Advance Devops Case Study

**CASE STUDY TOPIC:**

* **Automated Deployment with Monitoring**
* **Concepts Used**: Jenkins, EC2, Nagios.
* **Problem Statement**: "Set up a Jenkins CI/CD pipeline to deploy a simple web application on an EC2 instance. Configure Nagios to monitor the deployed application's availability."
* **Tasks**:
  + Create a Jenkins pipeline that builds and deploys a sample web app to an EC2 instance.
  + Install and configure Nagios to monitor the HTTP status of the deployed application.
  + Verify the pipeline by triggering a build and checking the monitoring status in Nagios.

**Introduction:**

In today’s fast-paced software development environment, automation is essential for ensuring rapid, reliable, and repeatable processes. Automated deployment with integrated monitoring allows organizations to streamline their continuous integration and deployment (CI/CD) pipelines while keeping a vigilant eye on the performance and availability of deployed applications. This case study focuses on implementing a solution that uses Jenkins for automation, Amazon EC2 for hosting, and Nagios for monitoring. The goal is to create a Jenkins CI/CD pipeline that deploys a sample web application to an EC2 instance and uses Nagios to monitor the application’s availability.

This study will guide you through setting up the Jenkins pipeline for deploying the web app, configuring Nagios to monitor its HTTP status, and verifying the deployment by triggering a build in Jenkins. The integration of Jenkins and Nagios creates a robust and efficient deployment workflow, ensuring both automation and continuous monitoring of the deployed application.

**Key Features:**

* **Jenkins CI/CD Pipeline**: Jenkins is an open-source automation server used for continuous integration and continuous deployment. It automates the process of building, testing, and deploying code to production environments. In this case study, Jenkins is configured to automatically build and deploy a simple web application to an EC2 instance, reducing manual effort and increasing efficiency.
* **Amazon EC2 Deployment**: Amazon EC2 (Elastic Compute Cloud) provides scalable computing capacity in the cloud. The pipeline deploys the web application onto an EC2 instance, allowing it to be hosted in a flexible, on-demand environment that can scale as needed.
* **Nagios Monitoring**: Nagios is a powerful monitoring tool used to track the status of services and infrastructure. It is configured to monitor the web application deployed on the EC2 instance, specifically checking the HTTP status of the application to ensure it remains available. In case of any downtime or issues, Nagios alerts the team for quick resolution.

**Benefits:**

* **Automation of Deployment**: With Jenkins automating the deployment process, repetitive tasks such as building and deploying applications become streamlined, reducing the likelihood of human error and allowing developers to focus on core development tasks.
* **Continuous Monitoring**: By integrating Nagios, the availability and performance of the deployed application are continuously monitored. Real-time alerts help the team identify and resolve issues before they impact end users, ensuring higher uptime and reliability.
* **Scalability**: Hosting the web application on an Amazon EC2 instance provides scalability, allowing the deployment environment to grow or shrink based on demand, ensuring cost efficiency and performance optimization.

**Third-Year Project Integration**

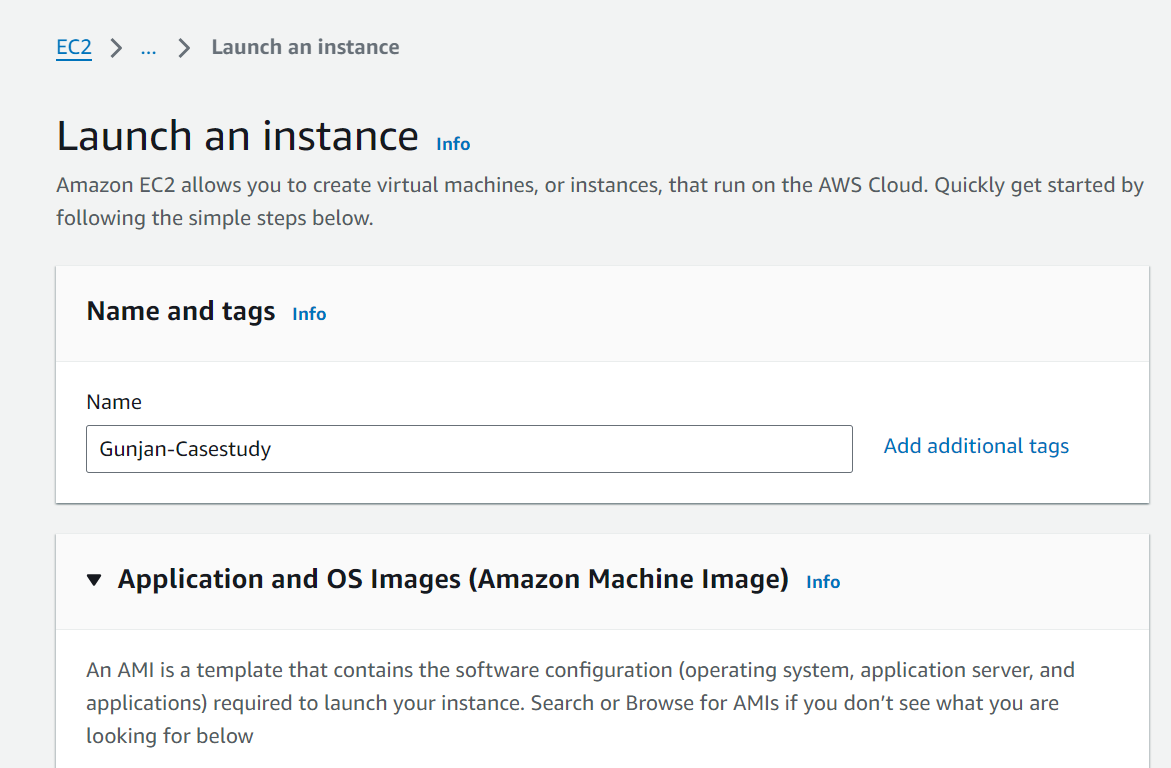
1. **Project Title**: Tiffin Service Website – A platform for students and working professionals to order home-cooked tiffins.
2. **Project Features**:
   * **User Capabilities**: Customers can browse tiffin services, check locations, view business hours, and place orders for tiffins to be delivered at specific times.
   * **User Interface**: The platform has a clean and intuitive interface, ensuring easy navigation, allowing users to view menus, select delivery options, and make payments seamlessly.
3. **Application of Case Study Principles**:
   * **Automated Deployment**:
     + Integrating a **Jenkins CI/CD pipeline** for the Tiffin Service Website will automate the deployment process, making it easier to roll out updates such as new menu items, changes in service locations, or pricing updates.
     + This ensures smooth and reliable updates, reducing the chances of downtime or errors during updates.
   * **Monitoring**:
     + Implementing **Nagios** to monitor critical services within the platform, such as order placement, payment processing, and user navigation, will ensure that the website remains available and functional at all times.
     + Alerts can be set up to notify the team if any of the key services experience downtime, allowing for proactive resolution of issues.
4. **Outcome**:
   * **Enhanced Reliability**: By automating deployment with Jenkins and ensuring continuous monitoring with Nagios, the tiffin service website will deliver a stable and reliable service to its users, ensuring customer satisfaction.
   * **Improved User Experience**: With the seamless deployment of new features and the continuous monitoring of website functionality, users will benefit from a smooth, uninterrupted experience when placing orders and interacting with the platform.

**IMPLEMENTATION**

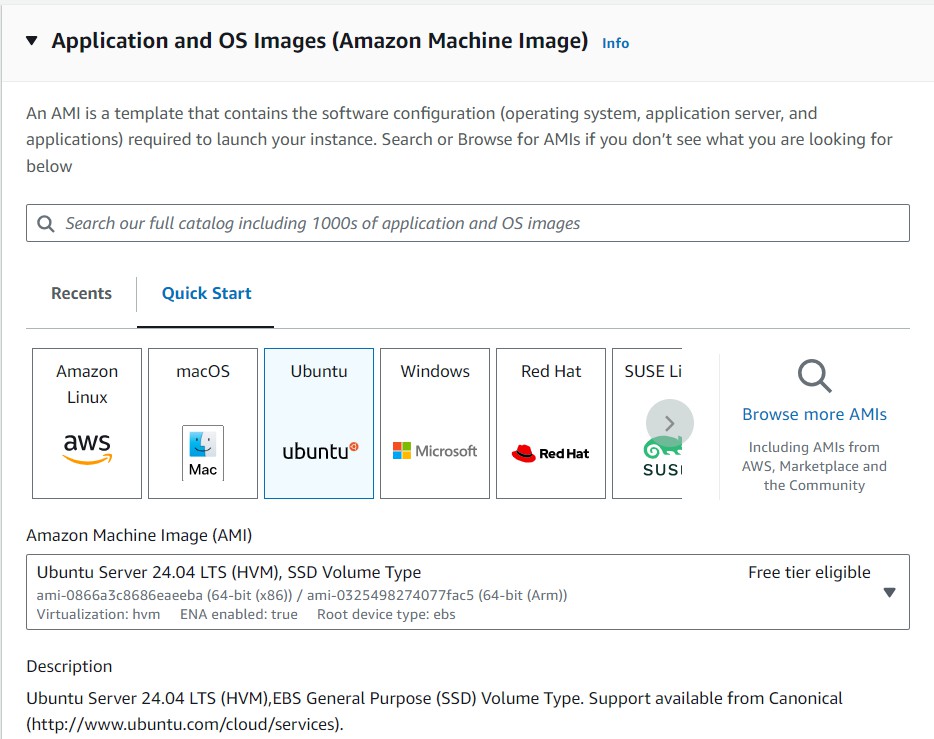
1. **Create a Jenkins pipeline that builds and deploys a sample web app to an EC2 instance.**

**Launch EC2 instance:**

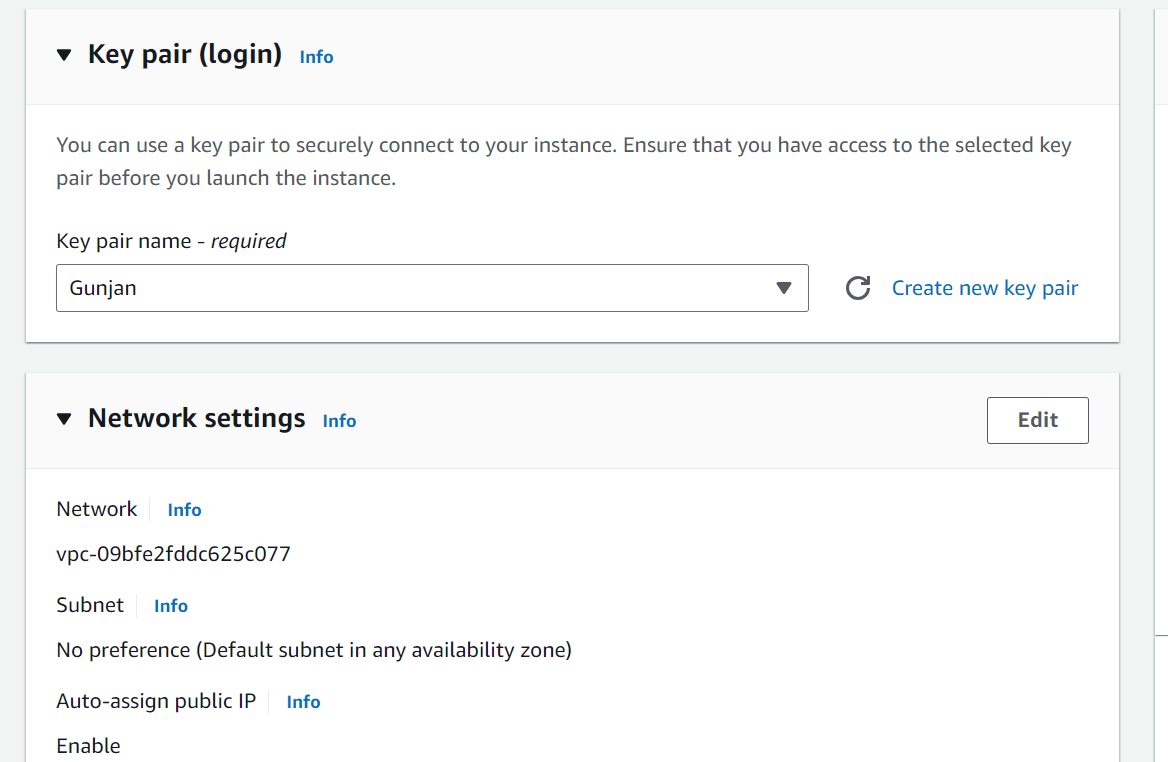
Click on "Launch Instance" and configure**:**



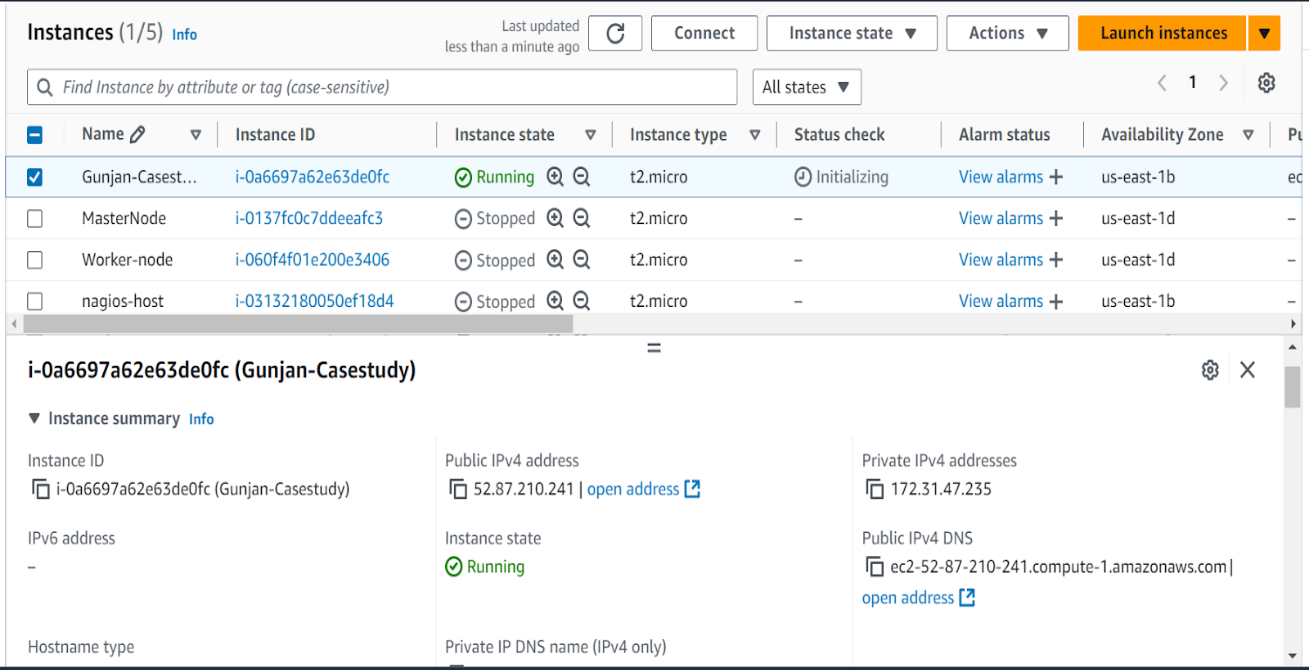
* + - **AMI: Select Amazon Linux 2 or Ubuntu Server 22.04.**



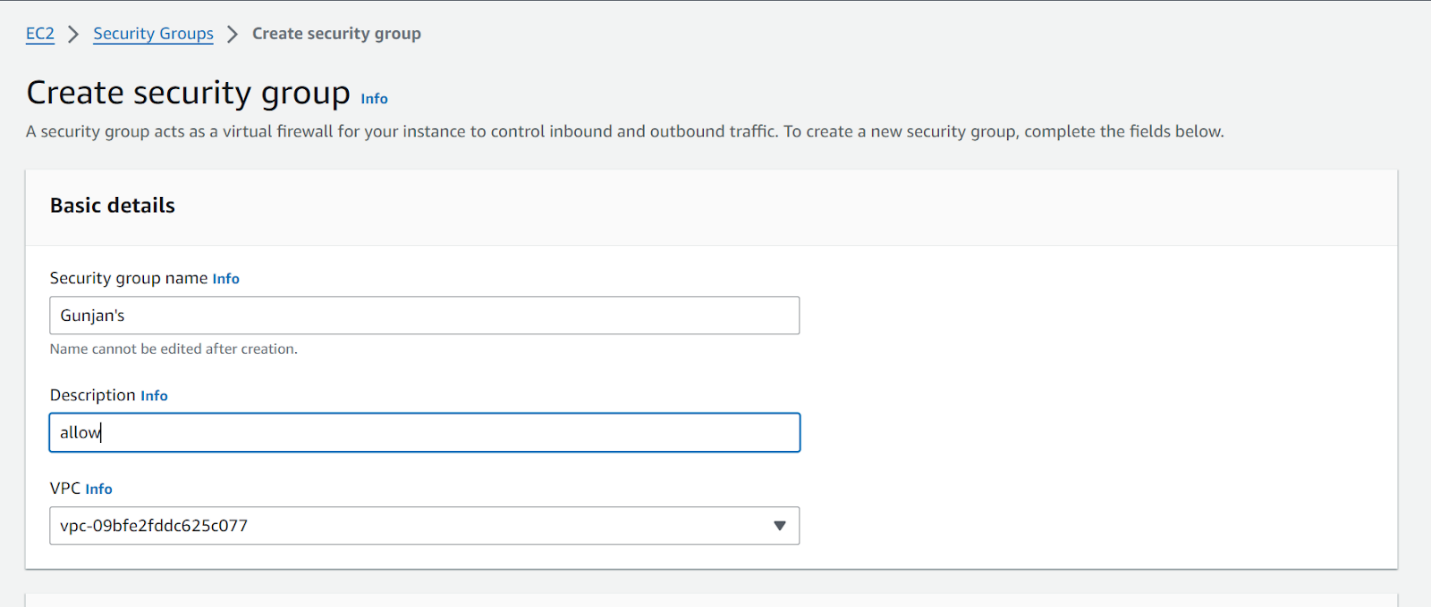
* + - **Instance type**: t2.micro (Free-tier eligible).
    - **Key pair**: Create or use an existing key pair for SSH access.



* + - **Launch** the instance.

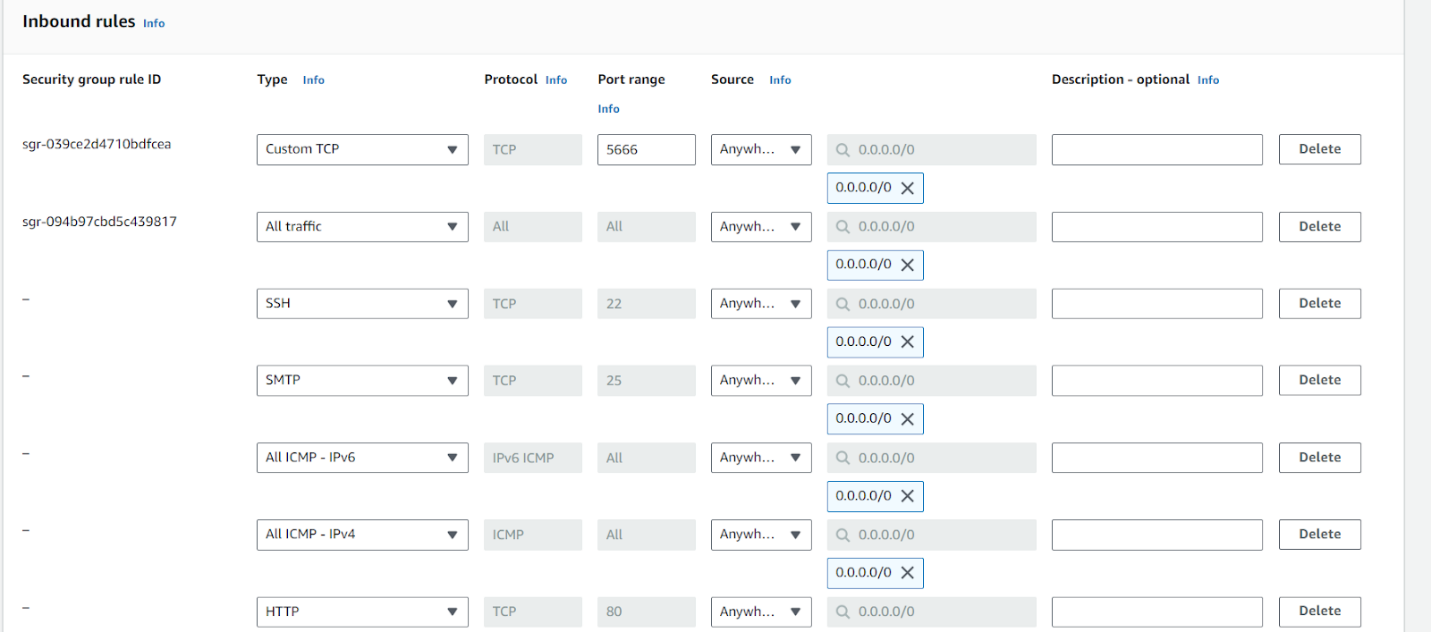


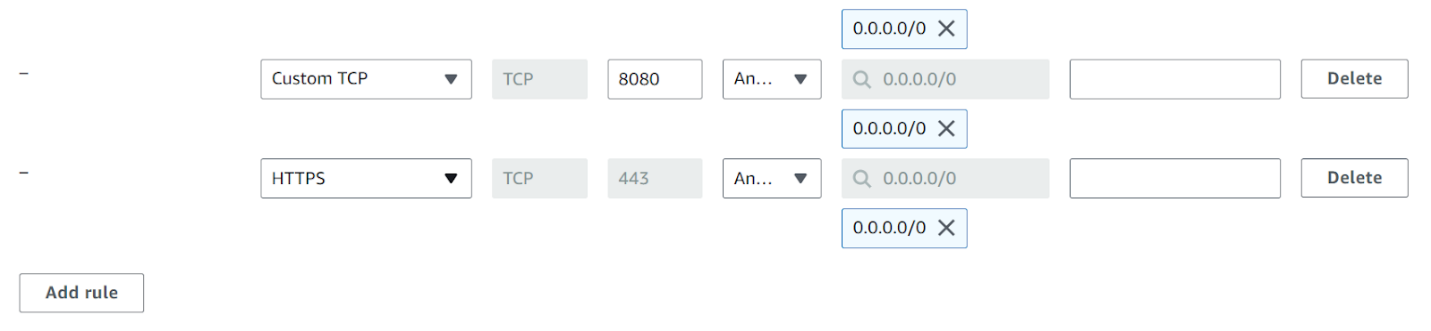
**Then in Security Group**:



### Configure inbound rules

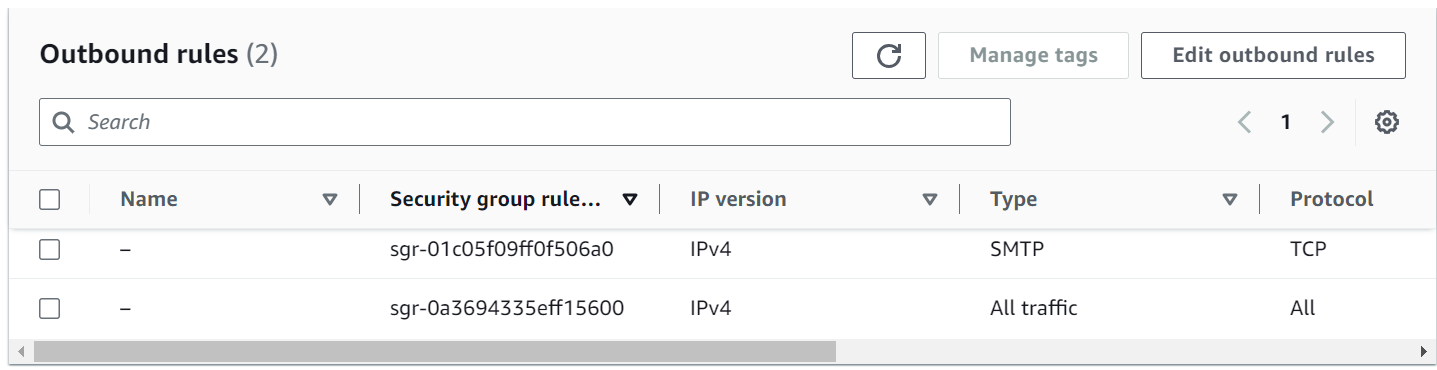
* + - SSH (TCP 22): Your IP.
    - HTTP (TCP 80): Open to all (0.0.0.0/0).
    - Nagios (TCP 5666): Open to all (0.0.0.0/0).
    - All TCP (TCP 0-65535): Open to all (0.0.0.0/0).
    - SMTP (TCP 587): Open to all (0.0.0.0/0)
    - 8080 (Jenkins)





### Configure Outbound rule for email-notification

* + - SMTP (TCP 587): Open to all (0.0.0.0/0), for sending emails through Gmail's SMTP server.



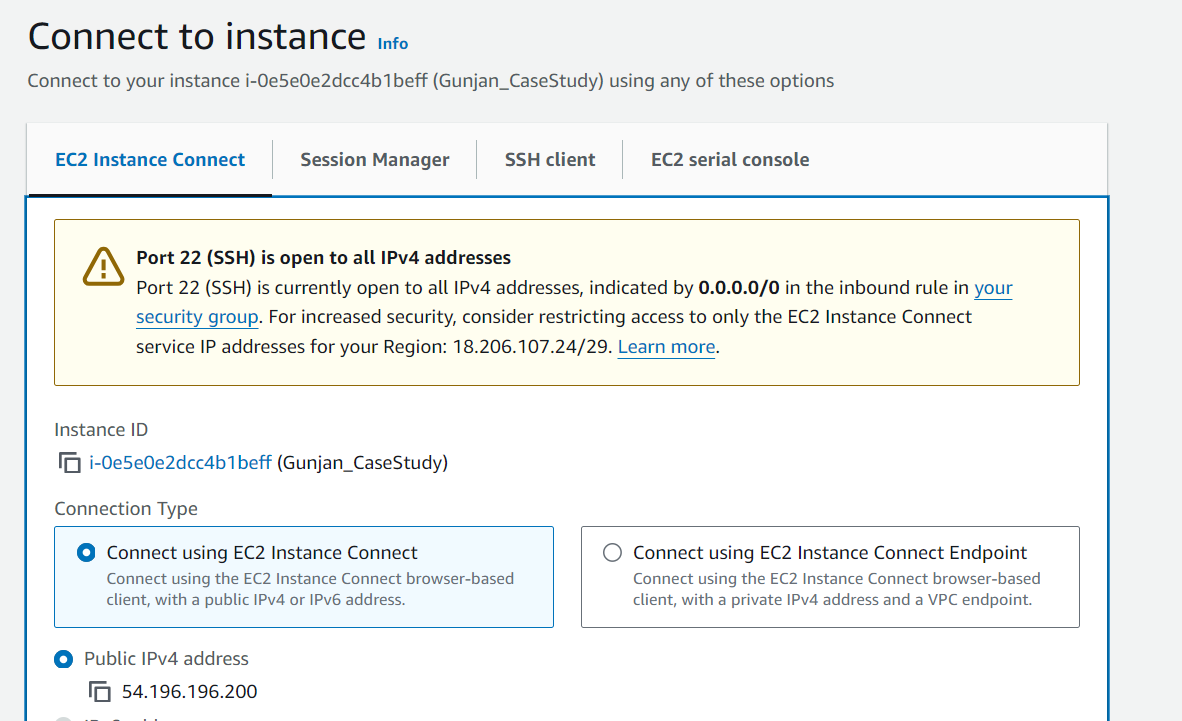


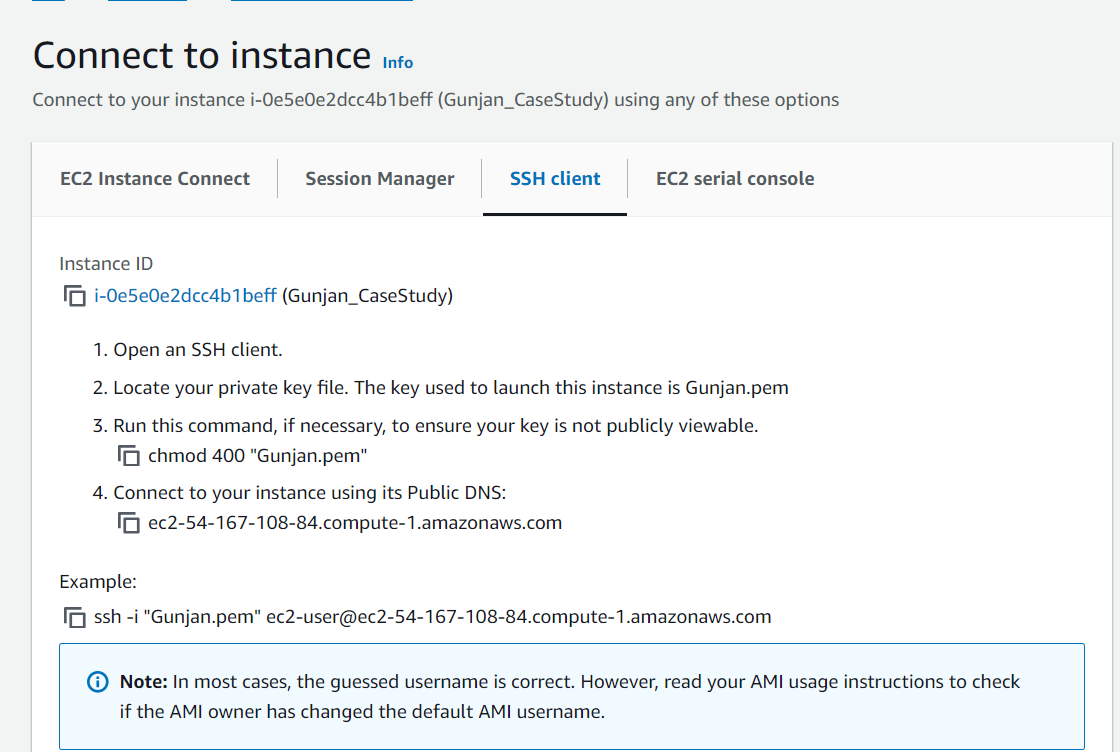
### Allocate an Elastic IP Address

* + - Go to the AWS Management Console.
    - Open the EC2 service.
    - In the left-hand menu, under Network & Security, click on Elastic IPs.
    - Click the Allocate Elastic IP address button.
    - Choose the Amazon pool of IPv4 addresses, then click Allocate.

### Connect to the Instance:

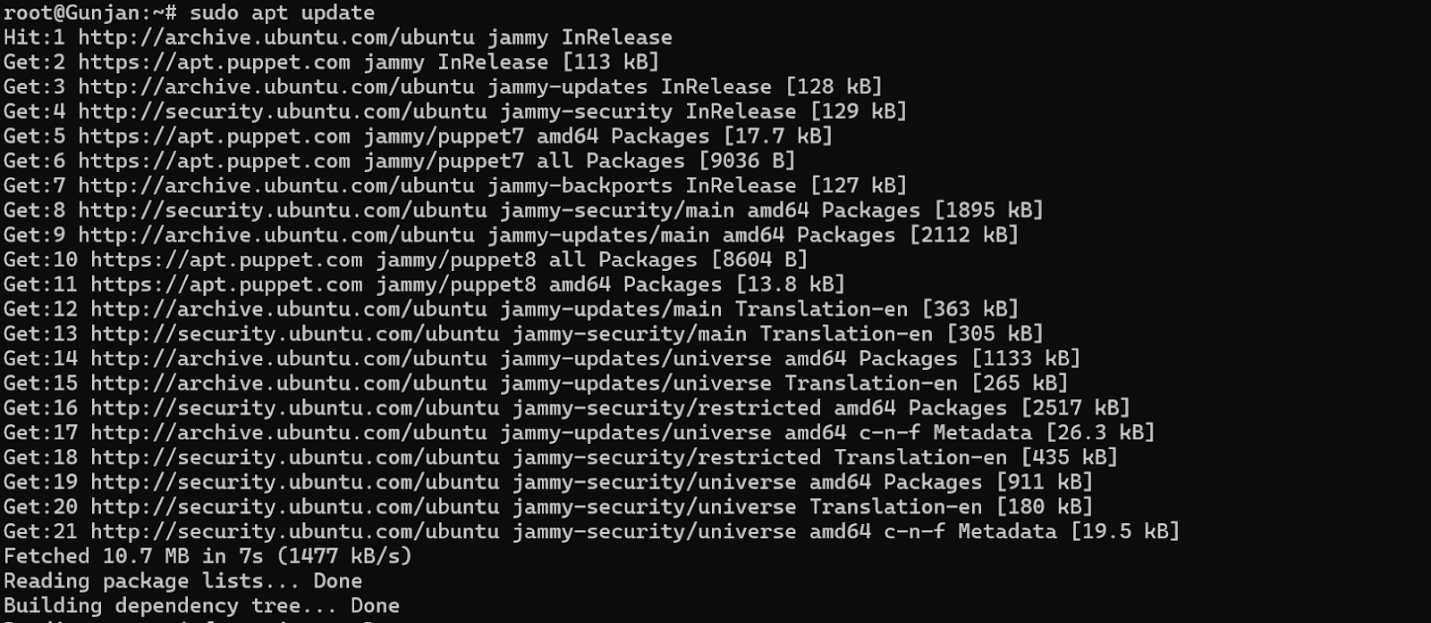
* + - Click the Connect button at the top of the Instances page.





### Install Apache (if not installed)

* + - sudo apt update



* + - sudo apt install apache2





### Enable Apache to Start on Boot

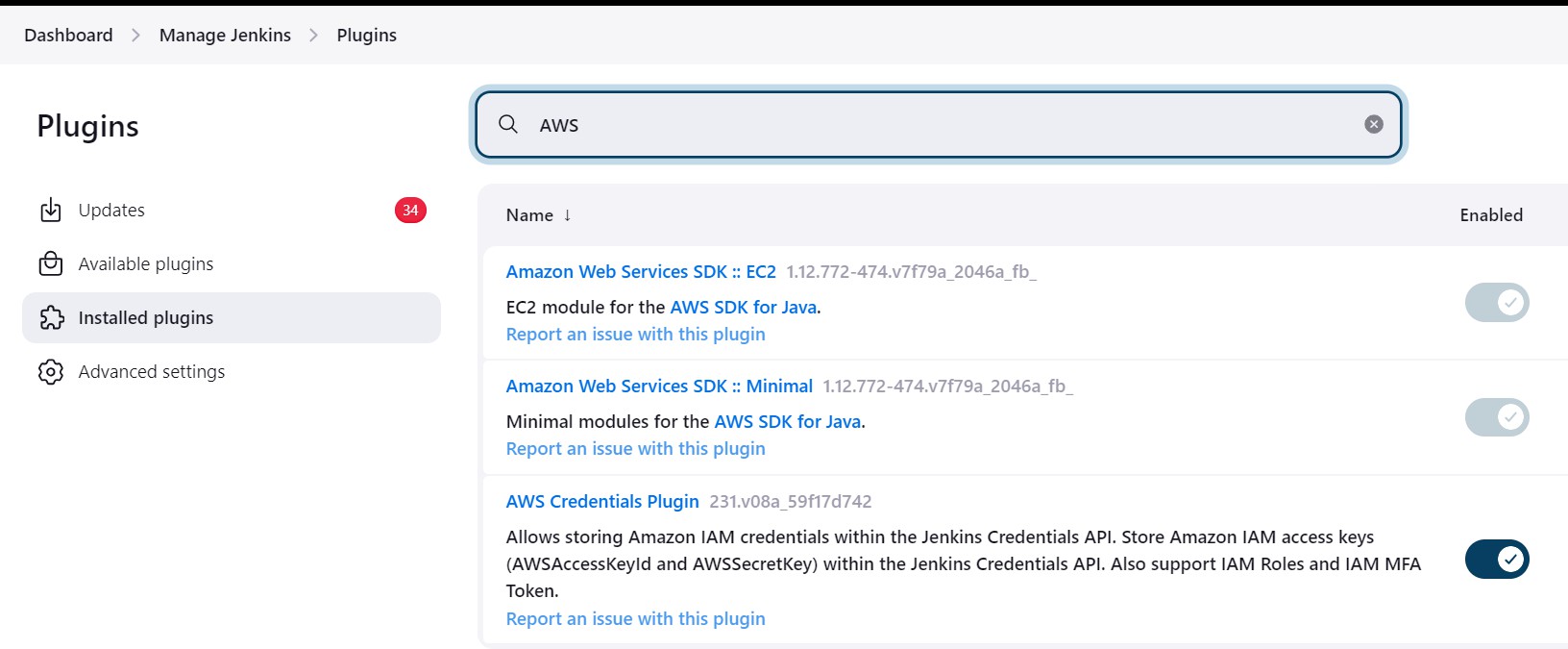
sudo systemctl enable apache2



* 1. **JENKINS Setup**

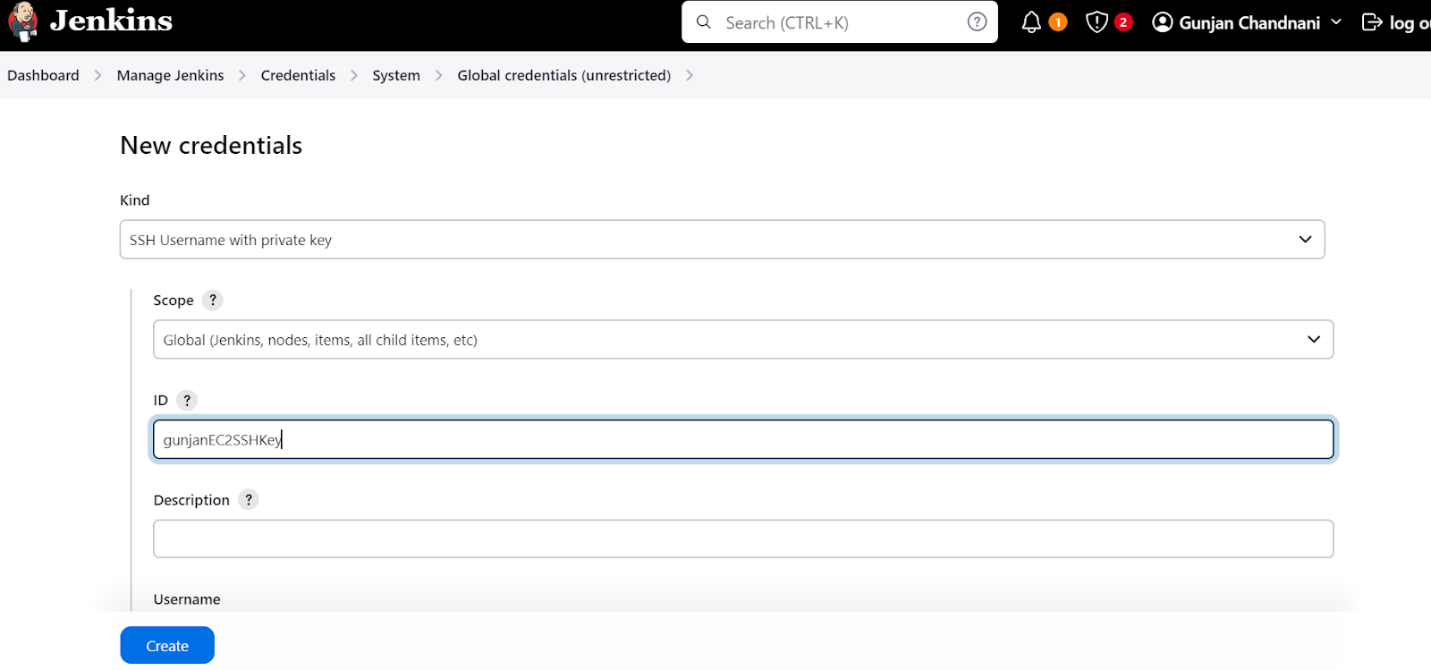
**Install Required Jenkins Plugins:**

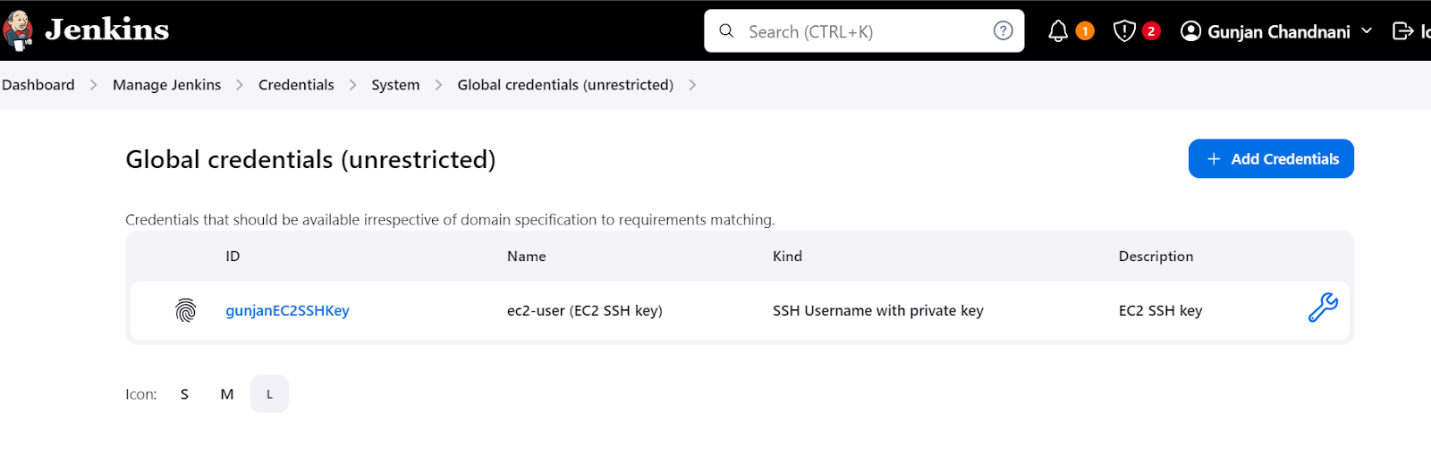
* + - Log in to Jenkins at http://localhost:8080.
    - Go to Manage Jenkins -> Manage Plugins.
    - Install the following plugins:
      * AWS EC2 Plugin (for managing EC2 instances).
      * SSH Agent Plugin (to SSH into your EC2 instance during deployment).

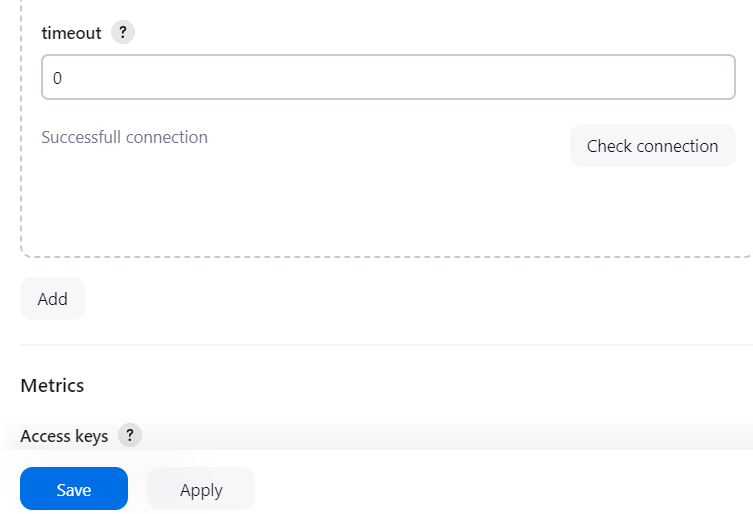
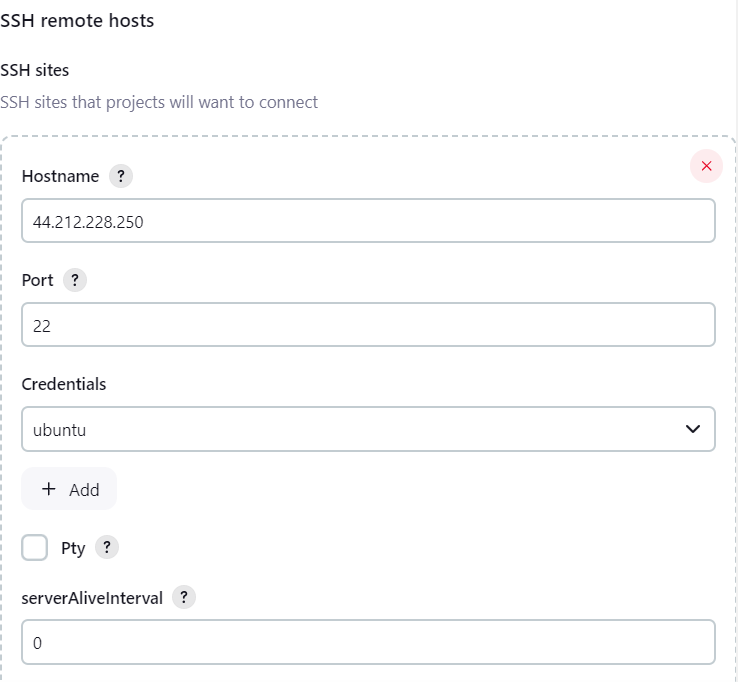


**Steps to Handle the SSH Key in Jenkins:**

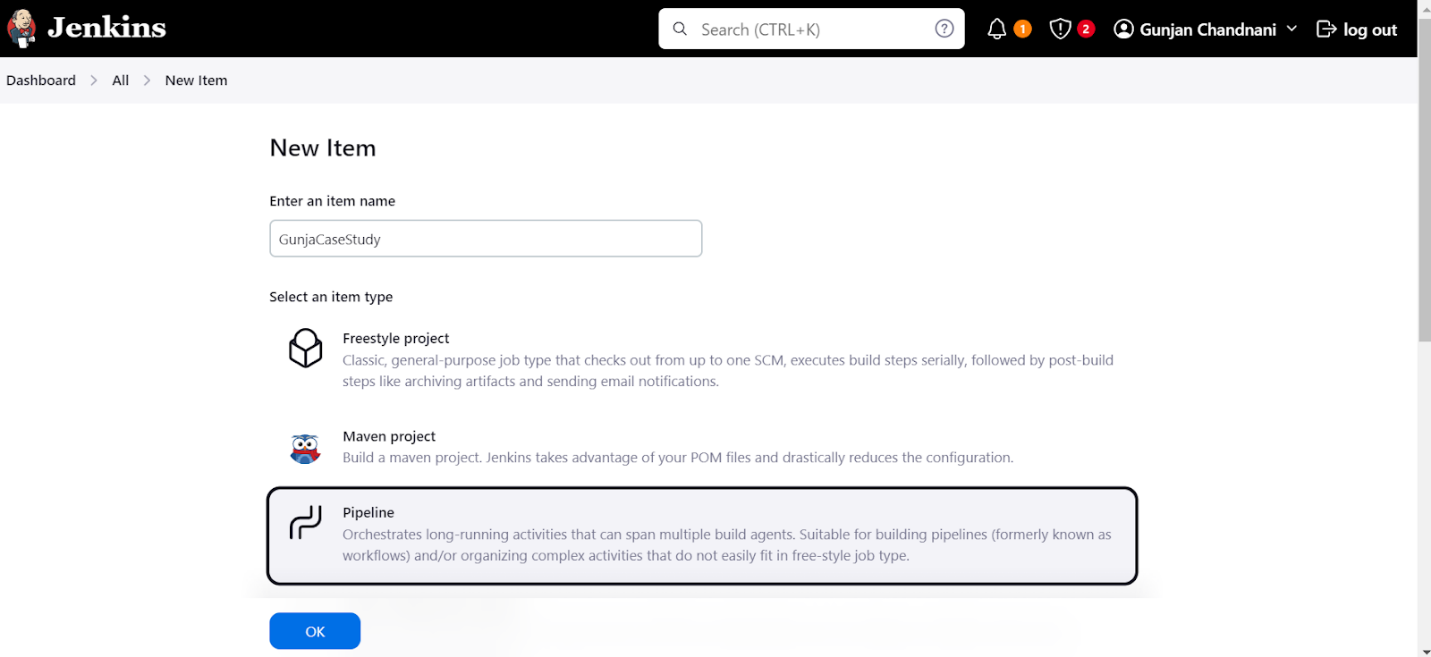
1. **Add SSH Credentials in Jenkins:**
   * Go to Manage Jenkins → Manage Credentials → (global) → Add Credentials.
   * Under Kind, select SSH Username with private key.
   * Fill in the Username (for example, ubuntu for Ubuntu instances).
   * For Private Key, select Enter directly, then paste the content of yourcasestudy.pem file.







**Build a Jenkins Pipeline:**



**This is the code to paste in Script:**

pipeline { agent any stages {

stage('Checkout Code') { steps {

// Clone your repository to get the latest index.html

git url: 'https://github.com/gunjan-chandnani/CaseStudy.git', branch: 'main'

}

}

stage('Build') { steps {

echo 'Building the application...'

}

}

stage('Test SSH Connection') { steps {

script {

echo "Testing SSH connection to EC2 instance..." try {

sshagent(['ec2-ssh-credential']) {

// Attempt to run a simple command on the EC2 instance sh """

ssh -o StrictHostKeyChecking=no -i "C:/Users/gunja/Downloads/casestudy.pem" [ubuntu@ec2-44-212-228-250.compute-](mailto:ubuntu@ec2-44-212-228-250.compute-) 1.amazonaws.com "echo 'SSH connection successful'"

"""

}

echo "SSH connection test successful."

} catch (Exception e) {

echo "SSH connection test failed: ${e.getMessage()}"

}

}

}

}

stage('Deploy') { steps {

script {

// Set the path to the index.html file in the cloned repository def indexFilePath = "${WORKSPACE}/index.html"

echo "Workspace directory: ${WORKSPACE}" echo "Index file path: ${indexFilePath}"

try {

echo "Copying index.html to EC2 instance..." sshagent(['ec2-ssh-credential']) {

sh """

scp -o StrictHostKeyChecking=no -i "C:/Users/Shravani/Downloads/casestudy.pem" ${indexFilePath} ubuntu@ec2-44-212-228- 250.compute-1.amazonaws.com:/var/www/html/index.html

"""

}

echo "Restarting Apache server..." sshagent(['ec2-ssh-credential']) {

sh """

ssh -o StrictHostKeyChecking=no -i "C:/Users/gunja/Downloads/casestudy.pem" [ubuntu@ec2-44-212-228-250.compute-](mailto:ubuntu@ec2-44-212-228-250.compute-) 1.amazonaws.com "sudo systemctl restart apache2"

"""

}

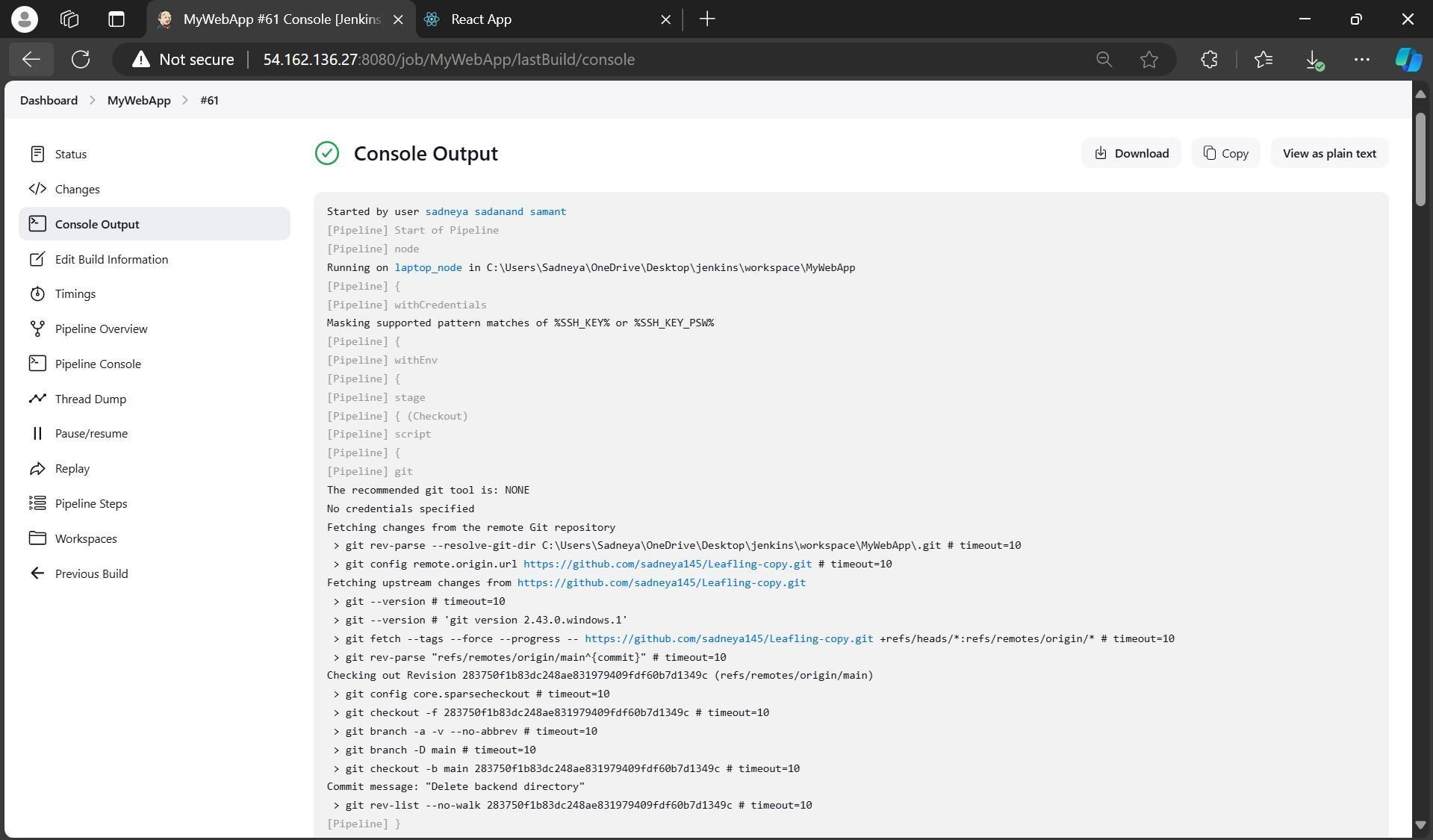
}

}}}}

} catch (Exception e) {

echo "Error during deployment: ${e.getMessage()}

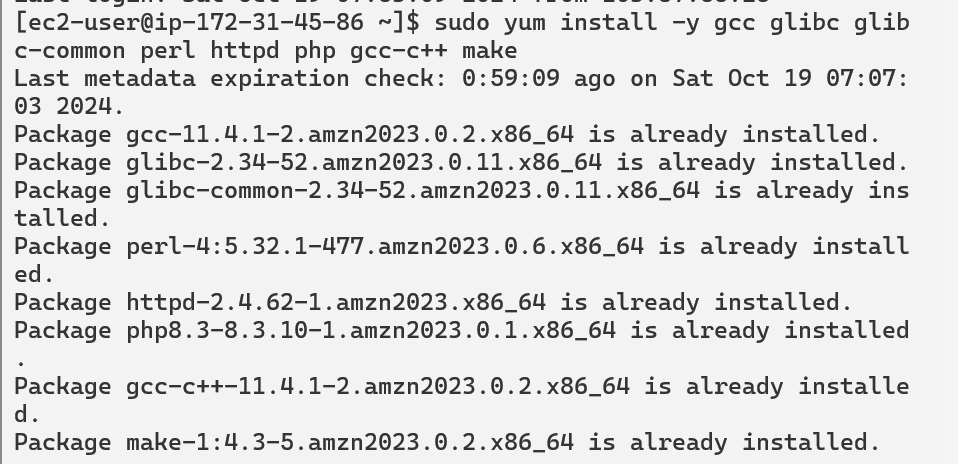
**Result:**



1. **Installation Of Nagios For Monitoring:**
   1. **Installing Required Packages Update the Instance:**

sudo yum update -y



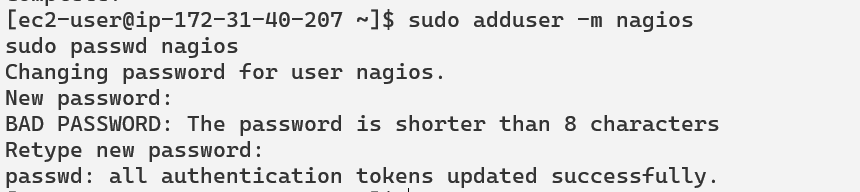
**Install Required Packages:**

sudo yum install gcc glibc glibc-common perl httpd php gcc-c++ make

**User and Group Creation Create a New User:**

sudo adduser -m nagios

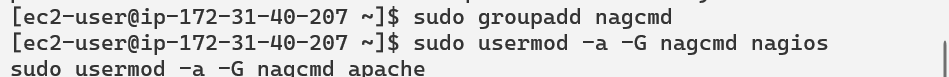
sudo passwd nagios



**Create a New User Group:**

sudo groupadd nagcmd

**Modify User Groups:**

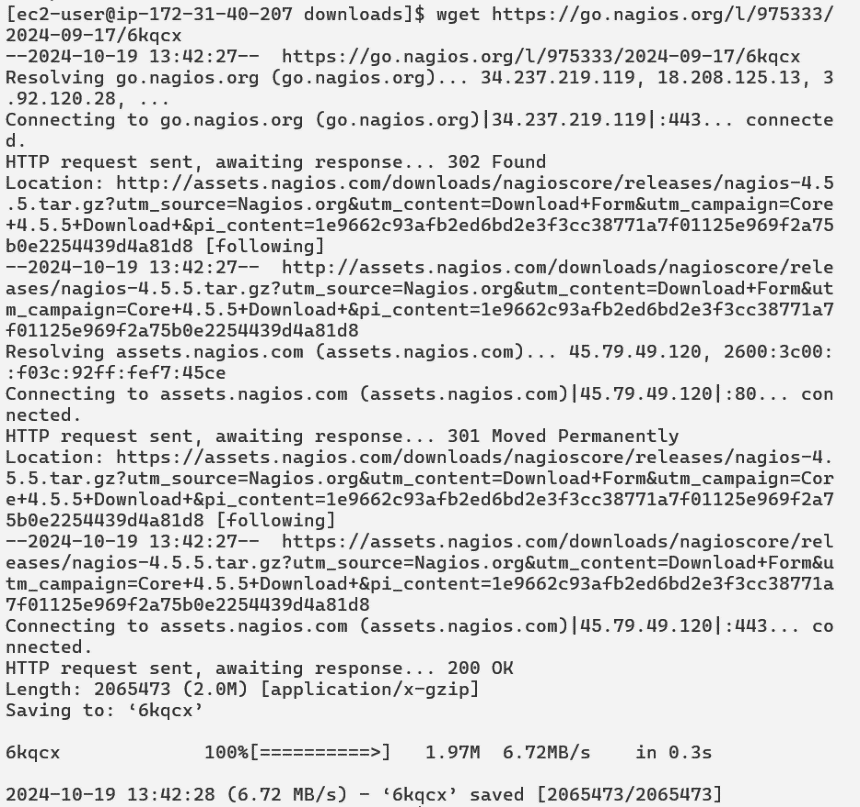
sudo usermod -a -G nagcmd nagios sudo usermod -a -G nagcmd apache

1. **Setting Up Nagios**

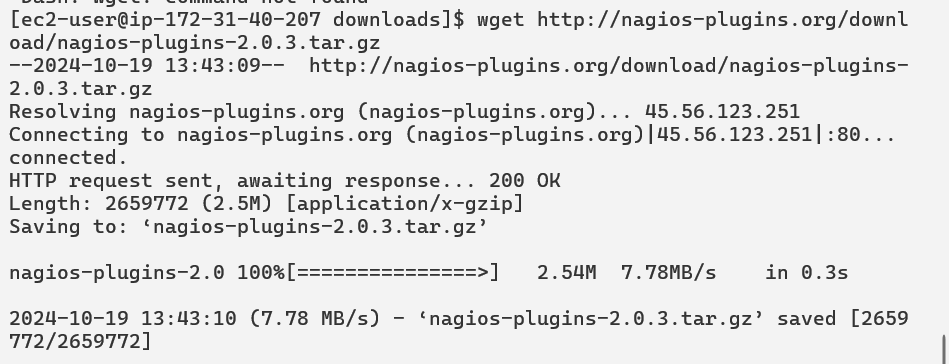
**Create a Directory for Nagios Downloads:**

mkdir ~/downloads cd ~/downloads

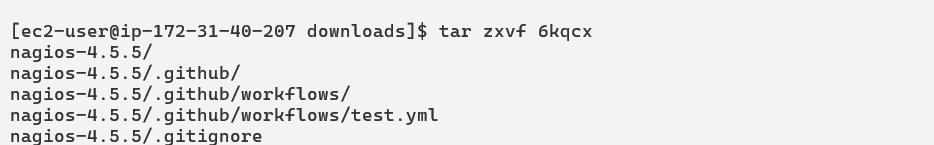
1. **Download Nagios Source Files:**

wget <https://go.nagios.org/l/975333/2024-09-17/6kqcx>

1. **Download Nagios Plugins:**

wget <http://nagios-plugins.org/download/nagios-plugins-2.0.3.tar.gz>

1. **Unzip the Nagios Source Files:**

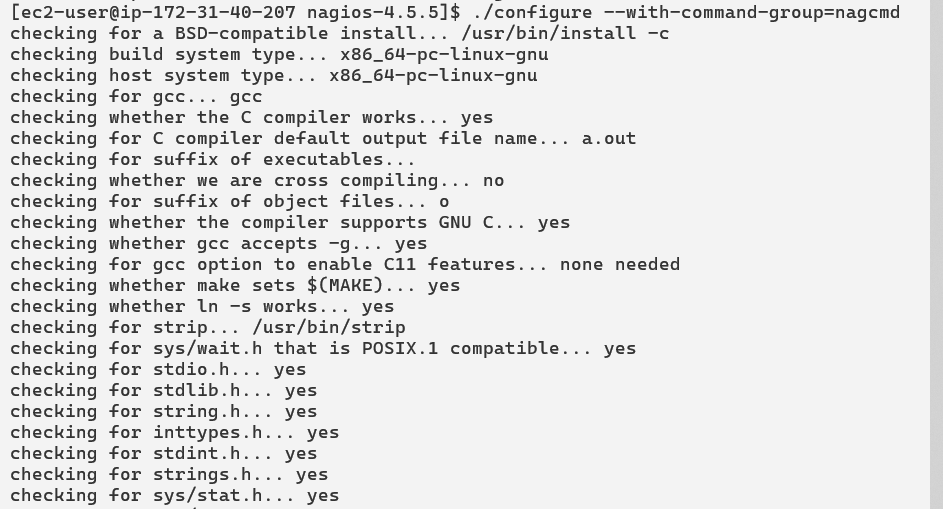
tar zxvf 6kqcx

cd nagios-4.5.5



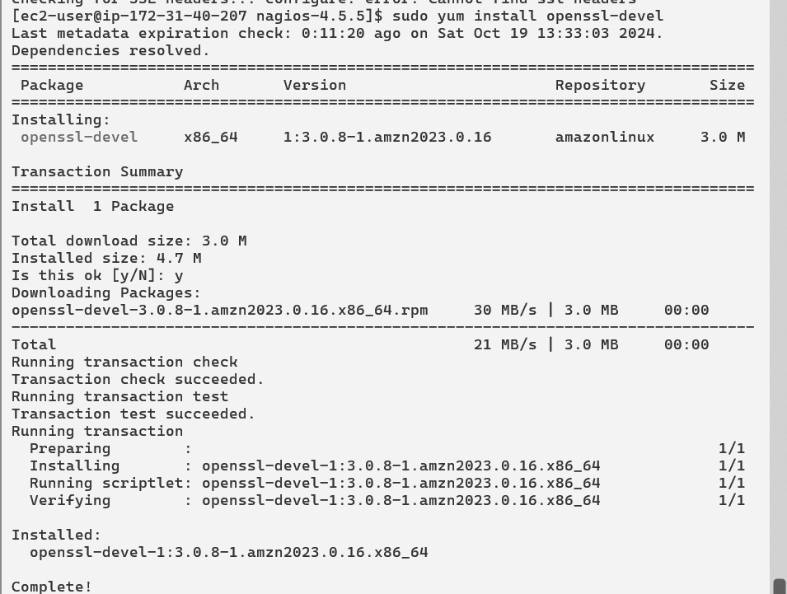
1. **Run Configuration Script:**

./configure --with-command-group=nagcmd



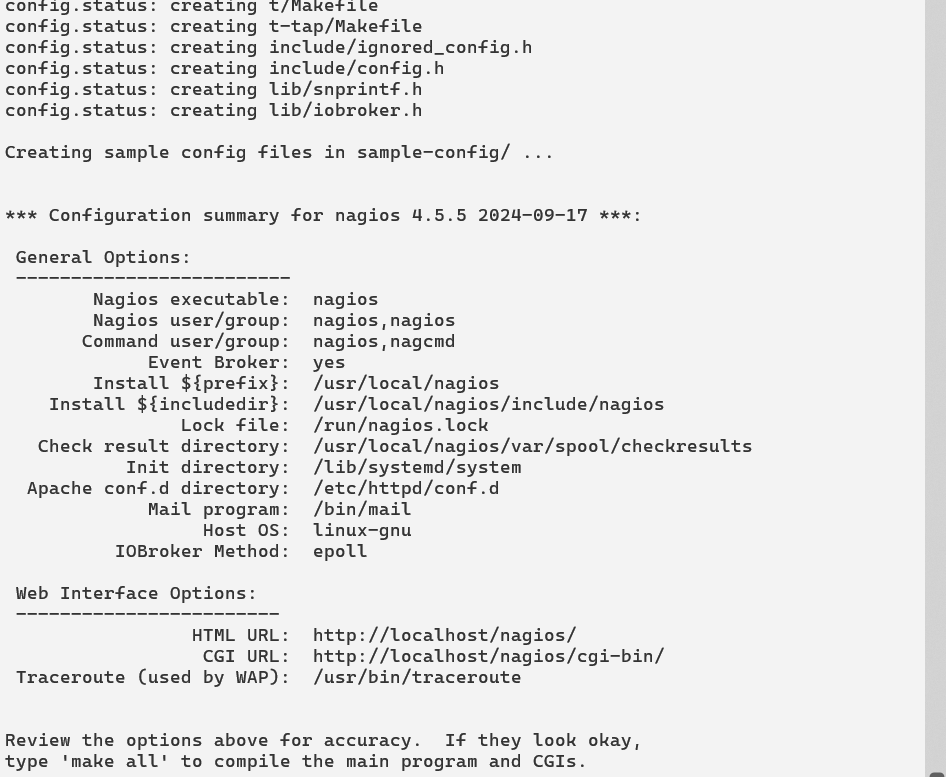
**\*\*Error Handling:** If you encounter an error about missing SSL headers, install the following:

1. **Install SSL Development Package:**

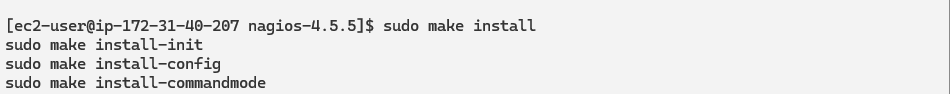
sudo yum install openssl-devel -y

**Rerun Configuration Script:** You will get final output like this

./configure --with-command-group=nagcmd

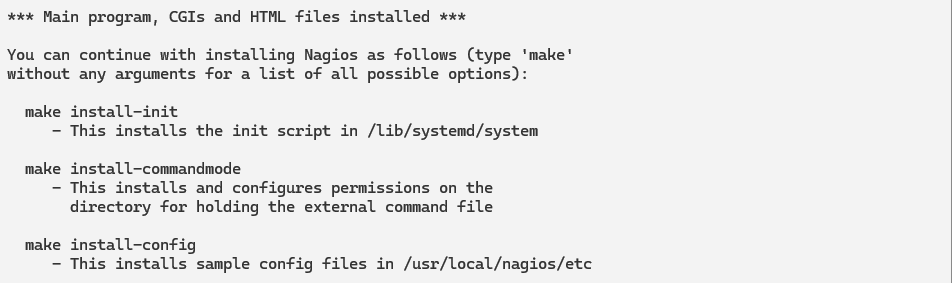


**Install Nagios:** sudo make install sudo make install-init

sudo make install-config

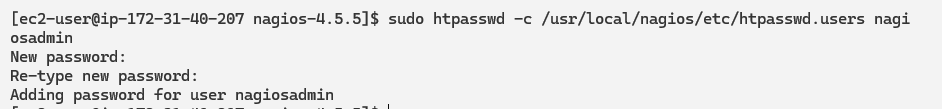
sudo make install-commandmode



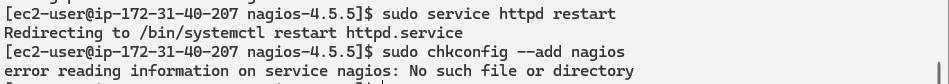


**Configure Nagios Web Interface:**

sudo make install-webconf

**Create Nagios Admin Account:**

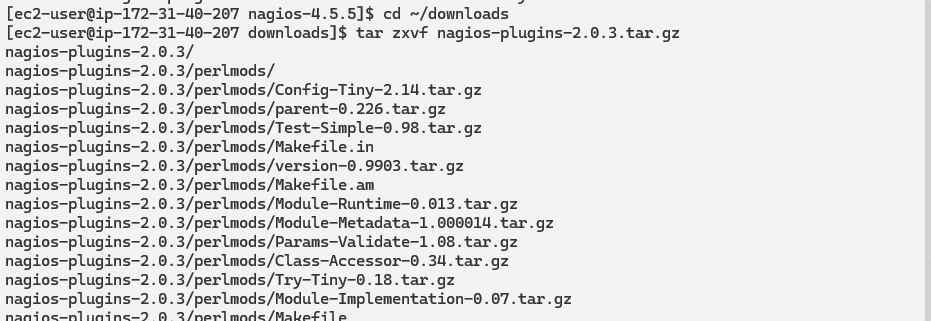
sudo htpasswd -c /usr/local/nagios/etc/htpasswd.users nagiosadmin

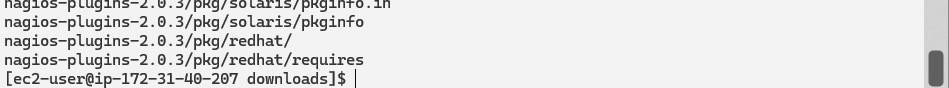
**Restart Apache:**

sudo service httpd restart

**Unzip Nagios Plugins:**

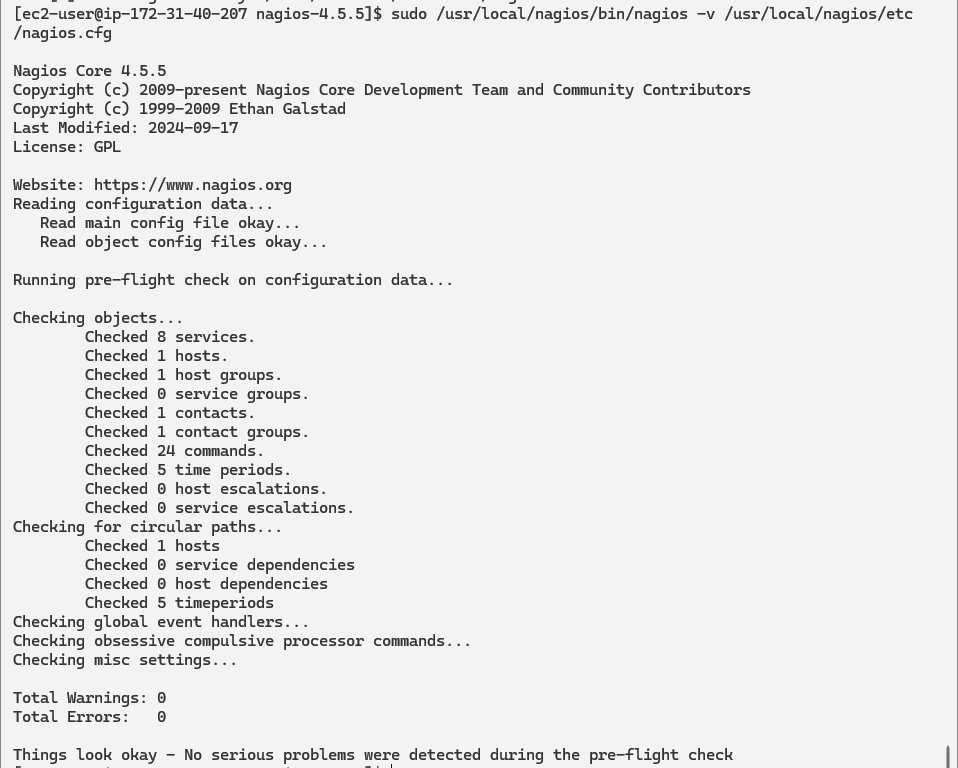
cd ~/downloads

tar zxvf nagios-plugins-2.0.3.tar.gz cd nagios-plugins-2.0.



**Verify Nagios Configuration:**

sudo /usr/local/nagios/bin/nagios -v /usr/local/nagios/etc/nagios.cfg



**Start Nagios Service:** sudo service nagios start

**Check Nagios Status:** sudo systemctl status nagios



**Get Public IP Address:**Go back to the EC2 Console and copy the public IP address of your instance.

**Access Nagios Web Interface:** Open your web browser and navigate to: http://<your\_public\_ip\_address>/nagios

Enter the username (nagiosadmin) and the password you set in Step 15.

**Making Changes for application in Nagios**

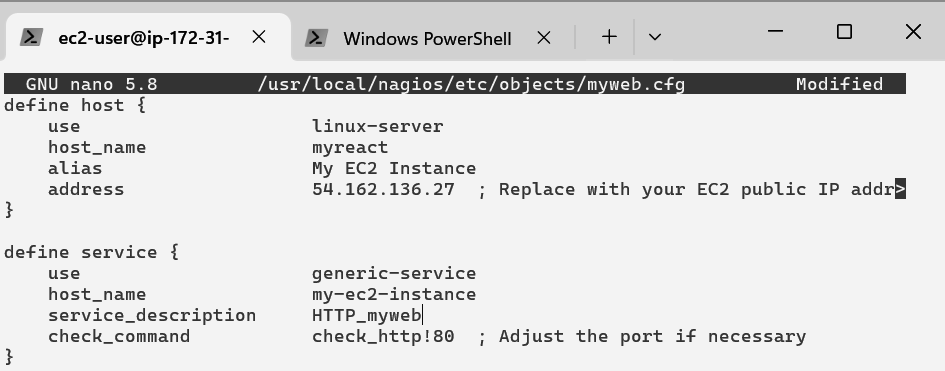
**Goto cofigurations file**



Add the file which we are newly creating for montoring of our application cfg\_file=/usr/local/nagios/etc/objects/myweb.cfg add this

**Write inside that file the required information**

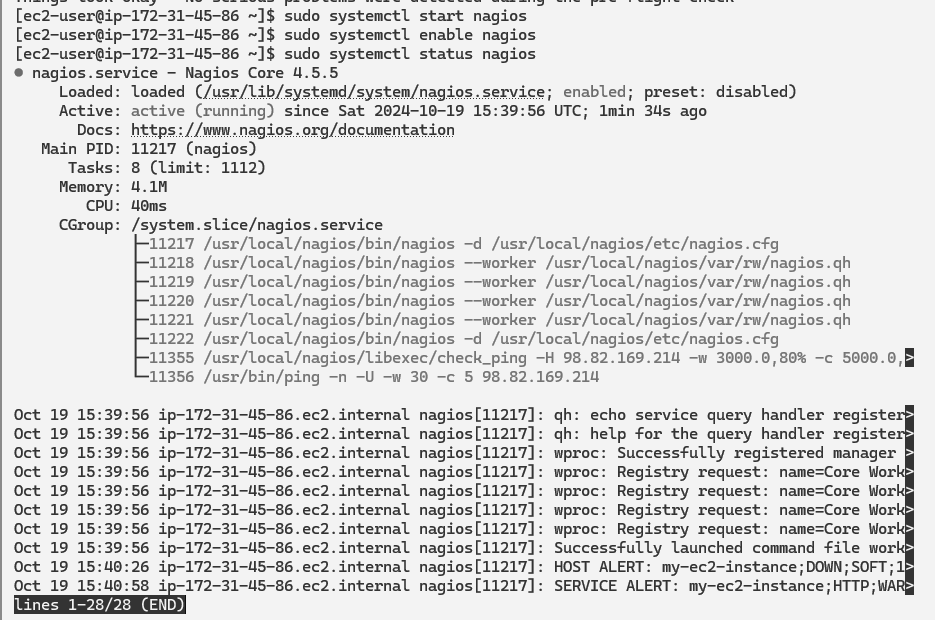
sudo nano /usr/local/nagios/etc/objects/myweb.cfg



**Again Verify the changes** by : sudo /usr/local/nagios/bin/nagios -v

/usr

**Again start nagios** sudo systemctl start nagios and check status by sudo systemctl status nagios



**Go to commands file** there make changes in check\_http section

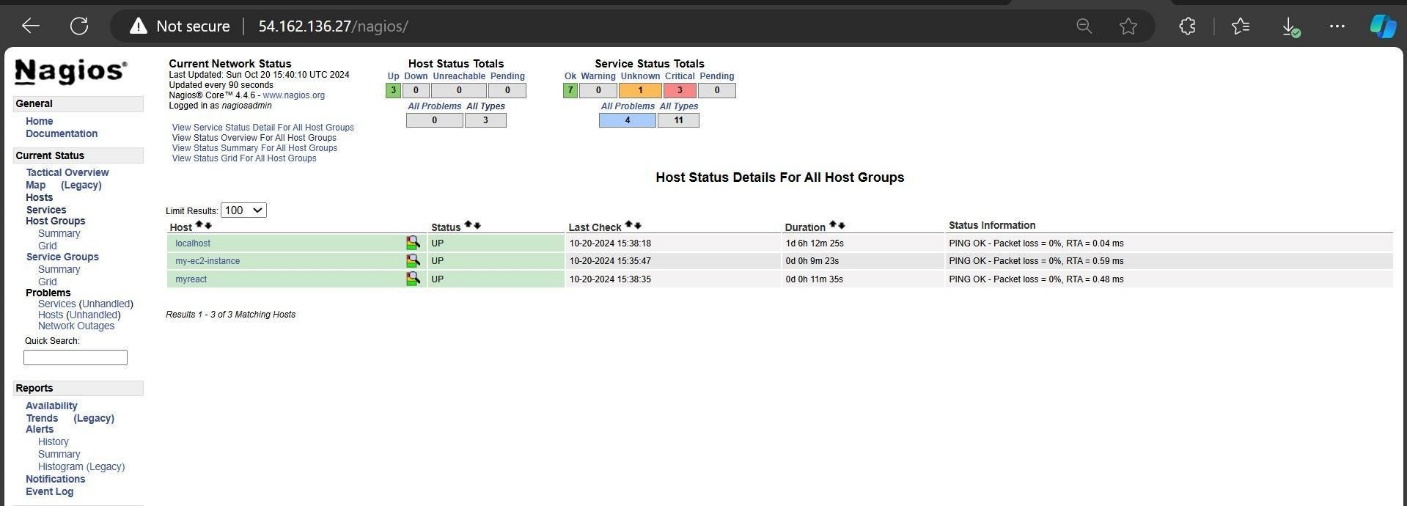
Add -> command line /usr/local/nagios/libexec/check\_http -H $HOSTADDRESS$ -p

$ARG1$ $ARG2$

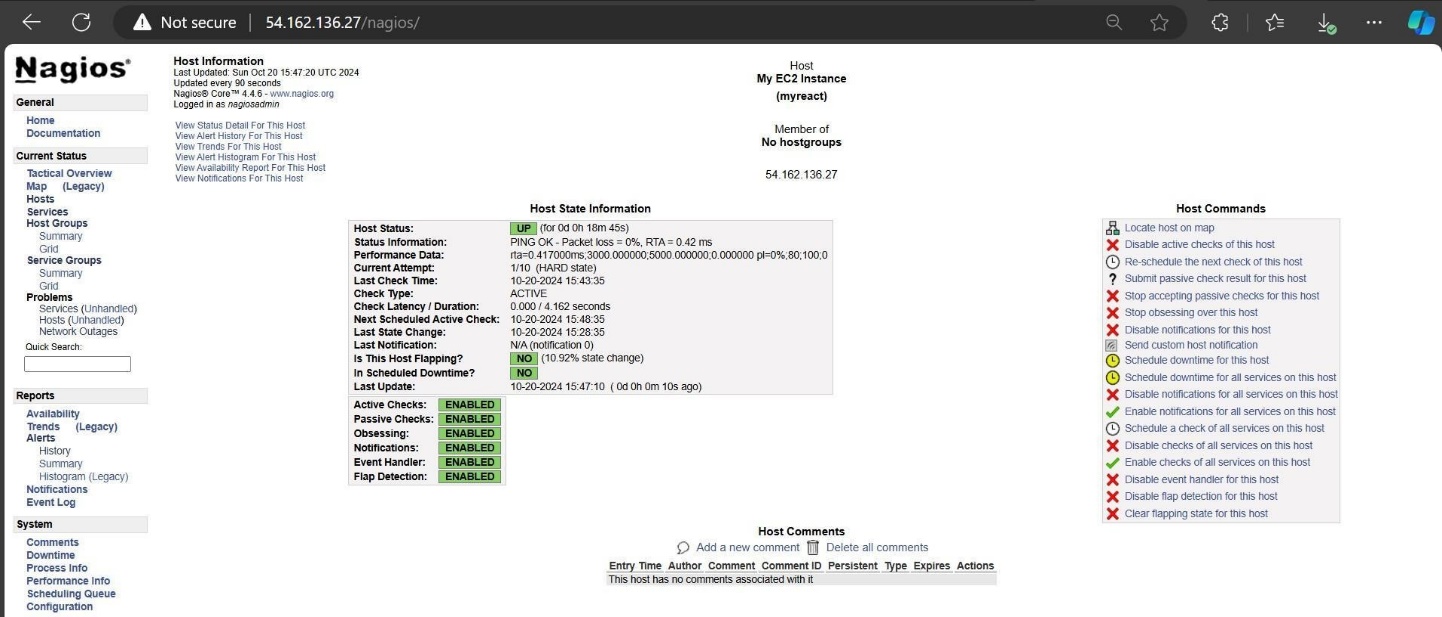


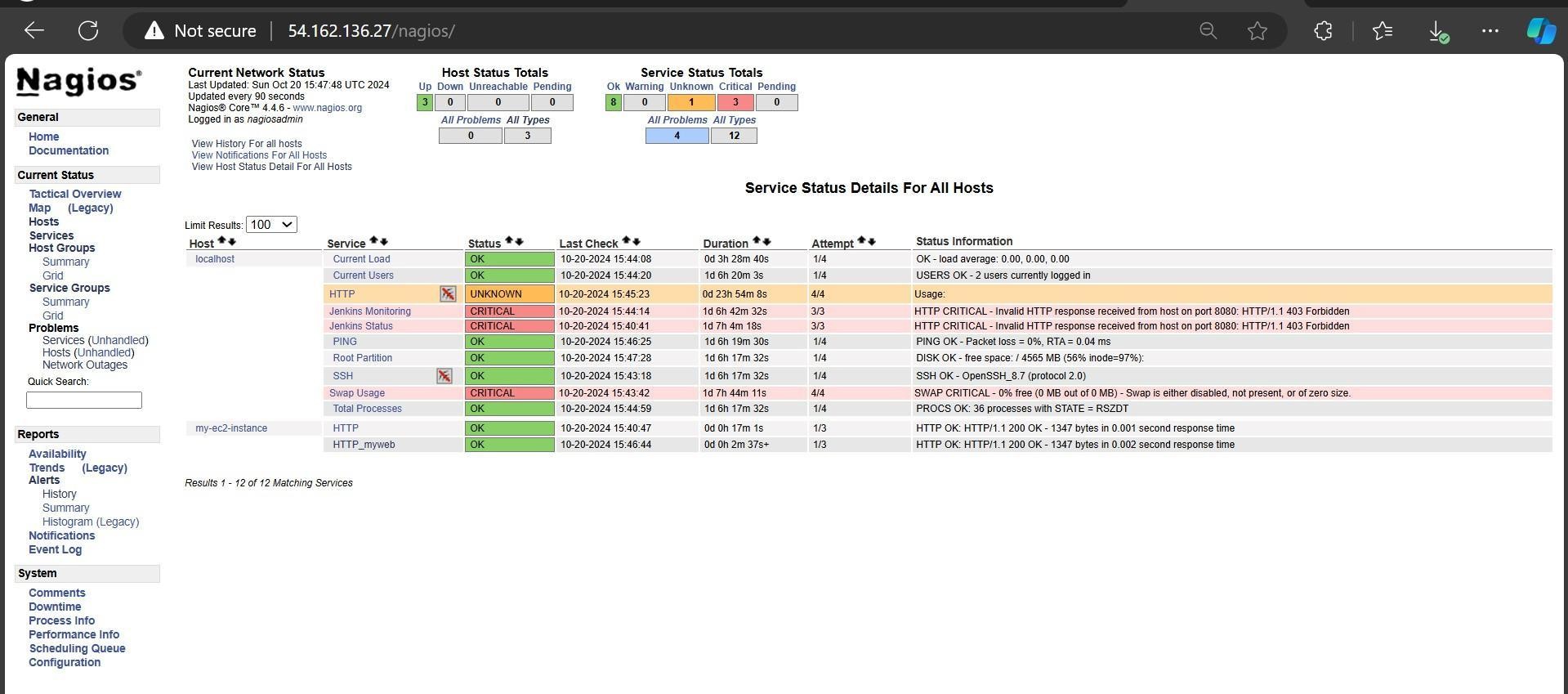
**Nagios page:** Go back to your nagios page you will se output.here you will see

my-ec2-instance.



Then Go to host section present on left sidebar and click on “my-ec2-intsance” it will give host information.



After clicking on services on left sidebar you will get detailed information about network status.

**Conclusion:**

Integrating automated deployment and monitoring into the Tiffin Service Website enhances reliability and user experience. The Jenkins CI/CD pipeline allows for quick updates, while Nagios ensures high availability and proactive issue resolution. Together, these practices create a solid foundation for the platform, ensuring seamless service delivery and fostering customer trust and satisfaction.