kura labs

Deployment #2 – Creation & Customization of CI/CD Pipeline

Author: Ricardo Bonnet

Date: 09-27-2022

Project Description: Three stage pipeline (build, test, deploy) creation utilizing Jenkins, AWS Elastic Beanstalk used to deploy application. 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, Elastic Beanstalk, IAM)
- GitHub Repository to be deployed
- Datadog
- Slack

Creation of EC2 Instance

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 an 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/

---

kura labs

Activate Jenkins User on the EC2

1) Enter the following commands:
    - sudo passwd jenkins
    - sudo su - jenkins -s /bin/bash

Creating Jenkins User in AWS Account

1) Navigate to the IAM section in the AWS console.
2) Select the "Users" option in the Access Management section, click "Add User".
3) Input a username, such as EB-User.
4) Place a checkmark in the box labeled "Access Key – Programmatic Access".
5) Navigate to the next page via the "Next" button and select "Attach existing policies". Check "Administrator Access" and select "Next".
6) Select "Next" for the following two pages labeled "Add Tags" and "Review" to finalize your user creation.
7) Download/Save the generated access key and secret access key to be used later and click "Close" to complete and exit the user creation process.

Installing and Configuring AWS CLI on the Jenkins EC2

1) If not already connected to your Jenkins EC2, ssh into the EC2 via the following command: ssh -i pemfilename.pem ubuntu@EC2_Public_IPV4_Address
2) Enter the following commands exactly as shown, in order, to install the AWS CLI:
    - curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
    - Install the unzip function via the command: sudo apt install unzip
    - unzip awscliv2.zip
    - sudo ./aws/install

- To check if AWS CLI was installed correctly enter the command: aws --version

3) Configuring the AWS CLI is done as the Jenkins user, to enter the Jenkins user enter:
   - sudo su - jenkins -s /bin/bash
4) Enter the following command to configure the AWS CLI while you're the Jenkins user:
   - aws configure
     - Enter Access Key
     - Enter Secret Access Key
     - Set region to, in our case, us-east-1
     - Set output format to: json
5) Your AWS CLI is now installed under your Root user and configured under your Jenkins user.

Installing EB CLI in the Jenkins EC2 User

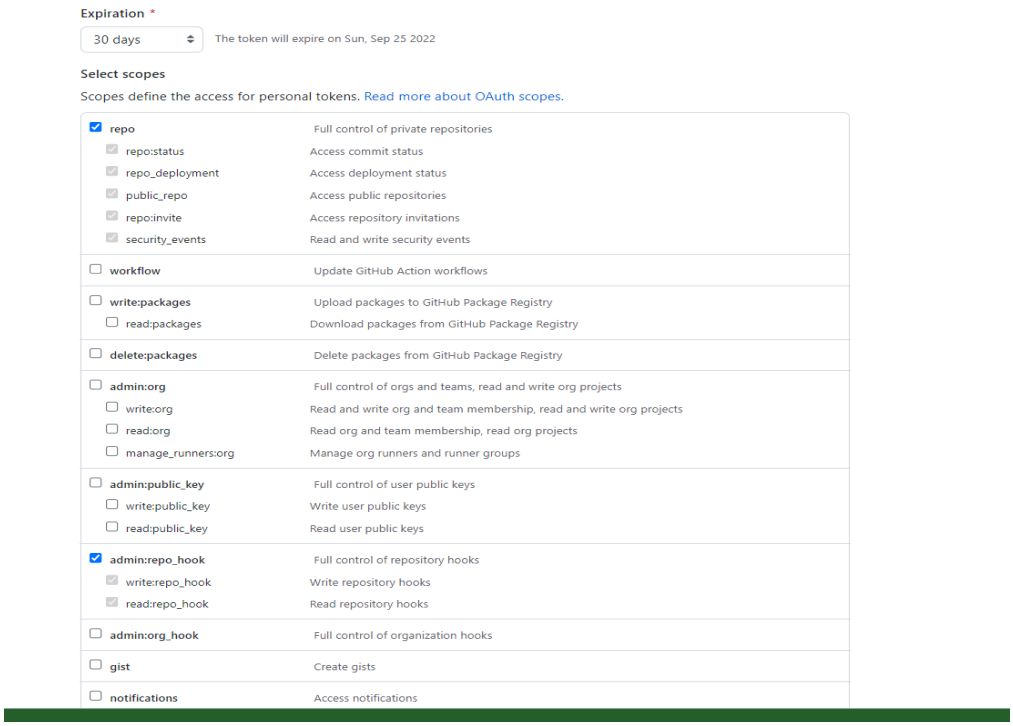
1) Enter the following command to install the EB CLI on your Jenkins EC2 user:
   - pip install awsebcli --upgrade --user
2) To ensure the EB CLI has been installed and its version, enter the following command:
   - eb --version
3) **If the EB CLI is not recognized by the system, the path must be added so that EB application is recognized within the shell! This is done by entering the following:**
   - Exit out of the jenkins user while performing these commands:
   - sudo nano etc/bash.bashrc
   - Enter "export PATH=$PATH:/var/lib/jenkins/.local/bin" at the bottom
   - Enter the jenkins user again via the command: sudo su - jenkins -s /bin/bash
   - "echo $PATH" to ensure the EB path has been added to the PATH variable
   - "eb --version" to verify the EB CLI has installed successfully

---

Connecting GitHub to Jenkins Server

1) Fork the designated deployment repository:

   - https://github.com/rbonnet9/Kura_Labs_Deployment_2
   - 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 EC2 terminal 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.

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.
    - Two stages will be created, your "Build" and "Test" 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 hover overing the build stage time and looking at the generated log file for the failed stage.

Deploying Application from the Elastic Beanstalk CLI

1) Ensure you're in the Jenkins user within the EC2. This is done via the following command: sudo su - jenkins -s /bin/bash
2) Enter the workspace created by Jenkins, by entering the following command:
    - cd /var/lib/jenkins/workspace/Deployment2_main
3) Initialize the Elastic Beanstalk workspace by entering:
    - eb init
4) Select the following parameters for your Elastic Beanstalk configuration:
    - Select us-east-1 → Press Enter
    - Select: Python
    - Select: The latest version of Python
    - Select: N (for CodeCommit)
    - Select: No (for setting up SSH for your instance)
    - Run the command: eb create
        - This will create the Elastic Beanstalk environment with your parameters
    - For the next 3 prompts select the default selected parameter
    - Remember Your Environment Name!
    - Select: No (for spot fleet)
5) Your environment will finish creating, and a fully functioning EB environment is up and running.

Adding Deployment Stage to Pipeline in Jenkins file

1) Add the following stage into your jenkinsfile:

```
stage ('Deploy') {

steps {

sh '/var/lib/jenkins/.local/bin/eb deploy Deployment2-main-dev'

        }

}
```

2) After updating your GitHub repository with the edits made to your jenkinsfile, go back to the Jenkins GUI and select the "Build Now" option within the left-hand menu.

3) After scheduling a new build, a fresh build, test, and deploy will be conducted within Jenkins. If performed correctly, all three stages will pass indicated by green boxes in each stage. Red boxes indicate a failed stage or stages, and this will be logged within Jenkins. Depending on the stage that failed and the reason for the failure, will dictate what potential fixes must be made.

4) If all stages pass, the application has successfully been deployed to Elastic Beanstalk. This can be verified by entering the AWS Elastic Beanstalk environment page and visiting the provided URL for the url-shortener application.

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.

kura labs

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.