



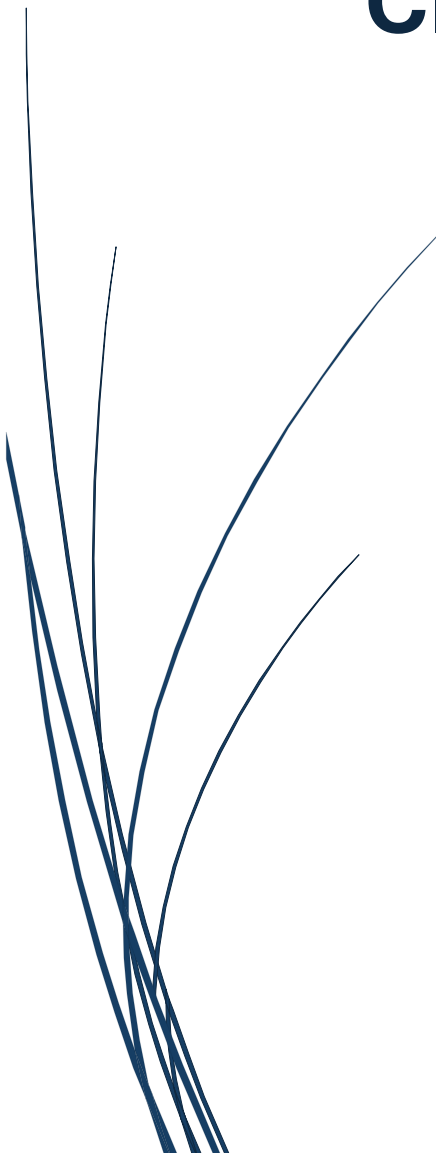
HashiCorp

Terraform



3-Tier Architecture with CI/CD Deployment

Project Documentation



William Gan

Table of Contents

1. Executive Summary	5
1.1 Introduction	5
1.2 Problem Statement	5
1.3 Project Goals and Expected Outcomes	5
2. Project Constraints.....	6
2.1 Cost Management for Tools, Platforms, and Services	6
2.2 Skill and Knowledge Requirements	9
3. Project Environment	10
3.1 Project Environment Overview.....	10
3.2 Project Environment Setup.....	11
3.2.1 Setting up the HCP Workspace	11
3.2.2 Configuring Terraform Provider	12
3.2.3 Authenticating Terraform with HCP	13
3.2.4 Initializing Terraform Environment	14
3.2.5 Authenticating HCP with AWS Cloud.....	15
3.3 Repositories, Access Endpoints, and Terraform Files.....	16
3.3.1 Repositories.....	16
3.3.2 Access Endpoints	16
3.3.3 Terraform Files	17
4. Project Scope.....	19
4.1 Full System Architecture Diagram.....	19
4.2 Full System Architecture Breakdown.....	20
4.3 Overview of Frontend System Architecture	21
4.4 Overview of Frontend CI/CD Pipeline Architecture	22
4.5 Overview of Backend System Architecture	23
4.6 Overview of Backend CI/CD Pipeline Architecture	24
5. Frontend Systems Implementation	25
5.1 Creating TLS Certification for CloudFront	25
5.2 Creating a CloudFront Distribution	26
5.2.1 CloudFront Origin Access Control.....	26
5.2.2 CloudFront Distribution Configuration	27
5.3 Creating an ALIAS record for CloudFront Distribution	32
6. Frontend CI/CD Pipeline Implementation	33
6.1 AWS CodeBuild - Source Credentials.....	33
6.2 AWS CodeBuild – Web Hook	33
6.3 AWS CodeBuild – Project Build Configuration	34
6.3.1 AWS CodeBuild – Main Configuration Details	34
6.3.2 AWS CodeBuild – Source Configuration	34
6.3.3 AWS CodeBuild – Environment Configuration	35

6.3.4	AWS CodeBuild – Artifacts Configuration	36
6.3.5	AWS CodeBuild – Logging Configuration	36
6.4	AWS CodeBuild – Trigger Upon Creation	37
7.	Backend Systems Implementation	38
7.1	VPC and Internet Gateway	38
7.2	Subnets	39
7.3	Public Route Table and Association	40
7.4	Application Load Balancer (ALB).....	41
7.5	Creating a TLS Certification for Listener	42
7.6	Configuring a Listener	43
7.7	Target Group	44
7.8	Auto-Scaling Group	45
7.9	Creating an ALIAS record for ALB	46
8.	Backend CI/CD Pipeline Implementation.....	47
8.1	AWS CodeDeploy Application Type	47
8.2	AWS CodeDeploy Deployment Group	48
8.3	AWS CodePipeline.....	49
8.3.1	AWS CodePipeline - Source	50
8.3.2	AWS CodePipeline - Deploy.....	51
9.	Project Evaluation	53
10.	Future Plans	54
10.1	Multi-Region Backend	54
10.2	Logging and Monitoring.....	54
10.3	Security	54
11.	Ethical Considerations	54
12.	Appendices	55
12.1	Appendix A: S3 Bucket Creation	55
12.1.1	S3 Bucket Public Access Block	56
12.1.2	S3 Bucket Policy for Origin Access Control	57
12.2	Appendix B: IAM Role for CodeBuild.....	58
12.2.1	CodeBuild IAM Role Creation	58
12.2.2	CodeBuild IAM Role Policies.....	59
12.3	Appendix C: Security Group Rules for ALB	60
12.4	Appendix D: EC2 Launch Template	61
12.4.1	IAM Instance Profile for EC2	62
12.4.2	EC2 Security Group Rules	63
12.4.3	EC2 userdata.sh Boot Script.....	64
12.5	Appendix E: GitHub Apps Connector	65
12.6	Appendix F: IAM Role for CodePipeline.....	69
12.6.1	CodePipeline IAM Role Creation	69
12.6.2	CodePipeline IAM Role Policies	70

12.7	Appendix G: IAM Role for CodeDeploy	71
12.7.1	IAM Role Creation for CodeDeploy	71
12.7.2	IAM Role Policies For CodeDeploy Role	72
12.8	Appendix H: Route 53 Hosted Zone.....	73
12.9	Appendix I: Validating Certificates with Route 53	74
12.9.1	Adding Certificate DNS Validation Records to Route 53	74
12.9.2	Verifying the Certificates Validity.....	75

1. Executive Summary

1.1 Introduction

This project focuses on harnessing the benefits of Infrastructure as Code in automating the provisioning of cloud infrastructure onto Amazon Web Service, specifically through infrastructure provisioning tools such as Terraform with a mixture of shell scripts, and AWS CLI to compliment CI/CD workflows.

1.2 Problem Statement

Manual cloud provisioning is prone to errors and inefficiencies. IaC ensures consistency across environments, significantly reduces deployment time, and facilitates the replication of infrastructure between cloud accounts.

The benefits of using IaC include enhanced reliability, ease of configuration, and the ability to easily replicate the provision of infrastructure setups. These benefits are particularly valuable for both enterprise and educational purposes.

For individuals working on expanding their own cloud projects, using IaC allows individuals to easily spin up or take down an entire system with a single *terraform apply* or *terraform destroy* command, which is an effective way to control costs within a pay-per-use cloud model.

For organizations, these benefits translate into reduced risk, increased operational efficiency and replication across different environments.

1.3 Project Goals and Expected Outcomes

This project aims to deploy the following using Terraform onto AWS Cloud:

- A Scalable and Highly Available 3-Tier Ecommerce Web Application.
- CI/CD Pipeline for the Backend.
- CI/CD Pipeline for the Frontend.
- Reasonably secure configuration within constraints.

2. Project Constraints

2.1 Cost Management for Tools, Platforms, and Services

This project leverages a comprehensive suite of tools, platforms, and AWS services to ensure a cost optimized and scalable infrastructure. Table 1, Table 2, and Table 3 is an overview of each area, including the role in the project and associated pricing details. AWS offers a free tier for many services, which can be leveraged during initial project development. Services such as the AWS NAT Gateway were omitted due to high costs.

Tools	Description	Pricing
Git	Version control for managing Terraform configurations and application code.	Free
Terraform	Core IaC tool for defining and automating cloud resource provisioning.	Free
Visual Studio Code	IDE for writing and testing Terraform configurations and deployment scripts.	Free

Table 1 - Cost Management Overview for Tools

Platforms	Description	Pricing
GitHub	Hosts Terraform configuration files and application code for version control and collaboration.	Free
Terraform Cloud (HCP)	Centralized platform for managing Terraform executions and securely storing infrastructure state.	Free with limitations
AWS Cloud	Cloud provider for hosting and managing infrastructure components.	Free tier available (Not everything)

Table 2 - Cost Management Overview for Platforms

AWS Services	Description	Cost (USD)
Route 53 (Domain Registration)	Domain Name Registration to use with both frontend endpoint, and backend api endpoint.	\$14
Route 53 (Hosted Zones)	A hosted zone is a container for DNS records for a specific domain.	\$0.50 per hosted zone / month
AWS Certificate Manager	Issues and manages SSL/TLS certificates for secure HTTPS communication.	Free
AWS Secrets Manager	Stores sensitive information like API keys, providing secure access for EC2 instances.	\$0.40 per Secret per month. \$0.05 per 10,000 API calls.
IAM Roles & Policies	Manages roles, policies, and permissions to ensure secure access to AWS resources.	Free
S3	Stores static assets and CI/CD pipeline artifacts.	Free within limits
CloudFront	Content Delivery Network (CDN) for distributing static assets with low latency.	Free within limits
VPC	Provides a virtual private cloud for isolating and securing resources.	Free
Internet Gateway	Facilitates outbound internet access for instances in public subnets.	Free
Subnets	Segregates resources into public subnets across availability zones.	Free
Route Tables	Directs network traffic between subnets and external resources.	Free
EC2	Virtual servers for hosting backend applications and supporting Auto Scaling Groups.	Free within limits
Application Load Balancer	Distributes traffic across multiple EC2 instances for high availability and scalability.	\$0.005/hour for IPv4 address

Target Groups	Directs traffic from the ALB to specific EC2 instances based on health checks.	Free
Auto Scaling Groups	Dynamically adjusts the number of EC2 instances based on traffic demand, ensuring scalability.	Free
CodePipeline	Automates the deployment process for frontend and backend applications.	\$0.002 per action execution minute.
CodeDeploy	Manages application updates across EC2 instances with support for various deployment strategies.	Free when deploying to EC2
CodeBuild	Builds and packages application code before deployment to S3 or EC2 instances.	\$0.00008 per build seconds

Table 3 - Cost Management Overview for AWS Services

2.2 Skill and Knowledge Requirements

Skill	Description
<u>Terraform (IaC Tool)</u>	Proficiency in Terraform for IaC is essential, following configuration practices from the official Terraform documentation.
<u>Cloud Architecture Design</u>	Understanding of cloud architecture patterns, tiered architecture is used in this project.
<u>AWS Services</u>	Familiarity with AWS Cloud services such as EC2, S3, Route 53, IAM, and ELB is absolutely a fundamental to execute the project.
<u>Networking Concepts</u>	Knowledge of VPCs, subnets, route tables, security groups, and DNS configuration.
<u>CI/CD Pipeline Configuration</u>	Skills and Knowledge in understanding and creation of CI/CD pipelines.
<u>Application Configuration</u>	Applications need to be configured for deployment, including setting up for CI/CD pipeline.
<u>Shell Scripting</u>	Ability to write user data scripts in bash/yaml for automating server configuration during instance startup or for the configuration of CI/CD pipelines.
<u>Basic Secure Practices</u>	Understanding of basic cloud security, including IAM roles, policies, security groups, and managing secrets.
<u>Version Control with Git</u>	Use of Git for versioning Terraform code and collaborating on infrastructure changes.
<u>Cost Management</u>	Strong awareness of managing cloud costs effectively to oversee the project creation while maintaining low costs.

Table 4 - Required Skills and Knowledge for Project Execution

Table 4 identifies the skills and levels requirement to acquire and carry out for this project within the timeframe.

3. Project Environment

3.1 Project Environment Overview

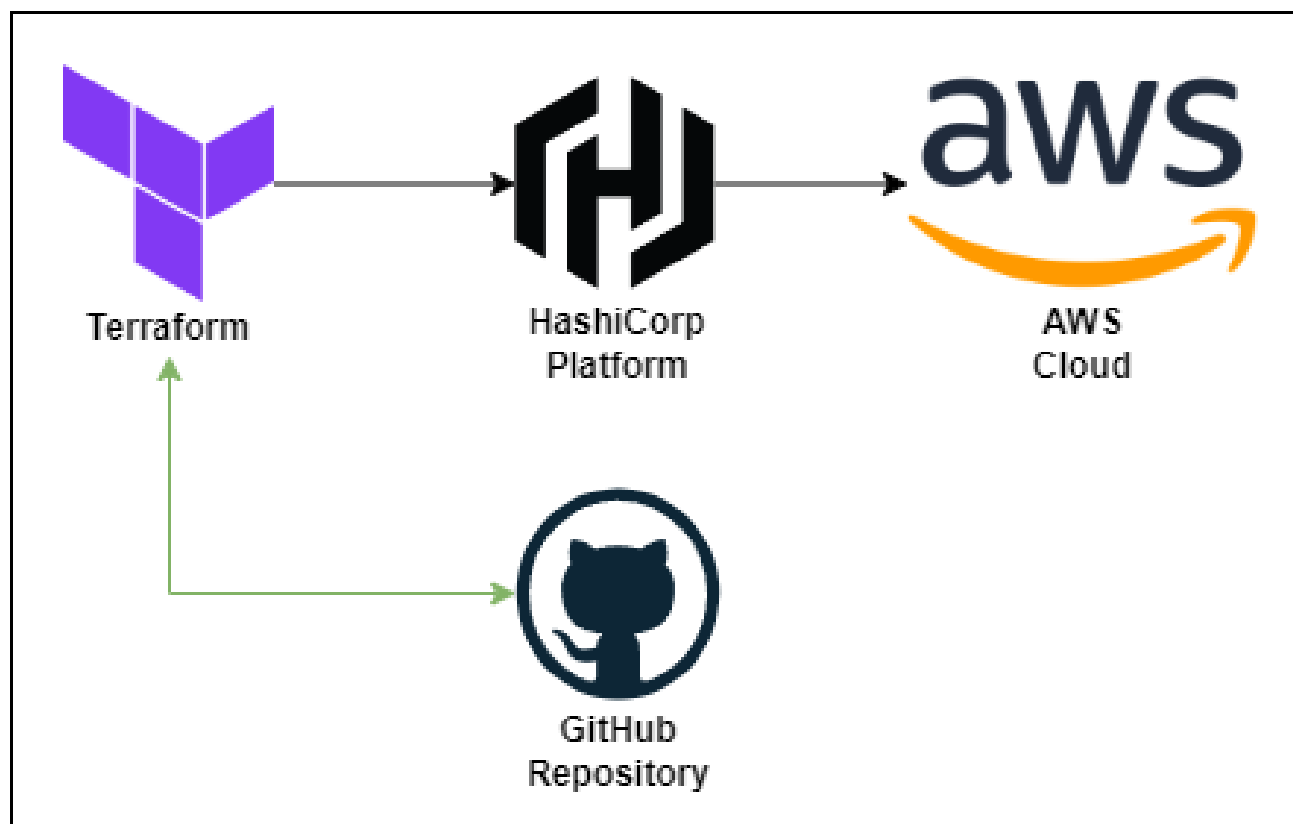


Figure 1 - Terraform HCP AWS Integration Workflow

Figure 1 illustrates a streamlined workflow integrating Terraform, HashiCorp Cloud Platform (HCP), GitHub, and AWS to build and manage cloud infrastructure.

The process begins with Terraform, an Infrastructure as Code (IaC) tool, where configurations for resources like EC2 instances, S3 buckets, and load balancers are defined in a declarative format. These configurations are stored in a GitHub repository for version control, enabling easy tracking of changes/

Terraform commands like plan, apply, deploy or destroy are issued from a local environment (Visual Studio Code etc) and passed directly to HCP, which then manages the execution and securely handles the interaction with AWS. HCP then applies the specified changes by sending commands to AWS to provision or update resources.

3.2 Project Environment Setup

The project setup involves configuring Terraform to work with HashiCorp Cloud Platform (HCP) and AWS, enabling secure and efficient infrastructure management. The basic configurations for setting up the project environment were adopted from official walkthrough by HCP.

3.2.1 Setting up the HCP Workspace

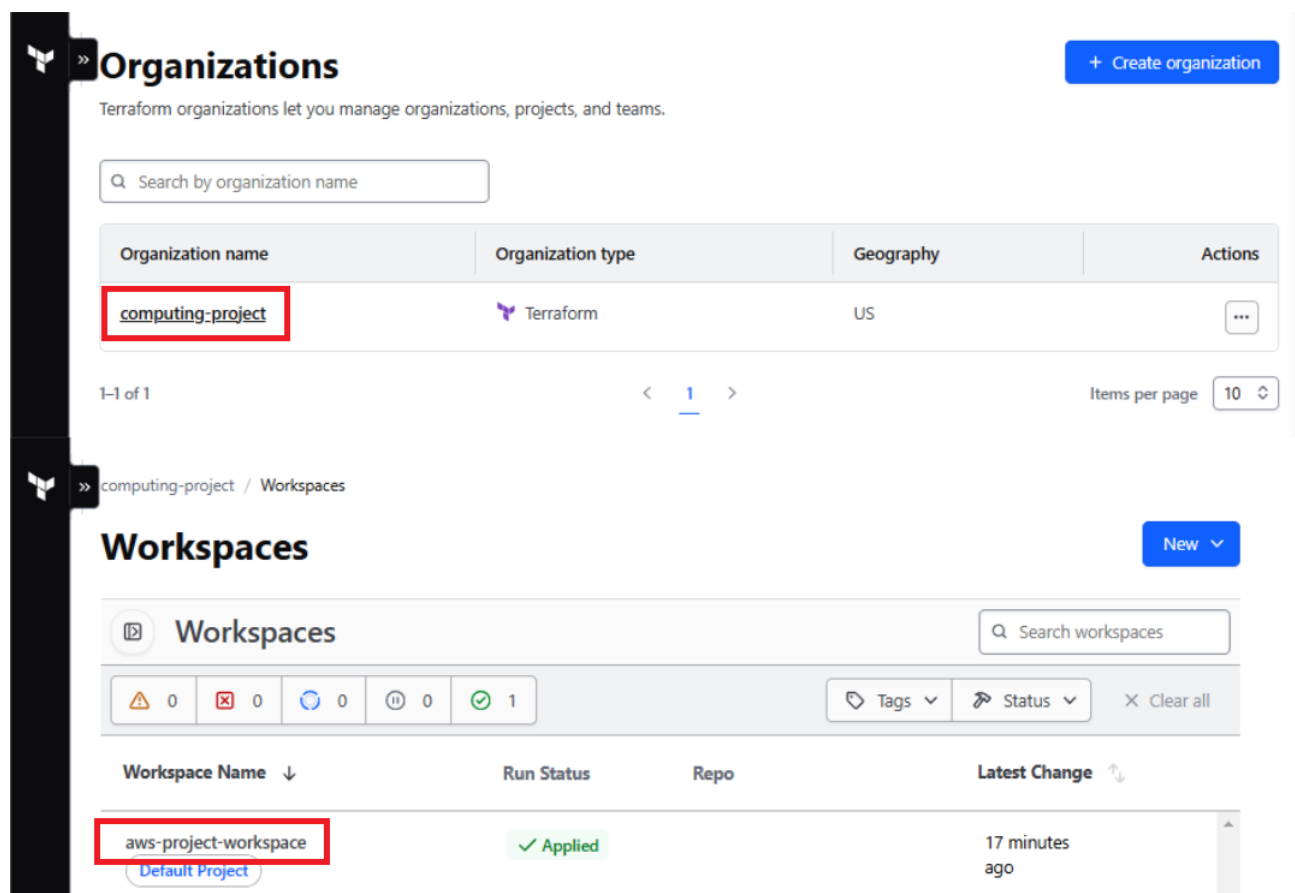


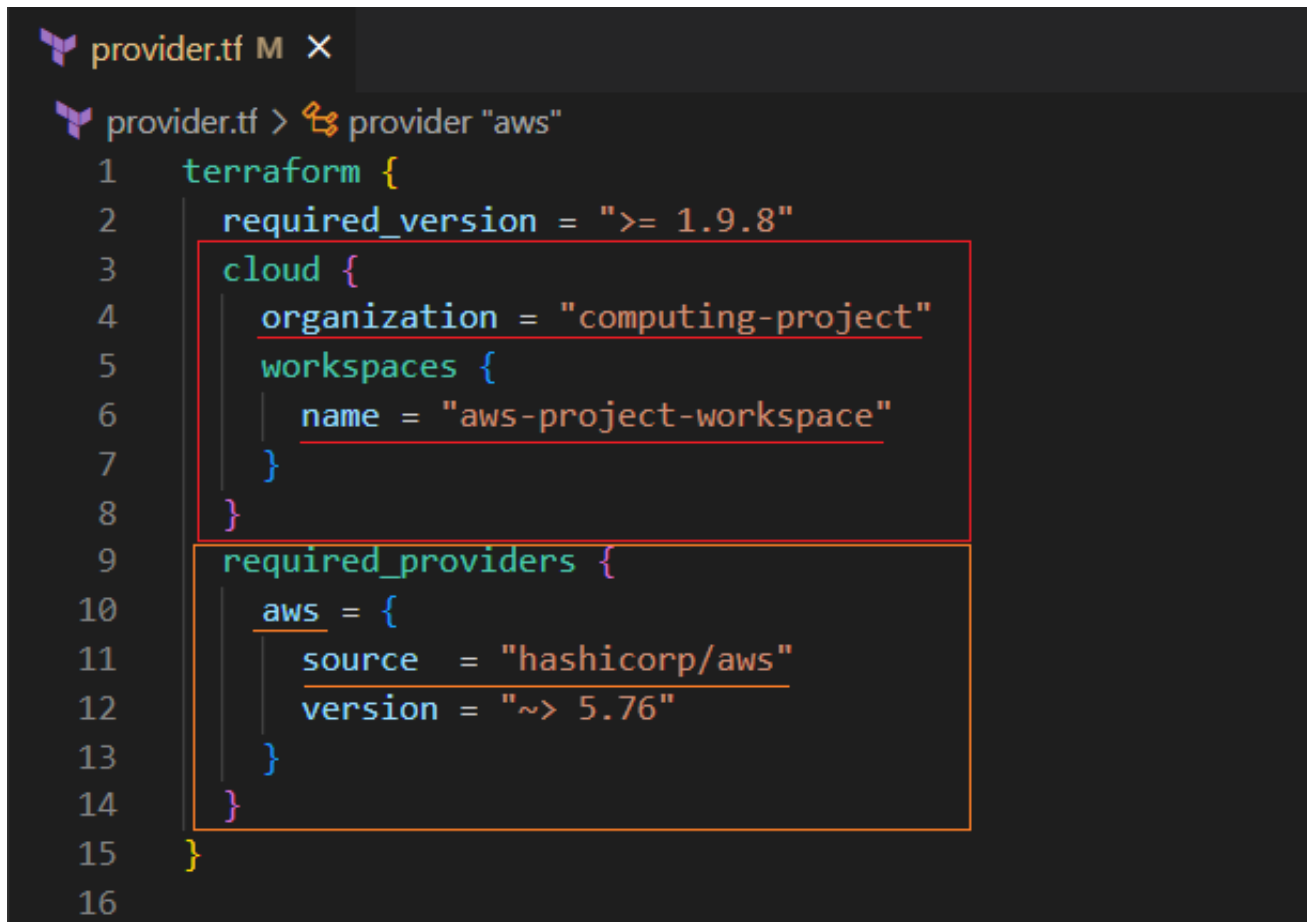
Figure 2 - HCP Workspace Setup

The first step in setting up the project environment is creating an organization and workspace in HCP.

Within the organization, a workspace named aws-project-workspace is created to act as a secure repository for Terraform state files. These state files are critical for tracking infrastructure changes, ensuring consistent deployments, and enabling collaboration among team members.

This essentially centralizes state management to eliminate the risks associated with maintaining local state files, such as loss or conflicts during concurrent updates.

3.2.2 Configuring Terraform Provider



```
provider.tf M X
provider.tf > provider "aws"
1 terraform {
2   required_version = ">= 1.9.8"
3   cloud {
4     organization = "computing-project"
5     workspaces {
6       name = "aws-project-workspace"
7     }
8   }
9   required_providers {
10    aws = {
11      source = "hashicorp/aws"
12      version = "~> 5.76"
13    }
14  }
15 }
16
```

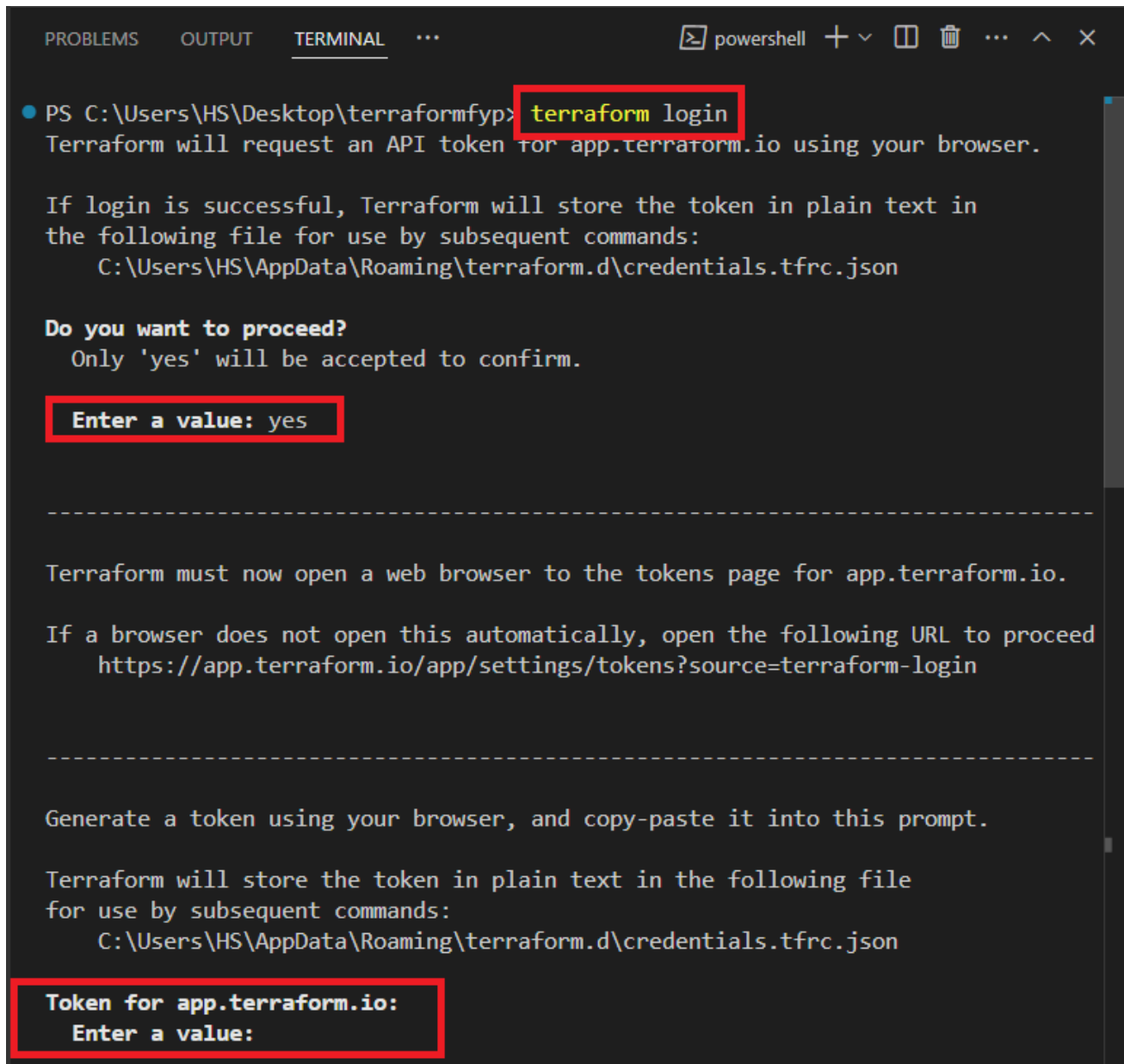
Figure 3 - Terraform Provider Configuration

After creating the workspace, the Terraform provider must be configured. This involves defining the backend for storing Terraform state files and specifying AWS as the cloud provider for resource provisioning.

The providers.tf file is updated to set HCP as the backend and connecting it to the aws-project-workspace in the computing-project organization. This configuration ensures that Terraform commands, such as plan and apply can interact with the created HCP workspace shown in Section 3.2.1.

Lastly, AWS is defined to serve as the cloud provider for this project.

3.2.3 Authenticating Terraform with HCP



```
PS C:\Users\HS\Desktop\terraformfyp> terraform login
Terraform will request an API token for app.terraform.io using your browser.

If login is successful, Terraform will store the token in plain text in
the following file for use by subsequent commands:
  C:\Users\HS\AppData\Roaming\terraform.d\credentials.tfrc.json

Do you want to proceed?
Only 'yes' will be accepted to confirm.

Enter a value: yes

-----

Terraform must now open a web browser to the tokens page for app.terraform.io.

If a browser does not open this automatically, open the following URL to proceed
https://app.terraform.io/app/settings/tokens?source=terraform-login

-----

Generate a token using your browser, and copy-paste it into this prompt.

Terraform will store the token in plain text in the following file
for use by subsequent commands:
  C:\Users\HS\AppData\Roaming\terraform.d\credentials.tfrc.json

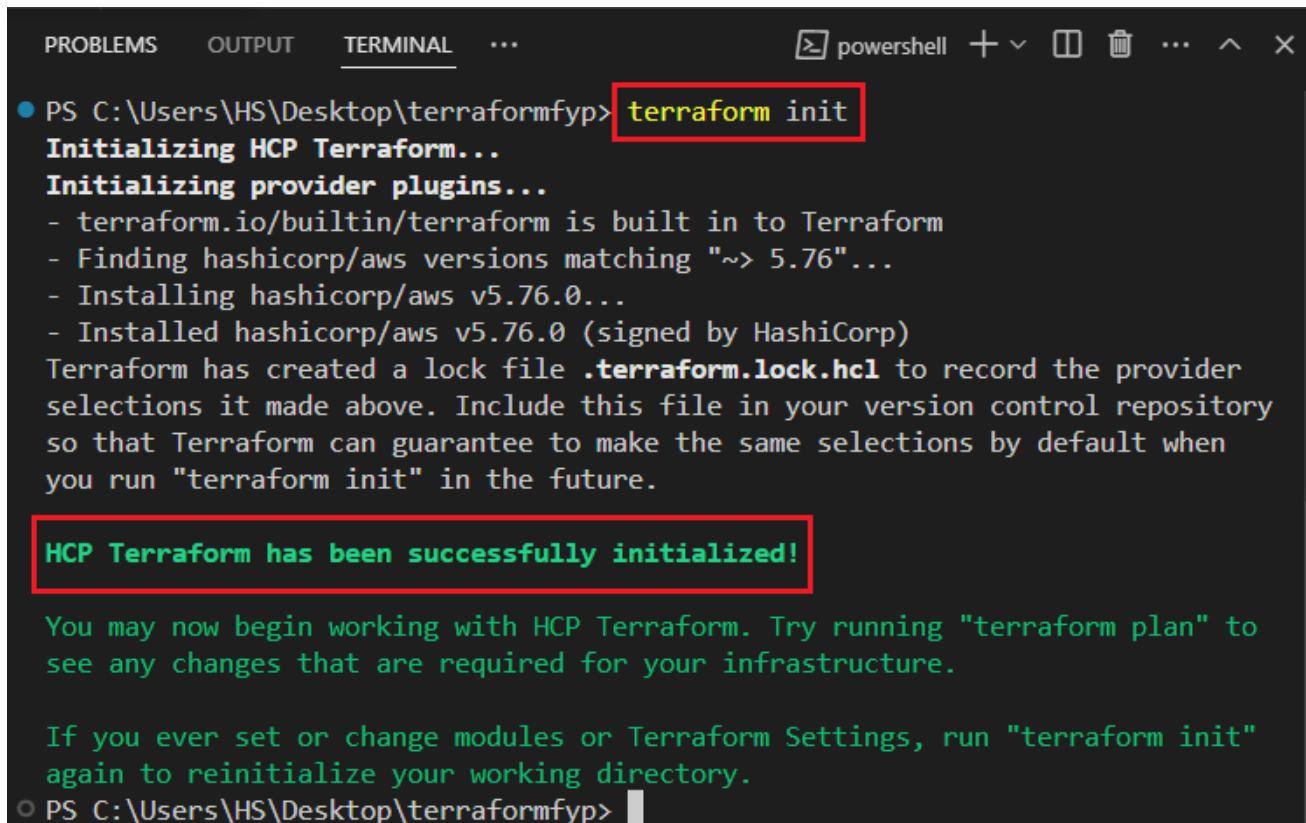
Token for app.terraform.io:
Enter a value:
```

Figure 4 - Terraform HCP Authentication Process

Terraform must be authenticated with HCP to allow terraform to access the organization and workspace in HCP. To achieve this, an authentication process must be executed in the local development environment.

The terraform login command brings the user to a page that generates an authentication token, which is then securely stored locally and used for subsequent interactions with HCP.

3.2.4 Initializing Terraform Environment



```
PS C:\Users\HS\Desktop\terraformfyp> terraform init
Initializing HCP Terraform...
Initializing provider plugins...
- terraform.io/builtin/terraform is built in to Terraform
- Finding hashicorp/aws versions matching "~> 5.76"...
- Installing hashicorp/aws v5.76.0...
- Installed hashicorp/aws v5.76.0 (signed by HashiCorp)
Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

HCP Terraform has been successfully initialized!

You may now begin working with HCP Terraform. Try running "terraform plan" to
see any changes that are required for your infrastructure.

If you ever set or change modules or Terraform Settings, run "terraform init"
again to reinitialize your working directory.
PS C:\Users\HS\Desktop\terraformfyp>
```

Figure 5 - Terraform Initialization Workflow

After Terraform is authenticated with HCP, the terraform init command is executed to initialize the project.

This step downloads the necessary provider plugins, such as the AWS provider, and validates the backend connection to HCP.

The initialization phase ensures that Terraform is ready to execute further commands and that all dependencies are resolved. The successful initialized message confirms that the environment is ready for infrastructure provisioning through HCP.

3.2.5 Authenticating HCP with AWS Cloud

Workspace variables (4)

Variables defined within a workspace always overwrite variables from variable sets that have the same type and the same key. Learn more about variable set [precedence](#) .

Key	Value	Category	Actions
<div><div>AWS_ACCESS_KEY_ID</div><div>Sensitive</div></div>	<i>Sensitive - write only</i>	env	<div>...</div>
AWS_DEFAULT_REGION	ap-southeast-1	env	<div>...</div>
<div><div>AWS_SECRET_ACCESS_KEY</div><div>Sensitive</div></div>	<i>Sensitive - write only</i>	env	<div>...</div>

Figure 6 - HCP Authentication with AWS Cloud

The final step in the setup involves granting HCP the necessary permissions to interact with AWS. The AWS keys are essentially AWS Account credentials that enables HCP to provision resources as defined in the Terraform configuration files into AWS Cloud.

This is achieved by securely storing the AWS Access Key ID and Secret Access Key in HCP workspace as environment variables. By storing credentials in HCP, it ensures credentials are encrypted and reduce the risk of exposing sensitive information.

3.3 Repositories, Access Endpoints, and Terraform Files

The files created from this project include the terraform code, and the frontend and backend of an ecommerce application. These files are stored in GitHub repositories online. The application deployed in this project can be accessed with the URLs provided in Table 6 - Application Access Endpoints.

3.3.1 Repositories

Name	Repository URL	Description
ecomm-terraform	https://github.com/hswg94/ecomm-terraform	This repository contains the code for terraform.
ecomm-react-frontend	https://github.com/hswg94/ecomm-react-frontend	The frontend of the ecommerce application.
ecomm-express-api	https://github.com/hswg94/ecomm-express-api	This backend of the ecommerce application

Table 5 - GitHub Repositories for Project Code

3.3.2 Access Endpoints

Area	URL Access Endpoint	Description
Frontend	https://ecomm.hswg94.com/	An accessible Public URL for users to access the frontend application interface.
Backend	https://api.hswg94.com	An accessible Public URL for API interaction.

Table 6 - Application Access Endpoints

3.3.3 Terraform Files

File	Description
aws-cert-manager.tf	Configures SSL/TLS certificates using AWS Certificate Manager in ap-southeast-1 and us-east-1 regions for secure HTTPS traffic.
buildspec.yaml	Specifies build instructions for AWS CodeBuild, including steps for installing dependencies, building the React app, and packaging artifacts.
cloudfront-cdn.tf	Configures CloudFront with an origin access control to serve static content securely from an S3 bucket, functioning as a CDN for the frontend application.
codestar-connections-policy.json	Defines IAM policies for AWS CodeStar, allowing integration with GitHub, which is necessary for pipeline automation.
ec2-launch-template.tf	Defines an EC2 launch template for application instances, including IAM roles, instance types, and security group configurations to be used with an auto-scaling group.
iam-roles.tf	Sets up IAM roles and policies for EC2 instances, granting access to AWS Secrets Manager and CodeDeploy.
load-balancer.tf	Configures an Application Load Balancer (ALB) to handle HTTPS traffic, distributing requests across EC2 instances and applying necessary security and listener settings.
main.tf	Initializes foundational infrastructure, including VPC, Internet Gateway, and public subnets, establishing the network environment for the application.
pipeline-backend.tf	Defines the backend CI/CD pipeline, connecting to GitHub for source control and using AWS CodePipeline and CodeDeploy for deployment automation.
pipeline-frontend.tf	Configures the frontend CI/CD pipeline with CodeBuild credentials for GitHub and webhook triggers,

	automating the build and deployment of frontend code to the S3 bucket integrated with CloudFront.
provider.tf	Specifies HCP configuration and AWS providers for two regions (ap-southeast-1 and us-east-1), enabling a multi-region setup to support the architecture's requirements.
route53.tf	Manages DNS using Route 53, configuring a hosted zone and DNS records to map the domain to application endpoints.
s3-buckets.tf	Defines S3 buckets used by CodePipeline for artifact storage and by CloudFront as the origin for static assets, with public access blocks configured for security.
security-groups.tf	Manages security groups for EC2 instances and the ALB, controlling ingress and egress traffic according to network security requirements.
userdata.sh	Script executed during EC2 instance initialization, including installation and setup of the CodeDeploy agent. It also retrieves secrets from AWS Secrets Manager, though this is migrated to AppSpec.yml for improved management.

Table 7 - Terraform Files and Configurations

4. Project Scope

4.1 Full System Architecture Diagram

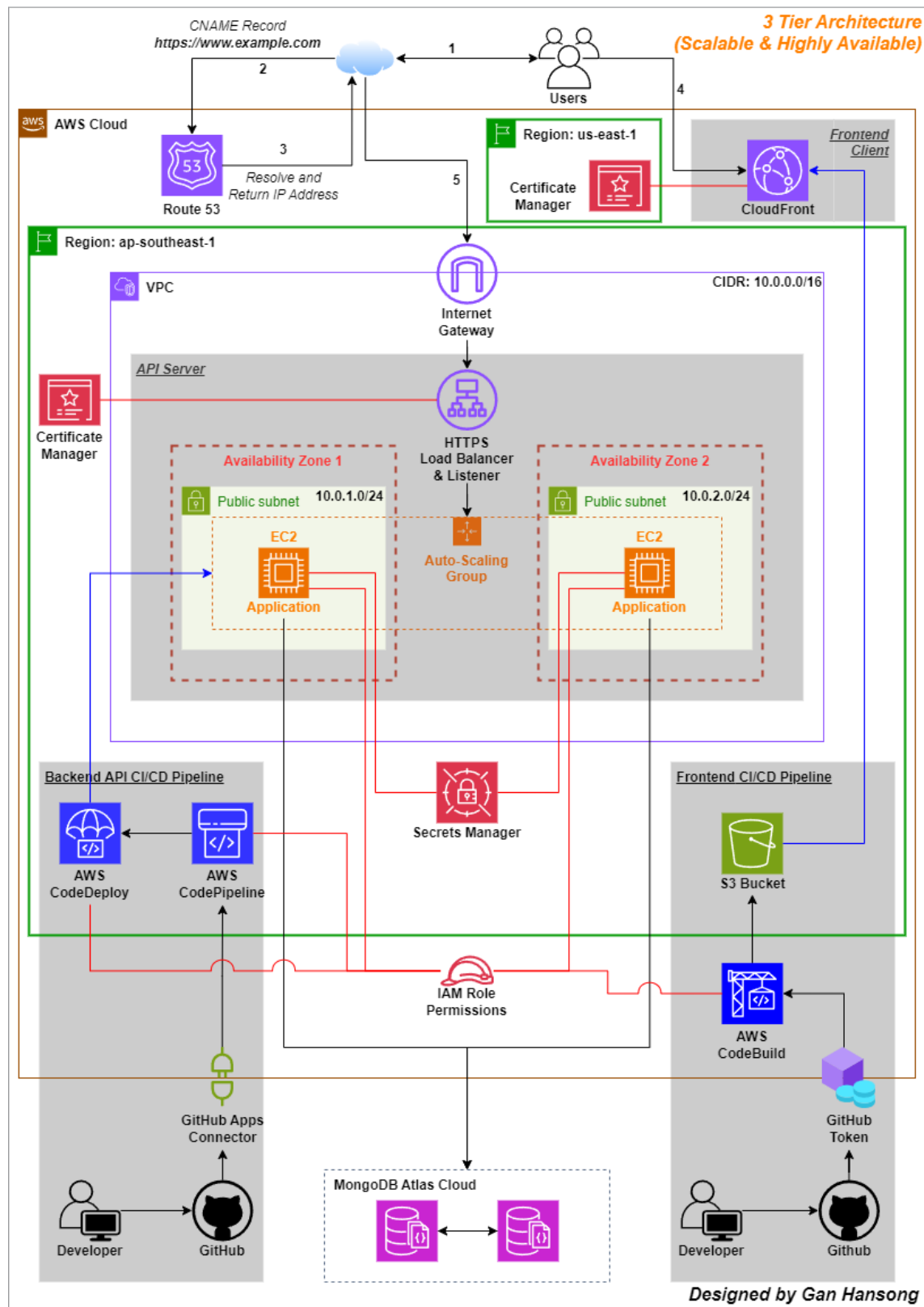


Figure 7 - Full System Architecture Diagram

4.2 Full System Architecture Breakdown

Figure 7 - Full System Architecture Diagram represents a scalable, highly available 3-tier application on AWS. It utilizes a globally distributed CDN for the frontend with AWS CloudFront and S3. The backend is hosted in a VPC across multiple Availability Zones to ensure scalability and high availability. The database is hosted externally on MongoDB Atlas Cloud.

The architecture includes CI/CD pipelines for automated deployments by detecting changes on GitHub Repositories. AWS Secrets Manager securely manages sensitive data such as API keys, while IAM roles allow resources to access another resource.

Database backups and disaster recovery are managed through MongoDB Atlas automated backup features externally.

The tables below provide navigation to detailed explanations of each architectural component in Table 8.

Area	Design Overview	Code Implementation
Frontend Architecture	Section 4.3	Section 5
Frontend CI/CD Pipeline Architecture	Section 4.4	Section 6
Backend Architecture	Section 4.5	Section 7
Backend CI/CD Pipeline Architecture	Section 4.6	Section 8

Table 8 - Navigation for Architecture and Implementation Details

4.3 Overview of Frontend System Architecture

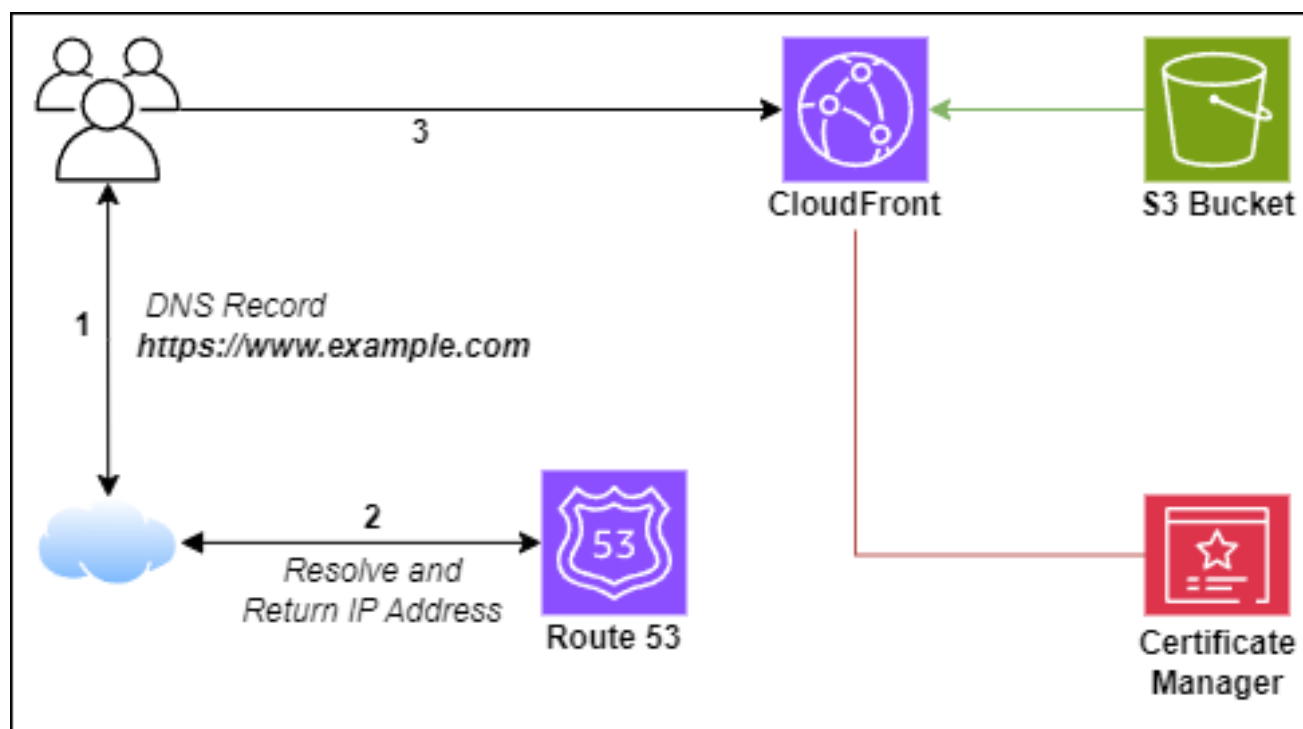


Figure 8 - Frontend System Architecture Overview

The frontend application for this project is a static site built from a React-based single-page application deployed onto AWS. Users access the frontend through a URL (e.g., <https://ecomm.hswg94.com>), which is managed by Amazon Route 53.

Route 53 serves as the Domain Name System (DNS) service, translating the URL to an IP address that directs user traffic into AWS infrastructure. Once resolved, requests are forwarded to AWS CloudFront, a global Content Delivery Network (CDN) that serves as the primary endpoint for frontend access.

CloudFront retrieves static content from the S3 bucket, caching it across edge locations worldwide. This caching ensures faster load times and reduces latency by delivering content from the nearest edge location to each user.

To secure data in transit, AWS Certificate Manager in the us-east-1 region manages TLS certificates, providing secure HTTPS communication between users and CloudFront. AWS Certificate Manager certificates for CloudFront must reside in us-east-1 to support global distributions.

By integrating Route 53, CloudFront, S3, and Certificate Manager, this architecture guarantees a secure, low-latency, and reliable frontend experience for users.

4.4 Overview of Frontend CI/CD Pipeline Architecture

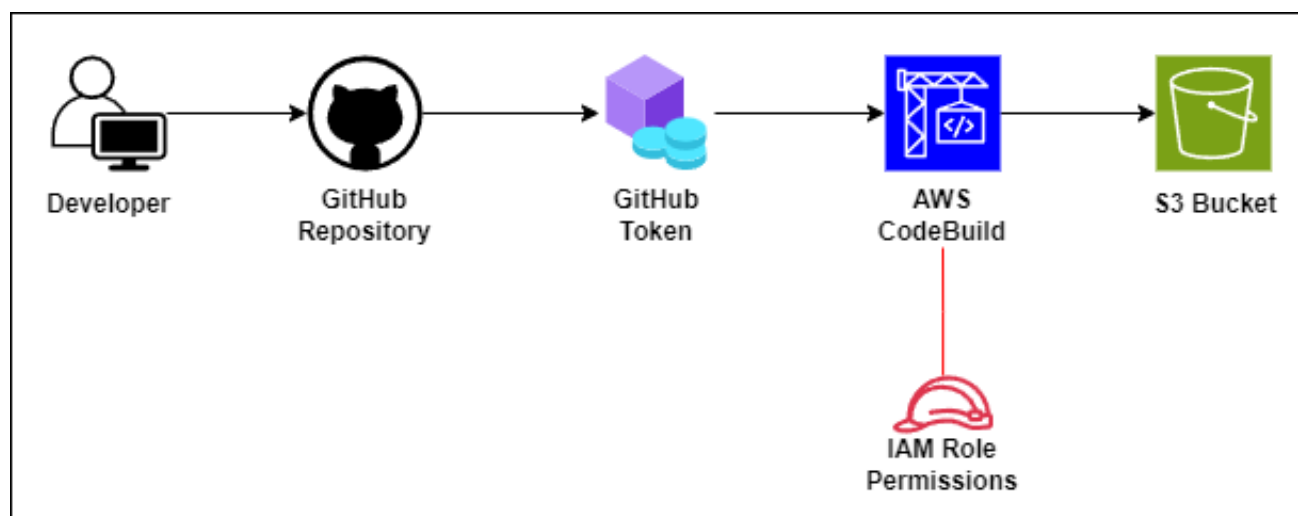


Figure 9 - Frontend CI/CD Pipeline Architecture

The goal of the frontend CI/CD pipeline is to automate the deployment of static assets to an S3 bucket.

When a developer pushes code to the main branch in the GitHub repository, AWS CodeBuild detects the latest changes via a connection established through a GitHub Token. The established connection, along with IAM role permissions allows CodeBuild to automatically pull, build, and upload the latest version of the frontend application to an S3 bucket.

As described in Section 4.4, the S3 bucket serves as a content repository, from which CloudFront retrieves the assets to distribute globally via its Content Delivery Network (CDN).

To ensure the frontend is immediately accessible to users upon deployment, the pipeline is configured to automatically trigger and deploy the latest version from the repository upon its creation.

4.5 Overview of Backend System Architecture

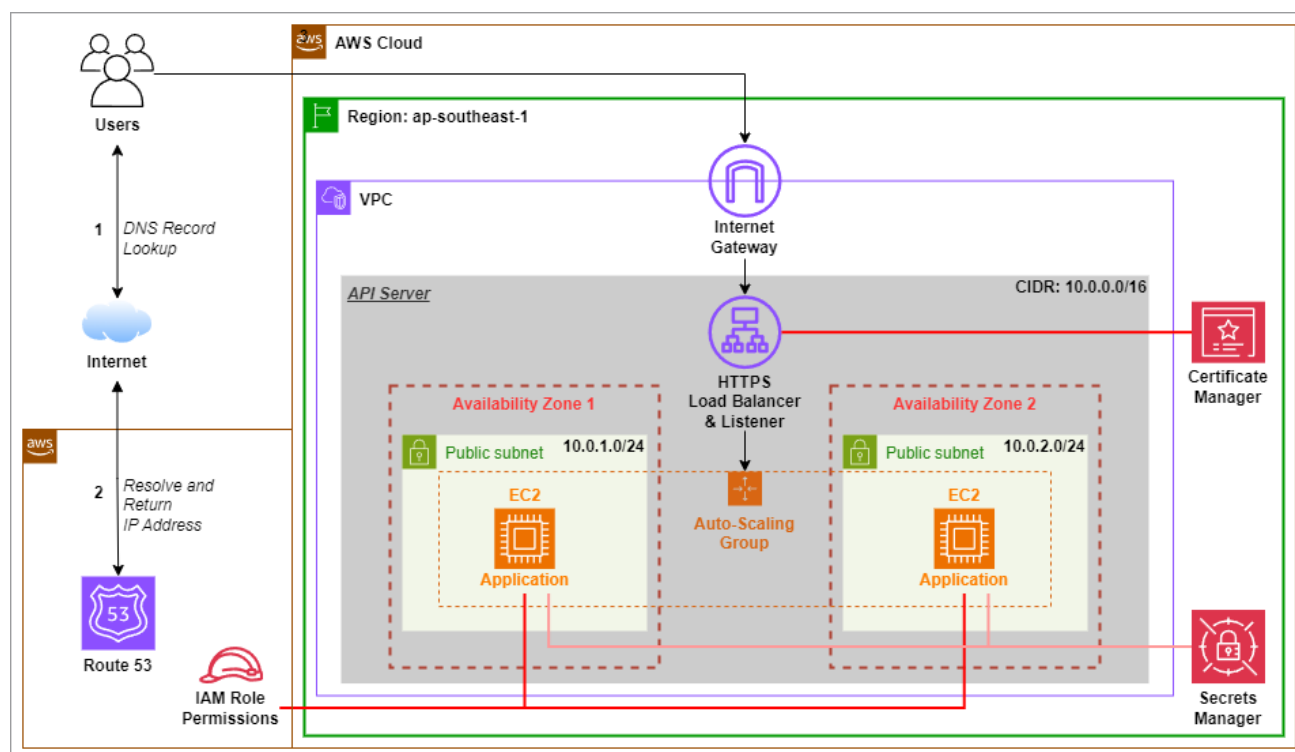


Figure 10 - Backend System Architecture Overview

The backend system architecture serves the API of the ecommerce application that is hosted in the ap-southeast-1 region within a Virtual Private Cloud (VPC) with a CIDR of 10.0.0.0/16, public internet access is granted to the VPC using an Internet Gateway (IGW). This VPC is divided into two Availability Zones (AZs) each containing a public subnet (10.0.1.0/24 and 10.0.2.0/24, respectively) to ensure high availability.

Inside each subnet, an EC2 instance hosts the application server, with an Auto-Scaling Group (ASG) balancing traffic across instances and dynamically adjusting the number of instances based on demand.

A Load Balancer with a HTTPS listener sits between users and the application, distributing incoming requests across the EC2 instances for load balancing and high availability, the load balancer also serves as the end point for assessing the API. A TLS certificate managed by the AWS Certificate Manager is attached to the load balancer to enable HTTPS communication.

IAM Role Permissions are provided to every EC2 instance in the auto scaling group allowing these instances to access the Secrets Managers that contains secret keys for the application, the permissions also allow it to communicate with CodeDeploy for the deployment of application into the instances.

4.6 Overview of Backend CI/CD Pipeline Architecture

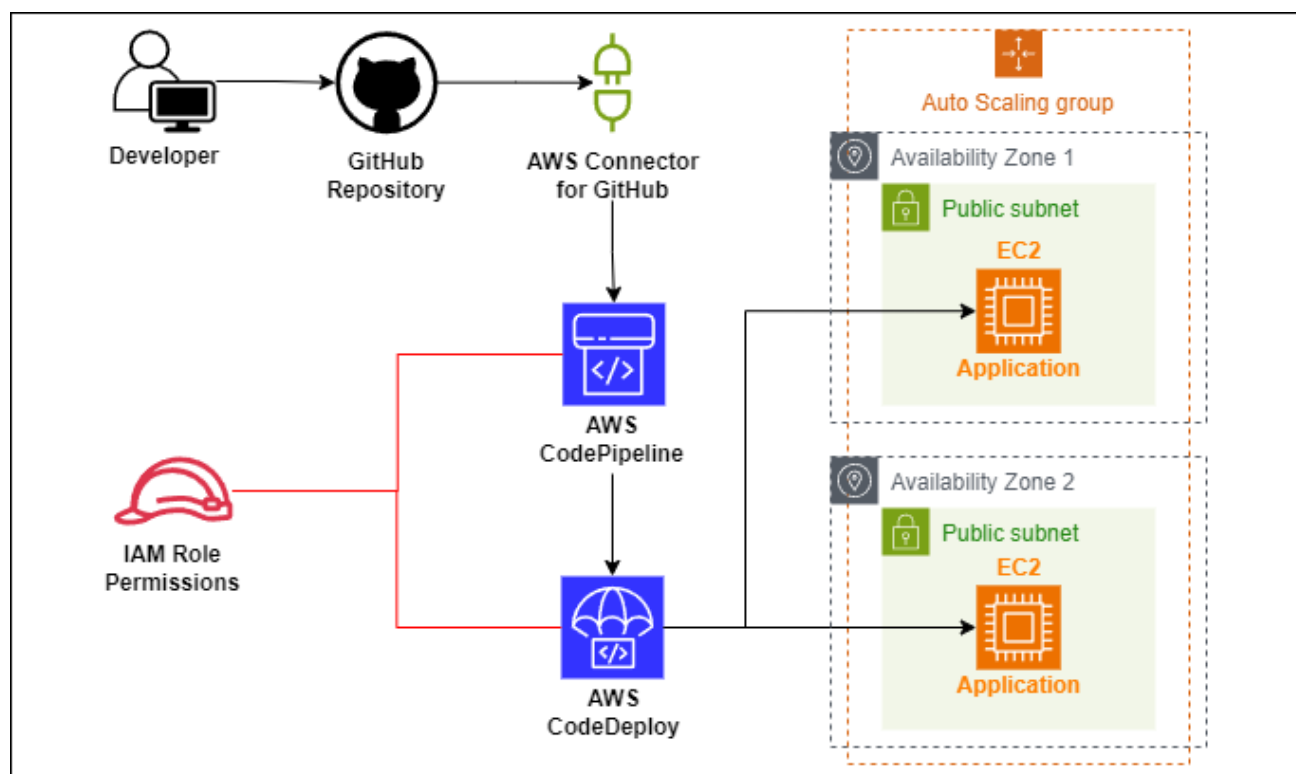


Figure 11 - Backend CI/CD Pipeline Architecture

The backend CI/CD pipeline automates the deployment of application to EC2 instances within an Auto Scaling Group. A connection between AWS and GitHub is established with AWS Connector for GitHub to allow communication between CodePipeline and GitHub. IAM roles are assigned to AWS CodePipeline and CodeDeploy, allowing them to securely access each other, as well as EC2 and Auto Scaling resources.

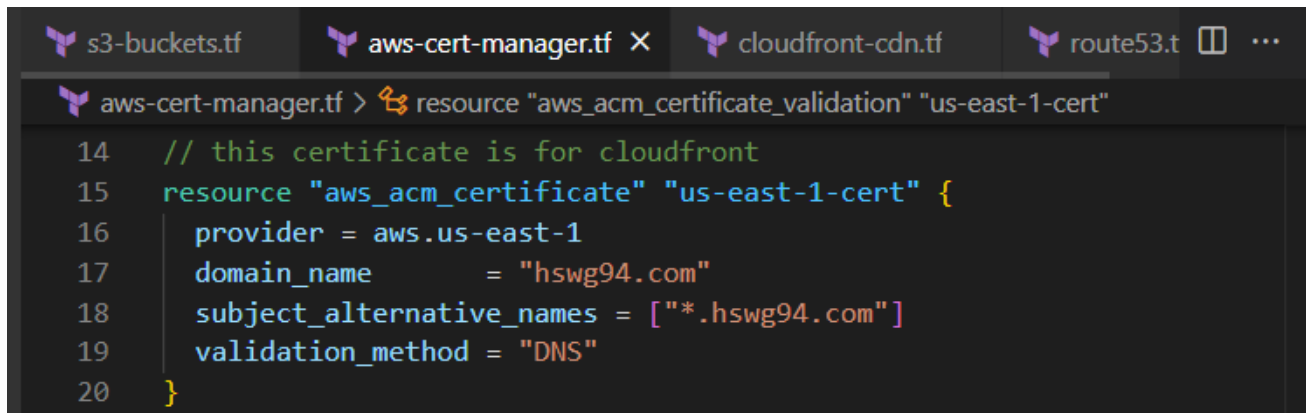
The pipeline is initiated automatically upon initial creation to deploy the latest commit, making the API accessible immediately. It also triggers after a developer pushes new code to the main branch of the GitHub repository.

When AWS CodePipeline is triggered, it pulls the latest code from GitHub and sends it to AWS CodeDeploy. CodeDeploy then manages the deployment across all EC2 instances in the Auto Scaling Group. The deployment strategy is configured for a blue/green deployment, allowing for controlled, gradual rollouts and ensuring high availability without downtime. This strategy facilitates seamless updates, keeping the API continuously accessible to users.

During deployment, CodeDeploy reads an **AppSpec.yml** file located in the root of the repository. This file defines deployment actions and lifecycle events, deployment, and post-deployment steps), ensuring the application is correctly booted up at each phase of deployment.

5. Frontend Systems Implementation

5.1 Creating TLS Certification for CloudFront



```
s3-buckets.tf  aws-cert-manager.tf X  cloudfront-cdn.tf  route53.t  ...
aws-cert-manager.tf > resource "aws_acm_certificate_validation" "us-east-1-cert"

14  // this certificate is for cloudfront
15  resource "aws_acm_certificate" "us-east-1-cert" {
16    provider = aws.us-east-1
17    domain_name      = "hswg94.com"
18    subject_alternative_names = ["*.hswg94.com"]
19    validation_method = "DNS"
20  }
```

Figure 12 - TLS Certificate Creation for CloudFront

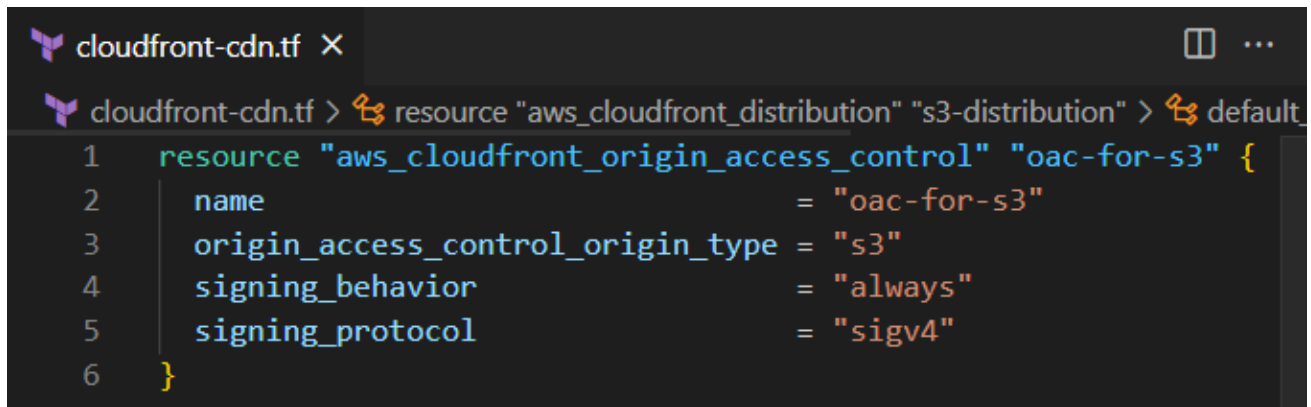
To enable secure HTTPS communication between users and the frontend application, an SSL/TLS certificate is issued using AWS Certificate Manager (ACM). This certificate is required by CloudFront to ensure encrypted data transmission.

- **<provider>** is set to `aws.us-east-1` as ACM certificates for CloudFront must reside in the `us-east-1` region.
- **<domain_name>** is specified as `hswg94.com`, representing the primary domain name for the application.
- **<subject_alternative_names>** includes `*.hswg94.com` to extend the certificate coverage to all subdomains, including `ecomm.hswg94.com`.
- **<validation_method>** is set to `DNS`, meaning domain ownership is verified by adding DNS records to the Route 53 hosted zone. The steps to validating the certificate can be found in Appendix I: Validating Certificates with Route 53.

5.2 Creating a CloudFront Distribution

The CloudFront Distribution serves as the entry point to the frontend of the application, it is established with Origin Access Control which allows the S3 Bucket to stay private for maximum security.

5.2.1 CloudFront Origin Access Control



```
cloudfront-cdn.tf X
cloudfront-cdn.tf > resource "aws_cloudfront_distribution" "s3-distribution" > default
1  resource "aws_cloudfront_origin_access_control" "oac-for-s3" {
2      name                                = "oac-for-s3"
3      origin_access_control_origin_type   = "s3"
4      signing_behavior                     = "always"
5      signing_protocol                     = "sigv4"
6  }
```

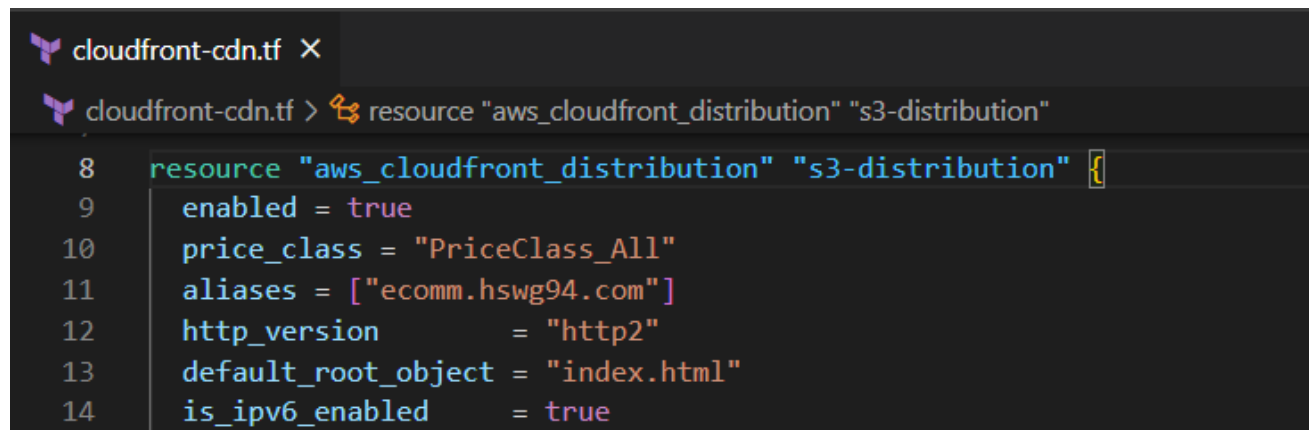
Figure 13 - CloudFront Origin Access Control Configuration

Origin Access Control (OAC) is used to secure interaction between CloudFront and the S3 bucket.

- **<origin_access_control_origin_type>** is set to "s3", specifying that the origin is an S3 bucket.
- **<signing_behavior>** "always" ensures CloudFront signs every request to the S3 bucket.
- **<signing_protocol>** "sigv4" enables secure communication using Signature Version 4.

5.2.2 CloudFront Distribution Configuration

5.2.2.1 CloudFront Distribution - Miscellaneous Parameters

A screenshot of a code editor with a dark theme. The top bar shows a file named 'cloudfront-cdn.tf' with a close button. Below it, a breadcrumb path reads 'cloudfront-cdn.tf > resource "aws_cloudfront_distribution" "s3-distribution"'. The main code area shows a Terraform resource block for 'aws_cloudfront_distribution' named 's3-distribution'. The code is as follows:

```
8 resource "aws_cloudfront_distribution" "s3-distribution" {  
9     enabled = true  
10    price_class = "PriceClass_All"  
11    aliases = ["ecomm.hswg94.com"]  
12    http_version = "http2"  
13    default_root_object = "index.html"  
14    is_ipv6_enabled = true  
}
```

Figure 14 - CloudFront Miscellaneous Parameters Configuration

Figure 14 shows a couple of miscellaneous parameters that are required for establishing the CloudFront Distribution.

- **<enabled>** is set to true to activate the CloudFront distribution upon creation.
- **<price_class>** is set to "PriceClass_All" to enable the distribution to use all available edge locations globally for maximum performance.
- **<aliases>** specifies ecomm.hswg94.com as the custom domain name associated with this distribution.
- **<http_version>** is set to http2 to leverage faster and more efficient HTTP/2 connections for improved user experience.
- **<default_root_object>** is configured as index.html, setting the default entry point for users accessing the distribution.
- **<is_ipv6_enabled>** is set to true to support modern IPv6 addressing, ensuring compatibility with newer network standards.

5.2.2.2 CloudFront Distribution - Origin

```
16  ✓ origin {  
17      domain_name           = aws_s3_bucket.ecomm-frontend-s3-for-cb-and-cf.bucket_regional_domain_name  
18      origin_access_control_id = aws_cloudfront_origin_access_control.oac-for-s3.id  
19      origin_id             = aws_s3_bucket.ecomm-frontend-s3-for-cb-and-cf.id  
20  }  
21
```

Figure 15 - CloudFront Distribution Origin Configuration

The origin configuration defines the source of content for the CloudFront distribution, which in this case is an S3 bucket. It also ensures secure access to the bucket via Origin Access Control (OAC).

- **<domain_name>** is set to `aws_s3_bucket.ecomm-frontend-s3-for-cb-and-cf.bucket_regional_domain_name`, which specifies the bucket domain name and region name.
- **<origin_access_control_id>** is set to `aws_cloudfront_origin_access_control.oac-for-s3.id`, linking the Origin Access Control (OAC) to the origin. This ensures secure communication between CloudFront and the S3 bucket, allowing only signed requests.
- **<origin_id>** is set to `"aws_s3_bucket.ecomm-frontend-s3-for-cb-and-cf.id"` to identify the name of the bucket.

5.2.2.3 CloudFront Distribution - Default Cache Behaviour

```
22  ✓ default_cache_behavior {
23      compress           = true
24      viewer_protocol_policy = "redirect-to-https"
25      allowed_methods     = ["DELETE", "GET", "HEAD", "OPTIONS", "PATCH", "POST", "PUT"]
26      cached_methods      = ["GET", "HEAD"]
27      target_origin_id     = aws_s3_bucket.ecomm-frontend-s3-for-cb-and-cf.id
28  ✓  forwarded_values {
29      query_string = false
30  ✓  cookies {
31      forward = "none"
32  }
33  }
34  }
```

Figure 16 - CloudFront Default Cache Behaviour Configuration

The default cache behaviour specifies how CloudFront interacts with the origin and serves content to users. It defines caching rules, allowed HTTP methods, and forwarding behaviours for improved performance and security. This configuration optimizes performance by caching content effectively while enforcing HTTPS and secure handling of user requests.

- **<compress>** is set to true, enabling gzip and Brotli compression for faster delivery of content to end users.
- **<viewer_protocol_policy>** is set to redirect-to-https, ensuring that all requests are redirected to HTTPS, enforcing secure communication.
- **<allowed_methods>** includes DELETE, GET, HEAD, OPTIONS, PATCH, POST, and PUT, defining all the HTTP methods that clients can use when interacting with the distribution. to ensure static content delivery while minimizing unnecessary traffic to the origin.
- **<cached_methods>** includes GET and HEAD, specifying the HTTP methods whose responses are cached by CloudFront for faster delivery by minimizing unnecessary traffic to the origin.
- **<target_origin_id>** is set to aws_s3_bucket.ecomm-frontend-s3-for-cb-and-cf.id, linking the cache behaviour to the S3 bucket origin.
- **<forwarded_values>**:
 - **<query_string>** is set to false, meaning query strings are not forwarded to the origin, simplifying caching behaviour.
 - **<cookies.forward>** is set to none, ensuring no cookies are forwarded to the origin, enhancing cache and security.

5.2.2.4 CloudFront Distribution - Restrictions

```
36     restrictions {
37         geo_restriction {
38             restriction_type = "none"
39             locations        = [] # ["US", "CA", "GB", "DE"]
40         }
41     }
```

Figure 17 - CloudFront Distribution Geo Restrictions

The geo restriction configuration defines the geographic locations from which the CloudFront distribution can or cannot serve content.

- **<restriction_type>** is set to "none", allowing the distribution to serve content globally without any restrictions.
- **<locations>** is set to an empty list [], meaning no specific geographic regions are restricted.

5.2.2.5 CloudFront Distribution - TLS Certification

```
43     viewer_certificate {
44         acm_certificate_arn      = aws_acm_certificate_validation.us-east-1-cert.certificate_arn
45         ssl_support_method       = "sni-only" # sni-only for cost-effectiveness
46         minimum_protocol_version = "TLSv1.2_2021" # Ensures TLS 1.2+ for modern security compliance
47     }
```

Figure 18 - CloudFront TLS Certificate Configuration

The viewer certificate configuration ensures secure communication between users and the CloudFront distribution by enabling HTTPS with a validated ACM certificate.

- **<acm_certificate_arn>** is set to `aws_acm_certificate_validation.us-east-1-cert.certificate_arn`, referencing the validated ACM certificate for enabling HTTPS.
- **<ssl_support_method>** is set to "sni-only", which uses Server Name Indication (SNI) to associate the certificate with the domain. This is a cost-effective method as it avoids the need for dedicated IP addresses.
- **<minimum_protocol_version>** is set to "TLSv1.2_2021", enforcing the use of modern and secure TLS 1.2+ protocols, aligning with modern security standards.

5.2.2.6 CloudFront Distribution - Custom Error Response

```
49  ✓  custom_error_response {  
50      error_caching_min_ttl = 300  
51      error_code = 403  
52      response_code = 200  
53      response_page_path = "/index.html"  
54  }  
55  }
```

Figure 19 - CloudFront Custom Error Response Configuration

This custom error response is used for Single Page Application (SPA) to handle cases where refreshing a page that would result in a 403 (Forbidden) error. Instead of showing an error page, the configuration redirects users to index.html which essentially lets the handling of routing to the SPA.

- **<error_caching_min_ttl = 300>** sets the minimum time (in seconds) CloudFront caches the error response, reducing repeated requests to the origin for the same error and improving performance.
- **<error_code = 403>** specifies that this custom error response applies to HTTP 403 (Forbidden) errors received from the origin.
- **<response_code = 200>** changes the HTTP response code returned to the user from 403 to 200, making the response appear successful and allowing the application to handle the routing.
- **<response_page_path = "/index.html">** defines the custom error page (index.html) that users are redirected to when a 403 error occurs, ensuring that the Single Page Application (SPA) can properly handle routing and provide a smooth user experience.

5.3 Creating an ALIAS record for CloudFront Distribution

```
51 resource "aws_route53_record" "api-frontend-endpoint" {  
52     zone_id = aws_route53_zone.primary.zone_id  
53     name     = "ecomm.hswg94.com"  
54     type     = "A"  
55  
56     alias {  
57         name           = aws_cloudfront_distribution.s3-distribution.domain_name  
58         zone_id        = aws_cloudfront_distribution.s3-distribution.hosted_zone_id  
59         evaluate_target_health = false  
60     }  
61 }
```

Figure 20 - DNS ALIAS Record for CloudFront

This configuration sets up DNS records in Route 53 and creates an A record for `ecomm.hswg94.com` to point to the CloudFront Distribution.

- **<zone_id>** specifies the hosted zone ID for `hswg94.com`. The creation of the hosted zone is detailed in Appendix H: Route 53 Hosted Zone.
- **<name = "ecomm.hswg94.com">** is the subdomain for the frontend access.
- **<name = aws_lb.ecomm-api-alb.dns_name>** points the A record to the CloudFront Distribution DNS name.
- **<zone_id = aws_lb.ecomm-api-alb.zone_id>** links it to the CloudFront Distribution hosted zone.
- **<evaluate_target_health = true>** enables health checks on the CloudFront Distribution endpoint to report errors.

6. Frontend CI/CD Pipeline Implementation

6.1 AWS CodeBuild - Source Credentials

```
pipeline-frontend.tf ×
pipeline-frontend.tf > resource "aws_codebuild_webhook" "ecomm-frontend-builder-webhook"
1 # Import the 'Terraform Variable' from HCP terraform
2 variable "GITHUB_TOKEN" {
3     type = string
4     description = "GitHub Token for CodeBuild"
5     sensitive = true
6 }
7
8 # Create the CodeBuild Source Credential
9 resource "aws_codebuild_source_credential" "codebuild-credentials" {
10     auth_type = "PERSONAL_ACCESS_TOKEN"
11     server_type = "GITHUB"
12     token = var.GITHUB_TOKEN
13 }
```

Figure 21 - AWS CodeBuild Source Credentials Configuration

To prepare the frontend CI/CD pipeline, a Terraform variable `GITHUB_TOKEN` is first stored in HCP terraform, it is then declared in the local environment to allow the usage of the token. The token is then passed and used as a source credential for CodeBuild to allow authentication with GitHub.

6.2 AWS CodeBuild – Web Hook

```
15 resource "aws_codebuild_webhook" "ecomm-frontend-builder-webhook" {
16     project_name = aws_codebuild_project.ecomm-frontend-builder.name
17     build_type = "BUILD"
18 }
```

Figure 22 - AWS CodeBuild Webhook Trigger Setup

The webhook configuration in AWS CodeBuild enables automatically trigger build from source code repositories upon changes in the linked GitHub repository. This ensures seamless CI/CD integration by eliminating the need for manual build triggers.

- **<project_name = aws_codebuild_project.ecomm-frontend-builder.name>** links the webhook to the AWS CodeBuild project **<ecomm-frontend-builder>**.
- **<build_type = "BUILD">** ensures the pipeline executes a build whenever a relevant GitHub event (e.g., a push) occurs.

6.3 AWS CodeBuild – Project Build Configuration

The project build configuration defines the resources, source, environment, and artifact setup required for AWS CodeBuild to execute the frontend CI/CD pipeline effectively. Each component plays a role in ensuring the code is retrieved, built, and deployed seamlessly.

6.3.1 AWS CodeBuild – Main Configuration Details

```
19
20  resource "aws_codebuild_project" "ecomm-frontend-builder" {
21      name          = "ecomm-frontend-builder"
22      description    = "This builds the ecomm-frontend and place it in S3 bucket"
23      service_role   = aws_iam_role.CodeBuildRole.arn
24  }
```

Figure 23 - AWS CodeBuild Main Configuration

The main configuration sets up the project details and links the required IAM role for resource access.

- **<name = "ecomm-frontend-builder">** specifies the name of the AWS CodeBuild project as **<ecomm-frontend-builder>**.
- **<service_role = aws_iam_role.CodeBuildRole.arn>** specifies the IAM role used by AWS CodeBuild for permissions to access resources. The configuration of the IAM role is detailed in Appendix B: IAM Role for CodeBuild.

6.3.2 AWS CodeBuild – Source Configuration

```
24
25  source {
26      type      = "GITHUB"
27      location   = "https://github.com/hswg94/ecomm-react-frontend"
28      buildspec = file("buildspec.yaml")
29  }
```

Figure 24 - AWS CodeBuild Source Configuration

The source configuration specifies where AWS CodeBuild will fetch the source code and defines the build instructions.

- **<type = "GITHUB">** sets the source repository type to **<GITHUB>**.
- **<location = "https://github.com/hswg94/ecomm-react-frontend">** specifies the repository URL where the source code is located.
- **<buildspec = file("buildspec.yaml")>** points to the **<buildspec.yaml>** file that defines the build instructions.

6.3.3 AWS CodeBuild – Environment Configuration

```
30
31  ✓ environment {
32      type = "LINUX_LAMBDA_CONTAINER"
33      compute_type = "BUILD_LAMBDA_4GB"
34      image_pull_credentials_type = "CODEBUILD"
35      image = "aws/codebuild/amazonlinux-x86_64-lamb
36  }
37
```

Figure 25 - AWS CodeBuild Environment Configuration

The environment configuration specifies the runtime environment for the build process, including the container type, compute capacity, and Docker image.

- **<type = "LINUX_CONTAINER">** specifies the build environment as <LINUX_CONTAINER>.
- **<compute_type = "BUILD_LAMBDA_4GB">** allocates <BUILD_LAMBDA_4GB> compute capacity for the build process.
- **<image_pull_credentials_type = "CODEBUILD">** ensures the Docker image is pulled using <CODEBUILD> credentials.
- **<image = "aws/codebuild/amazonlinux-x86_64-lambda-standard:nodejs20">** specifies the Docker image <amazonlinux-x86_64-lambda-standard:nodejs20> for the build.

6.3.4 AWS CodeBuild – Artifacts Configuration

```
37
38 artifacts {
39     type           = "S3"
40     location       = aws_s3_bucket.ecomm-frontend-s3-for-cb-and-cf.bucket
41     name           = "/"
42     encryption_disabled = true
43     packaging      = "NONE"
44 }
45
```

Figure 26 - AWS CodeBuild Artifacts Configuration

The artifacts configuration defines how the build output is stored and specifies the storage location and properties.

- **<type = "S3">** configures S3 as the output artifact storage type.
- **<location = aws_s3_bucket.ecomm-frontend-s3-for-cb-and-cf.bucket>** specifies the S3 bucket for storing build artifacts, the creation of the bucket is detailed in Appendix A: S3 Bucket Creation.
- **<encryption_disabled = true>** disables encryption for artifacts since they are publicly accessible static files served via CloudFront.
- **<packaging = "NONE">** specifies that no packaging is applied to the output artifacts.

6.3.5 AWS CodeBuild – Logging Configuration

```
45
46 logs_config {
47     cloudwatch_logs {
48         status = "ENABLED"
49     }
50 }
51
52
```

Figure 27 - AWS CodeBuild Logging Configuration

The logging configuration enables the monitoring of build processes by directing logs to AWS CloudWatch.

- **<status = "ENABLED">** enables logging to CloudWatch for monitoring build execution.

6.4 AWS CodeBuild – Trigger Upon Creation

```
54 # terraform_data resource to trigger CodeBuild on initial setup
55 resource "terraform_data" "initial_codebuild_trigger" {
56     triggers_replace = aws_codebuild_project.ecomm-frontend-builder.id
57     //must set AWS_DEFAULT_REGION in HCP Terraform as env var
58     provisioner "local-exec" {
59         command = "aws codebuild start-build --project-name ${aws_codebuild_project.ecomm-frontend-builder.name}"
60     }
61 }
```

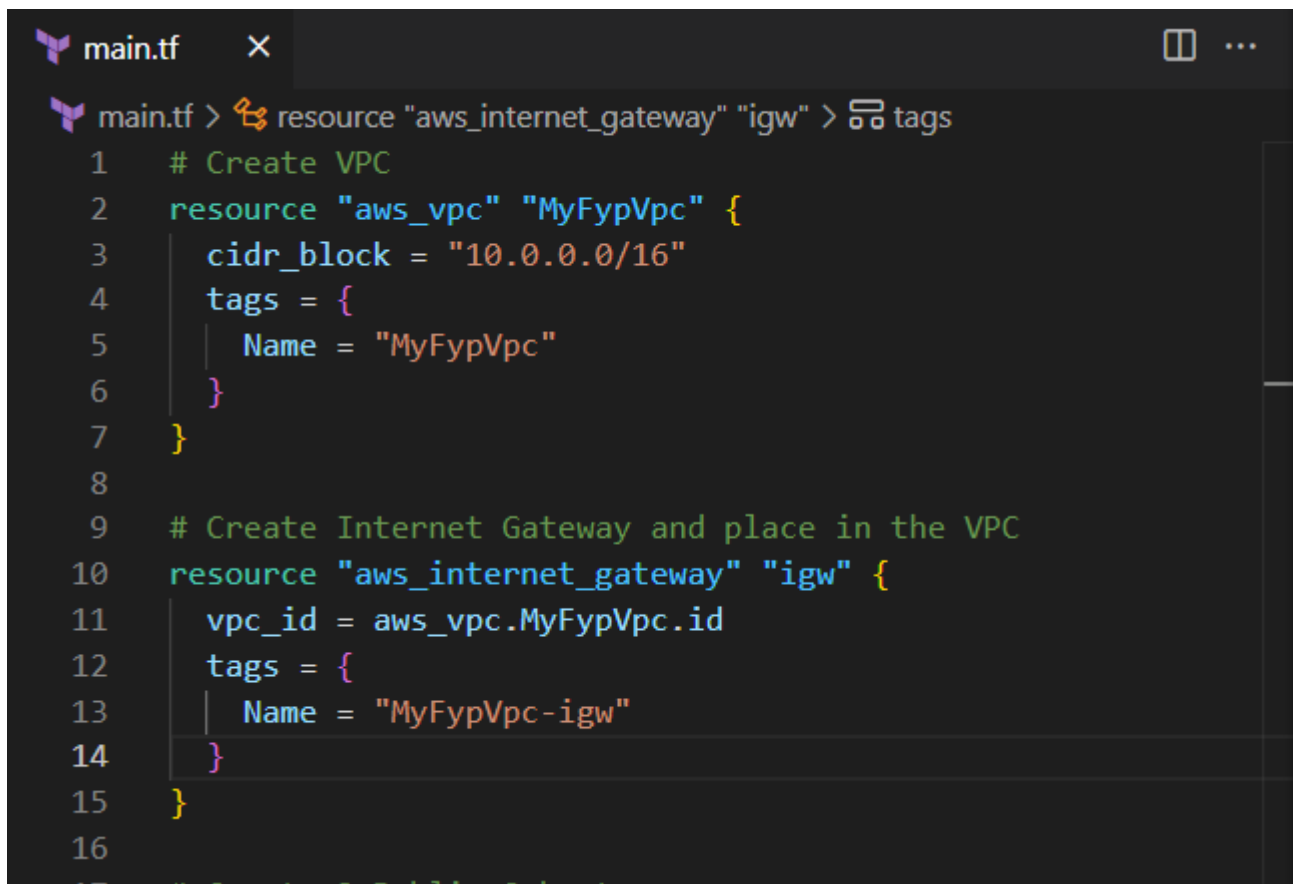
Figure 28 - AWS CodeBuild Trigger Upon Creation

This configuration ensures that a build is triggered automatically after the CodeBuild project is created.

- **<triggers.replace = aws_codebuild_project.ecomm-frontend-builder.id>** links the trigger to the <ecomm-frontend-builder> project when the id changes or when its created.
- **<provisioner "local-exec">** executes the build process by invoking the AWS CLI to start a build for the specified project.

7. Backend Systems Implementation

7.1 VPC and Internet Gateway



```
main.tf x
main.tf > resource "aws_internet_gateway" "igw" > tags
1  # Create VPC
2  resource "aws_vpc" "MyFypVpc" {
3      cidr_block = "10.0.0.0/16"
4      tags = {
5          Name = "MyFypVpc"
6      }
7  }
8
9  # Create Internet Gateway and place in the VPC
10 resource "aws_internet_gateway" "igw" {
11     vpc_id = aws_vpc.MyFypVpc.id
12     tags = {
13         Name = "MyFypVpc-igw"
14     }
15 }
16
```

Figure 29 - VPC and Internet Gateway Configuration

The backend architecture starts off by creating the VPC and attachment of the Internet Gateway to the VPC. A VPC is a virtual network dedicated to the AWS account, allowing to define an isolated section of the AWS cloud. It provides a secure environment for launching and managing resources.

Attaching an IGW to the VPC provides outbound internet access for instances in public subnets, which is essential for any public-facing applications.

7.2 Subnets

```
17  # Create 2 Public Subnets
18  ∨ resource "aws_subnet" "public-subnet-1" {
19      vpc_id          = aws_vpc.MyFypVpc.id
20      cidr_block       = "10.0.1.0/24"
21      availability_zone = "ap-southeast-1a"
22      map_public_ip_on_launch = true # Enable public IPs
23      tags = {
24          Name = "MyFypVpc-public-subnet-1"
25      }
26  }
27
28  ∨ resource "aws_subnet" "public-subnet-2" {
29      vpc_id          = aws_vpc.MyFypVpc.id
30      cidr_block       = "10.0.2.0/24"
31      availability_zone = "ap-southeast-1b"
32      map_public_ip_on_launch = true # Enable public IPs
33      tags = {
34          Name = "MyFypVpc-public-subnet-2"
35      }
36  }
37
```

Figure 30 - Subnet Creation in the VPC

Two Subnets are created in the VPC, spreading across 2 availability zones. Line 22 and 32 in Figure 30 indicates that instances initialized in these subnets would automatically be attached a public IP address to hit public network, a requirement by the application since it contains a third party payment gateway API.

7.3 Public Route Table and Association

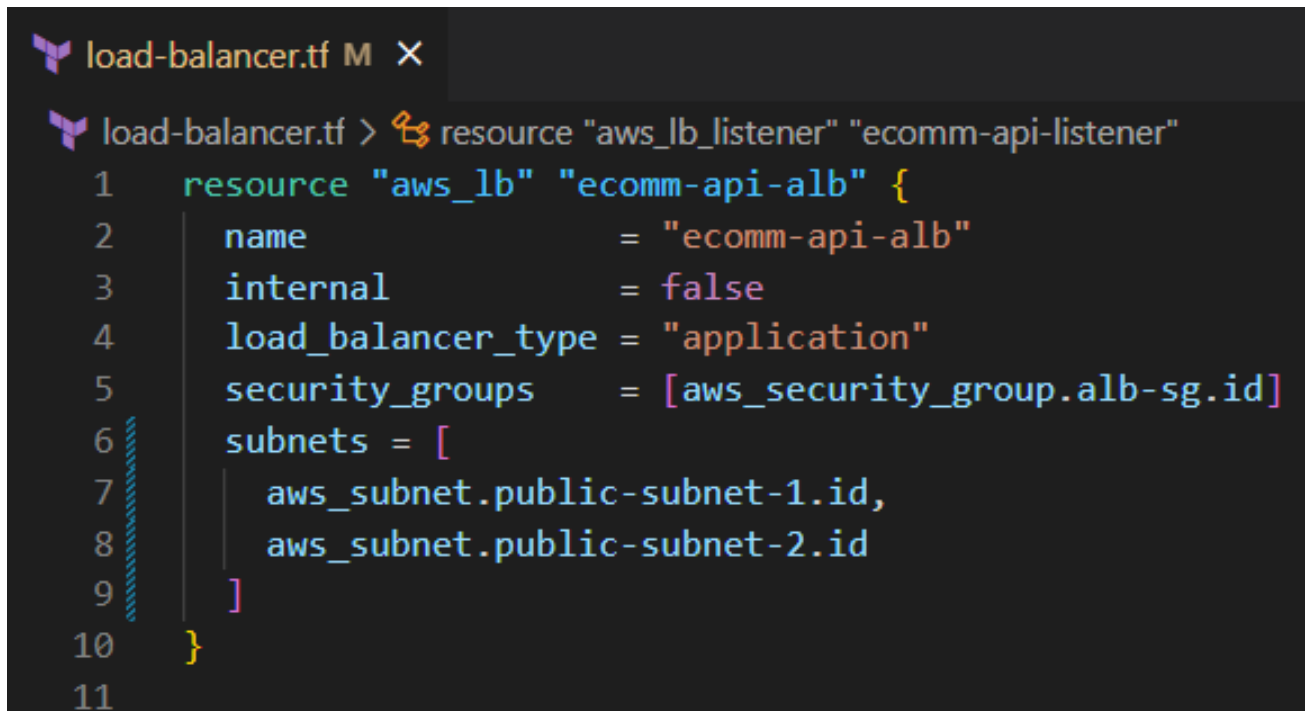
```
38 # Create Route Table for routing traffic to the Internet Gateway
39 resource "aws_route_table" "public-rt" {
40     vpc_id = aws_vpc.MyFypVpc.id
41     route {
42         cidr_block = "0.0.0.0/0"
43         gateway_id = aws_internet_gateway.igw.id
44     }
45     tags = {
46         Name = "MyFypVpc-public-rt"
47     }
48 }
49
50 # Associate the route table to the subnets
51 resource "aws_route_table_association" "public-subnet-association-1" {
52     subnet_id      = aws_subnet.public-subnet-1.id
53     route_table_id = aws_route_table.public-rt.id
54 }
55
56 resource "aws_route_table_association" "public-subnet-association_2" {
57     subnet_id      = aws_subnet.public-subnet-2.id
58     route_table_id = aws_route_table.public-rt.id
59 }
```

Figure 31 - Public Route Table and Subnet Association

The route table directs traffic from instances within the public subnet to the internet via the internet gateway. In the configuration shown in Figure 31:

- The **<aws_route_table>** resource defines a route that sends all outbound traffic (0.0.0.0/0) through the internet gateway (gateway_id = aws_internet_gateway.igw.id), allowing public internet access.
- The **<aws_route_table_association>** resources link this route table to specific public subnets (public-subnet-1 and public-subnet-2). This association ensures that instances in these subnets follow the route defined allow traffic to route to public network.

7.4 Application Load Balancer (ALB)



```
load-balancer.tf M X
load-balancer.tf > resource "aws_lb_listener" "ecomm-api-listener"
1  resource "aws_lb" "ecomm-api-alb" {
2      name                = "ecomm-api-alb"
3      internal             = false
4      load_balancer_type   = "application"
5      security_groups      = [aws_security_group.alb-sg.id]
6      subnets = [
7          aws_subnet.public-subnet-1.id,
8          aws_subnet.public-subnet-2.id
9      ]
10 }
11
```

Figure 32 - Application Load Balancer ALB Configuration

Figure 32 configures an Application Load Balancer (ALB), which is suitable for HTTP/HTTPS traffic and provides advanced routing features. The Load Balancer is configured to span across the two subnets (Created in Section 7.2) to distribute traffic evenly across instances in the declared subnets.

- **<load_balancer_type = "application">** specifies the ALB type, which is intended for web applications.
- The **<subnets>** array assigns the ALB to public subnets, allowing it to receive incoming traffic from the internet.
- **<security_groups>** attaches security groups to manage incoming and outgoing traffic to the ALB. The configuration of the security group is detailed in Appendix C: Security Group Rules for ALB.
- **<internal = false>** decides that the load balancer is not private and is set to public facing.

7.5 Creating a TLS Certification for Listener



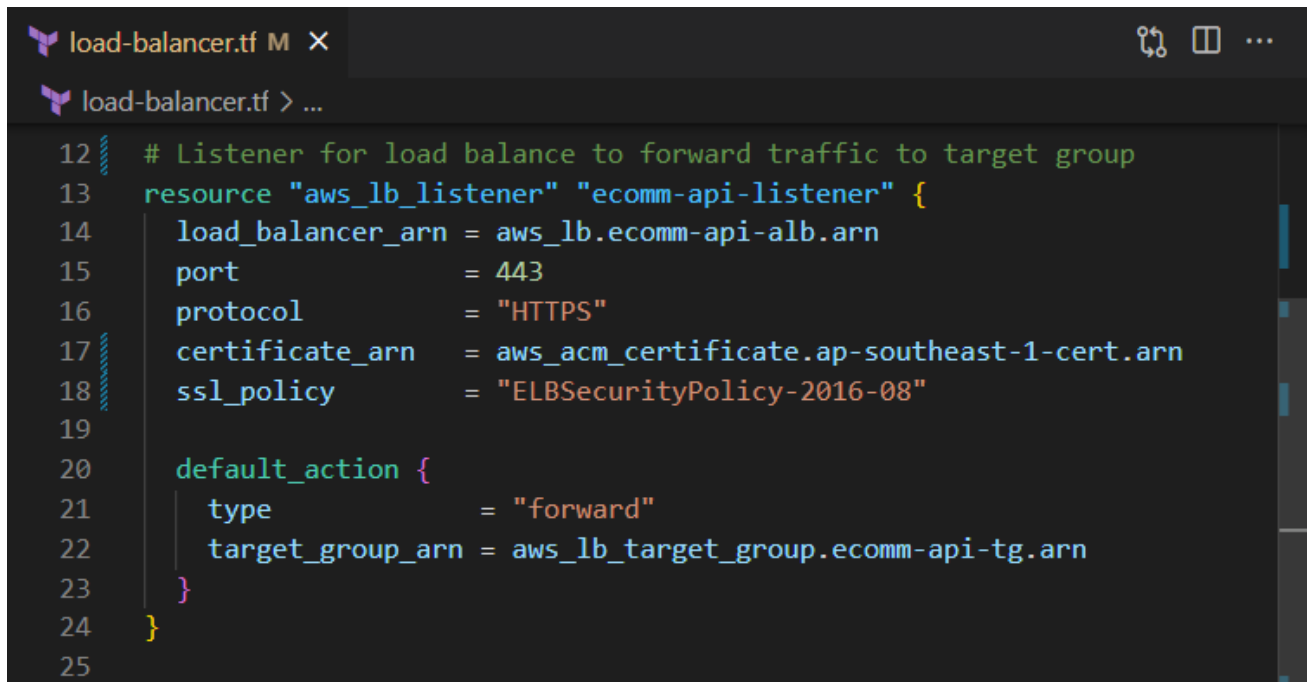
```
load-balancer.tf M X  aws-cert-manager.tf M X
aws-cert-manager.tf > resource "aws_acm_certificate" "ap-southeast-1-cert"
1  resource "aws_acm_certificate" "ap-southeast-1-cert" {
2      domain_name          = "hswg94.com"
3      subject_alternative_names = ["*.hswg94.com"]
4      validation_method    = "DNS"
5  }
6
```

Figure 33 - TLS Certificate Creation for ALB Listener

To enable secure HTTPS communication for the ALB, an SSL/TLS certificate is issued using AWS Certificate Manager (ACM).

- **<domain_name>** is specified as hswg94.com, representing the primary domain name for the application.
- **<subject_alternative_names>** includes *.hswg94.com to extend the certificate coverage to all subdomains, including ecomm.hswg94.com.
- **<validation_method>** is set to DNS, meaning domain ownership is verified by adding DNS records to the Route 53 hosted zone. The steps to validating the certificate can be found in Appendix I: Validating Certificates with Route 53.

7.6 Configuring a Listener

A screenshot of a code editor window with a dark theme. The title bar shows 'load-balancer.tf' with a magnifying glass icon and a close button. The editor content shows Terraform HCL code for configuring an ALB listener. Line numbers 12 through 25 are visible on the left. The code defines a resource 'aws_lb_listener' named 'ecomm-api-listener' with attributes for 'load_balancer_arn', 'port', 'protocol', 'certificate_arn', and 'ssl_policy'. It also includes a 'default_action' block with 'type' set to 'forward' and 'target_group_arn' pointing to 'aws_lb_target_group.ecomm-api-tg.arn'.

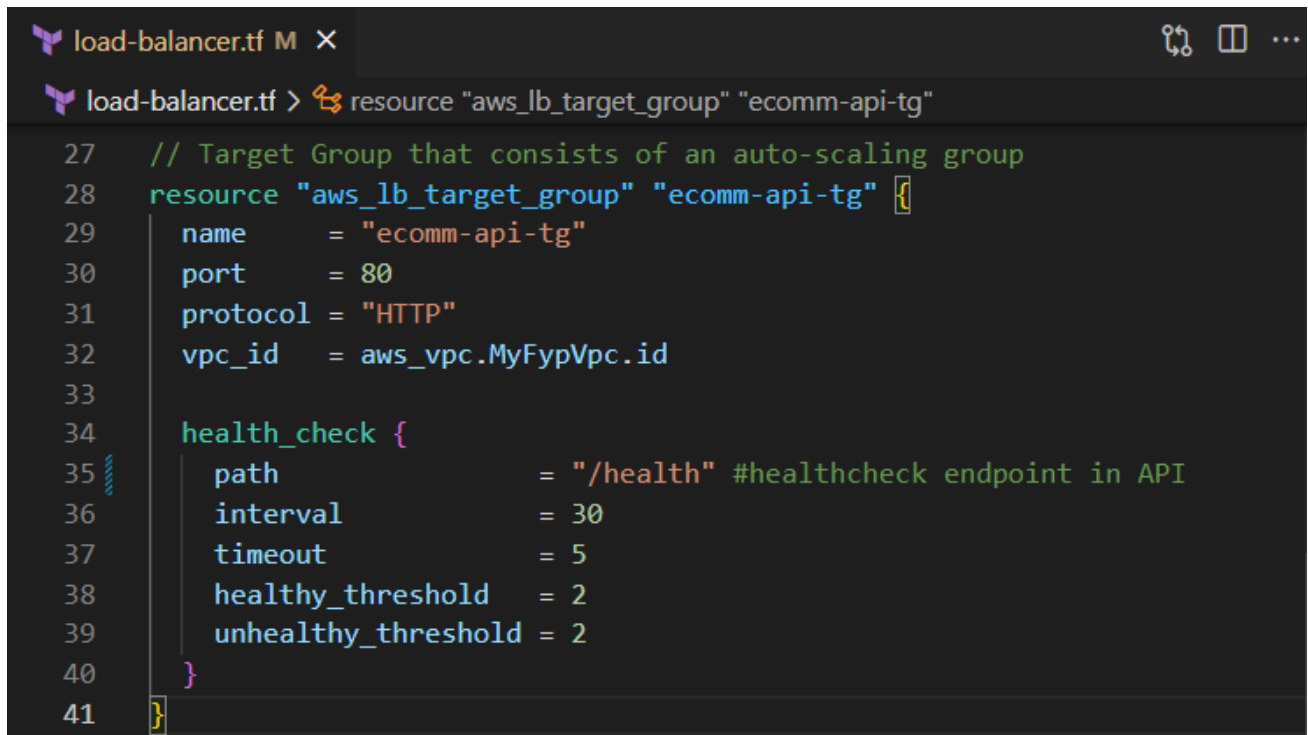
```
12 # Listener for load balance to forward traffic to target group
13 resource "aws_lb_listener" "ecomm-api-listener" {
14     load_balancer_arn = aws_lb.ecomm-api-alb.arn
15     port              = 443
16     protocol          = "HTTPS"
17     certificate_arn    = aws_acm_certificate.ap-southeast-1-cert.arn
18     ssl_policy         = "ELBSecurityPolicy-2016-08"
19
20     default_action {
21         type            = "forward"
22         target_group_arn = aws_lb_target_group.ecomm-api-tg.arn
23     }
24 }
25
```

Figure 34 - ALB Listener Configuration

The Listener is configured for the Load Balancer to handle incoming connections on a specific port and protocol (in this case, HTTPS on port 443). It forwards the traffic to the associated target group based on routing rules.

- **<port = 443>** and **<protocol = "HTTPS">** configure the listener to handle secure HTTPS traffic.
- **<certificate_arn>** links the listener to a validated SSL certificate managed by AWS ACM, enabling encrypted connections with TLS certification.
- **<default_action>** specifies the target group to forward traffic to, ensuring requests are directed to the correct backend instances.

7.7 Target Group

A screenshot of a code editor window titled 'load-balancer.tf'. The editor shows a Terraform configuration for an AWS Target Group. The configuration is as follows:

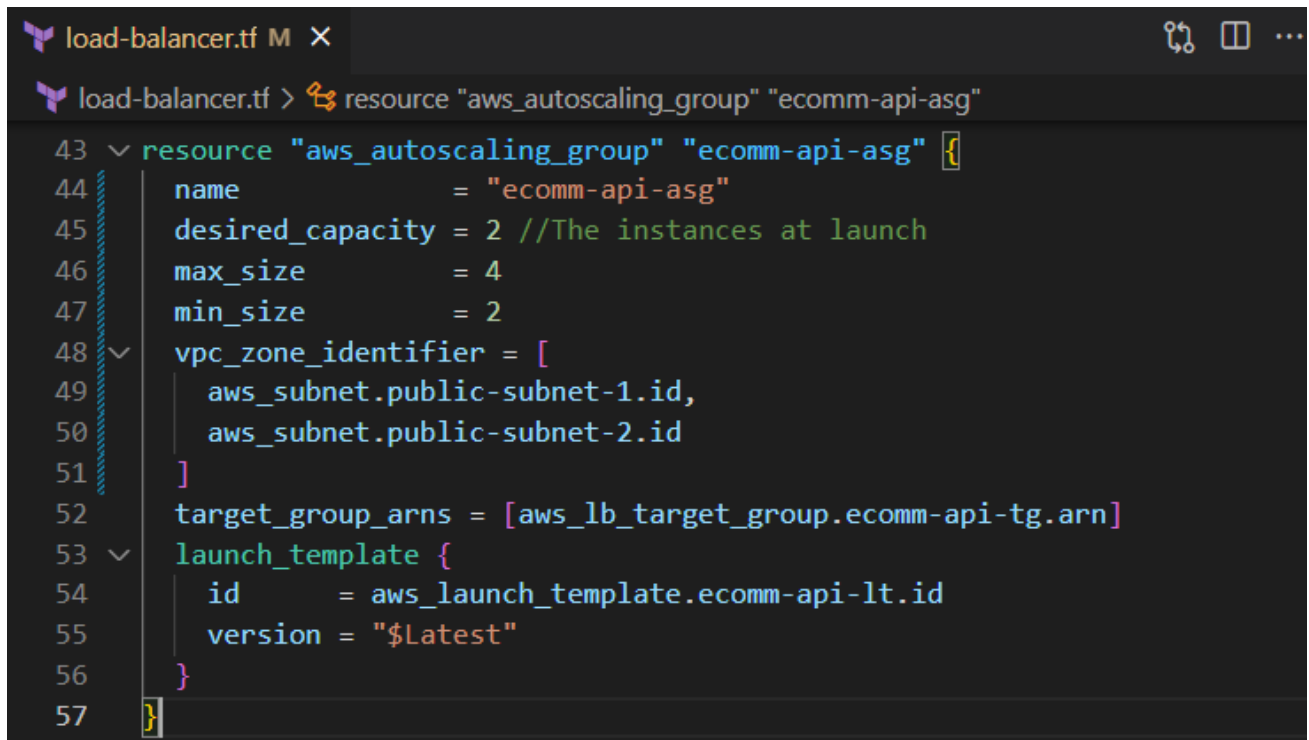
```
27 // Target Group that consists of an auto-scaling group
28 resource "aws_lb_target_group" "ecomm-api-tg" {
29     name      = "ecomm-api-tg"
30     port      = 80
31     protocol  = "HTTP"
32     vpc_id    = aws_vpc.MyFypVpc.id
33
34     health_check {
35         path            = "/health" #healthcheck endpoint in API
36         interval        = 30
37         timeout         = 5
38         healthy_threshold = 2
39         unhealthy_threshold = 2
40     }
41 }
```

Figure 35 - Target Group Configuration

The Target Group receives the traffic from the Load Balancer, and forwards them to the instances in the Auto-Scaling Group defined in Section 7.6. It also defines health checks to monitor instance health and ensure only healthy instances receive traffic.

- **<port = 80>** ensures that the target group routes incoming traffic to port 80 on each instance as the application in the instances are running on port 80.
- **<protocol = "HTTP">** specifies the protocol for health checks and traffic routing to instances.
- **<health_check>** settings define the path, interval, and thresholds for determining a healthy status.

7.8 Auto-Scaling Group



```
load-balancer.tf M X
load-balancer.tf > resource "aws_autoscaling_group" "ecomm-api-asg"

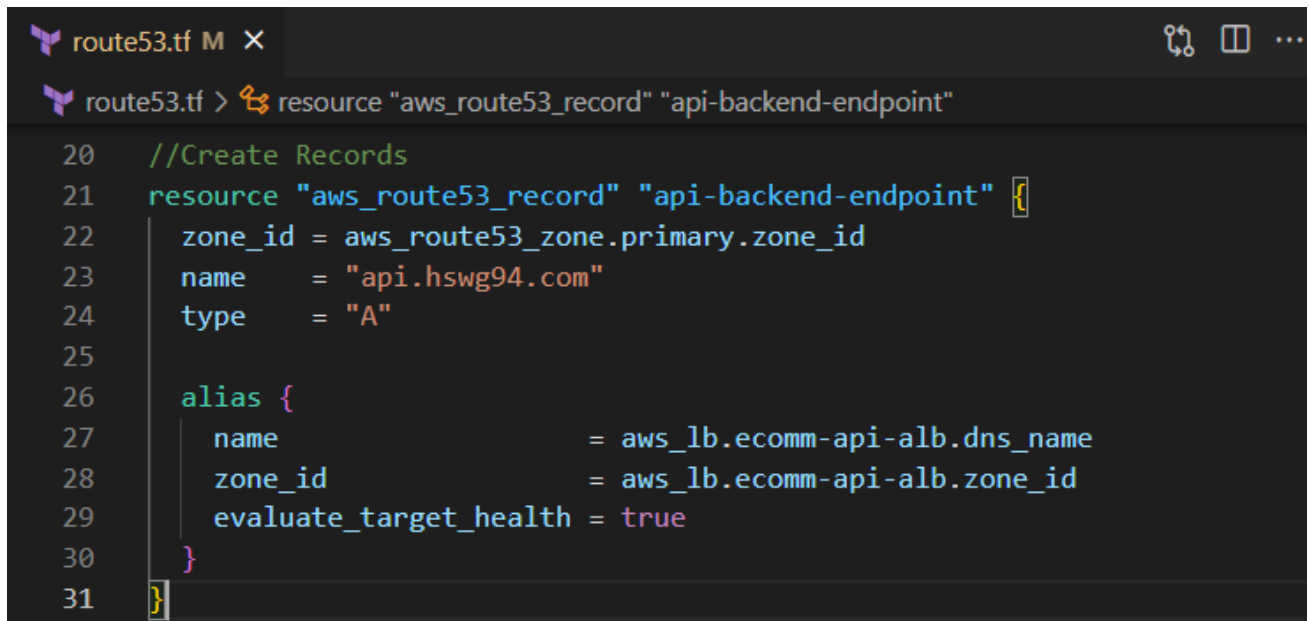
43 resource "aws_autoscaling_group" "ecomm-api-asg" {
44     name                = "ecomm-api-asg"
45     desired_capacity    = 2 //The instances at launch
46     max_size            = 4
47     min_size            = 2
48     vpc_zone_identifier = [
49         aws_subnet.public-subnet-1.id,
50         aws_subnet.public-subnet-2.id
51     ]
52     target_group_arns   = [aws_lb_target_group.ecomm-api-tg.arn]
53     launch_template {
54         id          = aws_launch_template.ecomm-api-lt.id
55         version     = "$Latest"
56     }
57 }
```

Figure 36 - Auto Scaling Group Configuration

The Auto-Scaling Group (ASG) dynamically manages the number of EC2 instances in response to demand, ensuring the backend scales to handle traffic spikes and scales down to save costs during low traffic.

- **<desired_capacity>**, **<max_size>**, and **<min_size>** define the number of instances at different load levels.
- **<target_group_arns>** attaches the ASG to the target group, allowing it to automatically register and deregister instances as they scale.
- **<launch_template>** defines the configuration for EC2 instances, including instance type, AMI, and security settings. The launch template is triggered when the Auto Scaling Group initializes, scales up or replaces instances to ensure consistent configuration between all instances in the group. The configuration of the launch template is detailed in Appendix D: EC2 Launch Template.

7.9 Creating an ALIAS record for ALB



```
route53.tf M X
route53.tf > resource "aws_route53_record" "api-backend-endpoint"

20 //Create Records
21 resource "aws_route53_record" "api-backend-endpoint" {
22     zone_id = aws_route53_zone.primary.zone_id
23     name     = "api.hswg94.com"
24     type     = "A"
25
26     alias {
27         name           = aws_lb.ecomm-api-alb.dns_name
28         zone_id        = aws_lb.ecomm-api-alb.zone_id
29         evaluate_target_health = true
30     }
31 }
```

Figure 37 - DNS ALIAS Record for ALB

This configuration sets up DNS records in Route 53 and creates an A record for `api.hswg94.com` to point to the Application Load Balancer (ALB).

- **<zone_id>** specifies the hosted zone ID for `hswg94.com`. The creation of the hosted zone is detailed in Appendix H: Route 53 Hosted Zone.
- **<name = "api.hswg94.com">** is the subdomain for the backend API.
- **<name = aws_lb.ecomm-api-alb.dns_name>** points the A record to the ALB's DNS name.
- **<zone_id = aws_lb.ecomm-api-alb.zone_id>** links it to the ALB's hosted zone.
- **<evaluate_target_health = true>** enables health checks on the ALB, ensuring that traffic is only directed to healthy endpoints.

8. Backend CI/CD Pipeline Implementation

8.1 AWS CodeDeploy Application Type

```
69  ∨ resource "aws_codedeploy_app" "ecomm-api" {  
70      |     name           = "ecomm-api"  
71      |     compute_platform = "Server"  
72      | }
```

Figure 38 - AWS CodeDeploy Application Configuration

The `<aws_codedeploy_app>` resource in AWS CodeDeploy represents an application that will be deployed. It acts as an overarching structure for managing and tracking deployment groups and associated configurations for the eCommerce backend API.

- **<name>** defines the name of the CodeDeploy application. Here, it's set to "ecomm-api", which will be used to refer to this application within AWS CodeDeploy.
- **<compute_platform>** specifies the environment type for the deployment. It's set to "Server", indicating that this application will be deployed on EC2 instances or on-premises servers, as opposed to Lambda or ECS.

8.2 AWS CodeDeploy Deployment Group

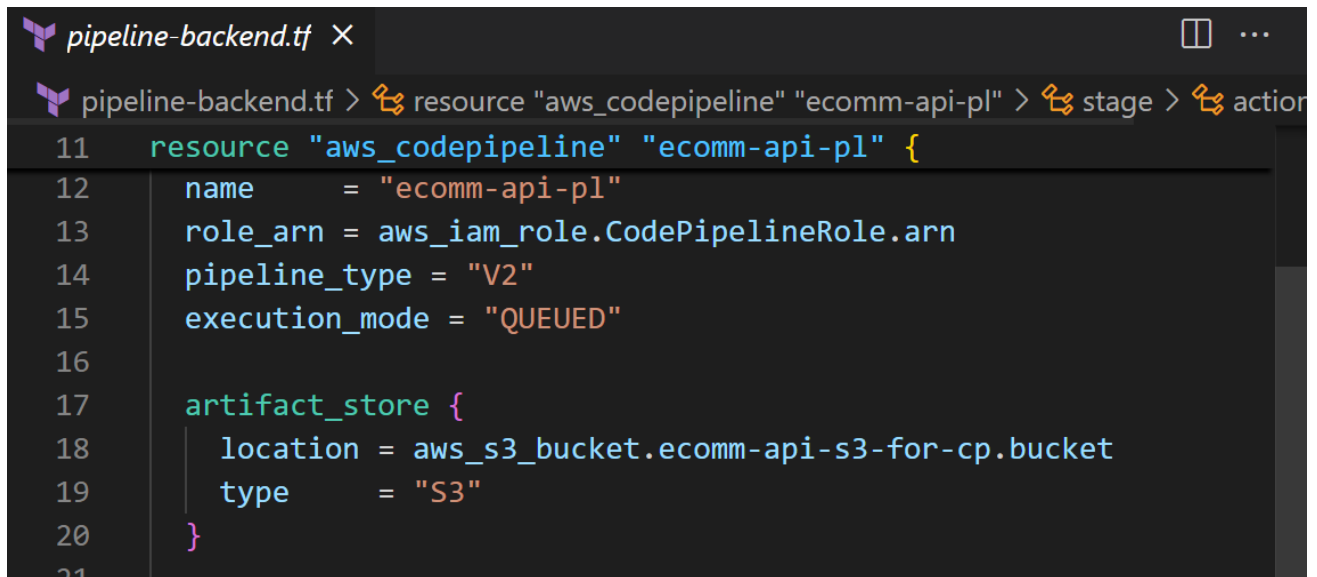
```
57 resource "aws_codedeploy_deployment_group" "ecomm-api-dg" {
58     app_name                = aws_codedeploy_app.ecomm-api.name
59     deployment_group_name   = "ecomm-api-dg"
60     # deployment_config_name = "CodeDeployDefault.AllAtOnce"
61     deployment_config_name  = "CodeDeployDefault.OneAtATime"
62     service_role_arn        = aws_iam_role.CodeDeployRole.arn
63     deployment_style {
64         deployment_type = "IN_PLACE"
65     }
66     autoscaling_groups = [aws_autoscaling_group.ecomm-api-asg.id]
67 }
```

Figure 39 - AWS CodeDeploy Deployment Group Configuration

The **<aws_codedeploy_deployment_group>** defines a deployment group within the specified CodeDeploy application (ecomm-api). The deployment group is essentially a collection of EC2 instances (here, managed by an Auto Scaling Group) that CodeDeploy will target for deployment.

- **<app_name>** references the application name created in the `aws_codedeploy_app` resource, ensuring that this deployment group is associated with the ecomm-api application.
- **<deployment_group_name>** sets the name of the deployment group, here identified as "ecomm-api-dg".
- **<deployment_config_name>** specifies the deployment strategy. In this case, "CodeDeployDefault.OneAtATime" is used, meaning the deployment will update instances one at a time, minimizing potential downtime by ensuring at least one instance remains active.
- **<service_role_arn>** is the ARN of the IAM role, the attached IAM role has the necessary permissions for CodeDeploy to access and manage resources such as EC2 instances and Auto Scaling groups. The configuration of the IAM role is detailed in Appendix G: IAM Role for CodeDeploy.
- **<deployment_style>** configures the type of deployment. **<deployment_type>** is set to "IN_PLACE", meaning CodeDeploy will update the application directly on the existing instances without replacement.
- **<autoscaling_groups>** Specifies the Auto Scaling Group (ecomm-api-asg) that manages the set of EC2 instances for this deployment group. Using an Auto Scaling Group ensures that any instance within this group can be targeted for deployment and that the infrastructure scales based on demand.

8.3 AWS CodePipeline



```
pipeline-backend.tf X
pipeline-backend.tf > resource "aws_codepipeline" "ecomm-api-pl" > stage > action
11 resource "aws_codepipeline" "ecomm-api-pl" {
12     name      = "ecomm-api-pl"
13     role_arn  = aws_iam_role.CodePipelineRole.arn
14     pipeline_type = "V2"
15     execution_mode = "QUEUED"
16
17     artifact_store {
18         location = aws_s3_bucket.ecomm-api-s3-for-cp.bucket
19         type     = "S3"
20     }
21 }
```

Figure 40 - AWS CodePipeline Configuration

The Terraform configuration in Figure 40 defines an AWS CodePipeline resource named `ecomm-api-pl`, which automates the deployment process for the backend API.

- **<name>** specifies the name of the pipeline as `ecomm-api-pl`. the naming conventions help in identifying the specific pipeline dedicated to the backend API service.
- **<role_arn>** is used to assign an IAM role (`aws_iam_role.CodePipelineRole.arn`) that grants it the necessary permissions to interact with other AWS services, such as S3 for storing artifacts and CodeDeploy for deployment. This role-based access control ensures that CodePipeline operates securely within the defined permissions. For details on the configuration of the IAM role refer to Appendix F: IAM Role for CodePipeline.
- **<pipeline_type>** is set to `V2` which provides enhanced functionality such as the `QUEUED` execution mode.
- **<execution_mode>** is set to `QUEUED`, this mode controls how the pipeline handles multiple concurrent executions. With `QUEUED`, new executions wait in a queue if another instance of the pipeline is currently running, ensuring sequential processing and avoiding conflicts.
- **<artifact_store>** configures an S3 bucket (`aws_s3_bucket.ecomm-api-s3-for-cp.bucket`) as the artifact store for the pipeline. Artifacts generated during the pipeline stages are stored here, allowing CodePipeline to pass these artifacts between stages securely. The type is specified as `"S3"`, indicating that the artifacts will be stored in an S3 bucket. For details on how the S3 bucket is created refer to Appendix A: S3 Bucket Creation.

8.3.1 AWS CodePipeline - Source

```
22  stage {
23      name = "Source"
24      action {
25          category      = "Source"
26          owner         = "AWS"
27          name          = "ApplicationSource"
28          provider      = "CodeStarSourceConnection"
29          version       = "1"
30          output_artifacts = ["source_output"]
31          configuration = {
32              ConnectionArn      = "arn:aws:codeconnections:ap-southeast-1:123456789012:connection:my-connection"
33              FullRepositoryId   = "hswg94/ecommerce-express-api"
34              BranchName        = "main"
35              DetectChanges      = "true"
36          }
37      }
38  }
```

Figure 41 - AWS CodePipeline Source Stage Configuration

The Source stage in AWS CodePipeline pulls the latest code from the GitHub repository whenever a new commit is made to the main branch. Key components include:

- **<category>** specifies this action as "Source", designating it as the stage where code is retrieved from the repository to initiate the pipeline process.
- **<owner>** identifies AWS as the action owner, indicating that this is a built-in AWS action within CodePipeline.
- **<name>** labels the action as "ApplicationSource", a recognizable identifier for the source action, which simplifies pipeline management and debugging.
- **<provider>** uses "CodeStarSourceConnection" to securely connect to GitHub, enabling seamless integration with third-party source control while maintaining secure access.
- **<version>** defines the action version as "1", specifying the version of the action used to maintain compatibility and stability in the pipeline.
- **<output_artifacts>** sets "source_output" as the name of the retrieved code artifact, which serves as the output for this stage and is passed to subsequent stages for further processing.
- **<ConnectionArn>** provides the ARN to connect to GitHub, allowing AWS CodePipeline to authenticate and retrieve code from the specified GitHub repository.

- **<FullRepositoryId>** specifies the repository as "hswg94/ecommerce-api", combining the GitHub username and repository name to uniquely identify the repository to CodePipeline.
- **<BranchName>** monitors the "main" branch for updates, ensuring that only changes to this primary branch trigger the pipeline.
- **<DetectChanges>** is set to "true" to enable automatic pipeline triggers on new commits to the monitored branch, supporting continuous integration by ensuring the pipeline always uses the latest code.

8.3.2 AWS CodePipeline - Deploy

```

40     stage {
41         name = "Deploy"
42         action {
43             category      = "Deploy"
44             owner          = "AWS"
45             name           = "ApplicationDeploy"
46             provider       = "CodeDeploy"
47             input_artifacts = ["source_output"]
48             version        = "1"
49             configuration = {
50                 ApplicationName = aws_codedeploy_app.ecomm-api.name
51                 DeploymentGroupName = aws_codedeploy_deployment_group.ecomm-api.name
52             }
53         }
54     }
55 }

```

Figure 42 - AWS CodePipeline Deploy Stage Configuration

This Deploy stage in CodePipeline uses AWS CodeDeploy to push the latest application code (from the Source stage) to the specified deployment group. By connecting the CodePipeline to CodeDeploy with defined application and deployment group names, this stage ensures that the application is consistently deployed to the intended environment.

- **<category>** specifies this action as "Deploy", indicating that this stage handles the deployment of the application to the target environment.
- **<owner>** identifies AWS as the action owner, showing that this is a native AWS action within CodePipeline.

- **<name>** labels the action as "ApplicationDeploy", making it easy to identify this specific deployment action in the pipeline.
- **<provider>** uses "CodeDeploy" to deploy the application to EC2 instances or other compute resources managed by AWS CodeDeploy.
- **<input_artifacts>** receives the "source_output" artifact from the Source stage, which contains the latest code to be deployed.
- **<version>** defines the action version as "1", specifying the version of this deployment action for compatibility.
- **<ApplicationName>** specifies the name of the CodeDeploy application, as defined in the aws_codedeploy_app resource. This links the pipeline to the specific application that will be deployed.
- **<DeploymentGroupName>** defines the CodeDeploy deployment group, linking to the aws_codedeploy_deployment_group resource. This group manages the set of EC2 instances or other resources targeted for deployment.

9. Project Evaluation

This project successfully implemented a scalable, and highly available using IaC tools like Terraform and deployed an ecommerce application with 2 establish CI/CD pipelines. The ease of provisioning resources using IaC addressed manual deployment inefficiencies and established a foundation for future enhancements in monitoring and security. The implementation reduced provisioning time by > 90%, taking up an average of only 8 minutes for a 3-tier web application with 2 CI/CD pipelines, it also eliminated human error completely, and allowing the ease of replicability through a single terraform command.

Triggered via CLI

✓ Applied

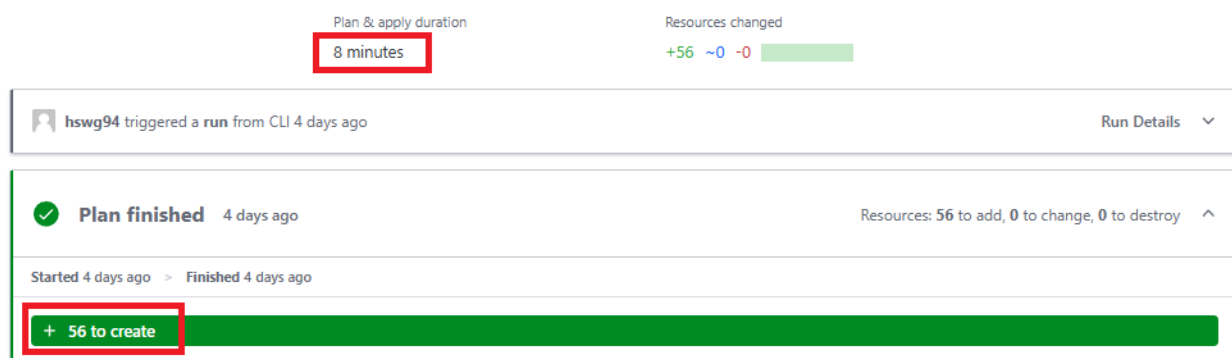


Figure 43 – Total Deployment Time

Metric	Outcome
Provisioning Time	Provisioning time was reduced by over 90%, averaging 8 minutes for a 3-tier application with two CI/CD pipelines.
Human Error Rates	Eliminated via Code and Automation.
Replicability Rate	All systems configurations are consistent with every attempt due to using code and automation.
Development Costs	Cloud operates on a pay per use model, development costs were greatly reduced due to the ease of taking down and spinning up systems using terraform command, ensuring that the resources only stay running when required.

Table 9 - Evaluation Metrics and Outcomes

10. Future Plans

10.1 Multi-Region Backend

In future, the project could include deploying backend services in an additional AWS region and incorporating Route 53 latency-based routing to direct users to the API backend that is the closest to their region. This will optimize user latency, improve availability, and enhance fault tolerance across geographical locations.

10.2 Logging and Monitoring

Logging and monitoring are essential for ensuring cloud infrastructure reliability and security. While basic monitoring was implemented using AWS CloudWatch for tracking instance health, the project lacked a centralized logging solution due to time constraints. Future enhancements include centralized logging with AWS CloudWatch Logs and alarms for proactive monitoring.

10.3 Security

The project did not incorporate AWS WAF (Web Application Firewall) or NAT Gateway, both of which are essential for enhancing security and network design. AWS WAF could have provided protection against common web vulnerabilities like SQL injection and cross-site scripting (XSS), which would strengthen the application layer's defences. The absence of a NAT Gateway resulted in certain instances using public IPs for outbound internet access, which poses potential security risks.

11. Ethical Considerations

Ethical considerations formed a large part during the project implementation. While sensitive user data was not processed, precautions were taken to secure AWS credentials and environment variables, preventing unauthorized access.

Resource usage was optimized to reduce environmental impact by leveraging AWS free-tier services and deploying only essential components. As the project scales, encryption for data at rest and in transit will be critical for protecting sensitive information.

Regular security audits will help identify vulnerabilities, and compliance with data protection regulations such as GDPR will ensure ethical and legal responsibility. These measures will further enhance the infrastructure's ethical and professional integrity.

12. Appendices

12.1 Appendix A: S3 Bucket Creation

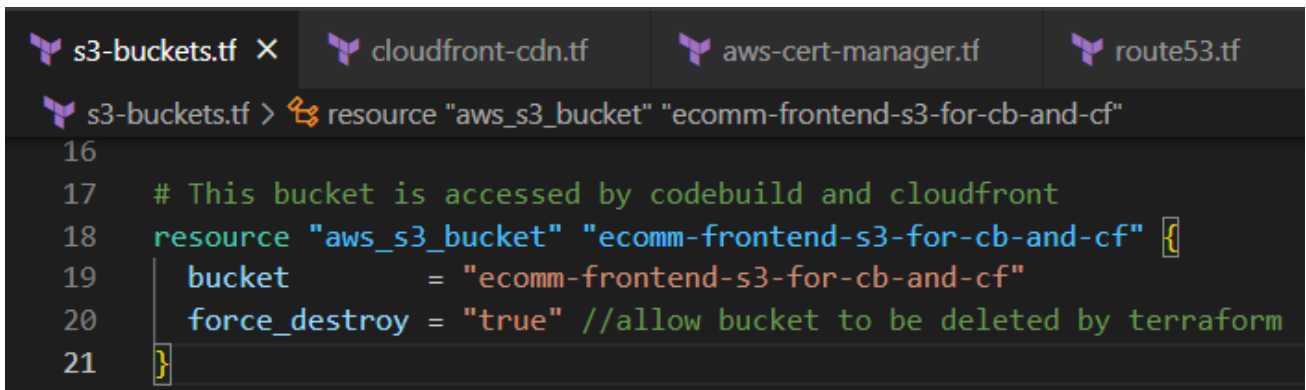
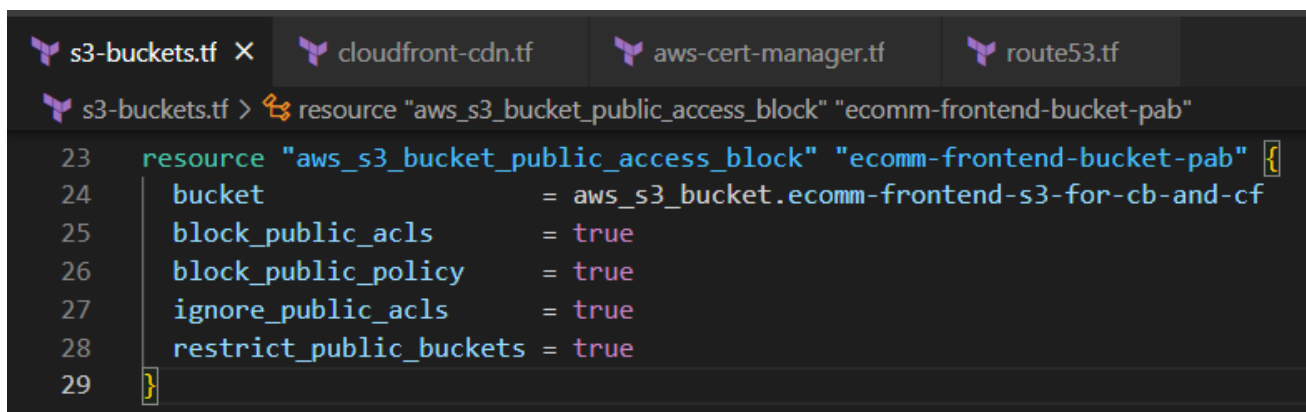
A screenshot of a code editor with a dark theme. At the top, there are four tabs: 's3-buckets.tf' (active), 'cloudfront-cdn.tf', 'aws-cert-manager.tf', and 'route53.tf'. The active tab shows Terraform code. Line 16 is empty. Line 17 is a comment: '# This bucket is accessed by codebuild and cloudfront'. Line 18 is 'resource "aws_s3_bucket" "ecomm-frontend-s3-for-cb-and-cf" {' with a closing brace. Line 19 is ' bucket = "ecomm-frontend-s3-for-cb-and-cf"'. Line 20 is ' force_destroy = "true" //allow bucket to be deleted by terraform'. Line 21 is a closing brace '}'.

Figure 44 - S3 Bucket Creation Using Terraform

The S3 bucket is created using Terraform to store the static files built by the React frontend application through AWS CodeBuild. The build files in the bucket are later served to users through the CloudFront Distribution.

- **<bucket>** is named (ecomm-frontend-s3-for-cb-and-cf) as a unique name is required.
- **<force_destroy>** attribute is set to true to allow Terraform to delete the bucket, allowing full automation.

12.1.1 S3 Bucket Public Access Block



```
s3-buckets.tf X cloudfront-cdn.tf aws-cert-manager.tf route53.tf
s3-buckets.tf > resource "aws_s3_bucket_public_access_block" "ecomm-frontend-bucket-pab"

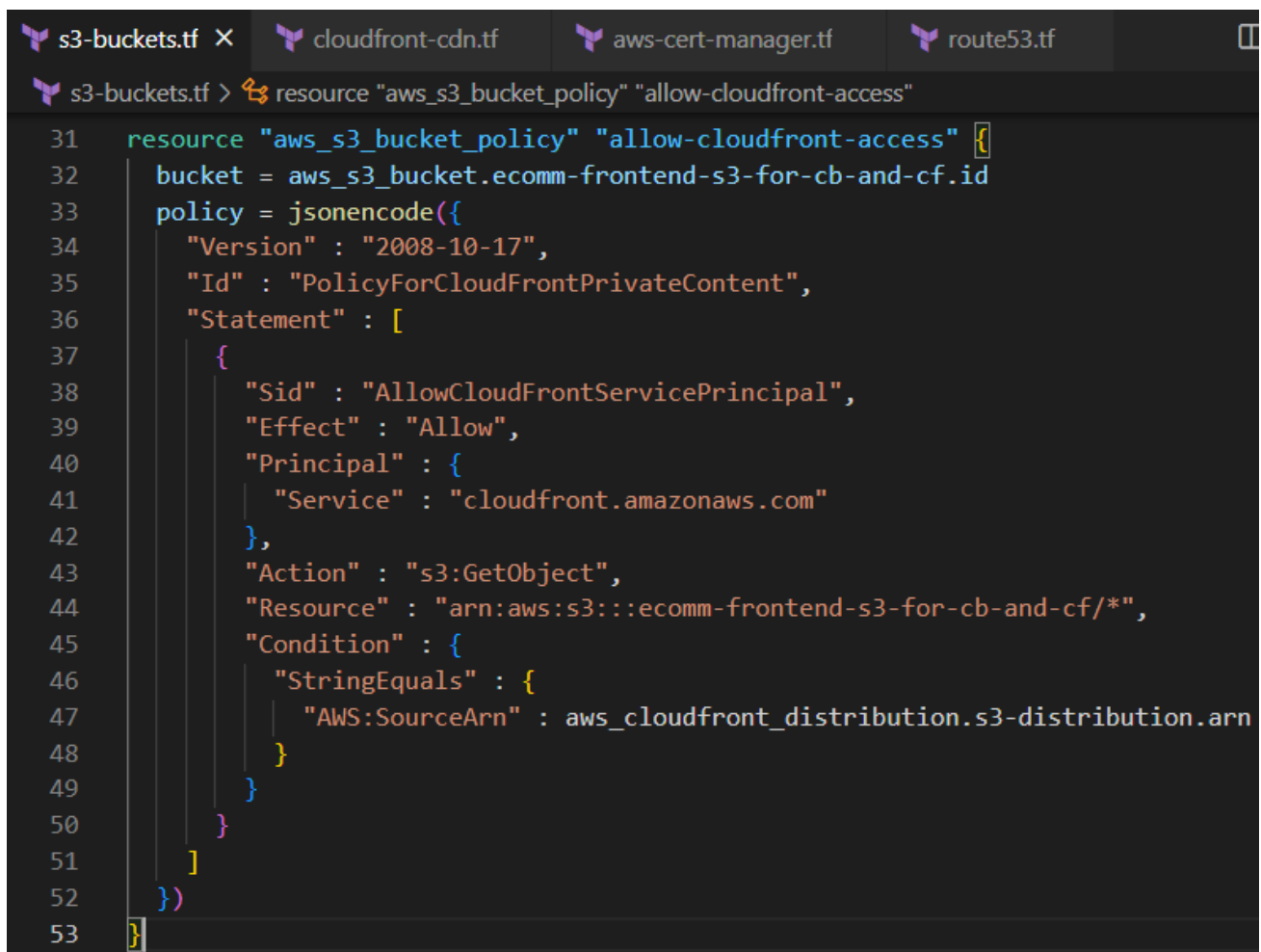
23 resource "aws_s3_bucket_public_access_block" "ecomm-frontend-bucket-pab" {}
24     bucket = aws_s3_bucket.ecomm-frontend-s3-for-cb-and-cf
25     block_public_acls = true
26     block_public_policy = true
27     ignore_public_acls = true
28     restrict_public_buckets = true
29 }
```

Figure 45 - S3 Bucket Public Access Block

To ensure security, public access needs to be blocked and the configuration in Figure 45 ensures no public access is allowed.

- **<bucket>** is referenced as `aws_s3_bucket.ecomm-frontend-s3-for-cb-and-cf.id` to apply the public access block settings to the previously created bucket.
- **<block_public_acls>** is set to `true` to prevent the application of public access control lists (ACLs) on the bucket or its objects.
- **<block_public_policy>** is set to `true` to block any public bucket policies from being attached.
- **<ignore_public_acls>** is set to `true` to ensure any existing public ACLs are ignored, effectively denying public access to all objects.
- **<restrict_public_buckets>** is set to `true` to ensure the bucket can only be accessed via authenticated requests, adding an additional layer of security.

12.1.2 S3 Bucket Policy for Origin Access Control



```
s3-buckets.tf X cloudfront-cdn.tf aws-cert-manager.tf route53.tf
s3-buckets.tf > resource "aws_s3_bucket_policy" "allow-cloudfront-access"

31 resource "aws_s3_bucket_policy" "allow-cloudfront-access" {
32     bucket = aws_s3_bucket.ecomm-frontend-s3-for-cb-and-cf.id
33     policy = jsonencode({
34         "Version" : "2008-10-17",
35         "Id" : "PolicyForCloudFrontPrivateContent",
36         "Statement" : [
37             {
38                 "Sid" : "AllowCloudFrontServicePrincipal",
39                 "Effect" : "Allow",
40                 "Principal" : {
41                     "Service" : "cloudfront.amazonaws.com"
42                 },
43                 "Action" : "s3:GetObject",
44                 "Resource" : "arn:aws:s3:::ecomm-frontend-s3-for-cb-and-cf/*",
45                 "Condition" : {
46                     "StringEquals" : {
47                         "AWS:SourceArn" : aws_cloudfront_distribution.s3-distribution.arn
48                     }
49                 }
50             }
51         ]
52     })
53 }
```

Figure 46 - S3 Bucket Policy for Origin Access Control

The bucket policy ensures allows CloudFront Distribution to fetch objects from the S3 Bucket.

- **<Principal>** specifies AWS CloudFront as the only service allowed to access the bucket (cloudfront.amazonaws.com).
- **<Action>** grants the s3:GetObject permission, enabling CloudFront to retrieve static files from the s3 bucket.
- **<condition>** restricts access only to requests originating specifically from the specified CloudFront distribution, identified by its ARN (aws_cloudfront_distribution.s3-distribution.arn).

12.2 Appendix B: IAM Role for CodeBuild

12.2.1 CodeBuild IAM Role Creation

```
112 # CodeBuild Role
113 v resource "aws_iam_role" "CodeBuildRole" {
114     name = "CodeBuildRole"
115     assume_role_policy = jsonencode({
116         "Version" : "2012-10-17",
117         "Statement" : [
118             {
119                 "Effect" : "Allow",
120                 "Principal" : {
121                     "Service" : "codebuild.amazonaws.com"
122                 },
123                 "Action" : "sts:AssumeRole"
124             }
125         ]
126     })
127 }
```

Figure 47 - IAM Role Creation for CodeBuild

The IAM role is created to allow CodeBuild to assume the role and access AWS resources.

- **<resource = "aws_iam_role" "CodeBuildRole">** defines the IAM role for CodeBuild.
- **<assume_role_policy>** specifies the trust relationship policy allowing AWS CodeBuild service to assume the role.

12.2.2 CodeBuild IAM Role Policies

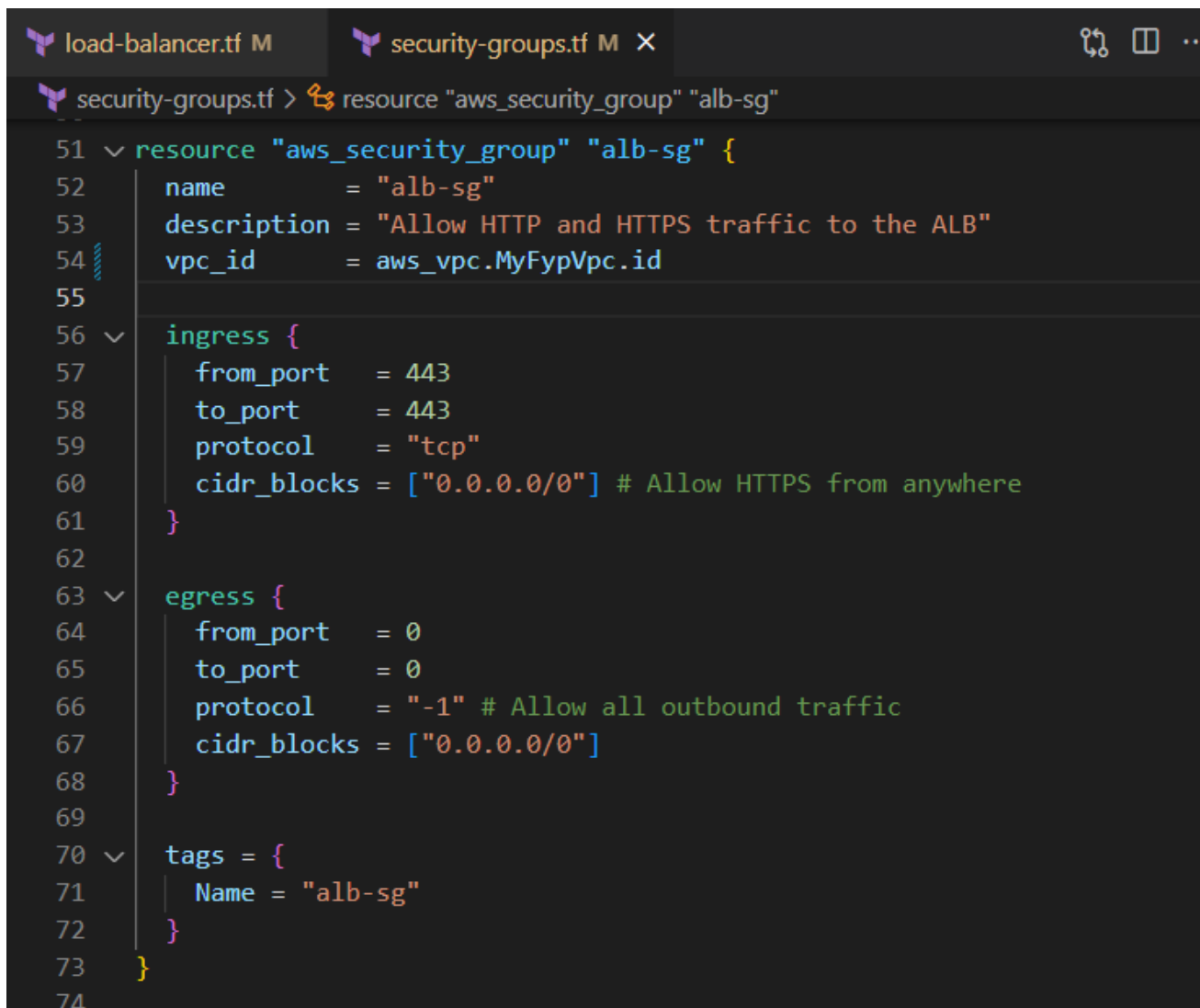
```
129  ✓ resource "aws_iam_role_policy" "AttachCodeConnectionsToCodeBuild" {
130      name = "codeconnections-policy"
131      role = aws_iam_role.CodeBuildRole.id
132      policy = file("codestar-connections-policy.json")
133  }
134
135  ✓ resource "aws_iam_role_policy_attachment" "AttachKMStoCodeBuild" {
136      role      = aws_iam_role.CodeBuildRole.id
137      policy_arn = "arn:aws:iam::aws:policy/AWSKeyManagementServicePowerUser"
138  }
139
140  ✓ resource "aws_iam_role_policy_attachment" "AttachS3toCodeBuild" {
141      role      = aws_iam_role.CodeBuildRole.id
142      policy_arn = "arn:aws:iam::aws:policy/AmazonS3FullAccess"
143  }
144
145  ✓ resource "aws_iam_role_policy_attachment" "AttachCloudWatchLogstoCodeBuild" {
146      role      = aws_iam_role.CodeBuildRole.id
147      policy_arn = "arn:aws:iam::aws:policy/CloudWatchLogsFullAccess"
148  }
```

Figure 48 - IAM Role Policies for CodeBuild

The following policies and permissions are associated with the <CodeBuildRole> to enable the required actions.

- **<resource = "aws_iam_role_policy" "AttachCodeConnectionsToCodeBuild">** attaches a custom inline policy to allow CodeBuild to manage connections.
- **<policy = file("codestar-connections-policy.json")>** links the policy file <codestar-connections-policy.json> for managing CodeStar connections.
- **<resource = "aws_iam_role_policy_attachment" "AttachKMSKeyToCodeBuild">** attaches the AWS Key Management Service PowerUser policy.
- **<policy_arn = "arn:aws:iam::aws:policy/AWSKeyManagementServicePowerUser">** provides access to manage KMS keys.
- **<resource = "aws_iam_role_policy_attachment" "AttachS3ToCodeBuild">** attaches the Amazon S3 Full Access policy.
- **<policy_arn = "arn:aws:iam::aws:policy/AmazonS3FullAccess">** enables CodeBuild to interact with S3 buckets.
- **<resource = "aws_iam_role_policy_attachment" "AttachCloudWatchLogstoCodeBuild">** attaches the CloudWatch Full Access policy.
- **<policy_arn = "arn:aws:iam::aws:policy/CloudWatchLogsFullAccess">** allows CodeBuild to write logs to CloudWatch.

12.3 Appendix C: Security Group Rules for ALB



```
load-balancer.tf M security-groups.tf M X
security-groups.tf > resource "aws_security_group" "alb-sg"

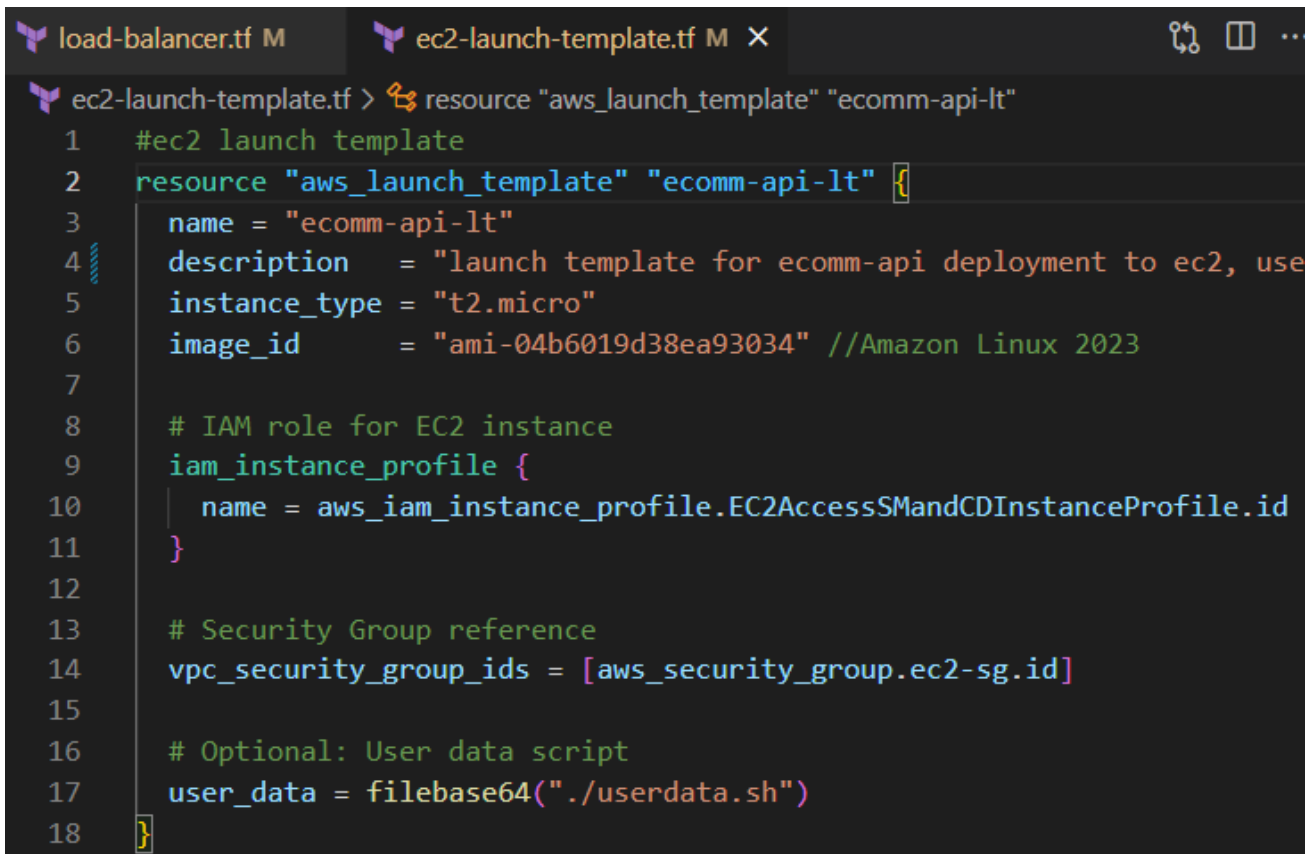
51 resource "aws_security_group" "alb-sg" {
52     name           = "alb-sg"
53     description    = "Allow HTTP and HTTPS traffic to the ALB"
54     vpc_id         = aws_vpc.MyFypVpc.id
55
56     ingress {
57         from_port   = 443
58         to_port     = 443
59         protocol    = "tcp"
60         cidr_blocks = ["0.0.0.0/0"] # Allow HTTPS from anywhere
61     }
62
63     egress {
64         from_port   = 0
65         to_port     = 0
66         protocol    = "-1" # Allow all outbound traffic
67         cidr_blocks = ["0.0.0.0/0"]
68     }
69
70     tags = {
71         Name = "alb-sg"
72     }
73 }
74
```

Figure 49 - Security Group Rules for ALB

The security group alb-sg controls traffic for the Application Load Balancer (ALB). This setup ensures the ALB only accepts secure HTTPS traffic while allowing it to send responses freely as it is hosting a public API.

- **Inbound Rule:** Allows HTTPS traffic on port 443 from any IP (0.0.0.0/0), making the ALB publicly accessible over a secure connection.
- **Outbound Rule:** Allows all outbound traffic, enabling the ALB to respond to incoming requests without restrictions.

12.4 Appendix D: EC2 Launch Template



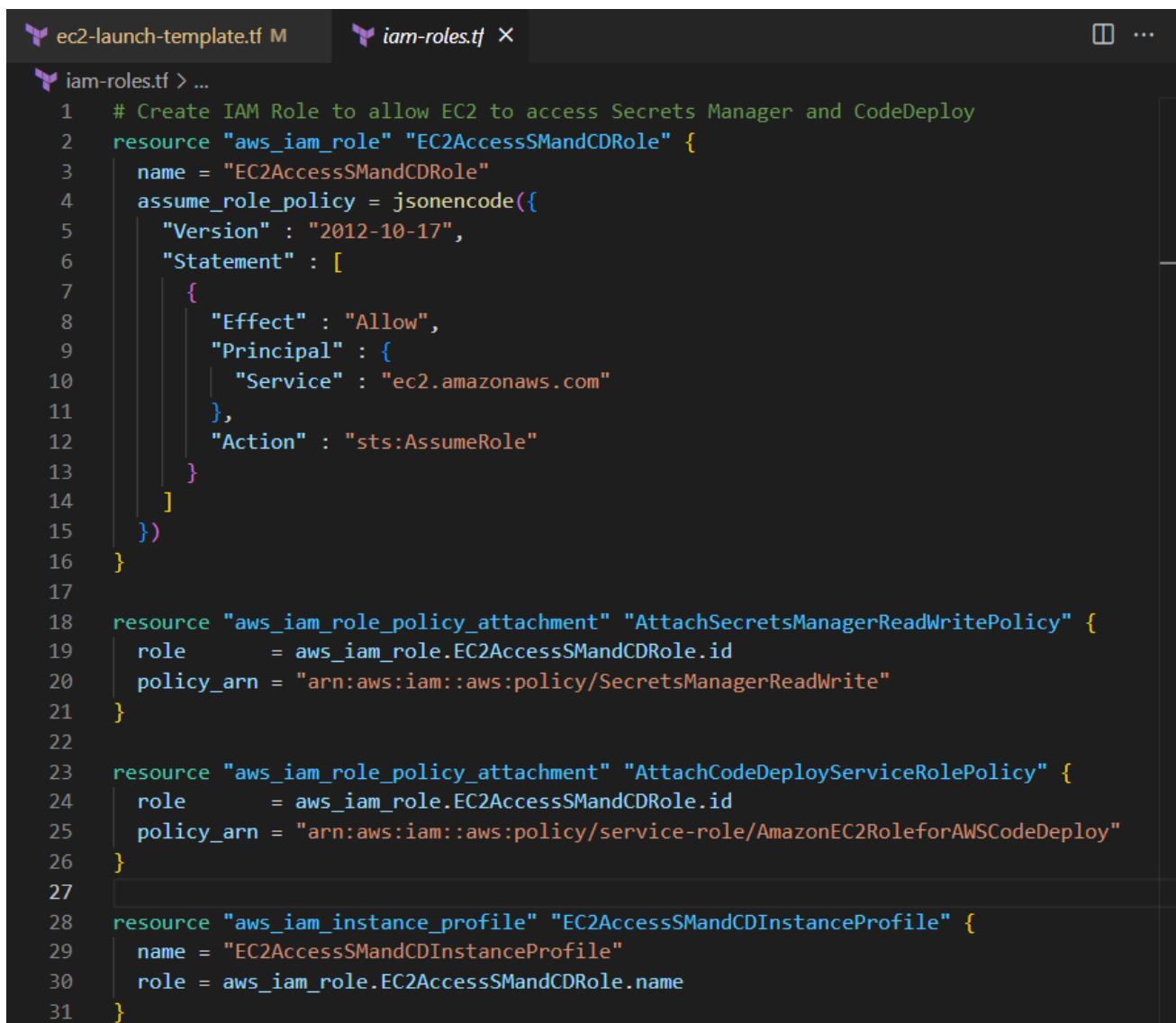
```
ec2-launch-template.tf > resource "aws_launch_template" "ecomm-api-lt"
1  #ec2 launch template
2  resource "aws_launch_template" "ecomm-api-lt" {
3      name = "ecomm-api-lt"
4      description = "launch template for ecomm-api deployment to ec2, use
5      instance_type = "t2.micro"
6      image_id = "ami-04b6019d38ea93034" //Amazon Linux 2023
7
8      # IAM role for EC2 instance
9      iam_instance_profile {
10         name = aws_iam_instance_profile.EC2AccessSMandCDInstanceProfile.id
11     }
12
13     # Security Group reference
14     vpc_security_group_ids = [aws_security_group.ec2-sg.id]
15
16     # Optional: User data script
17     user_data = filebase64("./userdata.sh")
18 }
```

Figure 50 - EC2 Launch Template Configuration

The launch template provides a reusable configuration for EC2 instances in the Auto Scaling Group. This template specifies essential parameters for deploying consistent EC2 instances.

- **<instance_type = "t2.micro">** defines the instance type, which determines the computational resources allocated (CPU, memory).
- **<image_id = "ami-04b6019d38ea93034">** specifies the Amazon Machine Image (AMI) ID. This ID corresponds to the operating system and software configuration for the instance—in this case, Amazon Linux 2023.
- **<iam_instance_profile>** associates an IAM role with the instance, providing it permissions needed to access other AWS resources securely. This role enables the instance to interact with services like AWS CodeDeploy to interact for deployments or Secrets Manager to access secrets.
- **<vpc_security_group_ids>** attaches the EC2 instances to a specific security group, controlling inbound and outbound traffic. Here, it references the security group ec2-sg, which is defined separately to manage network security.
- **<user_data = filebase64("./userdata.sh")>** includes a user data script that automatically executes at instance launch. The userdata.sh can automate setup tasks, such as installing software or configuring the application environment on startup.

12.4.1 IAM Instance Profile for EC2



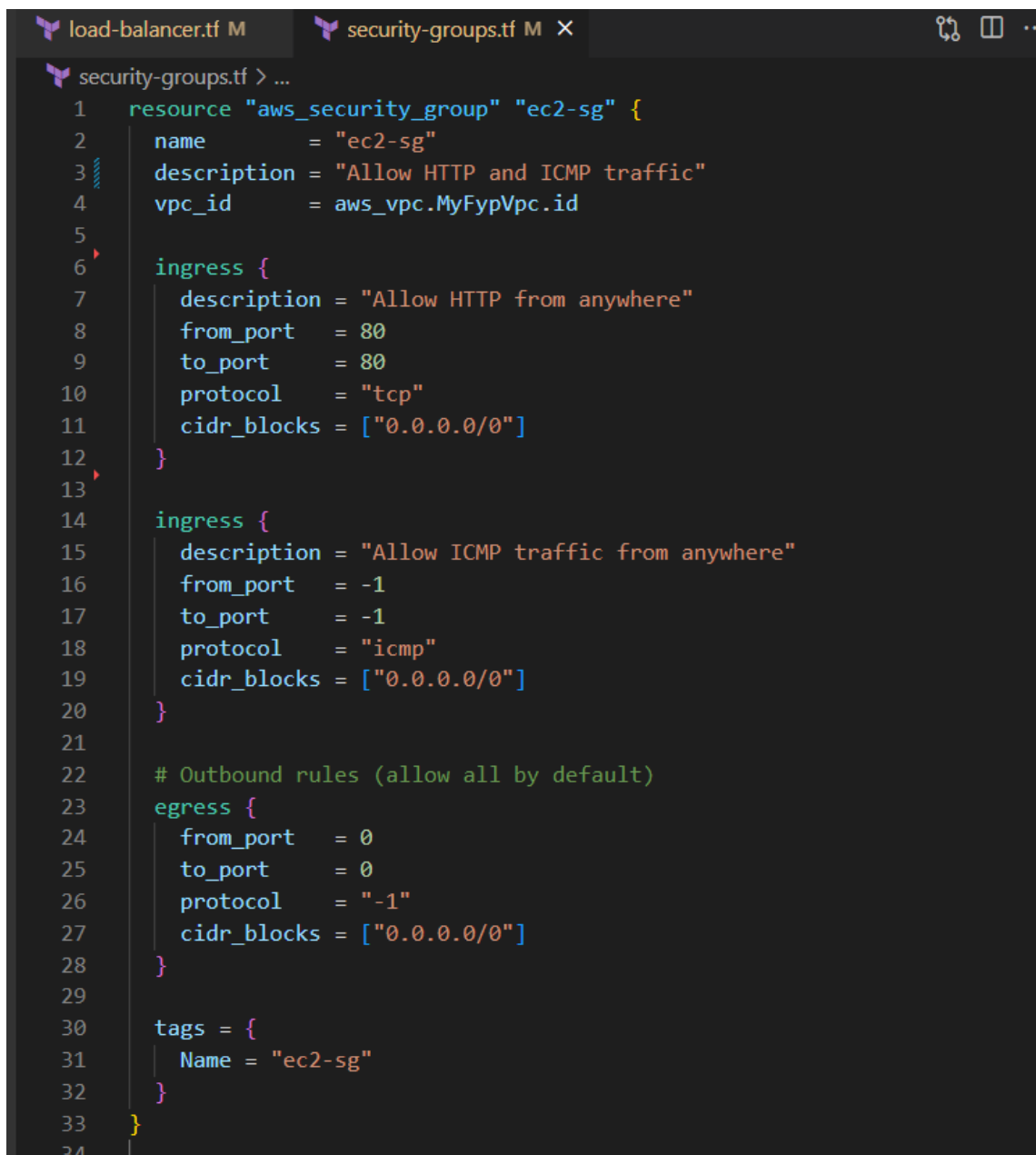
```
ec2-launch-template.tf M  iam-roles.tf X
iam-roles.tf > ...
1  # Create IAM Role to allow EC2 to access Secrets Manager and CodeDeploy
2  resource "aws_iam_role" "EC2AccessSMandCDRole" {
3      name = "EC2AccessSMandCDRole"
4      assume_role_policy = jsonencode({
5          "Version" : "2012-10-17",
6          "Statement" : [
7              {
8                  "Effect" : "Allow",
9                  "Principal" : {
10                     "Service" : "ec2.amazonaws.com"
11                 },
12                 "Action" : "sts:AssumeRole"
13             }
14         ]
15     })
16 }
17
18 resource "aws_iam_role_policy_attachment" "AttachSecretsManagerReadWritePolicy" {
19     role      = aws_iam_role.EC2AccessSMandCDRole.id
20     policy_arn = "arn:aws:iam::aws:policy/SecretsManagerReadWrite"
21 }
22
23 resource "aws_iam_role_policy_attachment" "AttachCodeDeployServiceRolePolicy" {
24     role      = aws_iam_role.EC2AccessSMandCDRole.id
25     policy_arn = "arn:aws:iam::aws:policy/service-role/AmazonEC2RoleforAWSCodeDeploy"
26 }
27
28 resource "aws_iam_instance_profile" "EC2AccessSMandCDInstanceProfile" {
29     name = "EC2AccessSMandCDInstanceProfile"
30     role = aws_iam_role.EC2AccessSMandCDRole.name
31 }
```

Figure 51 - IAM Instance Profile for EC2

This configuration creates an IAM role for EC2 instances to enable access to AWS Secrets Manager and CodeDeploy. Here's a breakdown of each part.

- **<assume_role_policy>** specifies that this role can be assumed by EC2 instances.
- **<AttachSecretsManagerReadWritePolicy>** attaches the SecretsManagerReadWrite policy, giving the EC2 instances read and write access to AWS Secrets Manager, allowing them to securely retrieve sensitive information.
- **<AttachCodeDeployServiceRolePolicy>** attaches the policy to the role, providing the EC2 instances with permissions needed to interact with AWS CodeDeploy for deployments.
- **<name = "EC2AccessSMandCDInstanceProfile">** creates an instance profile to apply the IAM role to EC2 instances.

12.4.2 EC2 Security Group Rules

A screenshot of a terminal window with a dark background. At the top, there are two tabs: 'load-balancer.tf M' and 'security-groups.tf M X'. The active tab is 'security-groups.tf M X'. Below the tabs, the terminal shows the command 'security-groups.tf > ...' followed by a Terraform configuration for an AWS security group named 'ec2-sg'. The configuration includes two ingress rules: one for HTTP (port 80) and one for ICMP (ping). It also includes an egress rule allowing all outbound traffic. The configuration is as follows:

```
1 resource "aws_security_group" "ec2-sg" {
2     name           = "ec2-sg"
3     description    = "Allow HTTP and ICMP traffic"
4     vpc_id         = aws_vpc.MyFypVpc.id
5
6     ingress {
7         description = "Allow HTTP from anywhere"
8         from_port   = 80
9         to_port     = 80
10        protocol    = "tcp"
11        cidr_blocks = ["0.0.0.0/0"]
12    }
13
14    ingress {
15        description = "Allow ICMP traffic from anywhere"
16        from_port   = -1
17        to_port     = -1
18        protocol    = "icmp"
19        cidr_blocks = ["0.0.0.0/0"]
20    }
21
22    # Outbound rules (allow all by default)
23    egress {
24        from_port   = 0
25        to_port     = 0
26        protocol    = "-1"
27        cidr_blocks = ["0.0.0.0/0"]
28    }
29
30    tags = {
31        Name = "ec2-sg"
32    }
33 }
34
```

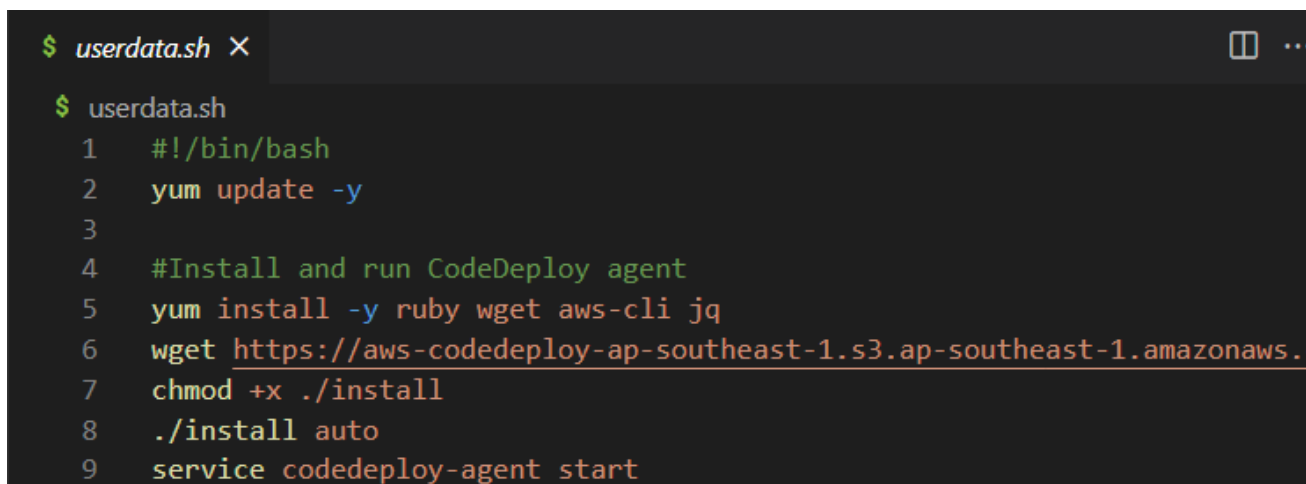
Figure 52 - EC2 Security Group Rules

The security group ec2-sg controls traffic for EC2 instances in the backend.

- HTTP (Port 80): Allows inbound HTTP traffic on port 80 from any IP (0.0.0.0/0) from the target group to reach the application.
- ICMP (Ping): Allows ICMP traffic from any IP, permitting ping requests to the instance for monitoring or testing connectivity.

Egress allows all outbound traffic (protocol = "-1") to any IP (0.0.0.0/0), enabling the instance to make unrestricted outgoing connections to support third-party API communication, which is critical for the application's payment gateway and for uploading images using Cloudinary API.

12.4.3 EC2 userdata.sh Boot Script



```
$ userdata.sh X
$ userdata.sh
1  #!/bin/bash
2  yum update -y
3
4  #Install and run CodeDeploy agent
5  yum install -y ruby wget aws-cli jq
6  wget https://aws-codedeploy-ap-southeast-1.s3.ap-southeast-1.amazonaws.com/latest/install
7  chmod +x ./install
8  ./install auto
9  service codedeploy-agent start
```

Figure 53 - EC2 userdata.sh Boot Script Configuration

The **<userdata.sh>** script executes several commands on instance startup to configure the environment for CodeDeploy which allow applications to be deployed to the instance.

- **<#!/bin/bash>** specifies the shell to execute the script.
- **<yum update -y>** updates the instance's package manager to ensure all packages are up to date.
- **<yum install -y ruby wget aws-cli jq>** installs necessary packages:
 - **<ruby>**: Required for running the CodeDeploy agent.
 - **<wget>**: Used for downloading files.
 - **<aws-cli>**: AWS Command Line Interface, enabling interaction with AWS services.
 - **<jq>**: A lightweight JSON processor, useful for handling JSON data.
- **<wget https://aws-codedeploy-ap-southeast-1.s3.ap-southeast-1.amazonaws.com/latest/install>** downloads the CodeDeploy agent installer from the specified AWS region, this will allow application to be deployed to the instance.
- **<chmod +x ./install>** grants execute permissions to the installer script, making it executable.
- **<./install auto>** runs the installer in auto mode, setting up the CodeDeploy agent automatically.
- **<service codedeploy-agent start>** starts the CodeDeploy agent service, enabling the instance to communicate with CodeDeploy for automated deployments.

12.5 Appendix E: GitHub Apps Connector

Resource: `aws_codestarconnections_connection`

Provides a CodeStar Connection.

⚠ NOTE:

The `aws_codestarconnections_connection` resource is created in the state `PENDING`. Authentication with the connection provider must be completed in the AWS Console. See the [AWS documentation](#) for details.

Figure 54 - GitHub Apps Connector Creation Restrictions

The GitHub Apps Connector is required to be created in the AWS Console User Interface as AWS does not allow the authentication to be completed within CLI. The GitHub apps connector serves to connect CodePipeline to the GitHub Repository.

Create a connection [Info](#)

Select a provider

☐ Bitbucket

☒ GitHub

☐ GitHub Enterprise Server

☐ GitLab

☐ GitLab self-managed

Create GitHub App connection [Info](#)

Connection name


github-ecomm-connection

► **Tags - optional**

Connect to GitHub

Figure 55 - AWS Console Connection Creation

The process begins by navigating to the Developer Tools section in AWS (under CodePipeline) and selecting Connections. From there, GitHub is chosen as the provider, and a unique connection name is created (e.g., github-ecomm-connection) to identify this specific integration.

① Beginning July 1, 2024, the console will create connections with codeconnections in the resource ARN. Resources with both service prefixes will continue to display in the console. [Learn more](#) 



Connect to GitHub

GitHub connection settings [Info](#)

Connection name

App installation - *optional*

Install GitHub App to connect as a bot. Alternatively, leave it blank to connect as a GitHub user, which can be used in AWS CodeBuild projects.



or

Install a new app

► **Tags - *optional***

Connect

Figure 56 - GitHub App Based Authentication Installation

AWS offers the option to install a GitHub App for the connection, allowing AWS CodePipeline to securely access the repository. This installation simplifies permissions management by following GitHub's app-based authentication, which provides only the necessary access levels required for automated deployment.


Repository access

☐ All repositories



This applies to all current *and* future repositories owned by the resource owner.
Also includes public repositories (read-only).

☒ Only select repositories

Select at least one repository.
Also includes public repositories (read-only).

 Select repositories ▼

Selected 2 repositories.

 hswg94/ecommerce-express-api	×
 hswg94/ecommerce-react-frontend	×

Save

Cancel

Figure 57 - GitHub Apps Connector Repository Permissions

Once configured, this connection allows CodePipeline to automatically detect changes in the selected GitHub repository and initiate the CI/CD pipeline. The “ecommerce-express-api” branch is to be utilized for the backend CI/CD pipeline.

12.6 Appendix F: IAM Role for CodePipeline

12.6.1 CodePipeline IAM Role Creation



```
iam-roles.tf x
iam-roles.tf > resource "aws_iam_role" "CodePipelineRole"

72 # CodePipeline Role
73 resource "aws_iam_role" "CodePipelineRole" {
74     name = "CodePipelineRole"
75     assume_role_policy = jsonencode({
76         "Version" : "2012-10-17",
77         "Statement" : [
78             {
79                 "Effect" : "Allow",
80                 "Principal" : {
81                     "Service" : "codepipeline.amazonaws.com"
82                 },
83                 "Action" : "sts:AssumeRole"
84             }
85         ]
86     })
87 }
```

Figure 58 - IAM Role Creation for CodePipeline

The IAM role for CodePipeline is created to allow the service to assume the role and perform necessary operations.

- **<resource = "aws_iam_role" "CodePipelineRole">** defines the IAM role for CodePipeline.
- **<assume_role_policy>** specifies the trust relationship allowing the CodePipeline service to assume the role.

12.6.2 CodePipeline IAM Role Policies

```
89  resource "aws_iam_role_policy" "AttachCodeConnectionsToCodePipeline" {
90      name = "codeconnections-policy"
91      role = aws_iam_role.CodePipelineRole.id
92      policy = file("codestar-connections-policy.json")
93  }
94
95  resource "aws_iam_role_policy_attachment" "AttachCodeDeploytoCodePipeline" {
96      role      = aws_iam_role.CodePipelineRole.id
97      policy_arn = "arn:aws:iam::aws:policy/AWSCodeDeployFullAccess"
98  }
99
100 resource "aws_iam_role_policy_attachment" "AttachKMStoCodePipeline" {
101     role      = aws_iam_role.CodePipelineRole.id
102     policy_arn = "arn:aws:iam::aws:policy/AWSKeyManagementServicePowerUser"
103 }
104
105 resource "aws_iam_role_policy_attachment" "AttachS3toCodePipeline" {
106     role      = aws_iam_role.CodePipelineRole.id
107     policy_arn = "arn:aws:iam::aws:policy/AmazonS3FullAccess"
108 }
```

Figure 59 - IAM Role Policies for CodePipeline

The following policies and permissions are associated with the <CodePipelineRole> to enable its operations:

- **<resource = "aws_iam_role_policy" "AttachCodeConnectionsToCodePipeline">** attaches a custom inline policy to allow CodePipeline to manage connections.
- **<policy = file("codestar-connections-policy.json")>** links the policy file <codestar-connections-policy.json> for managing CodeStar connections.
- **<resource = "aws_iam_role_policy_attachment" "AttachCodeDeployToCodePipeline">** attaches the AWS CodeDeploy Full Access policy.
- **<policy_arn = "arn:aws:iam::aws:policy/AWSCodeDeployFullAccess">** grants CodePipeline permissions for deployments using CodeDeploy.
- **<resource = "aws_iam_role_policy_attachment" "AttachKMStoCodePipeline">** attaches the AWS Key Management Service PowerUser policy.
- **<policy_arn = "arn:aws:iam::aws:policy/AWSKeyManagementServicePowerUser">** provides access to manage encryption keys.
- **<resource = "aws_iam_role_policy_attachment" "AttachS3ToCodePipeline">** attaches the Amazon S3 Full Access policy.
- **<policy_arn = "arn:aws:iam::aws:policy/AmazonS3FullAccess">** allows CodePipeline to interact with S3 buckets.

12.7 Appendix G: IAM Role for CodeDeploy

12.7.1 IAM Role Creation for CodeDeploy

```
35 // # CodeDeploy Role to access ASG and EC2
36 resource "aws_iam_role" "CodeDeployRole" {
37     name = "CodeDeployRole"
38     assume_role_policy = jsonencode({
39         "Version" : "2012-10-17",
40         "Statement" : [
41             {
42                 "Sid" : "",
43                 "Effect" : "Allow",
44                 "Principal" : {
45                     "Service" : "codedeploy.amazonaws.com"
46                 },
47                 "Action" : "sts:AssumeRole"
48             }
49         ]
50     })
51 }
```

Figure 60 - IAM Role Creation for CodeDeploy

The IAM role configuration provides AWS CodeDeploy with the permissions needed to manage Auto Scaling Groups (ASG), EC2 instances, and other resources during the deployment process.

- **<resource = "aws_iam_role" "CodeDeployRole">** defines the IAM role <CodeDeployRole> for CodeDeploy.
- **<assume_role_policy>** specifies the trust relationship allowing CodeDeploy to assume the role.

12.7.2 IAM Role Policies For CodeDeploy Role

```
53  resource "aws_iam_role_policy_attachment" "AttachAmazonEC2FullAccess" {
54      role      = aws_iam_role.CodeDeployRole.id
55      policy_arn = "arn:aws:iam::aws:policy/AmazonEC2FullAccess"
56  }
57
58  resource "aws_iam_role_policy_attachment" "AttachAutoScalingFullAccess" {
59      role      = aws_iam_role.CodeDeployRole.id
60      policy_arn = "arn:aws:iam::aws:policy/AutoScalingFullAccess"
61  }
62
63  resource "aws_iam_role_policy_attachment" "AttachIAMFullAccess" {
64      role      = aws_iam_role.CodeDeployRole.id
65      policy_arn = "arn:aws:iam::aws:policy/IAMFullAccess"
66  }
67
```

Figure 61 - IAM Role Policies for CodeDeploy

The following policies are attached to provide CodeDeploy with the required permissions:

- **<resource = "aws_iam_role_policy_attachment" "AttachAmazonEC2FullAccess">** attaches the Amazon EC2 Full Access policy.
- **<policy_arn = "arn:aws:iam::aws:policy/AmazonEC2FullAccess">** grants permissions to manage EC2 instances.
- **<resource = "aws_iam_role_policy_attachment" "AttachAutoScalingFullAccess">** attaches the Auto Scaling Full Access policy.
- **<policy_arn = "arn:aws:iam::aws:policy/AutoScalingFullAccess">** enables access to manage Auto Scaling Groups.
- **<resource = "aws_iam_role_policy_attachment" "AttachIAMFullAccess">** attaches the IAM Full Access policy.
- **<policy_arn = "arn:aws:iam::aws:policy/IAMFullAccess">** provides full access to manage IAM resources.

12.8 Appendix H: Route 53 Hosted Zone

A screenshot of a code editor window titled 'route53.tf M X'. The editor shows Terraform configuration code for creating an AWS Route 53 hosted zone and updating its name servers. The code is as follows:

```
1 //Create a hosted zone
2 resource "aws_route53_zone" "primary" {
3   name = "hswg94.com"
4 }
5
6 # Update name servers on the registered domain to match hosted zone
7 resource "aws_route53domains_registered_domain" "update-domain-ns" {
8   domain_name = "hswg94.com"
9   auto_renew = "false"
10  transfer_lock = "false"
11
12  dynamic "name_server" {
13    for_each = toset(aws_route53_zone.primary.name_servers)
14    content {
15      name = name_server.value
16    }
17  }
18 }
19
```

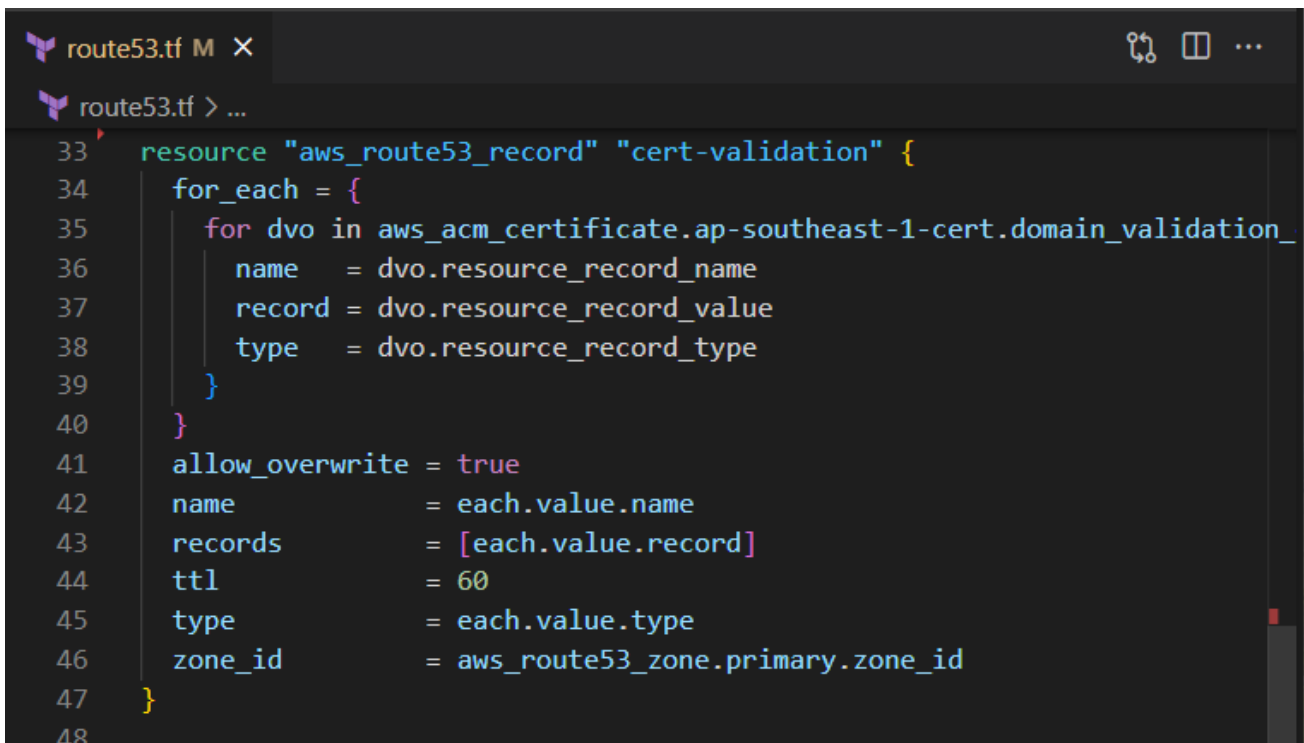
Figure 62 - Route 53 Hosted Zone Configuration

This configuration creates a hosted zone in AWS Route 53 and updates the name servers on the registered domain to match the hosted zone's settings. This configuration is necessary to delegate DNS management to AWS, allowing to create and manage DNS records for hswg94.com within Route 53.

- **<name = "hswg94.com">** specifies the domain name for the hosted zone.
- **<domain_name = "hswg94.com">** specifies the domain name to be updated.
- **<auto_renew = "false">** and **<transfer_lock = "false">** set the renewal and transfer lock settings for the domain.
- The dynamic **<name_server>** block iterates over the name servers in the hosted zone and updates them in the domain's configuration. This ensures that the domain's name servers are aligned with the hosted zone, allowing Route 53 to manage DNS records for hswg94.com.

12.9 Appendix I: Validating Certificates with Route 53

12.9.1 Adding Certificate DNS Validation Records to Route 53



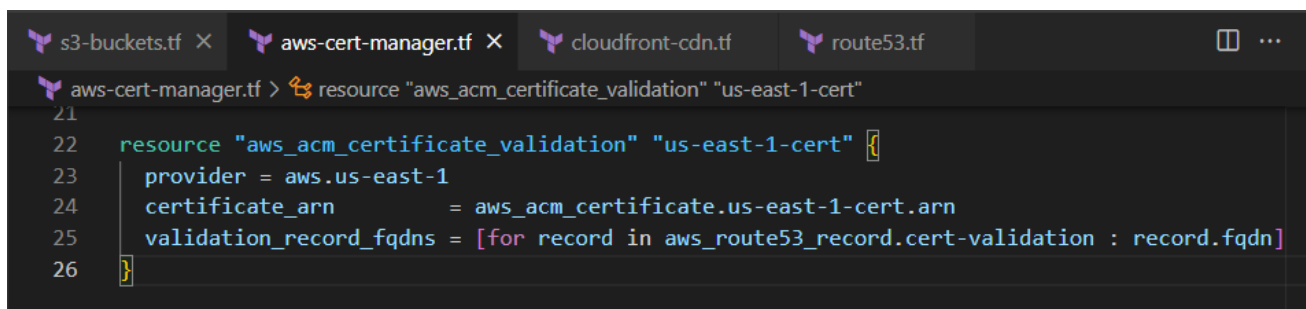
```
33 resource "aws_route53_record" "cert-validation" {
34   for_each = {
35     for dvo in aws_acm_certificate.ap-southeast-1-cert.domain_validation_
36       name    = dvo.resource_record_name
37       record  = dvo.resource_record_value
38       type    = dvo.resource_record_type
39     }
40   }
41   allow_overwrite = true
42   name            = each.value.name
43   records         = [each.value.record]
44   ttl            = 60
45   type           = each.value.type
46   zone_id        = aws_route53_zone.primary.zone_id
47 }
48
```

Figure 63 - Adding Certificate DNS Validation Records to Route 53

When an ACM (AWS Certificate Manager) certificate is requested, it generates DNS validation records. These records must be added to the domain's DNS provider (e.g., Route 53) to prove ownership of the domain. The Terraform configuration automates this process by creating and deploying the necessary DNS validation records.

- **<for_each>** forms an object by iterating over the domain records provided by ACM to create the necessary validation records.
- **<name>** sets the record name required for validation.
- **<record>** is the validation value for AWS Certificate Manager.
- **<type>** is the record type (e.g., CNAME) as required by ACM.
- **<allow_overwrite = true>** allows the record to be overwritten if changes are required.
- **<ttl = 60>** sets a short time-to-live (TTL), allowing quick propagation of validation changes.

12.9.2 Verifying the Certificates Validity



```
s3-buckets.tf × aws-cert-manager.tf × cloudfront-cdn.tf route53.tf
aws-cert-manager.tf > resource "aws_acm_certificate_validation" "us-east-1-cert"
21
22 resource "aws_acm_certificate_validation" "us-east-1-cert" {
23     provider = aws.us-east-1
24     certificate_arn = aws_acm_certificate.us-east-1-cert.arn
25     validation_record_fqdns = [for record in aws_route53_record.cert-validation : record.fqdn]
26 }
```

Figure 64 - Verifying Certificates Validity in AWS ACM

After creating the records, the TLS certificate must be first validated before it can be established and associated with other services. The steps to validating the certificate can be found in Appendix I: Validating Certificates with Route 53.

After the validation steps are performed, the **<aws_acm_certificate_validation>** resource shown in Figure 64 is established to ensure the certificate is correctly validated and marks it as ready to use.

- **<provider>** is set to us-east-1 as CloudFront requires the certificate to be in the specific region.
- **<certificate_arn>** references the ARN of the ACM certificate (aws_acm_certificate.us-east-1-cert etc.) being validated.