kura labs

Deployment #3 – Deploying Software Across VPCs

Author: Ricardo Bonnet

Date: 10-15-2022

Project Description: Four stage pipeline (build, test, clean, deploy) creation utilizing Jenkins. Utilizing two VPCs, deploy an application, with one VPC hosting the Jenkins Server EC2 and the custom VPC hosting the Jenkins Agent EC2. Pipeline customization: Slack for monitoring Jenkins build failure/success. Datadog for monitoring variety of Jenkins metrics (uptime, job success, etc.).

Prerequisites:

- CI/CD Environment (Jenkins)
- AWS Account (EC2, IAM, VPC)
- GitHub Repository to be deployed
- Datadog
- Slack

Creation of Jenkins Server EC2

1) Create an Ubuntu EC2 instance and save the .pem file associated with your EC2 to your local machine. This file will be used later to SSH into your EC2 instance.
    - **How to Create an EC2:** https://www.cloudbooklet.com/create-an-ec2-instance-on-aws-with-ubuntu-18-04/
2) Edit inbound rules and open ports 80, 8080, and 22 on your EC2.
3) Create EC2 and wait for instance to initialize and launch.
4) Navigate to .pem file directory in terminal and run the following command:
    - sudo chmod 600 pemfilename.pem
5) SSH into your EC2 instance using the following command:

- ssh -i pemfilename.pem ubuntu@EC2_Public_IPV4_Address

Installing Jenkins on Your Jenkin Server EC2

1) After connecting to your EC2 instance, run the following commands in order:
    - sudo apt update && sudo apt upgrade -y
    - sudo apt install default-jre
    - wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key |sudo gpg --dearmor -o/usr/share/keyrings/jenkins.gpg
    - sudo 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'
    - sudo apt update && sudo apt install Jenkins -y
    - sudo systemctl start Jenkins
    - sudo systemctl status Jenkins
        - **How to Install Jenkins Guides:**
            - https://www.jenkins.io/doc/tutorials/tutorial-for-installing-jenkins-on-AWS/
            - https://www.digitalocean.com/community/tutorials/how-to-install-jenkins-on-ubuntu-20-04

Installing Virtual Environment

1) First, check if python3 is installed on your EC2 via the following command:
    - Python3 –version
2) If not installed, run the following commands, in order, to install your virtual environment and upgrade to the latest version of python3 and pip:

- sudo apt install python3-pip
- sudo pip3 install --upgrade pip
- sudo apt install python3.10-venv
    - **Guides on Installing Python3, Pip, and Venv:**
        - https://packaging.python.org/en/latest/guides/installing-using-pip-and-virtual-environments/
        - https://installati.one/ubuntu/22.04/python3.10-venv/

Creation of Jenkins Agent EC2 on Custom VPC

1) Navigate to "VPC" on AWS and create a VPC. In our case, our custom Kura VPC has already been made with both public and private subnets.
2) After creating the custom VPC, navigate to the "EC2" section of AWS. Launch an instance to create a new EC2.
3) When configuring the Jenkins Agent EC2, click "Edit" under the Network Settings section. Select the custom VPC along with a public subnet under the VPC. Enable "Auto-assign Public IP".
4) When creating or selecting a security group, ensure inbound TCP ports 22 and 5000 are open, with the source set to 0.0.0.0/0. Port 5000 will be used to communicate with Nginx.
5) Select the same .pem file used for the Jenkins Server EC2 when creating this new EC2.
6) SSH into the Jenkins Agent EC2 and install the following packages via this command:
    - sudo apt install default-jre nginx python3-pip python3.10-venv
7) Run the nano command as follows: nano /etc/nginx/site-enabled/default
8) Make the following edits in the file - Change the port from 80 to 5000, then scroll down and replace the location text as directed in the edits below:

- **Issue:** To avoid issues further into the deployment, run the commands "sudo service nginx start" and "sudo service nginx stop" after making these edits. This will allow the changes to take effect.

Edit 1:

```
server {

        listen 5000 default_server;

        listen [::]:5000 default_server;
```

Edit 2:

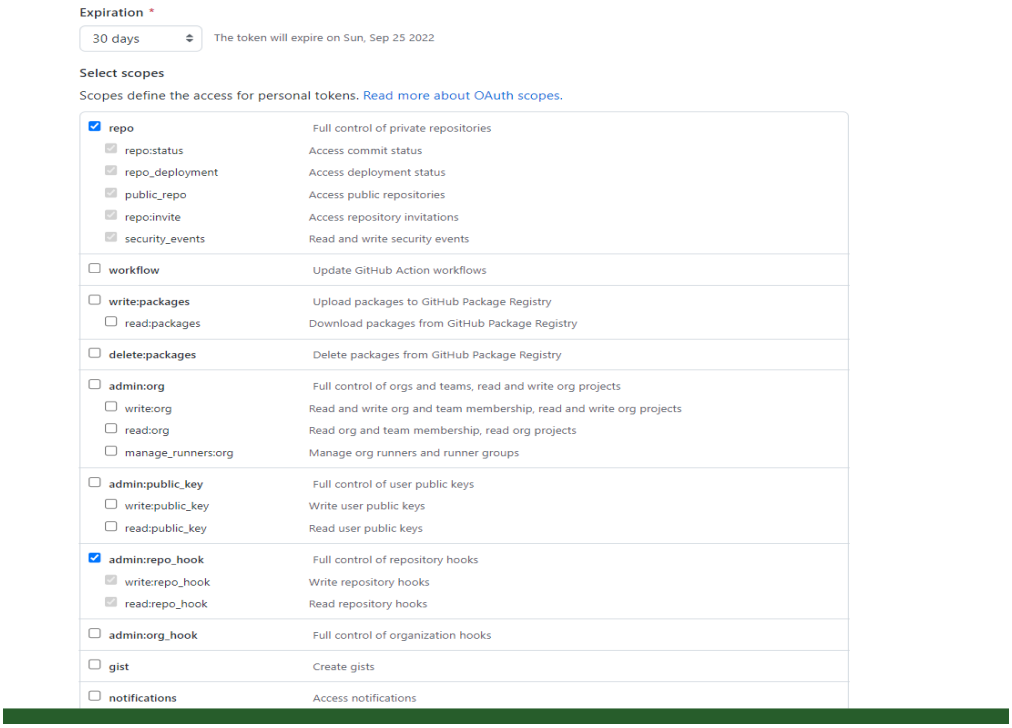
```
location / {
        proxy_pass http://127.0.0.1:8000;
        proxy_set_header Host $host;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

}
```

Connecting GitHub to Jenkins Server

1) Fork the designated deployment repository:

- https://github.com/rbonnet9/Kura_Labs_Deployment_3
- Create personalized access token from your GitHub:
    - Navigate to your GitHub Settings → Developer Settings → Personal Access Token → Generate New Token

- Select settings below when generating token:



- Once token is generated, copy the unique ID that is generated and store for later use.

Setting up Jenkins

1) Connect to http://"EC2_public_IPV4_Address":8080 without the quotes from browser. This directs you to the Jenkins interface.
2) Go into your terminal, SSH Into the Jenkins Server EC2, and run the following command to reveal your Jenkins password required to unlock the interface:
   - sudo cat ~/var/lib/jenkins/secrets/initialAdminPassword
   - Copy the password displayed and paste into Jenkins interface to continue with your setup.
   - Once unlocked, select "install recommended plugins" and let your environment finish its setup.

3) Create your admin user (username, password, full name, email) when prompted
4) The Jenkins URL provided after creating your user can be used to access your Jenkins interface in the future. It's your EC2 public IPV4 with port 8080.

Configure & Connect a Jenkins Agent to Jenkins Server

1) Enter the Jenkins GUI of the Jenkins Master server. Scroll down and select "Build Executor Status", and lastly select "+ New Node". Give the node the name "awsDeploy" and select "Permanent Agent". Create the node and continue to configure.
2) The node configuration is as follows:
    - Name: awsDeploy
    - Description: Deployment server
    - Number of executors: 1
    - Remote root directory: /home/ubuntu/agent
    - Labels: awsDeploy
    - Usage: only build jobs with label
    - Launch methods: launch agents via SSH
    - Host: Public IPv4 of Jenkins Agent EC2 (EC2 in the custom VPC)
    - Credentials: ubuntu (SSH-CALI)
        - Add → Jenkins → SSH username and private key
        - Scope: Global, ID: JenkinsAgent, Description: Deployment agent server
        - Username: ubuntu, private key: EC2 private key (via ssh -keygen)
    - Host verification strategy: non verifying verification strategy
    - Availability: Keep this agent online as much as possible
3) Click "Apply" then "Save" and wait as the new node for the Jenkins agent is launched by the Jenkins manager.

Adding Clean Stage and Modifying Deploy Stage

1) Download the “Pipeline Keep Running Step” plugin within the Jenkins GUI.
2) Enter the jenkinsfile to be modified within the repository. A clean stage utilizing Gunicorn will be added, as well as modifications to the deploy stage will be made.

Clean Stage Addition:

```
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
                 '''
                }
```

Deploy Stage Modification:

```
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
    '''
  }
```

Creating a Multibranch Build in Jenkins

1) In Jenkins, select “New Item”, and select “Multibranch Pipeline”
2) Enter a display name, “Build Flask” for this deployment.
3) Add a branch source, select “GitHub”.
    - Under GitHub credentials, select “Add” and select “Jenkins”
        - Under username, input your GitHub username.
        - Under password, enter your GitHub Token we generated earlier.
        - Enter the repository URL into the designated area.
        - Make sure the “mode” is set to “Jenkins file”.
4) Once all settings are verified as listed, hit apply and save at the bottom of the page.
5) A build will begin happening, if not select “Scan Repository” on the lefthand side of the page.
    - Four stages will be created, your "build", "test", "clean", and "deploy" stages, as designated in the jenkinsfile within your repository.
    - A green box indicates a successful deployment of the designated stages. A red box indicates a failed stage, and any errors can be accessed by hovering over the build stage time and looking at the generated log file for the failed stage.

- **Issue:** The test stage failed due to an extra space in the "test_app.py" file of the repository. Must modify "hi jeff " to "hi jeff". Navigate back Into the Jenkins GUI, click "scan repository" and schedule a new build.

Modification #1 of Pipeline - Slack Notifications

1) Slack integration with Jenkins is an easy, yet hugely insightful modification that can be added to the CI/CD pipeline. By adding Slack notifications, we're notified of the success or failure of each stage in the pipeline.

2) Configure Jenkins integration with Slack via the following link and steps:
   - https://my.slack.com/services/new/jenkins-ci
   - Copy and store the provided integration token for later use
   - Select or create the channel where the Jenkins notifications are to be sent, then select "Add Jenkins CI Integration"
   - Within the Jenkins GUI navigate to Dashboard → Manage Plugins → Configure Settings
   - Click on Slack Notifications → Add+ → Jenkins
   - Add the workspace name, channel name, integration token that was stored earlier, and select "Apply" followed by "Save"

3) To finalize the integration of Slack, add the following code to the jenkinsfile within the GitHub Repository:

```
post{
```

```
    success {
      slackSend channel: 'jenkinsnotifications',
                color: 'good',
                message: "Successful {stage_name_here}  Stage!"
    }
    failure  {
      slackSend channel: 'jenkinsnotifications',
                color: 'warning',
                message: "{stage_name_here} Stage Failed!"
    }
  }
```

4) This code is to be added to each stage that you wish to receive a notification from, such as the build, test, and/or deploy stages. Also enter the proper stage name in the {stage_name_here} section so you know what stage it is you're being notified about.

5) After successfully modifying the jenkinsfile in the GitHub Repository, schedule a build within the Jenkins GUI. Slack notifications will now appear for each of the designated stages of the build.

Modification #2 of Pipeline - Datadog Integration

1) Datadog and Jenkins integration is made possible in several ways, most easily via a Jenkins plugin. By utilizing Datadog, we can monitor the status of our Jenkins EC2 and be notified via email if the instance is running or powered down.

2) Configure Jenkins integration with Datadog via the following link and steps:
    - https://plugins.jenkins.io/datadog/
    - Within the Jenkins GUI navigate to Dashboard → Manage Plugins
    - Select "Available" and search for "Datadog". Select the checkbox next to the Datadog plugin.
    - Install the Datadog plugin by selecting the "Install" button at the bottom of the page.
    - The installation can be verified by navigating to the "Installed" tab and locating the Datadog plugin.

3) There are several ways to configure the plugin to submit data into the Datadog monitoring center. These include:
    - Sending data to Datadog directly through HTTP
    - Use a Datadog agent that acts as a data forwarder between Jenkins and Datadog
    - While the HTTP method is much easier and straight forward to configure, the Datadog agent method allows the access of Jenkins log while the HTTP method does not.

4) For our purposes, we're going to opt for the HTTP forwarding method. This is done via the following steps:
    - Within the Jenkins GUI, navigate to Manage Jenkins → Configure System
    - Scroll down and locate the Datadog Plugin section
    - Select the button next to "Use Datadog API URL and Key to report to Datadog"

- Paste your Datadog API key into the textbox on the Jenkins configuration screen.
    - If you need to locate or create a new Datadog API key, follow these steps:
        - Log in to your Datadog account at https://www.datadoghq.com
        - In the left-hand menu, navigate to the bottom of the menu and hover over your profile email/name → click Organization Settings → under the "Access" section click on API Keys
        - If a key already exists, click on it to display your API key. If no key exists yet, select the blue "+ New Key" button at the top right of the page to create one.
- Test your Datadog API key by selecting the "Test Key" button on the Jenkins configuration screen below the Datadog API key textbox.
- (Optional Step):
    - A hostname for your Jenkins server can be entered in the "Advanced" tab. This is helpful when you're monitoring several Jenkin servers so you can easily identify which server is paired with a specific event.
- Finalize configuration by saving at the bottom of the page. Datadog and Jenkins are now fully integrated and Datadog will begin monitoring certain metrics that can be customized on the Datadog webpage.
- For official documentation on integrating Datadog with Jenkins, refer to the following links:
    - https://docs.datadoghq.com/integrations/jenkins/
    - https://plugins.jenkins.io/datadog/#plugin-content-plugin-user-interface

---

Potential Deployment Improvements

1) Improvement #1: Automation of Initial Jenkins Install

- Instead of manually entering in the commands to install Jenkins, we could instead input a script in the initial EC2 setup to automatically run when the EC2 is launched. All that would need to be done after the EC2 is launched is run a "sudo apt update && sudo apt upgrade -y" command along with "python3 --version" to make sure everything is installed and updated.
- Script would be as follows:

```
#!/bin/bash

# Download JRE
sudo apt update && sudo apt install default-jre -y

# Get the key to access Jenkins package
wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo gpg --dearmor -
o/usr/share/keyrings/jenkins.gpg

# Using the key to authenticate the package
sudo 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'

# Update and install Jenkins
sudo apt update && sudo apt install jenkins -y

# Start Jenkins and check the status to make sure it is running
sudo systemctl start jenkins
sudo systemctl status jenkins >> ~/file.txt
```

2) Improvement #2: Monitoring via Datadog

- The modification I made to the pipeline could be included in the initial deployment. Monitoring is always an integral part of any CI/CD pipeline to pinpoint failures within a system and monitor important metrics.

- Some metrics we could monitor for our Jenkins EC2 include failed or successful builds, Jenkins system events, Jenkins security events, average job duration, buildable items, pending items, plugin count, and project count. These are just a handful of metrics out of the dozens more we could configure Datadog to monitor.

3) Improvement #3: Automation of Manually Entered Commands

- Many of the commands we entered manually throughout the deployment could be automated in a script. They could even be deployed during the initial creation of our EC2, such as with improvement #1 which would install Jenkins automatically upon launching our EC2.
- Essentially any command that we entered manually could be put into a script for a more rapid deployment. A handful of scripts could be created for future use in all future deployments to speed up the creation of certain components in the CI/CD pipeline.