MARYMOUNT UNIVERSITY

IT 351 Cloud Security and DevSecOps

Lab 7

Fall 2021 – Professor Josh Edgar

In this Lab, we will work with a continuous compliance tool known as Chef InSpec. Candidly, the hardest part of Chef InSpec is understanding its rather complex terminology, setting up an equally complex directory structure for the "profile" in which you will define the things that makes your environment "compliant," and the mere fact that there's not a lot of good documentation available on how to use or troubleshoot it.

Fortunately, I've suffered through most of that for you, so once you get things running with the sample code I provide via Canvas, this Lab shouldn't be terribly difficult. It will still be somewhat time-consuming though, as our pipelines have become quite large this semester and take about 15 minutes to run each time. You should also dedicate additional time to reviewing the terminology slide of the PowerPoint deck, as you will be questioned on the Final Exam about the various definitions of Chef InSpec terms.

Before you begin:

- Your GitLab Runner instance should be started, if you previously stopped it.

What you'll achieve:

- Create a Chef InSpec profile, including a YAML file and directory structure containing the controls that will be used to test your AWS environment for compliance;
- Add a Chef InSpec job to your GitLab CI/CD file, allowing the Chef InSpec Docker container to read your profile and execute the tests against your AWS account;
- Tailor your Chef InSpec controls to your AWS environment, ensuring that we get a successful job completion; and
- Finally, deliberately modify your controls to encourage a job failure, enabling you to see how the pipeline will be stopped if compliance is not achieved.

What you'll turn in with this lab:

- Simply send in a text submission via the Canvas assignment page stating "Done." The professor will check your GitLab repo for successful implementation of the necessary Chef InSpec code.

Lab begins on next page.

Task 1: Create a Chef InSpec Profile

As discussed in the PowerPoint deck, a Chef InSpec “profile” consists of basic information about the compliance tests you will run against your cloud environment, including things like “controls” and “libraries.” This is not something we can create in any sort of GUI, but rather something that we need to build out ourselves with what can often be a convoluted directory structure. Fortunately, for our relatively simple use case in this Lab, we will only build out the bare minimum requirements for a Chef InSpec profile.

1. In VS Code and in the Explorer tab for your repo, create a new directory for your Chef InSpec profile. This can be named anything, but I suggest “lab-7-profile” (without quotes).
2. Within your profile directory, create a file called “inspec.yml” (without quotes).
3. Download the sample Chef InSpec YAML code from this assignment’s page on Canvas. Copy and paste this code into your “inspec.yml” file, then save and close it.
4. Within your profile directory, create another directory called “controls” (without quotes).
5. Within the controls directory, create a file called “lab-7-controls.rb” (without quotes).
6. Download the sample Chef InSpec Controls Ruby code from this assignment’s page on Canvas. Copy and paste this code (only up to the comment line where you’re instructed to stop) into your “lab-7-controls.rb,” then save it.

Task 2: Tailor Chef InSpec Controls to Your Environment

The Chef InSpec Controls that are included in the “lab-7-controls.rb” file depend on the AWS API being able to find an instance by a provided Name tag. For this Lab, we’ll run a number of tests against our Terraform-created instance from Lab 3 (or as part of your Pipeline Project), however, each of you have probably named that instance something slightly different. We need to update our controls file so that the “names” we are asking the AWS API to describe to Chef InSpec match the actual names of instances in our AWS account.

7. In VS Code, open your “main.tf” file. *Don’t* make any changes, but do *locate* the Terraform resource stanza for the original Terraform instance you created in Lab 3 (or your Pipeline Project). Take note of the Name tag you assigned to it, as well as the AMI ID. You’ll need these later on.
8. Log in to the AWS Management Console, and navigate to your EC2 Instances table. Verify this same information with the instance details shown in the Console.
9. In VS Code, return to your “lab-7-controls.rb” file. Make sure to modify the “name” tag in each control’s “describe” blocks to reflect the Name tag assigned to your instance. Additionally, if your instance is currently built with a different AMI than AmazonLinux (this may have been assigned to you as part of your Pipeline Project), update the AMI ID that is being sought in the “check-instance-ami” control to reflect the AMI your instance uses.

Task 3: Add Job, Region Variable & Run Pipeline

10. Download the sample GitLab code from this assignment’s page on Canvas. Add the “chef-inspec” job to your “.gitlab-ci.yml” in VS Code. Save this file, but don’t commit or push yet.

Chef InSpec requires that the AWS_REGION environment variable be set, limiting its scope to one AWS Region at a time. Fortunately, we can set this in our GitLab CI/CD Variables, since we only use the Northern Virginia region for all of our Labs – this won’t impact any of our upstream jobs.

11. Log into GitLab. Navigate to Settings > CI/CD > Variables.
12. Add a new variable with a key of “AWS_REGION” (without quotes) and value of “us-east-1” (without quotes). Remember to uncheck the protect/mask variable boxes.
13. In VS Code, make sure all of your files are saved. Commit and push them.

Your pipeline will run, and, as always, require you to manually trigger the tf-apply job and then take quite a bit of time to run through our numerous jobs.

14. Once your pipeline gets to the “Continuous Compliance” stage, click on the “chef-inspec” job to examine its terminal output. Observe that the job completes successfully because all of the tests passed.

Task 4: Add a Control That Will Fail (And Fix It)

Our tests are currently passing because we’ve verified that the information being sought by our control tests is the actual information that exists in our AWS account. However, let’s experiment with how Chef InSpec reacts when a test fails.

15. Return to the sample Ruby code from step 6. You may now copy and paste the control block (below the comment you previously stopped at) into your “lab-7-controls.rb” file. This code looks for a tag on your EC2 instance with a key of “Lab7” and a value of “completed” (all without quotes).
16. Save, commit and push your changes. When your pipeline returns to the “chef-inspec” job again, observe the terminal output.

Notice how the job (and your pipeline) now fails. The terminal tells you that it expected to see that key-value tag on the instance, but it was not able to find that tag at all – it only found your Name tag. Now, let’s add that tag with the Lab7 key, *but* with a value that Chef InSpec is not expecting.

17. In VS Code, open your “main.tf” file and locate the tags block for this EC2 instance resource. Add a comma after the key-value pair for your Name tag, then drop down a

line, and add another key-value pair that sets a key of “Lab7” (without quotes) with a value of “in-progress”. Do not add another comma. Save, commit and push.

Your pipeline will run again, and the job will still fail. This time, it tells you that it saw the Lab7 tag, but it saw a value of “in-progress” rather than “completed” as it was expecting. Let’s go ahead and fix our instance’s tag to make Chef InSpec happy again.

18. Return to your “main.tf” file in VS Code. Update the code for the Lab7 tag to now have a value of “completed” (without quotes). Save, commit and push.

This time, when your pipeline gets to the “chef-inspec” job, it should pass. Examining the terminal output should reveal a very happy Chef InSpec, because the tag’s value now matches the expectations of the test.

That’s pretty much the Lab! Continuous Compliance isn’t super hard, and Chef InSpec isn’t that hard to use once you get used to the way it operates, but these controls and tests can get very complicated. Just like how a large company will have more complicated Terraform files that deploy hundreds of instances and complex Ansible files to configure all of those instances, it would be completely normal to have Chef InSpec profiles with hundreds of tests, making sure that all of the IT security protocols for the organization are being followed by not just the properties of your AWS infrastructure, but also the operating systems themselves.

Please make a text submission of “Done” via Canvas to let me know you’ve completed the Lab. I will then check your repo/pipeline for satisfactory completion.

Read on if you’d like to earn some extra credit…

Extra Credit Opportunity: Add an Additional Control

The Achilles’ heel of Chef InSpec is that the various resources it can interact with are not well-documented. As such, I will offer one (1) point of extra credit on this Lab (for a total of 6 possible points, although 5 points still equals 100%) for any student who is able to flex their Google muscles and track down how to add an additional control to their Ruby file that will run another test against your AWS environment. **This is meant to be difficult (it’s extra credit, after all) and I cannot assist you!** This is your opportunity to demonstrate that you can research – or even ask around on sites like Reddit or StackOverflow – how to complete a task without any oversight, a very important skill in the cutting-edge DevSecOps and cloud computing industry!

To receive this extra credit, add an additional control to Chef InSpec that checks for the AWS Access Key you created in Lab 3, Task 2, and ensures that *it has been used within the last 60 days.*