Configuration Manual

MSc Research Project
Cloud Computing

Dhanusha Siva Priya Chintakayala
Student ID: 23192887

School of Computing
National College of Ireland

Supervisor: Dr Giovani Estrada

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Dhanusha Siva Priya Chintakayala
Student ID:	23192887
Programme:	Cloud Computing **Year:** 2025
Module:	MSc Research Project
Lecturer:	Dr Giovani Estrada
Submission Due Date:	24 April 2025
Project Title:	Designing and Scaling OPA for PCI-DSS and HIPAA Compliance in AWS
Page Count:	11
Word Count:	1917

Signature: Dhanusha

Date: 26 May 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□

Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

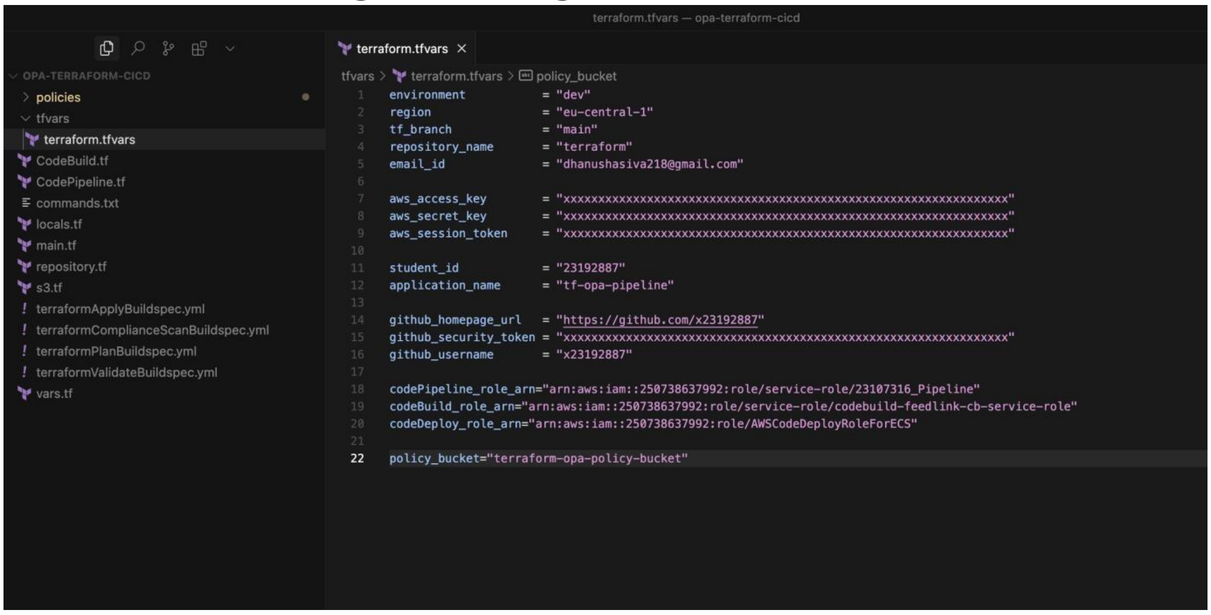
Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Dhanusha Siva Priya Chintakayala
Student ID: 23192887

1. Understanding the configuration file terraform.tfvars


terraform.tfvars — opa-terraform-cicd
OPA-TERRAFORM-CICD
policies
tfvars
terraform.tfvars
CodeBuild.tf
CodePipeline.tf
commands.txt
locals.tf
main.tf
repository.tf
s3.tf
terraformApplyBuildspec.yml
terraformComplianceScanBuildspec.yml
terraformPlanBuildspec.yml
terraformValidateBuildspec.yml
vars.tf
terraform.tfvars
tfvars > terraform.tfvars > policy_bucket

```
environment          = "dev"
region               = "eu-central-1"
tf_branch            = "main"
repository_name      = "terraform"
email_id             = "dhanushasiva218@gmail.com"

aws_access_key       = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
aws_secret_key       = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
aws_session_token    = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"

student_id           = "23192887"
application_name     = "tf-opa-pipeline"

github_homepage_url   = "https://github.com/x23192887"
github_security_token = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
github_username       = "x23192887"

codePipeline_role_arn="arn:aws:iam::250738637992:role/service-role/23107316_Pipeline"
codeBuild_role_arn="arn:aws:iam::250738637992:role/service-role/codebuild-feedlink-cb-service-role"
codeDeploy_role_arn="arn:aws:iam::250738637992:role/AWSCodeDeployRoleForECS"

policy_bucket="terraform-opa-policy-bucket"
```

Terraform variable definition file that supplies values for input variables in the Terraform configuration. Hardcoded values are not put directly into a .tf files but rather into variables.tf (or equivalent) and then set variable values in terraform.tfvars (or pass -var flags) (Kavas, 2023). It's a separation of concern that increases the modularity of our code, increases security, and it allows greater flexibility with regards to the environment. This is an explanation of the keys defined in the terraform.tfvars file (Birade, 2025).

environment: Determines the environment where this is going to be deployed such as dev, staging, prod. This enables Terraform to tell apart infrastructure resources by means of lifecycle stages.

region: Defines the AWS region to provision resources in (eg eu-central-1). This is key to ensuring geographic and compliance requirements. **tf_branch**: Refers to the Git branch that should pull Terraform source code from (e.g., main). That makes it important for automating CI/CD from version-controlled infrastructure.

repository_name: Defines the name of the GitHub repository containing the Terraform code.

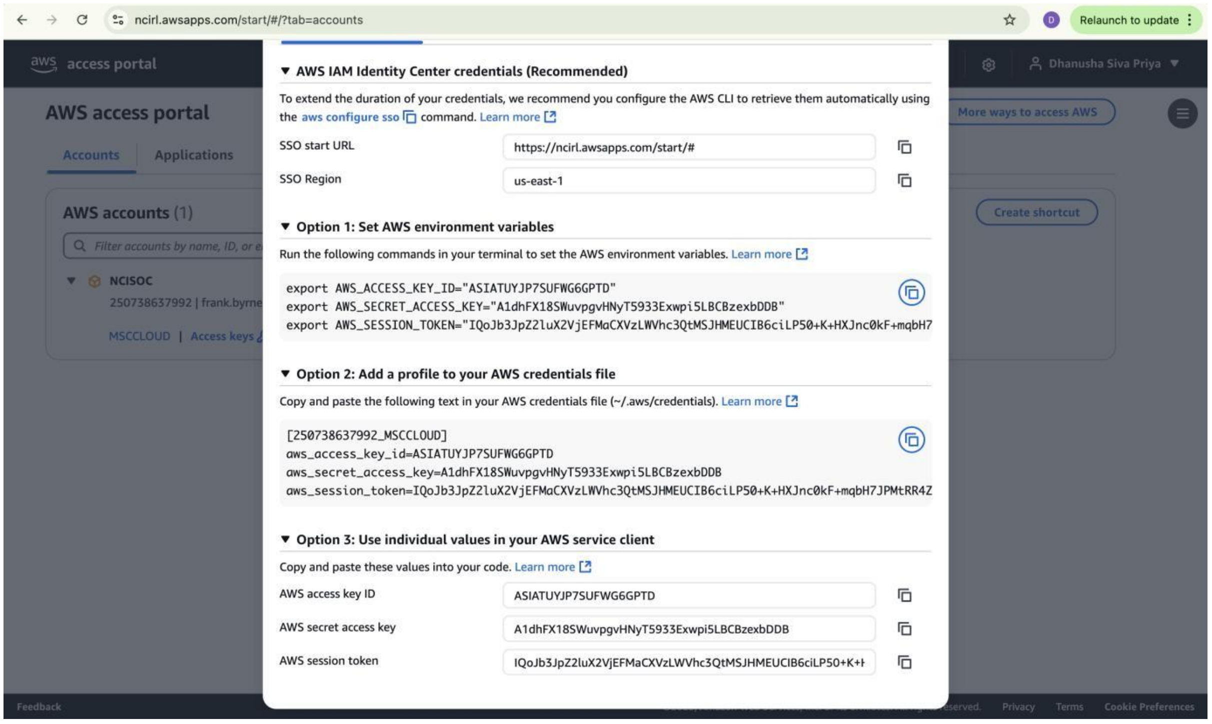
email_id: This is the address to which alerts or feedback are sent, especially in case OPA (Open Policy Agent) compliance checks fail. For example, the user would receive the following email from the notification system if a policy violation is detected during a CI/CD pipeline run.

aws_access_key, aws_secret_key, aws_session_token: These are used for Terraform to be authenticated and authorized to perform operations in AWS. These are sometimes provided via federated login or IAM roles via STS tokens.

student_id: This is an academic tracking unique identifier (can be used to namespace resources during multi-student or multi-project deployments). **application_name**: Logical

name of the application/pipeline (e.g., tf-opa-pipeline). Used for tagging resources and for bringing traceability to them.
github_homepage_url: The URL to your GitHub profile or repository. Can be applied for Terraform documentation, CI triggers or integration dashboards. **github_security_token**: A personal access token (PAT) is used to authenticate to the GitHub APIs. This is required for any code push/pull operation; triggering GitHub Actions; or integration with GitHub webhooks. **github_username**: The token and the repository belong to a GitHub handle. This is useful for labeling CI/CD runs triggering builds.
codePipeline_role_arn: Assumed IAM Role by AWS CodePipeline. To support the process, it requires access to source (GitHub), build (CodeBuild) and deploy (CodeDeploy/ECS). **codeBuild_role_arn**: AWS CodeBuild assumes IAM Role. Needs compilation, building Docker images and pushing artifacts such as those in ECR or S3 permissions.
codeDeploy_role_arn: This is the IAM Role used by CodeDeploy for deployment of artifacts to ECS or EC2 targets. **policy_bucket**: The name of S3 bucket storing the OPA Rego policies. During the pipeline, these policies are fetched to be enforced in compliance (i.e. prevent public S3 buckets, ensuring the encryption, etc.).

2. AWS Account Credentials Setup



AWS IAM Identity Center credentials (Recommended)

To extend the duration of your credentials, we recommend you configure the AWS CLI to retrieve them automatically using the aws configure sso command. Learn more

SSO start URL: https://ncirl.awsapps.com/start/#

SSO Region: us-east-1

Option 1: Set AWS environment variables

Run the following commands in your terminal to set the AWS environment variables. Learn more

```
export AWS_ACCESS_KEY_ID="ASIATUYJP7SUFWG6GPTD"
export AWS_SECRET_ACCESS_KEY="A1dhFX18SWuvpgvHNyT5933Exwpi5LBCBzexbDDB"
export AWS_SESSION_TOKEN="IQoJb3JpZ2luX2VjEFMaCXVzLWVhc3QtMSJHMEUCIB6ciLP50+K+HXJnc0kF+mqbH7
```

Option 2: Add a profile to your AWS credentials file

Copy and paste the following text in your AWS credentials file (~/.aws/credentials). Learn more

```
[250738637992_MSCCLOUD]
aws_access_key_id=ASIATUYJP7SUFWG6GPTD
aws_secret_access_key=A1dhFX18SWuvpgvHNyT5933Exwpi5LBCBzexbDDB
aws_session_token=IQoJb3JpZ2luX2VjEFMaCXVzLWVhc3QtMSJHMEUCIB6ciLP50+K+HXJnc0kF+mqbH7JPMtRR4Z
```

Option 3: Use individual values in your AWS service client

Copy and paste these values into your code. Learn more

AWS access key ID	ASIATUYJP7SUFWG6GPTD
AWS secret access key	A1dhFX18SWuvpgvHNyT5933Exwpi5LBCBzexbDDB
AWS session token	IQoJb3JpZ2luX2VjEFMaCXVzLWVhc3QtMSJHMEUCIB6ciLP50+K+I

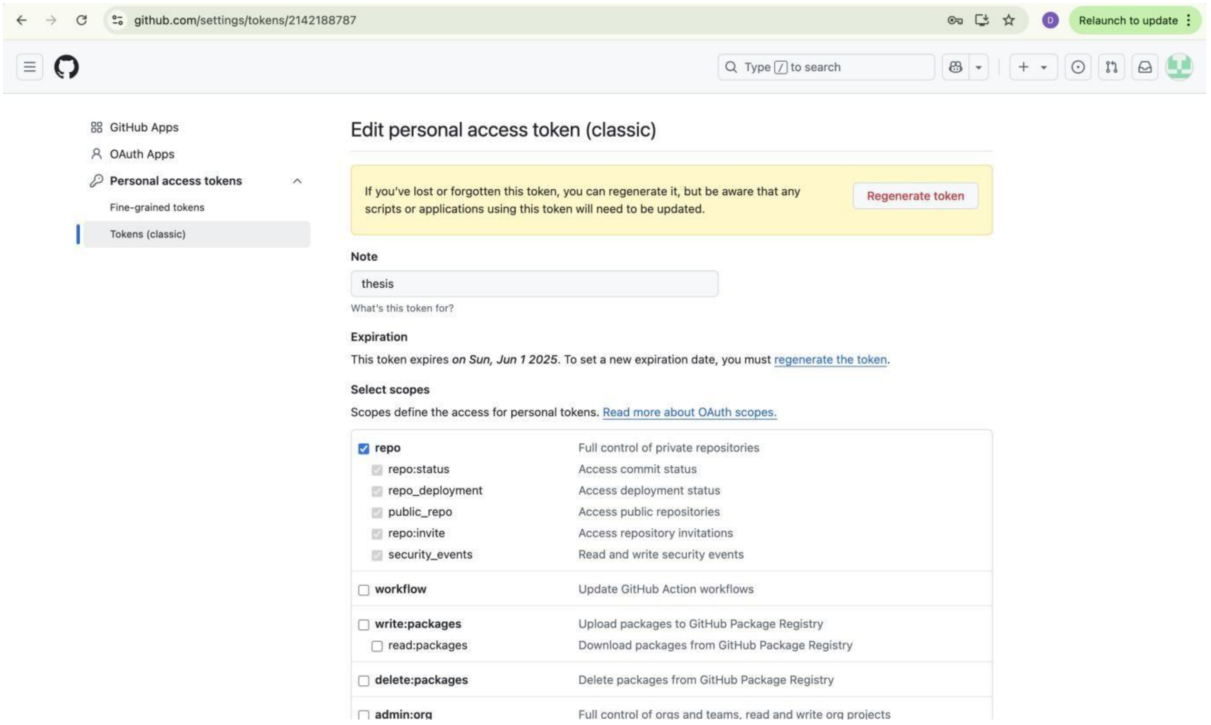
Valid AWS credentials are needed to provision infrastructure and deploy applications with Terraform and AWS Code services (Salecha, 2023).

Steps:
1) Log in to AWS Console:
Navigate to https://console.aws.amazon.com
2) Create an IAM User:

- Go to IAM > Users > Add User
- Choose Programmatic access

- Attach policies like AdministratorAccess (or custom least-privilege policies) 3) Create Access Keys:
- Under IAM tab on the application menu, go to Users > [Your User] > Security credentials
- Click Create access key
- Select the use case to be the Command Line Interface (CLI).
- Access the credentials – Access key ID, Secret access key and Session token.
- Make a terraform.tfvars file mentioned in step 1 and add above three credentials in this file.

3. Github Personal Access Token Setup



Edit personal access token (classic)

If you've lost or forgotten this token, you can regenerate it, but be aware that any scripts or applications using this token will need to be updated.

Note

thesis

What's this token for?

Expiration

This token expires *on Sun, Jun 1 2025*. To set a new expiration date, you must regenerate the token.

Select scopes

Scopes define the access for personal tokens. Read more about OAuth scopes.

✓ repo	Full control of private repositories
✓ repo:status	Access commit status
✓ repo_deployment	Access deployment status
✓ public_repo	Access public repositories
✓ repo:invite	Access repository invitations
✓ security_events	Read and write security events
☐ workflow	Update GitHub Action workflows
☐ write:packages	Upload packages to GitHub Package Registry
☐ read:packages	Download packages from GitHub Package Registry
☐ delete:packages	Delete packages from GitHub Package Registry
☐ admin:org	Full control of orgs and teams, read and write org projects

To authenticate Terraform and AWS CodePipeline with GitHub repositories a GitHub PAT (Personal Access Token) is necessary (GitHub, 2025).
Steps:
1) Go to GitHub:
Access the given link: https://github.com/settings/tokens.

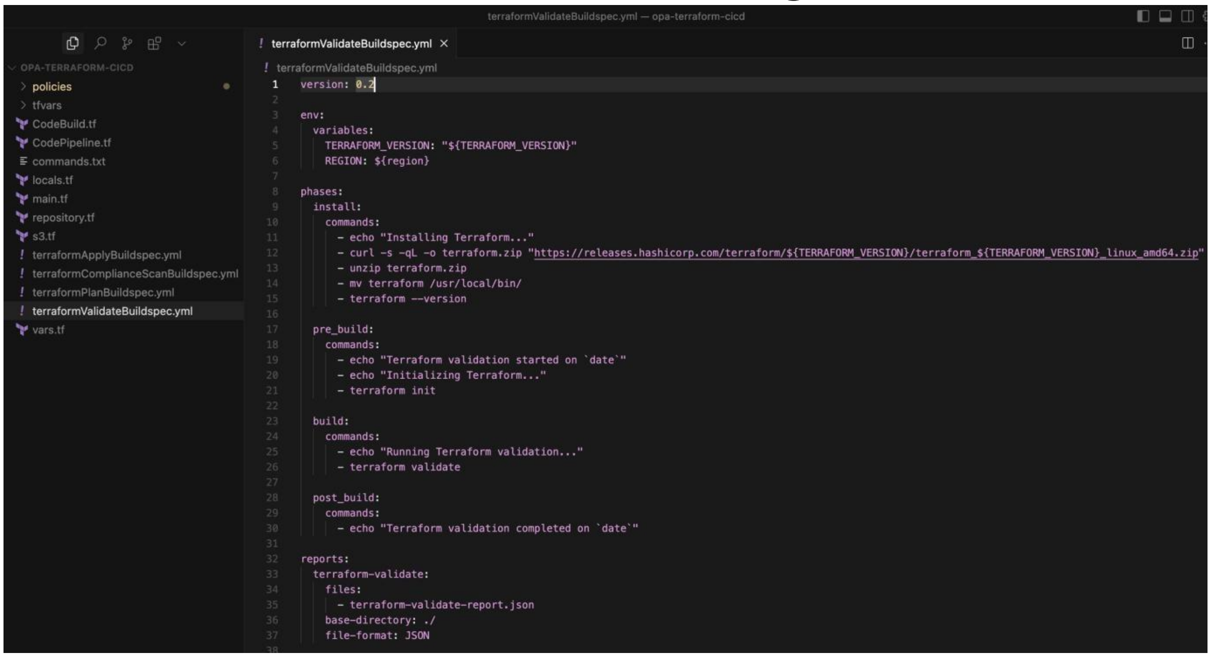
2) Generate New Token:
- Click Generate new token (Classic) or Fine-grained token if needed, depending on the settings.
- Scope selection:
- repo (Full control of private repositories)
- workflow (For GitHub Actions, if used)
- To manage webhooks for CI triggers: admin:repo_hook
- read:org (If part of an organization)

3) Set Expiration and Note:

Choose a suitable expiration (e.g., 30 days) and add a description like Terraform Thesis CI/CD.

4) Copy the Token:
   Important: This value is view once, hence copy and store it securely.
5) Use in terraform.tfvars: github_security_token = "your-generated-token"

4. Terraform Install In Code Build Stage



terraformValidateBuildspec.yml — opa-terraform-cicd
OPA-TERRAFORM-CICD
policies
tfvars
CodeBuild.tf
CodePipeline.tf
commands.txt
locals.tf
main.tf
repository.tf
s3.tf
terraformApplyBuildspec.yml
terraformComplianceScanBuildspec.yml
terraformPlanBuildspec.yml
terraformValidateBuildspec.yml
vars.tf

```yaml
version: 0.2

env:
  variables:
    TERRAFORM_VERSION: "${TERRAFORM_VERSION}"
    REGION: ${region}

phases:
  install:
    commands:
      - echo "Installing Terraform..."
      - curl -s -qL -o terraform.zip "https://releases.hashicorp.com/terraform/${TERRAFORM_VERSION}/terraform_${TERRAFORM_VERSION}_linux_amd64.zip"
      - unzip terraform.zip
      - mv terraform /usr/local/bin/
      - terraform --version

  pre_build:
    commands:
      - echo "Terraform validation started on `date`"
      - echo "Initializing Terraform..."
      - terraform init

  build:
    commands:
      - echo "Running Terraform validation..."
      - terraform validate

  post_build:
    commands:
      - echo "Terraform validation completed on `date`"

reports:
  terraform-validate:
    files:
      - terraform-validate-report.json
    base-directory: ./
    file-format: JSON
```

IaC Tool used to provision Amazon Web Service resources declaratively is Terraform(Arba, 2023).

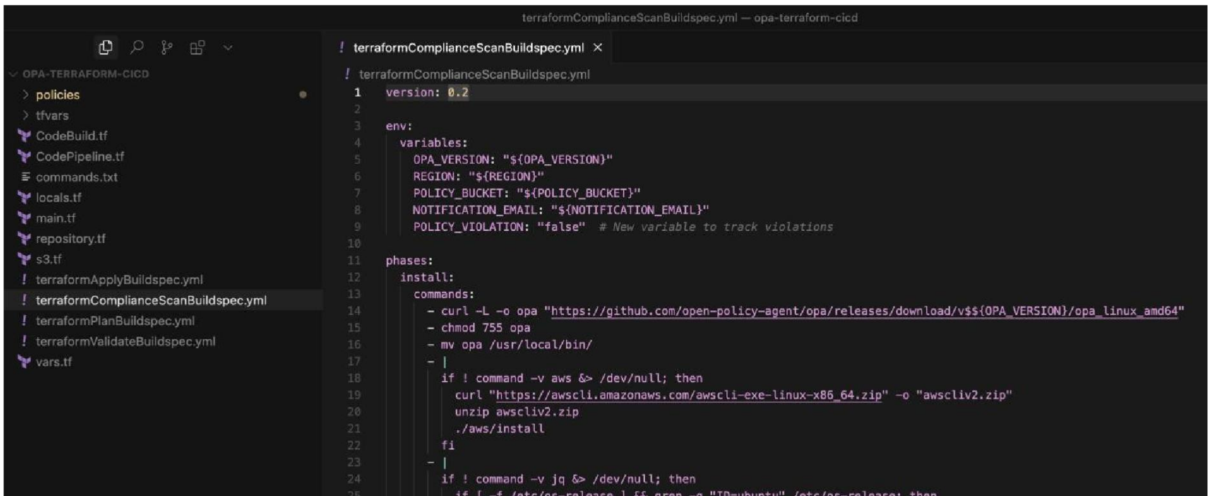
In the screenshots above, the commands install Terraform v1.5.7.

Explanation:

- curl downloads the specified Terraform zip archive from HashiCorp releases.
- unzip: Extracts the binary from the archive.
- mv terraform /usr/local/bin/: Moves the binary to a directory in the system's PATH for global CLI access.
- terraform --version: Verifies the installation.

5. Open Policy Agent Installation In Code Build Stage

Fine grained policies are written in the form of Rego and used with OPA to enforce these



terraformComplianceScanBuildspec.yml — opa-terraform-cicd
OPA-TERRAFORM-CICD
policies
tfvars
CodeBuild.tf
CodePipeline.tf
commands.txt
locals.tf
main.tf
repository.tf
s3.tf
terraformApplyBuildspec.yml
terraformComplianceScanBuildspec.yml
terraformPlanBuildspec.yml
terraformValidateBuildspec.yml
vars.tf

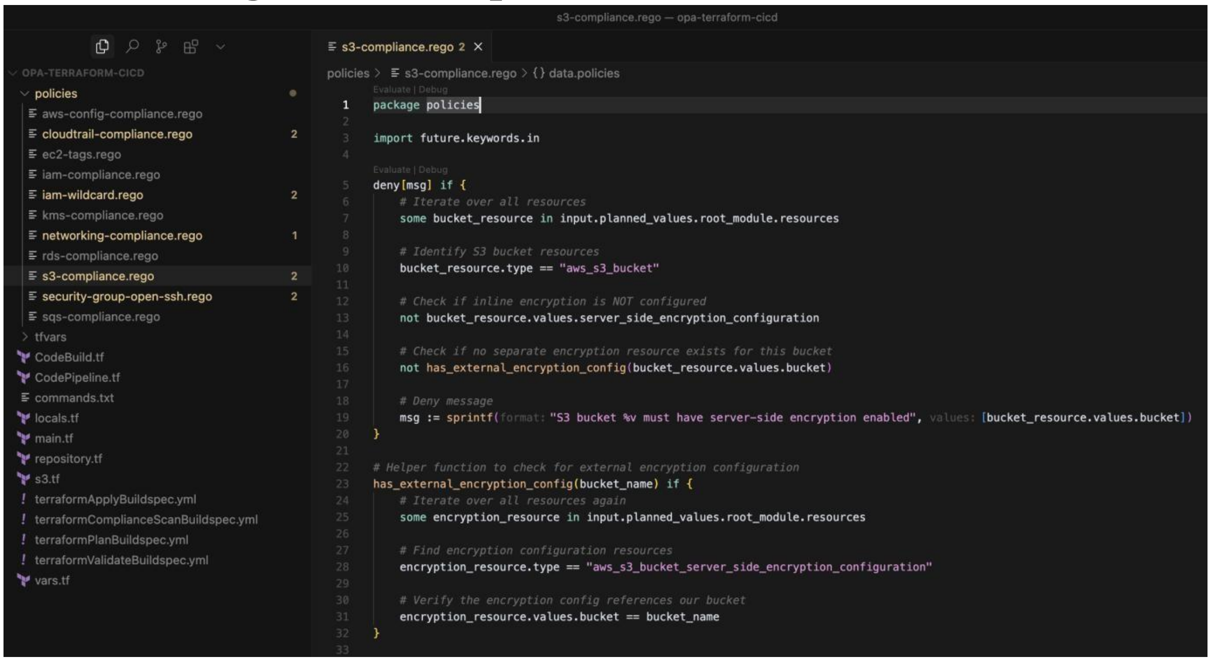
```yaml
version: 0.2

env:
  variables:
    OPA_VERSION: "${OPA_VERSION}"
    REGION: "${REGION}"
    POLICY_BUCKET: "${POLICY_BUCKET}"
    NOTIFICATION_EMAIL: "${NOTIFICATION_EMAIL}"
    POLICY_VIOLATION: "false"  # New variable to track violations

phases:
  install:
    commands:
      - curl -L -o opa "https://github.com/open-policy-agent/opa/releases/download/v$${OPA_VERSION}/opa_linux_amd64"
      - chmod 755 opa
      - mv opa /usr/local/bin/
      - |
        if ! command -v aws &> /dev/null; then
          curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
          unzip awscliv2.zip
          ./aws/install
        fi
      - |
        if ! command -v jq &> /dev/null; then
          if [ -f /etc/os-release ] && grep -q "ID=ubuntu" /etc/os-release; then
```

policies during the CI/CD pipeline execution (Paul, Manoj and S, 2024). The following commands install OPA v1.2.0 (OPA, 2025a):

Explanation:

- curl: Downloads the OPA binary from GitHub Releases.
- chmod 755: Makes the OPA binary executable.
- mv opa /usr/local/bin/: Makes the OPA CLI globally available by adding it in the system PATH.

6. OPA Rego Policies Upload To S3 Bucket


s3-compliance.rego — opa-terraform-cicd
OPA-TERRAFORM-CICD
policies
aws-config-compliance.rego
cloudtrail-compliance.rego 2
ec2-tags.rego
iam-compliance.rego
iam-wildcard.rego 2
kms-compliance.rego
networking-compliance.rego 1
rds-compliance.rego
s3-compliance.rego 2
security-group-open-ssh.rego 2
sqs-compliance.rego
tfvars
CodeBuild.tf
CodePipeline.tf
commands.txt
locals.tf
main.tf
repository.tf
s3.tf
terraformApplyBuildspec.yml
terraformComplianceScanBuildspec.yml
terraformPlanBuildspec.yml
terraformValidateBuildspec.yml
vars.tf
s3-compliance.rego 2
policies > s3-compliance.rego > {} data.policies
Evaluate | Debug

```
package policies

import future.keywords.in

deny[msg] if {
    # Iterate over all resources
    some bucket_resource in input.planned_values.root_module.resources

    # Identify S3 bucket resources
    bucket_resource.type == "aws_s3_bucket"

    # Check if inline encryption is NOT configured
    not bucket_resource.values.server_side_encryption_configuration

    # Check if no separate encryption resource exists for this bucket
    not has_external_encryption_config(bucket_resource.values.bucket)

    # Deny message
    msg := sprintf(format: "S3 bucket %v must have server-side encryption enabled", values: [bucket_resource.values.bucket])
}

# Helper function to check for external encryption configuration
has_external_encryption_config(bucket_name) if {
    # Iterate over all resources again
    some encryption_resource in input.planned_values.root_module.resources

    # Find encryption configuration resources
    encryption_resource.type == "aws_s3_bucket_server_side_encryption_configuration"

    # Verify the encryption config references our bucket
    encryption_resource.values.bucket == bucket_name
}
```

Open Policy Agent (OPA) uses Rego as the policy language to define rules and constraints that will dictate the system behavior. In the context of IaC (Infrastructure as a Code), Rego policies can be used to evaluate Terraform plans, to make sure that they are compliant with company's security, operational, and infrastructural standards before the infrastructure is deployed (Infralovers, 2024).

For instance, Rego can be used to write policies (OPA, 2025b)

- preventing the creation of public S3 buckets,
- ensure that all EBS volumes are encrypted
- prevent using expensive EC2 instance types
- require the usage of mandatory tags on all resources

All Rego policies are arranged under directory called **policies/** in this thesis. Each file inside this folder is a .rego which contains a set of rules of compliance.

The setup phase shall upload these policies to an S3 bucket (e.g. terraform-opa-policybucket). The policy repository is implemented using a bucket, which centralizes the policies.

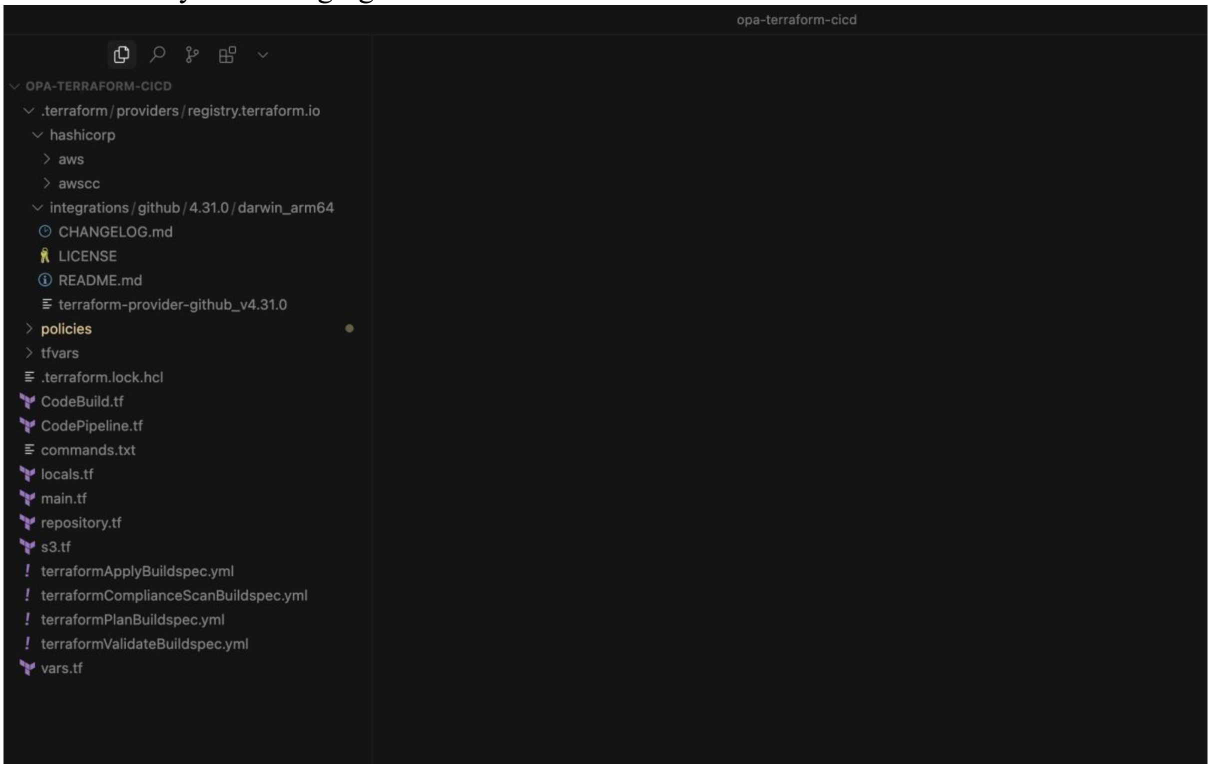
During the CI/CD pipeline execution:

- The latest Rego policies are first downloaded from S3 bucket by AWS CodeBuild
- The policies are then passed to OPA to evaluate the Terraform plan or configuration file.

- It will also fail the build if there are any policy violations detected (e.g., via email alerts)

7. Terraform init command

- The first and principal command that must be run in any new or existing Terraform project is the terraform init command (hashocorp, 2025).
- Then it starts and prepares working directory: set up backend, configure provider plugins, prepare environment for next operations like plan or apply.
- Terraform then downloads the appropriate provider binaries (i.e. AWS, GitHub, etc.) based on the .tf files and puts them in a hidden .terraform directory during the execution time.
- The configuration of a remote backend (such as an S3 with DynamoDB for state locking) will cause this to perform the initialization of that backend and any authentication that is needed. It is safe to run this command multiple times and should be rerunned whenever there are changes to the provider versions or backend configurations.This guarantees that there is a consistent and ready to use local working directory for managing infrastructure.


opa-terraform-cicd

```
OPA-TERRAFORM-CICD
  .terraform / providers / registry.terraform.io
    hashicorp
      aws
      awscc
    integrations / github / 4.31.0 / darwin_arm64
      CHANGELOG.md
      LICENSE
      README.md
      terraform-provider-github_v4.31.0
  policies
  tfvars
  .terraform.lock.hcl
  CodeBuild.tf
  CodePipeline.tf
  commands.txt
  locals.tf
  main.tf
  repository.tf
  s3.tf
  terraformApplyBuildspec.yml
  terraformComplianceScanBuildspec.yml
  terraformPlanBuildspec.yml
  terraformValidateBuildspec.yml
  vars.tf
```

8. Terraform Apply command

Terraform configuration files (.tf) declare the infrastructure they manage, and then the terraform apply command provisions or updates infrastructure as defined in those files. It is executed in such a way that it compares it's current state (stored in the state file) against the desired state given by the configuration and generates an execution plan. Next, it creates, updates or deletes the appropriate cloud resources in order to comply with the desired configuration, once user confirmation is received (Murali and Zeifman, 2025). terraform apply -var-file="tfvars/terraform.tfvars" -auto-approve

Performs the below operation:

- -var-file="tfvars/terraform.tfvars":

  Defines a variable file that contains the value of the input variables like AWS credentials, region, repository info, etc. This allows the dynamic configuration without the documentation of values inside the .tf files.

- -auto-approve:
  For the apply operation, skips the interactive approval step and proceeds automatically. This flag is useful in CI/CD pipelines where human intervention is not necessary or possible.

After following the above steps an infrastructure will be orchestrated on chosen AWS cloud account containing a codepipeline that generates a AWS infrastructure using terraform.

References

Arba, A. (2023) ‘CI/CD defined through terraform using AWS CodePipeline, AWS CodeCommit, and AWS CodeBuild’, *Medium*, 22 August. Available at: https://medium.com/@adrianarba/ci-cd-defined-throughterraform-using-aws-codepipeline-aws-codecommit-and-aws-codebuild-12ade4d9cfa3 (Accessed: 23 April 2025).

Birade, O. (2025) *Terraform .tfvars files: Variables Management with Examples*, *Spacelift*. Available at: https://spacelift.io/blog/[slug] (Accessed: 23 April 2025).

GitHub, D. (2025) *Managing your personal access tokens*, *GitHub Docs*. Available at: https://docsinternal.github.com/en/authentication/keeping-your-account-and-data-secure/managing-your-personalaccess-tokens (Accessed: 23 April 2025).

hashocorp (2025) *terraform init command reference | Terraform | HashiCorp Developer*, *terraform init command reference | Terraform | HashiCorp Developer*. Available at: https://developer.hashicorp.com/terraform/cli/commands/init (Accessed: 23 April 2025).

Infralovers (2024) *Enforcing Compliance with OPA and Terraform: A Practical Guide*, *Infralovers*. Available at: https://www.infralovers.com/blog/2024-06-28-terraform-opa-policies/ (Accessed: 23 April 2025).

Kavas, E. (2023) *Architecting AWS with Terraform: Design resilient and secure Cloud Infrastructures with Terraform on Amazon Web Services*. Packt Publishing Ltd.

Murali, A. and Zeifman, I. (2025) *Terraform Apply Command | Options, Examples and Best Practices | env0*. Available at: https://www.env0.com/blog/terraform-apply-guide-command-options-and-examples (Accessed: 23 April 2025).

OPA, D. (2025a) *Introduction*, *Open Policy Agent*. Available at: https://www.openpolicyagent.org/docs/latest/ (Accessed: 23 April 2025).

OPA, D. (2025b) *Policy Language*, *Open Policy Agent*. Available at: https://www.openpolicyagent.org/docs/latest/policy-language/ (Accessed: 23 April 2025).

Paul, A., Manoj, R. and S, U. (2024) 'Amazon Web Services Cloud Compliance Automation with Open Policy Agent', in *2024 International Conference on Expert Clouds and Applications (ICOECA)*. *2024 International Conference on Expert Clouds and Applications (ICOECA)*, pp. 313–317. Available at: https://doi.org/10.1109/ICOECA62351.2024.00063.

Salecha, R. (2023) 'Authentication and Authorization', in R. Salecha (ed.) *Practical GitOps: Infrastructure Management Using Terraform, AWS, and GitHub Actions*. Berkeley, CA: Apress, pp. 323–395. Available at: https://doi.org/10.1007/978-1-4842-8673-9_8.