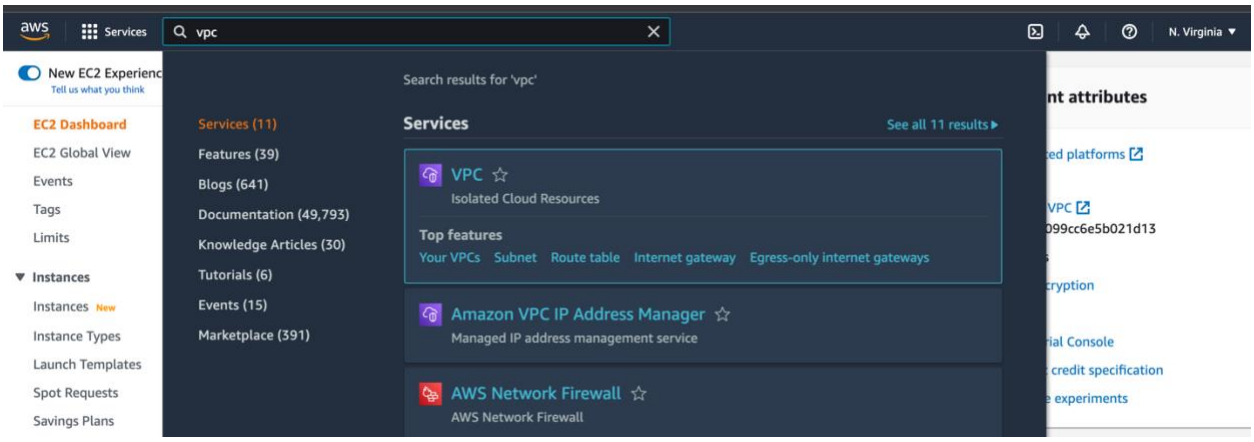
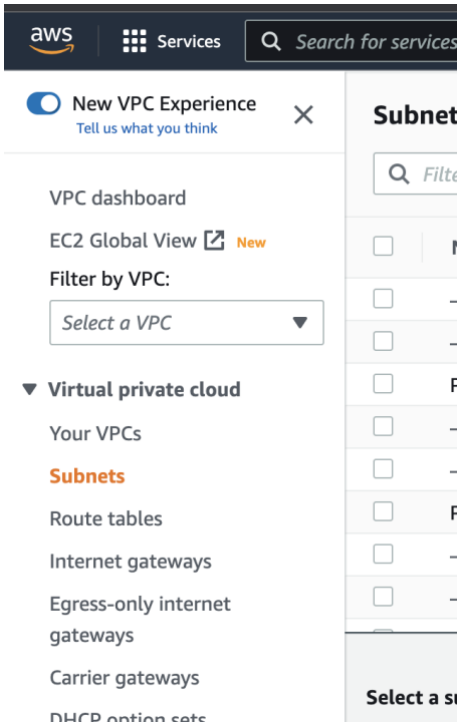
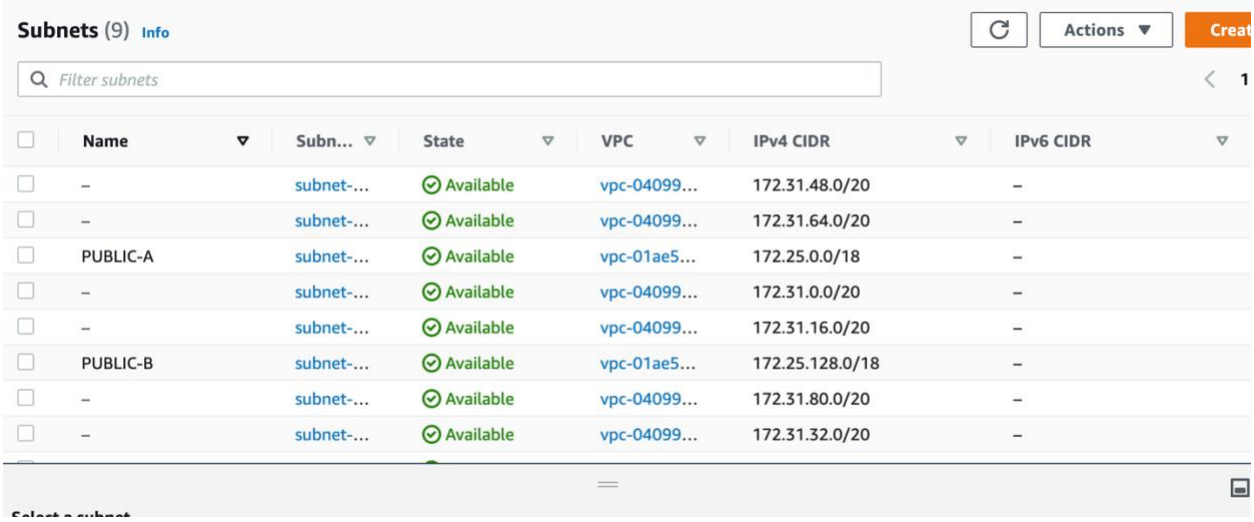
Deployment 3 Documentation

Creating an EC2 in the Public Subnet of my VPC

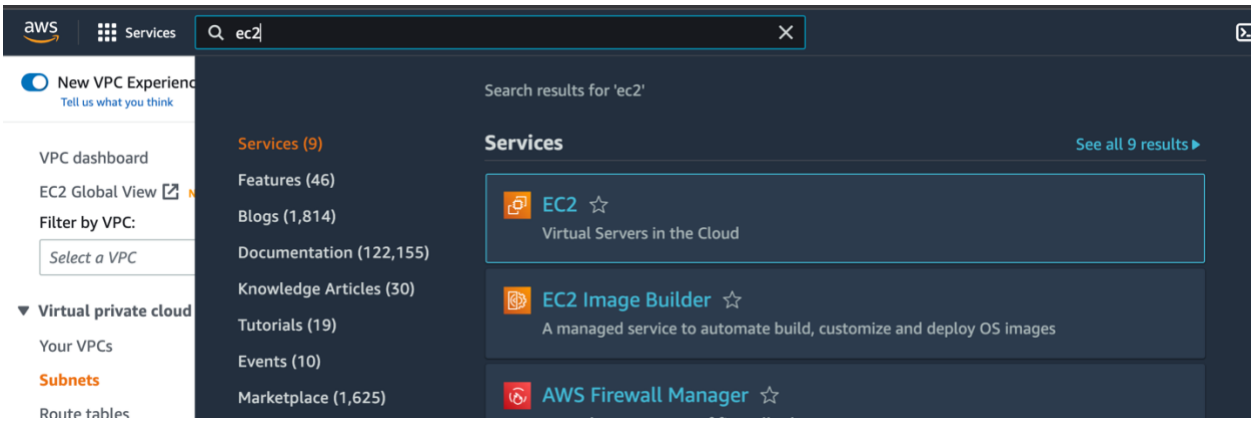
- First, I created a public subnet inside of my VPC (mine was named Kura_VPC) and I named it PUBLIC-A. I also had a default VPC as well, I needed two separate VPCs with the two different subnets attached to them.



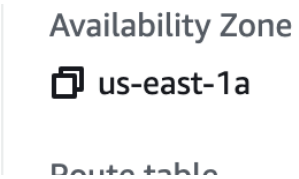


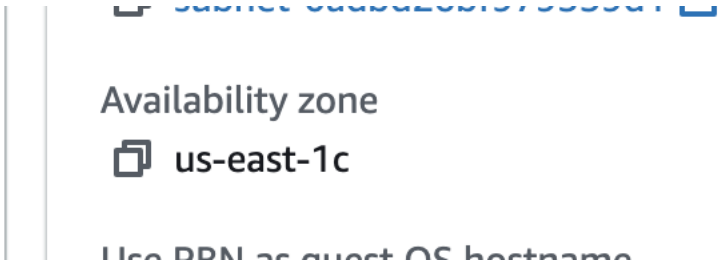


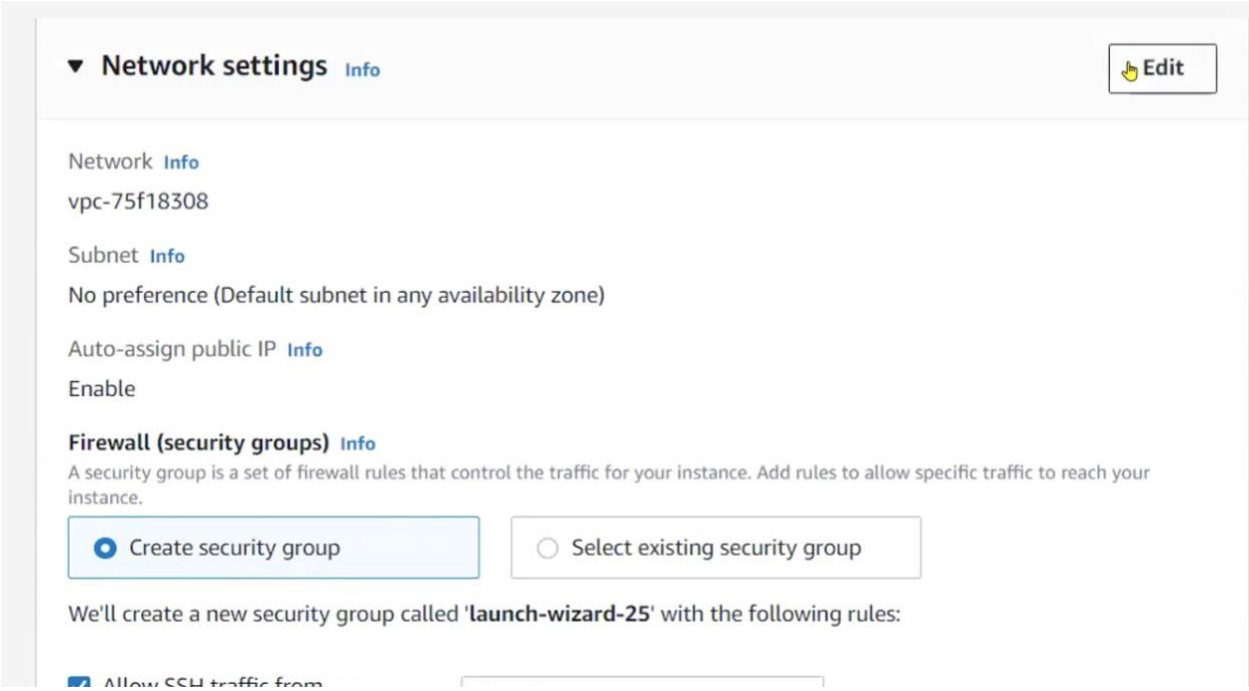
- Then, I created two Ubuntu EC2 instances. I launched my first instance in PUBLIC-A (I used a keypair for both instances). For my Network settings, I chose my VPC, so that's Kura_VPC under the VPC dropdown. Then, I chose PUBLIC-A under the subnet dropdown. I enabled "auto-assign public IP" for both instances because the instance is in a public subnet.



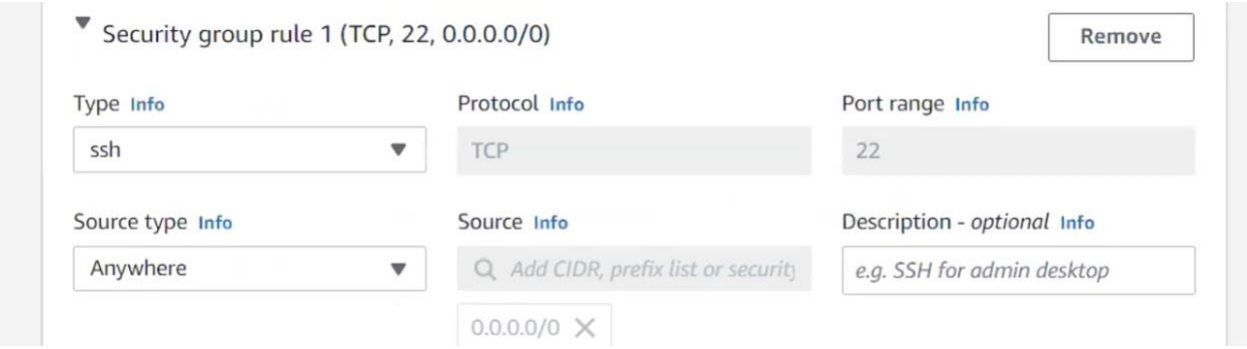
- My first subnet PUBLIC-A has availability zone 1a therefore when I choose this subnet under network settings when I launch my instance, then it should have the same availability zone. It's different for my second one because I left the dropdown alone underneath subnet and allowed for no preference since I have many default subnets and it ended up being in availability zone us-east-1c.

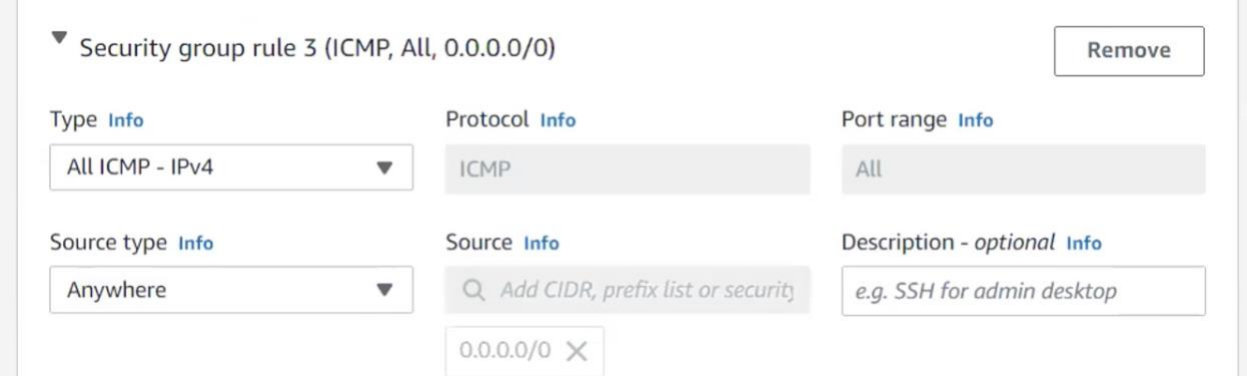


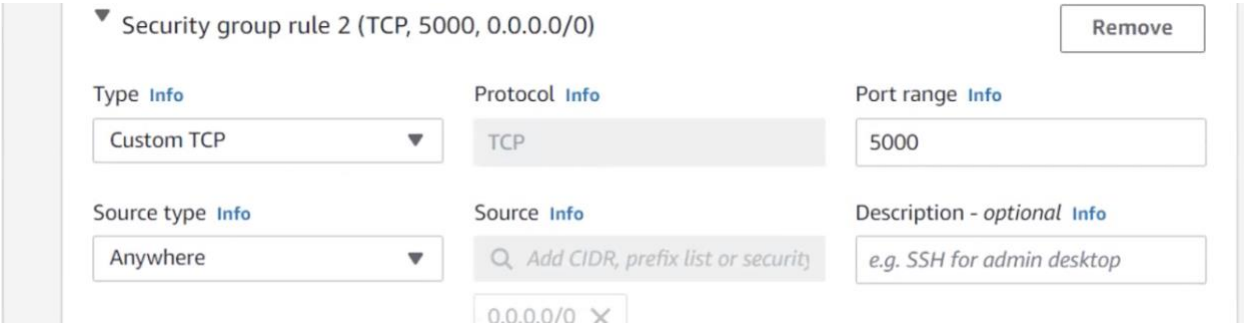




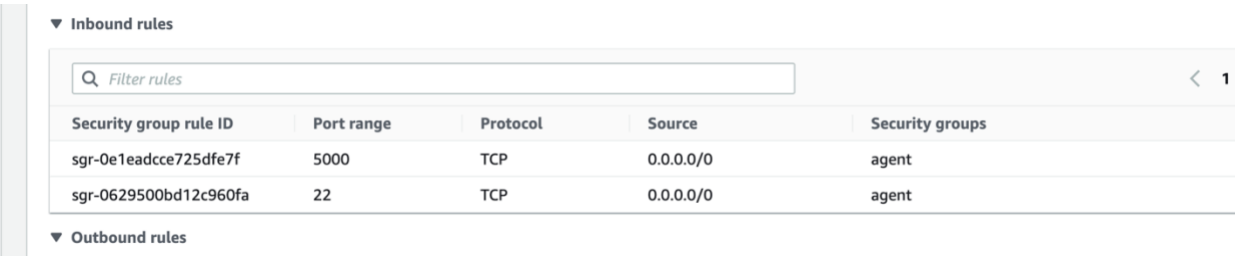
- Underneath security groups, for my first EC2 called EC2-PubA, I created 3 rules with ports 22, 5000, and All for my ICMP port for pinging. When I ran the instance because I enabled auto-assign public IP, my instance now has a public IP assigned to it when I start my machine.



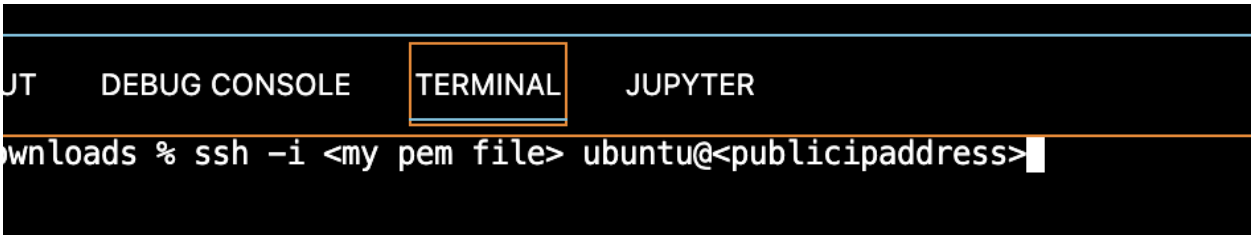




- The same goes for my second instance I made which is my Jenkins agent EC2. When I ran the instance because I enabled auto-assign public IP, my instance now has a public IP assigned to it. I only created 2 security rules with ports 22 and 5000 open.



- At this point, both of my instances are running in AWS. Now in VSCode, I open 1 terminal. I cd into the directory my key is in and then ssh into my instance called EC2-PubA first using the format in the screenshot.



- Now I'm in both of the instances.

```
Ubuntu comes with ABSOLUTELY NO WARR
applicable law.

To run a command as administrator (u
See "man sudo_root" for details.

ubuntu@ip-172-25-5-255:~$
```

```
ubuntu@ip-172-31-86-157:~$
ubuntu@ip-172-31-86-157:~$
```

Install default-jre, python3-pip, python3.10-venv, nginx and Run Jenkins server

- In my first terminal, I created a bash file and set my permissions. I nano the file and copy and paste a script that will automate steps and installations. These steps are: installing Java, installing Jenkins package and its keys, installing pip penv and python3-pip, and installs nginx (although, I learned I only needed nginx on my Jenkins Agent EC2)…

```
/home/ubuntu
ubuntu@ip-172-25-5-255:~$ nano installpj.sh
ubuntu@ip-172-25-5-255:~$ chmod 777 installpj.sh
ubuntu@ip-172-25-5-255:~$
```

- I installed packages default-jre, python3-pip, python3.10-venv and nginx by creating a simple script and running it to automate that process for me instead of installing each package separately and manually. I only did this in my EC2-PubA instance. This is my script:

```
#!/bin/bash

#Adds the Jenkins Keyrings without user interaction/input
wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | gpg --batch --yes --dearmor -o /usr/share/keyrings/jenkins.gpg

#Adds the Jenkins repo to the sources list of apt
sh -c 'echo deb [signed-by=/usr/share/keyrings/jenkins.gpg] http://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list'

#Updates system so there are no descrepencies during installation
apt-get update

#Installs java, Jenkins, pip, python venv, and nginx, answers yes to all prompts
apt-get install default-jre -y
sleep 5
apt-get install jenkins -y
sleep 5
apt-get install python3-pip -y
apt-get install python3.10-venv -y
sleep 5
sudo apt install nginx -y

exit 0
```

- I then ran an <sudo apt update> just to make sure everything is up to date on my end. No errors at the end of installation:

```
No VM guests are running outdated hypervisor (qemu) binaries on this host.
ubuntu@ip-172-25-5-255:~$
```

- I didn't get any errors at the end, only 2 prompts telling me that python3-pip and python3.10-venv were already installed of course. Other than that, the entire script ran all the way through. If I check the status of Jenkins and Nginx, they should be running.

```
ubuntu@ip-172-25-5-255:~$ sudo systemctl status jenkins
● jenkins.service - Jenkins Continuous Integration Server
     Loaded: loaded (/lib/systemd/system/jenkins.service; enable
     Active: active (running) since Sat 2022-10-15 03:01:52 UTC;
   Main PID: 5926 (java)
      Tasks: 36 (limit: 1143)
```

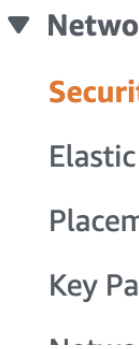
```
ubuntu@ip-172-25-5-255:~$ sudo systemctl status nginx
● nginx.service - A high performance web server and a reverse
     Loaded: loaded (/lib/systemd/system/nginx.service; enable
     Active: active (running) since Sat 2022-10-15 03:02:13 UT
       Docs: man:nginx(8)
```

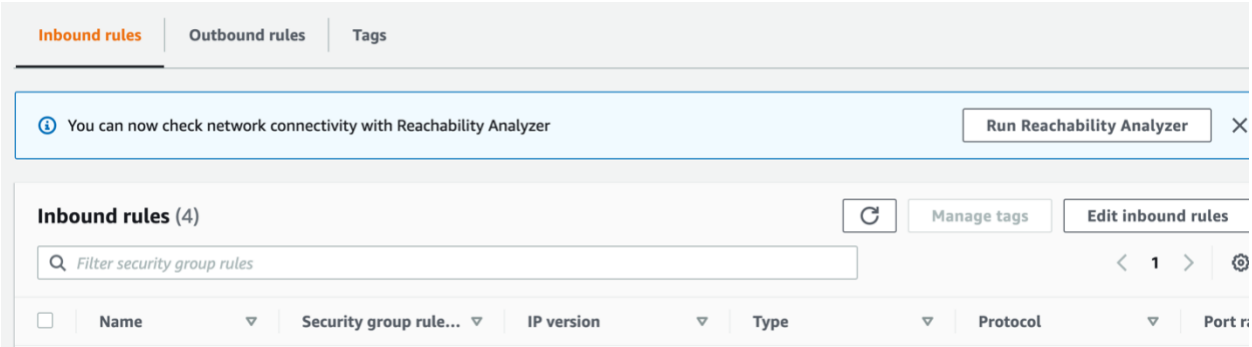
- My Jenkins ended up not loading because its on port 8080 and I didn't include port 8080 in my security group

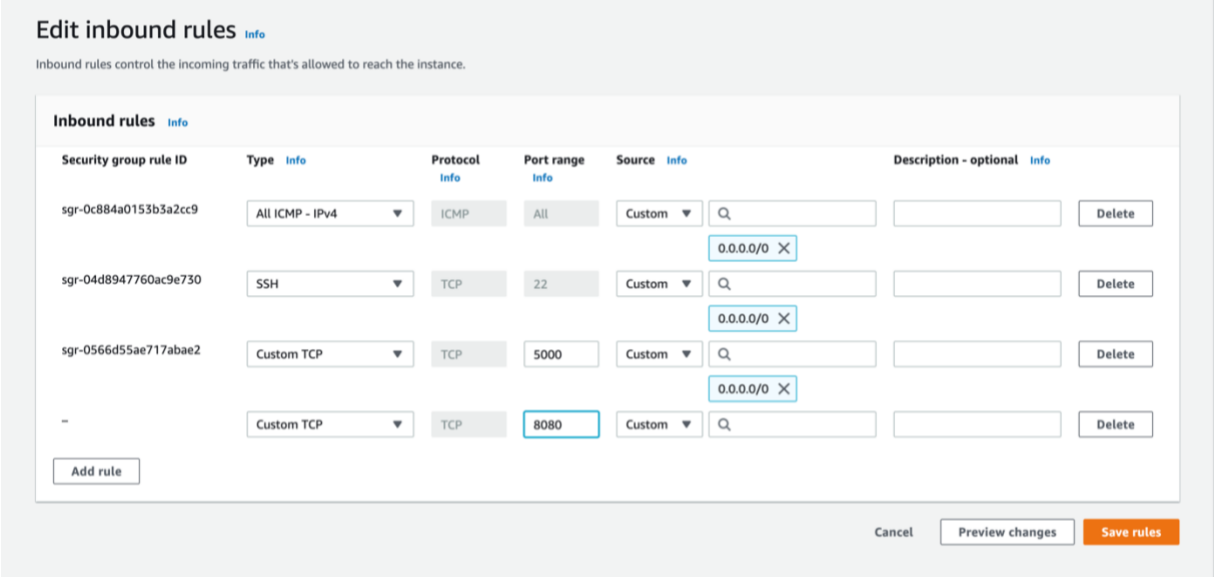
```
ubuntu@ip-172-25-5-255:~$ sudo systemctl status jenkins
● jenkins.service - Jenkins Continuous Integration Server
     Loaded: loaded (/lib/systemd/system/jenkins.service; enabled; vendor preset: enabled)
     Active: active (running) since Sat 2022-10-15 03:01:52 UTC; 47min ago
   Main PID: 5926 (java)
      Tasks: 36 (limit: 1143)
     Memory: 296.0M
        CPU: 45.044s
     CGroup: /system.slice/jenkins.service
             └─5926 /usr/bin/java -Djava.awt.headless=true -jar /usr/share/java/jenkins.war --webroot=/var/cache/jenkins/war --httpPort=8080
```

```
--httpPort=8080
```

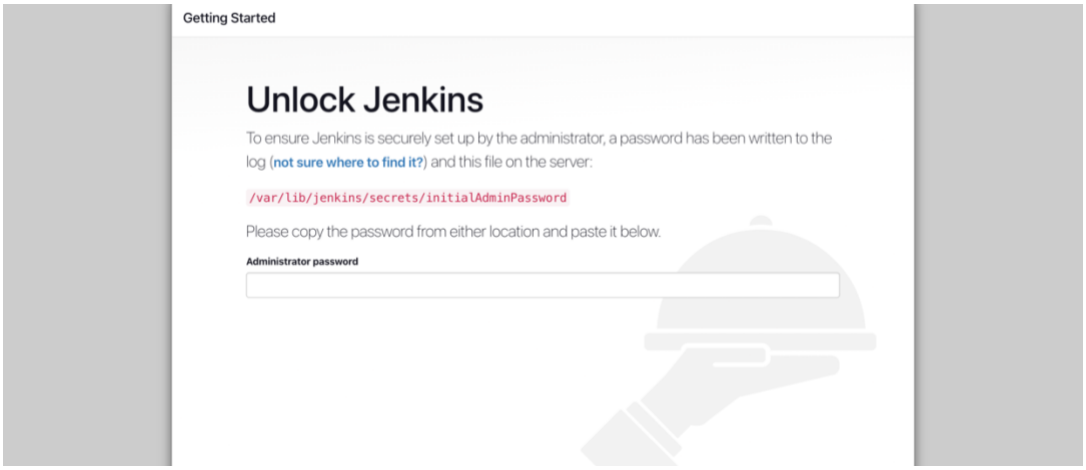
- I went back to my instance security group settings and edit the inbound rules. I included port 8080.







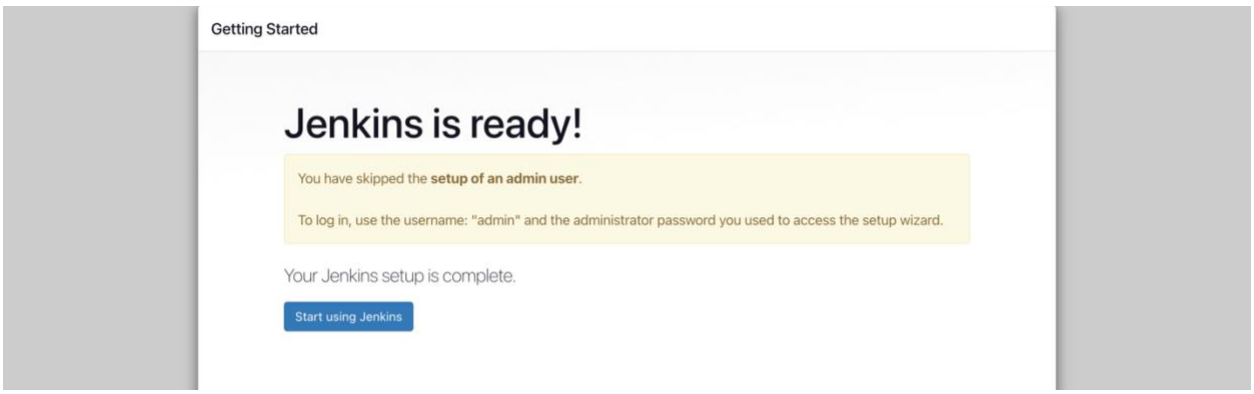
- So when I input my public ip address and port 8080 into a blank search bar, Jenkins pops up. To find my admin password, like the screenshot says, I just cat out that path in my terminal.



- And sure enough it popped up.

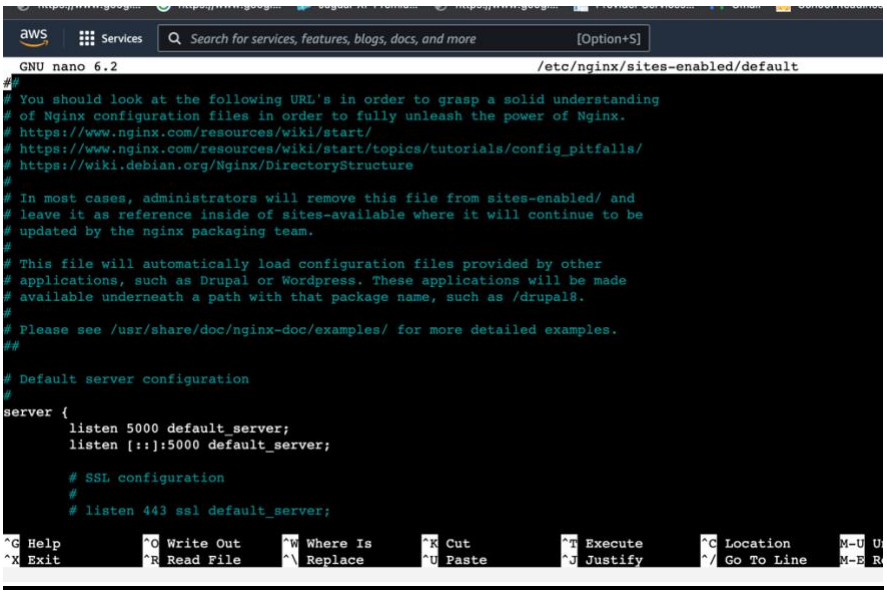
```
lines 1-20/20 (END)
ubuntu@ip-172-25-5-255:~$ sudo cat /var/lib/jenkins/secrets/initialAdminPassword
8f5d0621eaad48289[redacted]
```

- After placing the admin password in I skipped making a custom user and created an admin user. Now Jenkins is ready to use.

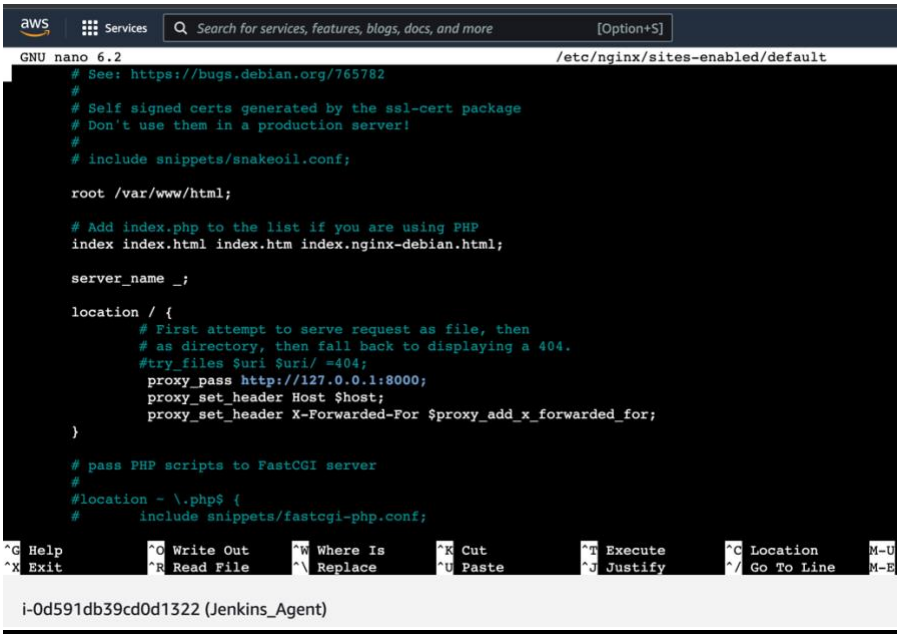


Configure Nginx

- Now that Jenkins is ready, before I actually build the pipeline, I created a default file via the path=</etc/nginx/sites-enabled/default> by running <sudo nano /etc/nginx/sites enabled/default> from my Jenkins Agent EC2 terminal. I changed the ports which were “80” to “5000”.



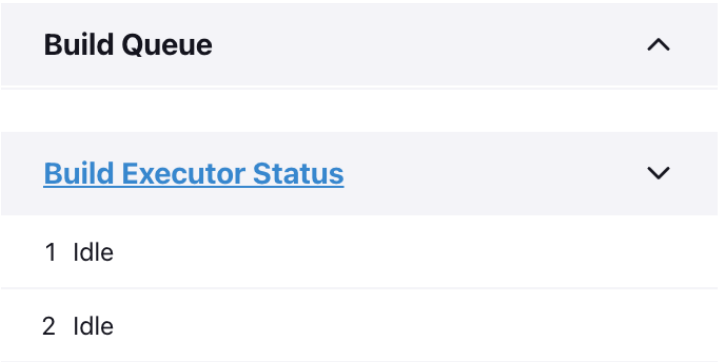
- Listen tells NGINX the hostname/IP and the TCP port where it should listen for HTTP connections and location covers requests for specific files and folders.



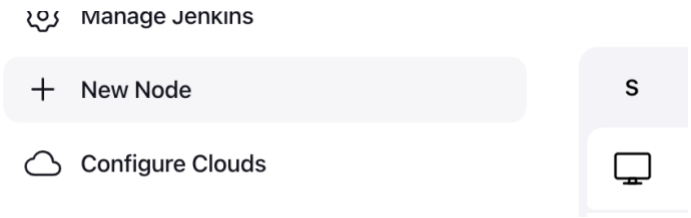
- Lastly, I restarted ngnix by running the <sudo systemctl restart ngnix> command

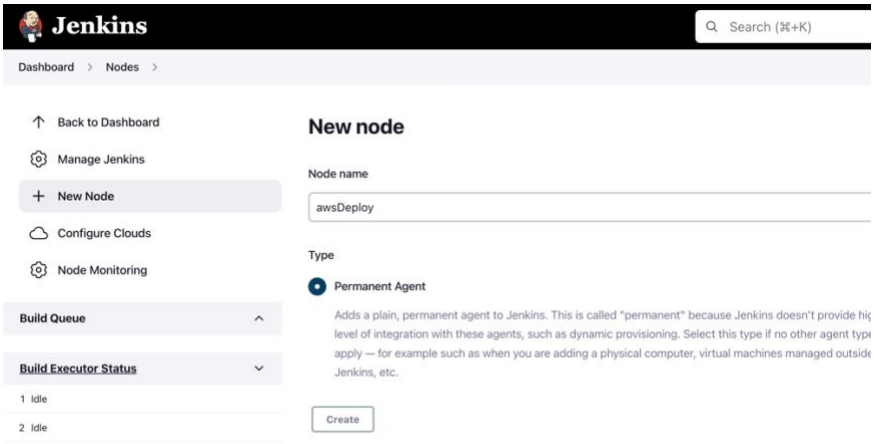
Configure and connect a Jenkins agent to Jenkins

- In this stage, I created my Jenkins agent manually in Jenkins by selecting build executor status from the lower left-hand side.



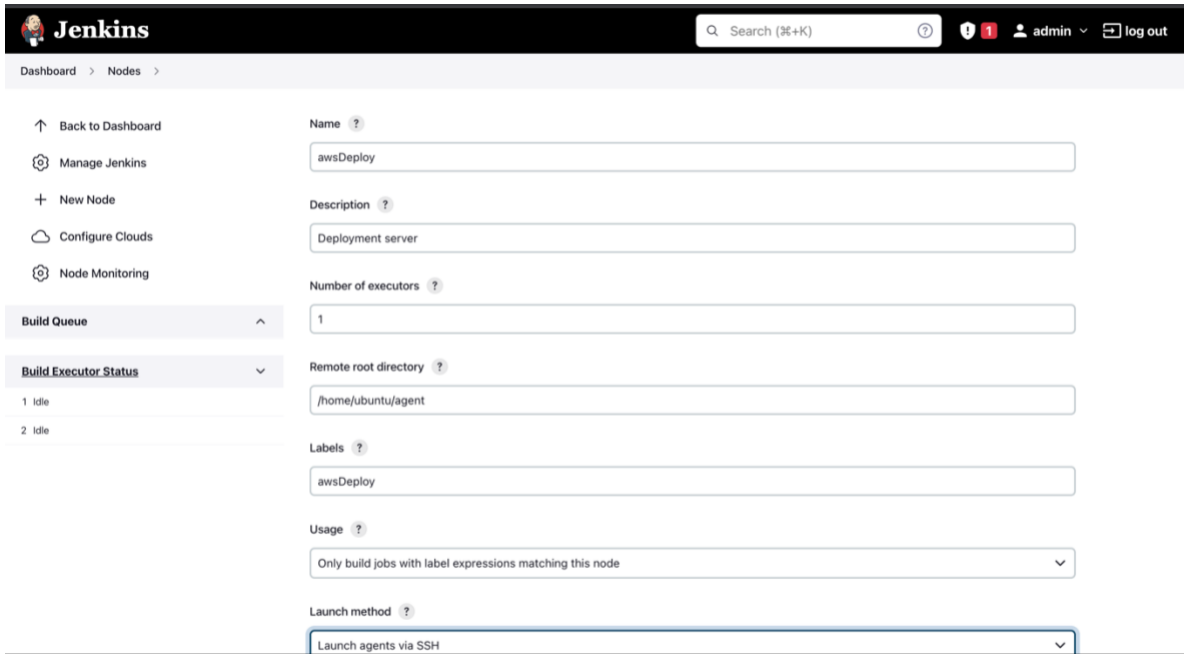
- Next I selected “New Node” to configure and add the Jenkins agent. My node name is “awsDeploy” and I selected “Permanent Agent” and then create.



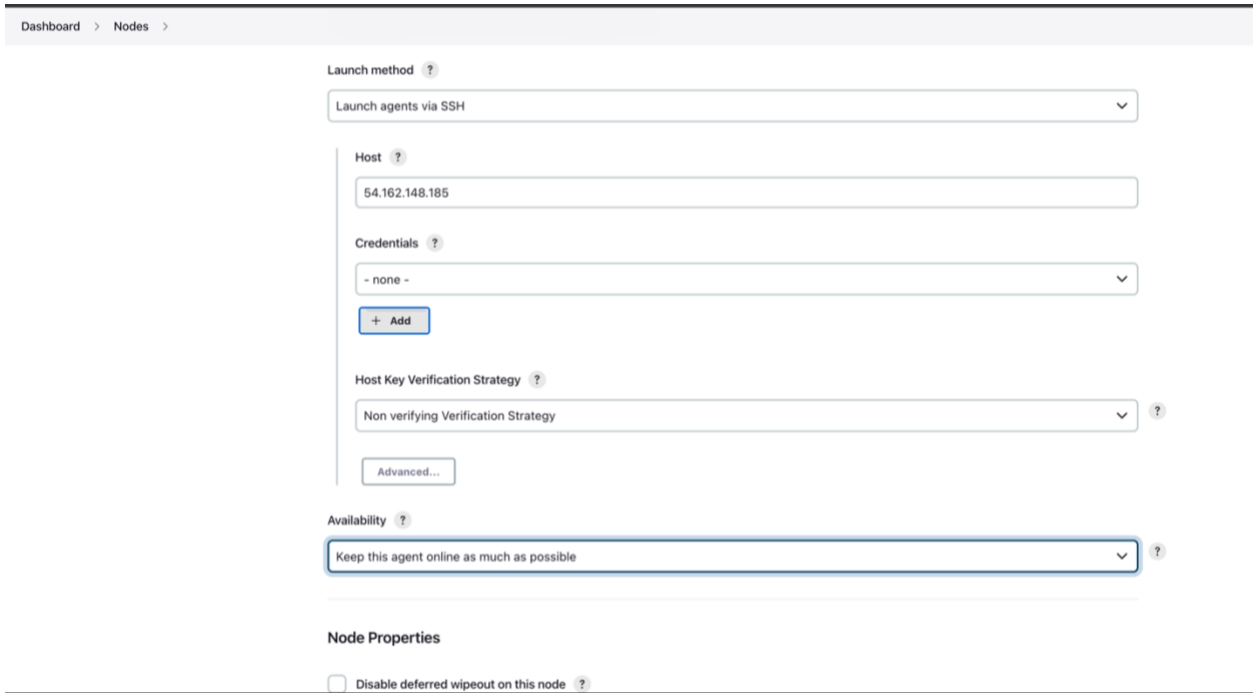


- When configuring the agent, I input these settings.

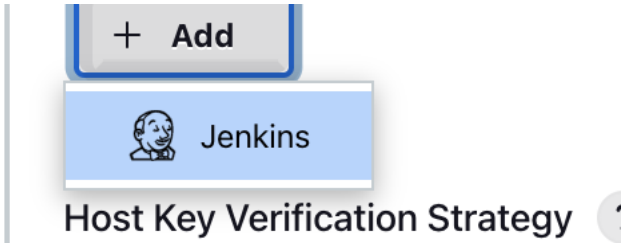
- Name: **awsDeploy**
- Description: **Deployment server**
- Number of executors: **1**
- Remote root directory: **/home/ubuntu/agent**
- Labels: **aweDeploy**
- Usage: **only build jobs with label….**
- Launch method: **launch agents via ssh**
- Host: {Enter the public IP of your EC2 in the Public subnet and not this text}
- **Credentials: see below**
- Host key verification strategy: **non verifying verification strategy**



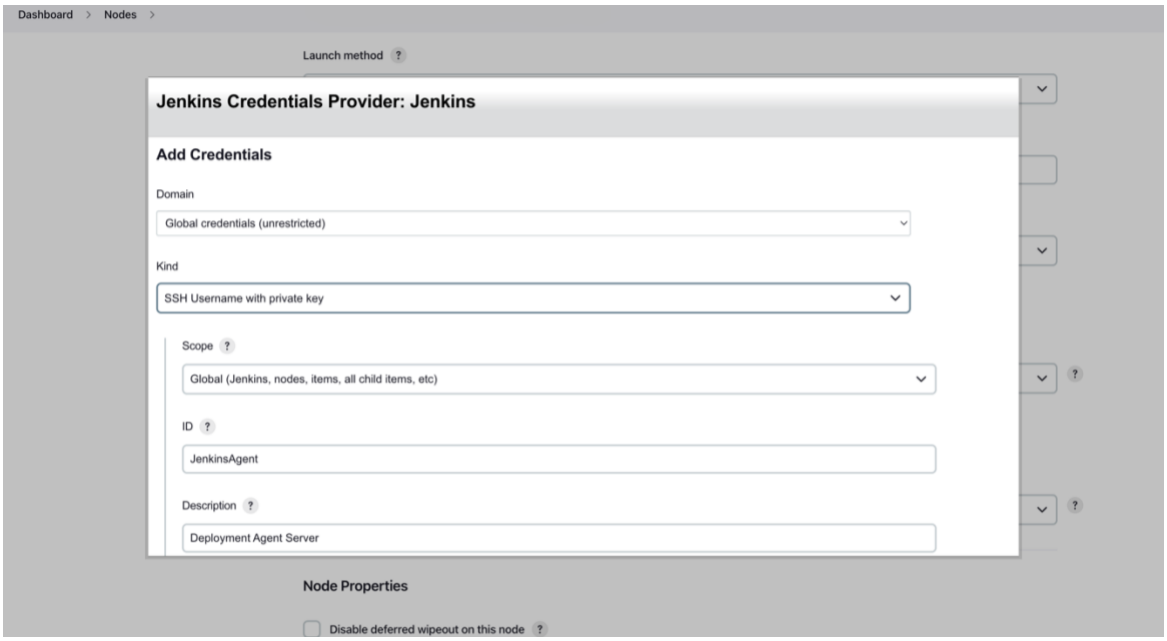
- When it came time to select the launch method I chose the launch agents via ssh option, I pasted my public IP of my Jenkins Agent EC2 in the “host” box.

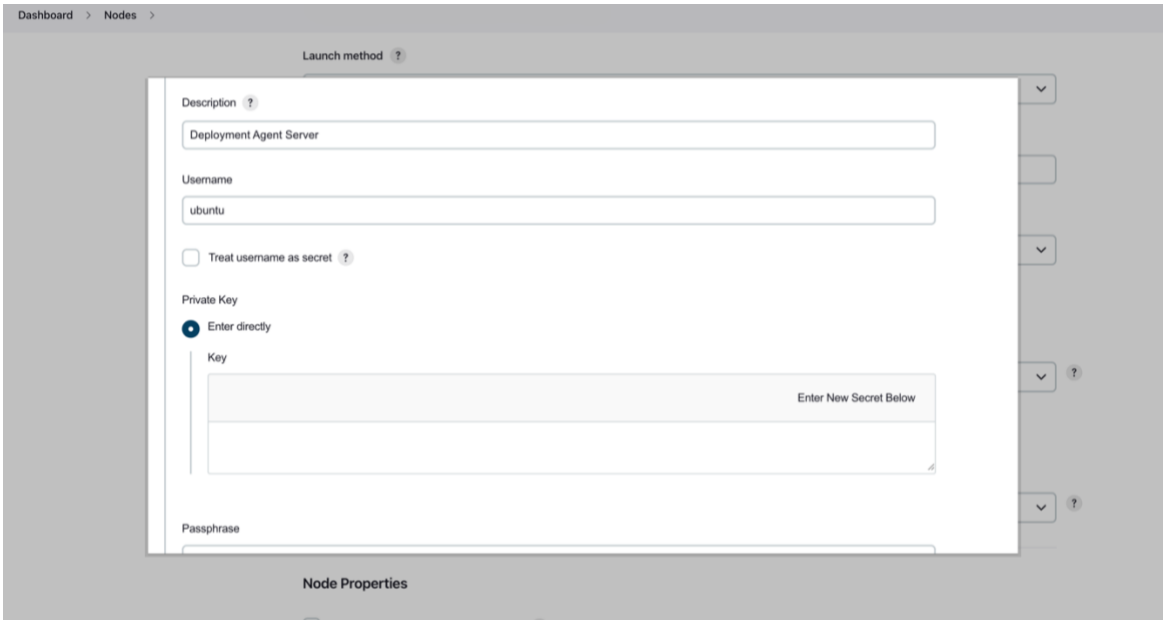


- I added credentials and chose Jenkins option.

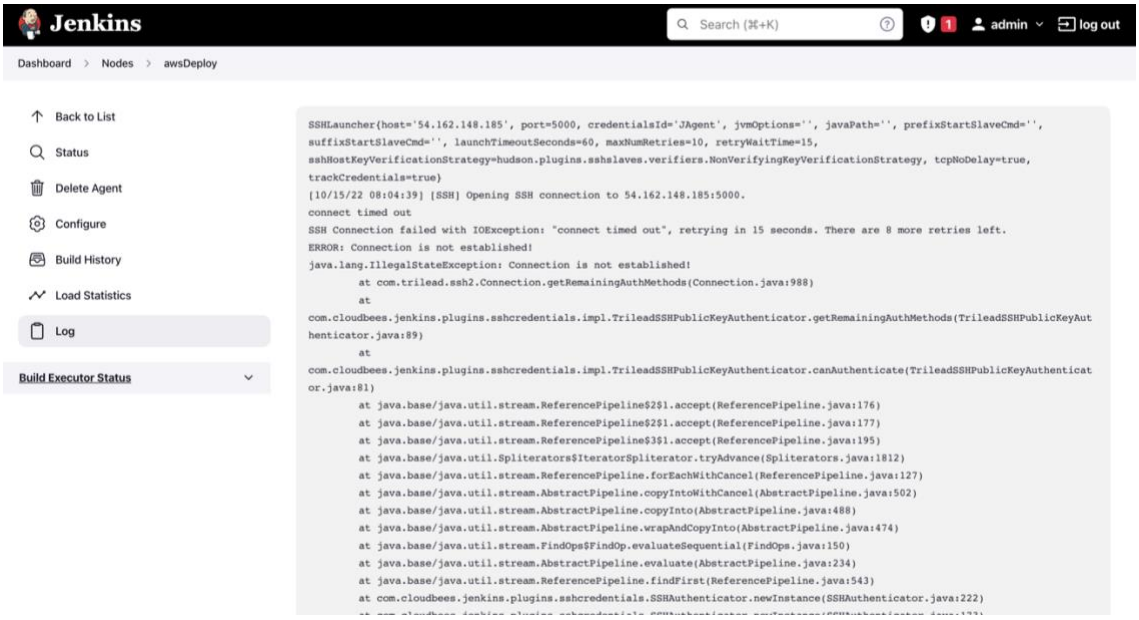


- I applied these settings displayed in the screenshot below and when it came time to enter the private key directly I pasted my key from my /home/Ubuntu/.ssh/authorized_keys file which held my pem file into the box.





- Lastly, for “availability” I chose the keep this agent online as much as possible option. I saved the configurations and waited for the agent to launch.



- What ended up happening was that my agent failed to launch. After hours of playing around with the configurations, I found out that my private key was written in the OpenSSH format private keys by default instead of using OpenSSL's PEM format. Jenkins doesn't support the OpenSSH format so of course the connection couldn't be established.

```
-----BEGIN OPENSSH PRIVATE KEY-----
$ ssh-keygen  -f blah.key -m PEM -p
```

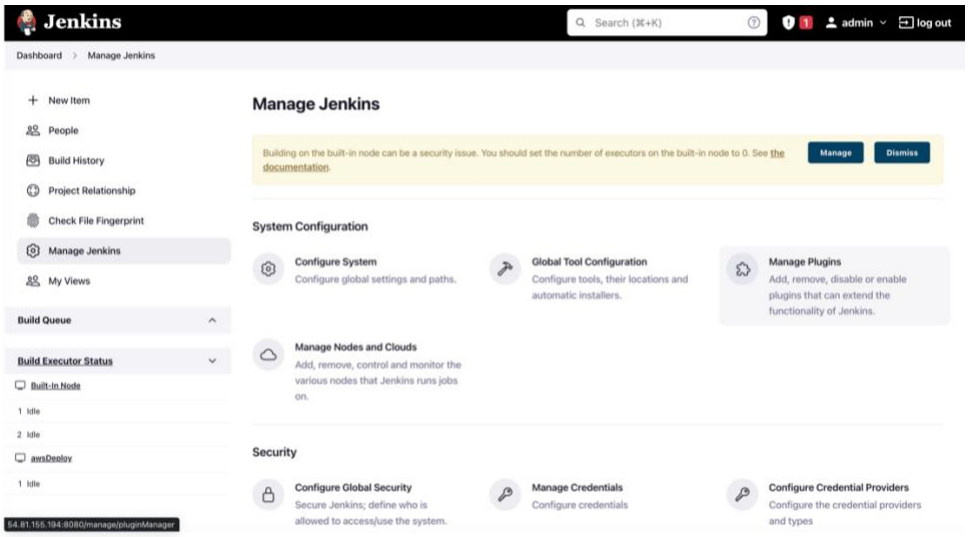
- This was the correct format which I found by opening my downloaded pem file in Sublime text editor, copying and pasting the key again in the "private key" box on Jenkins. Helpful thread on StackOverflow regarding this topic: https://stackoverflow.com/questions/53636532/jenkins-what-is-the-correct-format-for-private-key-in-credentials

> ```
> Private key: <Enter directly>
>  -----BEGIN RSA PRIVATE KEY-----
>  ......
>  -----END RSA PRIVATE KEY-----
> ```

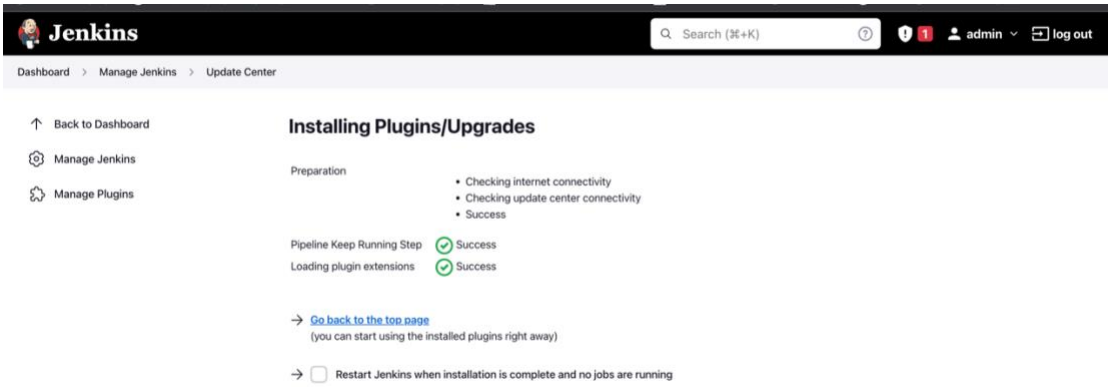
- I launched the agent again and a connection was established on port 5000. Eureka!

```
Launcher: SSHLauncher
Communication Protocol: Standard in/out
This is a Unix agent
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by jenkins.slaves.StandardOutputSwapper$ChannelSwapper to constructo
java.io.FileDescriptor(int)
WARNING: Please consider reporting this to the maintainers of jenkins.slaves.StandardOutputSwapper$Chan
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
Evacuated stdout
Agent successfully connected and online
```

- I downloaded the Jenkins plugin “Pipeline Keep Running” by selecting “manage Jenkins” from the left side of the screen >> manage plugins >> typing Pipeline Keep Running in the empty text box >> selecting the plugin >> installing it. Two check marks means it was installed successfully.



- Two check marks means it was installed successfully.



Edit Jenkins file in Github Repo and Build CI/CD Pipeline

- I edited my Jenkinsfile in the Github repository by adding a Clean and Deploy stage after "junit 'test-reports/results.xml," line.

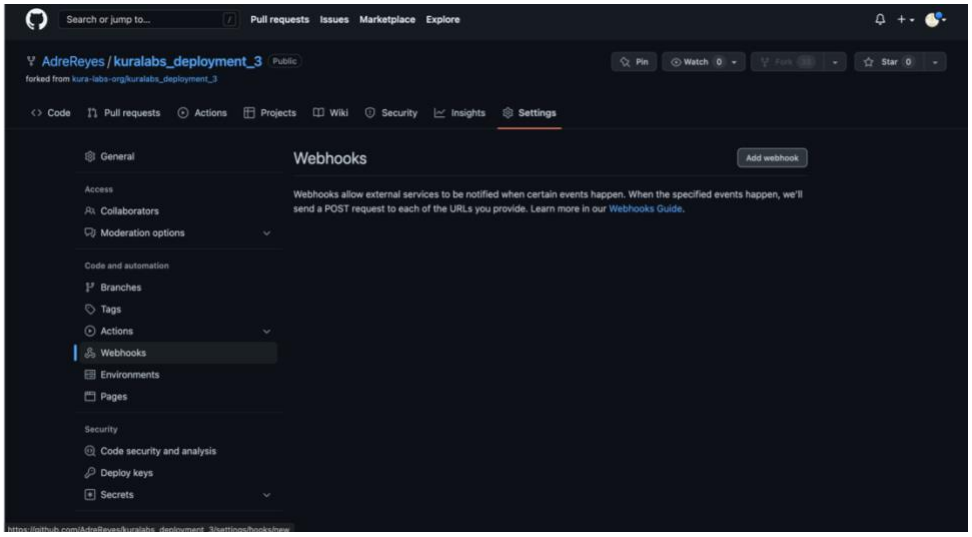
50 lines (49 sloc) | 1.12 KB

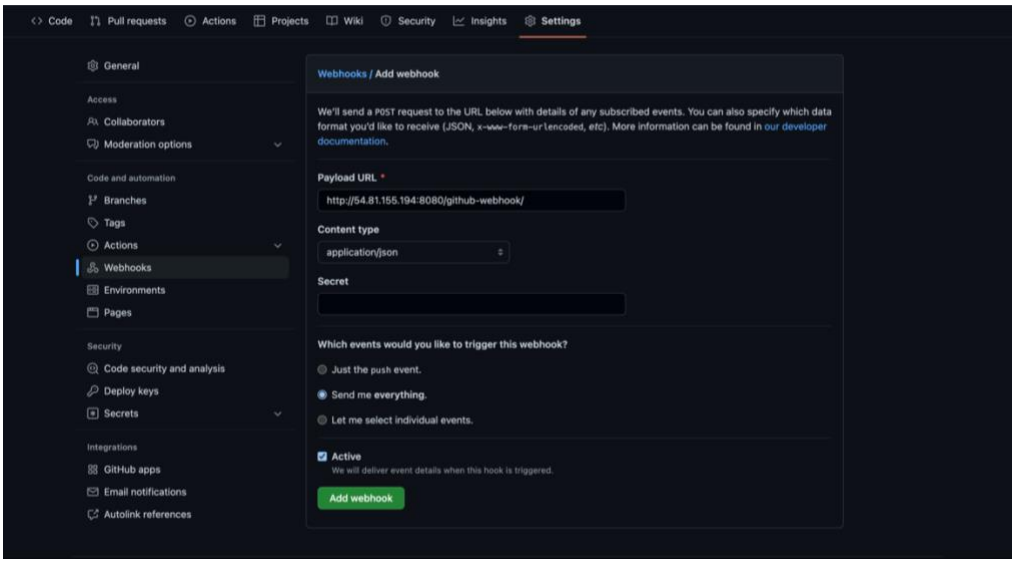
```
pipeline {
  agent any
   stages {
    stage ('Build') {
      steps {
        sh '''#!/bin/bash
        python3 -m venv test3
        source test3/bin/activate
        pip install pip --upgrade
        pip install -r requirements.txt
        export FLASK_APP=application
        flask run &
        '''
} }
    stage ('test') {
      steps {
        sh '''#!/bin/bash
        source test3/bin/activate
        py.test --verbose --junit-xml test-reports/results.xml
        '''
}
      post{
        always {
          junit 'test-reports/results.xml'
        }
  }
```

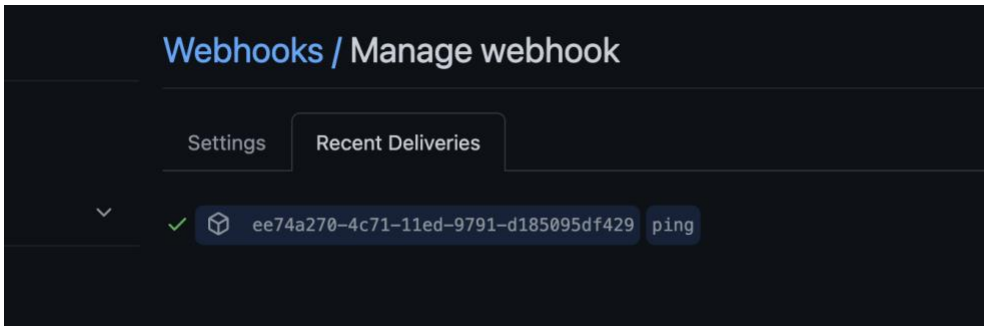
```
  }
}

  stage ('Clean') {
     agent{label 'awsDeploy'}
     steps {
sh '''#!/bin/bash
if[[$(psaux|grep-i"gunicorn"|tr-s""|head-n1|cut -d""-f2)!=0]] then
psaux|grep-i"gunicorn"|tr-s""|head-n1|cut -d""-f2>pid.txt kill $(cat pid.txt)
exit 0
fi
''' }
    }
   stage ('Deploy') {
     agent{label 'awsDeploy'}
     steps {
     keepRunning {
       sh '''#!/bin/bash
       pip install -r requirements.txt
       pip install gunicorn
       python3 -m gunicorn -w 4 application:app -b 0.0.0.0 --daemon
       '''
} }
} }
}
```

- Next, I added a webhook so that the activity in my Github repo will trigger and match up with the Jenkins builds and I can keep track of the activity in Github. I changed the Payload URL to the url my Jenkins runs on which is <https://54.81.155.194:8080>.

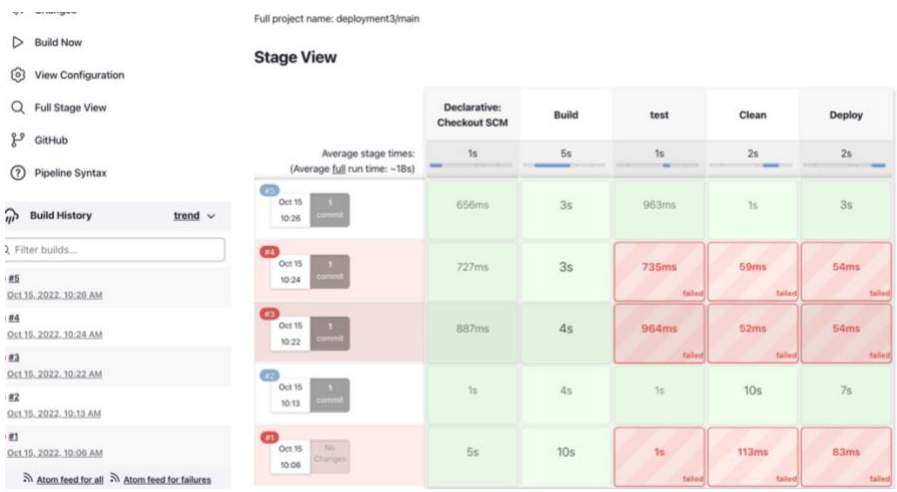






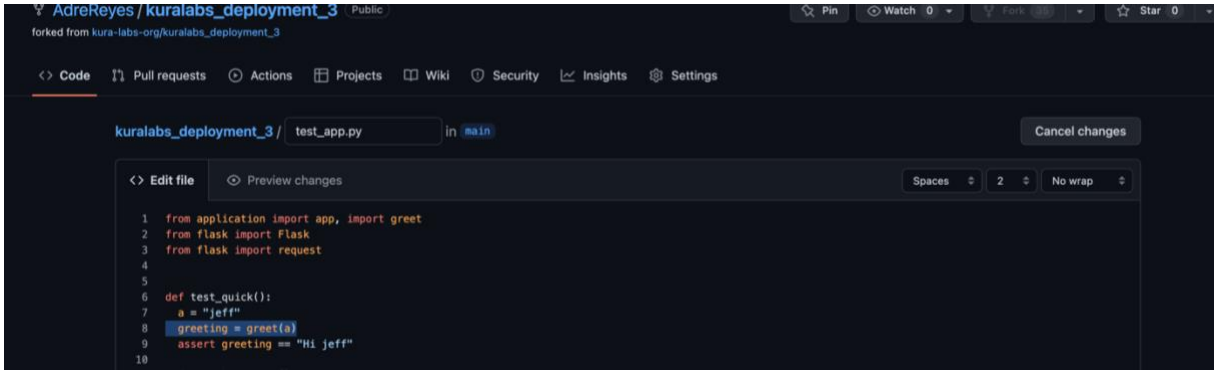
Build CI/CD Pipeline

- So, my first build failed and that was before I edited my test file so after correcting the script the second build was successful.

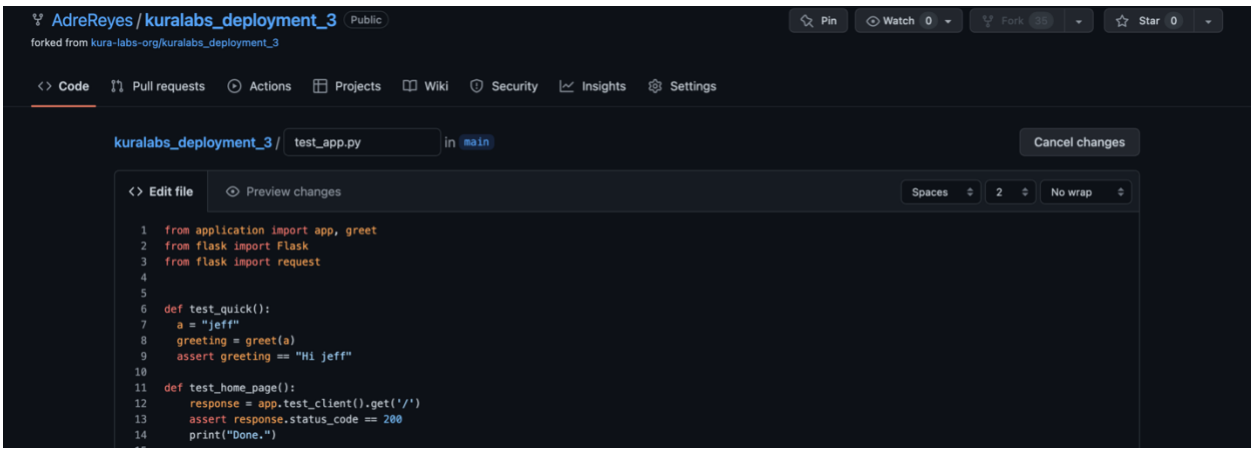


- However, the third build failed and the error I received was from my test_app.py file. The error occurred because I added code with incorrect syntax from the first "test_quick()".

- I thought I fixed everything, but my build failed again (for the fourth time) because of the small "import greet" code mistake which is incorrect. Greet is not a module so I deleted "import".



- I also added the test from Deployment 2 which tests the response of the homepage of the Flask application.



- I built the application again and it resulted in a successful build. 😎