IT 351 Cloud Security and DevSecOps

Lab 4

Fall 2021 – Professor Josh Edgar

For this Lab, we will continue building upon our pipeline. Following Lab 3, our pipeline currently uses Terraform to deploy infrastructure in our AWS account, but nothing has been installed on this new instance yet. This is where Ansible enters the equation. Ansible is a tool, developed by Red Hat, that uses YAML "playbooks" to carry out Linux system configuration and administration tasks automatically, and we will be integrating it into our CI/CD pipeline to quickly configure a basic web server – all without having to manually log in to any terminal!

Before you begin:

- You should have already completed Lab 3 successfully.
- Your GitLab Runner instance should be started, if you stopped it after Lab 3.
- You should have run the "destroy" stage of your Lab 3 pipeline, terminating the Terraform-created instance from the Lab (do not terminate your Runner).

What you'll achieve:

- Create an Ansible playbook to install and configure Apache Web Server
- Add an Ansible stage to your GitLab CI/CD file, allowing an Ansible Docker container to execute your playbook and configure your instances created by Terraform

What you'll turn in with this lab:

- The Public IPv4 Address of the EC2 instance that is deployed/configured by your pipeline

Task 1: Prepare Files

Before we can start implementing the actual Ansible functionality into our pipeline, we need to get a few files into our repo that will be used by the Ansible job. Specifically, these include our private key (.pem) file that will be used for SSH authentication into our Terraform-created instances, as well as an index.html file that we will copy into our target web server.

1. Open VS Code and ensure that you are working in your repo in the Explorer tab.
2. On your local file system, locate your private key (.pem) file that you typically use for SSH authentication with your EC2 instances. Copy it into your repo in VS Code.
3. Create a new file called "index.html" (without quotes). Write some basic HTML code (or even just text) in here, something that will display a webpage that indicates your web server is working. For example, "This is Prof. Edgar's new web server," but replacing my name with your own.

Task 2: Make Minor Terraform Changes

Also before we begin working with Ansible, we need to make two minor changes to our Terraform code in support of the Ansible job. These changes will include specifying an SSH key pair for our instance – since Ansible will need to use the .pem file you copied in step 2 to connect to the instance for configuration – as well as ensuring that we get the Public IP address of the instance from Terraform, as this will also be provided to Ansible for connection purposes.

I don't recommend copy-pasting any of the code below from Word into VS Code – you should retype it manually, as formatting issues can occur between the quotes used by Word and the types of quotes that are expected by our code interpreters.

4. Open the "main.tf" file. Beneath the instance_type line, add a line that indicates which SSH key pair should be used for authenticating with this instance. (It will most likely match the name of your .pem file without the file extension at the end, but you can double check what key names are stored in your AWS account by going into the EC2 Console and clicking "Key Pairs" on the left sidebar.) The new line of code should look something like:
   ```
   key_name = "my-key-pair-name"
   ```
5. In this same file, we will also add a short stanza that asks Terraform to provide an output from the resource it creates. Specifically, we want Terraform to output the Public IP address of our my_instance EC2. The actual output comes from a command that we will issue from our GitLab CI/CD file, but we still need to declare it in our main.tf file so that it knows to store the output before we request it. Add this to the bottom of your main.tf:
   ```
   output "instance_ip" {
      value = aws_instance.my_instance.public_ip
   }
   ```

Save and close your main.tf file.

Task 3: Write An / Steal My Ansible Playbook

As previously mentioned, Ansible uses "playbooks" to know what types of configuration tasks to carry out on the hosts that it administers. These are really just YAML files that use Ansible-specific markup to conduct Linux administration tasks. The syntax can be a little hard to grasp, so I've provided the exact code you'll need on Canvas, but be prepared to do some research into how to write your own playbook syntax for our upcoming pipeline project.

6. Download the Ansible Sample Code that was linked from the Lab assignment page on Canvas. In VS Code, create a new file in your repo called "playbook.yml" and paste the sample code into this file. Save and close the file.

Task 4: Update Our GitLab CI/CD File to Add Ansible Stage

Now that we've put all of the files we need for an Ansible job into our repo, we need to update our GitLab CI/CD instructions to add a configuration stage, which will contain our Ansible playbook execution job. Unlike Terraform, this is only one job that will come after our Terraform "apply" stage, but it will fetch a container with the ansible-playbook utility installed, and then use a hosts file created by our updated CI/CD file to connect to and configure our EC2 instance.

7. In VS Code, open your ".gitlab-ci.yml" file.
8. In the top section where our "stages" are listed, add an entry after "apply" and call this new stage "configure" (without quotes).
9. In our "tf-apply" job code, add a line to the "script" section under the "terraform apply" command:
   `terraform output instance_ip >> ansible-hosts.yml`
   This recalls our Terraform output we declared in step 5, and routes the output to a file called "ansible-hosts.yml."
10. We will also add a new section to "tf-apply" to store this hosts file. After the "rules" section, add the following:
```
artifacts:
   name: ansible-hosts
   paths:
     - ansible-hosts.yml
```
11. Finally, add an entirely new section of job code between the "tf-apply" and "tf-destroy" sections. This will be for our Ansible job. Make sure to change the .pem file name that is referenced to the name of your private key file in your repo!
```
ansible:
   image:
      name: mullnerz/ansible-playbook
      entrypoint: [""]
   stage: configure
   before_script:
     - chmod 0600 my-key-pair-name.pem
   script:
     - ansible-playbook -i ansible-hosts.yml playbook.yml --key-file my-key-pair-name.pem
   needs:
     - job: tf-apply
   tags:
     - it-351-runner
```

Save and close this file, but don't commit or push anything just yet.

Task 5: Add a GitLab Environment Variable to Keep Ansible Happy

While testing this Lab, I was not able to get Ansible to connect to even a brand new EC2 instance without complaining that the host keys did not match (something that typically happens when a system is rebuilt, but in our case, we're building a fresh instance to configure.) The workaround to this is to add another GitLab environment variable that will tell Ansible to not check for matching host keys. This is not ideal for a production environment, since it *can* expose your system to man-in-the-middle attacks, but for the purposes of this Lab, we're going to bend the rules and do it anyway.

12. Log into GitLab and navigate to your repo/project. Navigate to Settings > CI/CD > Variables and add a new variable.
    a. Key: ANSIBLE_HOST_KEY_CHECKING
    b. Value: False (case sensitive)
13. Make sure that the protect/mask variable checkboxes are unchecked and save the variable.

Task 6: Commit and Push Changes / Run the Pipeline

14. Make sure all of your files are saved in VS Code, named appropriately, and that your code looks accurate and that there are no obvious syntax issues. If it all looks good, commit your changes with a message about Ansible/Lab 4, and push them up to your server-side repo.
15. Navigate to GitLab > CI/CD > Pipelines to watch your pipeline's progress. Remember to manually trigger the "apply" job for Terraform. The Ansible job should trigger automatically once the Terraform apply is completed. Observe this configure stage/Ansible job for any errors – or, if the job is successful, load up your instance's public IP address (found in the EC2 Console) in a web browser, and see if you see your HTML file!