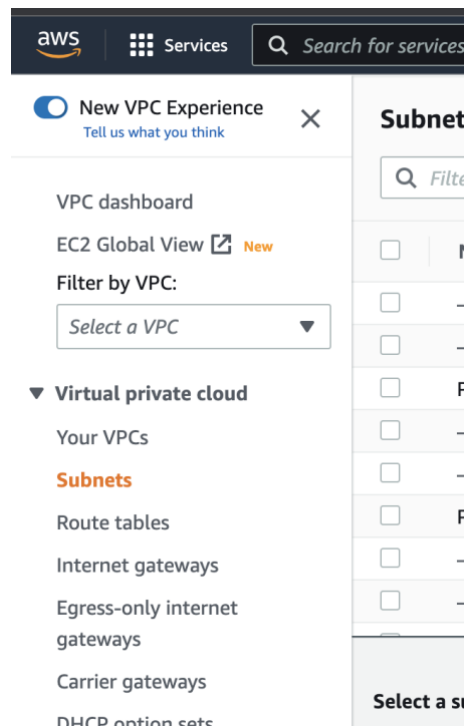
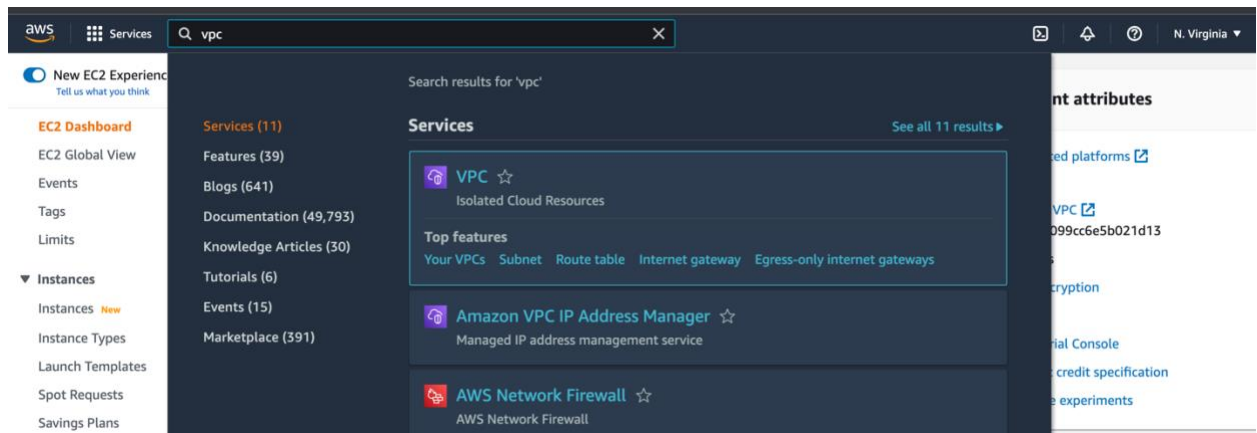


# Deployment 3 Documentation

## 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.



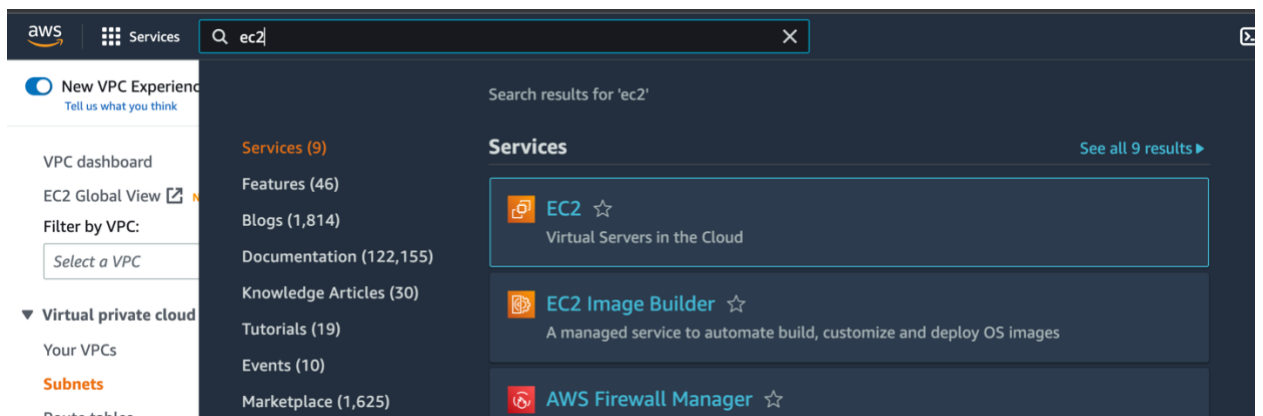
Subnets (9) [Info](#)

Filter subnets

<input type="checkbox"/>	Name	Subn...	State	VPC	IPv4 CIDR	IPv6 CIDR
<input type="checkbox"/>	-	subnet-...	Available	vpc-04099...	172.31.48.0/20	-
<input type="checkbox"/>	-	subnet-...	Available	vpc-04099...	172.31.64.0/20	-
<input type="checkbox"/>	PUBLIC-A	subnet-...	Available	vpc-01ae5...	172.25.0.0/18	-
<input type="checkbox"/>	-	subnet-...	Available	vpc-04099...	172.31.0.0/20	-
<input type="checkbox"/>	-	subnet-...	Available	vpc-04099...	172.31.16.0/20	-
<input type="checkbox"/>	PUBLIC-B	subnet-...	Available	vpc-01ae5...	172.25.128.0/18	-
<input type="checkbox"/>	-	subnet-...	Available	vpc-04099...	172.31.80.0/20	-
<input type="checkbox"/>	-	subnet-...	Available	vpc-04099...	172.31.32.0/20	-

Select a subnet

- 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.

## Availability Zone

 us-east-1a

Route table

## Availability zone

 us-east-1c

Use PBN as guest OS hostname

▼ Network settings [Info](#)

Edit

Network [Info](#)

vpc-75f18308

Subnet [Info](#)

No preference (Default subnet in any availability zone)

Auto-assign public IP [Info](#)

Enable


Firewall (security groups) [Info](#)

A security group is a set of firewall rules that control the traffic for your instance. Add rules to allow specific traffic to reach your instance.

☒ Create security group

☐ Select existing security group

We'll create a new security group called 'launch-wizard-25' with the following rules:

 Allow SSH traffic from

- 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.

▼ Security group rule 1 (TCP, 22, 0.0.0.0/0)

Remove

Type [Info](#)

ssh

Protocol [Info](#)

TCP


Port range [Info](#)

22

Source type [Info](#)

Anywhere

Source [Info](#)

 Add CIDR, prefix list or security group

0.0.0.0/0 X

Description - optional [Info](#)

e.g. SSH for admin desktop

▼ Security group rule 3 (ICMP, All, 0.0.0.0/0)

Remove

Type [Info](#)

All ICMP - IPv4 ▼

Protocol [Info](#)

ICMP

Port range [Info](#)

All

Source type [Info](#)

Anywhere ▼

Source [Info](#)

Description - optional [Info](#)

e.g. SSH for admin desktop

▼ Security group rule 2 (TCP, 5000, 0.0.0.0/0)

Remove

Type [Info](#)

Custom TCP ▼

Protocol [Info](#)

TCP

Port range [Info](#)

5000

Source type [Info](#)

Anywhere ▼

Source [Info](#)

Description - optional [Info](#)

e.g. SSH for admin desktop

- 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.

▼ Inbound rules

< 1

Security group rule ID	Port range	Protocol	Source	Security groups
sgr-0e1eadcce725dfe7f	5000	TCP	0.0.0.0/0	agent
sgr-0629500bd12c960fa	22	TCP	0.0.0.0/0	agent

▼ Outbound rules

- 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.

JT
DEBUG CONSOLE

TERMINAL

JUPYTER

```

downloads % ssh -i <my pem file> ubuntu@<publicipaddress>

```

- Now I'm in both of the instances.

```
Ubuntu comes with ABSOLUTELY NO WARRANTY.  
no applicable law.  
  
To run a command as administrator (user root),  
you must use the 'sudo' command. See "man sudo_root"  
for details.  
  
ubuntu@ip-172-25-5-255:~$
```

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

```
ubuntu@ip-172-25-5-255:~$ pwd  
/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 discrepancies 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.

## ▼ Network & Security

### Security Groups

Elastic IPs

Placement Groups

Key Pairs

Network interfaces

Inbound rulesOutbound rulesTags

You can now check network connectivity with Reachability Analyzer

Run Reachability Analyzer

Inbound rules (4)

Manage tagsEdit inbound rules

Filter security group rules

< 1 >

	Name	Security group rule...	IP version	Type	Protocol	Port range
--	------	------------------------	------------	------	----------	------------

Edit inbound rules

Inbound rules control the incoming traffic that's allowed to reach the instance.

Inbound rules

Security group rule ID	Type	Protocol	Port range	Source	Description - optional
sgr-0c884a0153b3a2cc9	All ICMP - IPv4	ICMP	All	Custom	
sgr-04d8947760ac9e730	SSH	TCP	22	Custom	
sgr-0566d55ae717abae2	Custom TCP	TCP	5000	Custom	
-	Custom TCP	TCP	8080	Custom	

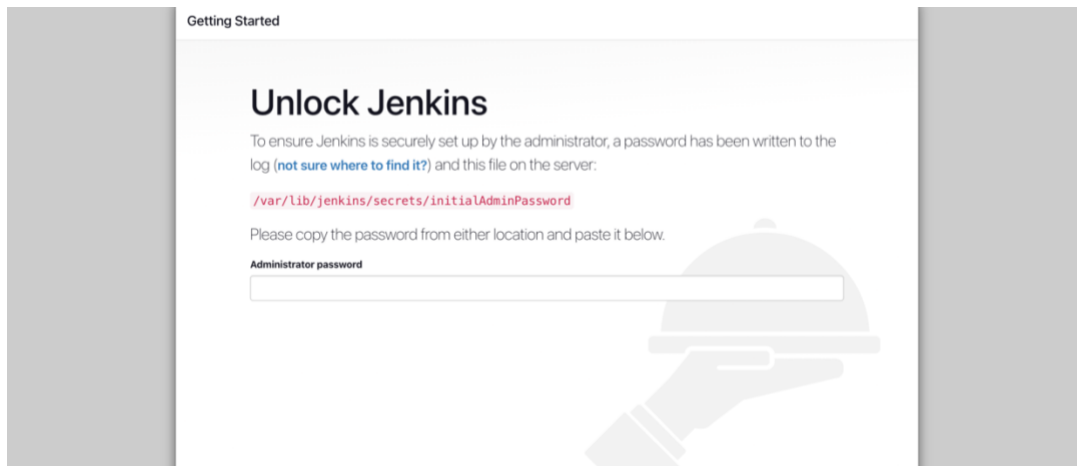
Add rule

Cancel

Preview changes

Save rules

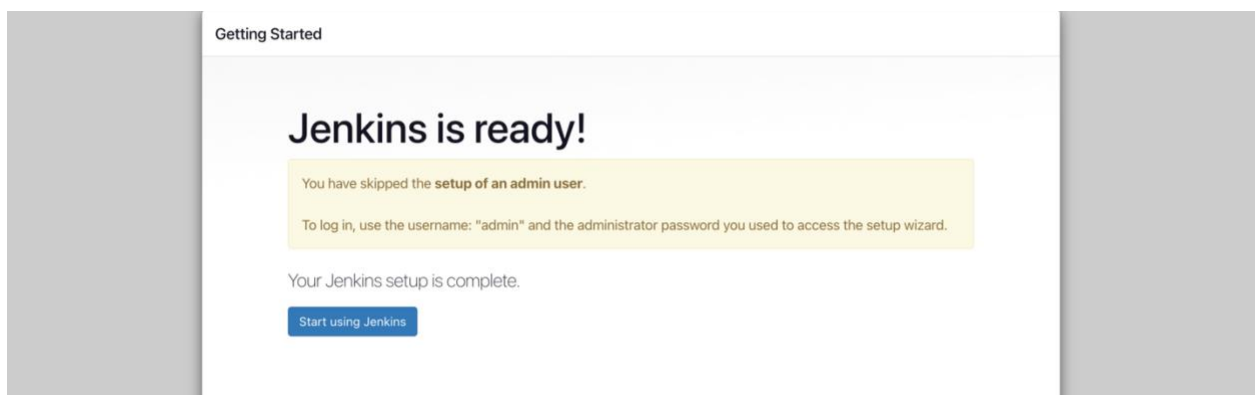
- So when I input my public ip address and port 8080 into a blank search bar, Jenkins pops up. To find my admin password, like the screenshot says, I just cat out that path in my terminal.



- 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
```

- 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”.



```

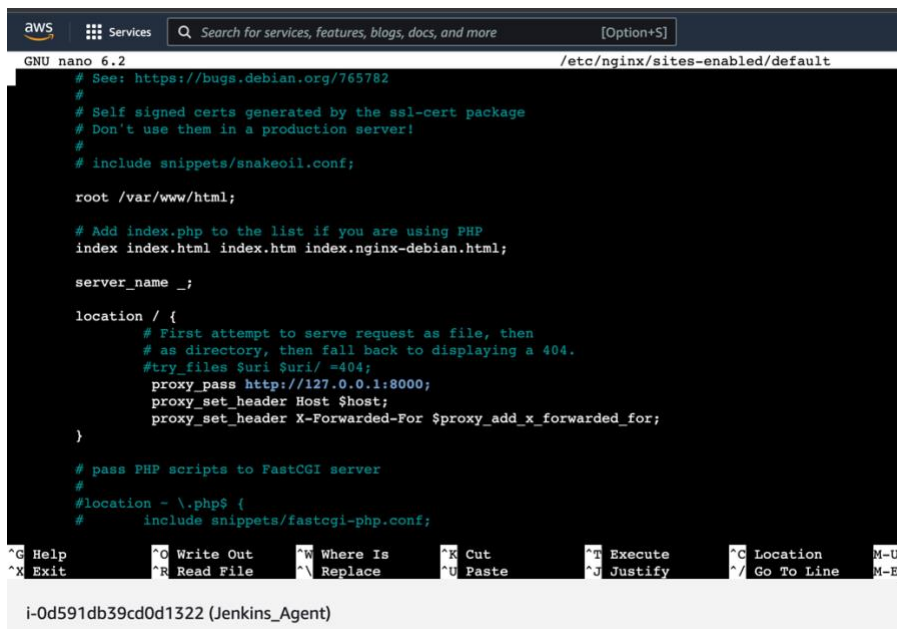
GNU nano 6.2 /etc/nginx/sites-enabled/default
#
# You should look at the following URL's in order to grasp a solid understanding
# of Nginx configuration files in order to fully unleash the power of Nginx.
# https://www.nginx.com/resources/wiki/start/
# https://www.nginx.com/resources/wiki/start/topics/tutorials/config_pitfalls/
# https://wiki.debian.org/Nginx/DirectoryStructure
#
# In most cases, administrators will remove this file from sites-enabled/ and
# leave it as reference inside of sites-available where it will continue to be
# updated by the nginx packaging team.
#
# This file will automatically load configuration files provided by other
# applications, such as Drupal or Wordpress. These applications will be made
# available underneath a path with that package name, such as /drupal8.
#
# Please see /usr/share/doc/nginx-doc/examples/ for more detailed examples.
##

# Default server configuration
#
server {
    listen 5000 default_server;
    listen [::]:5000 default_server;

    # SSL configuration
    #
    # listen 443 ssl default_server;
}

```

- 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.



```

GNU nano 6.2 /etc/nginx/sites-enabled/default
# See: https://bugs.debian.org/765782
#
# Self signed certs generated by the ssl-cert package
# Don't use them in a production server!
#
# include snippets/snakeoil.conf;

root /var/www/html;

# Add index.php to the list if you are using PHP
index index.html index.htm index.nginx-debian.html;

server_name _;

location / {
    # First attempt to serve request as file, then
    # as directory, then fall back to displaying a 404.
    #try_files $uri $uri/ =404;
    proxy_pass http://127.0.0.1:8000;
    proxy_set_header Host $host;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
}

# pass PHP scripts to FastCGI server
#
#location ~ \.php$ {
#    include snippets/fastcgi-php.conf;
}

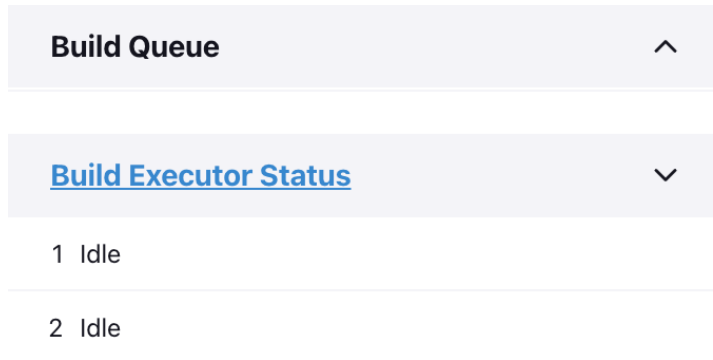
```

i-0d591db39cd0d1322 (Jenkins\_Agent)

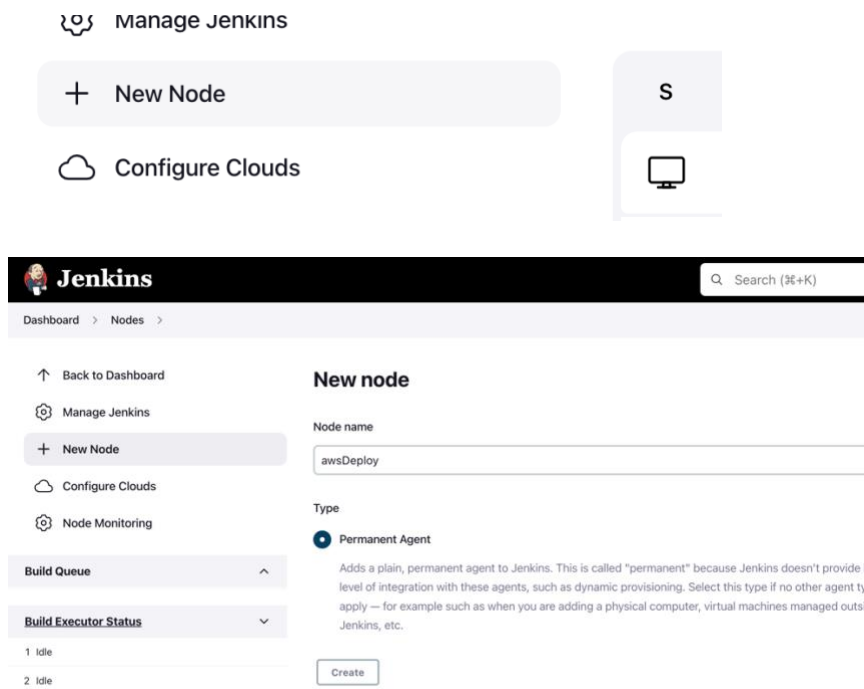
- Lastly, I restarted nginx by running the `<sudo systemctl restart nginx>` command

## 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.

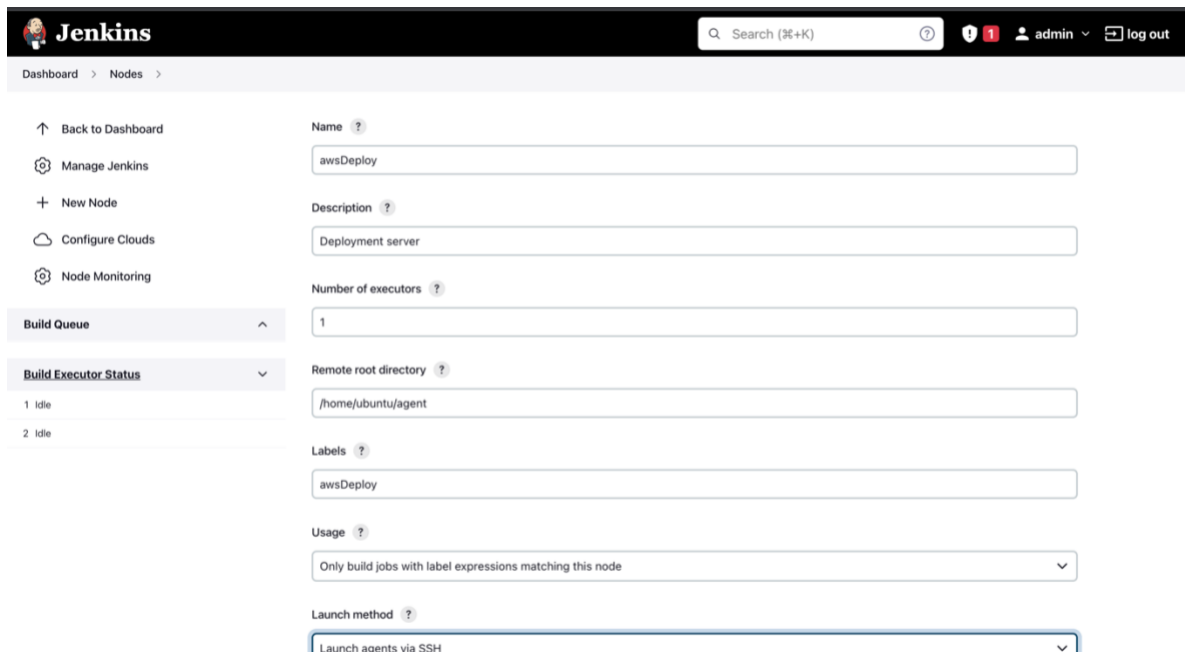


- Next I selected “New Node” to configure and add the Jenkins agent. My node name is “awsDeploy” and I selected “Permanent Agent” and then create.



- When configuring the agent, I input these settings.

- Name: **awsDeploy**
- Description: **Deployment server**
- Number of executors: **1**
- Remote root directory: **/home/ubuntu/agent**
- Labels: **awsDeploy**
- 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**



The screenshot shows the Jenkins 'New Node' configuration page. The left sidebar contains navigation links: 'Back to Dashboard', 'Manage Jenkins', 'New Node', 'Configure Clouds', and 'Node Monitoring'. The main form area is titled 'Name' and contains the following fields:

- Name:** awsDeploy
- Description:** Deployment server
- Number of executors:** 1
- Remote root directory:** /home/ubuntu/agent
- Labels:** awsDeploy
- Usage:** Only build jobs with label expressions matching this node
- Launch method:** Launch agents via SSH

- 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.

Dashboard > Nodes >

Launch method ?  
Launch agents via SSH

Host ?  
54.162.148.185

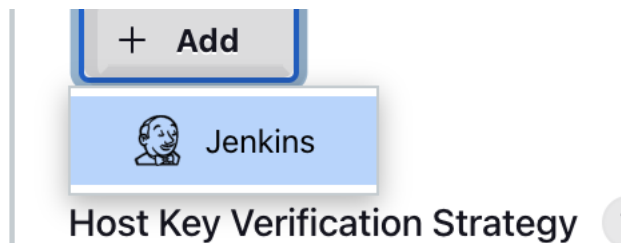
Credentials ?  
- none -  
[+ Add](#)

Host Key Verification Strategy ?  
Non verifying Verification Strategy  
[Advanced...](#)

Availability ?  
Keep this agent online as much as possible

Node Properties  
☐ Disable deferred wipeout on this node ?

- 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.

Dashboard > Nodes >

Launch method ?

### Jenkins Credentials Provider: Jenkins

#### Add Credentials

Domain  
Global credentials (unrestricted) ▼

Kind  
SSH Username with private key ▼

Scope ?  
Global (Jenkins, nodes, items, all child items, etc) ▼

ID ?  
JenkinsAgent

Description ?  
Deployment Agent Server

#### Node Properties

☐ Disable deferred wipeout on this node ?

Dashboard > Nodes >

Launch method ?

Description ?  
Deployment Agent Server

Username  
ubuntu

☐ Treat username as secret ?

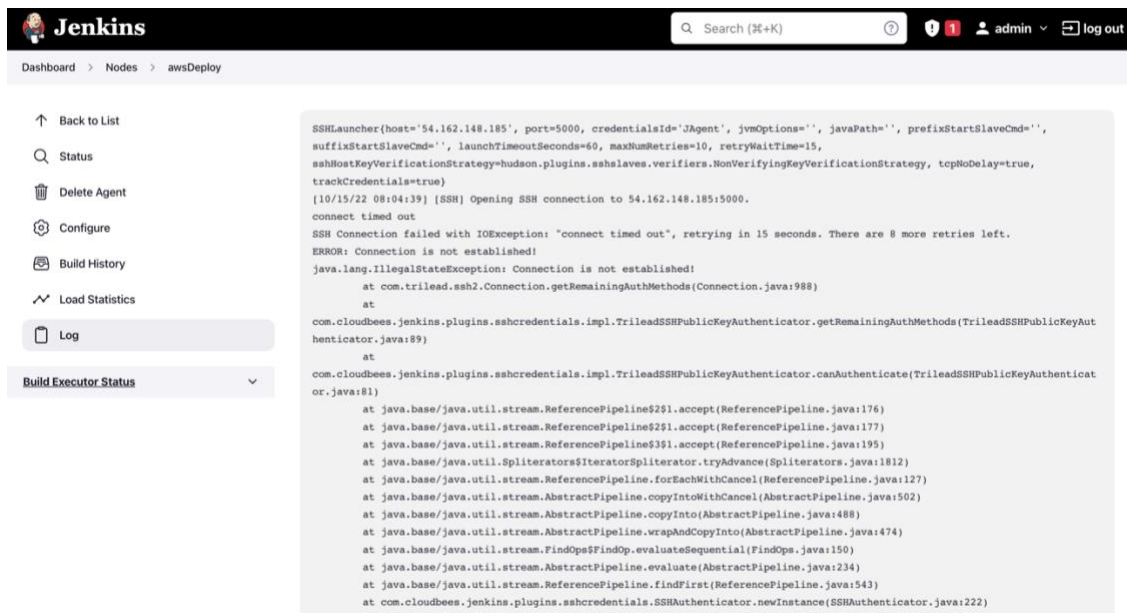
Private Key  
☒ Enter directly

Key  
Enter New Secret Below

Passphrase

#### Node Properties

- 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.



Jenkins

Dashboard > Nodes > awsDeploy

Back to List

Status

Delete Agent

Configure

Build History

Load Statistics

Log

Build Executor Status

```

SSHLauncher(host='54.162.148.185', port=5000, credentialsId='JAgent', jvmOptions='', javaPath='', prefixStartSlaveCmd='',
suffixStartSlaveCmd='', launchTimeoutSeconds=60, maxNumRetries=10, retryWaitTime=15,
sshHostKeyVerificationStrategy=hudson.plugins.sshslaves.verifiers.NonVerifyingKeyVerificationStrategy, tcpNoDelay=true,
trackCredentials=true)
[10/15/22 08:04:39] [SSH] Opening SSH connection to 54.162.148.185:5000.
connect timed out
SSH Connection failed with IOException: "connect timed out", retrying in 15 seconds. There are 8 more retries left.
ERROR: Connection is not established!
java.lang.IllegalStateException: Connection is not established!
    at com.trilead.ssh2.Connection.getRemainingAuthMethods(Connection.java:988)
    at
    com.cloudbees.jenkins.plugins.sshcredentials.impl.TrileadSSHPublickeyAuthenticator.getRemainingAuthMethods(TrileadSSHPublickeyAut
henticator.java:89)
    at
    com.cloudbees.jenkins.plugins.sshcredentials.impl.TrileadSSHPublickeyAuthenticator.canAuthenticate(TrileadSSHPublickeyAuthenticat
or.java:81)
    at java.base/java.util.stream.ReferencePipeline$2$1.accept(ReferencePipeline.java:176)
    at java.base/java.util.stream.ReferencePipeline$2$1.accept(ReferencePipeline.java:177)
    at java.base/java.util.stream.ReferencePipeline$3$1.accept(ReferencePipeline.java:195)
    at java.base/java.util.Spliterators$IteratorSpliterator.tryAdvance(Spliterators.java:1812)
    at java.base/java.util.stream.ReferencePipeline.forEachWithCancel(ReferencePipeline.java:127)
    at java.base/java.util.stream.AbstractPipeline.copyIntoWithCancel(AbstractPipeline.java:502)
    at java.base/java.util.stream.AbstractPipeline.copyInto(AbstractPipeline.java:488)
    at java.base/java.util.stream.AbstractPipeline.wrapAndCopyInto(AbstractPipeline.java:474)
    at java.base/java.util.stream.FindOps$FindOp.evaluateSequential(FindOps.java:150)
    at java.base/java.util.stream.AbstractPipeline.evaluate(AbstractPipeline.java:234)
    at java.base/java.util.stream.ReferencePipeline.findFirst(ReferencePipeline.java:543)
    at com.cloudbees.jenkins.plugins.sshcredentials.SSHAuthenticator.newInstance(SSHAuthenticator.java:222)
  
```

- 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
Key has comment: I deduced it!
  
```

- 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>

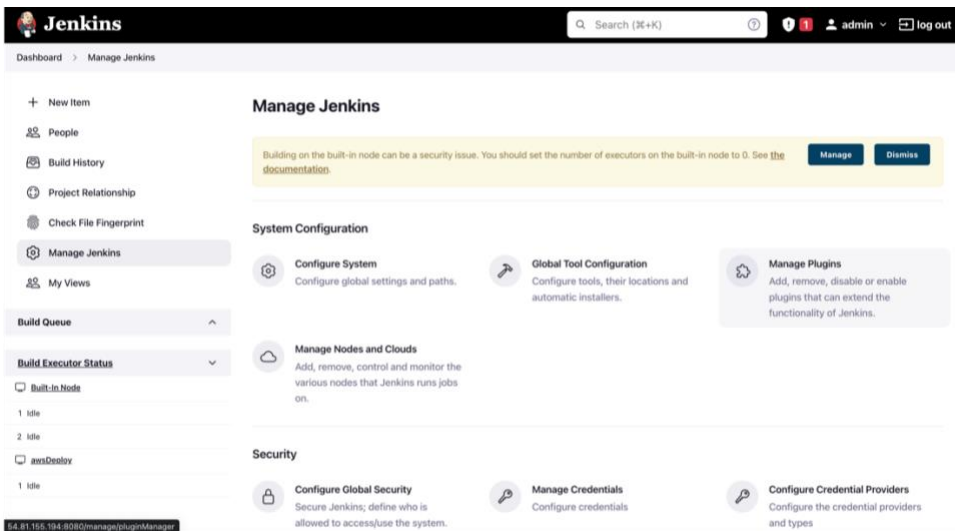
```

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

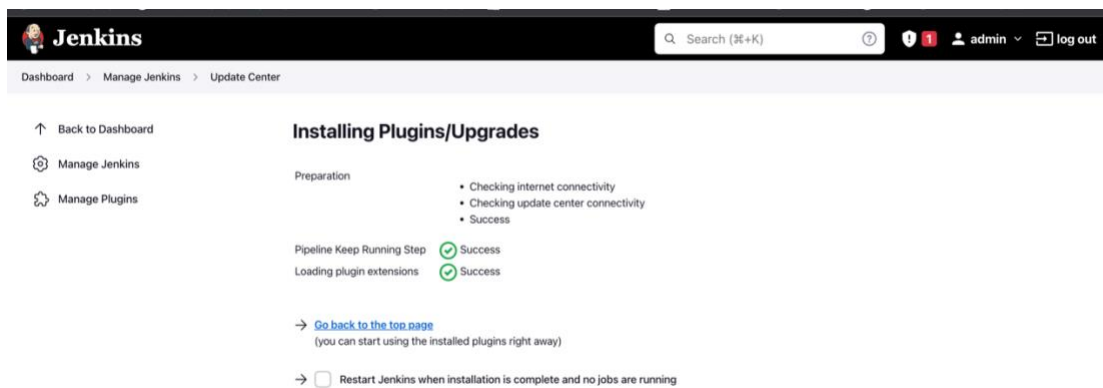
- I launched the agent again and a connection was established on port 5000. Eureka!

```
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$ChannelSwapper to construct
java.io.FileDescriptor(int)
WARNING: Please consider reporting this to the maintainers of jenkins.slaves.StandardOutputSwapper$Cha
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.

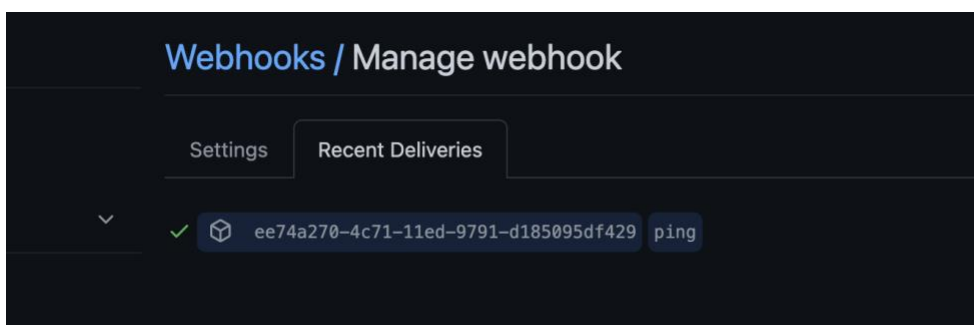
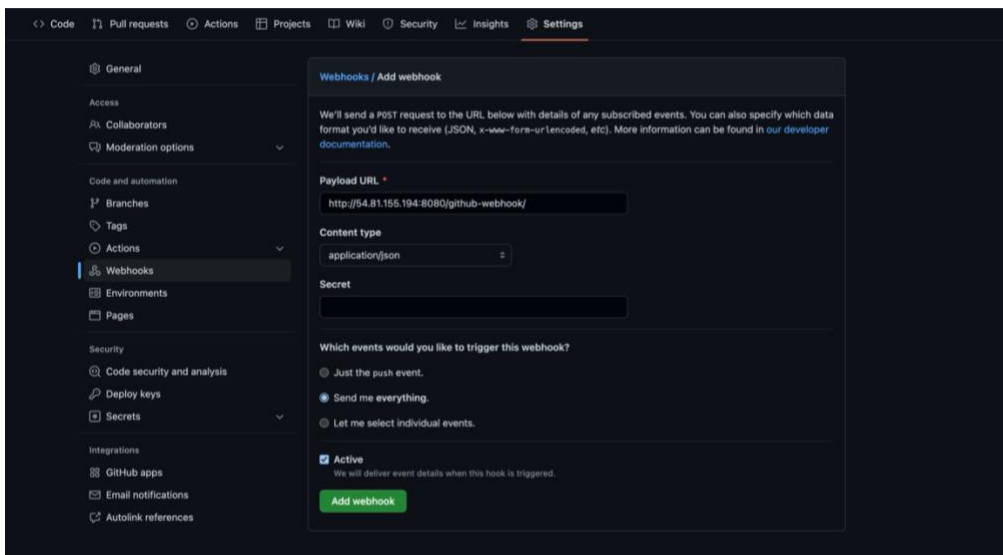
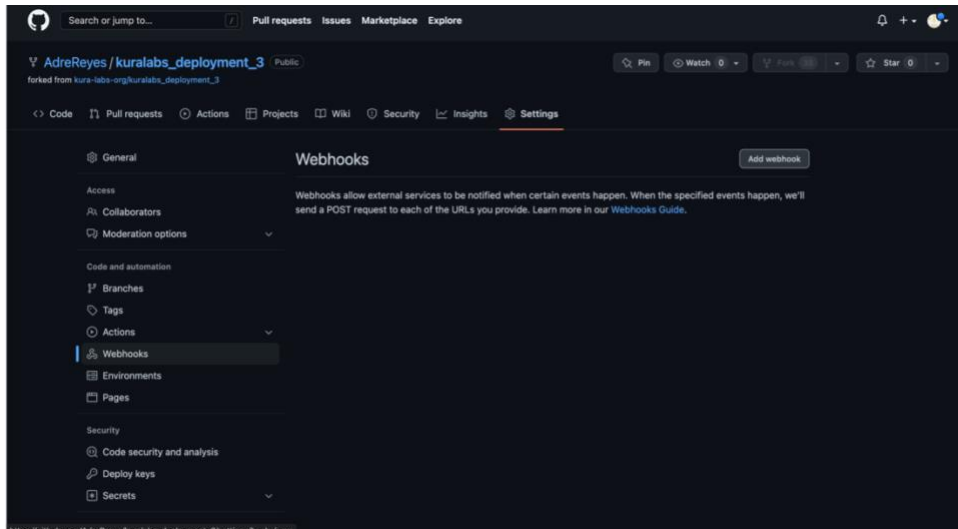
50 lines (49 sloc) | 1.12 KB

```
1 pipeline {
2   agent any
3   stages {
4     stage ('Build') {
5       steps {
6         sh '''#!/bin/bash
7         python3 -m venv test3
8         source test3/bin/activate
9         pip install pip --upgrade
10        pip install -r requirements.txt
11        export FLASK_APP=application
12        flask run &
13        '''
14      }
15    }
16    stage ('test') {
17      steps {
18        sh '''#!/bin/bash
19        source test3/bin/activate
20        py.test --verbose --junit-xml test-reports/results.xml
21        '''
22      }
23      post{
24        always {
25          junit 'test-reports/results.xml'
26        }
27      }
28    }
29  }
30 }
```

```
26   }
27 }
28
29 stage ('Clean') {
30   agent{label 'awsDeploy'}
31   steps {
32     sh '''#!/bin/bash
33     if [[ $(psaux|grep -i "unicorn"|tr -s ""|head -n1|cut -d "-" -f2) != 0 ]] then
34       psaux|grep -i "unicorn"|tr -s ""|head -n1|cut -d "-" -f2 > pid.txt kill $(cat pid.txt)
35     fi
36     exit 0
37   }
38 }
39
40 stage ('Deploy') {
41   agent{label 'awsDeploy'}
42   steps {
43     keepRunning {
44       sh '''#!/bin/bash
45       pip install -r requirements.txt
46       pip install gunicorn
47       python3 -m gunicorn -w 4 application:app -b 0.0.0.0 --daemon
48       '''
49     }
50   }
51 }
```

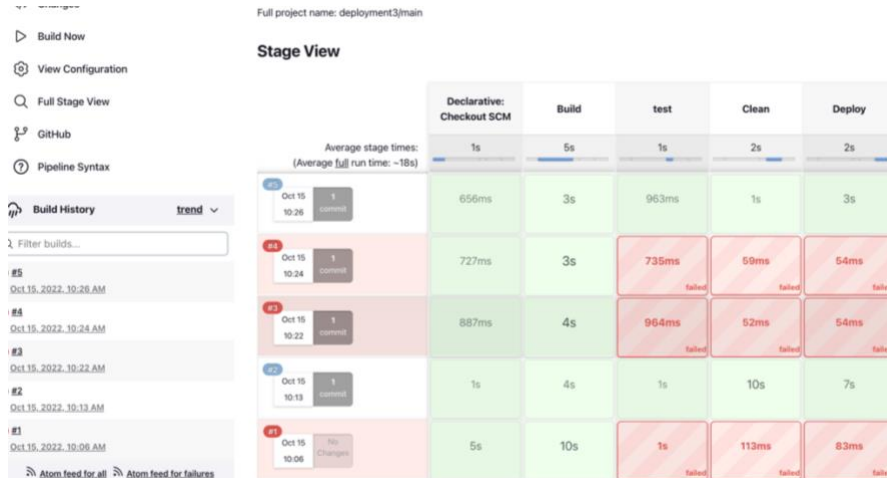


- Next, I added a webhook so that the activity in my Github repo will trigger and match up with the Jenkins builds and I can keep track of the activity in Github. I changed the Payload URL to the url my Jenkins runs on which is <https://54.81.155.194:8080>.



## Build CI/CD Pipeline

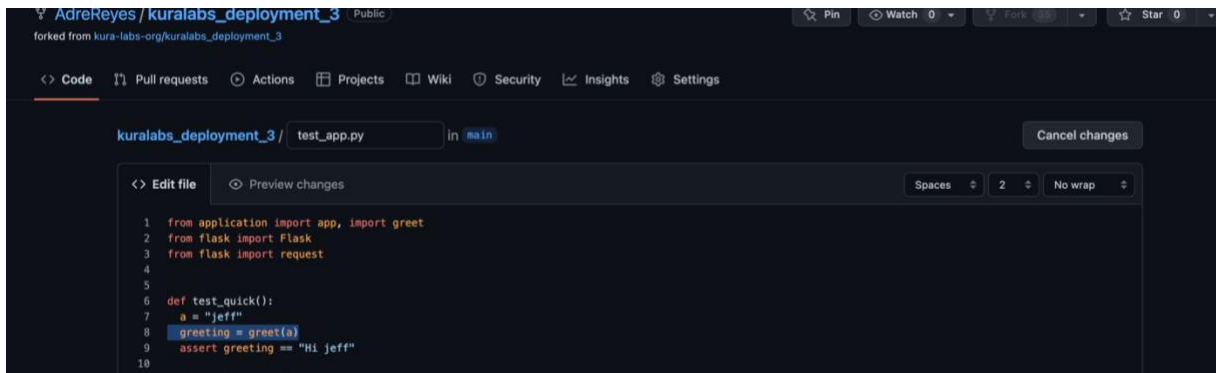
- So, my first build failed and that was before I edited my test file so after correcting the script the second build was successful.



- 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()”.

```
Stage Logs (test)
Shell Script -- #!/bin/bash source test3/bin/activate py.test --verbose --junit-xml test-reports/results.xml (self time 1s)
test_app.py::test_quick FAILED
===== FAILURES =====
test_quick
def test_quick():
    a = "jeff"
    greeting = greet(a)
    > assert greeting == "Hi jeff"
    AssertionError: assert 'Hi jeff' == 'Hi jeff'
    E - Hi jeff
    E ? -
    E + Hi jeff
test_app.py:6: AssertionError
- generated xml file: /var/lib/jenkins/workspace/deployment3_main/test-reports/results.xml -
===== short test summary info =====
FAILED test_app.py::test_quick - AssertionError: assert 'Hi jeff' == 'Hi jeff'
===== 1 failed in 0.17s =====
Archive JUnit-formatted test results -- test-reports/results.xml (self time 203ms)
Permalinks
#1
Oct 15, 2022, 10:08 AM
Last build (#1), 31 sec ago
Last failed build (#1), 31 sec ago
```

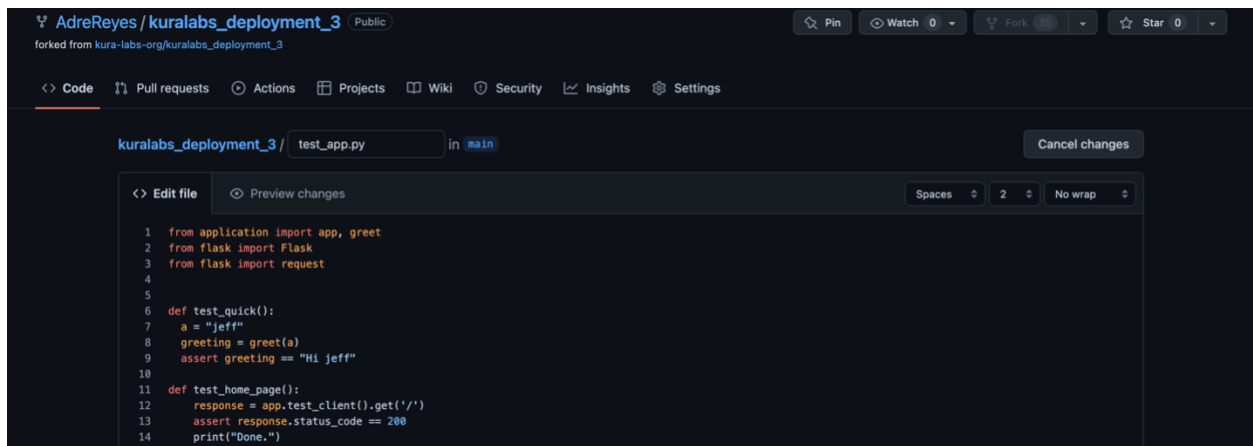
- I thought I fixed everything, but my build failed again (for the fourth time) because of the small “import greet” code mistake which is incorrect. Greet is not a module so I deleted “import”.



The screenshot shows a GitHub repository named 'AdreReyes/kuralabs\_deployment\_3' which is a fork of 'kura-labs-org/kuralabs\_deployment\_3'. The repository is public and has 0 stars and 0 forks. The 'Code' tab is selected, showing the file 'test\_app.py' in the 'main' branch. The code is as follows:

```
1 from application import app, import greet
2 from flask import Flask
3 from flask import request
4
5
6 def test_quick():
7     a = "jeff"
8     greeting = greet(a)
9     assert greeting == "Hi jeff"
10
```

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



The screenshot shows the same GitHub repository, but the code in 'test\_app.py' has been updated to include a new test function 'test\_home\_page()'. The code is as follows:

```
1 from application import app, greet
2 from flask import Flask
3 from flask import request
4
5
6 def test_quick():
7     a = "jeff"
8     greeting = greet(a)
9     assert greeting == "Hi jeff"
10
11 def test_home_page():
12     response = app.test_client().get('/')
13     assert response.status_code == 200
14     print("Done.")
15
```

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