Kevin Sanaycela

Deployment 3

**Step 1: Installing Jenkins on an EC2 in the Default VPC**

**Summary:**

* I first started by launching an EC2 instance with Jenkins in my default VPC using the Ubuntu operating system. I ensured ports 22, 80, and 8080 were open for the inbound rules as in the previous two deployments.
* Jenkins operates using a Controller-Agent model. In this deployment, the controller server sits in our default VPC, whereas our agent server (which carries out the actual deployment) sits in our newly created VPC, discussed next.

**Step 2: Installing Jenkins on an EC2 in a Custom VPC**

**Summary:**

* Here, I launched an Ubuntu EC2 instance in a newly created VPC. Using the CIDR method, the routing table for this VPC is 172.25.0.0/16, which is Class B. This means that our IP addresses in this network span from 172.25.0.1 – 172.25.255.254, inclusive.
  + The above IP addresses are private IPs, but we can create a public subnet using 172.25.0.0/18, which divides our network into four groups. In our subnet configurations, we must ensure it auto-assigns a public IP.
* I configured the EC2 using ports 22 and 5000. Port 5000 allows the client to connect to nginx, while port 22 allows for SSH requests.
* Nginx can serve as either a front-end web server for static content or as a proxy server.
  + As a web server, Nginx can handle general domain logic and HTTP connections.
  + As a proxy server, nginx acts as an intermediary between the client and the application server containing gunicorn.
    - Proxy servers can add an extra layer of security, since gunicorn and the client are not directly communicating with each other.
* Gunicorn is a Python Web Server Gateway Interface. On the server side sits nginx, while on the application side sits Flask. Its key advantage is in allowing our application to handle many more requests per second than an application stack with only nginx.
  + Gunicorn works by internally handling the calling of our Flask/Python code. It does this by providing nginx (the server side) with a callable object when there is a request.
  + In summary, “workers” within gunicorn are ready to handle the requests by providing said object when invoked, resulting in reduced latency compared to a sequential, one-at-a-time model of handling requests.

**Step 3: Configuring Jenkins Agent and Connecting to Jenkins**

**Summary:**

* In this step, we created a Jenkins agent named awsDeploy, which sits in our custom VPC and deploys our application.
* We configure awsDeploy so that it creates a workspace for our application in a newly created directory with the path /home/ubuntu/agent/workspace.

**Issues:**

* When setting the host IP address for the agent, I initially used the IP address for the EC2 sitting in our Default VPC, which is incorrect. We instead put the public IP of the agent.
  + When stopping the EC2 instance and relaunching, the public IP address of our agent server can change. Thus, we must be careful and ensure in our agent configurations that the IP address is updated to reflect any changes.
  + The above changes must be made to the GitHub webhook as well.
* When setting the token and entering the contents of the .pem file, make sure to include *all* the text, including the lines at the beginning and end of the file with all the dashes. Otherwise, the agent will not be able to deploy the app.

Successful connection of agent to controller:

Text

Description automatically generated

**Step 4: Creating Pipeline Build in Jenkins**

**Summary:**

* We configure the “/etc/nginx/sites-enabled/default” file so that the client can connect to nginx via port 5000. From there, nginx can connect to gunicorn in its role as a proxy server via port 8000. Information the user requested can then be sent back via these two ports back to the client.
* It is important to note that the application itself is running on port 8000, but nginx allows us to access the application as a user from port 5000 since nginx is “listening” on this port.
* In the Jenkinsfile, we build out our pipeline, which has 4 stages: Build, Test, Clean and Deploy. The Build and Test stages are run on the Jenkins controller, whereas the other two stages are executed in the agent.
  + We make use of the “keepRunning” plug-in so that our gunicorn processes do not terminate once moving on to the next stage.

**Issues:**

* Groovy syntax – I initially did not have the correct number of closing curly brackets, which resulted in errors when running the pipeline. Spacing is also very important, even within the blocks of code where we include Bash scripts. Missing spaces resulting from copying and pasting resulted in a myriad of errors.

**Successful deployment of Jenkins pipeline:**

Graphical user interface, application

Description automatically generated

**Connecting to our web application via port 5000 on our Jenkins agent:**

Graphical user interface, application

Description automatically generated

**Step 5: Adding slacksend() Plug-in for Notifications**

**Summary:**

* Head to the Slack App Directory on Slack app > search for Jenkins CI > install to the channel you’d like to receive notifications (I created a new channel called “deployment3” > edit Configuration > make note of the token and the team subdomain
* Head to Manage Jenkins > Configure System > Scroll to Slack and add the team subdomain to Workspace tab > Add Credentials
  + Click on Jenkins > Kind = Secret Text > enter the token as the Secret and ID/Description at your discretion

**Graphical user interface, text, application, email

Description automatically generated**

**Updated Jenkinsfile with plug-in included after the deployment stage is complete:**

|  |
| --- |
|  |

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

python3 -m venv test3

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 [[ $(ps aux | grep -i "gunicorn" | tr -s " " | head -n 1| cut -d " " -f 2) != 0 ]]

then

ps aux | grep -i "gunicorn" | tr -s " " | head -n 1| cut -d " " -f 2 > 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

'''

}

}

post{

success {

slackSend(message:

"""

DEPLOYMENT SUCCESSFUL${custom\_msg()}

""")

}

failure {

slackSend(message: """

DEPLOYMENT FAILED ${custom\_msg()}

""")

}

}

}

}

}

def custom\_msg()

{

def JENKINS\_LOG=

"""

Job: [${env.JOB\_NAME}]

Path to log of each step: http://3.89.115.40:8080/job/deployment3/indexing/console

"""

return JENKINS\_LOG

}

Graphical user interface, text, application, email

Description automatically generated

**Improvement**

* In the future, I would place the Jenkins agent and controller in the same VPC, but just have them on different public subnets within the network. This could help reduce latency, since we can still have multiple availability zones in one VPC. This allows us to still be able to have multiple copies of our infrastructure in case one of the AZs goes down.
  + In essence, we can still have redundancy without the need for more than one VPC.

**Resources Used**

<https://vsupalov.com/what-is-gunicorn/>

<https://blog.ironboundsoftware.com/2016/06/27/faster-flask-need-gunicorn/>

<https://peps.python.org/pep-3333/>

<https://docs.digitalocean.com/tutorials/app-deploy-flask-app/#:~:text=Flask%20is%20a%20Python%2Dbased,a%20pre%2Dfork%20worker%20model>.