

Final Project: Two-Tier web application automation with Terraform, Ansible and GitHub Actions

|  |  |
| --- | --- |
| **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 usage of Ansible Playbooks for Configuration Management with in AWS, and efficient use of source control and GitHub Actions to automate entire Terraform security scan and deployment.   * In the scope of the Final Project for ACS730, you will create **Terraform configuration as specified in Section 2, Architecture, to provision 6 VMs across 4 Public and 2 Private Subnets in different availability zones (Out of 6 VMs, 5 of them will be Webservers)**. Then you will use **Terraform to deploy Webservers in the first two VMs (Webserver 1 & 2) in the Public subnet (which is accessible by SSH and HTTP)**. You can also bring these two as part of Application Load Balancer (ALB) and Auto Scaling Group. * In addition, Webserver 2 should act as Bastion host to access the VMs (#5 and 6) in Private subnets. * You enable NAT GW in Public Subnet 1 and connect VM 5 from private subnet to access internet via this NAT GW (and make this VM as a Webserver – Download Apache package from Internet) * Now you can use Cloud 9 (or one of the provisioned Webservers) to install Ansible and use that machine as Control Host to connect to remaining 2 VMs in the Public Subnet and enable them as Web servers (Webserver 3 & 4). You would use Ansible Playbooks and Dynamic Inventory to configure the remaining WebServers (3 and 4) and perform other requried configuration checks (Check the connectivity, Check the service status, and update the patch)   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 and automate entire Terraform Deployments via GitHub Actions (Provision of VPC, all 6 VMs, NAT GW and Enable first two VMs as Webservers). Note: GitHub/Actions are mainly for Terraform config and deployment. Hosting Ansible playbooks in GitHub is optional.  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. |
|  |

1. Architecture

A diagram of a network

Description automatically generated

The application will serve a static website with the images downloaded from **your S3 bucket.** The application can be [based on this application](https://github.com/jimini55/catsdogs-cloud9) or you can create your own web application/use a static httpd-based website.

Note: If the application is deployed on an Auto-Scaling Group (ASG), you could use EC2 instances with a minimum of **1** and a maximum of **4** instances across 2 or **3** availability zones. You can have your own scaling policy to **scale in and/ out**.

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.**
7. Traffic FlowsA diagram of a network

   Description automatically generated
8. Evaluation

Please see the evaluation breakdown on the last page.

1. 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:    1. 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.    2. User **bhargavamuralidharan** has Contributor permissions in the project repo    3. All the team members have either Admin or contributor permissions    4. GitHub Actions are used to perform security scans on each push to staging branch and when pull request is open to merge code into prod branch. Also, GitHub Action is used to Automate future deployment of Terraform code for provisioning the Infrastucture resources and the first two Webservers 2. **Report** with the content below:    1. Explanation the traffic flows depicted in the Traffic Flows section       1. Red flow       2. Blue Flow    2. Mapping of the team members to their GitHub users    3. The webserver should showcase an image loaded from an S3 bucket.    4. 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 project. 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 not more than that. |

1. 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 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.  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](https://github.com/aquasecurity/trivy-action) * 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.  In your explanation, refer to the following questions:   * 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 different Webservers can be successfully deployed **using Terraform cloned from your GitHub repo.** |
| 2 | Demonstrate the use of Dynamic Inventory for Ansible Deployments |
| 4 | If you enable ALB, explain the rationale behind the use of Load balancer. 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. |

1. 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. |

1. 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 |
| GitHub Actions to automate Terraform future deployments | 5 |
| Report | Traffic flows explained | 10 |
| Challenges faced during implementation | 5 |
| Code | README | 3 |
| Multi-environment implementation | 2 |
| Modules for Launch Configuration, SG, EC2s and networking (and ALB) | 20 |
| Loading image(s) from S3 | 5 |
| Ansible Playbooks for WebServer Deployment and Config management | 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. |  |
| Recording | Demonstrate that WebServers are deployed using Terraform | 5 |
|  | Demonstrate the use of Dynamic Inventory and WebServers are deployed using Anisble | 5 |
|  | Bastion Host access to VMs and Configuration Mangement via Ansible | 5 |
|  | **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. If you plan to use ALB, you can 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
   1. Create relevant S3 buckets for your environments
   2. Implement networking module
   3. Implement Launch configuration
7. Deploy multiple environments, verify the website is loading successfully in all of them
8. Create Ansible Playbooks from Cloud 9 or one of the VMs. Use Boto3 and use right Python version for Dynamic inventory

Useful Links

[Git and GitHub tutorial](https://product.hubspot.com/blog/git-and-github-tutorial-for-beginners)

[GitHub Actions Documentation](https://docs.github.com/en/actions)

[Ansible Dynamic Inventory](https://docs.ansible.com/ansible/latest/collections/amazon/aws/aws_ec2_inventory.html)

[AWS Workshop “Running EC2 Workloads at Scale”](https://ec2spotworkshops.com/running-amazon-ec2-workloads-at-scale.html)

[Implementing Auto-Scaling with Terraform](https://medium.com/geekculture/how-to-manage-auto-scaling-group-and-load-balancer-with-terraform-9ece263060b5)

[AWS resources naming convention for Terraform deployment](https://www.terraform-best-practices.com/naming)