Deployment 3

October 11, 2022

By: Kingman Tam- Kura Labs

Purpose:

To deploy an application in a customized VPC using a Jenkins agent, Nginx, and Gunicorn.

Previously, Flask Applications were deployed in an Elastic Beanstalk environment that was in the same VPC and subnet as the tools used to 'Build', 'Test', and 'Deploy' the application. Although effective, this leaves the deployment pipeline vulnerable to malicious attacks due to all of the programs and files being in the same place. One solution to this issue is to segregate the different servers into separate public and private subnets. Only the web server would be easily accessible to the public while the application server and database (if applicable) would be private.

This deployment separates the Jenkins Server and the web server into separate VPCs and subnets. This configuration is not typical, but it is another step in learning how to optimize an application and its servers.

Steps:

1. Launch an EC2 with Jenkins installed and activated
    - Jenkins is the main tool used in this deployment for pulling the program from the GitHub repository, then building, testing, and deploying it to a server.
    - The Jenkins EC2 from previous deployments was used. It is on the default VPC that AWS provides
2. Create an EC2 in public subnet in a custom VPC
    - Since this EC2 is located in a separate VPC and subnet than that of the Jenkins server, both services are completely separate and have limited access to each other.
    - This new EC2 needed to have the following packages installed:
        - default-jre: A Jenkins agent will need to run on this EC2. As Jenkins is a Java application, the Java Runtime Environment needs to be installed.
        - python3-pip: Pip is the manager for Python packages. This is needed to create and configure a virtual environment for the application to run.
        - python3.10-venv: This is the virtual environment that the URL-Shortener application will run in.
        - nginx: nginx has many different uses. It can function as a web server, a reverse proxy, and many others. In this deployment, we are using nginx to redirect access to the application from one port to another and to deliver the content of the application (more details below)
3. Configure and connect a Jenkins agent
    - In this step a "node" was added to Jenkins and configured to be used on the EC2 on our custom VPC

- This node launches an agent to the EC2 via SSH. During the configuration, the IP address, username, and private key (.pem file) are input so that it can do so successfully.
- The Jenkins agent allows Jenkins to run commands in a workspace of another server despite not being installed on a machine there.

4. Modify the nginx "sites-enabled/default" file
   - This file tells nginx to take whatever request is made FROM the port that it is LISTENING and then PASS it TO another LOCATION.
     - A port that is "listening" is waiting for a process or application to arrive.
   - This step tells nginx to pass the request for the application that will be from port 5000 to port 8000
     - Gunicorn will later host the application on port 8000 by default (More details below).
   - After being installed, nginx is automatically started and running. After modifying the "sites-enabled/default" file, nginx needs to be restarted for the changes to take effect.
     - The command to run nginx is: `sudo systemctl restart nginx`
5. Edit the Jenkinsfile to include a "Deploy" stage
   - When Jenkins runs the commands in the Jenkinsfile, it will run the commands within each stage.
   - The deploy stage will:
     - Clone the GitHub Repo
     - Download the test3 venv
     - Launch the virtual environment
     - Install the dependencies in the requirements.txt file
     - Install gunicorn
     - Use GUnicorn to launch the application in the background, on any IP: port 8000 (default), with 4 workers
       - `gunicorn -w 4 application:app -b 0.0.0.0 --daemon`
       - THIS COMMAND NEEDS TO BE MODIFIED! SEE TROUBLESHOOTING
6. Configure a multi-branch pipeline on Jenkins
   - When setting up the pipeline, Jenkins needs to be configured to be able to connect to the GitHub repository.
     - This step includes connecting to GitHub with a token and inputting the URL of the repository Jenkins is to connect to.
     - The previous token that was used in Deployment 2 has not expired so the same credentials were used.
7. Build/Test/Deploy
   - After the Pipeline is created, the Github Repository is automatically scanned and Jenkins uses the Jenkinsfile to iterate through the stages.
     - During the first build the "Test" stage failed. The test_app.py file needed to be edited. SEE TROUBLESHOOTING
   - After Jenkins successfully completed all the stages of the Jenkinsfile, attempts to access the application through the web browser were unsuccessful.
     - This issue was resolved by modifying the "JENKINS_NODE_COOKIE" environment variable within the command to launch gunicorn SEE TROUBLESHOOTING
       - `JENKINS_NODE_COOKIE=stayAlive gunicorn -w 4 application:app -b 0.0.0.0 --daemon`

8. Create additions Pipeline
    - Similar to Deployment 2, an additional test and notifications were added to the pipeline
        - The file “test_urls.py” was added to the repository that checked for specific content in the “urls.json” file
        - Slack notification commands were added to the Jenkinsfile so that the results of each stage in the pipeline would be forwarded to the specified member/channel via Slack.
            - The “Slack Notifications” plugin needs to be updated whenever the EC2 that hosts Jenkins is restarted due to the Public IP address changing each time that happens.

Issues/Troubleshooting:

- Jenkins “test” stage fails- The initial build of the pipeline in Jenkins resulted in a failed “test” stage. The logs showed that a report was generated so that means that the py.test did run but the function test did not return a “True” result. The test_app.py had a test of a function in application.py that should have returned a string that matched what was in the test. Upon closer inspection of the test conditions, an extra character was included in the comparative string so when the function was completed, the returned string would never match it. This was corrected as follows and the result was a successful Jenkins “test” stage.

- Application not accessible despite having a “successful” deployment stage- The “Deploy” stage in jenkins file is meant to run the application with gunicorn. That the stage was completed meant that the commands ran with no errors. To test this, each command was run line by line manually in the terminal. The repo was cloned, the virtual environment created and set up, and the application launched. However, every time the Jenkins agent tried to run the application, a 502 error would display:

Nginx was running properly but the request was lost somewhere along the way to the application. Possible causes were that gunicorn was not running or nginx could not connect to it. Since everything ran manually, there should be no issues with connectivity so gunicorn must not be running at the time

that the application needed to be accessed. With that logic, gunicorn must have run during the deploy stage but no longer afterward. To test this theory, a "sleep" command of 5000 seconds was added to the end of the deploy stage:

```
40              pip install gunicorn
41              sudo systemctl restart nginx
42              gunicorn -w 4 application:app -b 0.0.0.0 --daemon
43  +           sleep 5000
44              '''
45            }
46          }
```

The commands preceding it would run, but the deploy stage would stay open for approximately 1.5 hours before it would complete. The Deploy stage historically took 4 seconds on average to complete so after waiting 15 seconds, an attempt to access port 5000 on the servers public IP was tried and the application was available. Aborting the build and refreshing the application page returned the same 502 from previous attempts. This test positively confirmed that Jenkins was killing any ongoing processes that are started during any stage promptly after the stage is completed. Further research on the subject verified that this was a function of Jenkins and a modification was needed to be made to prevent it from happening. An environmental variable called "JENKINS_NODE_COOKIE" needed to be set to "stayAlive" in the same line as the process that the user wants to keep alive after the stage ends. The new command read:

```
JENKINS_NODE_COOKIE=stayAlive gunicorn -w 4 application:app -b 0.0.0.0 --daemon
```

Including this allowed the Jenkins agent to both complete the pipeline and launch the application on the server.

Conclusion:

Adding layers of security to any application also adds complexity. As the application server was located on a VPC that was separate from the server that was responsible for the pipeline, new methods were applied to make sure that it still performed efficiently. This included using a Jenkins agent that connected the two servers, a program (nginx) that would act as a reverse proxy and web server, and an application server (gunicorn) to host the application. Again, this server design is not typical but is useful in understanding the separation of services. Future goals will be to host applications on private servers and accessing them through a NAT gateway for even more security.