HCLTech | Supercharging Progress™

Infrastructure as Code (IaC) Handover - GCP Landing Zone Deployment

Table of Contents

Introduction

This document serves as a comprehensive knowledge transfer for the IaC deployed for Quest Diagnostics. Its primary aim is to equip the designated team with the necessary understanding and practical guidance to effectively manage, maintain, and troubleshoot the current GCP landing Zone via IaC. This document consolidates information derived from the original Technical Design Document (TDD) prepared by GCP Architects and includes implementation details, configuration specifics, and validation procedures for all core networking components. Ultimately, this handover document empowers the team to independently operate and evolve the IaC environment to meet future business requirements. It promotes continuity, reduces reliance on specific individuals, and ensures the long-term stability and security of the cloud infrastructure.

Infrastructure as code (IaC)

Terraform

Terraform is an open-source Infrastructure as Code (IaC) tool that allows organizations to provision and manage cloud infrastructure in a consistent and automated manner across various cloud providers. As part of this project, Terraform has been utilized to automate the provisioning, configuration, and management of infrastructure resources, providing a declarative approach to defining infrastructure as code.

At its core, Terraform enables the definition of infrastructure through high-level configuration files, ensuring that all resources are deployed and maintained in a consistent and repeatable manner. This approach reduces manual intervention, mitigates human error, and ensures that infrastructure changes are controlled, auditable, and easily reproducible.

The use of Terraform in this project aligns with enterprise-level requirements for high availability, meeting compliance, offering a secure and transparent infrastructure deployment strategy.

State management

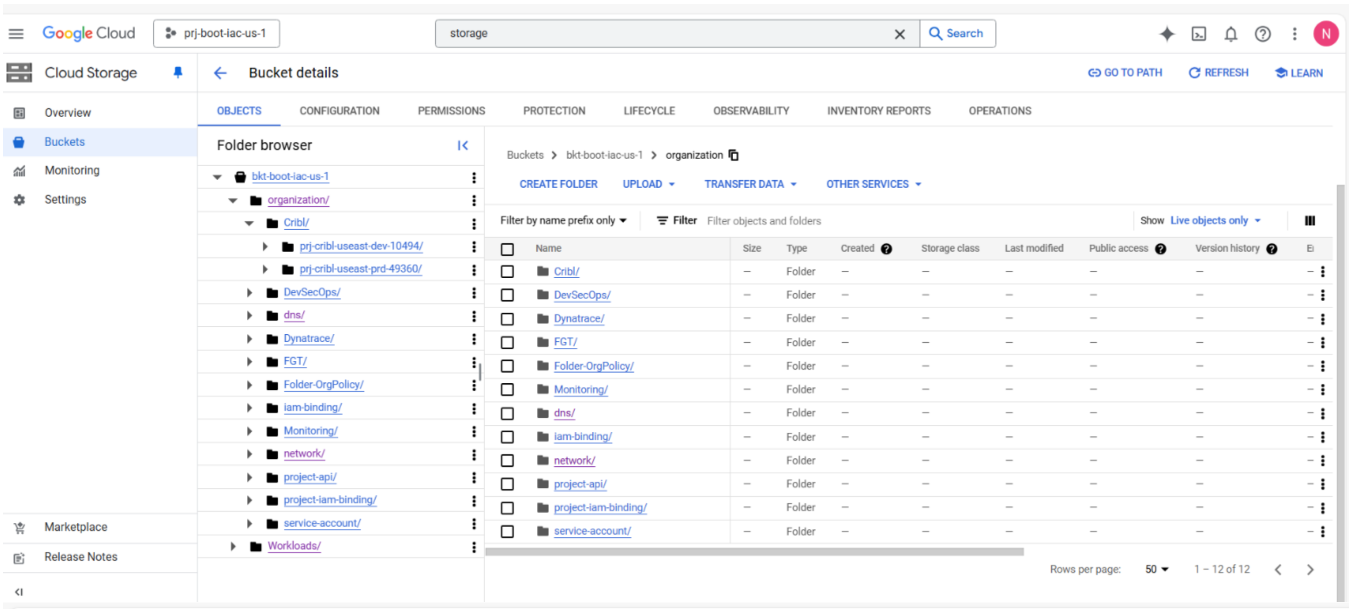
For this project, Google Cloud Storage (GCS) is utilized as the remote backend to store and manage Terraform state files.This centralized approach ensures that the Terraform state is securely stored, highly available, and can be accessed by multiple team members or automation tools while ensuring consistency in the infrastructure provisioning process.

- **Backend Structure and Segregation**

The state files are organized into a well-structured folder hierarchy within a dedicated GCS bucket, ensuring that the state management process remains organized and scalable. The folder structure is logically divided into two key sections:

- **Organization Folder**: This folder contains the state files for the resources associated with the landing zone. These resources are shared across the organization, including foundational infrastructure like networking, IAM policies, and other shared services that provide the foundation for workloads.

- **Workloads Folder**:This folder holds the state files for project-specific workloads. Each project's infrastructure—such as compute, storage, and application-specific resources—has its own isolated state file within this folder, ensuring that state for each workload is maintained separately.



Google Cloud | prj-boot-iac-us-1 | storage | Search

Cloud Storage — Bucket details

GO TO PATH | REFRESH | LEARN

Overview · Buckets · Monitoring · Settings · Marketplace · Release Notes

OBJECTS | CONFIGURATION | PERMISSIONS | PROTECTION | LIFECYCLE | OBSERVABILITY | INVENTORY REPORTS | OPERATIONS

Folder browser
- bkt-boot-iac-us-1
  - organization/
    - Cribl/
      - prj-cribl-useast-dev-10494/
      - prj-cribl-useast-prd-49360/
    - DevSecOps/
    - dns/
    - Dynatrace/
    - FGT/
    - Folder-OrgPolicy/
    - iam-binding/
    - Monitoring/
    - network/
    - project-api/
    - project-iam-binding/
    - service-account/
  - Workloads/

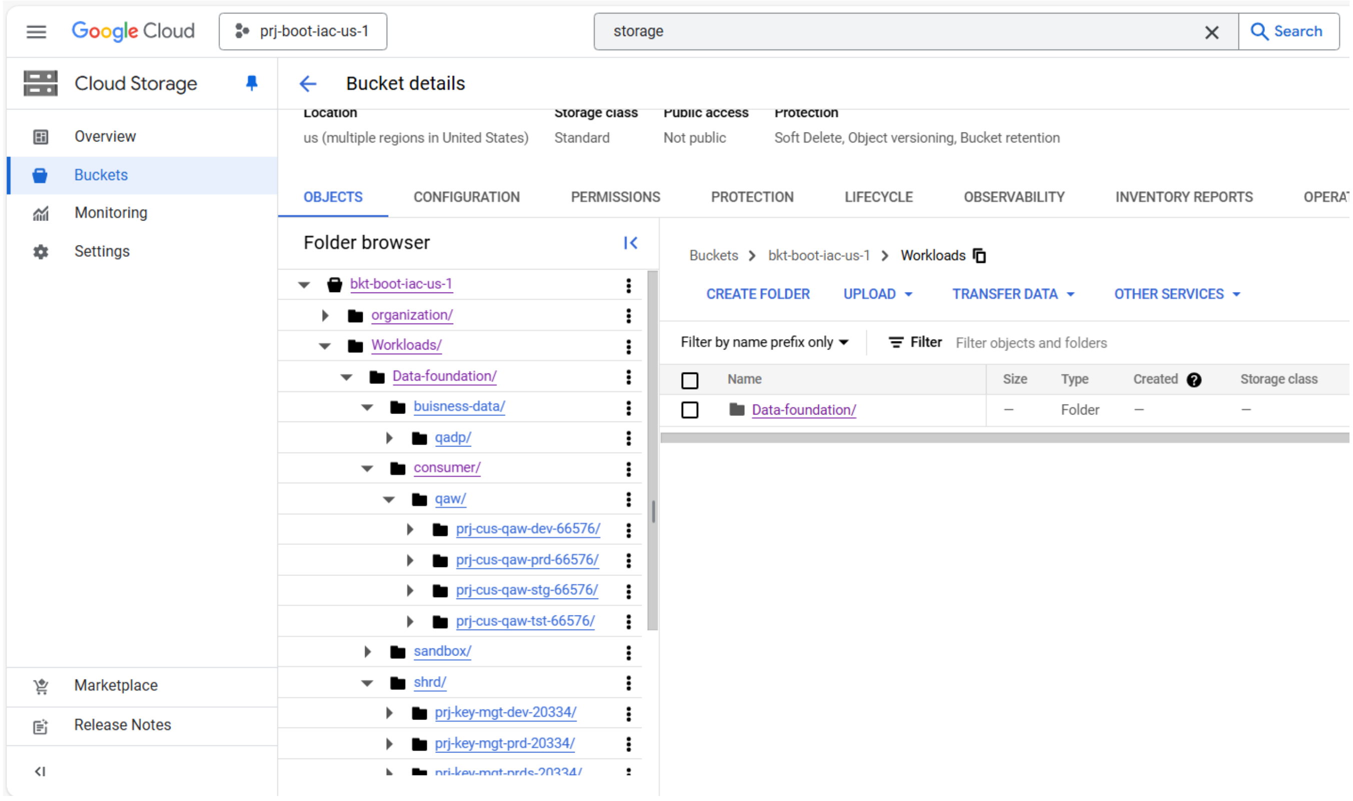
Buckets > bkt-boot-iac-us-1 > organization

CREATE FOLDER | UPLOAD | TRANSFER DATA | OTHER SERVICES

Filter by name prefix only | Filter | Filter objects and folders | Show Live objects only

Name	Size	Type	Created	Storage class	Last modified	Public access	Version history	E
Cribl/	—	Folder	—	—	—	—	—	—
DevSecOps/	—	Folder	—	—	—	—	—	—
Dynatrace/	—	Folder	—	—	—	—	—	—
FGT/	—	Folder	—	—	—	—	—	—
Folder-OrgPolicy/	—	Folder	—	—	—	—	—	—
Monitoring/	—	Folder	—	—	—	—	—	—
dns/	—	Folder	—	—	—	—	—	—
iam-binding/	—	Folder	—	—	—	—	—	—
network/	—	Folder	—	—	—	—	—	—
project-api/	—	Folder	—	—	—	—	—	—
project-iam-binding/	—	Folder	—	—	—	—	—	—
service-account/	—	Folder	—	—	—	—	—	—

Rows per page: 50 | 1 – 12 of 12



Google Cloud | prj-boot-iac-us-1 | storage | Search

Cloud Storage — Bucket details

Location	Storage class	Public access	Protection
us (multiple regions in United States)	Standard	Not public	Soft Delete, Object versioning, Bucket retention

Overview · Buckets · Monitoring · Settings · Marketplace · Release Notes

OBJECTS | CONFIGURATION | PERMISSIONS | PROTECTION | LIFECYCLE | OBSERVABILITY | INVENTORY REPORTS | OPERA

Folder browser
- bkt-boot-iac-us-1
  - organization/
  - Workloads/
    - Data-foundation/
      - buisness-data/
        - qadp/
      - consumer/
        - qaw/
          - prj-cus-qaw-dev-66576/
          - prj-cus-qaw-prd-66576/
          - prj-cus-qaw-stg-66576/
          - prj-cus-qaw-tst-66576/
      - sandbox/
      - shrd/
        - prj-key-mgt-dev-20334/
        - prj-key-mgt-prd-20334/
        - prj-key-mgt-prds-20334/

Buckets > bkt-boot-iac-us-1 > Workloads

CREATE FOLDER | UPLOAD | TRANSFER DATA | OTHER SERVICES

Filter by name prefix only | Filter | Filter objects and folders

Name	Size	Type	Created	Storage class
Data-foundation/	—	Folder	—	—

Repositories

In this project, two types of repositories are utilized to manage the Terraform infrastructure code: Shared Module Repositories and Template Repositories. Each repository type serves a specific purpose in the overall infrastructure as code (IaC) management process, ensuring a clear separation of concerns and promoting reusability and maintainability across environments.

➢ Shared Module Repositories

Shared Module Repositories contain the core Terraform modules that define the infrastructure resources for different services. These modules contain reusable code blocks that represent infrastructure components, such as networking, IAM, compute, storage, and other services. The shared modules are designed to be environmentagnostic and serve as a central resource for defining and provisioning infrastructure components across multiple projects.

Each module repository is structured in a way that supports best practices for modularity, allowing users to easily reference and apply specific modules to different environments. The modules are stored separately for each service (e.g., networking, IAM, storage), ensuring that they can be independently maintained and updated as needed.

Reference: Attached is the table listing the shared module repositories.

S.No	Repository Name	Link
1	**dso-gcpfoundation-iac-module-folder**	QDXEnterpriseOrg/dso-gcpfoundation-iac-module-folder: Repository for gcpfoundation-iac
2	**dso-gcpfoundation-iac-module-network**	QDXEnterpriseOrg/dso-gcpfoundation-iac-module-network: Repository for gcpfoundation-iac
3	**dso-gcpfoundation-iac-module-gke**	QDXEnterpriseOrg/dso-gcpfoundation-iac-module-gke: Repository for gcpfoundation
4	**dso-gcpfoundation-iac-module-service-account**	QDXEnterpriseOrg/dso-gcpfoundation-iac-module-service-account: Repository for gcpfoundation-iac
5	**dso-gcpfoundation-iac-module-secret-manager**	QDXEnterpriseOrg/dso-gcpfoundation-iac-module-secret-manager: Repository for gcpfoundation
6	**dso-gcpfoundation-iac-module-compute-instance**	QDXEnterpriseOrg/dso-gcpfoundation-iac-module-compute-instance: Repository for gcpfoundation
7	**dso-gcpfoundation-iac-module-instance-template**	QDXEnterpriseOrg/dso-gcpfoundation-iac-module-instance-template: Repository for gcpfoundation
8	**dso-gcpfoundation-iac-module-fortigate**	QDXEnterpriseOrg/dso-gcpfoundation-iac-module-fortigate: Repository for gcpfoundation
9	**dso-gcpfoundation-iac-module-composer**	QDXEnterpriseOrg/dso-gcpfoundation-iac-module-composer: Repository for gcpfoundation
10	**dso-gcpfoundation-iac-module-iam**	QDXEnterpriseOrg/dso-gcpfoundation-iac-module-iam: Repository for gcpfoundation-iac
11	**dso-gcpfoundation-iac-module-kms**	QDXEnterpriseOrg/dso-gcpfoundation-iac-module-kms: Repository for gcpfoundation
12	**dso-gcpfoundation-iac-module-storage**	QDXEnterpriseOrg/dso-gcpfoundation-iac-module-storage: Repository for gcpfoundation
13	**dso-gcpfoundation-iac-module-dns**	QDXEnterpriseOrg/dso-gcpfoundation-iac-module-dns: Repository for gcpfoundation
14	**dso-gcpfoundation-iac-module-artifact-registry**	QDXEnterpriseOrg/dso-gcpfoundation-iac-module-artifact-registry: Repository for gcpfoundation
15	**dso-gcpfoundation-iac-module-project**	QDXEnterpriseOrg/dso-gcpfoundation-iac-module-project: Repository for gcpfoundation-iac
16	**dso-gcpfoundation-iac-module-org-policy**	QDXEnterpriseOrg/dso-gcpfoundation-iac-module-org-policy: Repository for gcpfoundation-iac
17	**dso-gcpfoundation-iac-module-auto-scaler**	QDXEnterpriseOrg/dso-gcpfoundation-iac-module-auto-scaler: Repository for gcpfoundation
18	**dso-gcpfoundation-iac-module-mig**	QDXEnterpriseOrg/dso-gcpfoundation-iac-module-mig: Repository for gcpfoundation

➢ **Template Repositories:**

Template Repositories, on the other hand, are used for defining project-specific configurations that reference the shared modules. These repositories allow users to pass environment-specific variables (such as .tfvars files) to the core modules, thereby enabling the deployment of infrastructure tailored to specific applications, environments, or stages.
Shared module Repositories.

Each Template Repository is associated with a particular application or environment and typically contains the following:

- **Environment-specific variables**: A .tfvars file where values for variables like project names, regions, and resource configurations are defined.
- **Module Calls**: The Terraform code that calls the appropriate shared modules and applies the specific values defined in the .tfvars files.

Additionally, Landing Zone resources such as projects, folders, and networks are defined in separate template repositories. These templates define the foundational infrastructure elements that support all workloads and applications, such as organizational structure, network setup, and other shared resources that span multiple environments.

Reference: Attached is the table listing the foundational repositories.

S.No	Repository Name	Link
1	**dso-gcpfoundation-iac-org**	QDXEnterpriseOrg/dso-gcpfoundation-iac-org: Repository for gcpfoundation-iac
2	**dso-gcpfoundation-iac-inf**	QDXEnterpriseOrg/dso-gcpfoundation-iac-inf: Repository for gcpfoundation-iac
3	**dso-gcpfoundation-iac-network**	QDXEnterpriseOrg/dso-gcpfoundation-iac-network: Repository for gcpfoundation-iac
4	**dso-gcpfoundation-iac-iam**	QDXEnterpriseOrg/dso-gcpfoundation-iac-iam: Repository for gcpfoundation-iac
5	**dso-gcpfoundation-iac-gha-runner**	QDXEnterpriseOrg/dso-gcpfoundation-iac-gha-runner: Repository for gcpfoundation
6	**dso-gcpfoundation-iac-prod-gha-runner**	QDXEnterpriseOrg/dso-gcpfoundation-iac-prod-gha-runner: Repository for gcpfoundation
7	**dso-gcp-dynatrace-useast-prod**	QDXEnterpriseOrg/dso-gcp-dynatrace-useast-prod: Repository for gcp
8	**dso-gcp-dynatrace-useast-dev-75825**	QDXEnterpriseOrg/dso-gcp-dynatrace-useast-dev-75825: Repository for gcps
9	**dso-gcp-cribl-useast-dev-10494**	QDXEnterpriseOrg/dso-gcp-cribl-useast-dev-10494: Repository for gcp
10	**dso-gcp-cribl-useast-prd**	QDXEnterpriseOrg/dso-gcp-cribl-useast-prd: Repository for gcp
11	**dso-gcpfoundation-iac-prod-fortigate**	QDXEnterpriseOrg/dso-gcpfoundation-iac-prod-fortigate: Repository for gcpfoundation
12	**dso-gcpfoundation-iac-non-prod-fortigate**	QDXEnterpriseOrg/dso-gcpfoundation-iac-non-prod-fortigate: Repository for gcpfoundation

Reference: Attached is the table listing the Data Projects (Infrastructure) repositories.

S.No	Repository Name	Link

1	**dso-gcp-eda-qadp-bus-dev**	QDXEnterpriseOrg/dso-gcp-eda-qadp-bus-dev: Repository for gcp
2	**dso-gcp-eda-qadp-prt-dev**	QDXEnterpriseOrg/dso-gcp-eda-qadp-prt-dev: Repository for gcp
3	**dso-gcp-eda-qadp-int-dev**	QDXEnterpriseOrg/dso-gcp-eda-qadp-int-dev: Repository for gcp
4	**dso-gcp-eda-qadp-raw-dev**	QDXEnterpriseOrg/dso-gcp-eda-qadp-raw-dev: Repository for gcp
5	**dso-gcp-eda-qadp-prt-prd**	QDXEnterpriseOrg/dso-gcp-eda-qadp-prt-prd: Repository for gcp
6	**dso-gcp-eda-qadp-int-prd**	QDXEnterpriseOrg/dso-gcp-eda-qadp-int-prd: Repository for gcp
7	**dso-gcp-eda-qadp-bus-prd**	QDXEnterpriseOrg/dso-gcp-eda-qadp-bus-prd: Repository for gcp
8	**dso-gcp-eda-qadp-raw-prd**	QDXEnterpriseOrg/dso-gcp-eda-qadp-raw-prd: Repository for gcp
9	**dso-gcp-cus-qaw-dev-66576**	QDXEnterpriseOrg/dso-gcp-cus-qaw-dev-66576: Repository for gcp
10	**dso-gcp-cus-qaw-prd-66576**	QDXEnterpriseOrg/dso-gcp-cus-qaw-prd-66576: Repository for gcp
11	**dso-gcp-shrd-dev-67236**	QDXEnterpriseOrg/dso-gcp-shrd-dev-67236: Repository for gcp
12	**dso-gcp-shrd-prd-67236**	QDXEnterpriseOrg/dso-gcp-shrd-prd-67236: Repository for gcp
13	**dso-gcp-sb-eda-tst01-raw-dev**	QDXEnterpriseOrg/dso-gcp-sb-eda-tst01-raw-dev: Repository for gcp
14	**dso-gcp-sb-eda-tst02-raw-dev**	QDXEnterpriseOrg/dso-gcp-sb-eda-tst02-raw-dev: Repository for gcp
15	**dso-gcp-sb-eda-tst02-prt-dev**	QDXEnterpriseOrg/dso-gcp-sb-eda-tst02-prt-dev: Repository for gcp
16	**dso-gcp-sb-eda-tst02-int-dev**	QDXEnterpriseOrg/dso-gcp-sb-eda-tst02-int-dev: Repository for gcp
17	**dso-gcp-sb-eda-tst02-bus-dev**	QDXEnterpriseOrg/dso-gcp-sb-eda-tst02-bus-dev: Repository for gcp
18	**dso-gcp-sb-eda-tst01-prt-dev**	QDXEnterpriseOrg/dso-gcp-sb-eda-tst01-prt-dev: Repository for gcp
19	**dso-gcp-sb-eda-tst01-int-dev**	QDXEnterpriseOrg/dso-gcp-sb-eda-tst01-int-dev: Repository for gcp
20	**dso-gcp-sb-eda-tst01-bus-dev**	QDXEnterpriseOrg/dso-gcp-sb-eda-tst01-bus-dev: Repository for gcp
21	**dso-gcpfoundation-kms**	QDXEnterpriseOrg/dso-gcpfoundation-kms: Repository for gcpfoundation
22	**dso-gcp-key-mgt-prd-20334**	QDXEnterpriseOrg/dso-gcp-key-mgt-prd-20334: Repository for gcp

Modules Structure

The module structure in this project follows best practices for Terraform module organization, ensuring modularity, reusability, and maintainability. Each module represents a distinct component of the infrastructure, and it is comprised of several files, each serving a specific role. These files collectively define and manage the infrastructure resources, handle inputs/outputs, and ensure that the module behaves consistently.

Each module in this project follows a standard set of files that work together to define the desired infrastructure state. The main files typically included in a module are:

- **main.tf – Resource Definitions**
  - This is the primary file that defines the Terraform resources that make up the infrastructure component. It includes the actual configuration for the resources, such as virtual machines, VPCs, databases, storage, and IAM roles.
  - Example: For a networking module, this would contain the configuration for VPC, subnets, and firewall rules.

- **variables.tf** – **Input Variables**
    - This file defines the input variables that are passed into the module. These variables allow the module to be reused in multiple contexts with different values, making the module flexible and customizable.
    - Example: The networking module could have variables for project_id, region, vpc_name, etc., which are used in the main.tf to create resources specific to the input values.

- **outputs.tf** – **Output Values**
    - This file defines the output variables that provide important information back to the parent module or to other modules. These outputs can be used to reference resources created within the module, such as the ID of a newly created VPC or the IP address of a deployed instance.
    - Example: The networking module could output the VPC ID (vpc_id) or subnet IP ranges (subnet_ips) which are needed by other modules or environments.

- **README.md – Module Documentation**
    - The README.md file provides detailed documentation about the module, including its purpose, how to use it, the required input variables, outputs. This ensures that users can easily understand and implement the module in their own environments.

Continuous integration and delivery

CI/CD pipeline overview

To ensure consistency, automation and security in Infrastructure deployment, we have implemented a CI/CD Pipeline using GitHub Actions for Iac deployment.

CI/CD tooling

The CI/CD pipeline in this project is designed to automate the provisioning, updating, and management of infrastructure using Terraform. The pipeline is powered by GitHub Actions, ensuring a smooth and efficient workflow from code commit to deployment.

GitHub Actions provides a robust and scalable solution for continuous integration and delivery, which tightly integrates with the GitHub version control system, ensuring that infrastructure code is managed, validated, and deployed efficiently.

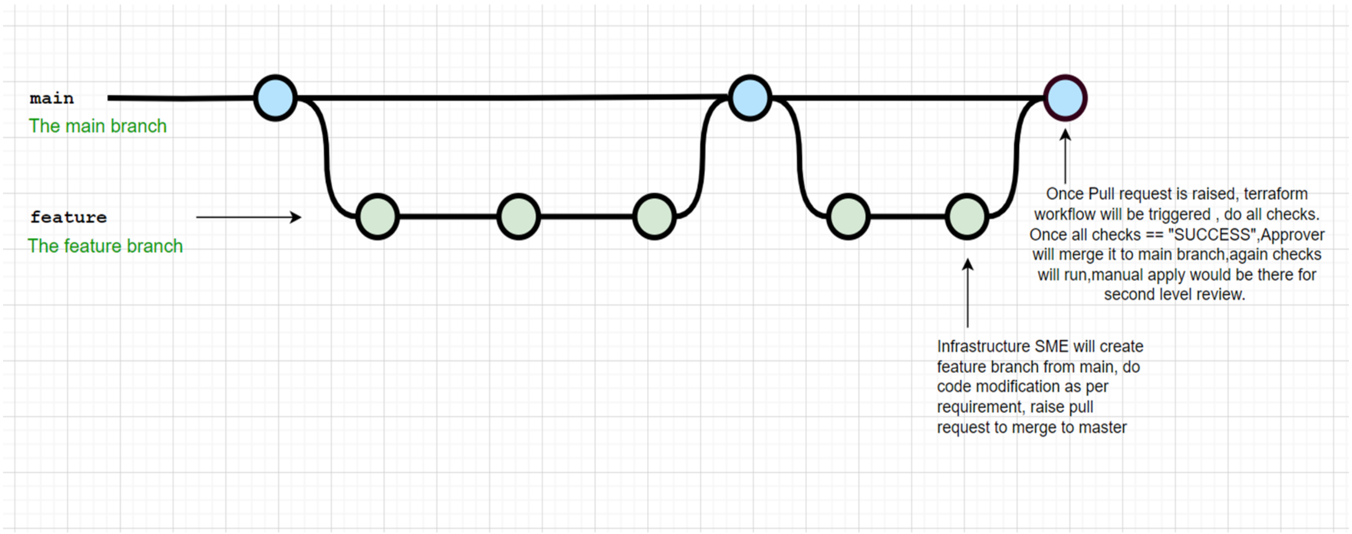
Build vs Deploy

The build phase involves creating reusable and standardized terraform modules that encapsulate best practices and configurations. These modules serves as the foundation for deploying various infrastructure components across deployment environments.

The deploy Phase involves provisioning by leveraging the shared terraform modules. This is done through template repositories which serves as the entry point for deployment and Integrates with GitHub Actions Workflows for automation. These configurations call the shared modules, passing values to .tfvars files(configuration files).This approach ensures a modular ,automated, and scalable infrastructure provisioning process while maintain security & compliance.

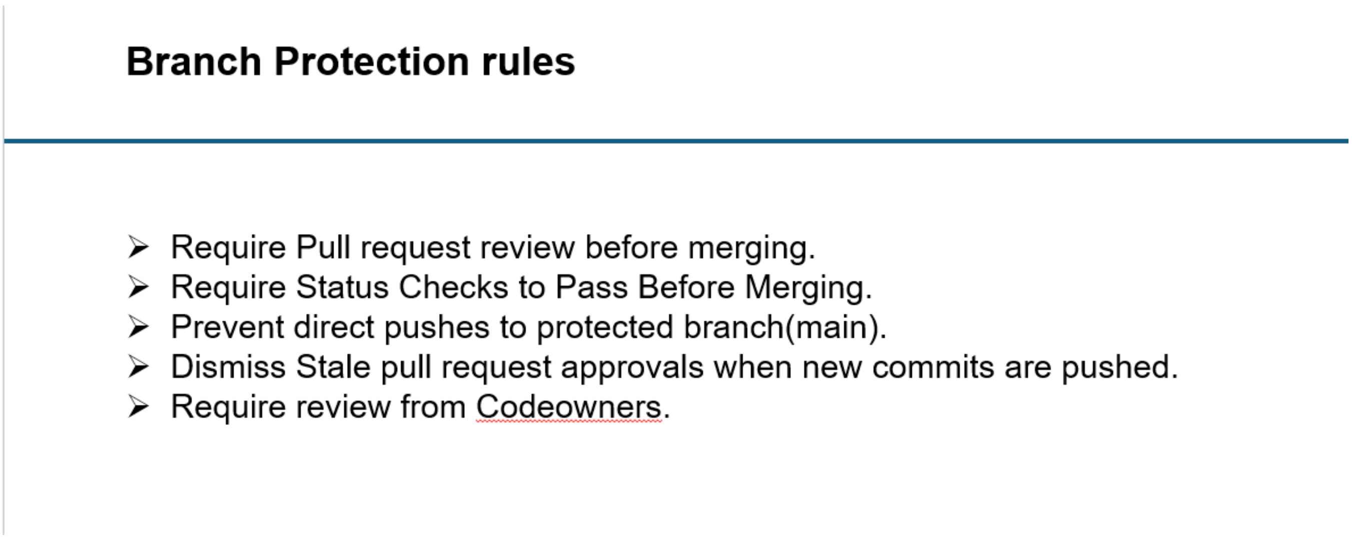
Branching strategy

Our branching strategy follows a streamlined process where all changes are developed in feature branches and merged into the main branch after successful checks and approvals. This ensures that the main branch always contains stable and production-ready code.


main
The main branch
feature
The feature branch
Once Pull request is raised, terraform workflow will be triggered , do all checks. Once all checks == "SUCCESS",Approver will merge it to main branch,again checks will run,manual apply would be there for second level review.
Infrastructure SME will create feature branch from main, do code modification as per requirement, raise pull request to merge to master

- **Branch Protection rules**

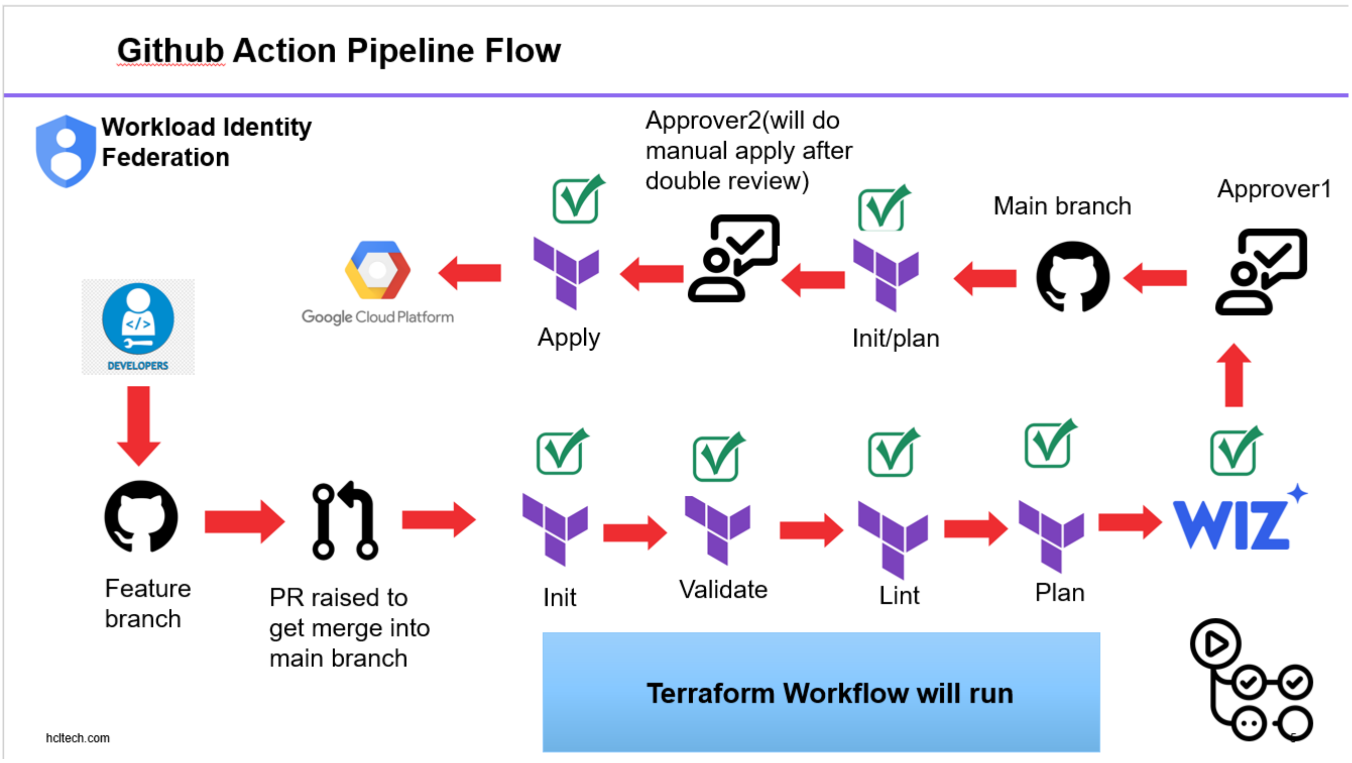
Branch Protection rules has been enforced to maintain code integrity and stability. Key rules include mandatory pull request reviews, required status checks and restrict direct pushes to main.



Branch Protection rules

- ➢ Require Pull request review before merging.
- ➢ Require Status Checks to Pass Before Merging.
- ➢ Prevent direct pushes to protected branch(main).
- ➢ Dismiss Stale pull request approvals when new commits are pushed.
- ➢ Require review from Codeowners.

CI/CD Workflow


Github Action Pipeline Flow
Workload Identity Federation
DEVELOPERS
Feature branch
PR raised to get merge into main branch
Init
Validate
Lint
Plan
WIZ
Approver1
Main branch
Init/plan
Approver2(will do manual apply after double review)
Apply
Google Cloud Platform
Terraform Workflow will run
hcltech.com

The CI/CD workflow is designed to ensure that infrastructure changes are tested, validated, and deployed in a controlled and predictable manner. The process is divided into two main workflows:

1. **Terraform Plan Workflow**: This workflow runs during the Pull Request (PR) creation and ensures that all checks, validations, and planning steps are performed before changes are merged into the main branch.
2. **Terraform Apply Workflow**: This workflow runs after the changes are merged into the main branch. It executes the plan and apply stages and requires manual approval before applying the changes to the infrastructure.

CI/CD Workflow Steps

The workflow follows a well-structured, multi-step process for handling infrastructure changes. The general process is as follows:

Step 1: Commit Changes to Feature Branch

When an infra engineer (or any team member) needs to make changes to the infrastructure, they begin by committing changes to a feature branch. These changes could involve modifications to Terraform configurations or other related resources.

Step 2: Create a Pull Request (PR)

Once the changes are committed to the feature branch, the next step is to raise a Pull Request (PR) against the main branch. This triggers the Terraform Plan Workflow in the pipeline.

Step 3: Run Terraform Plan Workflow

When the PR is created, a set of checks and validation steps will automatically be triggered:

- **Terraform Init**: Initializes the working directory for Terraform, ensuring the backend and required providers are configured.
- **Terraform Validate**: This step validates the syntax and correctness of the Terraform code.
- **TFlint**: Runs TFlint to check the Terraform code for best practices, conventions, and potential issues.
- **Wiz.io Scan**: A security scanning tool is run to detect potential vulnerabilities or misconfigurations in the Terraform code.
- **Terraform Plan**: This generates the execution plan, previewing the infrastructure changes that will be made once applied.

Step 4: Approval and Merge

Once all checks have passed successfully (i.e., all validations, linting, and security scans), the PR is ready for review. Here’s how the approval process works:

- **Approver 1** (usually a senior infrastructure engineer or team lead) reviews the code in the PR.
- **Approver 1** checks the Terraform code, the output of the terraform plan command, and any other relevant details.
- **Approval**: Once Approver1 confirms that everything looks correct, they merge the PR into the main branch.

Step 5: Trigger Terraform Apply Workflow

After the PR is merged into the main branch, a separate Terraform Plan & Apply Workflow is triggered. This workflow is responsible for ensuring that the infrastructure changes are applied to the target environment.

Step 6: Run Plan Job

- **Terraform Plan**: The plan job is executed again after the changes are merged. This job creates a new execution plan, previewing the changes that will be applied to the cloud infrastructure.
- This ensures that the infrastructure changes reflect what was previously tested and validated during the PR phase and confirms that no additional changes are introduced.

Step 7: Manual Approval for Apply Job

Before applying the infrastructure changes, the pipeline will wait for manual approval. This is an important safeguard to ensure that someone (typically a senior team member or operations lead) reviews the plan and approves the changes before they are applied.

- **Approver 2** (typically an operations or security engineer) reviews the execution plan, confirming that all proposed infrastructure changes are correct and aligned with the desired state.
- **Manual Approval**: Only after Approver 2 approves the changes will the apply job proceed to execute and apply the infrastructure changes.

Step 8: Apply Changes

- **Terraform Apply**: Once the manual approval is granted, the apply job runs. This will apply the Terraform plan and provision the required resources in the target cloud environment (e.g., Google Cloud Platform).
- The infrastructure is now updated according to the changes committed in the PR, and the workflow completes successfully

Key Decisions

❖ **Secrets Management**:

All sensitive information, including API keys and service account credentials, are securely stored in GCP Secrets Manager. This ensures that sensitive data is never exposed in the codebase or version control.

❖ **Authentication**:

Authentication to Google Cloud Platform (GCP) is handled securely via Workload Identity Federation. For non-production environments, non-prod service accounts (SA) are used, while for production, a separate prod service account is utilized to ensure strict separation of duties.

❖ **Environment Isolation**:

Separate backends are maintained for different services and projects to ensure that non-production and production environments are isolated and managed independently. This minimizes the risk of cross-environment issues.

❖ **CI/CD Runners**:

Separate Dedicated CI/CD runners are configured for both production and non-production environments. This ensures that the production environment has a dedicated, secure pipeline while the non-prod environment runs in its own isolated pipeline for testing and development purposes. CI/CD Runner & WIF details has been covered in below doc.

Reference: **GH_selfhost_runner_Documentation.docx**

❖ **Review and Approval Process**:

- **Code Review**: All changes are reviewed through a structured merge request (MR) process. Each MR must undergo a thorough review by a designated approver who ensures that best practices, security protocols, and terraform coding standards are followed.
- **Automated Checks**: As part of the merge request process, automated checks (including Terraform plan validation, security scans, and linting) are run to ensure the changes are safe and do not introduce any issues or vulnerabilities.

HCLTech | Supercharging Progress™

hcltech.com