**Seneca College**

**[School of Information Technology Administration & Security](https://www.senecacollege.ca/school/information-technology-administration-and-security.html)**

**Cloud Automation and Control Systems ACS730**

**Group Project Report**

**Traffic Flows and Implementation Challenges**

|  |  |
| --- | --- |
| **Name** | **ID** |
| **Asif Raja** | **142487222** |
| **Harniaz Brar** | **117781229** |
| **Mohd Danish Khan** | **100037225** |

**Submitted to: Dhanasekaran Karuppusamy**

**Mapping of the team members to their GitHub users:**

• Mohd Danish Khan – dkhan0103

• Harniaz Brar – Harniaz-Brar

• Asif Raja – brtasif

**Explain the traffic flows depicted in Section 3, Traffic Flows for Below Diagram**

**A picture containing diagram

Description automatically generated**

This project involves creating a two-tier static web application hosting and configuration solution using Terraform, Ansible Playbooks, GitHub Actions. The architecture of the application includes 6 VMs across 4 Public and 2 Private Subnets in different availability zones, with Web Servers deployed in the first two VMs in the Public subnet using terraform, and the remaining Web Servers configured using Ansible Playbooks and Dynamic Inventory.

Note: Infrastructure deployment has been done using Terraform IaaC

**The Red Flow**

The Red Traffic Flow depicted section is a series of tasks executed using Ansible Playbook to deploy a static website on the VM instances. The Ansible Playbook includes several tasks to facilitate the deployment process.

The first task involves installing and starting the Apache web server on the instances using the Ansible module. This task ensures that the web server is up and running and ready to serve the website content.

The second task involves deploying the static website using Ansible Playbook and dynamic inventory. In this task, the Playbook extracts the static code from an S3 bucket and deploys it on the instances using the Ansible module. This task ensures that the website content is available on the instances.

The Ansible Playbook also includes additional tasks to provide further insights into the deployment process. These tasks include displaying the uptime and printing the Linux family on the instances. These tasks help to ensure that the instances are healthy and running the desired operating system.

Furthermore, the Ansible Playbook copies the static website code to the host, starts the Apache web server, and prints the result for uptime. These tasks ensure that the website content is accessible to the users and that the instances are running optimally.

To aid in troubleshooting and debugging, the Ansible Playbook also includes a debug message. This message provides valuable insights into the execution of the tasks and any issues that may arise during the deployment process. The debug message can be used to quickly identify and resolve any issues, thereby ensuring that the deployment process is smooth and efficient.

**How is this flow triggered?**

When we run the ansible playbook using ansible-playbook command

Ansible Host (Cloud9 IDE) >>> Access to EC2 instance over port 22

**How/what are the users in this flow?**

User is the system in which we have installed the ansible and aws\_ec2 plugins, cloud9 IDE in this case.

**What kind of traffic is it (web, systems administration etc.)?**

It is a system administration traffic on port 22.

**The Blue Flow**

The Blue Traffic Flow depicted section of the provided image has been implemented using Terraform. The infrastructure deployed consists of a Virtual Private Cloud (VPC) with four availability zones, four public subnets, and two Private subnets. Additionally, there are four web servers, with one web server acting as the bastion host to connect to the private EC2 instances using SSH.

The static website for the project has been deployed on two of the web servers using Terraform. The code and images for the static website are hosted on an S3 bucket, and the user data feature is used to start and install the Apache HTTP server on the web servers. Furthermore, the user data feature is also utilized to extract the code and images from the S3 bucket and display them on the website.

It is important to note that an Internet Gateway has been deployed in this flow to allow for connectivity to and from the internet. The bastion host provides an added layer of security by limiting access to the private EC2 instances through SSH access only. In addition, the IaaC for Terraform is modularized and can be reused.

Overall, the Blue Flow has been successfully implemented using Terraform, with a focus on security and automation using the bastion host. The static website has been deployed and is accessible to users through the internet.

**How is this flow triggered?**

When any user from try to access web servers over internet on port 80

Internet User >> AWS VPC>>> IGGW >>> EC2 Public IP >>>> on port 80

When any user tries who has the private key to SSH into web servers using port 22

Internet User >> AWS VPC>>> IGGW >>> EC2 Public IP >>>> on port 22

When the bastion host having the private key try to access private web servers on port 22

Bastion Host >>> Access to Private EC2 >>>> on port 22

**How/what are the users in this flow?**

* Any user over internet for HTTP access
* Any user with private key for SSH access
* Bastion host with private key

**What kind of traffic is it (web, systems administration etc.)?**

* It is a system administration traffic on port 22.
* It is a web traffic when servers are accessed on port 80.

**Challenges Faced**

Completing a project can often be a challenging task, and there can be several hurdles that one needs to overcome to ensure its success. In this case, there were several challenges faced while completing the project, which are discussed below.

The first challenge was related to IAM issues, which prevented EC2 from accessing the S3 bucket. IAM is an AWS service that enables the management of user access to AWS resources.   
As per Seneca AWS subscription we are not allowed to create IAM roles.

The second challenge was related to getting the static website code and images from S3 to EC2. This was resolved by writing playbooks to extract the code and images from S3 and transfer them to EC2.

The third challenge was automating these processes using user data. This was resolved by configuring user data scripts that would run on EC2 instances to automate the process of extracting code and images from S3.

The fourth challenge was tweaking trivy and tfsec GitHub actions according to the project's needs. Trivy is a vulnerability scanner for containers, while tfsec is a static analysis tool for Terraform code. Tweaking these actions ensured that the project met the required security standards.

The fifth challenge was creating a GitHub action for Terraform deployment that could only be triggered when the approver clicked on the merge request button  
Generally the deployment happen whenever there is a push or pull\_request.  
But this is not optimal since the code is getting deployed without doing

This was achieved by configuring GitHub Actions to trigger the Terraform deployment workflow only when a merge request was approved by a specific user.

In conclusion, completing a project can be challenging, but with proper planning and execution, these challenges can be overcome. The challenges faced in this project were related to IAM issues,automating processes, transferring code and images, and tweaking GitHub actions. By resolving these challenges, the project was completed successfully.