IT 351 Cloud Security and DevSecOps

Lab 2

Fall 2021 – Professor Josh Edgar

In this second Lab, we will continue exploring GitLab by setting up infrastructure for a "GitLab Runner," an executor that will take our code after every push to our repo and act upon it. The ways this tool can be used are pretty expansive, since we will also configure our Runner with Docker, another tool that can run jobs in a number of different "containers" that come pre-packaged with specific operating system and application dependencies.

Before you begin:

- You should have already completed Lab 1 successfully.
- You should have already signed up for an AWS account, or have access to your AWS Free Tier account from a previous course.

What you'll achieve:

- Deploy an EC2 instance to host your GitLab Runner
- Install the GitLab Runner and Docker binaries
- Write an initial GitLab CI/CD instruction file to run a pipeline that outputs "Hello World"

What you'll turn in with this lab:

- Please simply go to the Lab 2 assignment on Canvas, and author a text submission that simply says "Done." This will notify the professor to check your GitLab repo for satisfactory completion of the Lab, and enter a grade into Canvas for the same.

Task 1: Deploy an Amazon Linux EC2 Instance

Before we can install a GitLab Runner, we must first deploy an EC2 instance to host it. If you've taken IT-321 or IT-331 before, this process is hopefully burned into your memory. If not, please make sure to watch the Panopto tutorial to ensure that you follow the steps correctly and that the instance is configured and deployed properly.

For this Lab, we will use the Amazon Machine Image (AMI) for Amazon Linux, which is a fork of Red Hat that has been engineered specifically for use with EC2 instances. It works very similar to Red Hat, CentOS, Fedora, etc.

1. Log in to the AWS Management Console. Make sure you are in the US East Northern Virginia Region and navigate to the Instances table of the EC2 Console.
2. Click "Launch instances." You should automatically see a result for the Amazon Linux AMI – select it. Navigate through the instance launch wizard until you get to the "Configure Security Groups" page.

3. Create a new Security Group that has inbound rules allowing traffic on ports 22 (SSH), 80 (HTTP) and 443 (HTTPS), with a source of "Anywhere." Once these rules have been added, continue through the wizard.
4. When prompted to configure the instance with a key pair:
   a. If you have used AWS for a previous course, and still have your original .pem private key file from that course, you may select that existing key pair and check the "I acknowledge" box. **IMPORTANT:** If you do not have that .pem file any longer, you will not be able to re-download it. That key pair is now essentially useless, and you must proceed to step 4(b) to create a new one.
   b. If you have not used AWS previously or do not have access to a previously created private key file, make sure to change the dropdown selection to "Create a new key pair," and come up with a unique name for the key pair. Click the "Download Key Pair" button to download the associated .pem file to your system. Save it somewhere that it will be easily accessible to type its path from a command line later.
5. Finish deploying the instance. Change its name in the Instances table to "gitlab-runner" (without quotes).

Task 2: Install GitLab Runner and Docker Services

Now that we have an Amazon Linux EC2 instance deployed, we can launch an SSH session into it, and install the GitLab Runner and Docker binaries that will be needed for our Runner to execute CI/CD jobs from our GitLab repo.

Note: All commands should be entered as one line. You may want to manually retype these commands into your terminal as opposed to copy-pasting them, as the formatting between Word and your command line tool will likely differ. In any event, disregard any line breaks/word wrapping issues that appear in the commands provided – each step gives you a single command that is intended to be input into the terminal as one line.

6. Open a command line on your local system (either Command Prompt on Windows or Terminal on Mac/Linux). Type in the following command. Make sure to modify the IP address to reflect the Public IPv4 address assigned to your new EC2 instance – this can be found in the instance details pane when you highlight the instance in the EC2 Console Instances table. Also make sure to modify path of the key file to reflect the location where you saved it previously.
   ```
   ssh ec2-user@1.2.3.4 -i ./path/to/key-file.pem
   ```
7. Once you have successfully connected to your instance via SSH, run the following command to elevate your privileges to the root user:
   ```
   sudo su
   ```

8. Run the following command to add the GitLab Runner package repository to the package manager (so that the Runner binaries can be found when we install them).
   ```
   curl -L
   https://packages.gitlab.com/install/repositories/runner/git
   lab-runner/script.rpm.sh | sudo bash
   ```
9. Once you receive the message indicating that "the repository is setup," run the below command to download the GitLab Runner binary from the repository:
   ```
   curl -L --output /usr/local/bin/gitlab-runner
   https://gitlab-runner-
   downloads.s3.amazonaws.com/latest/binaries/gitlab-runner-
   linux-386
   ```
10. Once the package has been downloaded, install it using the following command. Enter "y" for any yes/no prompts.
    ```
    export GITLAB_RUNNER_DISABLE_SKEL=true; sudo -E yum install
    gitlab-runner
    ```
11. Check the status of the Runner service to ensure that it is active and running.
    ```
    systemctl status gitlab-runner
    ```
12. Run the following command to install Docker, again answering "y" for all prompts.
    ```
    yum install docker
    ```
13. Check the status of the Docker service:
    ```
    systemctl status docker
    ```
14. Docker is most likely stopped by default. Run the following command to start it:
    ```
    systemctl start docker
    ```
15. We want to be sure that both of these services start automatically whenever our instance is booted up. To do this, run the following two (2) commands:
    a. `systemctl enable gitlab-runner`
    b. `systemctl enable docker`

Task 3: Register Runner with GitLab & Disable Shared Runners

Now that our GitLab Runner has been installed, we must register it with our GitLab repo. This will allow the repo to contact the Runner after each code push to execute any changes to the repo, in accordance with the GitLab CI/CD instruction file we will write later.

16. Log in to GitLab. Navigate to your repo for this class. On the left sidebar, hover over the "Settings" link and choose "CI/CD" from the sub-menu.
17. Locate the section entitled "Runners" and click the "Expand" button to see more details.
18. Under the "Shared runners" column to the right, locate the slider for "Enable shared runners for this project." Disable this slider, as we do not want to use the paid Runners that are provided by GitLab – we will exclusively be using the one we've just created.
19. Under the "Specific runners" column to the left, copy the alphanumeric registration token to your clipboard.

20. Return to your SSH terminal and run the following command to start the Runner registration process:
    `gitlab-runner register`
21. You will be asked several questions through this process.
    a. When prompted for the GitLab URL, use `https://gitlab.com`
    b. When prompted for the registration token, paste the token you copied from step 19.
    c. When prompted for a name/description for the Runner, name it something like `your-name-first-runner`.
    d. When prompted for tags for the runner, provide just one: `it-351-runner`
    e. When prompted for an executer, simply provide the word: `docker`
22. Refresh the GitLab CI/CD Settings page in your browser. You should now see your Runner identified by the name and tags under the "Specific runners" column.

Task 4: Write a GitLab CI/CD File & Run a Pipeline

In this Task, you will run your very first, albeit simple, DevOps pipeline. GitLab CI/CD instructions, also known as ".gitlab-ci.yml files" after the filename itself, are YAML files with a relatively straightforward syntax to dictate how a pipeline should be executed by our Runner. We can declare numerous stages for different pipeline components, and each stage can even use a different Docker container image from the Docker Hub, which we also specify in this file. We will write a simple file to just demonstrate that our Runner is working in AWS, by asking it to echo a "Hello World" statement back to us.

23. Open VS Code and ensure that you are still able to access your repo from Lab 1.
24. In the Explorer tab, click to create a new file. Name it ".gitlab-ci.yml" (without quotes), paying special attention to the leading period.
25. Download the sample code file that was linked from this Lab assignment on Canvas. Copy and paste this code into your new file. Modify it so that the "echo" command will say "Hello world, this is (your name's) first GitLab Runner at work!"
26. Save, commit, and push this file back to your GitLab repo.
27. Navigate back to GitLab in your browser. On the left sidebar, hover over the main "CI/CD" menu and choose "Pipelines" from the sub-menu. You should see a pipeline for your recent commit – select it. You can then click on the "my-job" stage to see the terminal output from your Runner, which will pull a CentOS Docker image, build a container from it, and then execute the "echo" command inside of it. This is all being performed by your Runner in AWS – not on GitLab's servers. Congratulations! You are now running a DevOps pipeline in the cloud!