Seneca

Final Project: Two-Tier web application automation with Terraform

Submission Instructions	To be submitted via Blackboard. Refer to Blackboard for submission instructions
Value	25% of final grade
Due Date	

Learning Outcomes Covered in Assignment

1. Explain benefits of DevOps practices and deployment automation to support organizational shift towards DevOps culture
2. Analyze core infrastructure requirements to create a deployment automation design for base cloud components
3. Analyze application's functional and operational requirements to recommend software development lifecycle components
4. Apply Infrastructure as a Code approach to deploy all parts of the application hosting solution in a repeatable and reliable way
5. Support the security posture of the cloud infrastructure by identifying and implementing preventive and detective security controls

1. Assignment Outline

The objective of this assignment is to verify your skills in applying deployment automation, configuration management, and source control tools to create two-tier static web application hosting and configuration solution.

In addition to your knowledge of Terraform, the assignment verifies your understanding of scalable cloud architecture using load balancers and autoscaling groups, Identity, and Access management in AWS, and efficient use of source control and GitHub Actions to create security scanning automation.

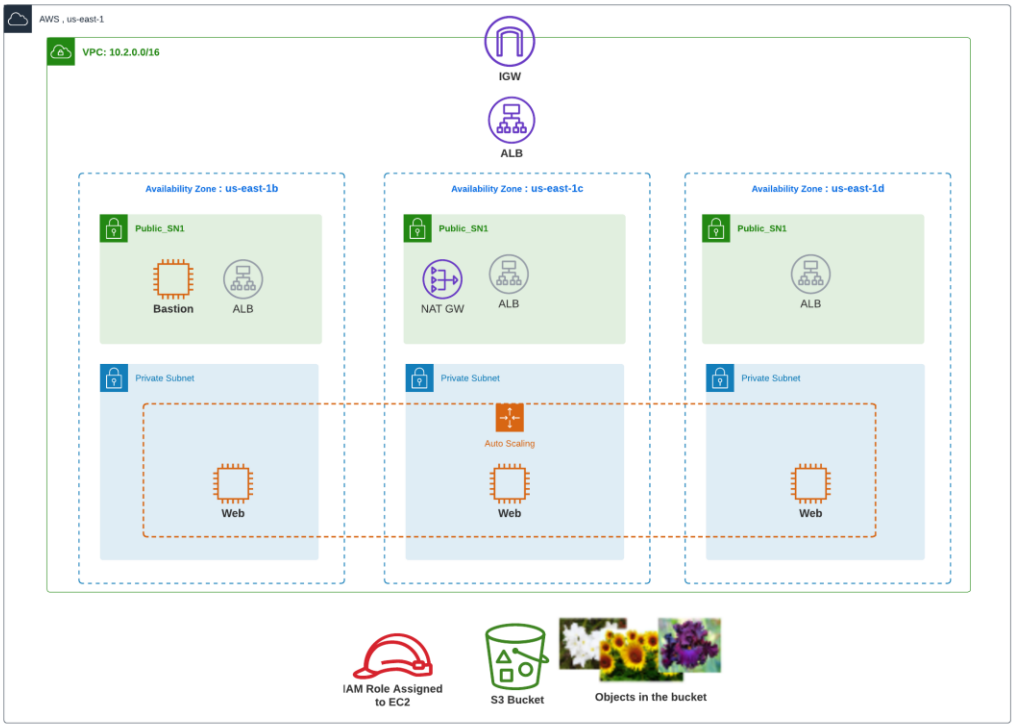
In the scope of the Final Project for ACS730, you will create Terraform configuration as specified in Section 2, Architecture, GitHub project that will host your Terraform code and it's change history. To make sure the Terraform configuration that you create complies with the security standards, the code should undergo static code analysis and security validation as part of the git-flow.

The Terraform configuration should be modular, have a consistent naming convention and tagging strategy.

Each completed task should be supported by the artifacts outlined in the submission requirements.

Please read the submission requirements carefully. This is a group assignment and the git commits are the best way to evaluate the individual contributions of all the team members.

2. Architecture



The application will serve a static website with the images downloaded from **your S3 bucket.** The application can be based on this application or you can create your own web application/use a static httpd-based website.

The application will be deployed on an Auto-Scaling Group (ASG) of EC2 instances with a minimum of **1** and a maximum of **4** instances across **3** availability zones.

The solution should have a scaling policy to **scale out if the load is above 10% of the CPU** and **scale in if the load is below 5% of the CPU**.

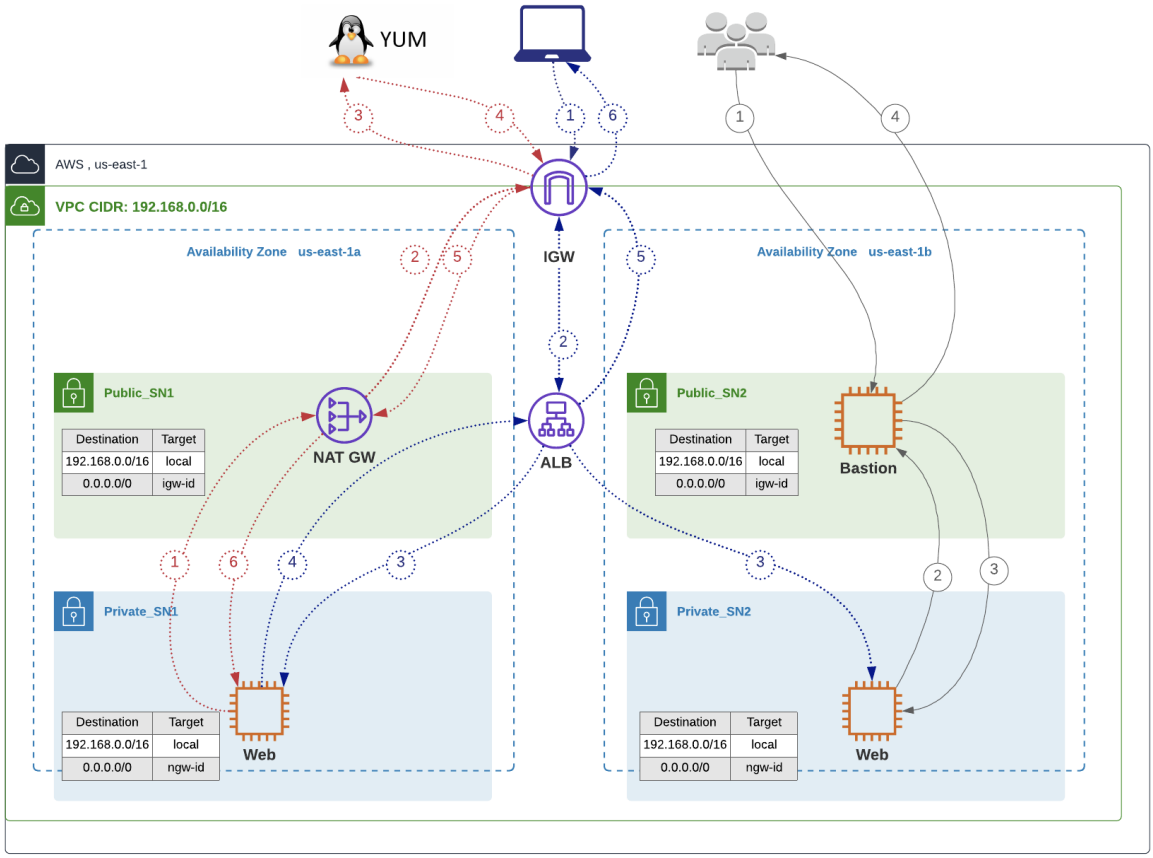
Environment	CIDR	Number of Instances	Instance Type

Dev	10.100.0.0/16	2	t3.micro
Staging	10.200.0.0/16	3	t3.small
Prod	10.250/0.0/16	3	t3.medium

All environments have the same architecture apart from CIDR ranges of respective VPCs (Virtual Private Clouds) and the minimum number of instances in ASG and instance sizes. The sizes of all subnets are identical; each subnet should have 256 IP addresses.

1. Each environment should use its own S3 bucket to store Terraform state and images.
2. The webpage should reflect the name and the names of the team members.
3. Resources names should reflect the name of the environment and the name of the group, e.g., "Group1-Dev-VPC where possible.
4. Use Camel case in your naming convention across all the resources.
5. You can create the S3 bucket and upload the images for your website via CLI or AWS Management Console.
6. Do not forget to clean the submission from the commented-out lines, .terraform folder, and **do not upload your SSH (Secure Shell) keys to the public GitHub repository.**

3. Traffic Flows



4. Evaluation

Please see the evaluation breakdown on the last page.

5. Submission

Your submission should include the artifacts below:

1. Link to **GitHub repository** that hosts your **deployable** Terraform configurations and is configured with the items below:

a. Frequent commits done by all team members. If there are no commits associated with your GitHub user, I will assume that you did not take part in the implementation of the Architecture outlines in Section 2.
b. User **igeiman13** has Contributor permissions in the project repo
c. All the team members have either Admin or contributor permissions
d. GitHub Actions are used to perform security scan on each push to staging branch and when pull request is open to merge code into prod branch

2. **Report** with the content below:
    a. Explanation the traffic flows depicted in the Traffic Flows section
        i. Red flow
        ii. Grey flow
        iii. Blue Flow
    b. Mapping of the team members to their GitHub users
    c. Screenshot of the webservers serving traffic behind the Load Balancer. The webserver should showcase an image loaded from an S3 bucket.
    d. Conclusion outlining the challenges you faced when implementing the requirements, the way you solved the problems, and the new things that you learned thanks to this assignment.

3. **Recording** that showcases the deployment process that results in functional website deployed in dev, staging and prod environments.

Note: make sure to stop the recording when the deployment is running. You can pre-deploy all the infrastructure and change README file only so the deployment will be

quick. Ideally, the recording should be 15 min long. Recordings longer than 30 minutes will not be accepted.

6. Submission Requirements Description

6.1 GitHub Repo

Task	Submission Requirements Description
README	The README file should clearly explain all the pre-requisites for the successful deployment (e.g. S3 bucket that stores Terraform state and image displayed by the webservers. Image should be uploaded manually) and the deployment process. README should provide instructions for successful deployment and cleanup of the provided solution. Note: keep this file concise, this is the a report.
Terraform configuration for prod, dev and staging environments	The Terraform configuration should be parametrized and modularized. Avoid code repetitions. Consider creating modules for networking, ALB, Launch templates, SG, ASG. Make sure your code supports different number of webservers in different environments.

	The solution should use remote Terraform state . The deployed webserver should load an image from S3 bucket, please see an Appendix. The S3 bucket cannot be public.
GitHub Actions for security scan automation	Use any combination of the tools below to perform security scans on each push and on pull requests to prod. • trivy • tflint

6.2 Report

Task #	Submission Requirements Description
	Title Page Your name, course, section, date, the professor's name.
	Mapping of the team members to their GitHub users .
1	Explain the traffic flows depicted in Section 3, Traffic Flows. Have a separate explanation for purple, grey and pink flows. In your explanation, refer to the following questions: • At what stage of application lifecycle will this flow happen? • How is this flow triggered? • How/what are the users in this flow?

	• What kind of traffic is it (web, systems administration etc.) ?
3	About one page of free text outlining the challenges you faced when implementing the requirements, the way you solved the problems, and the new things that you learned thanks to this assignment.

6.3 Recording

Task #	Submission Requirements Description
	Introduction (~ 1 min) Describe in your own words the main objectives of this assignment
1	Demonstrate that dev, staging and prod environments can be successfully deployed **using Terraform cloned from your GitHub repo.**
2	Demonstrate that the webservers are successfully loading behind the load balancer and all instances are showing as healthy in the AWS Console
3	Explain the rationale behind the use of Load balancer with the Auto-scaling group.
4	Demonstrate that your solution is highly available. Simulate server failure by stopping one of the webservers hosting your application and show that there is no impact on the application availability and users can still access it.

7. Plagiarism:

Plagiarized assignments will receive a mark of zero on the assignment and a failing grade on the course. You may also receive a permanent note of plagiarism on your academic record.

8. Assignment Grade Breakdown

Section/Task		**Points**
GitHub Repo	All users defined with the adequate permissions	2
	Frequent commit with clear added value	10
	GitHub Actions to perform security scans	10
	Prod Branch protection	3
Report	Traffic flows explained	10
	Challenges faced during implementation	5
Terraform code	README	3

	Multi-environment implementation	2
	Modules for Launch Configuration, Target Group, ALB, SG, EC2s and networking	25
	Loading image(s) from S3	5
	Auto-scaling implementation	10
	Important Note: the code that cannot be deployed based on the instructions provided in the README will result in grade 0 for the Terraform Code section. As a result of the deployment, your code should print out the DNS that should be used to access the static website.	
Recording	Demonstrate that dev, staging and prod environments can be successfully deployed using Terraform cloned from your GitHub repo	5
	Demonstrate that the webservers are successfully loading behind the load balancer and all instances are showing as healthy in the AWS Console	3

	Explain the rationale behind the use of Load balancer with the Auto-scaling group.	5
	Demonstrate that your solution is highly available. Simulate server failure by stopping one of the webservers hosting your application and show that there is no impact on the application availability and users can still access it.	2
	Total	**100**

9. Recommended Implementation Flow

1. Setup your GitHub repo to enable collaboration. Created protected branches, choose reviewers.
2. Define branching strategy and git flow
3. Agree on naming and tagging approach, document it for the report
4. Manually create ALB with ASG via AWS Console, make sure you understand the way it works
5. Setup GitHub actions with security scans to start scanning early in the development process
6. Start Terraforming
    a. Create relevant S3 buckets for your environments
    b. Implement networking module
    c. Implement ALB

   d. Implement Launch configuration
   e. Implement Listener
   f. Implement scaling policies
7. Deploy multiple environments, verify the website is loading successfully in all of them

Useful Links

Git and GitHub tutorial

GitHub Actions Documentation

AWS Workshop "Running EC2 Workloads at Scale"

Implementing Auto-Scaling with Terraform

AWS resources naming convention for Terraform deployment