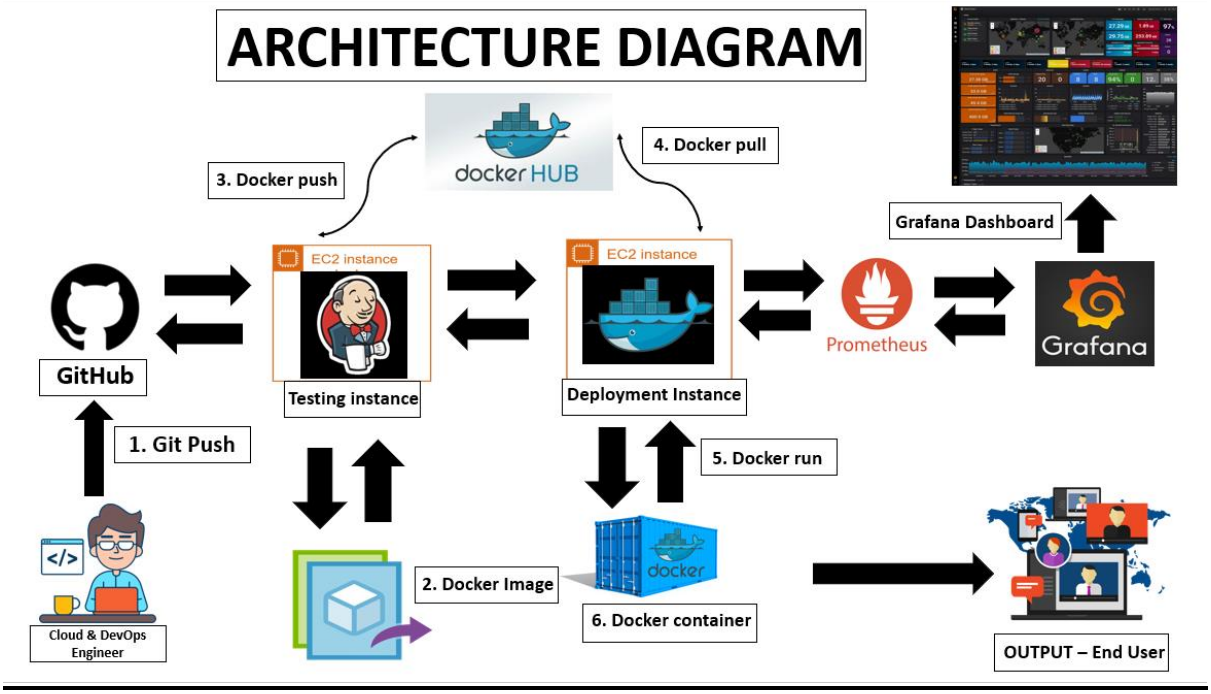
****

SOLUTION:

PRE-REQUIREMENTS:

Cloud	AWS
IAC	Terraform
Environment	Nodejs
Containerization	Docker
CI/CD	Jenkins
Monitoring	CloudWatch, Prometheus & Grafana
Version control	Git & GitHub

ARCHITECTURE DIAGRAM:

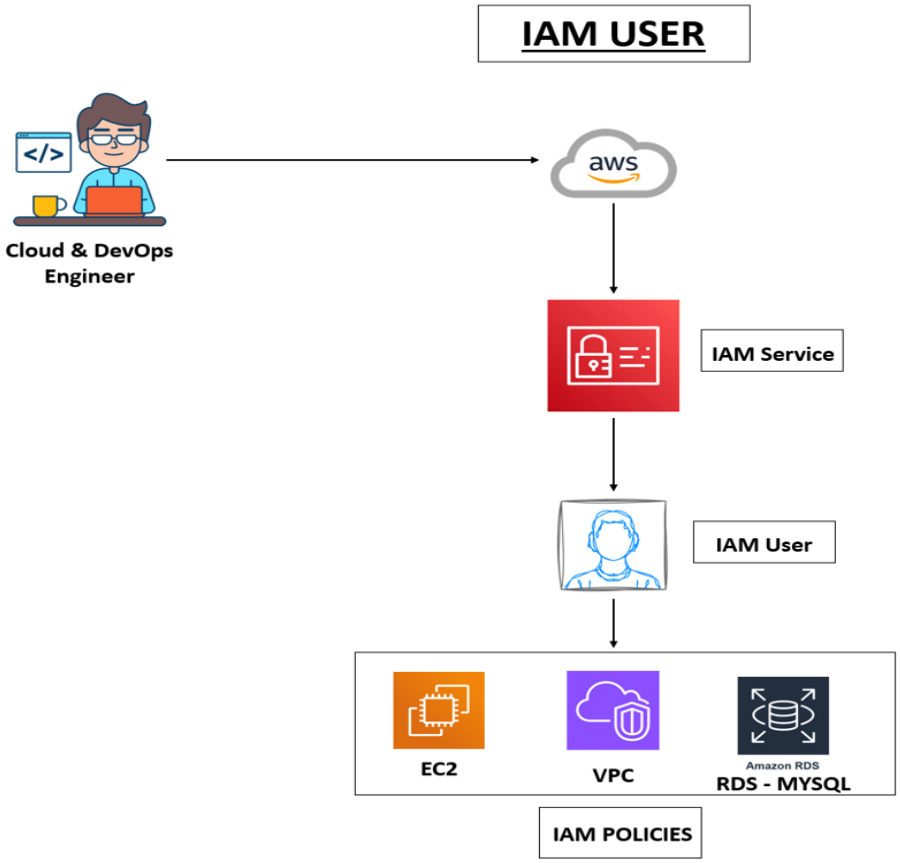


GitHub Repository: https://github.com/Ravivarman16/mysql-nodejs-application_deployment.git

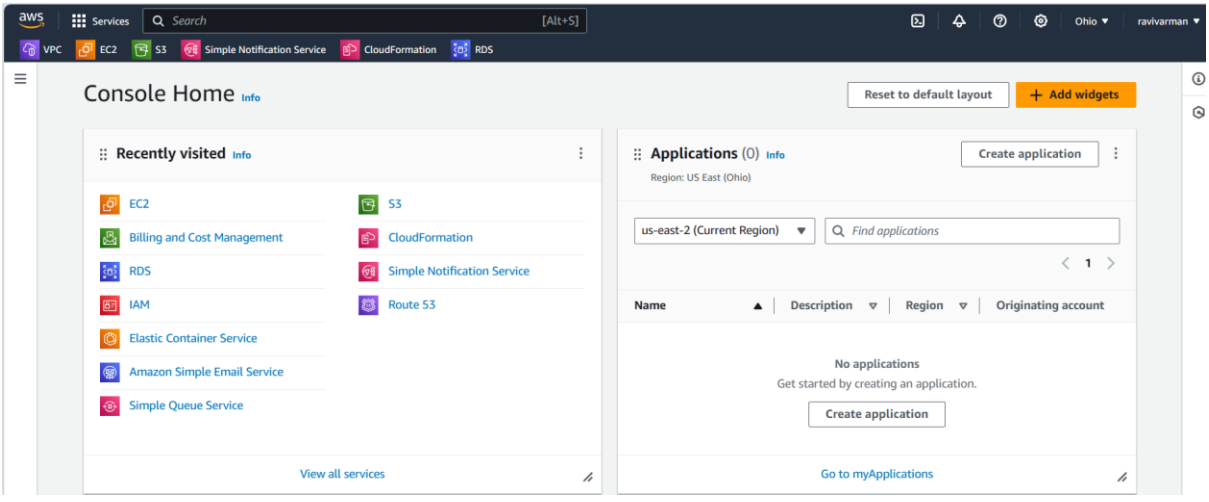
****

STEP:1 – CREATING AN IAM USER FOR TERRAFORM:

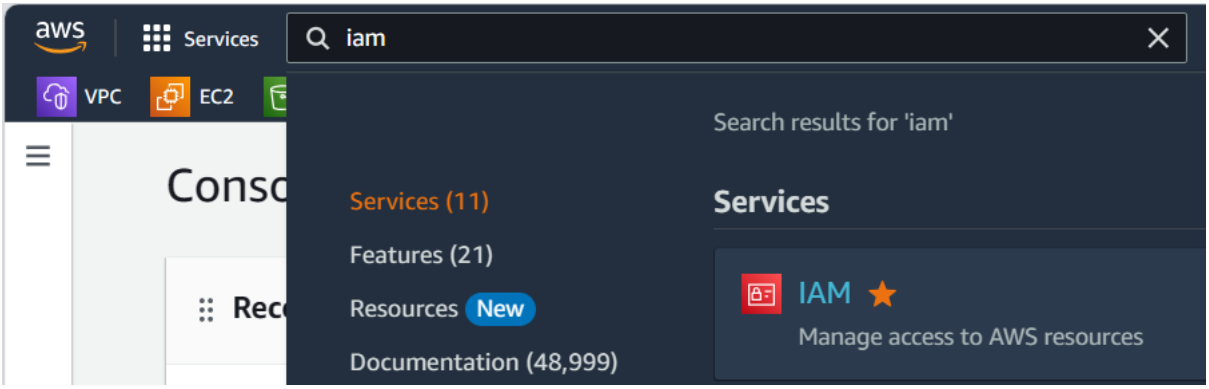
OVERVIEW DIAGRAM



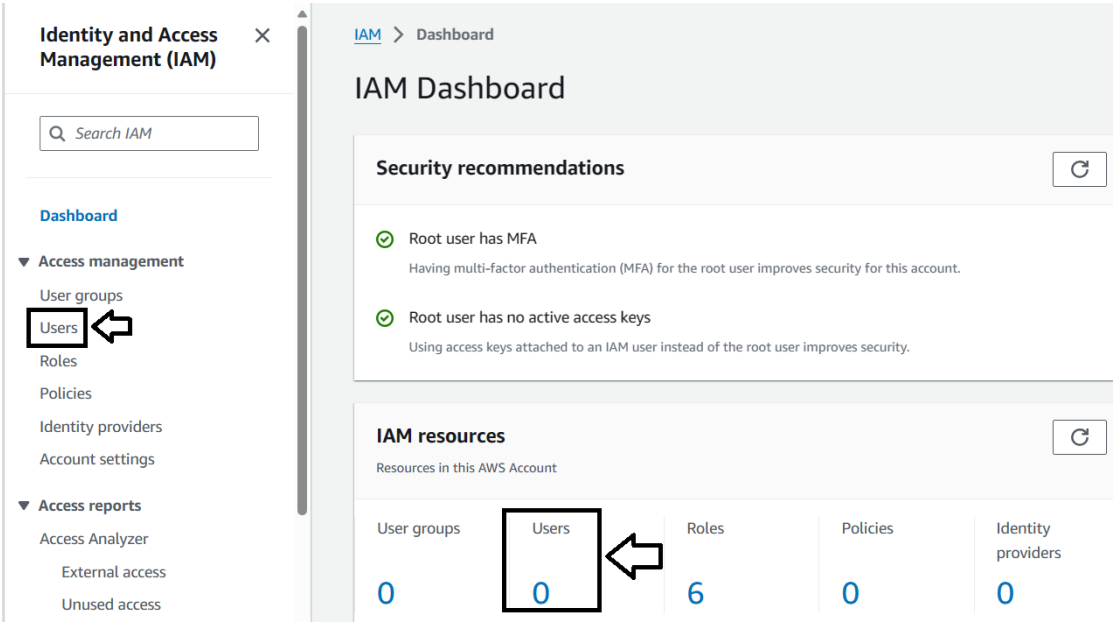
➔ First, we need to login into our **AWS Management Console**:



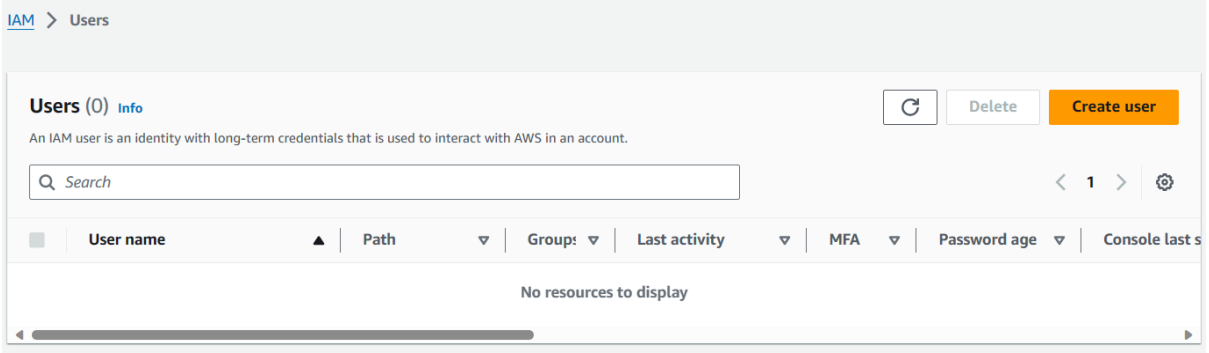
➔ Then search **IAM** on service panel, click that one:



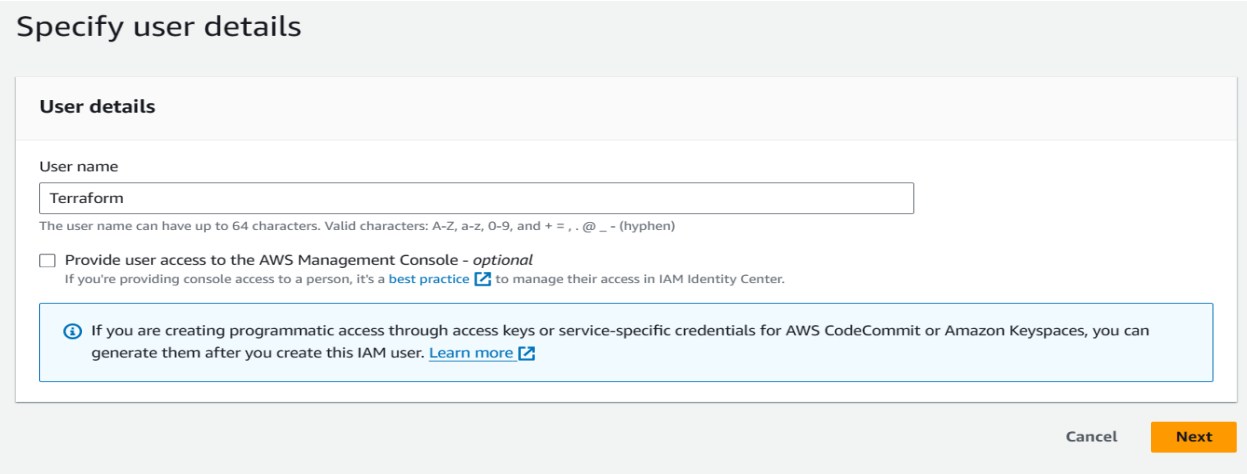
➔ Then on IAM Management console: we need to create **IAM user** for that, **click users on left side** of the console or **click users at centre** of the screen according to your wish:



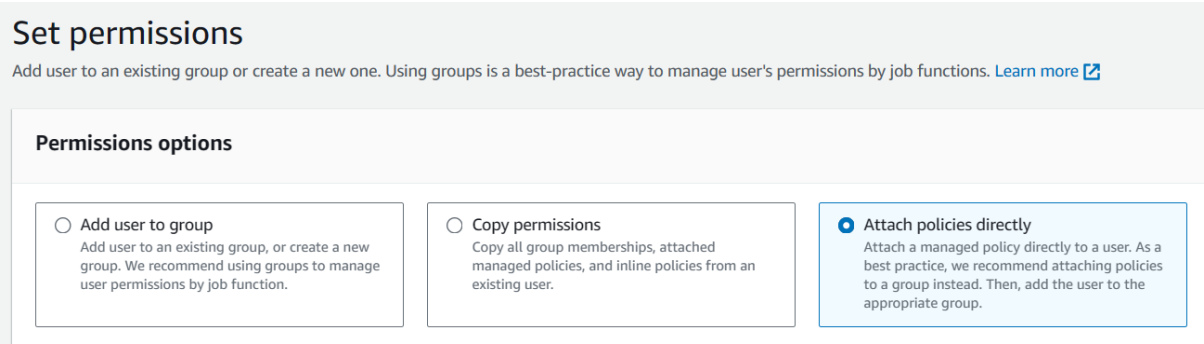
➔ Then click **create user** option:



➔ Then name IAM User as Terraform and click next:



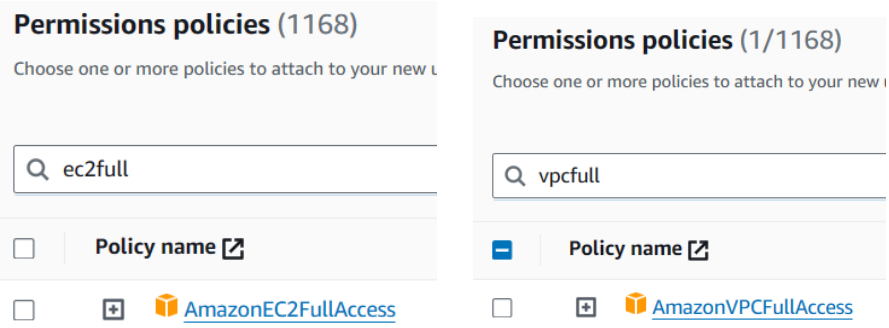
➔ Then select **attach policies directly** option to attach policies we required for this task:

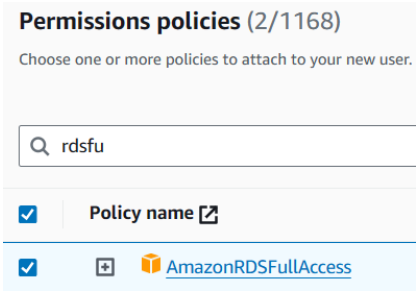


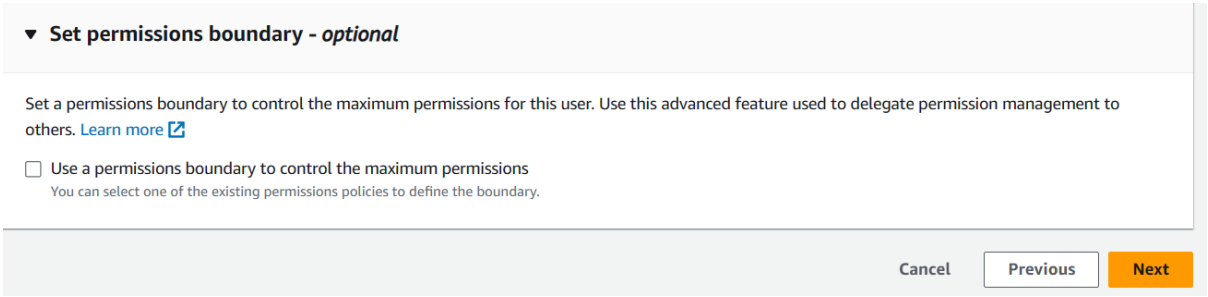
➔ Then we need to attach policies for this task, the required policies are

- ❖ **AmazonEC2FullAccess**
- ❖ **AmazonRDSFullAccess**
- ❖ **AmazonVPCFullAccess**

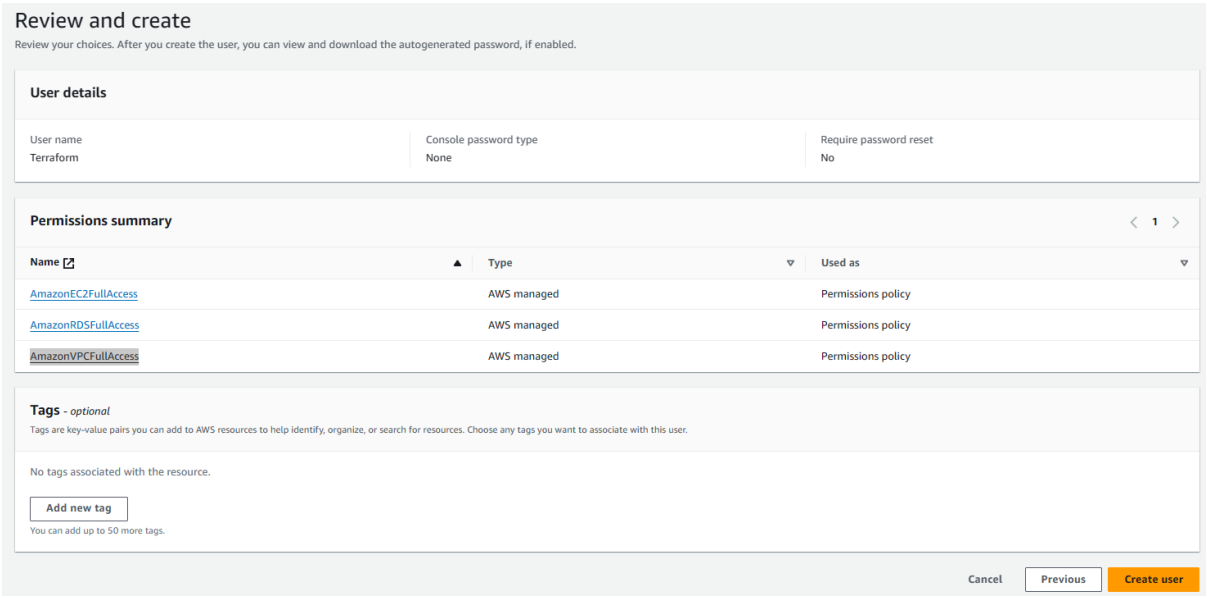
Attach these policies, click next:



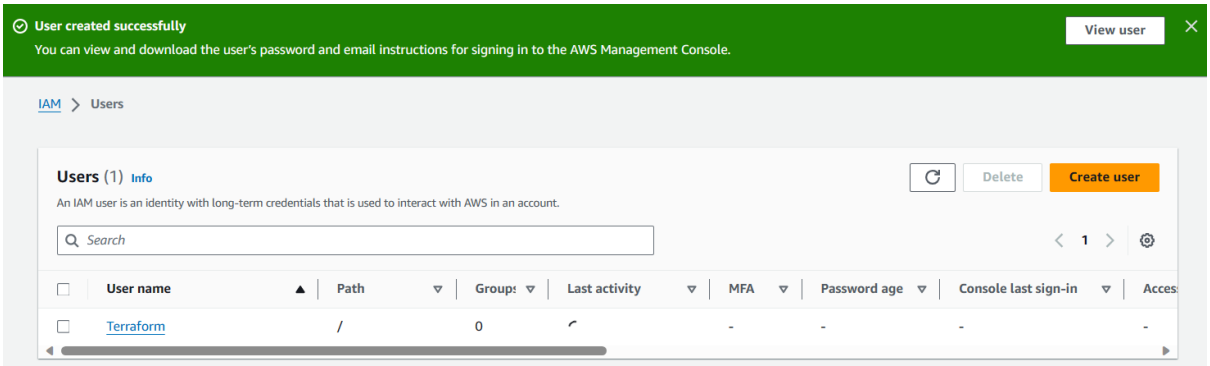




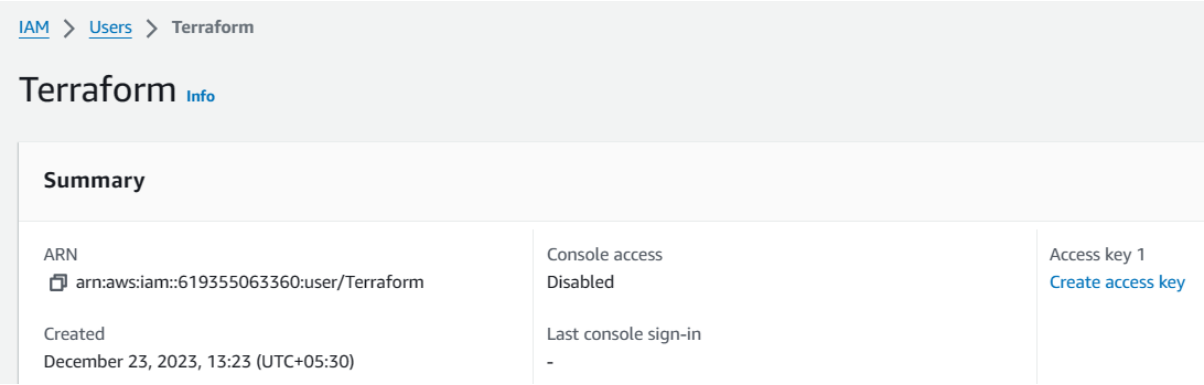
➔ After that just **review and create** page will appear, just review the configurations, click **create user:**



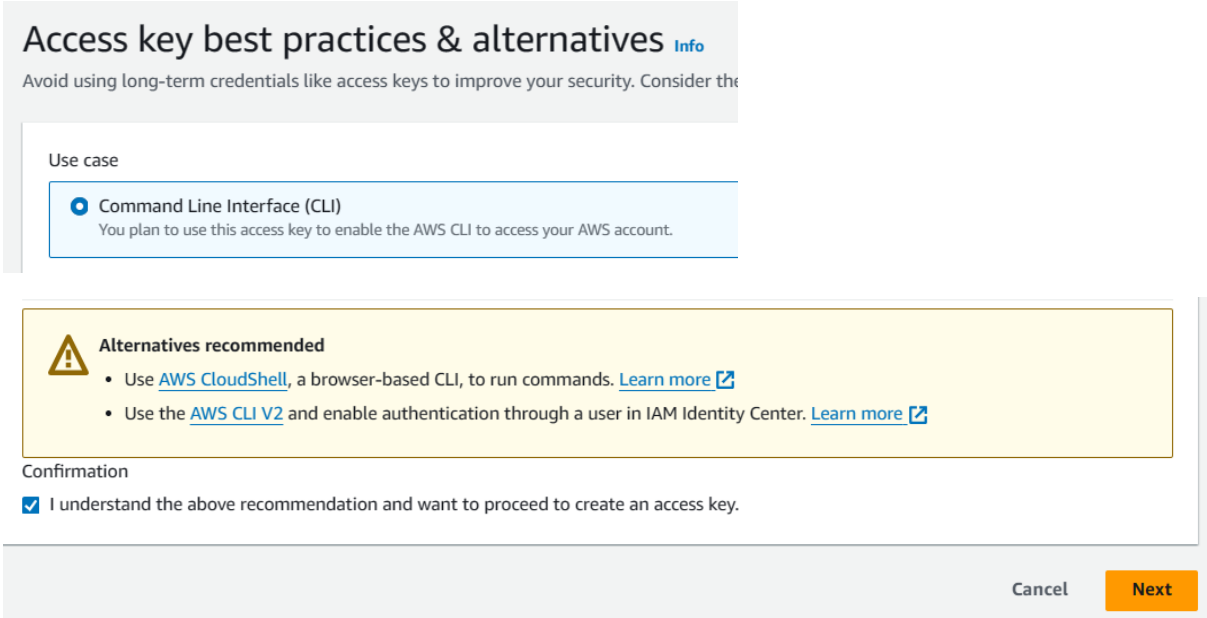
➔ The user has been created successfully:



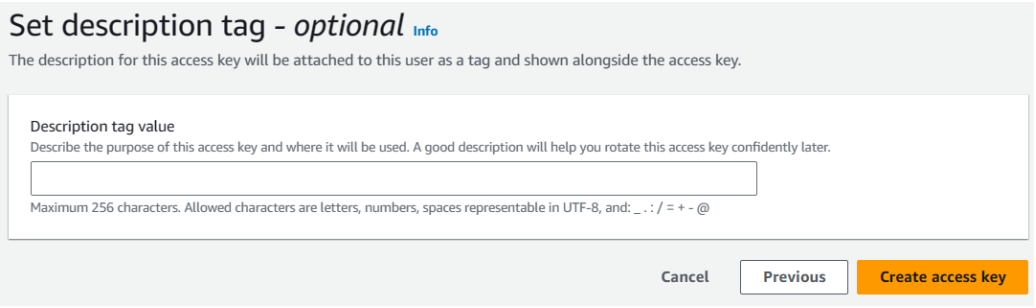
➔ Then click the **created IAM user**, click **Create access key** option to **create access and secret key** for above user:



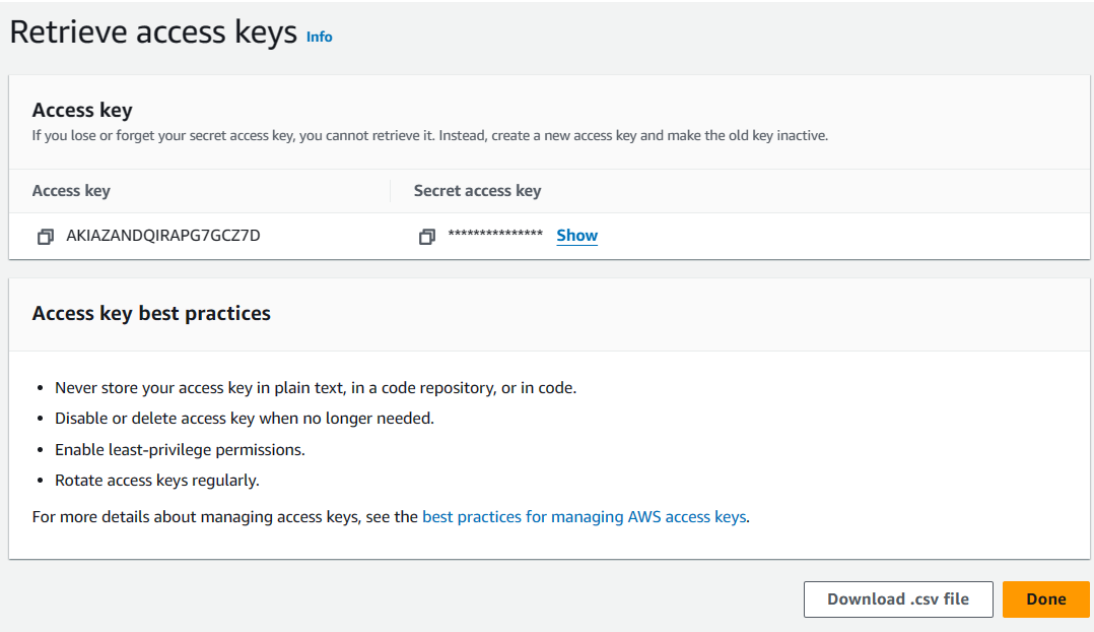
➔ Then click the use case **as Command line interface (CLI):** Just **acknowledge the confirmation, click next:**



➔ Then just click **create access key:**



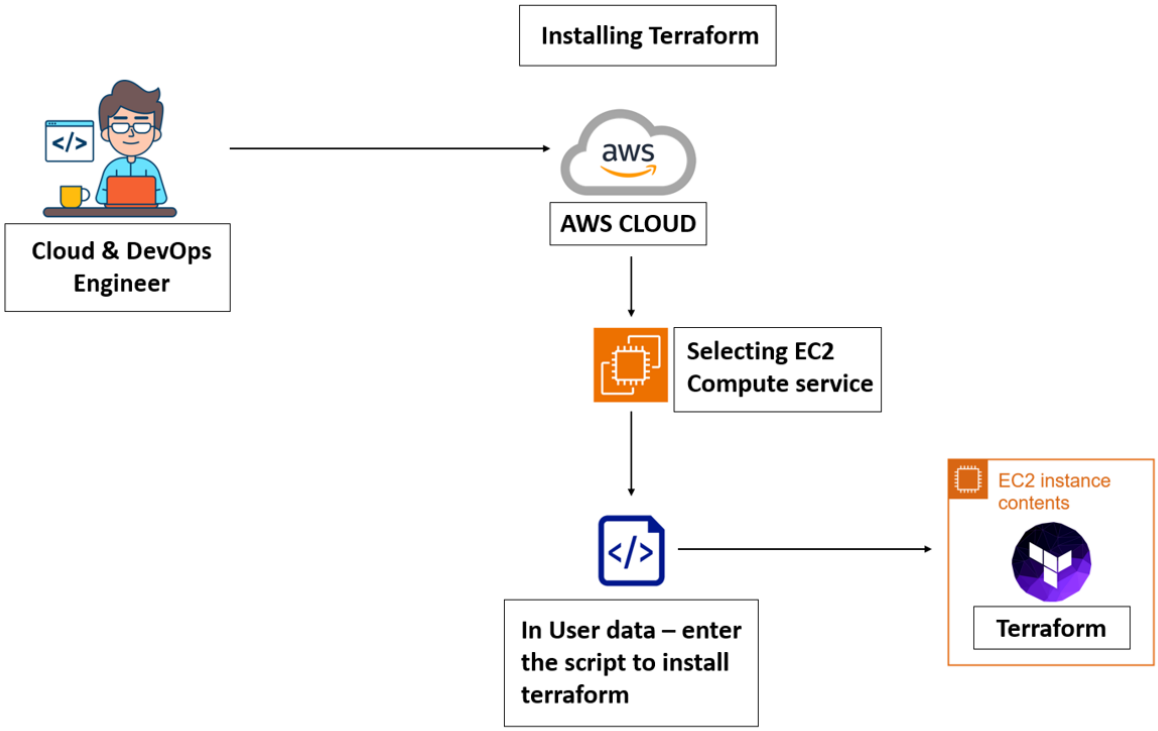
➔ The access and secret key have been created successfully:



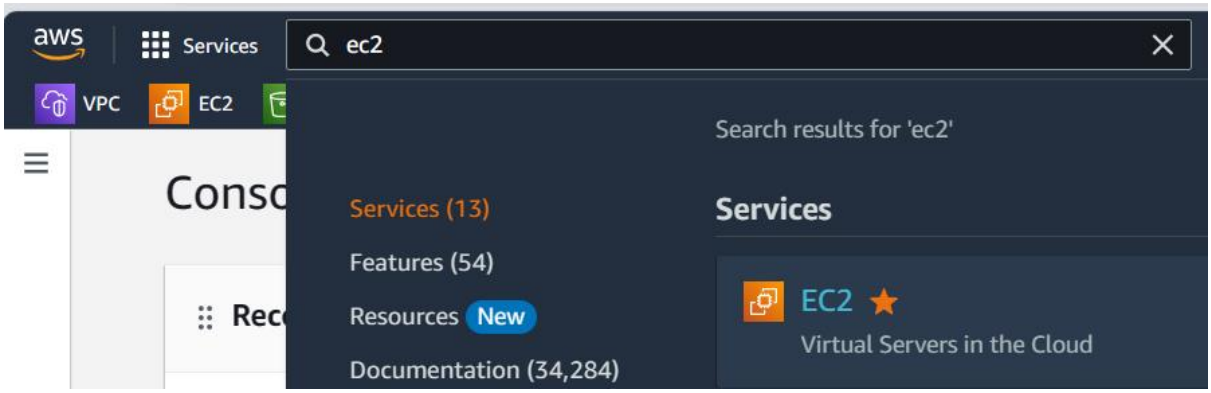
****

STEP:2 – CREATING AN INSTANCE FOR TERRAFORM:

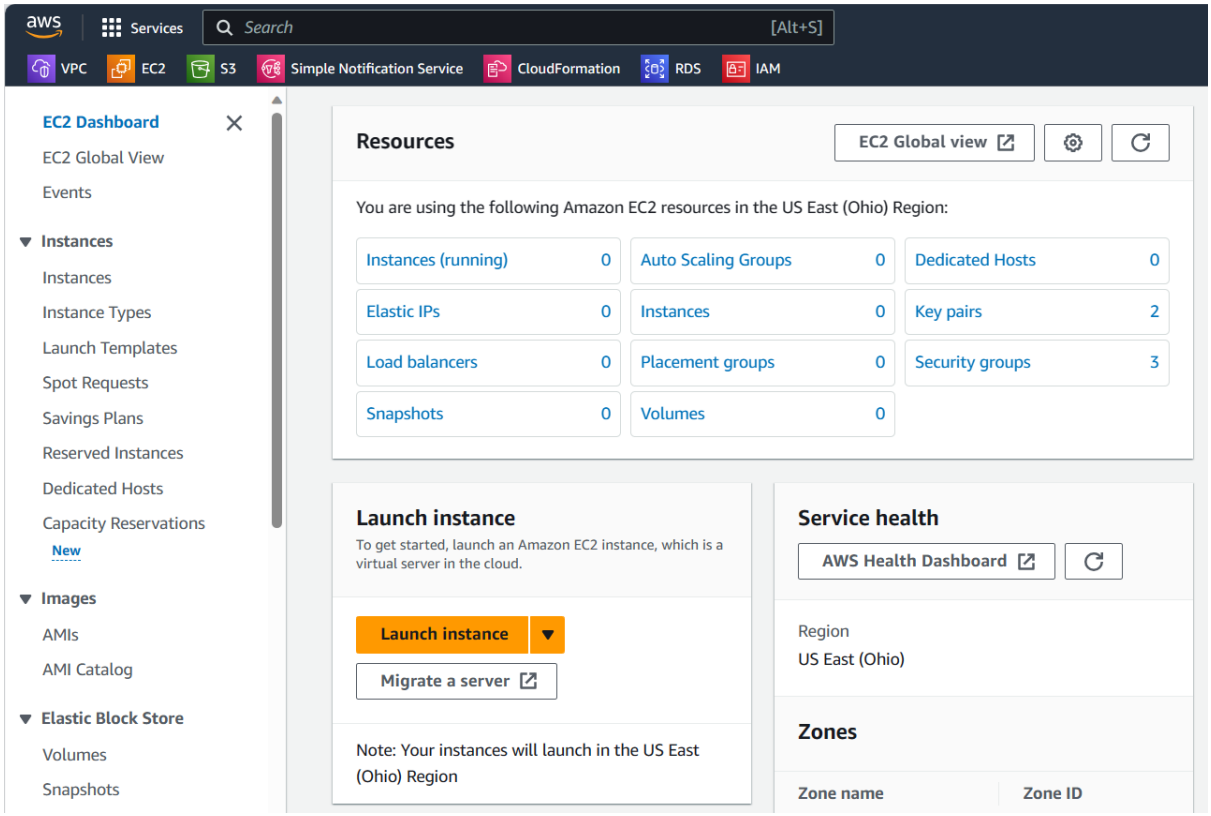
OVERVIEW – DIAGRAM



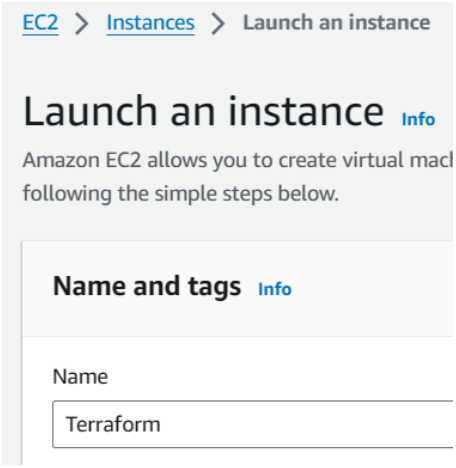
➔ Search **EC2** on service panel, click that one:



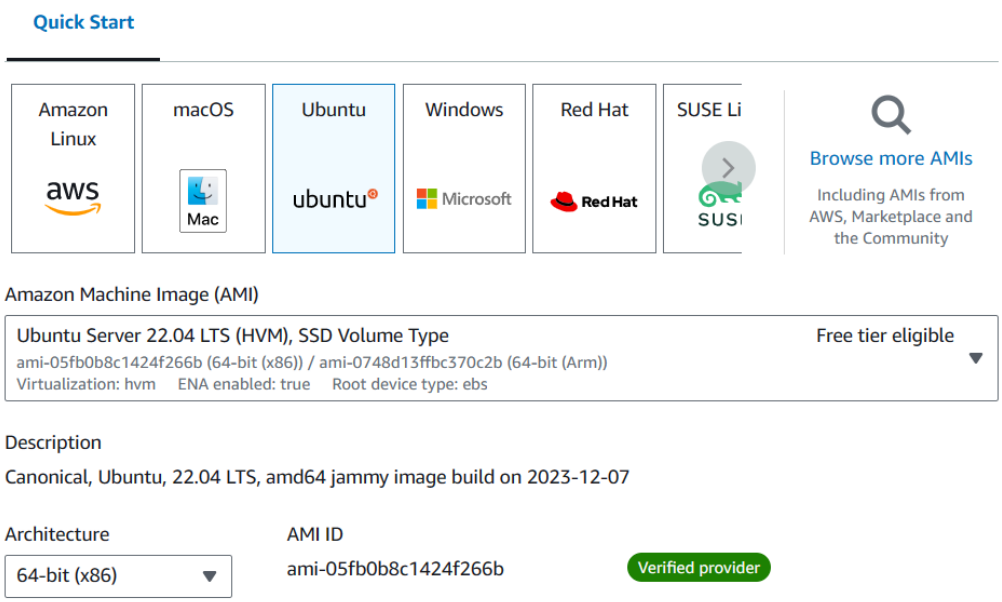
➔ Then on EC2 Management Console, just click **launch instance** option:



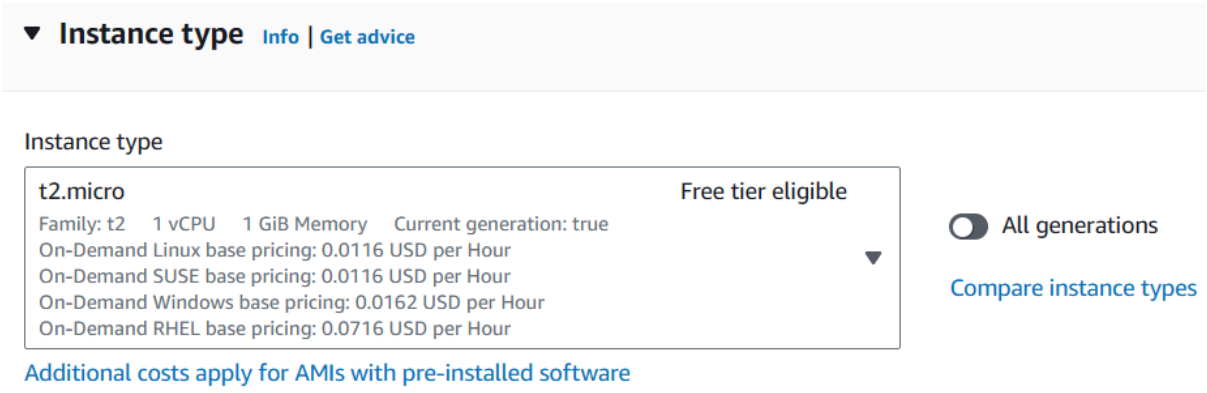
➔ Then name the instance according to your wish:



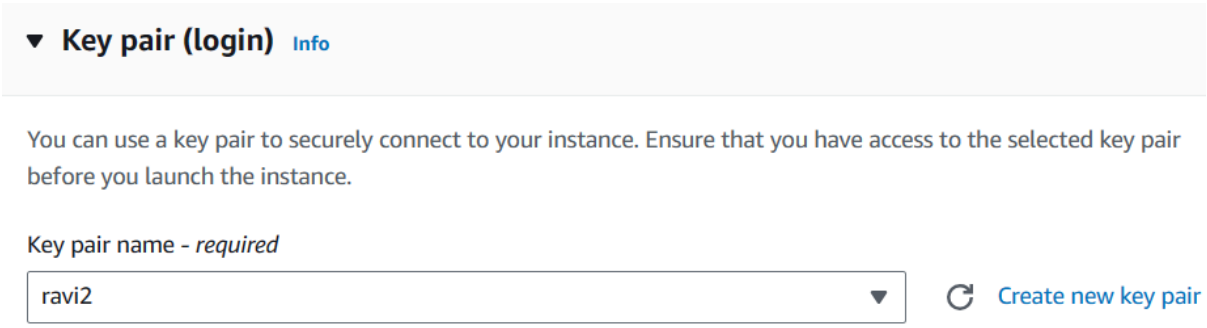
➔ Then select the **operating system** according to your wish, here I am selecting **ubuntu** as operating system:



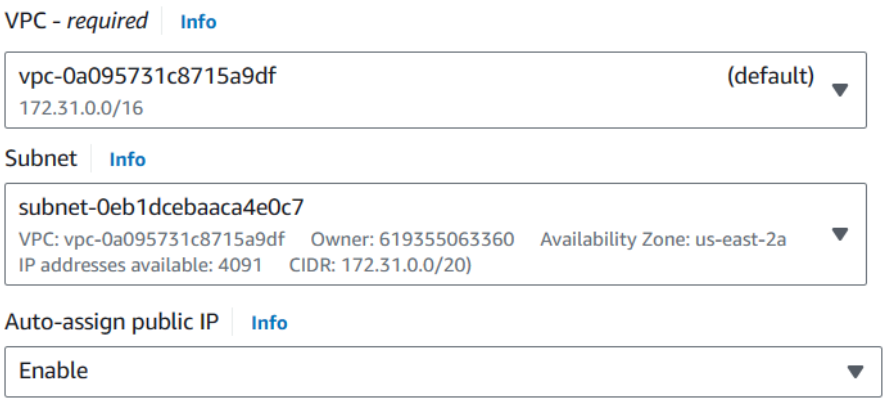
➔ Then select the **instance type** as **t2.micro**



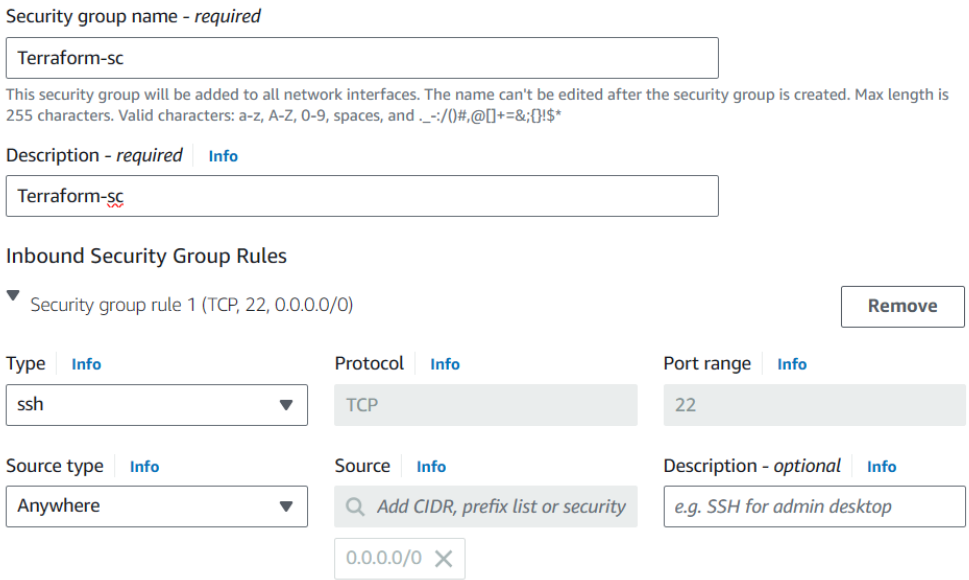
➔ Then select the **keypair**:



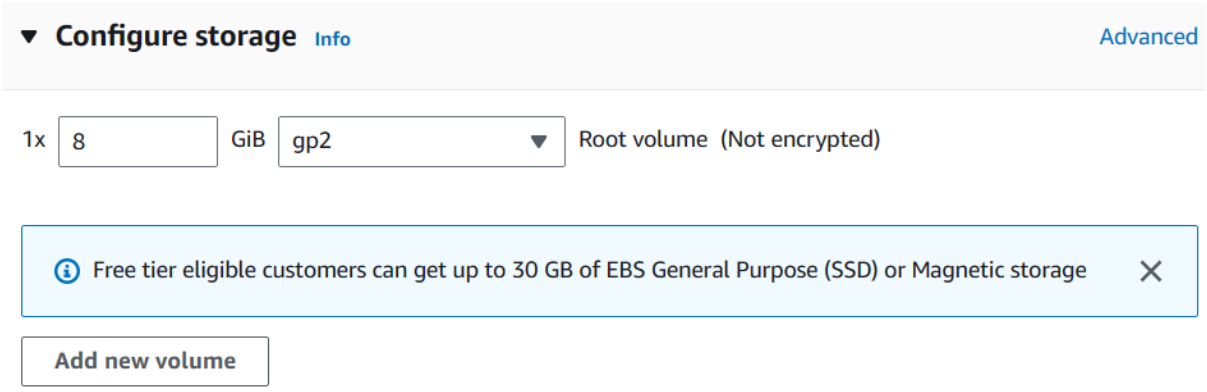
➔ Then under the **network settings**, select the **subnet** according to your wish:



➔ Then name the security group as **terraform-sc:**



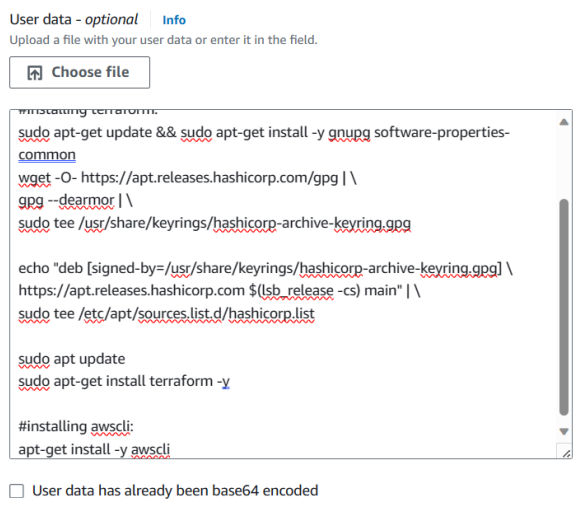
➔ Then keep the default storage as it is:

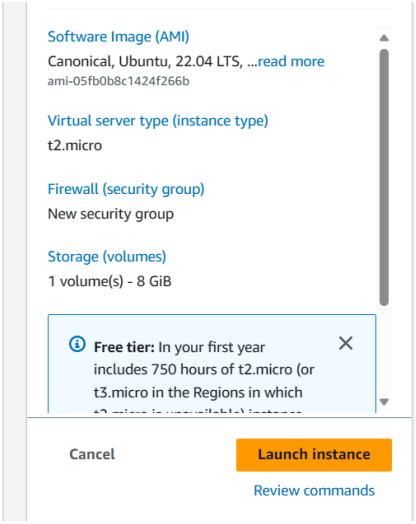


➔ Then under advanced settings, user data section, enter the terraform script to install **terraform & awscli. Click launch instance:**

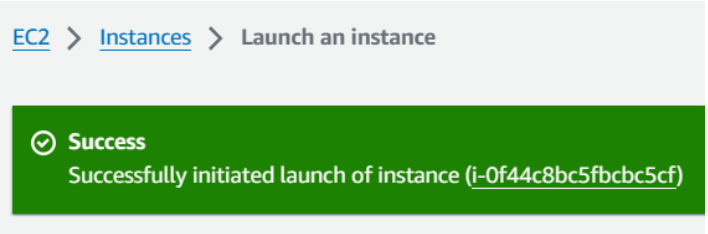
Script contains:

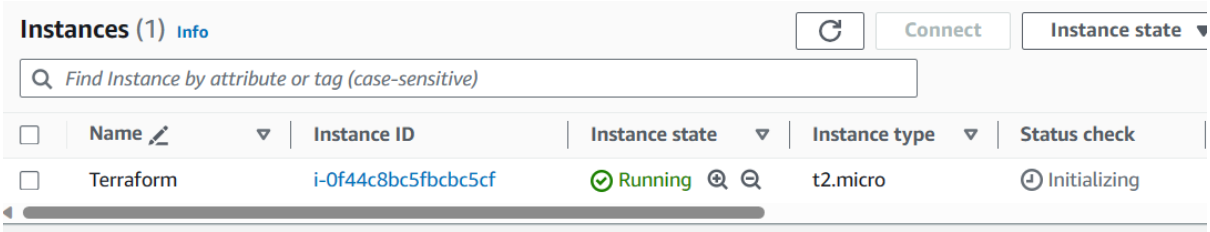
```
#!/bin/bash
#updating the os:
apt-get update

#installing terraform:
sudo apt-get update && sudo apt-get install -y gnupg
software-properties-common
wget -O- https://apt.releases.hashicorp.com/gpg | \
gpg --dearmor | \
sudo tee /usr/share/keyrings/hashicorp-archive-keyring.gpg

echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-
keyring.gpg] \
https://apt.releases.hashicorp.com $(lsb_release -cs) main"
| \
sudo tee /etc/apt/sources.list.d/hashicorp.list

sudo apt update
sudo apt-get install terraform -y

#installing awscli:
apt-get install -y awscli
```





➔ The instance has launched successfully:





➔ Then connect the instance and verify whether **terraform & awscli** is installed or not by using command:

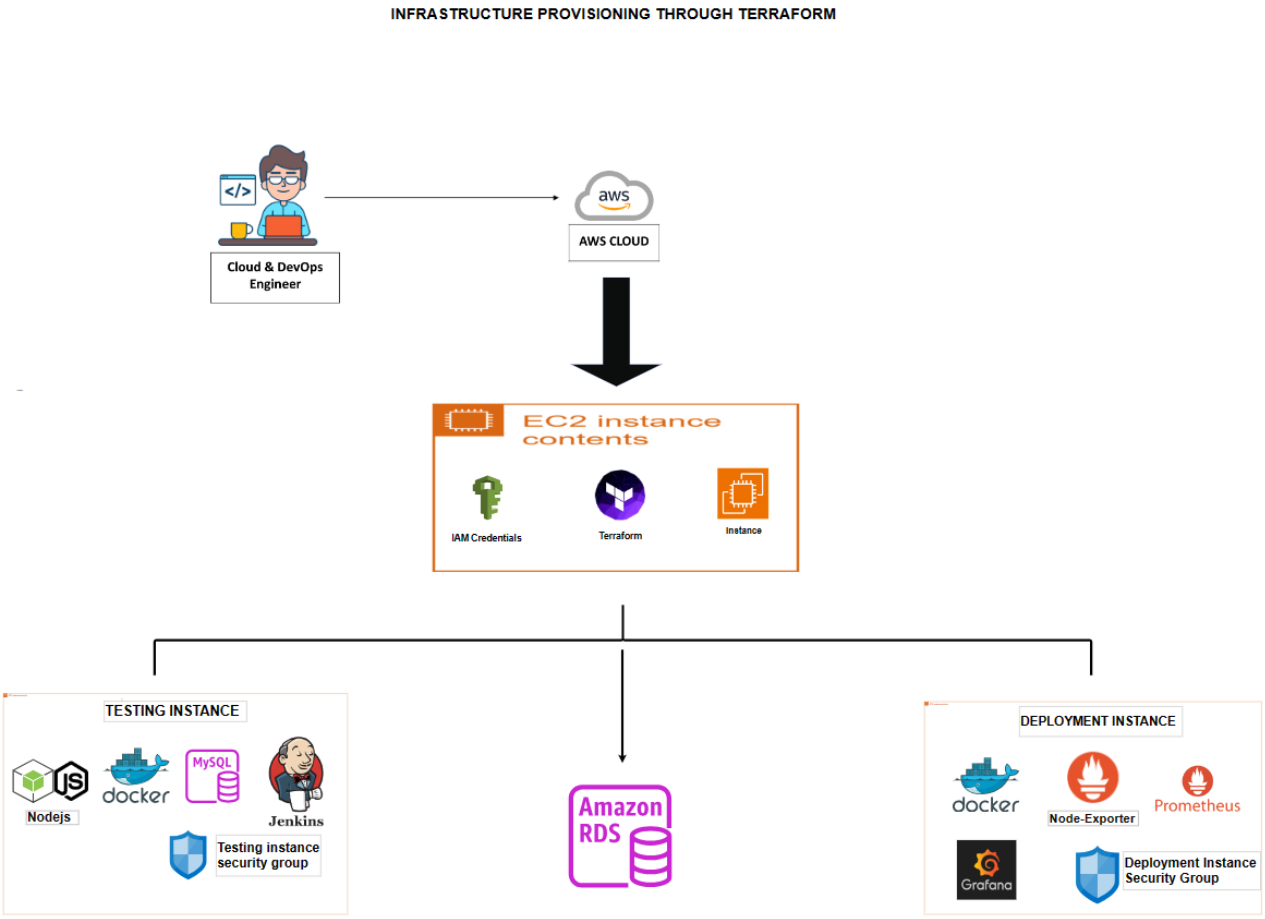
```
terraform --version
aws --version
```

```
root@ip-172-31-10-219:/home/ubuntu# terraform --version
Terraform v1.6.6
on linux_amd64
root@ip-172-31-10-219:/home/ubuntu# aws --version
aws-cli/1.22.34 Python/3.10.12 Linux/6.2.0-1017-aws botocore/1.23.34
root@ip-172-31-10-219:/home/ubuntu#
```

****

STEP:3 – PROVISIONING THE INFRASTRUCTURE THROUGH TERRAFORM

OVERVIEW – DIAGRAM



➔ First, we need to **configure the aws credentials of IAM User** for terraform to use it for creating the required resources for this task:

```
root@ip-172-31-10-219:/home/ubuntu# aws configure
AWS Access Key ID [None]: AKIAZANDQIRAPG7GCZ7D
AWS Secret Access Key [None]: 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
Default region name [None]: us-east-2
Default output format [None]: json
root@ip-172-31-10-219:/home/ubuntu#
```

➔ Creating a terraform script for provisioning the infrastructure required for this task:

Terraform script contains:

```
# 1.Providing aws details:
provider "aws" {
  profile = "default"
  region  = "us-east-2"
}

# 2.Creating security group for testing instance:
resource "aws_security_group" "testing-sc" {
  name        = "testing-instance"
  description = "security group for AWS EC2 instances"

 # Ingress rules (inbound traffic)
# Allow SSH (port 22) from anywhere
  ingress {
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

# Allow application & jenkins (port 8080) from anywhere
  ingress {
    from_port   = 8080
    to_port     = 8080
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

# Allow mysql database (port 3306) from anywhere
  ingress {
    from_port   = 3306
    to_port     = 3306
    protocol    = "tcp"
    cidr_blocks = ["172.31.0.0/16"]
  }

  # Allow HTTP (port 80) from anywhere
  ingress {
```

```
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  # Egress rules (outbound traffic)
  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }

  tags = {
    Name = "testing-instance"
  }
}

# 3.Creating security group for deployment instance:
resource "aws_security_group" "deployment-sc" {
  name        = "deployment-instance"
  description = "security group for AWS EC2 instances"

# Ingress rules (inbound traffic)
# Allow SSH (port 22) from anywhere
  ingress {
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

# Allow HTTP (port 80) from anywhere
  ingress {
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
```

```
    cidr_blocks = ["0.0.0.0/0"]
  }

# Allow node-exporter (port 9100) from anywhere
  ingress {
    from_port   = 9100
    to_port     = 9100
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

# Allow prometheus (port 9090) from anywhere
  ingress {
    from_port   = 9090
    to_port     = 9090
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

# Allow grafana (port 3000) from anywhere
  ingress {
    from_port   = 3000
    to_port     = 3000
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

# Allow docker-daemon (port 9323) from anywhere
  ingress {
    from_port   = 9323
    to_port     = 9323
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  # Egress rules (outbound traffic)
  egress {
    from_port   = 0
```

```
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }

  tags = {
    Name = "deployment-instance"
  }
}

#4.Creating testing instance:
resource "aws_instance" "jenkins-instance" {
  ami                  = "ami-07b36ea9852e986ad"
  instance_type        = "t2.small"
  availability_zone = "us-east-2a"
  key_name             = "ravi2"
  vpc_security_group_ids = ["${aws_security_group.testing-
sc.id}"]
  user_data            = "${file("testing.sh")}"

  tags = {
    Name = "testing-instance"
  }
}

#5.Creating deployment instance:
resource "aws_instance" "deployment-instance" {
  ami                  = "ami-07b36ea9852e986ad"
  instance_type        = "t2.micro"
  availability_zone = "us-east-2b"
  key_name             = "ravi2"
  vpc_security_group_ids =
["${aws_security_group.deployment-sc.id}"]
  user_data            = "${file("deployment.sh")}"

  tags = {
    Name = "deployment-instance"
  }
```

```
}

#6.Creating mysql database:
resource "aws_db_instance" "mydb" {
  identifier             = "nodejs-db"
  allocated_storage      = 20
  storage_type           = "gp2"
  db_name                = "nodejs"
  engine                 = "mysql"
  engine_version         = "5.7"
  instance_class         = "db.t2.micro"
  username               = "admin"
  password               = "nodejs123"
  parameter_group_name   = "default.mysql5.7"
  availability_zone      = "us-east-2c"
  skip_final_snapshot    = true
  vpc_security_group_ids = ["${aws_security_group.testing-
sc.id}"]
  tags = {
    Name = "nodejs-db"
  }
}

#7. Getting the output of mysql database:
output "rds_endpoint" {
  value = aws_db_instance.mydb.endpoint
}
```

testing.sh script contains:

```
#!/bin/bash
set -e  # Exit script if any command returns a non-zero
status

#1. Updating the operating system:
apt-get update

#2. Installing java:
```

```
apt-get install -y openjdk-17-jre

#3. Installing jenkins:
sudo wget -O /usr/share/keyrings/jenkins-keyring.asc \
  https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key
echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc]
\
  https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
  /etc/apt/sources.list.d/jenkins.list > /dev/null
apt-get update
apt-get install jenkins -y

#4. Installing docker:
apt-get install -y docker.io
chmod 777 /var/run/docker.sock

#5. Installing nodejs:
cd ~
curl -sL https://deb.nodesource.com/setup_16.x -o
/tmp/nodesource_setup.sh
sudo bash /tmp/nodesource_setup.sh
sudo apt-get install -y nodejs

#6. installing mysql:
apt install mysql-client-core-8.0 -y
```

deployment.sh script contains:

```
#!/bin/bash

#1. Updating the OS:
apt-get update

#2. Installing docker:
apt-get install -y docker.io
chmod 777 /var/run/docker.sock
```

```
#3. Installing node-exporter:
#!/bin/bash
NODE_EXPORTER_VERSION="1.7.0"
wget
https://github.com/prometheus/node_exporter/releases/downloa
d/v${NODE_EXPORTER_VERSION}/node_exporter-
${NODE_EXPORTER_VERSION}.linux-amd64.tar.gz
tar -xzvf node_exporter-${NODE_EXPORTER_VERSION}.linux-
amd64.tar.gz
cd node_exporter-${NODE_EXPORTER_VERSION}.linux-amd64
cp node_exporter /usr/local/bin

# create user
useradd --no-create-home --shell /bin/false node_exporter

chown node_exporter:node_exporter
/usr/local/bin/node_exporter

echo '[Unit]
Description=Node Exporter
Wants=network-online.target
After=network-online.target

[Service]
User=node_exporter
Group=node_exporter
Type=simple
ExecStart=/usr/local/bin/node_exporter

[Install]
WantedBy=multi-user.target' >
/etc/systemd/system/node_exporter.service

# enable node_exporter in systemctl
systemctl daemon-reload
systemctl start node_exporter
systemctl enable node_exporter
```

```
#4. Installing prometheus:
#!/bin/bash
PROMETHEUS_VERSION="2.48.0"
wget
https://github.com/prometheus/prometheus/releases/download/v
${PROMETHEUS_VERSION}/prometheus-
${PROMETHEUS_VERSION}.linux-amd64.tar.gz
tar -xzvf prometheus-${PROMETHEUS_VERSION}.linux-
amd64.tar.gz
cd prometheus-${PROMETHEUS_VERSION}.linux-amd64/
# if you just want to start prometheus as root
#./prometheus --config.file=prometheus.yml

# create user
useradd --no-create-home --shell /bin/false prometheus

# create directories
mkdir -p /etc/prometheus
mkdir -p /var/lib/prometheus

# set ownership
chown prometheus:prometheus /etc/prometheus
chown prometheus:prometheus /var/lib/prometheus

# copy binaries
cp prometheus /usr/local/bin/
cp promtool /usr/local/bin/

chown prometheus:prometheus /usr/local/bin/prometheus
chown prometheus:prometheus /usr/local/bin/promtool

# copy config
cp -r consoles /etc/prometheus
cp -r console_libraries /etc/prometheus
cp prometheus.yml /etc/prometheus/prometheus.yml

chown -R prometheus:prometheus /etc/prometheus/consoles
```

```
chown -R prometheus:prometheus
/etc/prometheus/console_libraries

# setup systemd
echo '[Unit]
Description=Prometheus
Wants=network-online.target
After=network-online.target

[Service]
User=prometheus
Group=prometheus
Type=simple
ExecStart=/usr/local/bin/prometheus \
    --config.file /etc/prometheus/prometheus.yml \
    --storage.tsdb.path /var/lib/prometheus/ \
    --web.console.templates=/etc/prometheus/consoles \
    --
web.console.libraries=/etc/prometheus/console_libraries

[Install]
WantedBy=multi-user.target' >
/etc/systemd/system/prometheus.service

systemctl daemon-reload
systemctl enable prometheus
systemctl start prometheus

#5. Installing grafana:
#!/bin/bash
echo 'deb https://packages.grafana.com/oss/deb stable main'
>> /etc/apt/sources.list
curl https://packages.grafana.com/gpg.key | sudo apt-key add
-
sudo apt-get update
sudo apt-get -y install grafana

systemctl daemon-reload
```

```
systemctl start grafana-server
systemctl enable grafana-server.service
```

➔ The above resources have been created:

```
root@ip-172-31-10-219:/home/ubuntu# vi instance.tf
root@ip-172-31-10-219:/home/ubuntu# vi testing.sh
root@ip-172-31-10-219:/home/ubuntu# vi deployment.sh
```

➔ First, we need to perform **terraform init** command: to initialize the plugins for terraform to control our aws cloud:

```
root@ip-172-31-10-219:/home/ubuntu# terraform init

Initializing the backend...

Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v5.31.0...
- Installed hashicorp/aws v5.31.0 (signed by HashiCorp)

Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
root@ip-172-31-10-219:/home/ubuntu#
```

➔ Then **terraform fmt** to format the syntax of terraform script:

```
root@ip-172-31-10-219:/home/ubuntu# terraform fmt
instance.tf
root@ip-172-31-10-219:/home/ubuntu#
```

➔ Then **terraform validate** to validate the keywords of terraform script:

```
root@ip-172-31-10-219:/home/ubuntu# terraform validate
Success! The configuration is valid.

root@ip-172-31-10-219:/home/ubuntu#
```

➔ Then **terraform plan,** for the blueprint of terraform execution:

```
root@ip-172-31-10-219:/home/ubuntu# terraform plan

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following
symbols:
  + create

Terraform will perform the following actions:

  # aws_db_instance.mydb will be created
  + resource "aws_db_instance" "mydb" {
      + address                               = (known after apply)
      + allocated_storage                     = 20
      + apply_immediately                     = false
      + arn                                   = (known after apply)
      + auto_minor_version_upgrade            = true
```

➔ Then terraform apply -auto-approve command, to initiate terraform to create above resources:

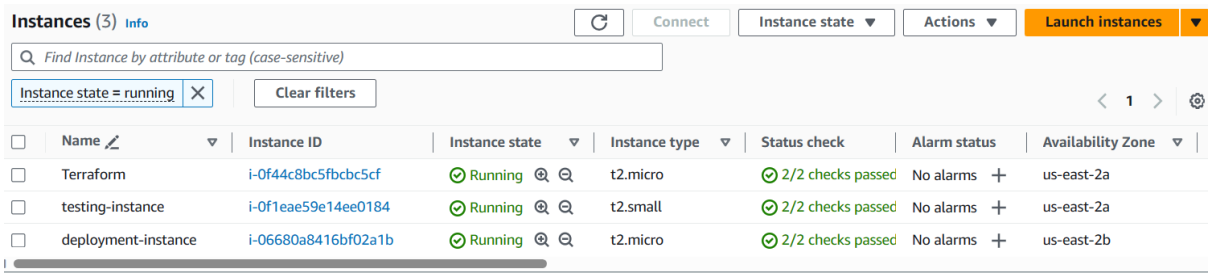
```
root@ip-172-31-10-219:/home/ubuntu# terraform apply -auto-approve
```

```
Plan: 5 to add, 0 to change, 0 to destroy.

Changes to Outputs:
  + rds_endpoint = (known after apply)
aws_security_group.testing-sc: Creating...
aws_security_group.deployment-sc: Creating...
aws_security_group.testing-sc: Creation complete after 2s [id=sg-069c69df61373e3c4]
aws_security_group.deployment-sc: Creation complete after 2s [id=sg-030220b5aa4ffba5f]
aws_db_instance.mydb: Creating...
aws_instance.jenkins-instance: Creating...
aws_instance.deployment-instance: Creating...
aws_db_instance.mydb: Still creating... [10s elapsed]
aws_instance.jenkins-instance: Still creating... [10s elapsed]
aws_instance.deployment-instance: Still creating... [10s elapsed]
aws_db_instance.mydb: Still creating... [20s elapsed]
aws_instance.jenkins-instance: Still creating... [20s elapsed]
aws_instance.deployment-instance: Still creating... [20s elapsed]
```

➔ Terraform has done its job, now we need to check the console:

```
Apply complete! Resources: 5 added, 0 changed, 0 destroyed.

Outputs:

rds_endpoint = "nodejs-db.chttjdyzo3c7.us-east-2.rds.amazonaws.com:3306"
root@ip-172-31-10-219:/home/ubuntu#
```

➔ **EC2 instance page:**



➔ Testing instance details;

Instance: i-0f1eae59e14ee0184 (testing-instance)

Security groups

sg-069c69df61373e3c4 (testing-instance)

➔ Deployment instance details:

Instance: i-06680a8416bf02a1b (deployment-instance)

Security groups

sg-030220b5aa4ffba5f (deployment-instance)

➔ Testing security group:

Security group rule ID	Type Info	Protocol Info	Port range Info	Source Info		Description - optional Info	
sgr-0188a65a152cba47d	HTTP	TCP	80	Custom	0.0.0.0/0		Delete
sgr-02012600e682ffd89	SSH	TCP	22	Custom	0.0.0.0/0		Delete
sgr-02b61a3b71b64c892	Custom TCP	TCP	8080	Custom	0.0.0.0/0		Delete
sgr-04b68d257eebfc847	MYSQL/Aurora	TCP	3306	Custom	172.31.0.0/16		Delete

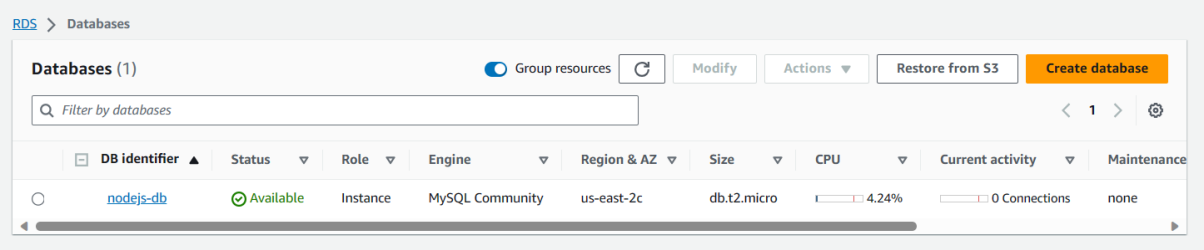
Add rule

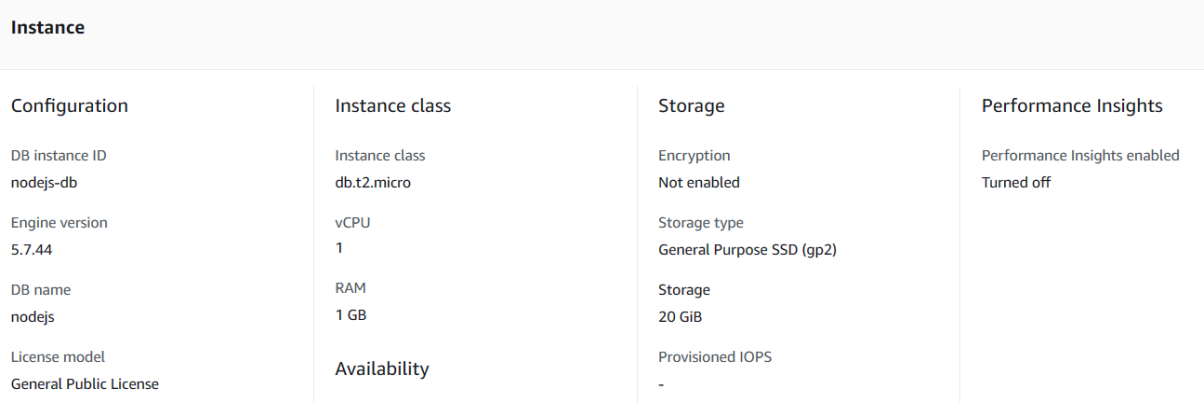
➔ Deployment security group:

Security group rule ID	Type Info	Protocol Info	Port range Info	Source Info		Description - optional Info	
sgr-09ee6a74d739e76db	Custom TCP	TCP	9323	Custom	0.0.0.0/0		Delete
sgr-0f35a9dcbfc870e5f	Custom TCP	TCP	9090	Custom	0.0.0.0/0		Delete
sgr-018aa93d80e23a027	Custom TCP	TCP	9100	Custom	0.0.0.0/0		Delete
sgr-068d86d029efe7cd2	SSH	TCP	22	Custom	0.0.0.0/0		Delete
sgr-028c1c0c71b61da4f	HTTP	TCP	80	Custom	0.0.0.0/0		Delete
sgr-024a189925a2b4804	Custom TCP	TCP	3000	Custom	0.0.0.0/0		Delete

Add rule

➔ **MySQL RDS database:**





➔ By using the above **terraform code**, the resources created are:

Security groups:

TESTING INSTANCE SC	
Rules	**Port number**
HTTP	80
SSH	22
Jenkins & Application	8080
Database	3306
DEPLOYMENT INSTANCE SC	
HTTP	80
SSH	22
Grafana	3000
Prometheus	9090
Docker – daemon	9323
Node-exporter	9100

Instances:

- **Testing instance:**
  - **Instance type:** t2.small
  - **Os:** Ubuntu

- **Security group:** testing instance
- **Services and packages installed:**
  - Nodejs
  - Docker
  - Jenkins
  - MySQL

❖ **Deployment instance:**

- **Instance type:** t2.micro
- **Os:** Ubuntu
- **Security group:** deployment instance
- **Services and packages installed:**
  - Docker
  - Node-exporter
  - Grafana
  - Prometheus

❖ **Database: MySQL RDS Database**

- **Database name: nodejs**
- **Database endpoint:** nodejs-db.chttjdyzo3c7.us-east-2.rds.amazonaws.com
- **User:** admin

****

STEP:4 – TESTING THE APPLICATION ON LOCAL ENVIRONMENT

➔ First connecting the testing instance, and checking the required resources has installed or not:

```
root@ip-172-31-3-204:/home/ubuntu# node -v
v16.20.2
root@ip-172-31-3-204:/home/ubuntu# npm -v
8.19.4
root@ip-172-31-3-204:/home/ubuntu# docker -v
Docker version 24.0.5, build 24.0.5-0ubuntu1~20.04.1
root@ip-172-31-3-204:/home/ubuntu# jenkins --version
2.426.2
root@ip-172-31-3-204:/home/ubuntu# java --version
openjdk 17.0.9 2023-10-17
OpenJDK Runtime Environment (build 17.0.9+9-Ubuntu-120.04)
OpenJDK 64-Bit Server VM (build 17.0.9+9-Ubuntu-120.04, mixed mode, sharing)
root@ip-172-31-3-204:/home/ubuntu# mysql
ERROR 2002 (HY000): Can't connect to local MySQL server through socket '/var/run/mysqld/mysqld.sock' (2)
root@ip-172-31-3-204:/home/ubuntu#
```

➔ Then we need to test the application on local environment:

```
root@ip-172-31-3-204:/home/ubuntu# git clone https://github.com/Ravivarman16/mysql-nodejs-application_deployment.git
Cloning into 'mysql-nodejs-application_deployment'...
Username for 'https://github.com': Ravivarman16
Password for 'https://Ravivarman16@github.com':
remote: Enumerating objects: 24, done.
remote: Counting objects: 100% (24/24), done.
remote: Compressing objects: 100% (18/18), done.
remote: Total 24 (delta 1), reused 21 (delta 1), pack-reused 0
Unpacking objects: 100% (24/24), 23.45 KiB | 2.93 MiB/s, done.
root@ip-172-31-3-204:/home/ubuntu# cd mysql-nodejs-application_deployment/
root@ip-172-31-3-204:/home/ubuntu/mysql-nodejs-application_deployment# ls
README.md  app.js  package-lock.json  package.json  public  routes  t_user.sql  views
root@ip-172-31-3-204:/home/ubuntu/mysql-nodejs-application_deployment#
```

➔ First, we need to connect the database:

```
root@ip-172-31-3-204:/home/ubuntu# mysql -u admin -p nodejs -h nodejs-db.chttjdyzo3c7.us-east-2.rds.amazonaws.com
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 15
Server version: 5.7.44 Please upgrade to 8.0 or opt-in to the paid RDS Extended Support service before 5.7 reaches end of standard sup
port on 29 February, 2024: https://a.co/hQqiIn0

Copyright (c) 2000, 2023, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

➔ Then we need to create table by using above database:

```
mysql>
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| innodb             |
| mysql              |
| nodejs             |
| performance_schema |
| sys                |
+--------------------+
6 rows in set (0.00 sec)

mysql> use nodejs
Display all 759 possibilities? (y or n)
mysql> use nodejs;
Database changed
mysql> CREATE TABLE IF NOT EXISTS `t_user` (
    ->   `user_id` int(11) NOT NULL AUTO_INCREMENT,
    ->   `name` varchar(100) NOT NULL,
    ->   `email` varchar(100) NOT NULL,
    ->   `password` varchar(100) NOT NULL,
    ->   `created` DATE,
    ->   PRIMARY KEY (`user_id`)
    -> ) ENGINE=InnoDB  DEFAULT CHARSET=latin1 AUTO_INCREMENT=6 ;
Query OK, 0 rows affected (0.02 sec)

mysql> show tables;
+------------------+
| Tables_in_nodejs |
+------------------+
| t_user           |
+------------------+
1 row in set (0.00 sec)
```

➔ Then on application code we need to make few changes:

app.js contains:

```
app.use(connection(mysql, {
  host: "nodejs-db.chttjdyzo3c7.us-east-2.rds.amazonaws.com",
  user: "admin",
  password: "nodejs123",
  database: "nodejs",
  debug: false
}, "request"));
```

routes/index.js contains:

```
    connection = mysql.createConnection({
      host: "nodejs-db.chttjdyzo3c7.us-east-2.rds.amazonaws.com",
      user: "admin",
      password: "nodejs123",
      database: "nodejs",
      debug: false
    });
```

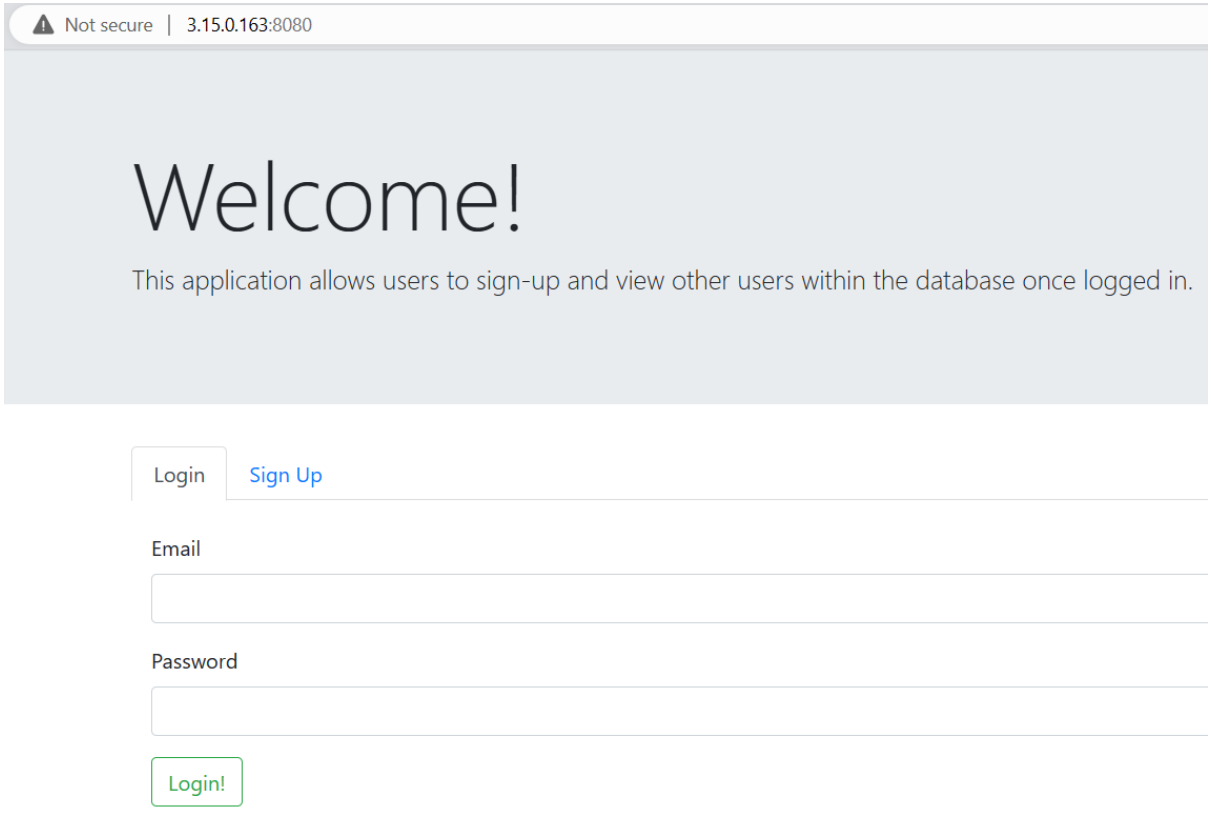
➔ Then we need to install an application dependency by using **npm install** command:

```
root@ip-172-31-3-204:/home/ubuntu/mysql-nodejs-application_deployment# npm install

added 98 packages, and audited 99 packages in 3s

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

3 vulnerabilities (2 moderate, 1 critical)

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.
npm notice
npm notice New major version of npm available! 8.19.4 -> 10.2.5
npm notice Changelog: https://github.com/npm/cli/releases/tag/v10.2.5
npm notice Run npm install -g npm@10.2.5 to update!
npm notice
root@ip-172-31-3-204:/home/ubuntu/mysql-nodejs-application_deployment#
```

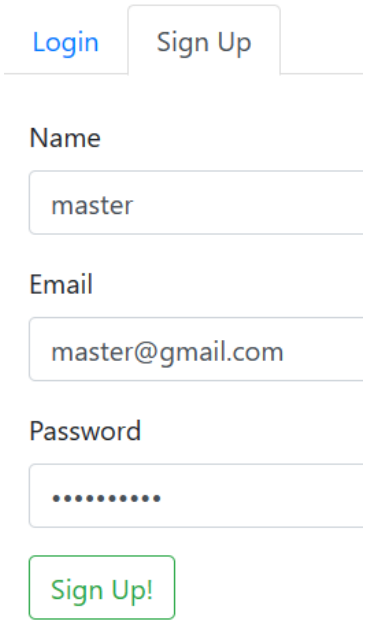
➔ Then we need to start this application by using **node app.js** as command:

```
root@ip-172-31-3-204:/home/ubuntu/mysql-nodejs-application_deployment# node app.js
Server is listening on port 8080
Database is connected
```

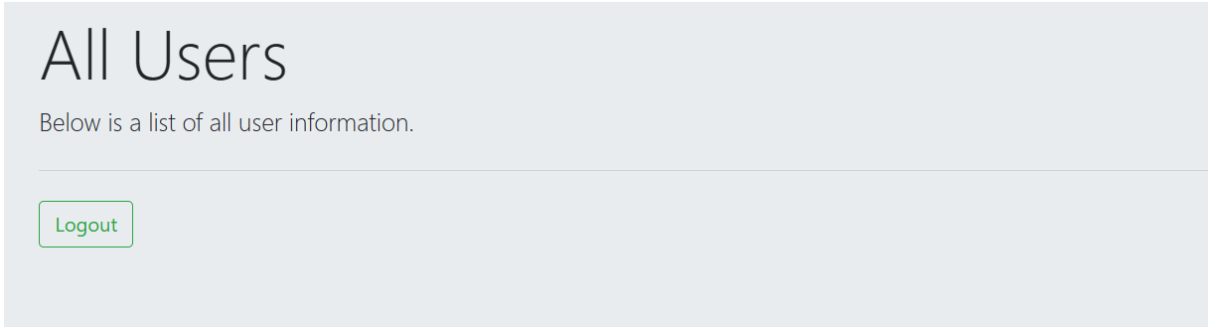
Application output:



➔ Checking the application by entering the signup details:

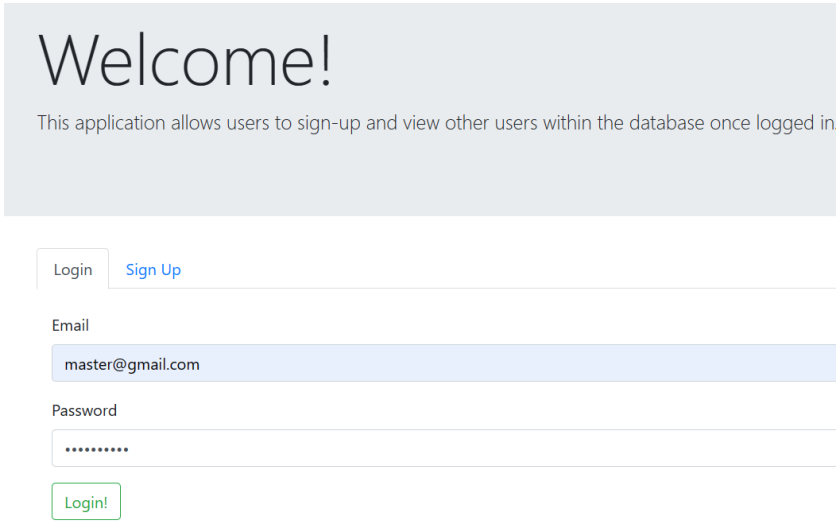


➔ The details have been added: then click the logout option:

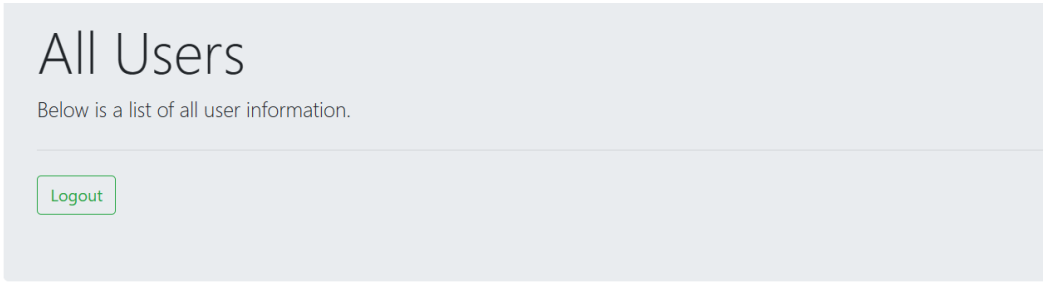


ID	Name	Email	Password	Date Added
1	master	master@gmail.com	master1234	Sat Dec 23 2023 00:00:00 GMT+0000 (Coordinated Universal Time)

➔ Now try to login with those credentials:



➔ We can able to login successfully with those credentials. The application is working fine:



ID	Name	Email	Password	Date Added
1	master	master@gmail.com	master1234	Sat Dec 23 2023 00:00:00 GMT+0000 (Coordinated Universal Time)

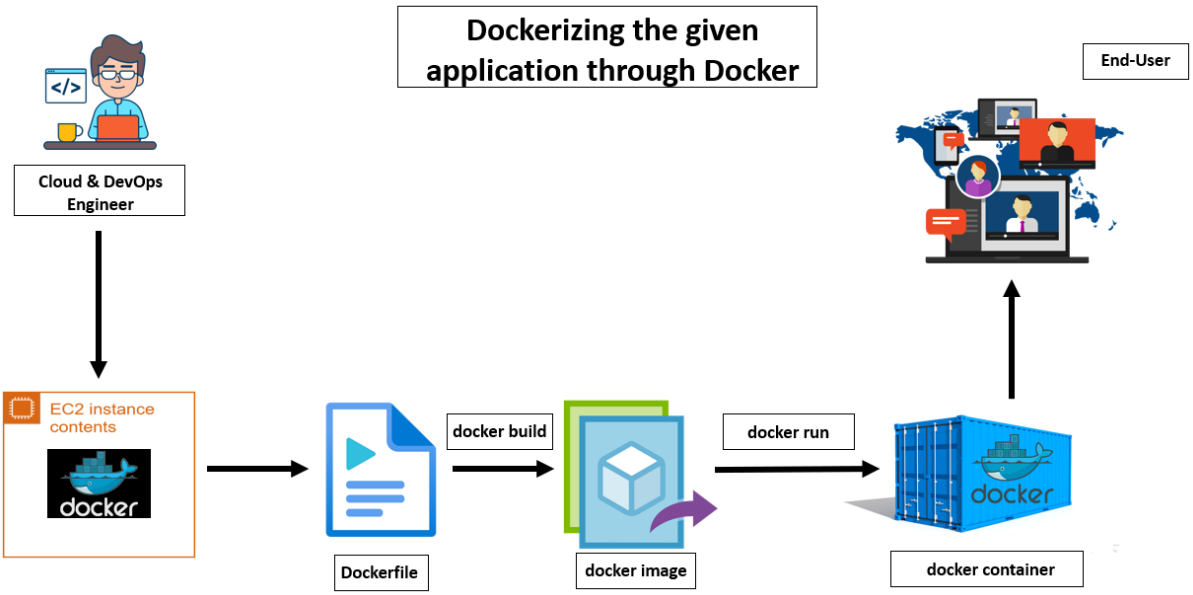
➔ The command line output of the application:

```
Database is connected
[
  RowDataPacket {
    user_id: 6,
    name: 'master',
    email: 'master@gmail.com',
    password: 'master1234',
    created: 2023-12-23T00:00:00.000Z
  }
]
master@gmail.com
```

****

STEP:5 – DOCKERIZING THE APPLICATION THROUGH DOCKER

OVERVIW – DIAGRAM



➔ Now we need to Dockerize this application, for that we need to create a dockerfile:

Dockerfile contains:

```
#choosing the base image:
FROM node:16-alpine
```

```
#choosing working directory for the application:
WORKDIR /app

#copying the application code:
COPY . .

#installing the dependencies:
RUN npm install

#exposing the application:
EXPOSE 8080

#starting the application:
CMD [ "node", "app.js" ]
```

➔ Then building a docker image by using above dockerfile with the command:

docker build -t <image name> .

```
root@ip-172-31-3-204:/home/ubuntu/mysql-nodejs-application_deployment# vi dockerfile
root@ip-172-31-3-204:/home/ubuntu/mysql-nodejs-application_deployment# docker build -t ravivarman46/mysql-nodejs:app1 .
DEPRECATED: The legacy builder is deprecated and will be removed in a future release.
            Install the buildx component to build images with BuildKit:
            https://docs.docker.com/go/buildx/

Sending build context to Docker daemon  11.31MB
Step 1/6 : FROM node:16-alpine
16-alpine: Pulling from library/node
7264a8db6415: Pull complete
eee371b9ce3f: Pull complete
93b3025fe103: Pull complete
d9059661ce70: Pull complete
Digest: sha256:a1f9d027912b58a7c75be7716c97cfbc6d3099f3a97ed84aa490be9dee20e787
Status: Downloaded newer image for node:16-alpine
 ---> 2573171e0124
Step 2/6 : WORKDIR /app
 ---> Running in 9c87d8939a98
Removing intermediate container 9c87d8939a98
 ---> ea94cbfbd545
Step 3/6 : COPY . .
 ---> 3f34204bc5db
Step 4/6 : RUN npm install
 ---> Running in 171566b75bc0

up to date, audited 99 packages in 1s

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

➔ Checking the image is created or not by using the command:

docker images

```
Step 6/6 : CMD [ "node", "app.js" ]
 ---> Using cache
 ---> 029ef293c265
Successfully built 029ef293c265
Successfully tagged ravivarman46/mysql-nodejs:app1
root@ip-172-31-3-204:/home/ubuntu/mysql-nodejs-application_deployment# docker images
REPOSITORY                   TAG         IMAGE ID       CREATED          SIZE
ravivarman46/mysql-nodejs    app1        029ef293c265   3 minutes ago    130MB
node                         16-alpine   2573171e0124   4 months ago     118MB
root@ip-172-31-3-204:/home/ubuntu/mysql-nodejs-application_deployment#
```

➔ Then a running a container by using above image:

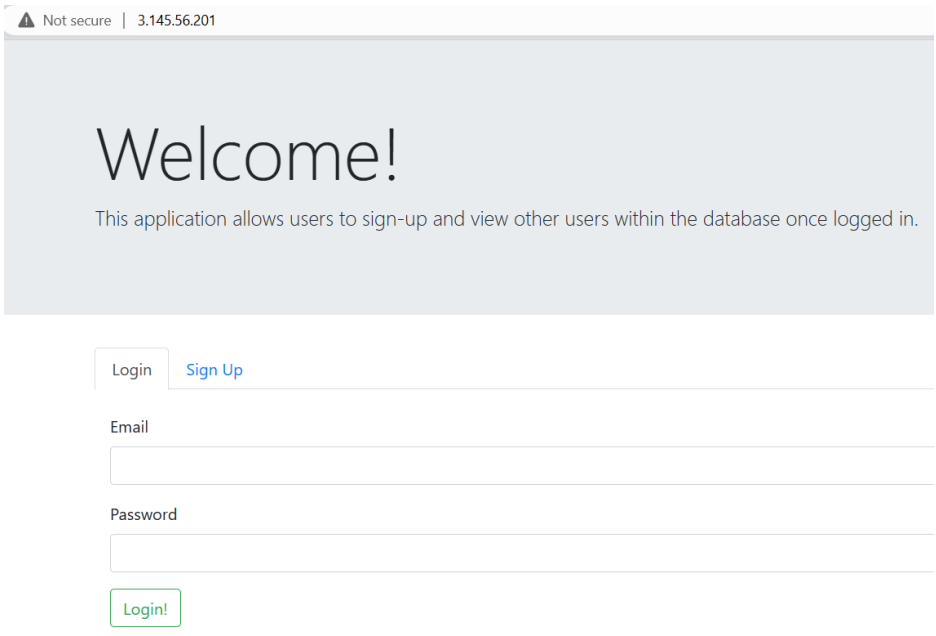
docker run -d -it --name nodejs -p 80:8080 <image name>

```
root@ip-172-31-3-204:/home/ubuntu/mysql-nodejs-application_deployment# docker run -d -it --name nodejs -p 80:8080 ravivarman46/mysql-n
odejs:app1
c87dc1b6fd3db2bd15e62b75ea03f0a63c5c610bf397da886b8d02a936c00018
root@ip-172-31-3-204:/home/ubuntu/mysql-nodejs-application_deployment# docker ps
CONTAINER ID   IMAGE                          COMMAND                  CREATED         STATUS         PORTS                                   NA
MES
c87dc1b6fd3d   ravivarman46/mysql-nodejs:app1   "docker-entrypoint.s…"   8 seconds ago   Up 7 seconds   0.0.0.0:80->8080/tcp, :::80->8080/tcp   no
dejs
```

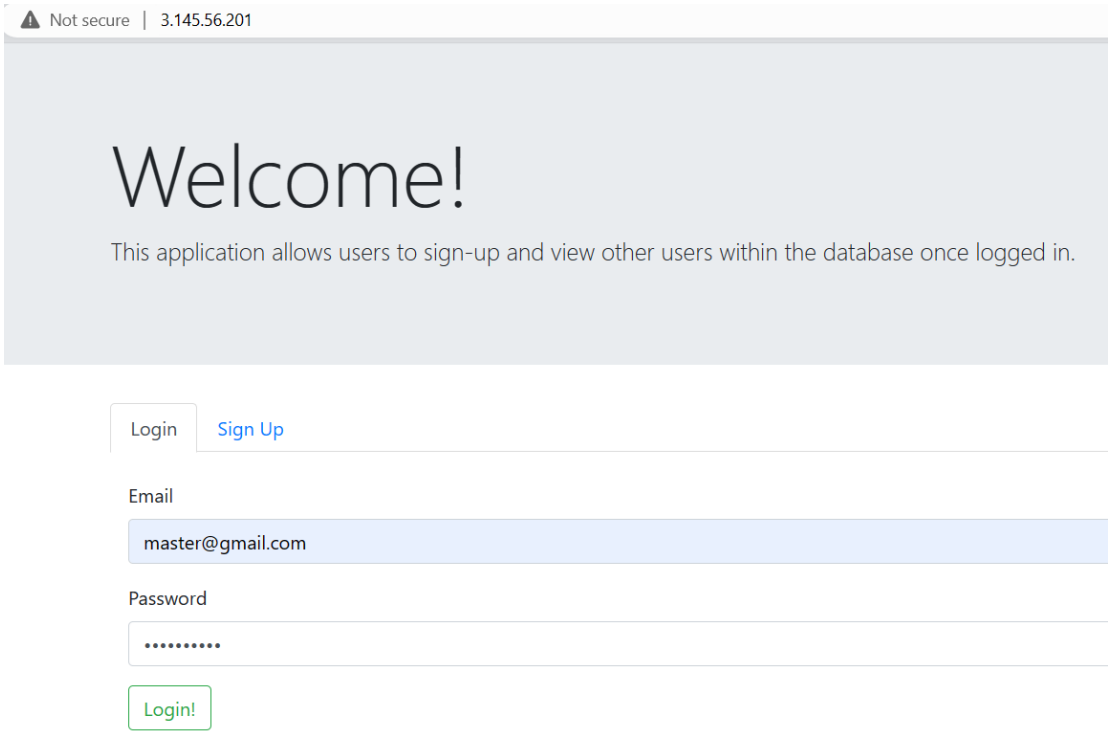
➔ Container output checking, by using previous credentials created while testing in local environment:

Email id: master@gmail.com

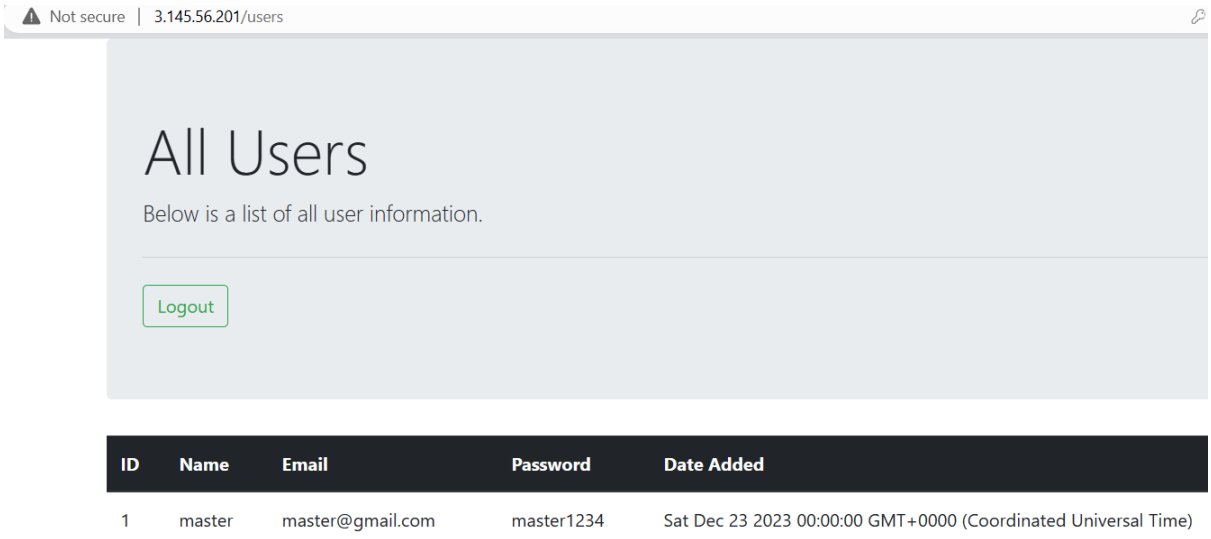
Password: master1234



➔ Entering the above details in login session and click login:



➔ We can able to login into the application with the previous credentials:

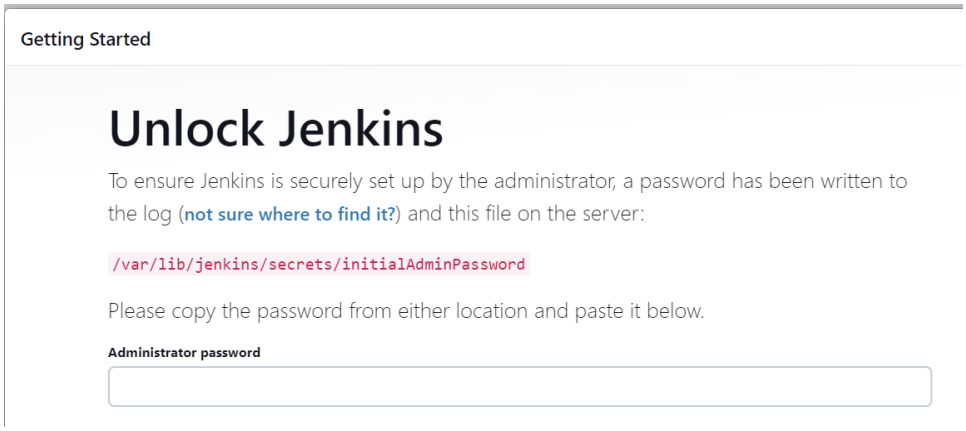


Dockerization had been successfully

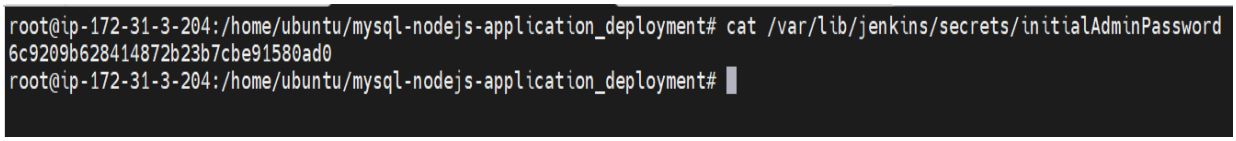
****

STEP:6 – SETTING UP THE JENKINS CONSOLE

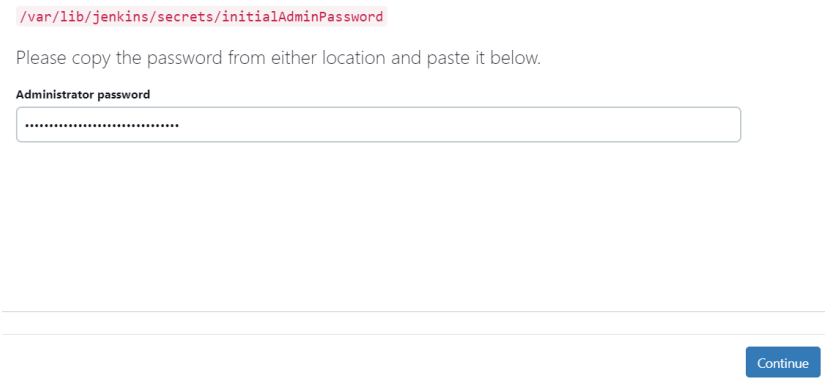
➔ Copy the public ip address of the testing instance, paste it on browser:



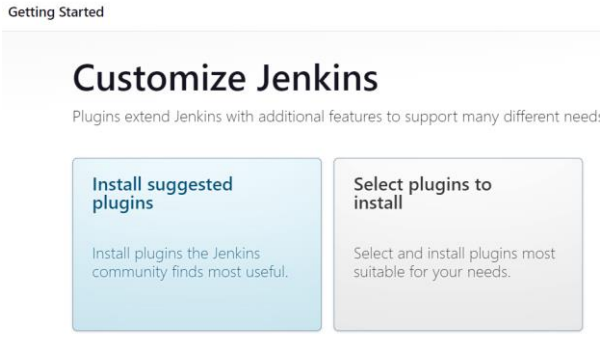
➔ Then copy the path and paste it on command line along with **cat** command to get the password:



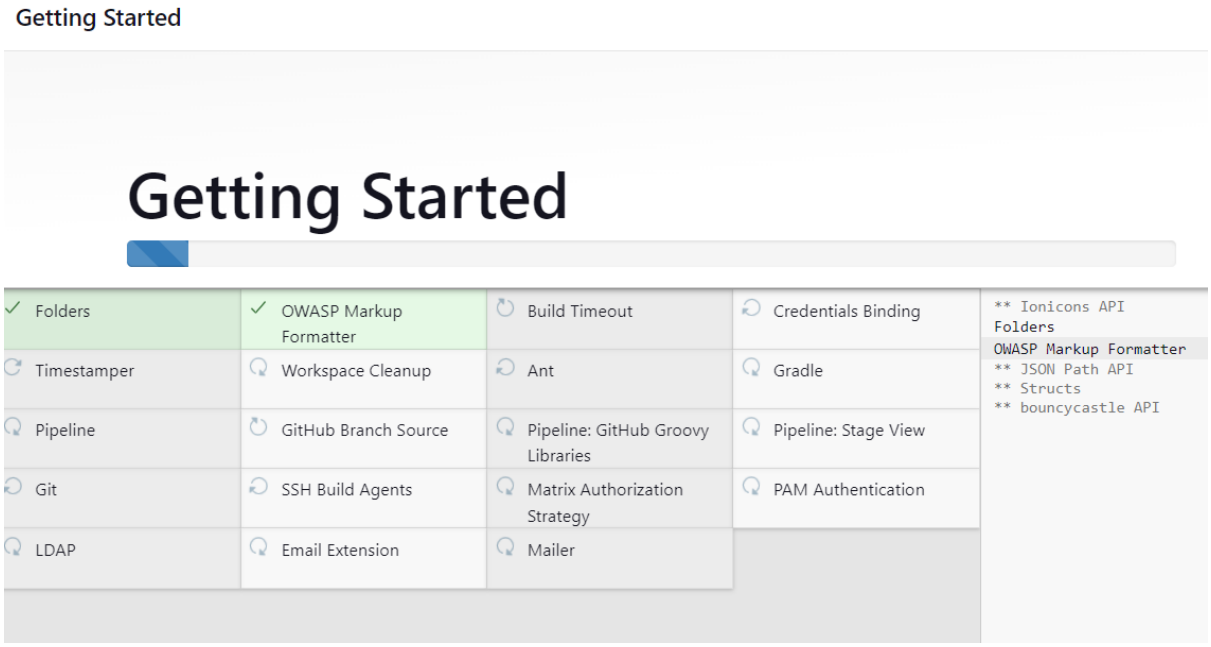
➔ Then click continue after the entering the password:



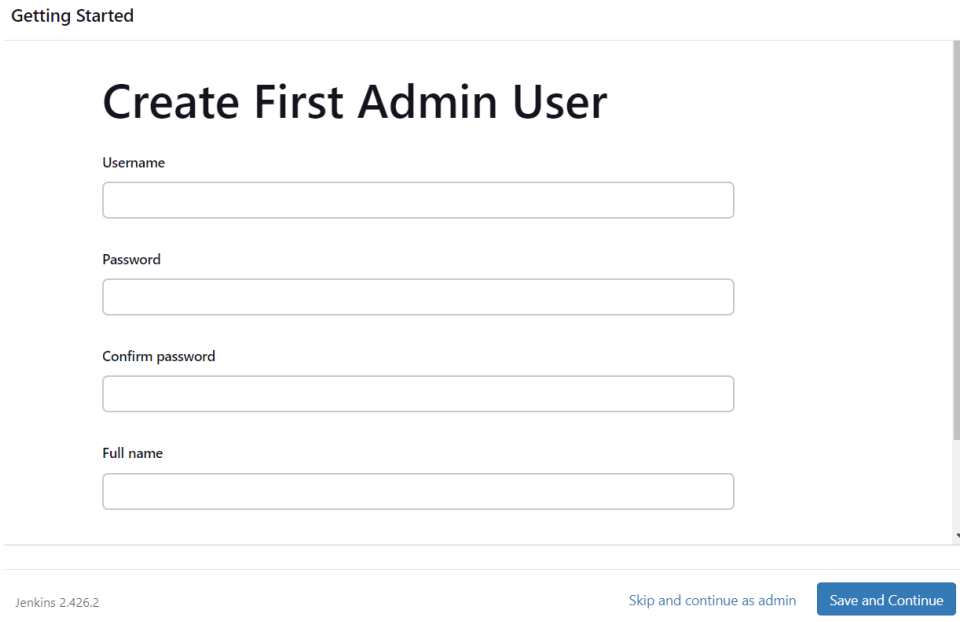
➔ Then select **install suggested plugins** option:



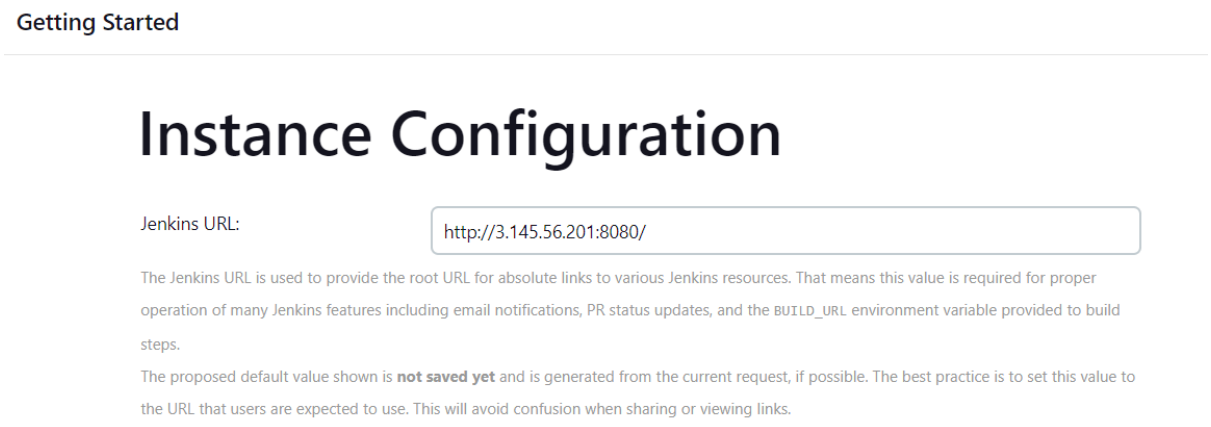
➔ The plugins will start to download:



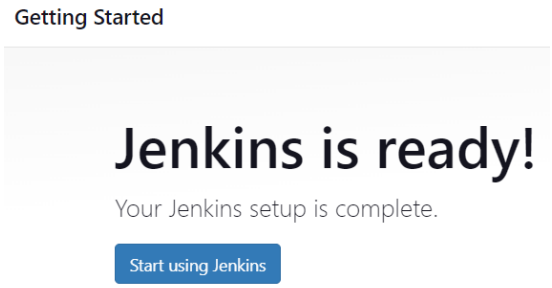
➔ Then setup the **login credentials for Jenkins**, after that click save and continue:



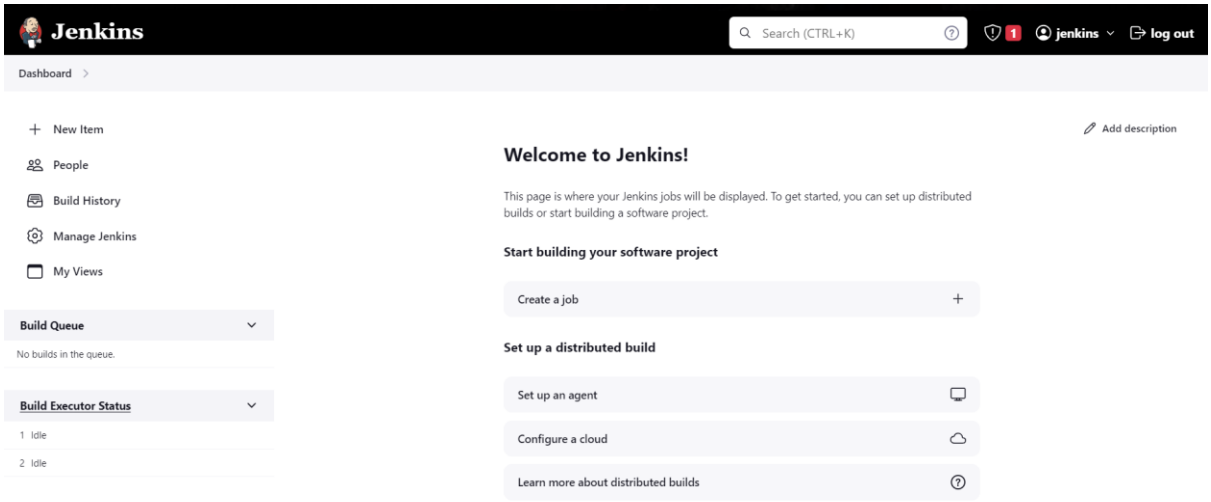
➔ Then click save and finish:



➔ Then the Jenkins console is ready for using, click **start using Jenkins option:**



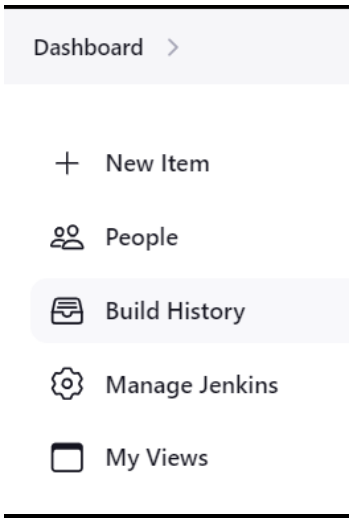
➔ Jenkins console is available for using:



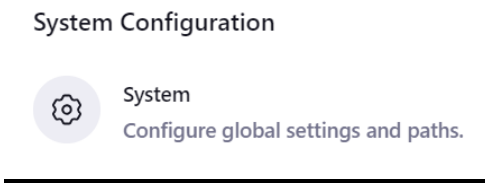
****

STEP:7 – SETTING UP THE DOCKERHUB, GITHUB & SSH CREDENTIALS:

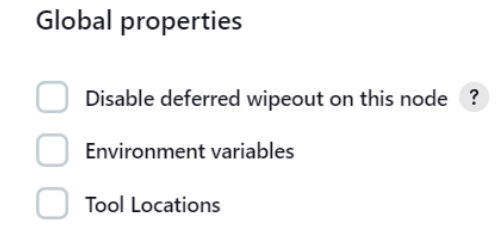
➔ On Jenkins dashboard, we can able to see **manage Jenkins**, click that one:



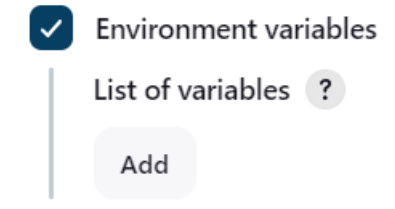
➔ Then click **system** on manage Jenkins page:



➔ Then click **environment variables** under **global properties:**



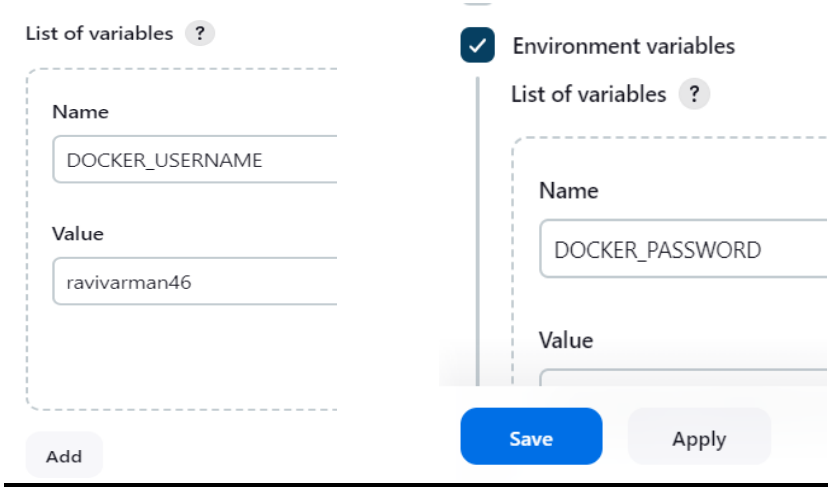
➔ Then click **add** option:



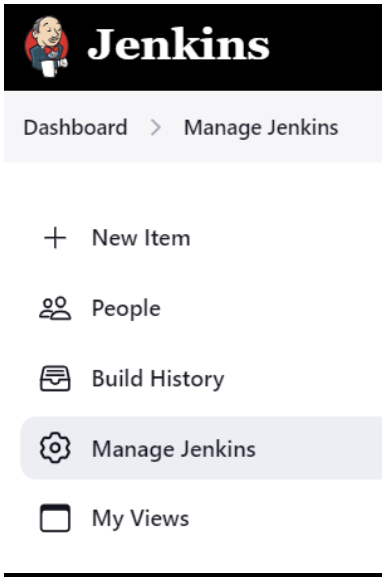
➔ Then enter the environmental variables for **DockerHub credentials:**

- ❖ **DOCKER_USERNAME**
- ❖ **DOCKER_PASSWORD**

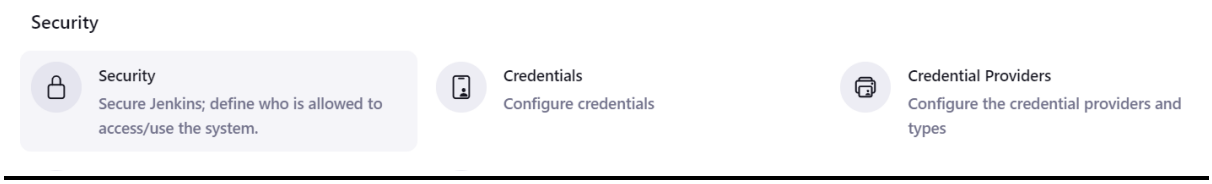
After adding the environment variables**, click apply & save:**



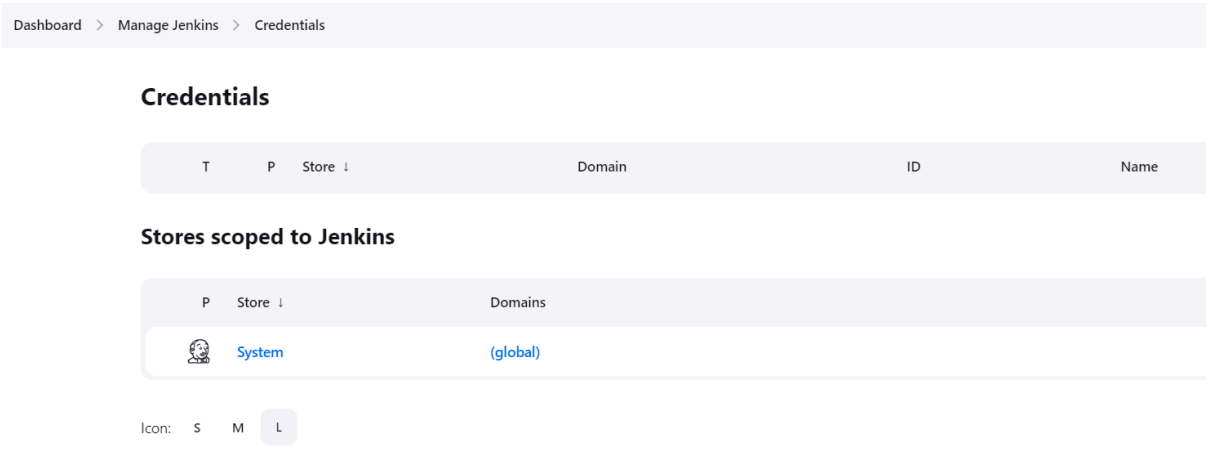
➔ For adding **GitHub credentials**, click **manage Jenkins:**



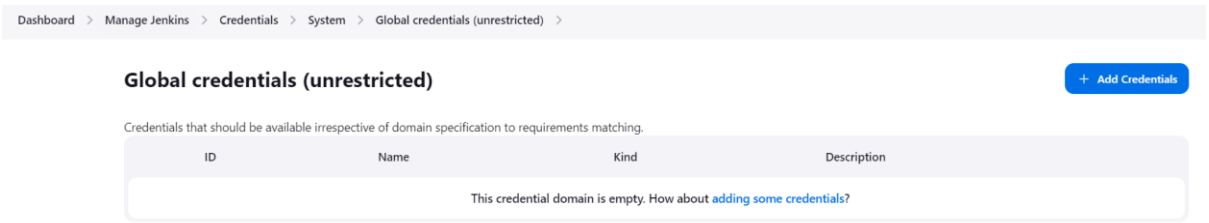
➔ Then under manage Jenkins, click **credentials:**



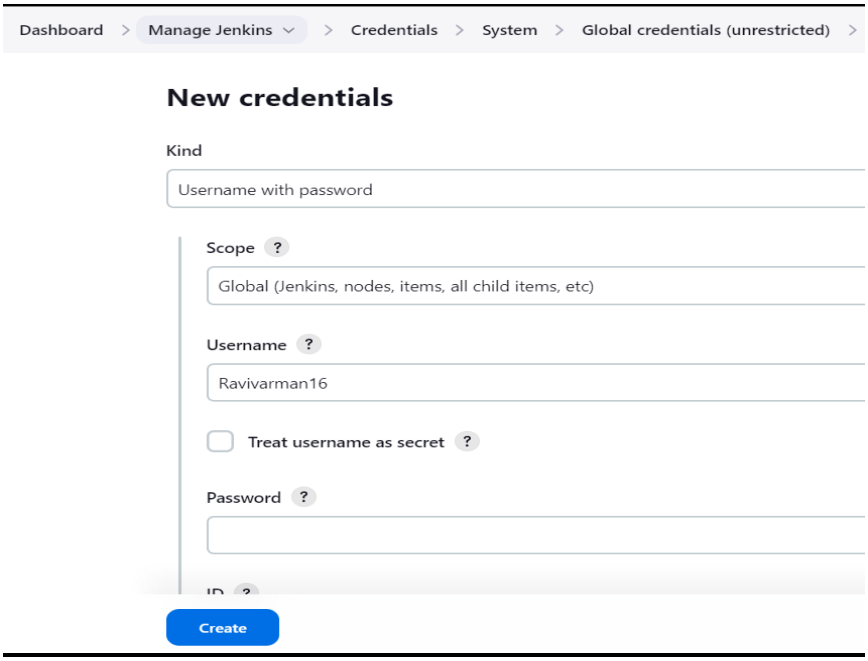
➔ Then click **global:**



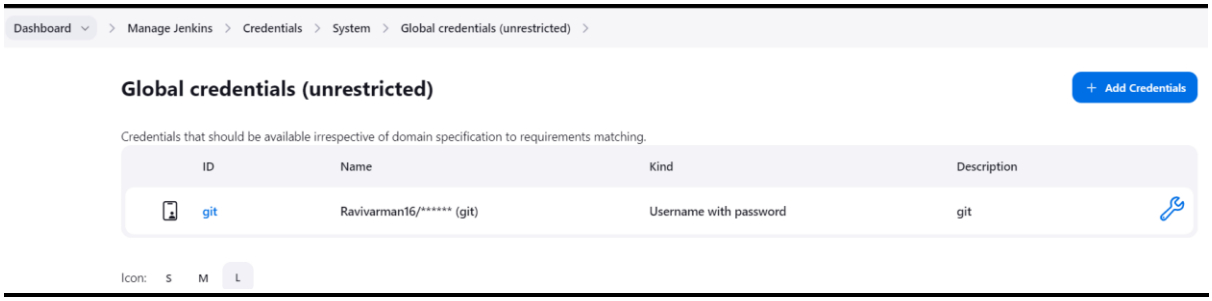
➔ Then click **add credentials:**



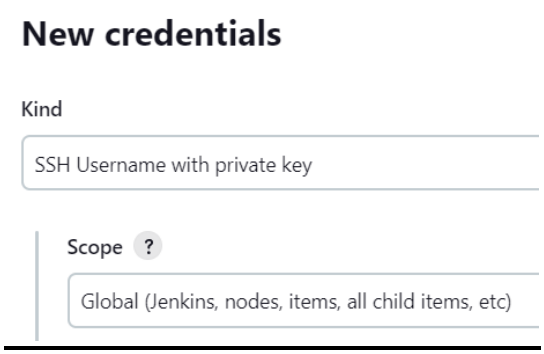
➔ Then add **GitHub Credentials**, after adding the credentials click **create** option:



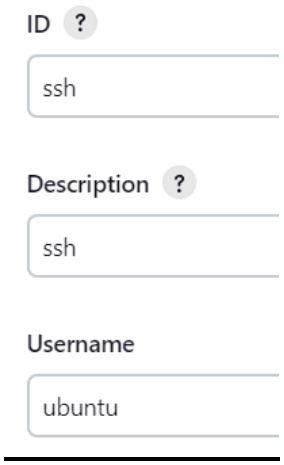
➔ The GitHub credentials has been added successfully:



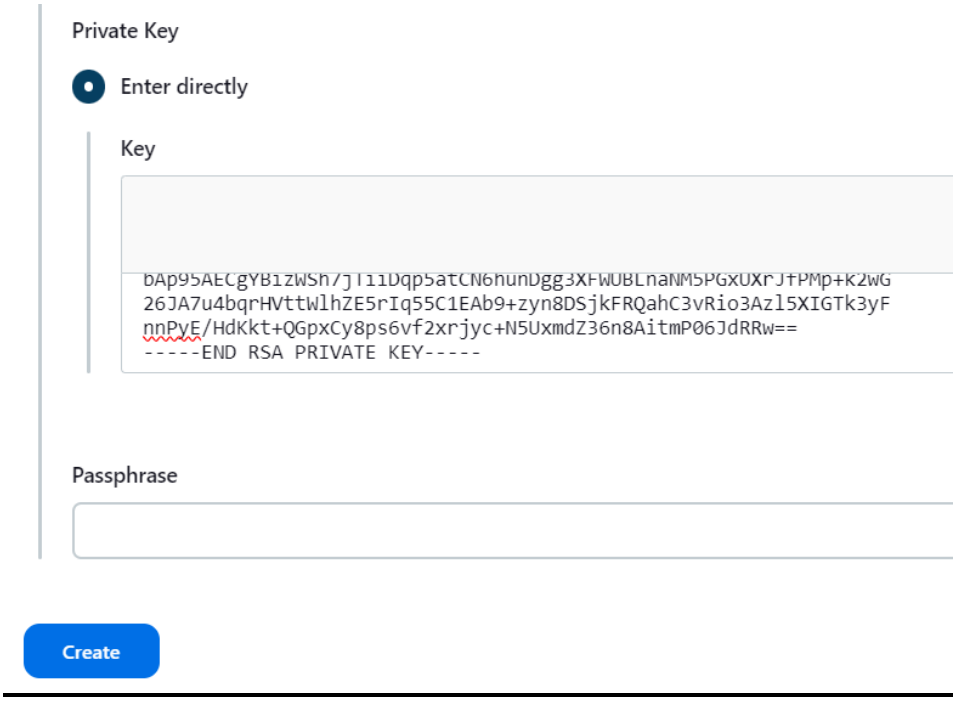
➔ Like this we need to add SSH credentials: under **kind select SSH Username with private key:**



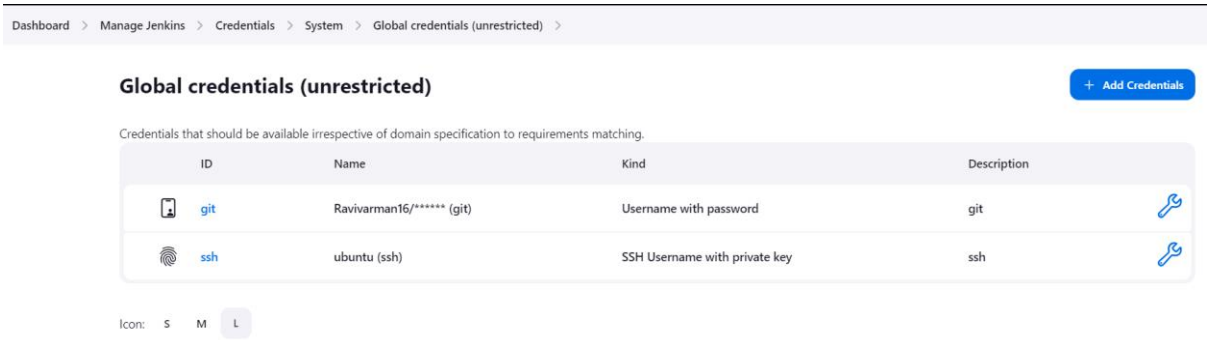
➔ Under ID, I am mentioning it as **ssh**:



➔ Then under private key, I am adding the **keypair of the deployment instance:** after that click create:



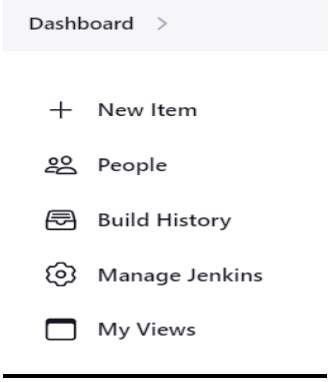
➔ The ssh credentials has been added successfully:



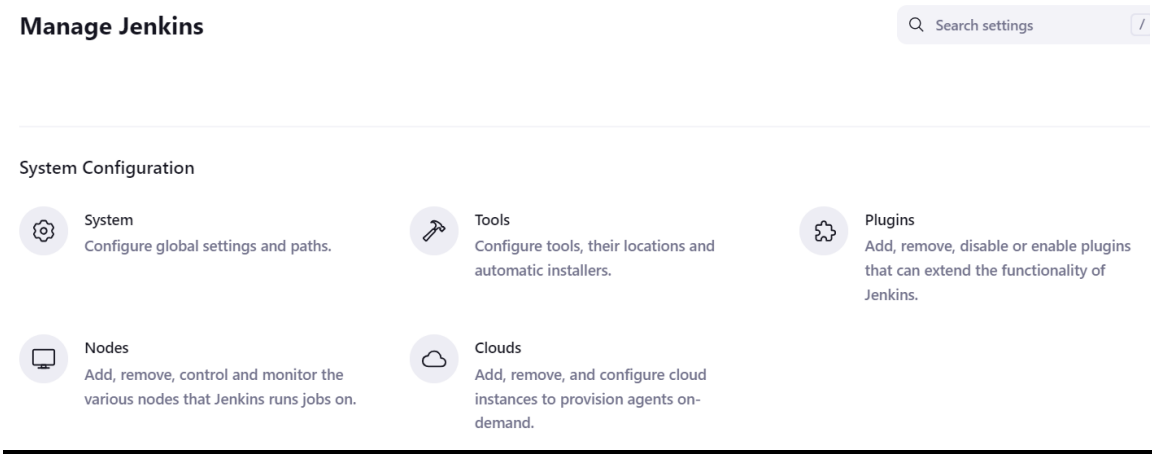
****

STEP:8 - SETTING UP SSH CONNECTION

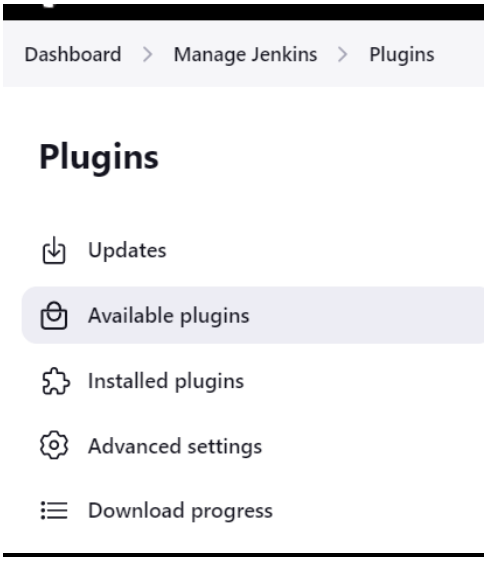
➔ For setting up **SSH connection**, we need to download **two plugins**, for that click **manage Jenkins:**



➔ Under manage Jenkins, click **plugins:**



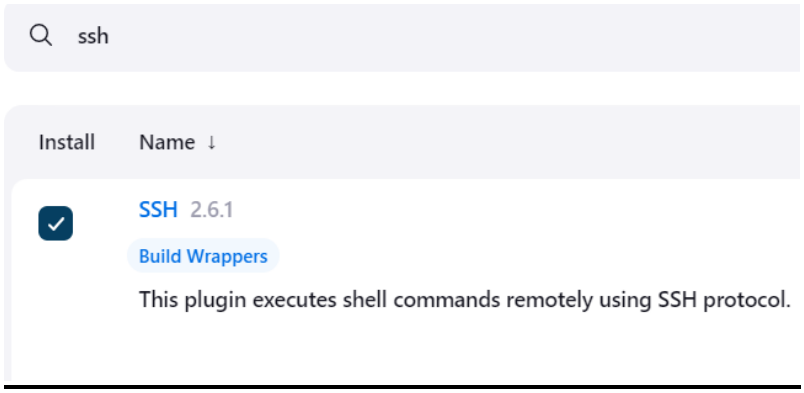
➔ Then under plugins, click **available plugins:**

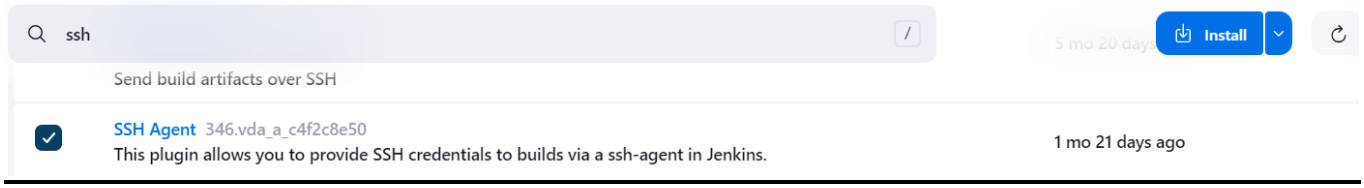


➔ We need to download these **two plugins** for SSH connection to an **EC2 instance:**

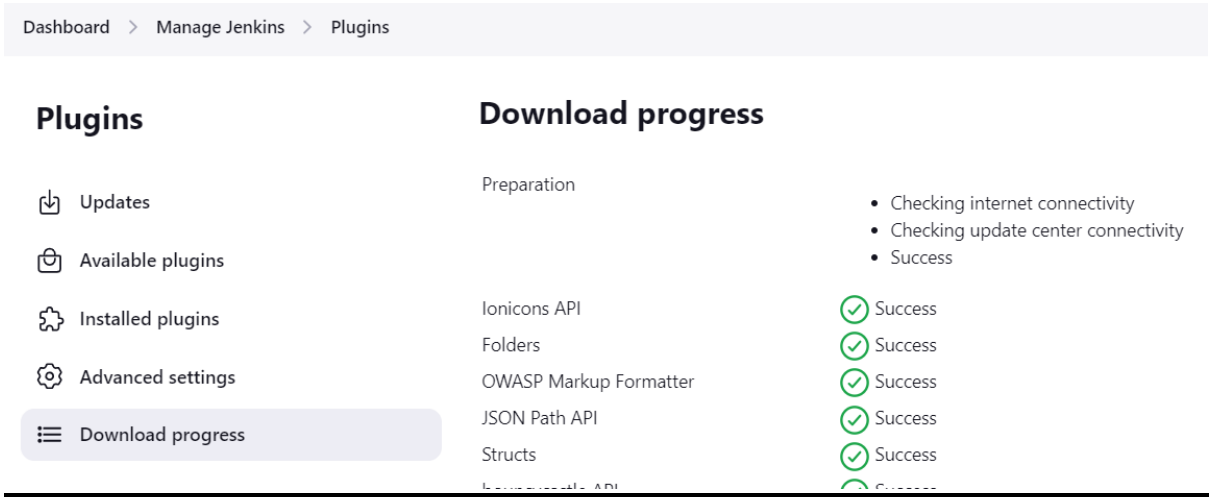
- ❖ **SSH**
- ❖ **SSH Agent**

After selecting these two plugins, click **install** option:

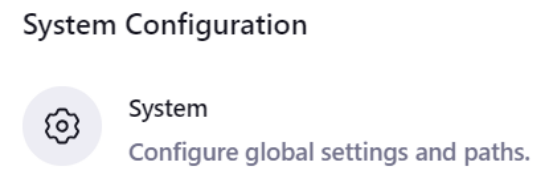




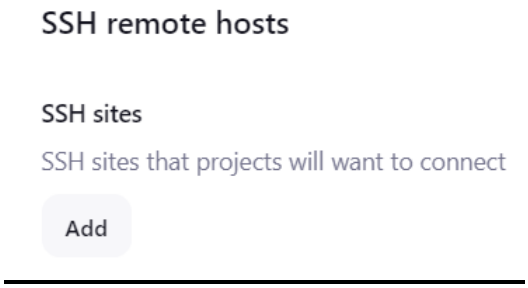
➔ The plugins have been installed successfully:



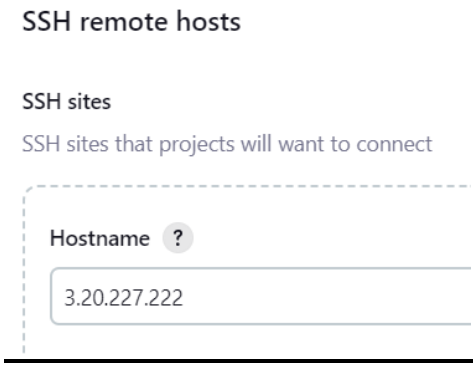
➔ Then under **manage Jenkins click system:**



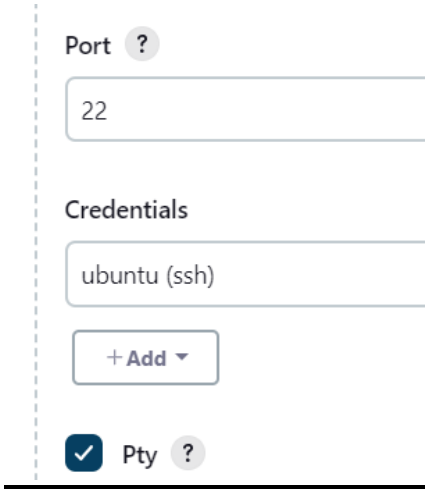
➔ Then under SSH remote hosts, click that add option:



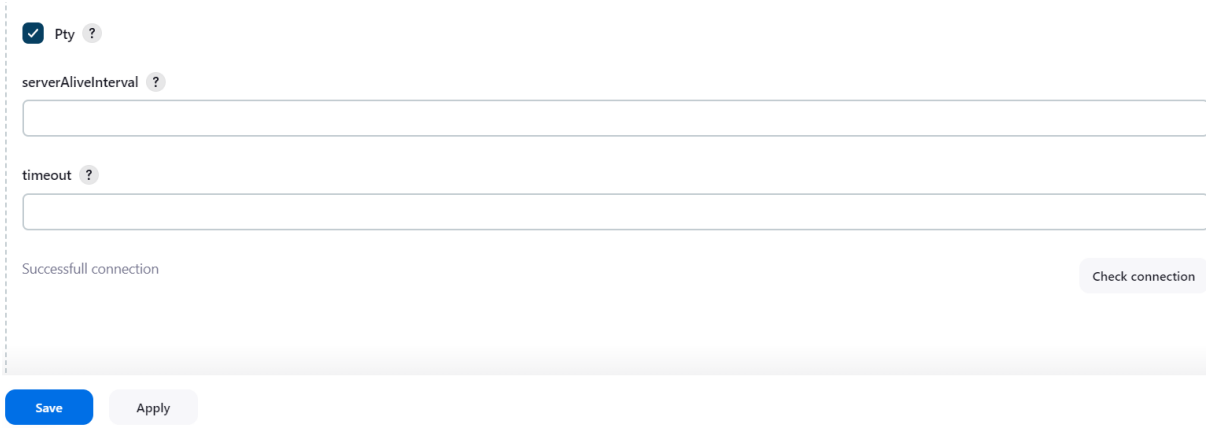
➔ Then under **hostname**, enter the **public Ip of deployment instance:**



➔ Then under **port** enter **SSH port number 22**, under credentials select the SSH Credentials we added.



➔ Then on left side we can able to **check connection,** click that one to check whether the **testing instance can reach to deployment instance:** we should able to see **successful connection**



****

STEP:9 - CREATING A JENKINSFILE FOR CI/CD PIPELINE

➔ We need to create Jenkinsfile for CI/CD Pipeline by using above application:

Jenkinsfile contains:

```
pipeline {
    agent any

    stages {

        stage ('Docker login & Building image') {
            steps {
                sh 'docker login -u $DOCKER_USERNAME -p
$DOCKER_PASSWORD'
                sh 'docker build -t $DOCKER_USERNAME/mysql-
nodejs:app1 .'
            }
        }

        stage ('Pushing the Docker image to DockerHub') {
            steps {
                sh 'docker push $DOCKER_USERNAME/mysql-
nodejs:app1'
            }
        }
```

```
        stage('Deploy to Deployment server') {
            steps {
                script {
                    sshagent(['ssh']) {
                        // Execute the command within the
sshagent block using sh step
                        sh 'ssh -o StrictHostKeyChecking=no
ubuntu@3.20.227.222 "docker run -d -it --name nodejs -p
80:8080 ravivarman46/mysql-nodejs:app1"'
                    }
                }
            }
        }
    }
}
```

➔ Then pushing the **contains and source files to GitHub repository:**

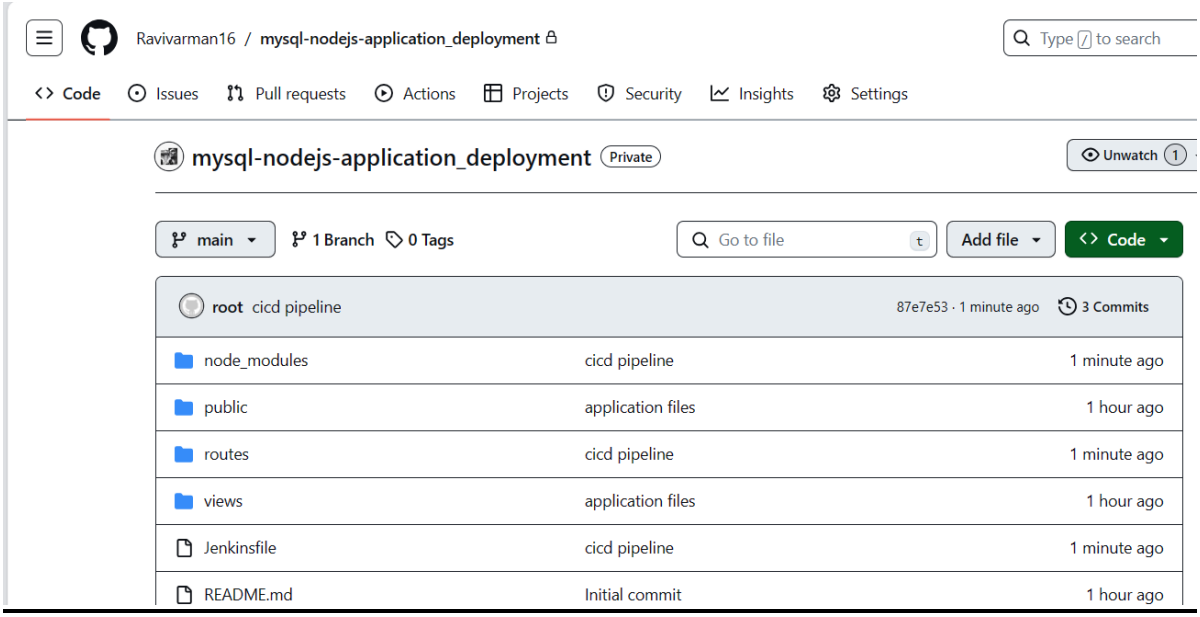
```
root@ip-172-31-3-204:/home/ubuntu/mysql-nodejs-application_deployment# vi Jenkinsfile
root@ip-172-31-3-204:/home/ubuntu/mysql-nodejs-application_deployment# ls
Jenkinsfile  README.md  app.js  dockerfile  node_modules  package-lock.json  package.json  public  routes  t_user.sql  views
root@ip-172-31-3-204:/home/ubuntu/mysql-nodejs-application_deployment#
```

```
root@ip-172-31-3-204:/home/ubuntu/mysql-nodejs-application_deployment# git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   app.js
        modified:   routes/index.js

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        Jenkinsfile
        dockerfile
        node_modules/

no changes added to commit (use "git add" and/or "git commit -a")
root@ip-172-31-3-204:/home/ubuntu/mysql-nodejs-application_deployment# git add .
root@ip-172-31-3-204:/home/ubuntu/mysql-nodejs-application_deployment#
```

```
root@ip-172-31-3-204:/home/ubuntu/mysql-nodejs-application_deployment# git commit -m "cicd pipeline"
[main 87e7e53] cicd pipeline
 Committer: root <root@ip-172-31-3-204.us-east-2.compute.internal>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
```

```
root@ip-172-31-3-204:/home/ubuntu/mysql-nodejs-application_deployment# git push origin main
Username for 'https://github.com': Ravivarman16
Password for 'https://Ravivarman16@github.com':
Enumerating objects: 2116, done.
Counting objects: 100% (2116/2116), done.
Compressing objects: 100% (1998/1998), done.
Writing objects: 100% (2112/2112), 1.96 MiB | 4.68 MiB/s, done.
Total 2112 (delta 506), reused 0 (delta 0)
remote: Resolving deltas: 100% (506/506), completed with 2 local objects.
To https://github.com/Ravivarman16/mysql-nodejs-application_deployment.git
   38b9dfe..87e7e53  main -> main
root@ip-172-31-3-204:/home/ubuntu/mysql-nodejs-application_deployment#
```

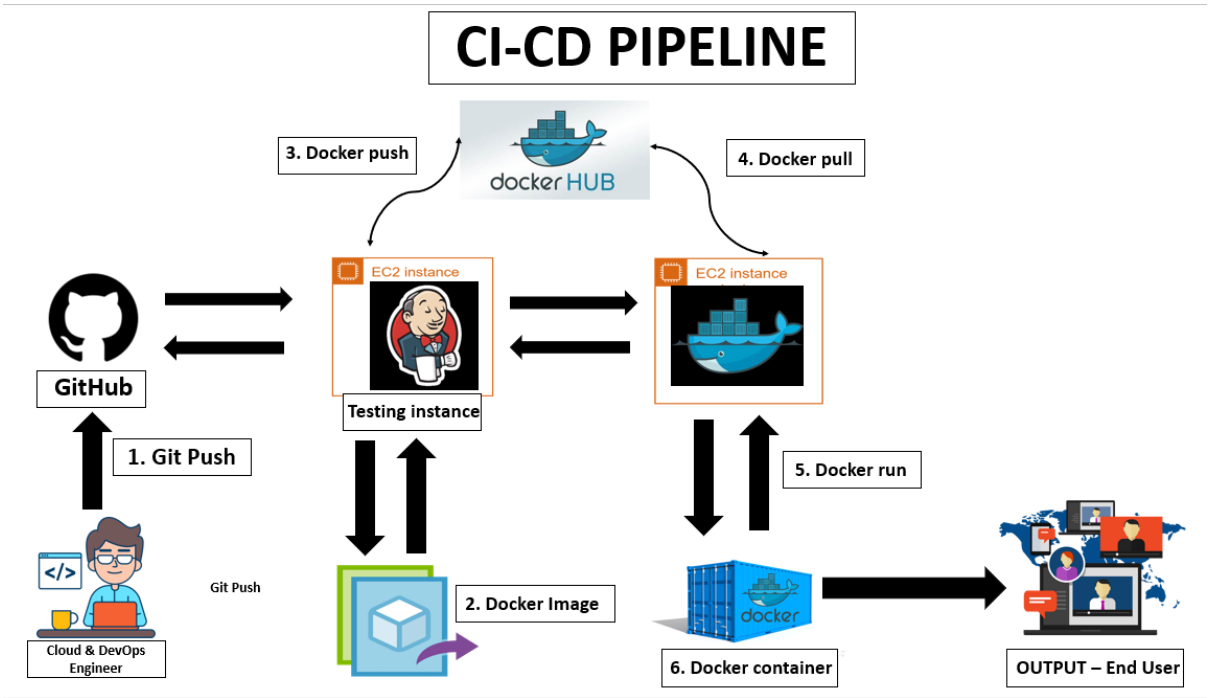
➔ source code has been pushed to GitHub repository from local repository successfully. Confirm it by checking in GitHub



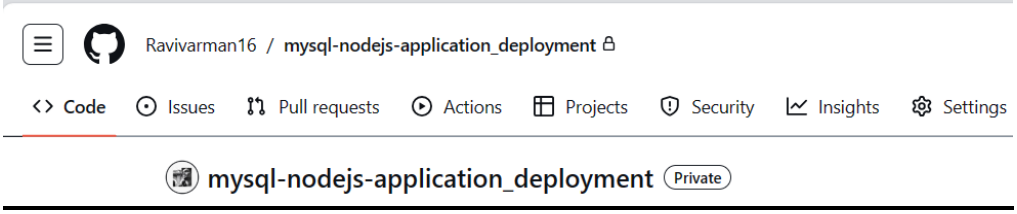
****

STEP:10 - CREATING A CI/CD PIPELINE IN JENKINS

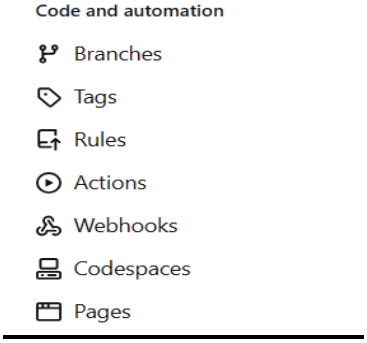
OVERVIEW – DIAGRAM



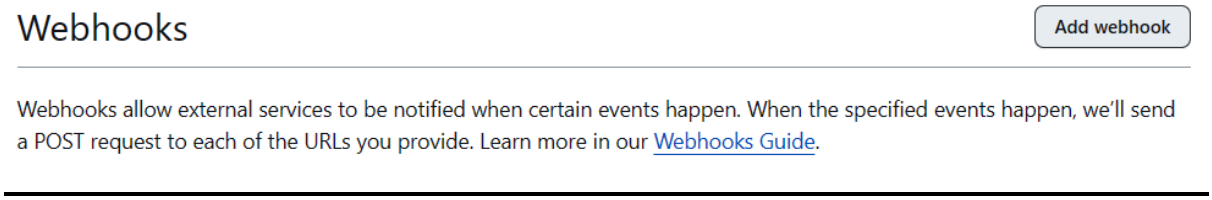
➔ On GitHub repository, click settings:



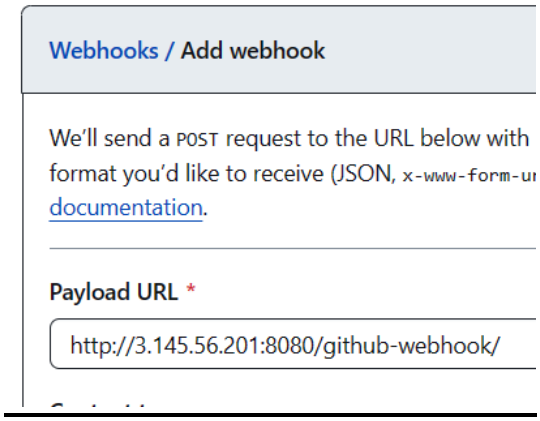
➔ Then click **webhooks** on left side of screen:



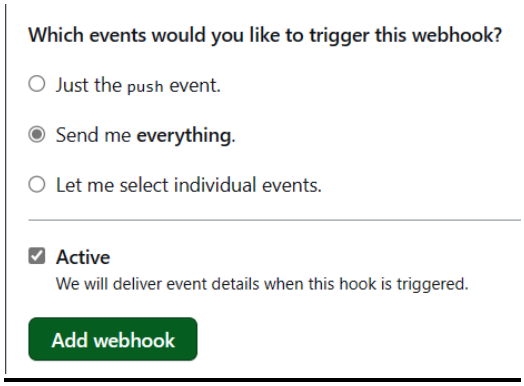
➔ Then click **add webhook:**



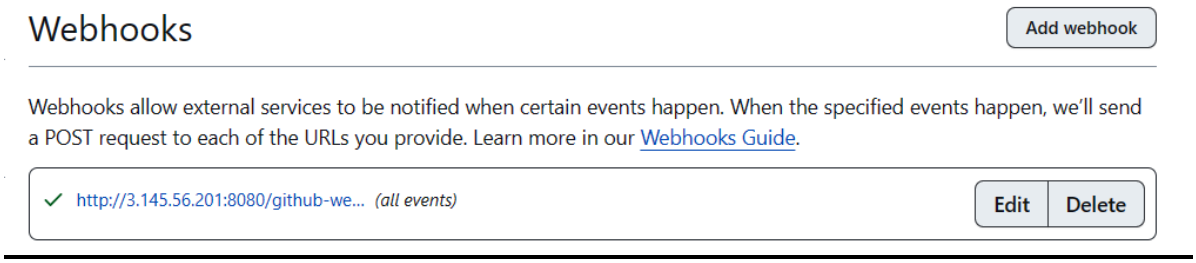
➔ Then under **payload URL,** enter **Jenkins URL along with github-webhook:**



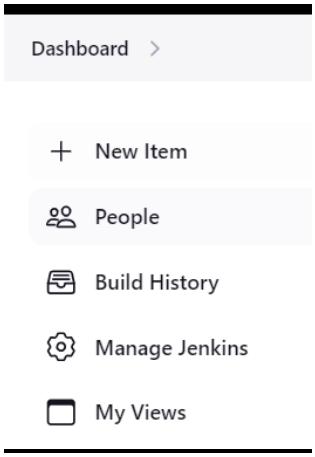
➔ Then select the **event** according to your preferences:



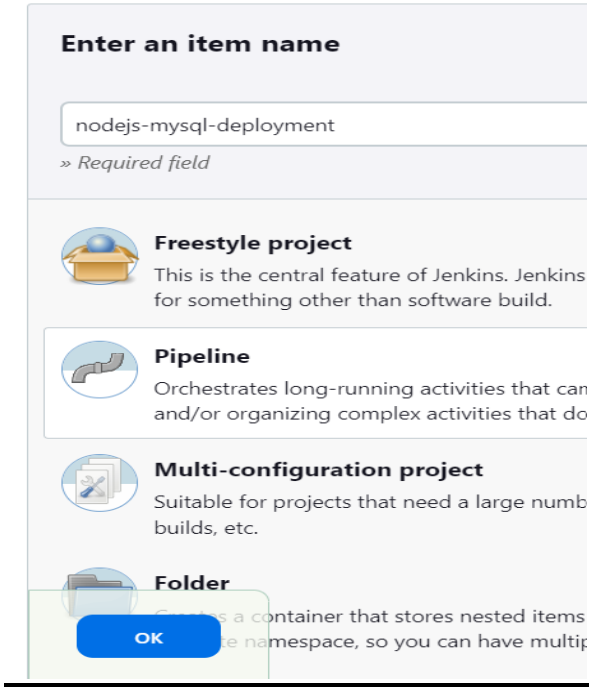
➔ We can able to see webhooks has been successfully for **automatic trigger:**



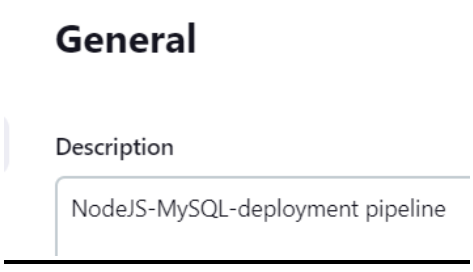
➔ Then on **Jenkins dashboard**, click **new item** to create a new job:



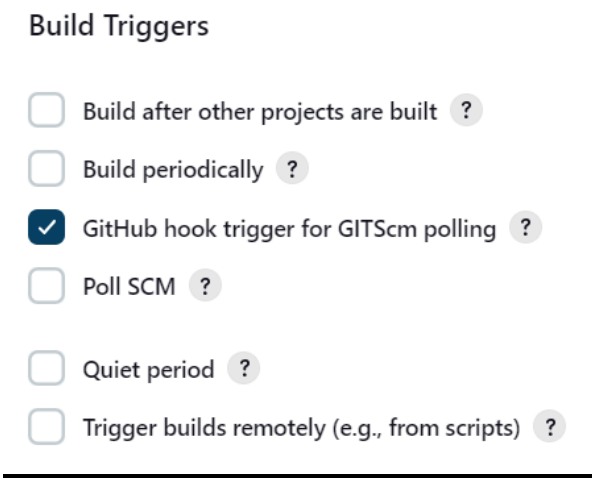
➔ Then **enter name for the job**, select **pipeline and click okay:**



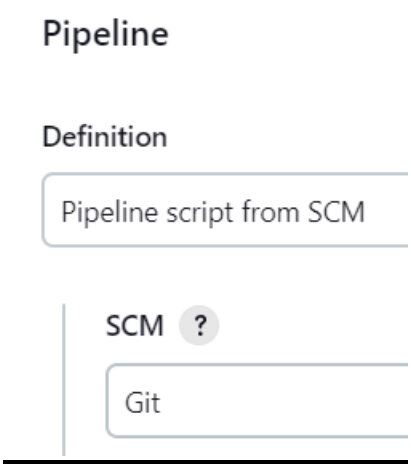
➔ Then under **General**, give **description** according to the job:



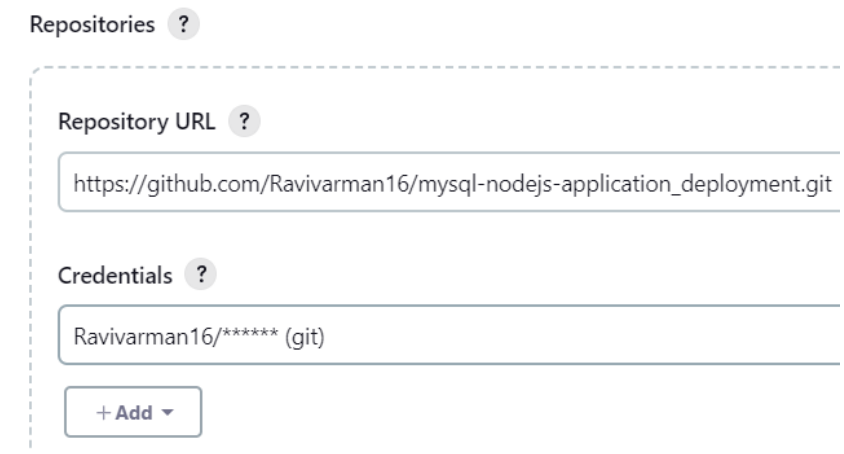
➔ Then under **build triggers,** select **GitHub hook trigger for GITScm polling option:**



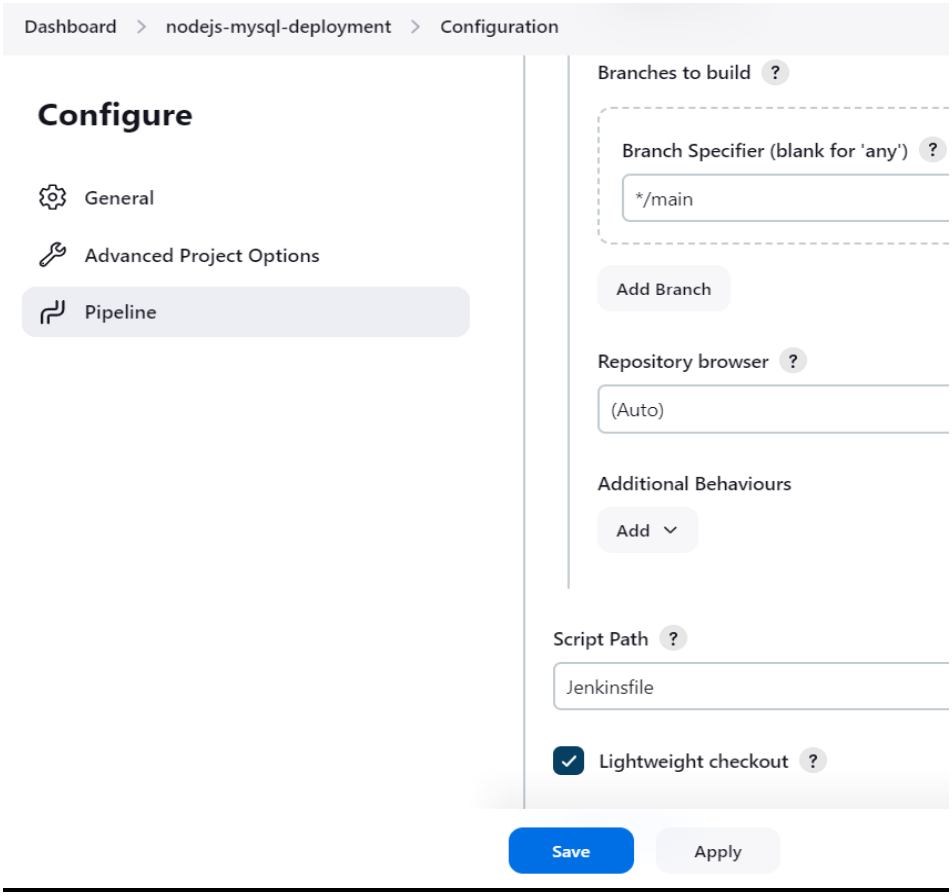
➔ Then under **Pipeline** select **Pipeline script from SCM, select Git:**



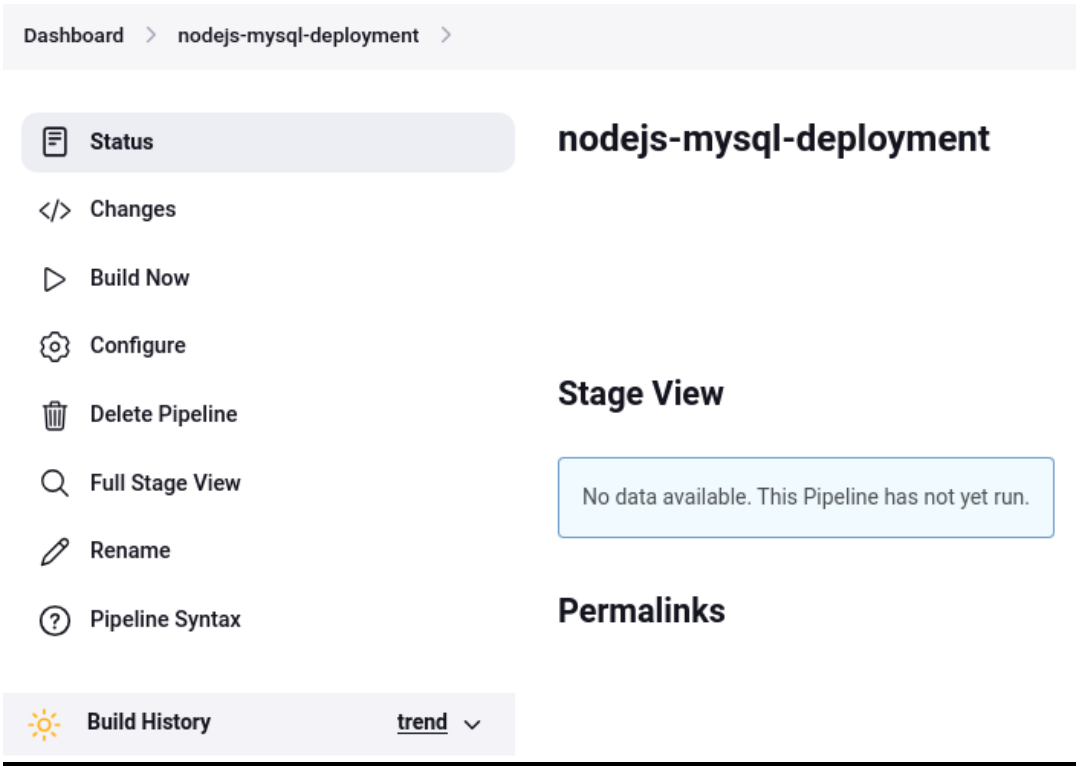
➔ Then enter the **GitHub repository link** under **Repository URL:**



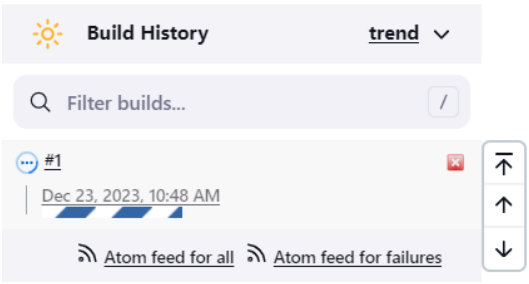
➔ then enter the **branch** according to your GitHub repository:



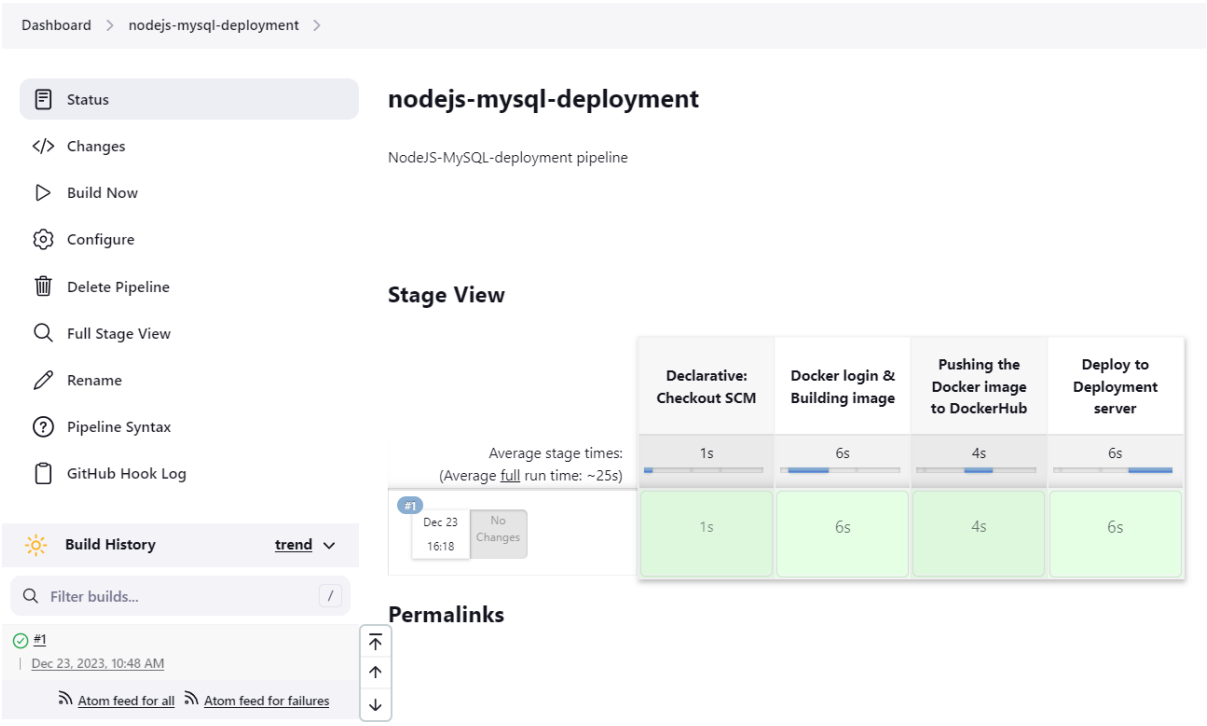
➔ Pipeline job has been created successfully, now click **build now** option to start pipeline job:



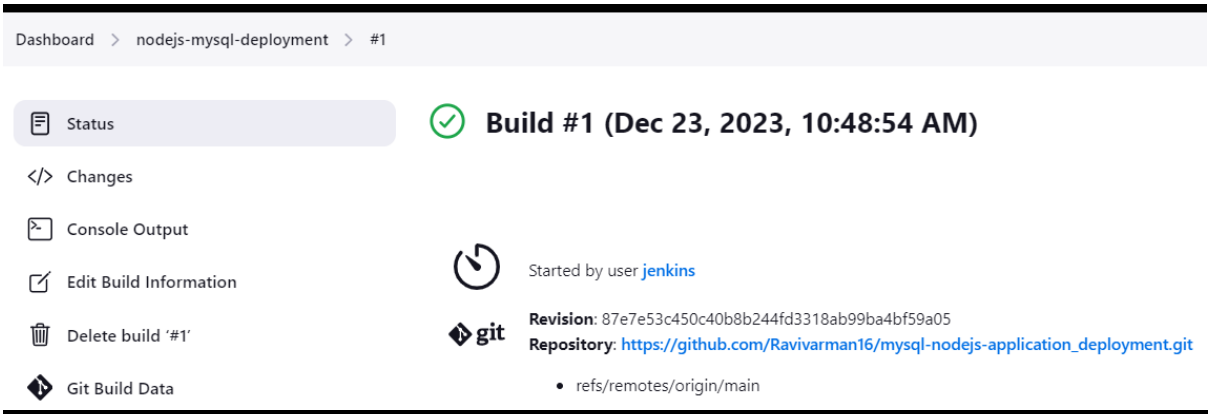
➔ The job has started successfully:

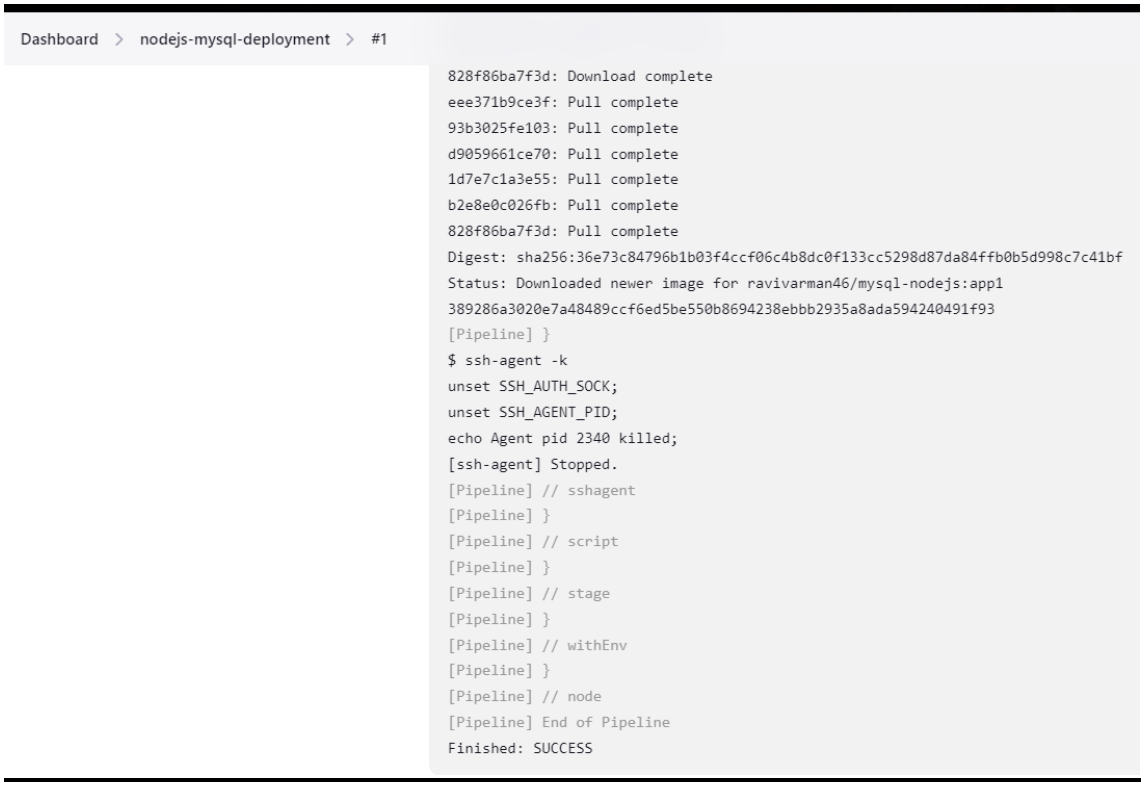


➔ The job has executed successfully: **click the build number 1:**

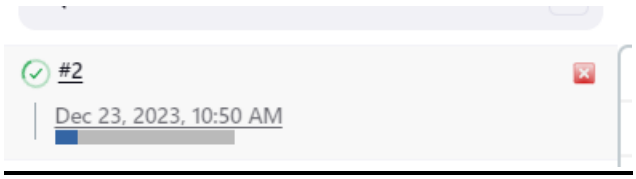


➔ Then **click console output,** to know how Jenkins executed this job:

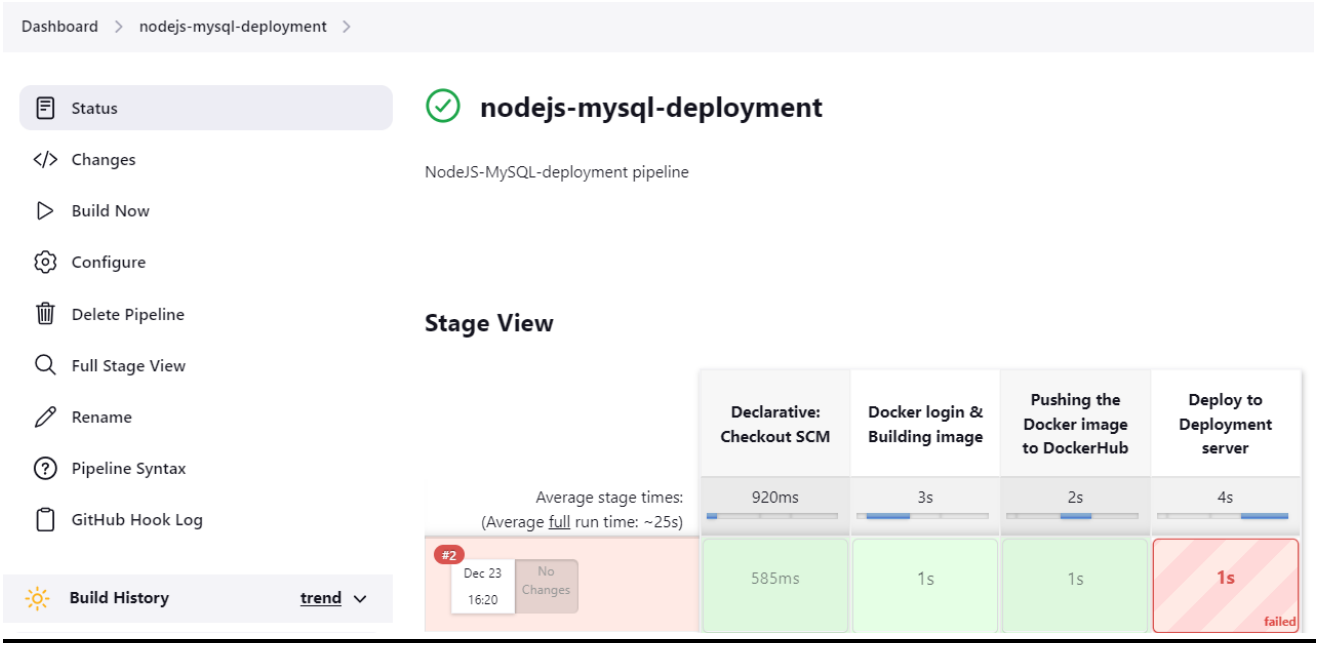




➔ Then build the job again:



➔ The job has failed, to know about the failed status, click the **console output:**



➔ The job is failed because already a container is running on port number 80. For this we need to alter our Jenkinsfile.

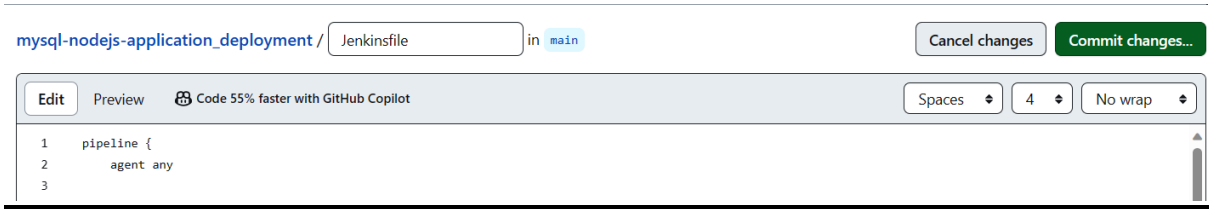
```
+ ssh -o StrictHostKeyChecking=no ubuntu@3.20.227.222 docker run -d -it --name nodejs -p 80:8080 ravivarman46/mysql-nodejs:app1
docker: Error response from daemon: Conflict. The container name "/nodejs" is already in use by container
"389286a3020e7a48489ccf6ed5be550b8694238ebbb2935a8ada594240491f93". You have to remove (or rename) that container to be able to reuse that name.
See 'docker run --help'.
[Pipeline] }
$ ssh-agent -k
unset SSH_AUTH_SOCK;
unset SSH_AGENT_PID;
echo Agent pid 2508 killed;
[ssh-agent] Stopped.
[Pipeline] // sshagent
[Pipeline] }
[Pipeline] // script
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
ERROR: script returned exit code 125
Finished: FAILURE
```

➔ Jenkinsfile altering portion: update this portion on Jenkinsfile:

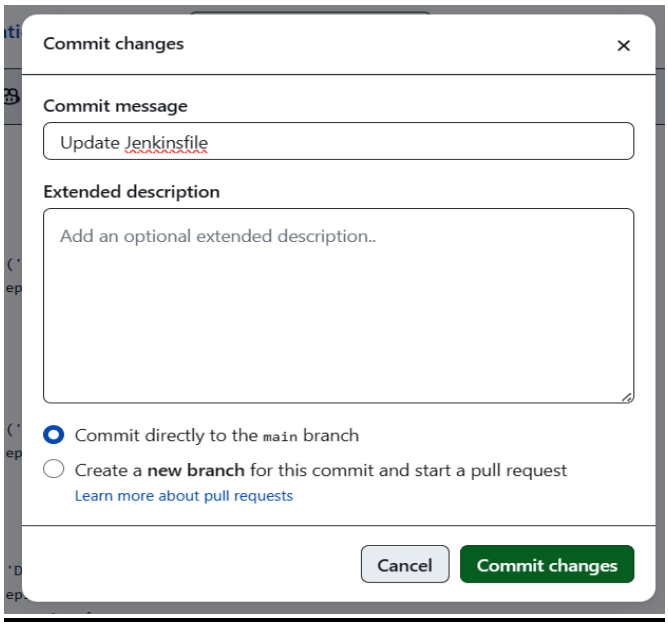
```
script {
                    sshagent(['ssh']) {
                        // Execute the command within the
sshagent block using sh step
```

```
                        sh 'ssh -o StrictHostKeyChecking=no
ubuntu@3.20.227.222 "docker stop nodejs; docker rm nodejs;
docker run -d -it --name nodejs -p 80:8080
ravivarman46/mysql-nodejs:app1"'
                    }
```

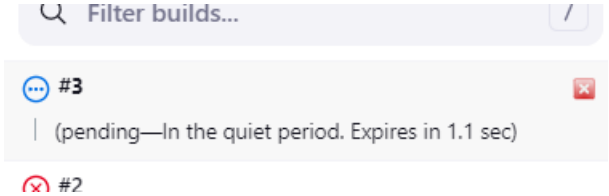
➔ Click **commit changes:**



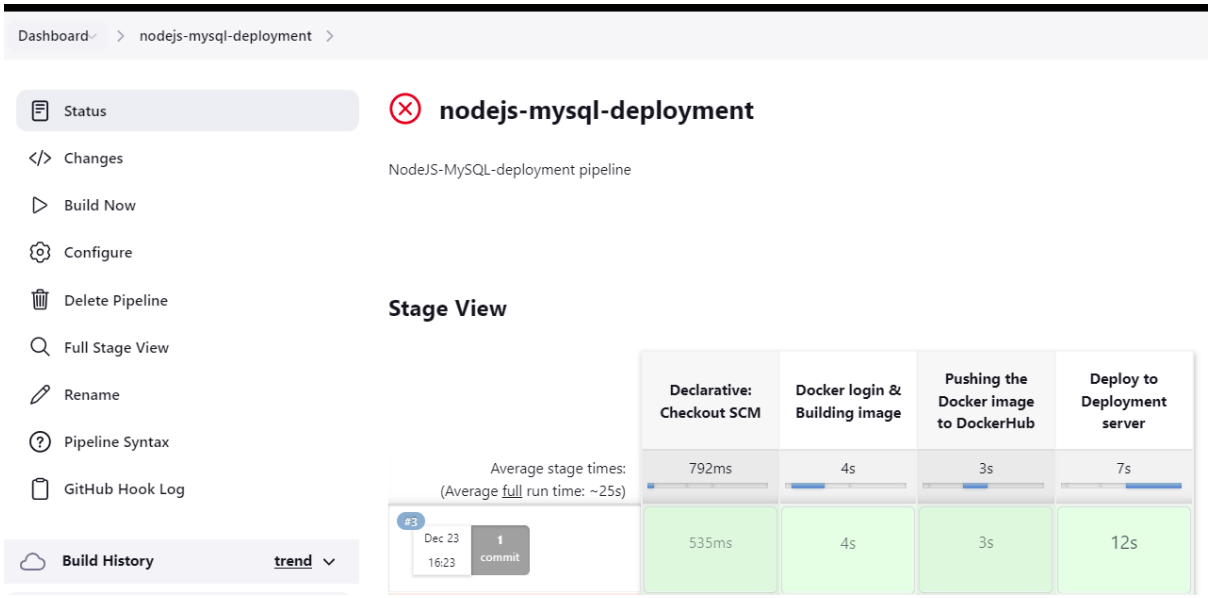
➔ Again, click commit changes:



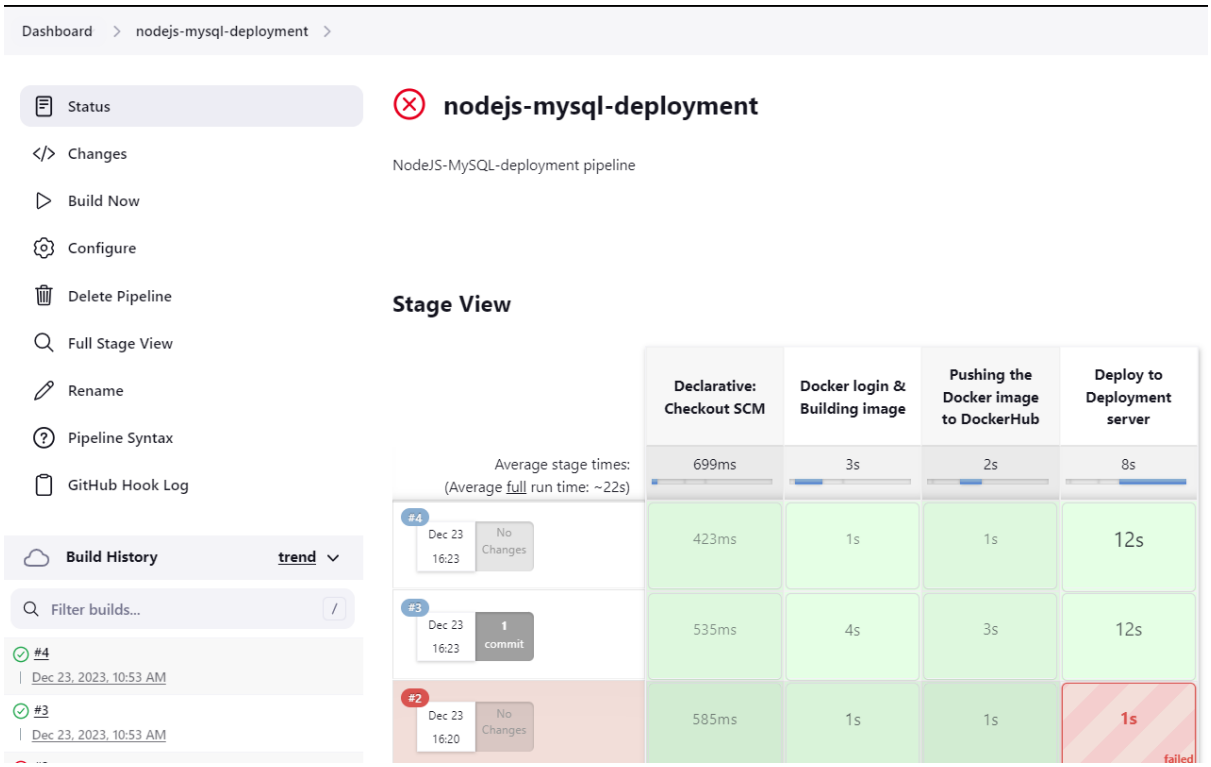
➔ The job got triggered automatically when we made a change in GitHub:



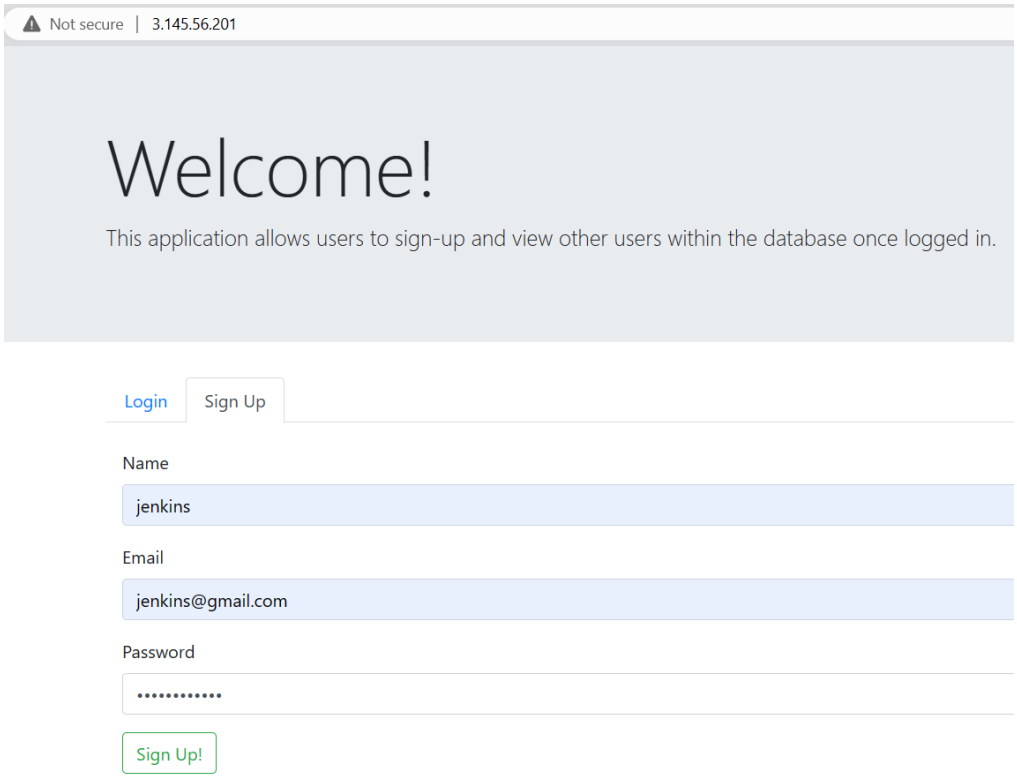
➔ The job has executed successfully:

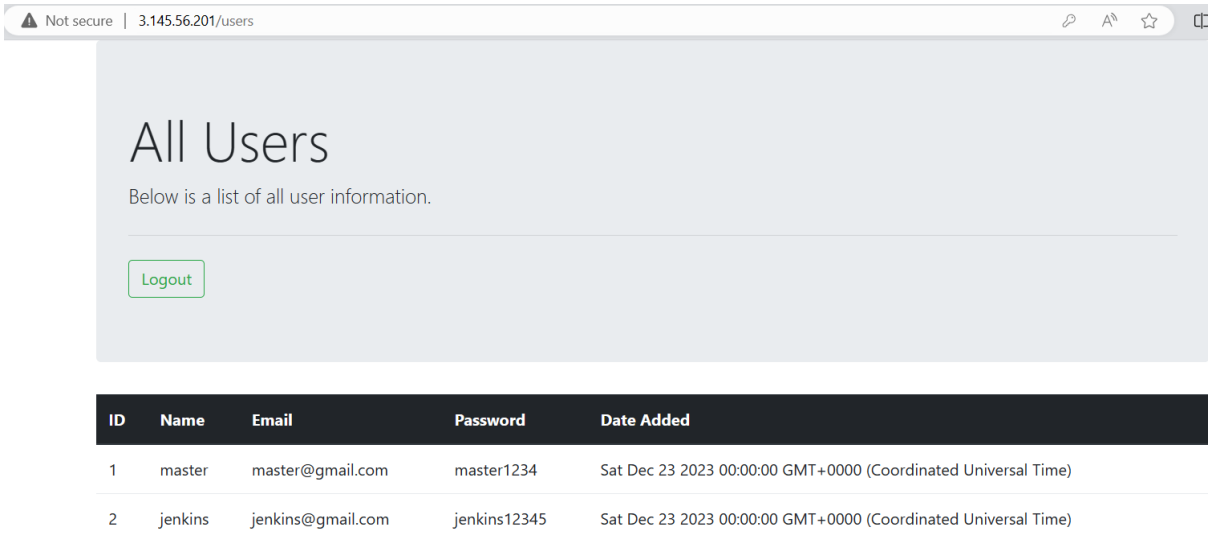


➔ Now build the job again, to know whether pipeline is working fine. As per the result pipeline is working perfectly.



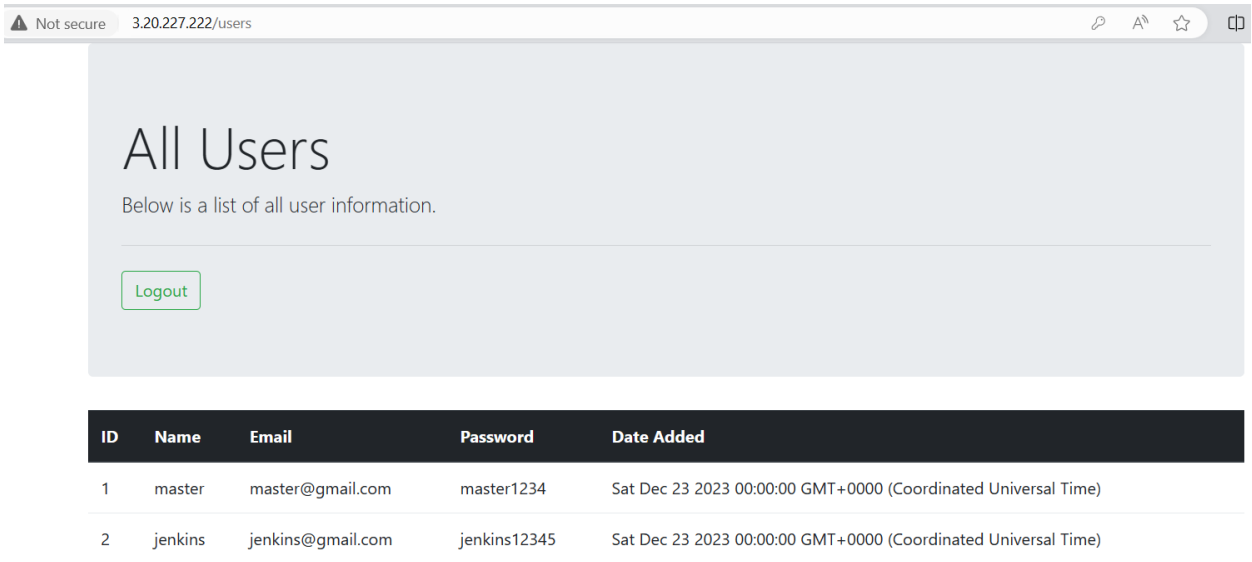
➔ Checking the container, **by adding new credentials from testing instance:**





ID	Name	Email	Password	Date Added
1	master	master@gmail.com	master1234	Sat Dec 23 2023 00:00:00 GMT+0000 (Coordinated Universal Time)
2	jenkins	jenkins@gmail.com	jenkins12345	Sat Dec 23 2023 00:00:00 GMT+0000 (Coordinated Universal Time)

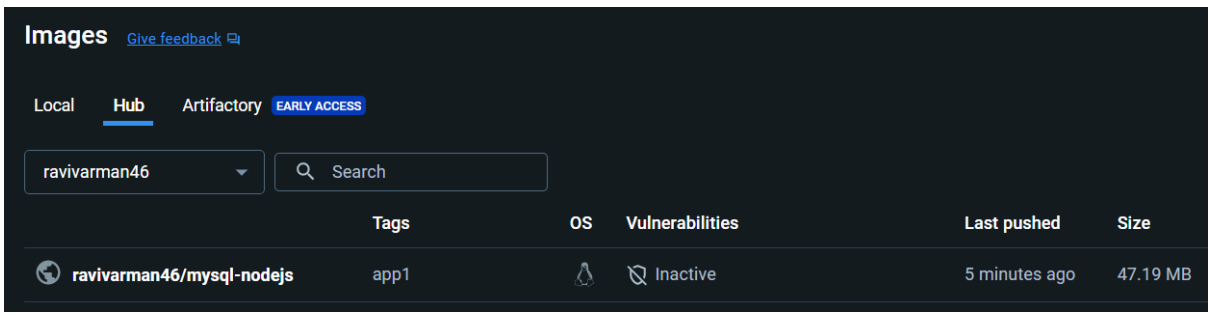
➔ Trying to **login with the new credentials on deployment instance** where we **deployed the container through Jenkins pipeline.**



➔ Checking the deployment instance:

```
root@ip-172-31-23-65:/home/ubuntu# docker images
REPOSITORY                   TAG       IMAGE ID       CREATED          SIZE
ravivarman46/mysql-nodejs    app1      67b9173a1b7e   10 minutes ago   132MB
root@ip-172-31-23-65:/home/ubuntu# docker ps
CONTAINER ID   IMAGE                            COMMAND                  CREATED         STATUS         PORTS
          NAMES
2b30f398d7eb   ravivarman46/mysql-nodejs:app1   "docker-entrypoint.s…"   5 minutes ago   Up 5 minutes   0.0.0.0:80->8080/tcp, :::80->8
080/tcp   nodejs
root@ip-172-31-23-65:/home/ubuntu#
```

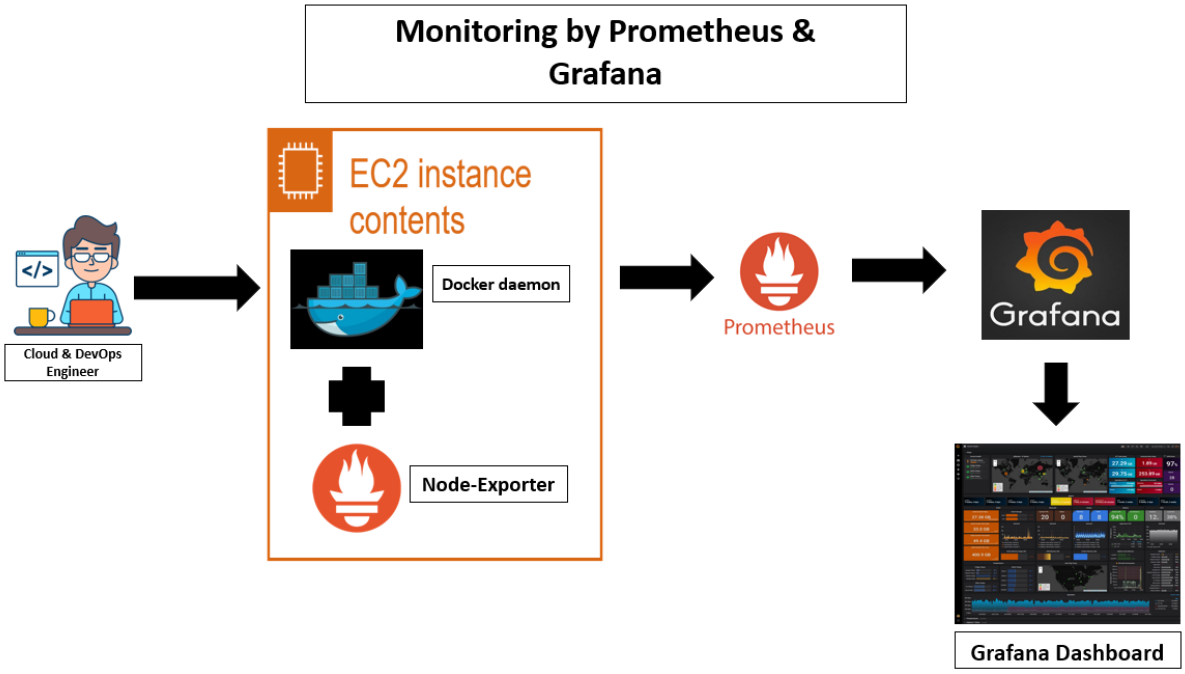
➔ Checking the **DockerHub:**



****

STEP:11 - SETTING UP THE MONITORING THROUGH PROMETHEUS & GRAFANA

OVERVIEW – DIAGRAM



➔ On the deployment instance, we need to create **daemon.json** under **/etc/docker** for **getting metrics from docker daemon.**

➔ After that **restart the docker service.**

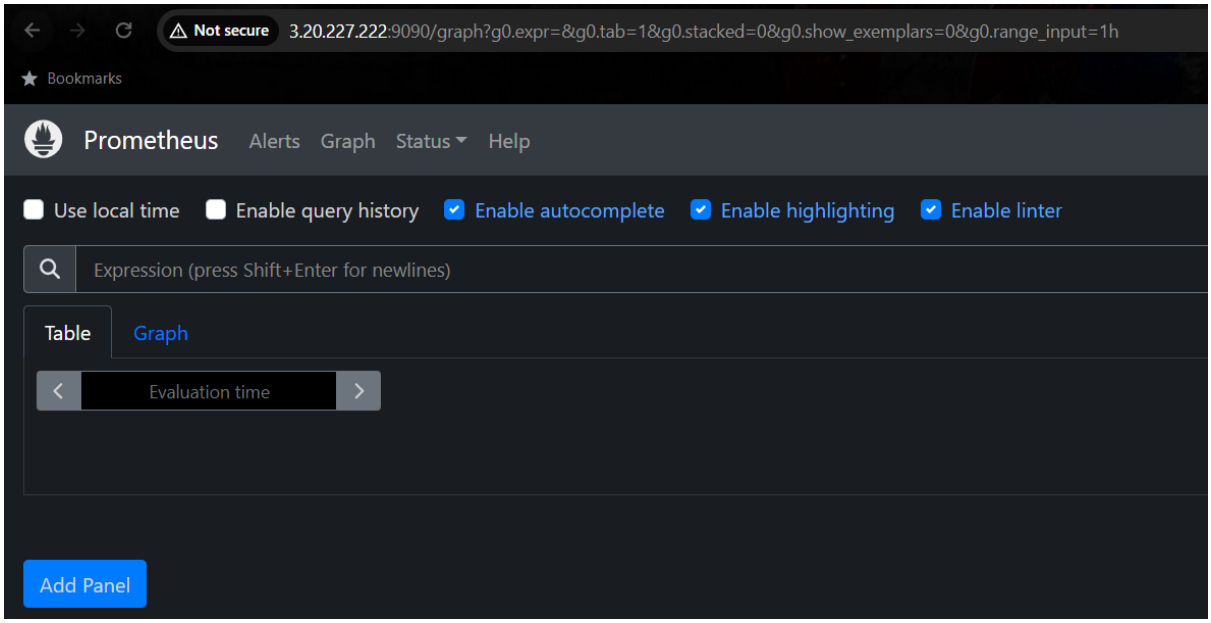
```
root@ip-172-31-23-65:/home/ubuntu# cd /etc/docker/
root@ip-172-31-23-65:/etc/docker# ls
root@ip-172-31-23-65:/etc/docker# nano daemon.json
root@ip-172-31-23-65:/etc/docker# cat daemon.json
{
  "metrics-addr" : "0.0.0.0:9323",
  "experimental" : true
}
root@ip-172-31-23-65:/etc/docker# service docker restart
```

➔ Then we need to add this one to Prometheus service by altering **prometheus.yaml** file which will be under this location **/etc/prometheus/prometheus.yaml**

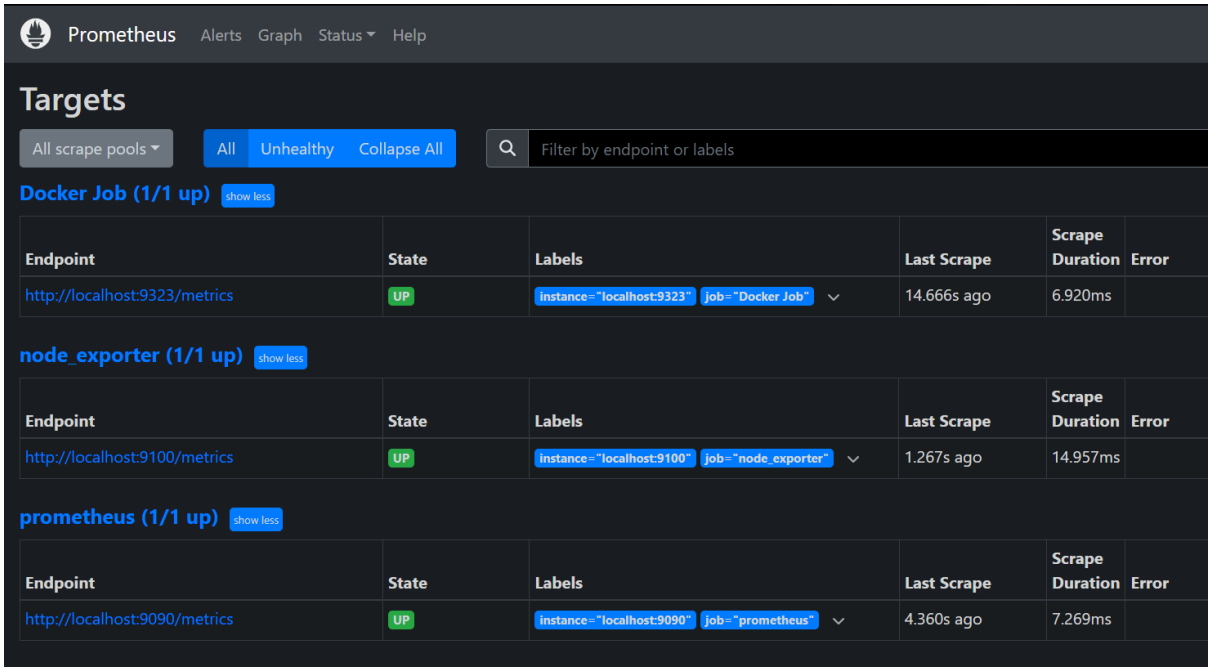
```
scrape_configs:
  # The job name is added as a label `job=
  - job_name: "prometheus"

    # metrics_path defaults to '/metrics'
    # scheme defaults to 'http'.
    static_configs:
      - targets: ["localhost:9090"]

  - job_name: 'node_exporter'
    scrape_interval: 5s
    static_configs:
      - targets: ['localhost:9100']

  - job_name: "Docker-container"
    static_configs:
      - targets: ["localhost:9323"]
```

➔ Then restart **Prometheus service** by using the command: **service prometheus restart**

➔ Then enter the public Ip address of deployment instance on browser along with the port number **9090.**

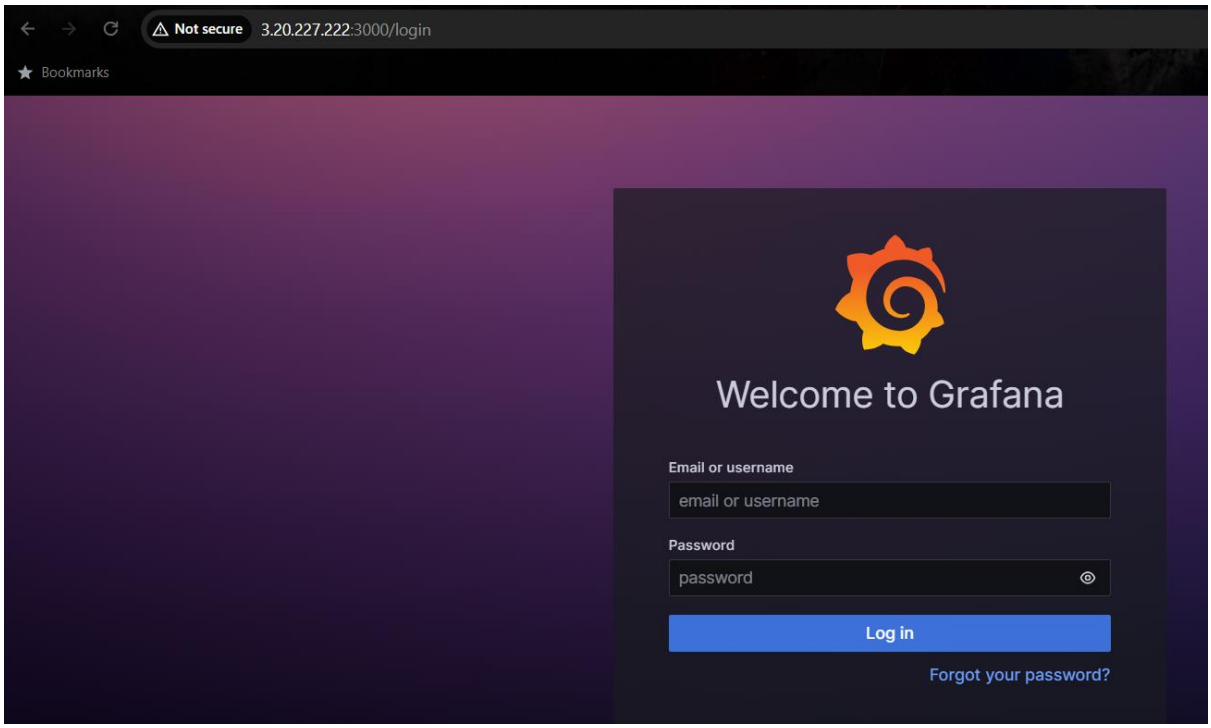


➔ Then **click the status**, under that **select targets:**



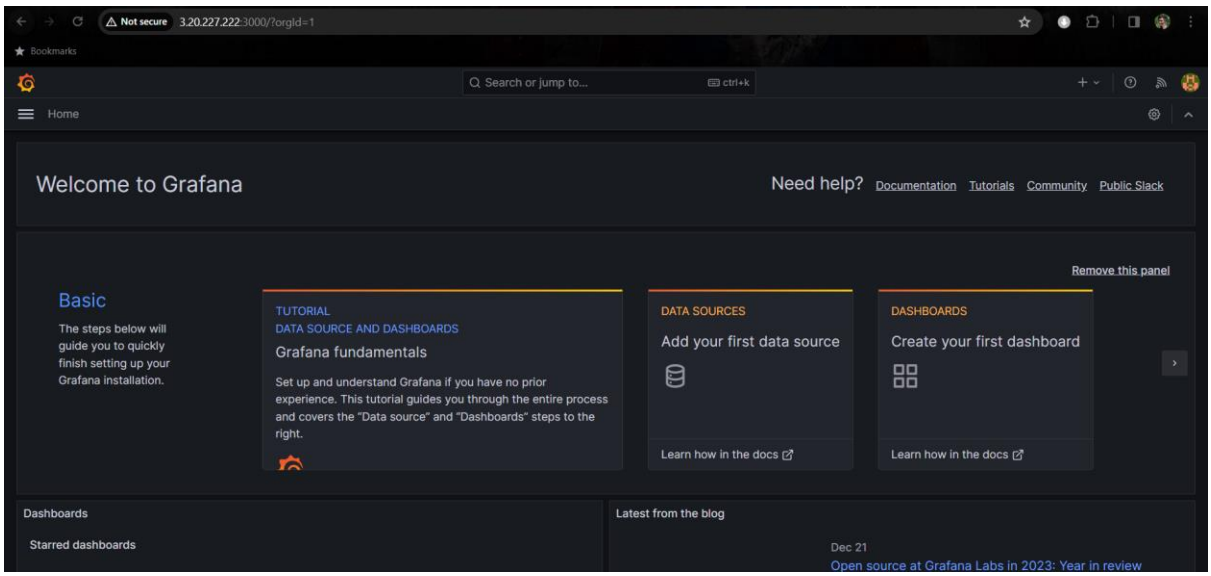
We could able to see all the targets is up and running successfully.

➔ Then enter the public Ip address of deployment instance along with the port number **3000**, **for login into Grafana.**

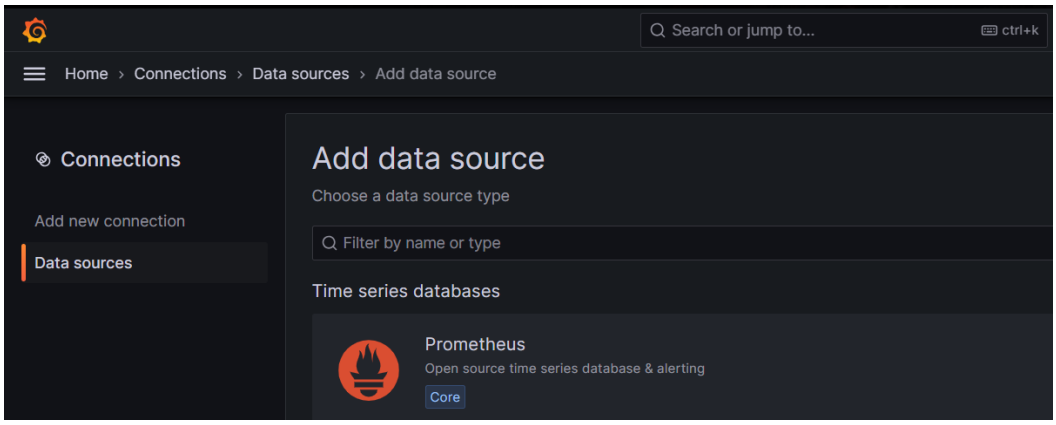


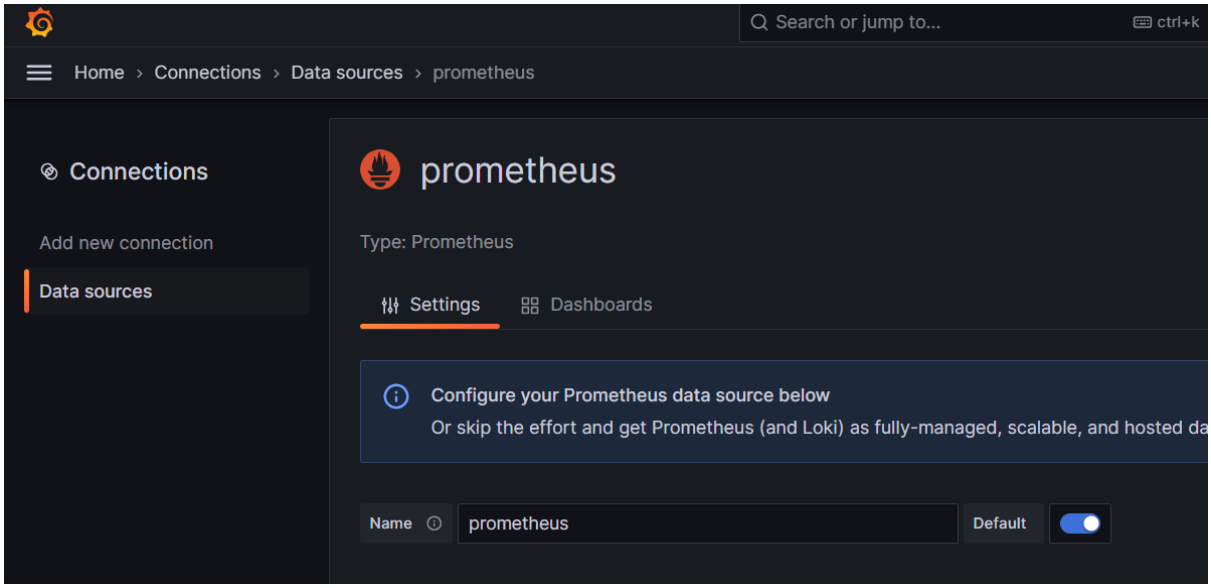
➔ For login into Grafana, **the username & password is admin. Click log in**

➔ Now we need to add **Prometheus to Grafana server,** for that click **data sources:**

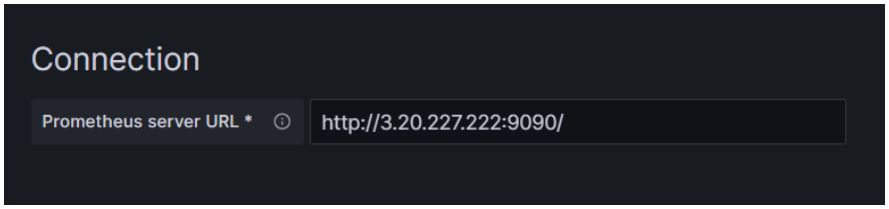


➔ Then select **Prometheus:**

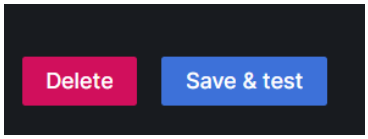




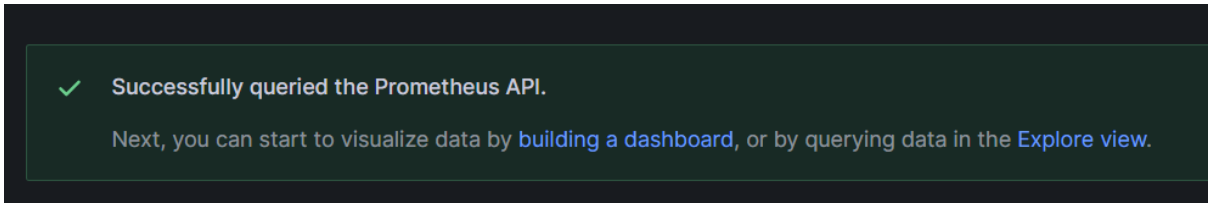
➔ Then enter the **Prometheus URL:**



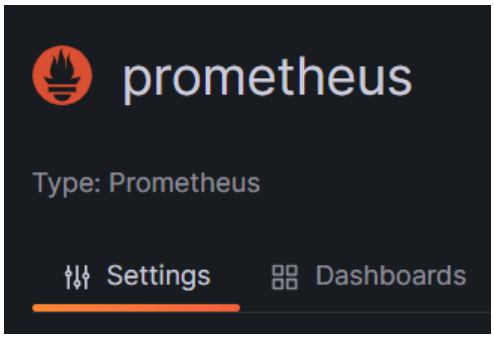
➔ After that **click save & test,** to know whether Grafana can able to connect Prometheus and get the logs from it.



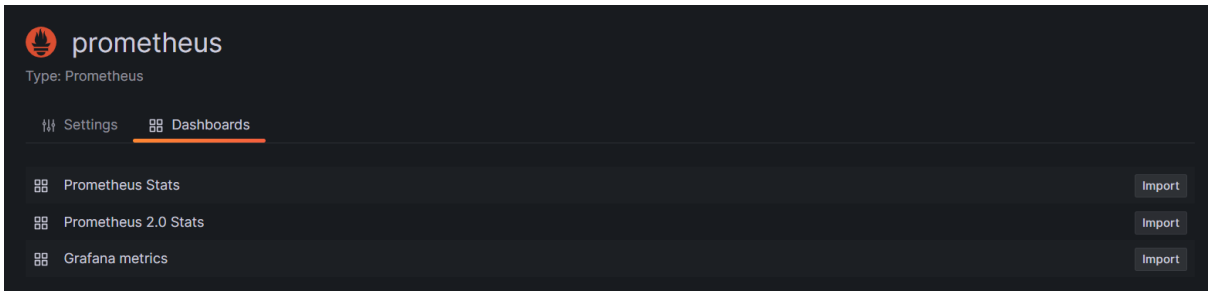
➔ We could able to see **Grafana can able to connect Prometheus successfully.**

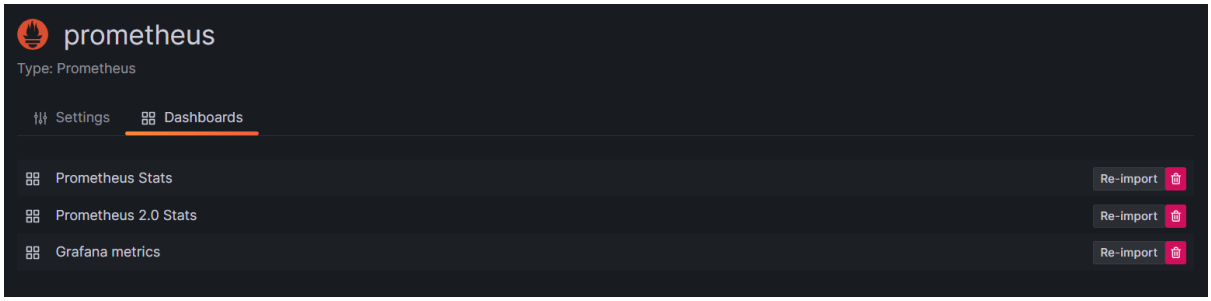


➔ Then at the top of screen we could able to see **dashboards,** click that one:

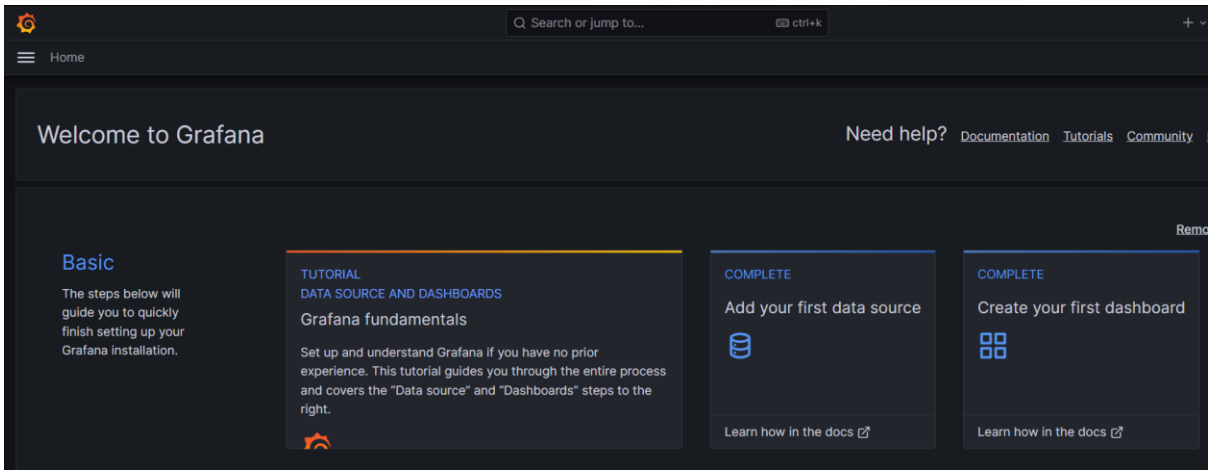


➔ Then to get the **logs from Prometheus**, click **import option for all three:**

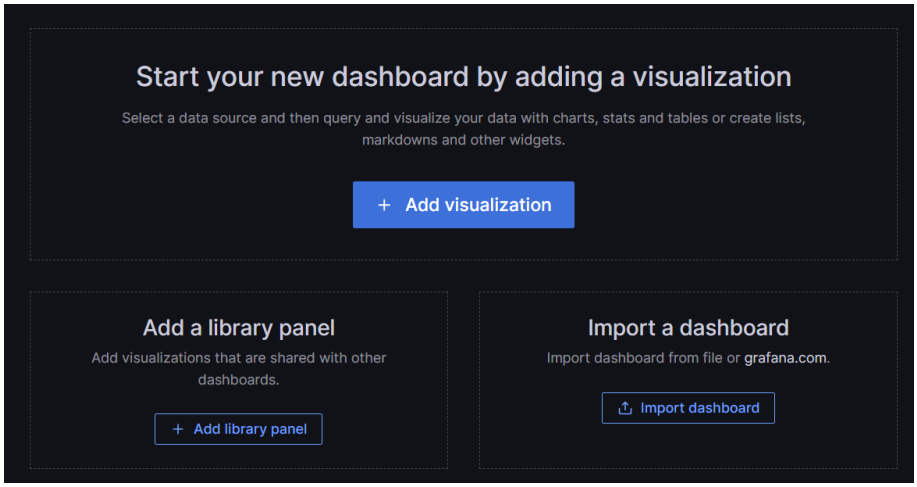




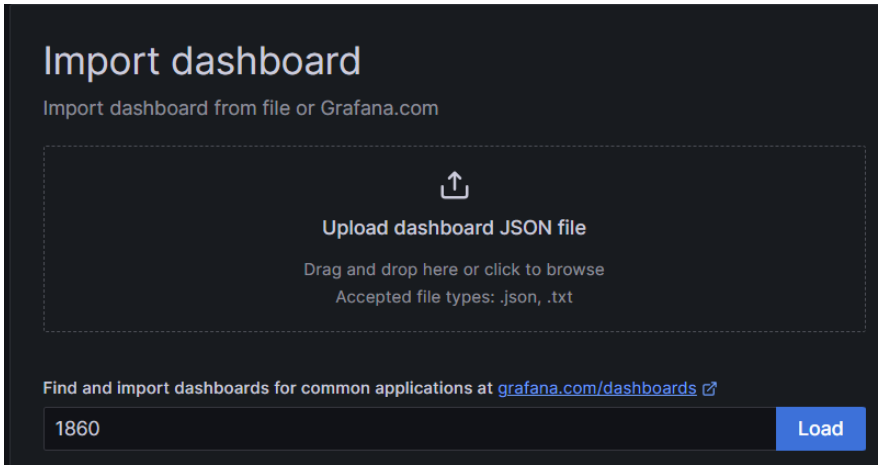
➔ Then for creating dashboards, on home screen **click dashboard:**



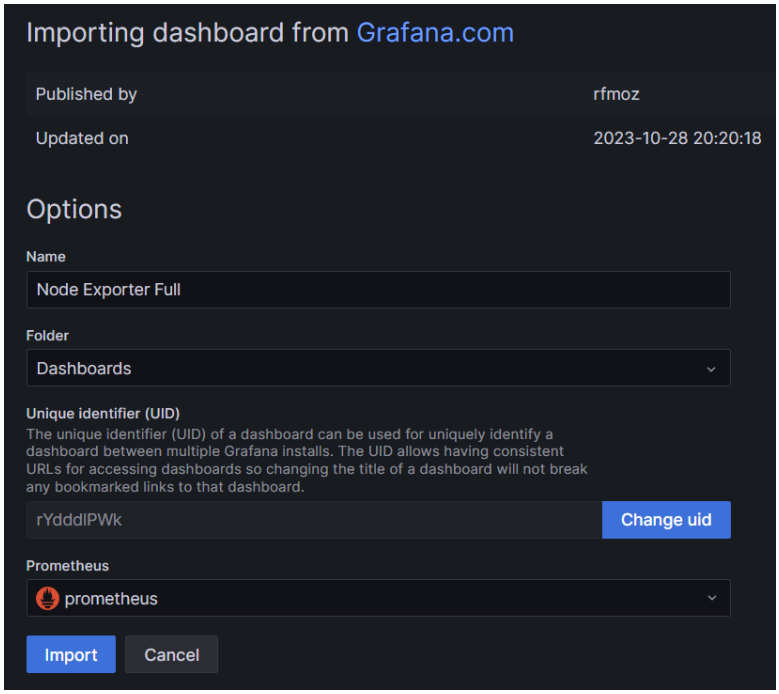
➔ Then select **import a dashboard option:**



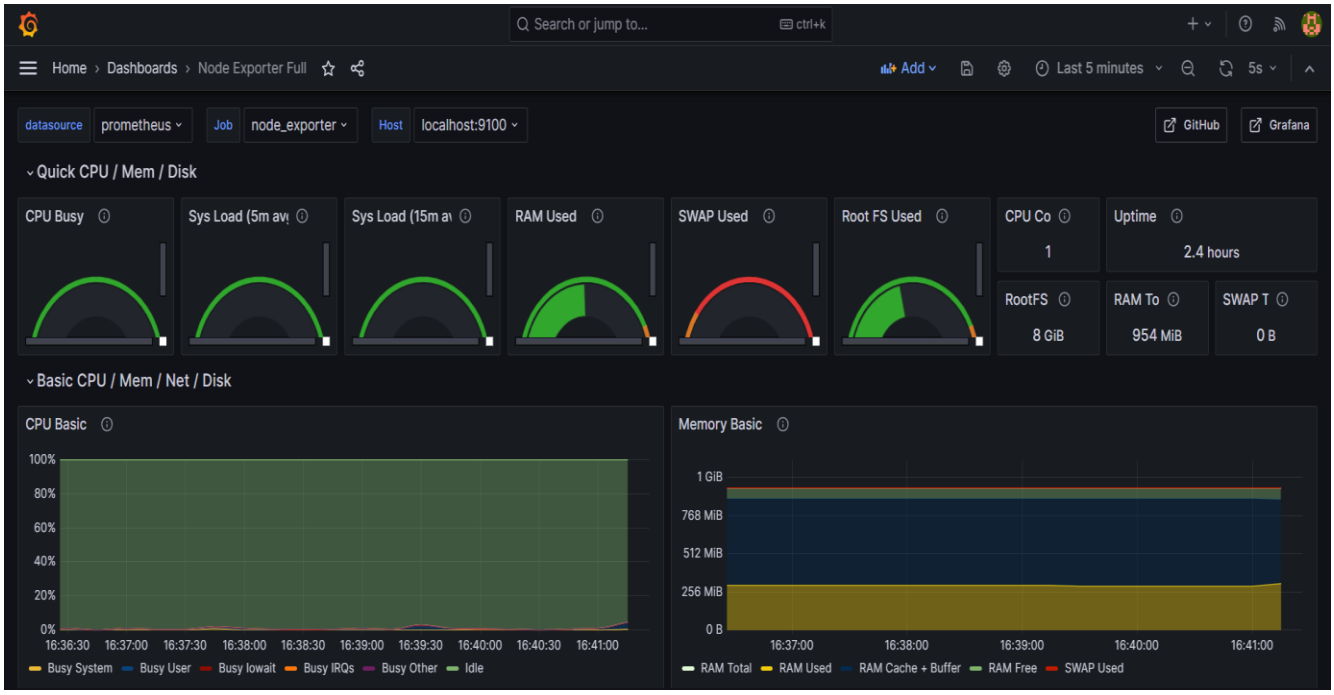
➔ Then enter **1860** which is **node exporter dashboard number**, click load option:



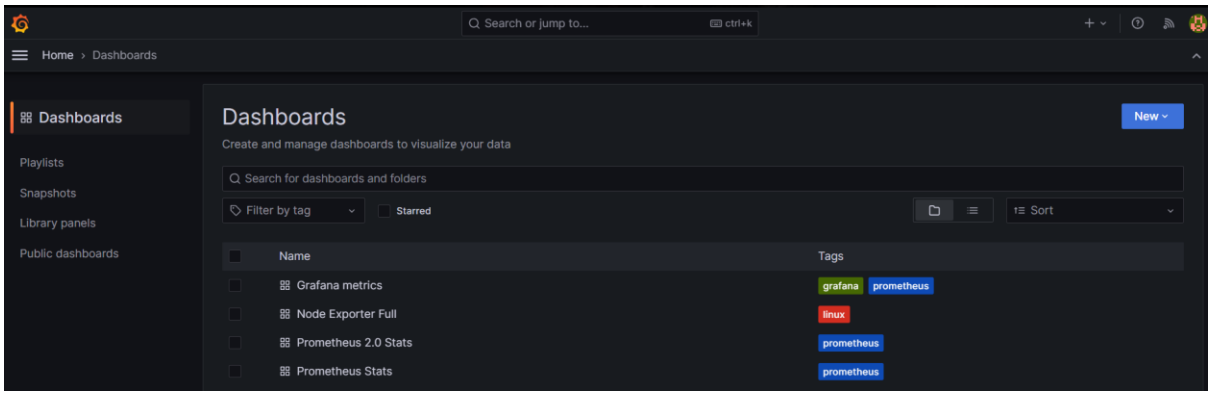
➔ Then under data source, select **Prometheus** click the **import** option:



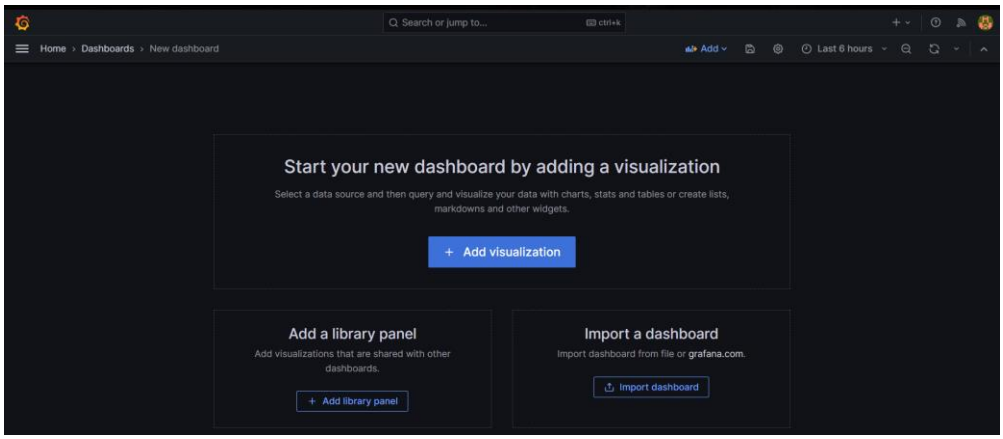
➔ The dashboard has been created successfully for **deployment instance,** where the container is running.



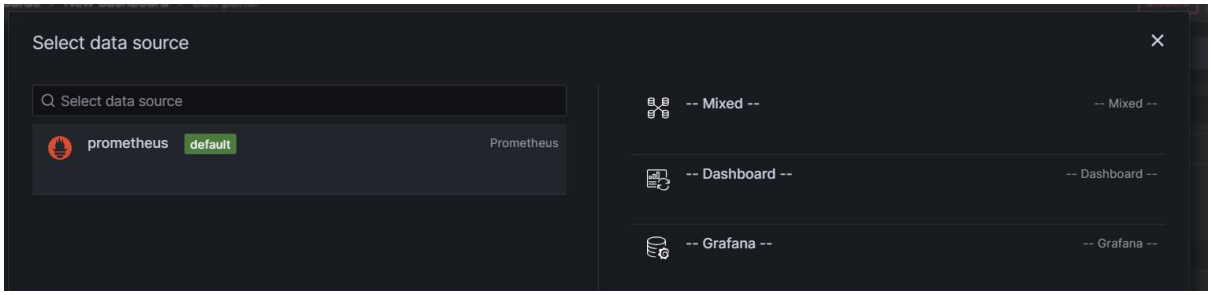
➔ To create our own dashboard, click **new** option on **right side of dashboard screen:**



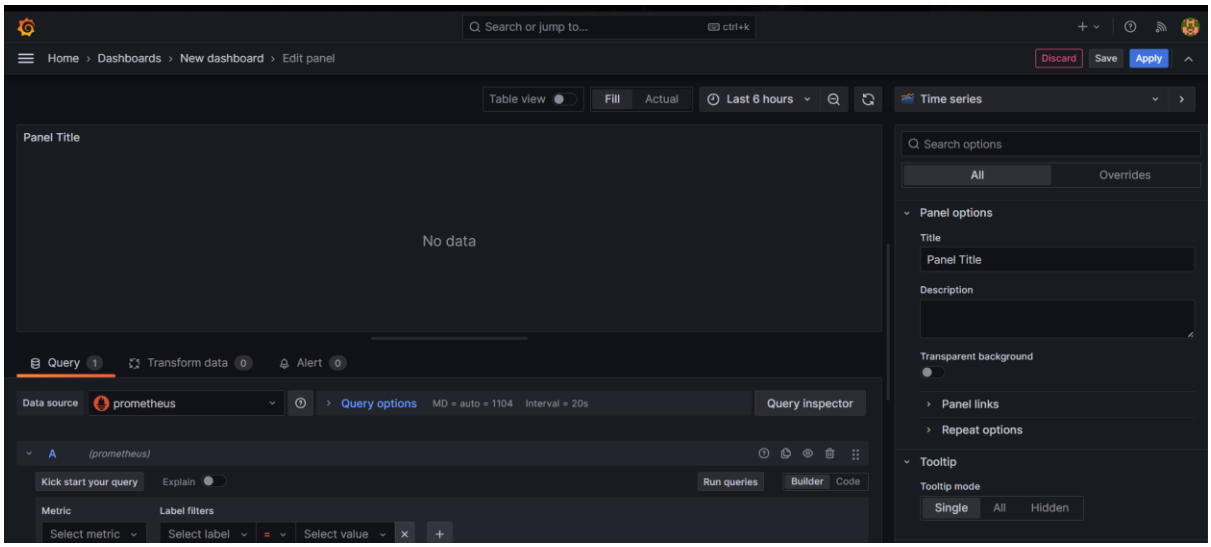
➔ Then click **add visualization:**



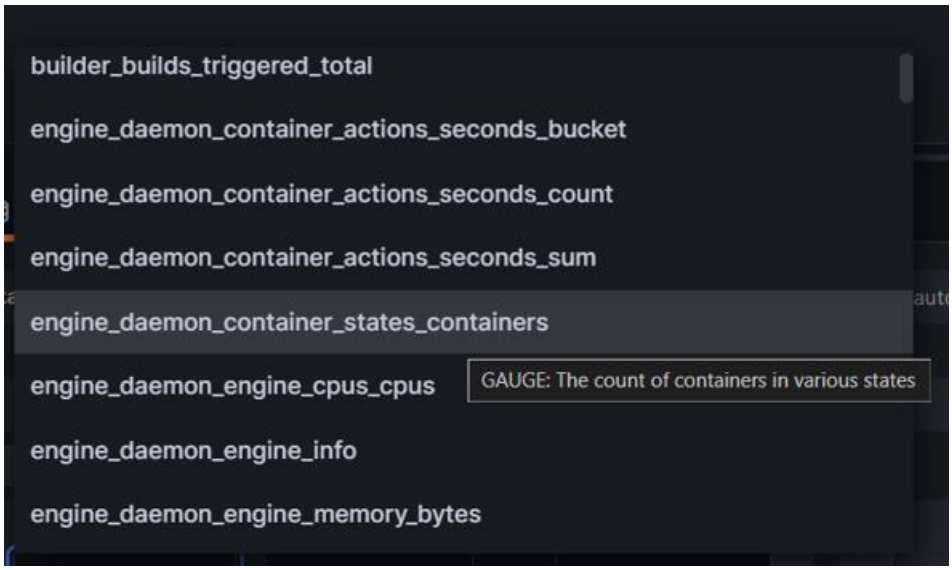
➔ Then select Prometheus as data source:



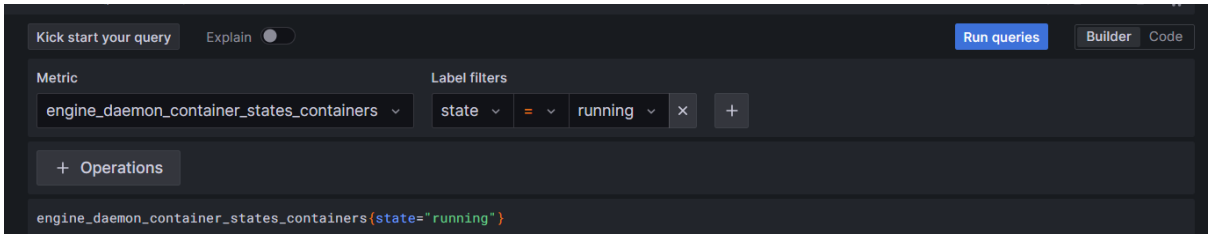
➔ Then in the dashboard, we need to make a few changes, for creating our own dashboard, **for that click metrics:**



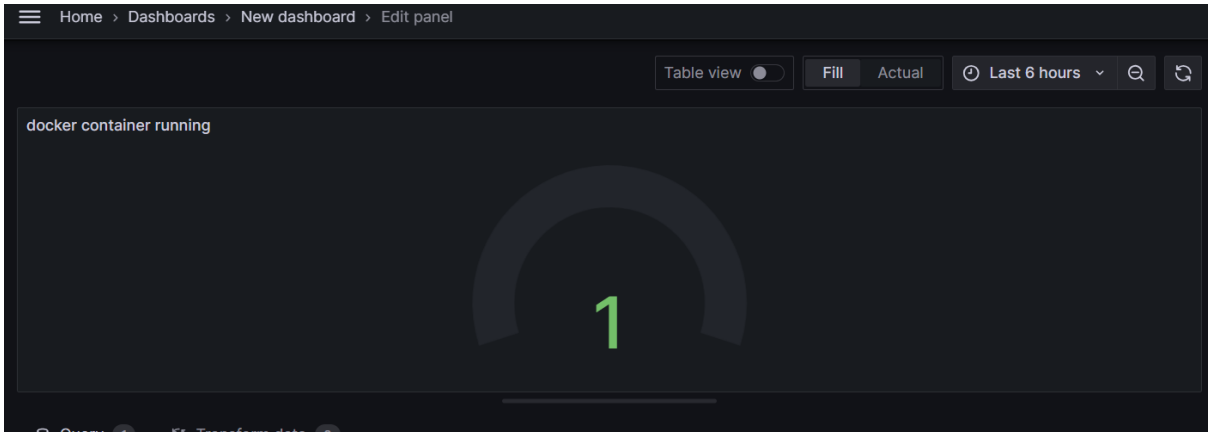
➔ Select **engine_daemon_container_states_containers option:**



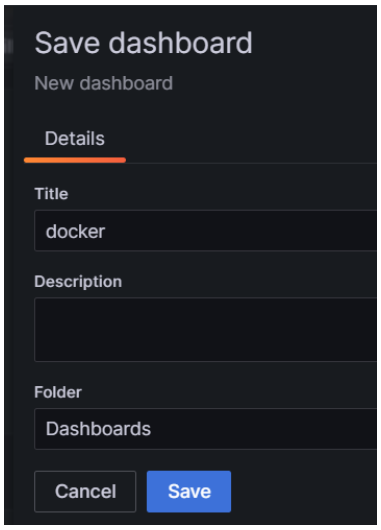
➔ Then select the **state running,** click **run queries**



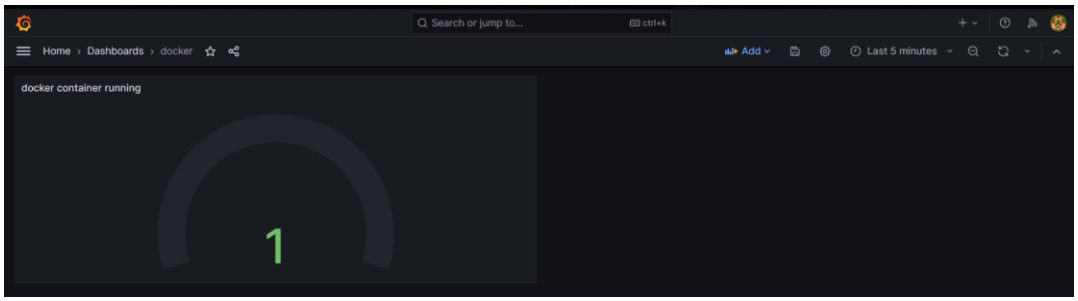
➔ We could able to see in the dashboard that **1 container is running**



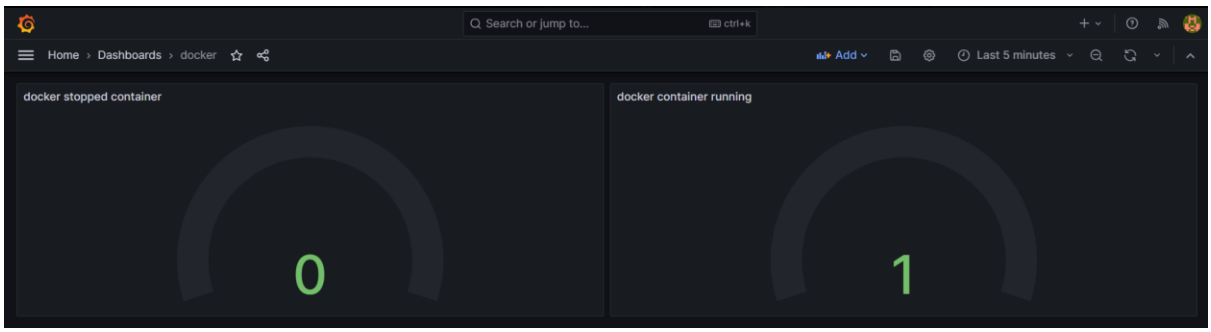
➔ Then save this dashboard:



➔ Like this we can add more features to this dashboard by **clicking add option:**



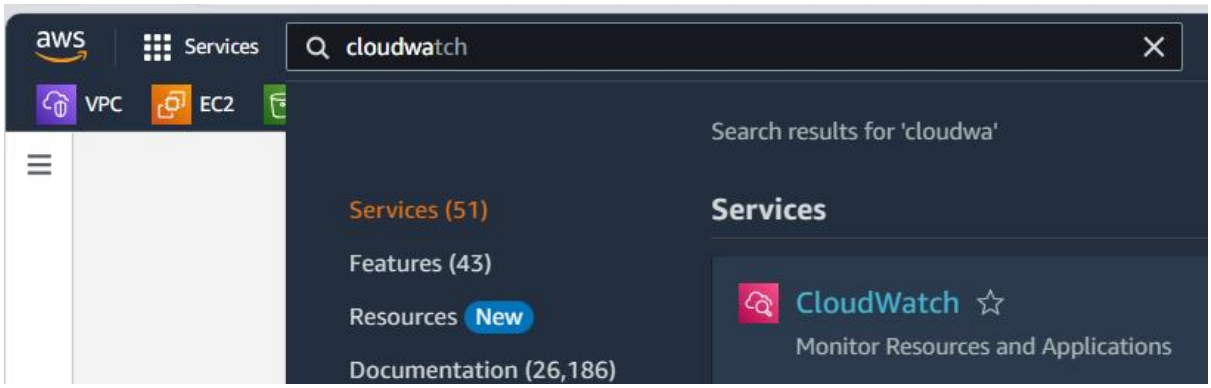
➔ Here we had setup two features – **container running & stopped container:**



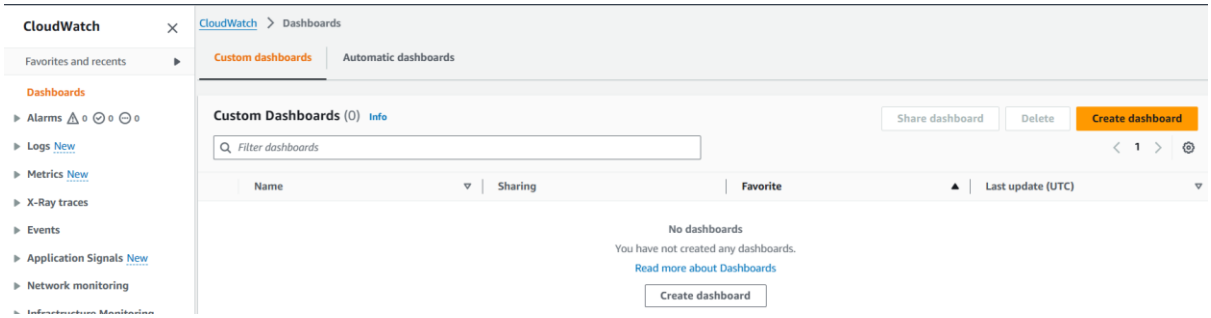
****

STEP:12 - SETTING UP THE MONITORING & ALTERING THROUGH CLOUDWATCH (OPTIONAL)

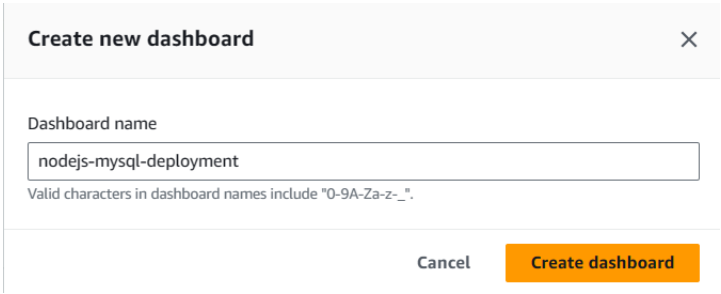
➔ On service panel search **CloudWatch,** click that one:



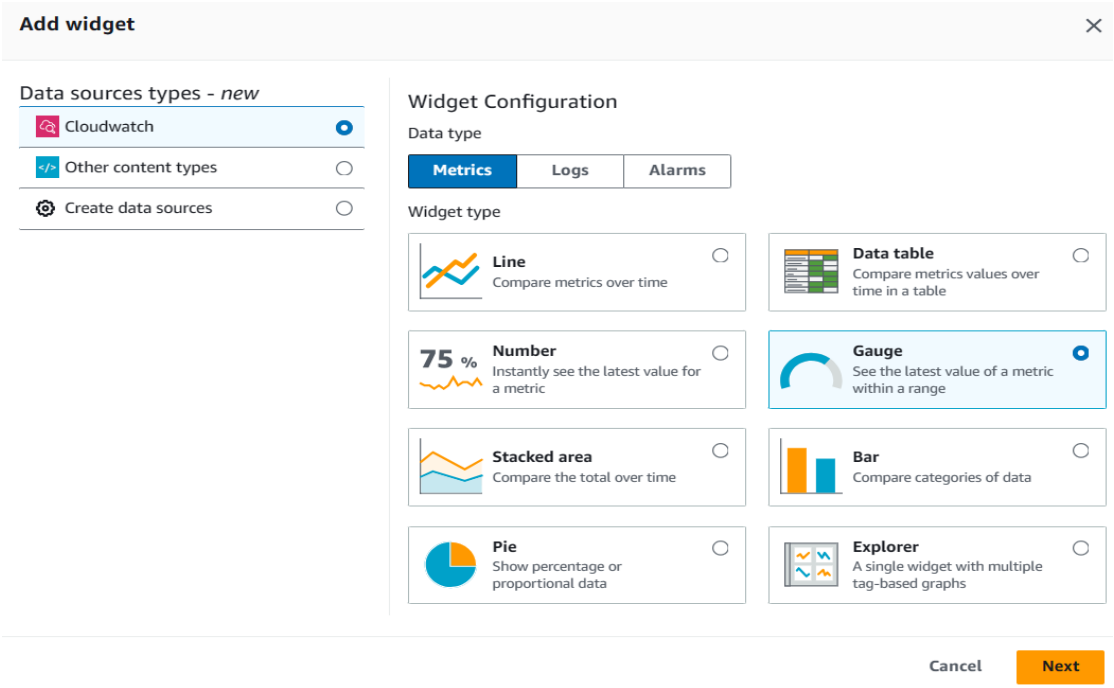
➔ Then on left side, we can able to see **dashboards**, select that one.
➔ Then select **create dashboard:**



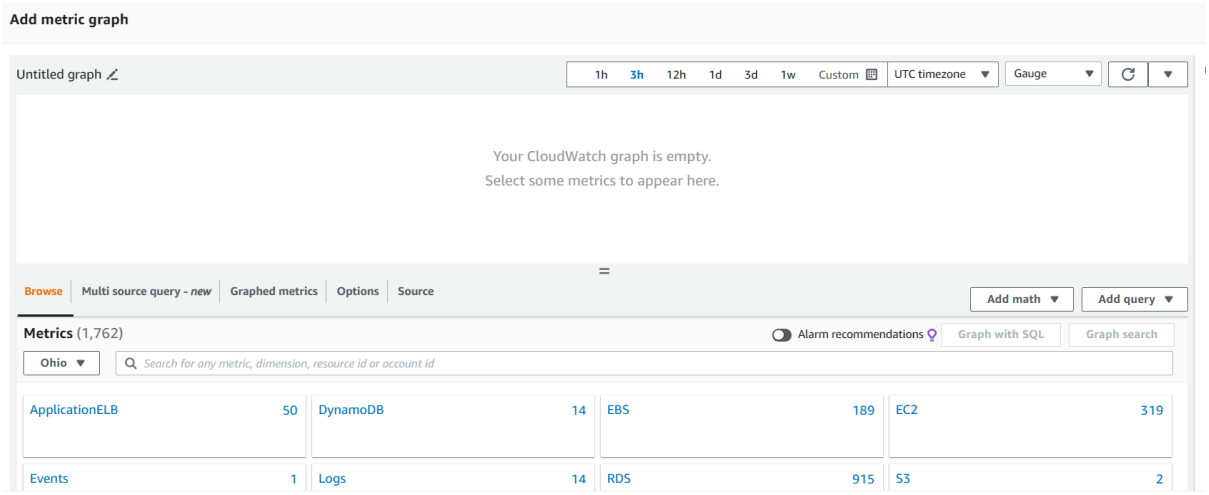
➔ Then **name dashboard, click create dashboard:**



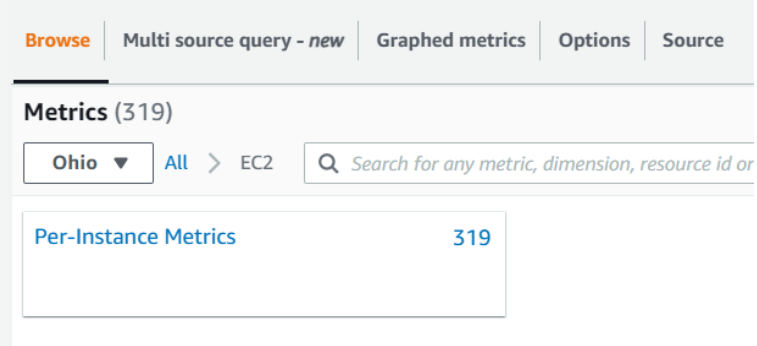
➔ Then select the widget type, according to your preferences:



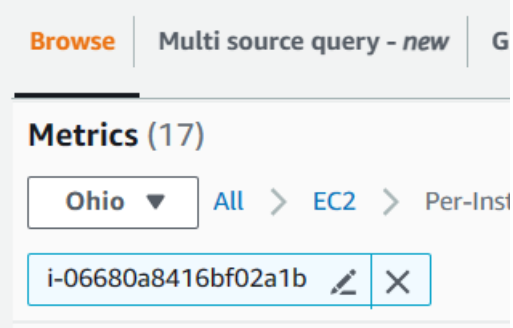
➔ Then select **EC2:**



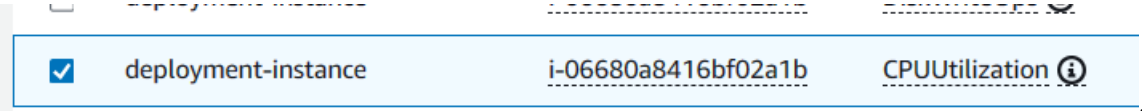
➔ Then select **per-instance metrics:**



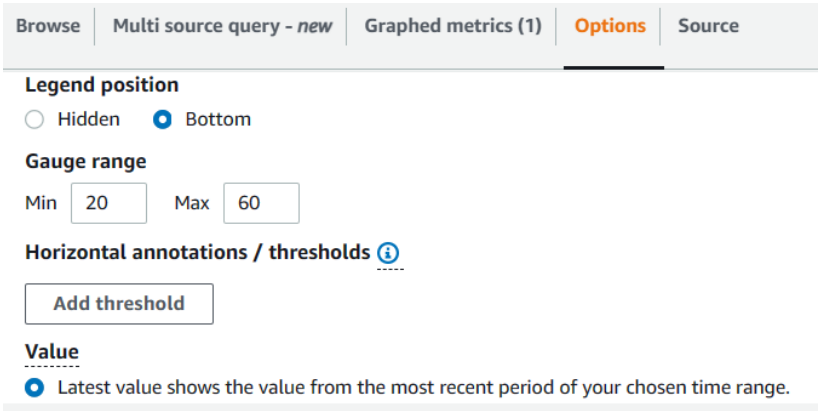
➔ Then **copy the instance id of deployment instance,** and paste it:



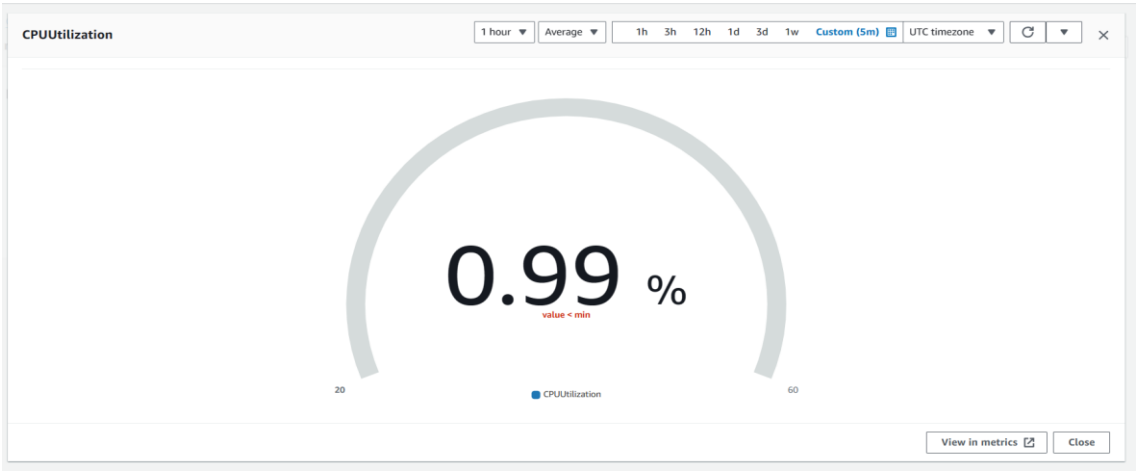
➔ Then select the metrics type according to your preferences, but here I am selecting **CPUUtilization:** click next:

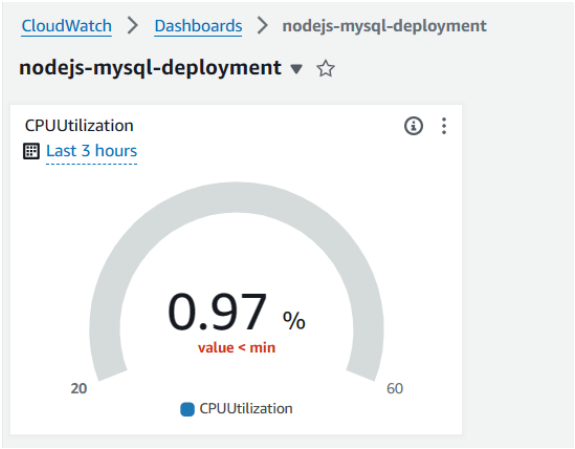


➔ Then **select range** according to your preferences, click create dashboard:

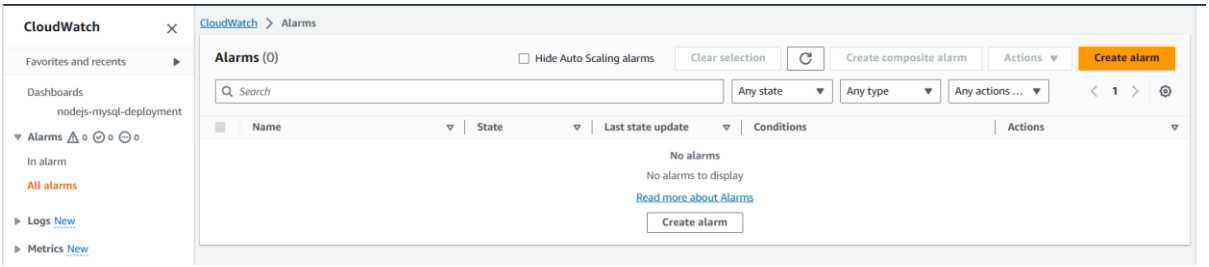


➔ The dashboard has been setup-ed successfully:

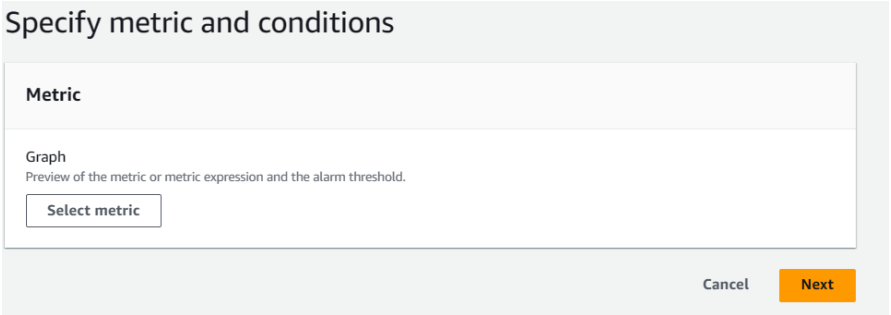




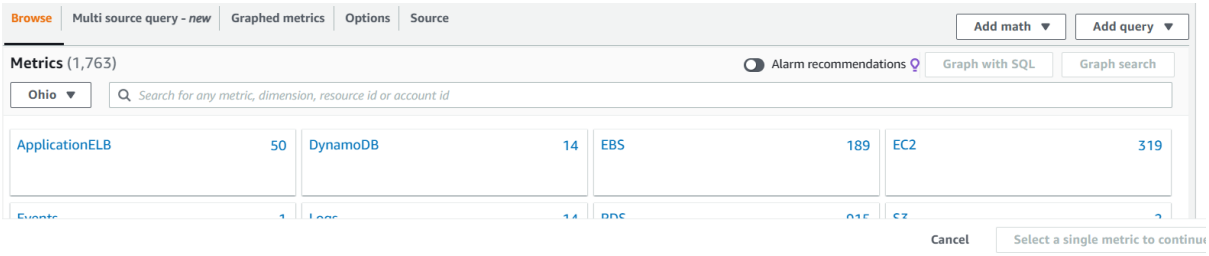
➔ Then for **alerting**, we need to **create CloudWatch alarm,** for that select **In alarm -> create alarm:**



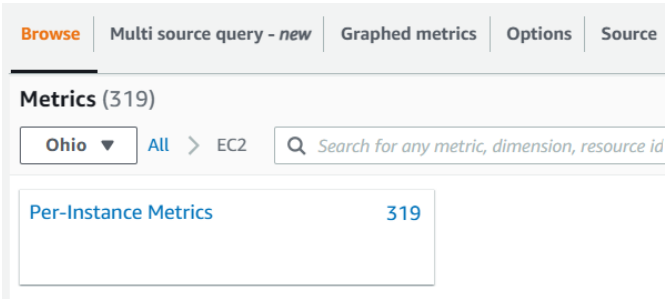
➔ Then click **select metrics:**



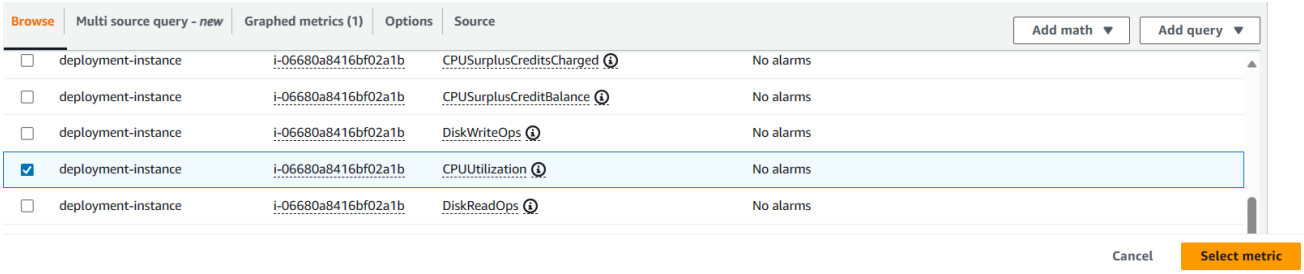
➔ Then click **EC2:**



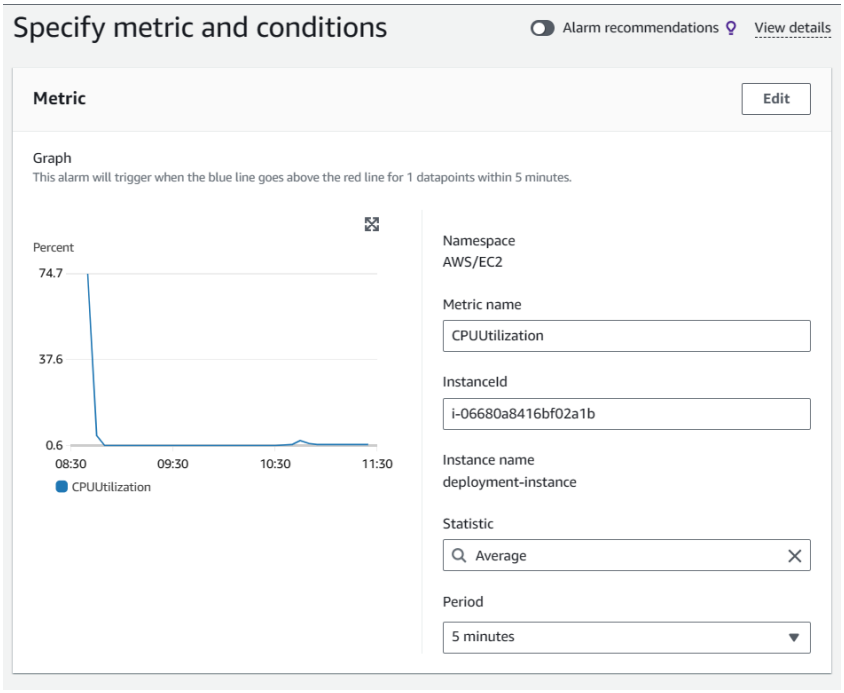
➔ Then click **per-instance metrics:**



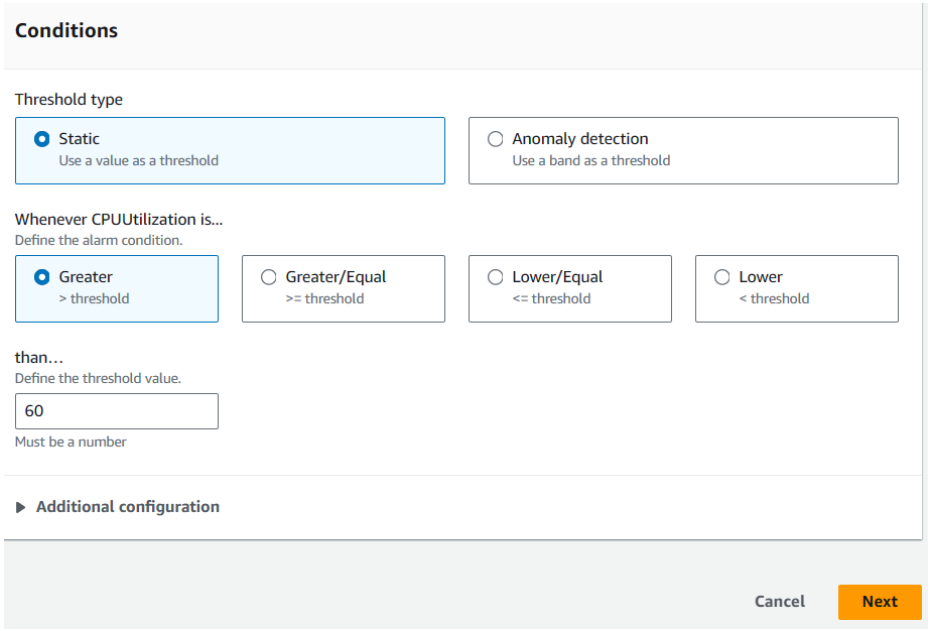
➔ Then select the metrics type for deployment instance: **click select metric:**



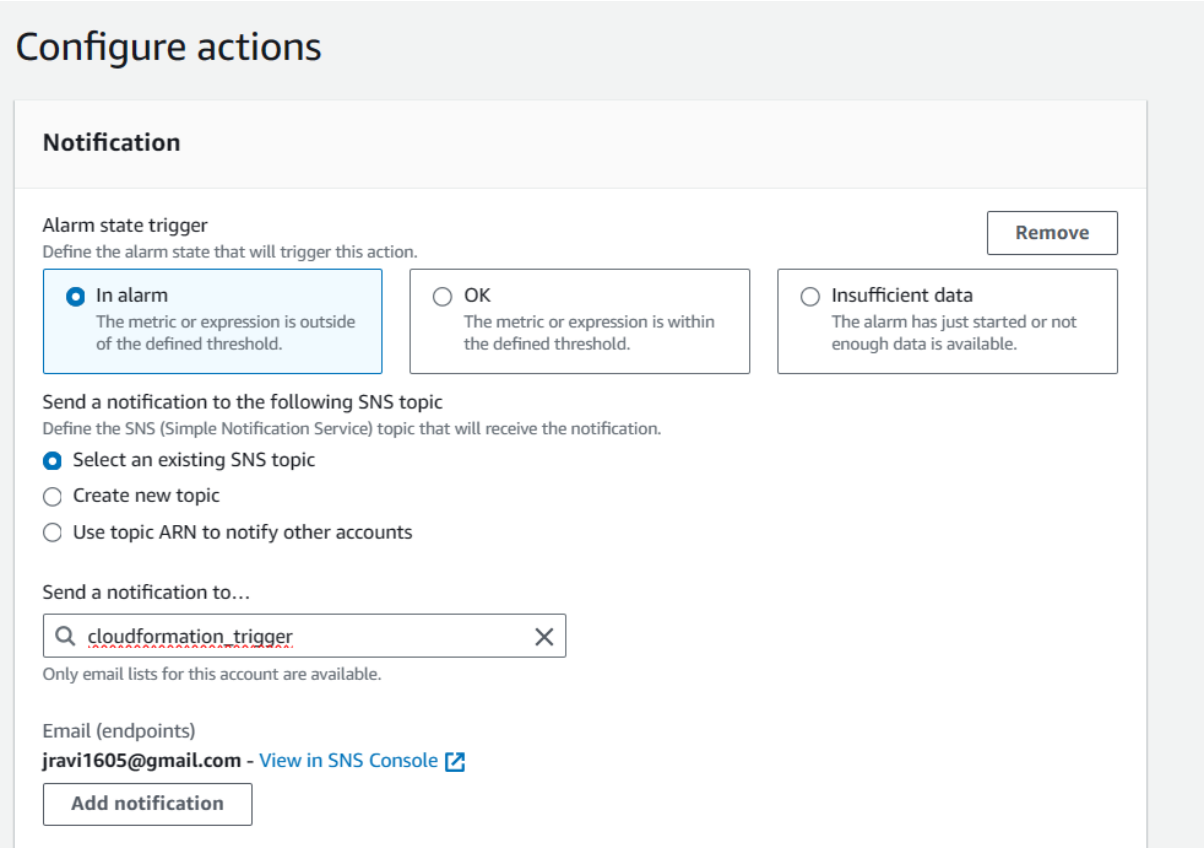
➔ The metrics has been selected successfully for the alarm:



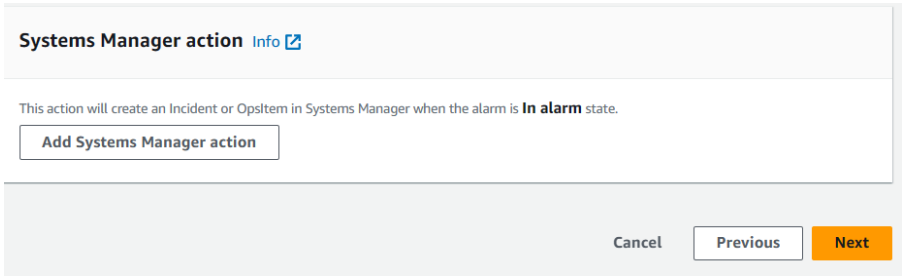
➔ Then select the **threshold as static,** and enter the threshold value according to your preferences: then click next



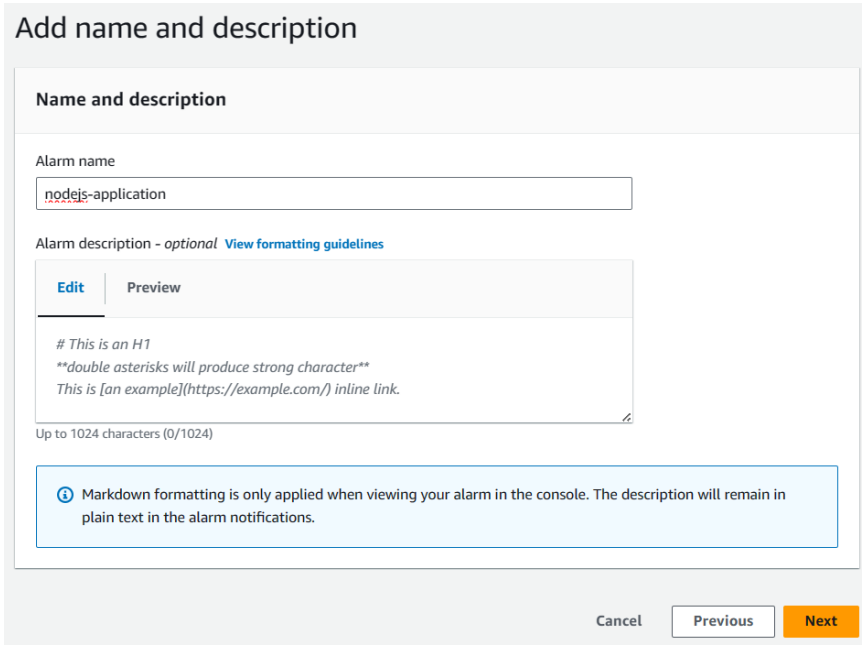
➔ Then for the **notification: select an existing SNS Topic,** or else **click create new topic** for sending notification through email.



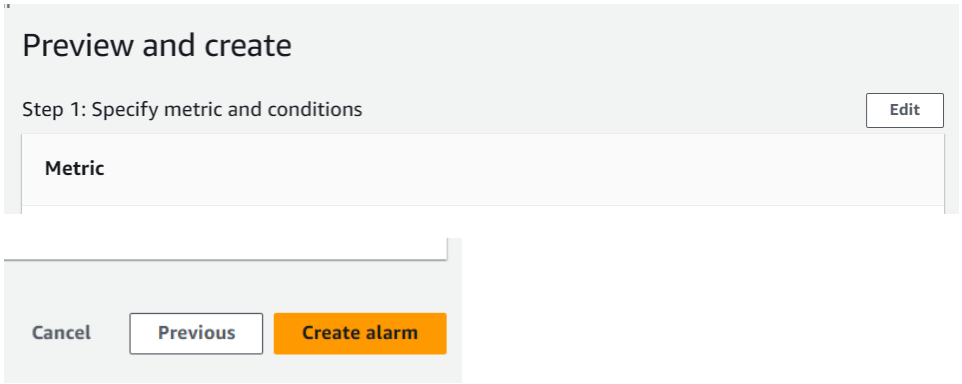
➔ Then click **next:**



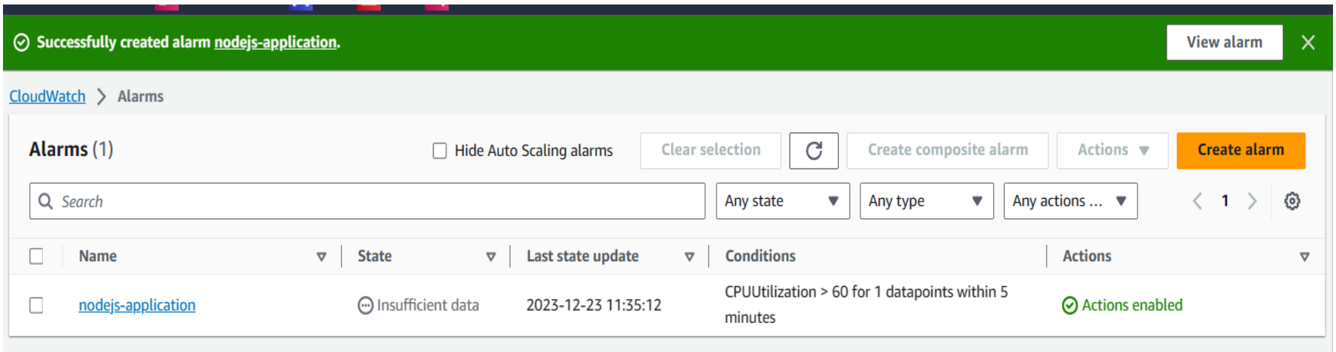
➔ Then **enter alarm name** according to your preferences, click **next**

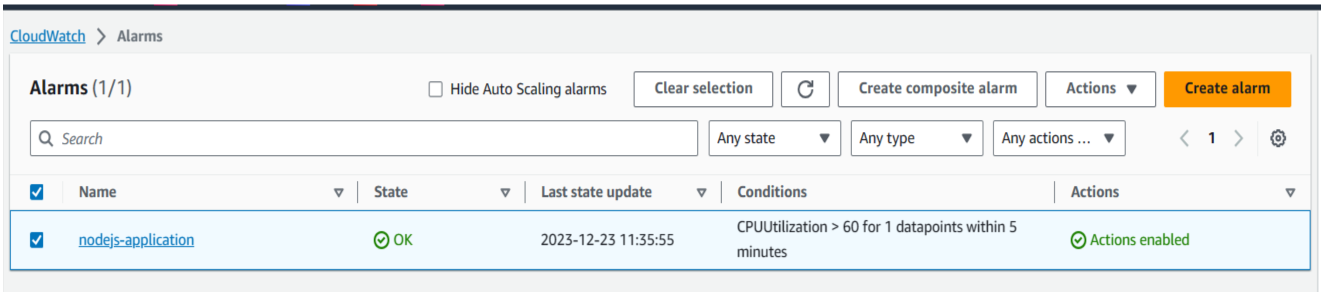


➔ Then just **review and create page** will appear, just review the configurations, **click create alarm:**

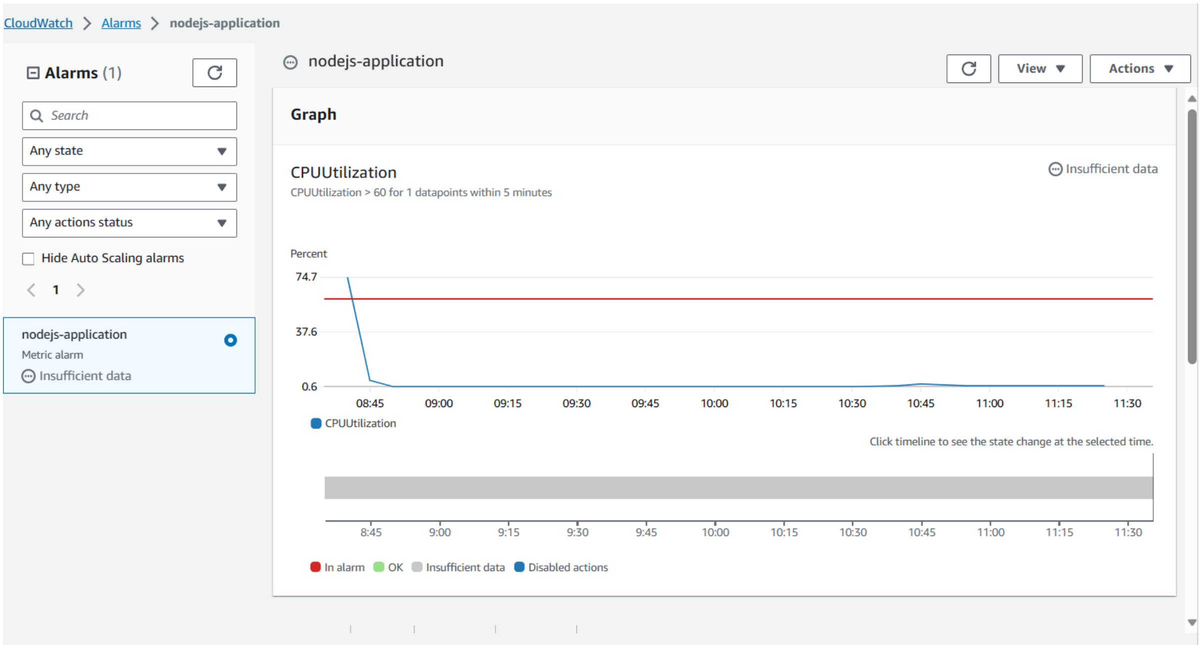


➔ The alarm has been set successfully:





➔ To know more about alarm, select that alarm:



****************************THE END*****************************