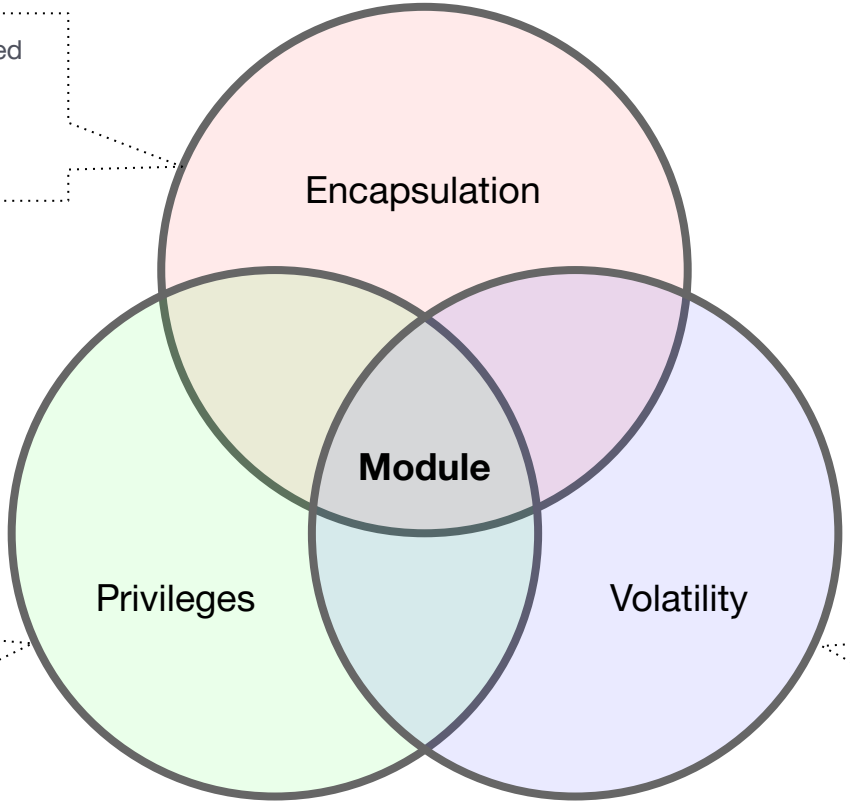
Why modules?

- Speed up adoption
- Economies of scale
- Good for code organisation
- Treat infrastructure pieces as a black box
- Lowers barrier of entry
- DRY principle

Module creation considerations

Can the infrastructure be grouped as one entity?

Convenience vs flexibility.



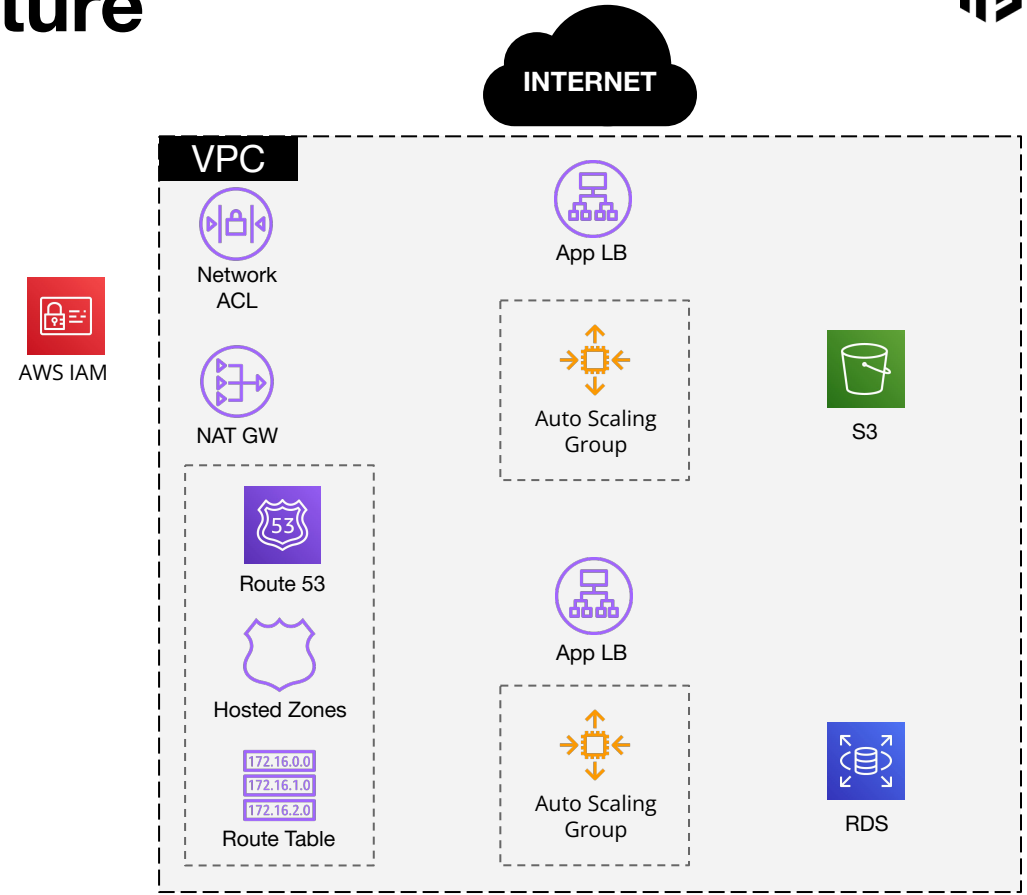
If the infrastructure contains sensitive data, having it in a single module increases its separation.

Is the infrastructure fairly static or under constant change? Group by volatility to reduce risk of unnecessary change.

Example infrastructure

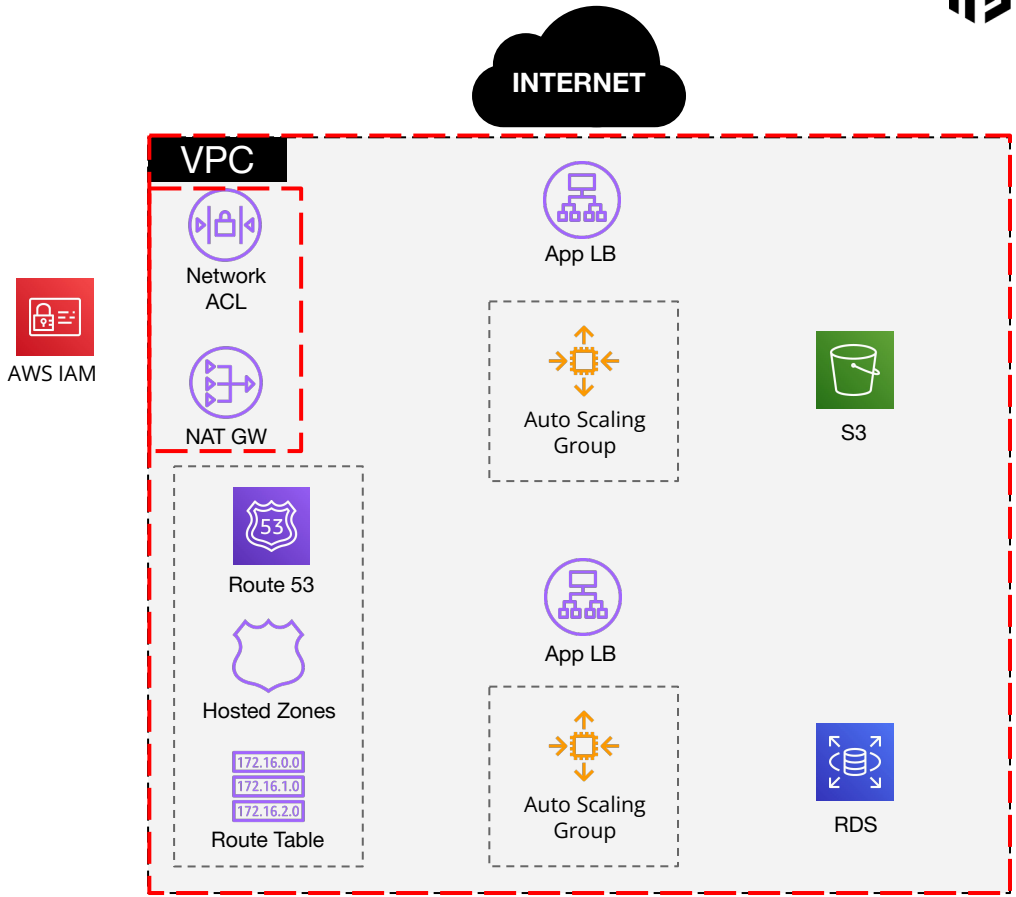
Basic 3 Tier Infrastructure.

- VPC
- Front-end application
- Back-end application
- DB
- Storage
- Networking
- Routing



VPC module

- VPC
- Subnets
- NAT gateway
- Network ACLs
- Cloudtrail logs
- Networking
  - Peering
  - Direct Connect

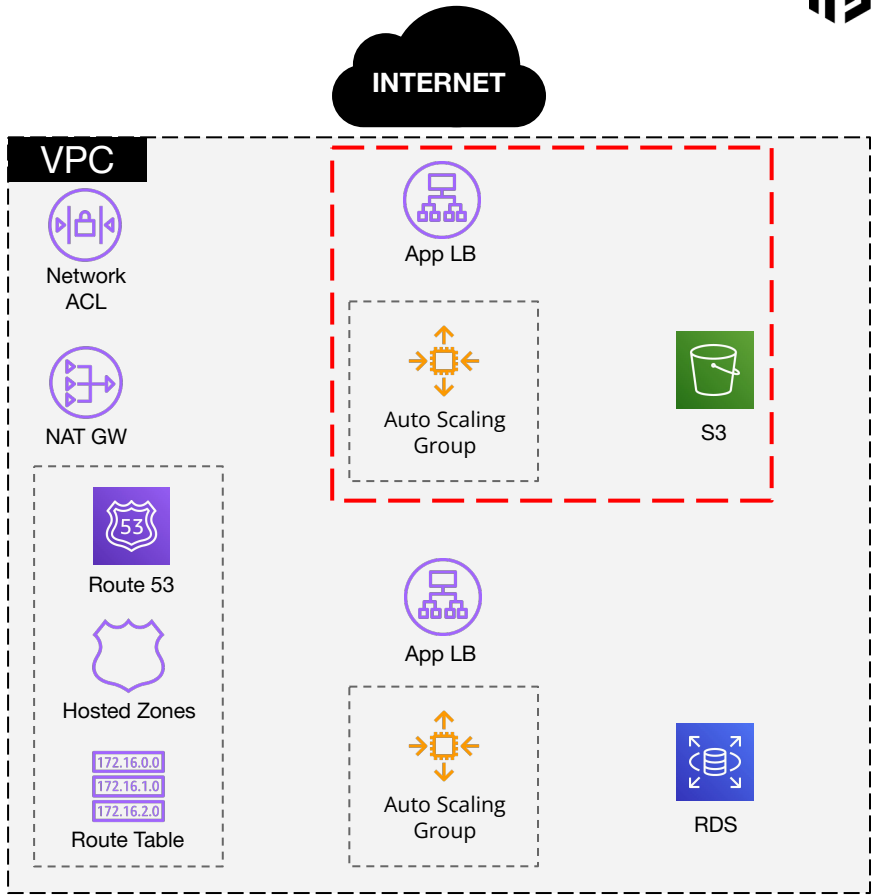


Application 1 module

- Application 1 load balancers
- Application 1 autoscaling group
- Associated storage
- Security groups inside the application
- Logging

AWS IAM

Everything to run the app and deploy.

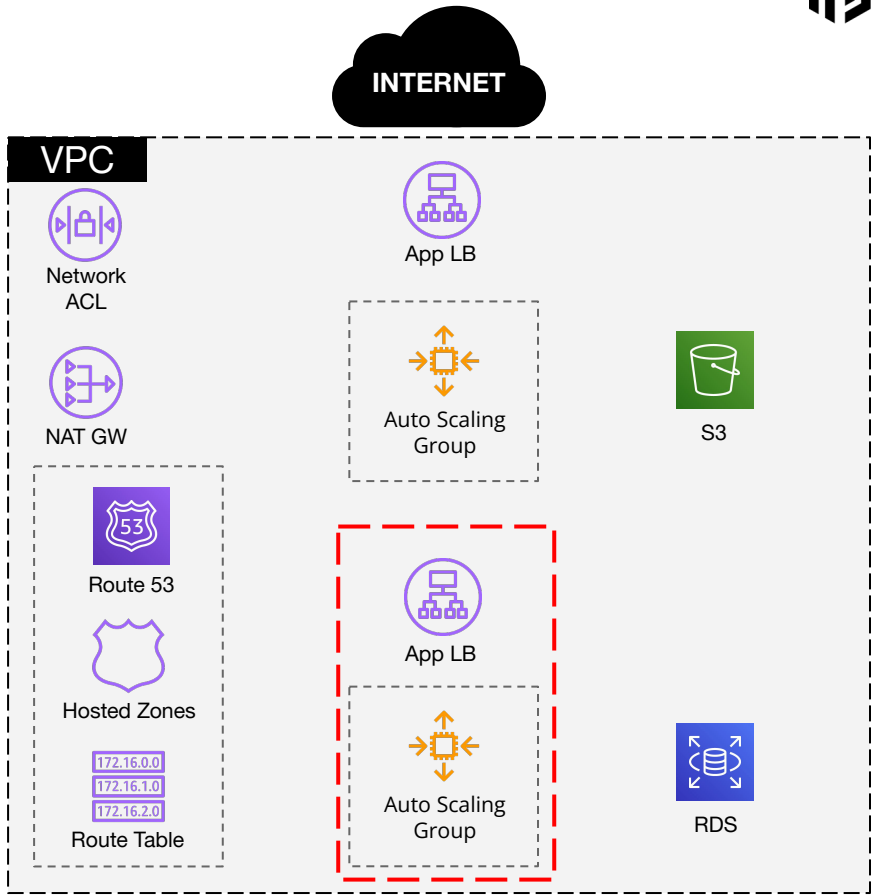


Application 2 module

- Application 2 load balancers
- Application 2 autoscaling group
- Associated storage
- Security groups inside the application
- Logging

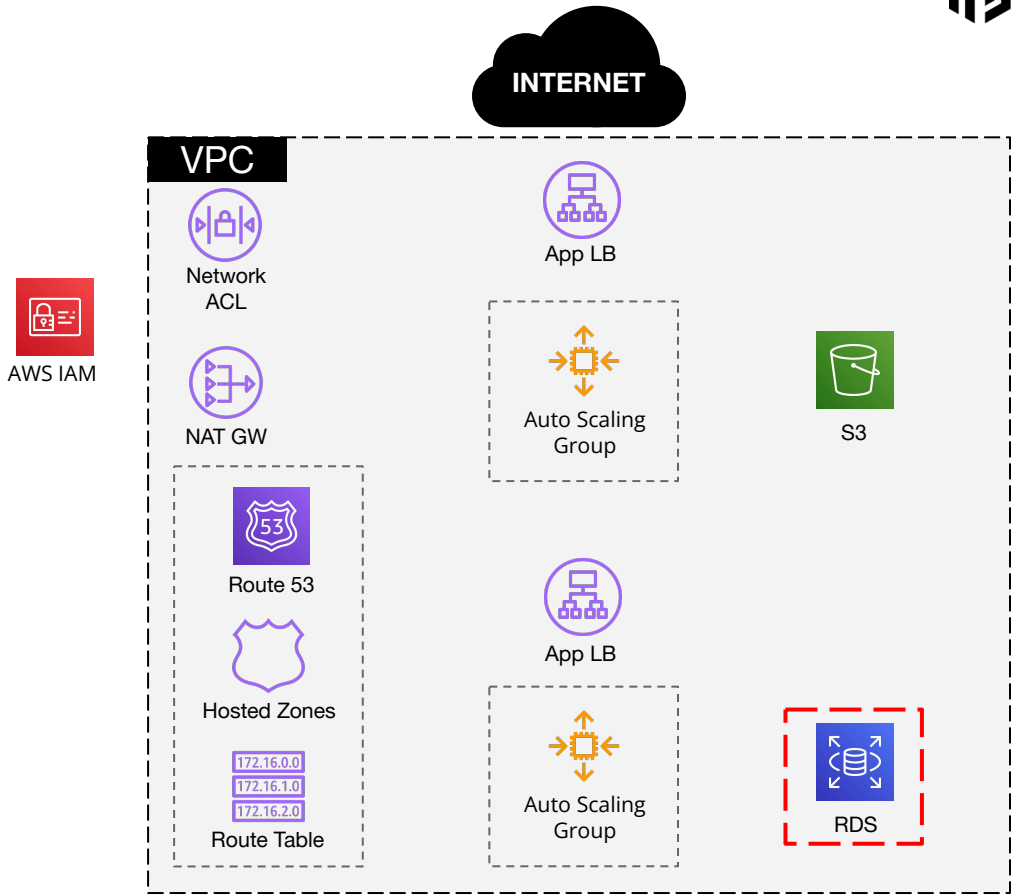
AWS IAM

Everything to run the app and deploy.



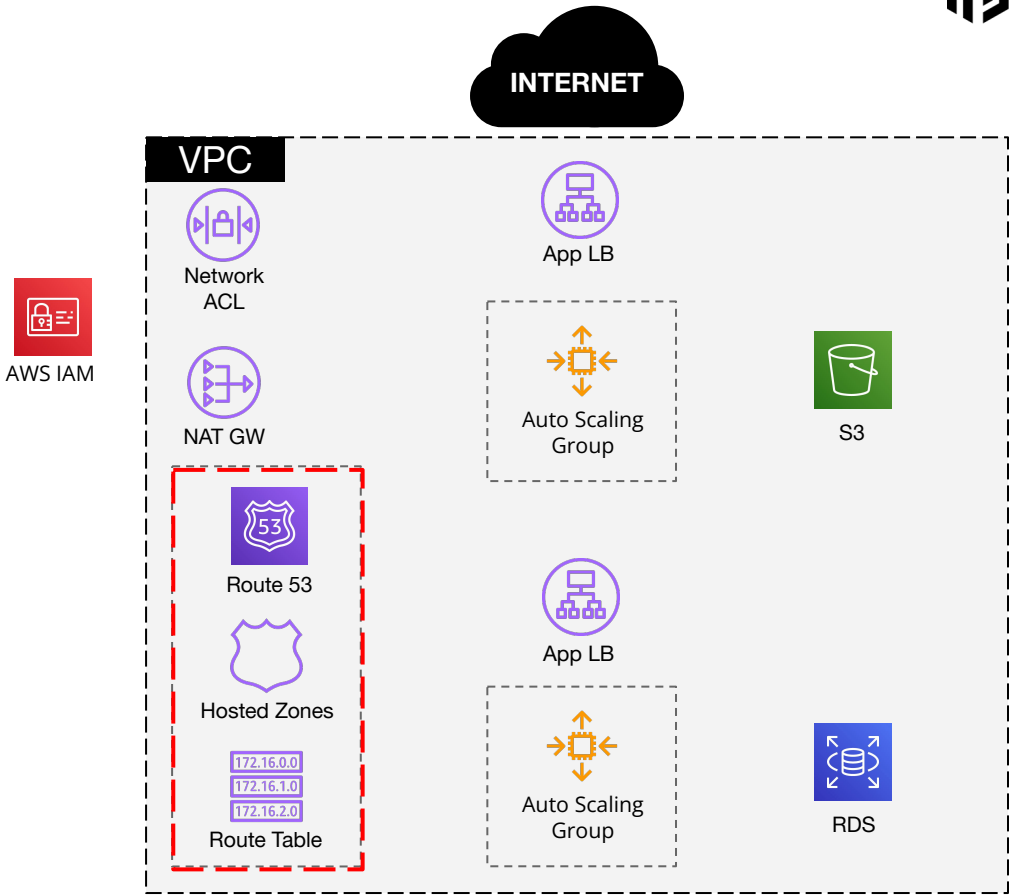
Database module

- SQL instances
- All associated storage
- All backup data
- Logging



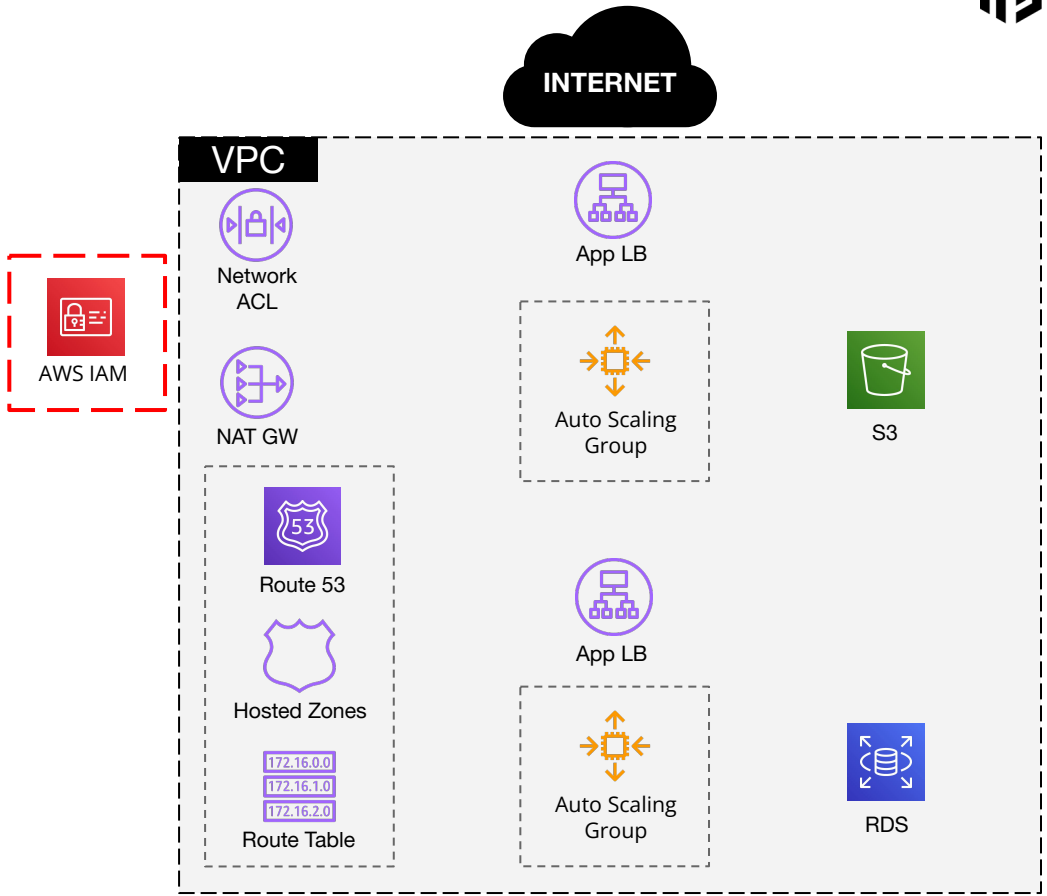
Routing module

- Route 53
- Hosted Zones
- Route tables



Security module

- IAM
- Access
- Security groups
- MFA



The module journey

- Simplicity
  - Can you define the module purpose in a reasonably simple sentence?
- One module = One repo = One owner/group
  - Tagged releases to the default branch (main) - pin
- Work with an early adopter and develop as new groups join
- Have an advertised, jointly-curated roadmap bringing in your users
- Create a community
  - Some users just want to use, other to contribute

Module terminology

- Child module
  - A set of Terraform files constituting a set of resources
  - Typically versioned in a Git repository
  - Designed to be called by a root module
- No-code module
  - Child module with added provider and review of variables
- Root module
  - A set of Terraform files describing desired topology
  - Synonymous with a TFE workspace configuration
- Submodule
  - Optional subdirectories containing sets of Terraform files
  - Used to logically arrange larger configurations

Terraform Test

Terraform Test

A systematic approach to testing

- Write tests using HCL
- Subcommand: **`terraform test`**
- Stored with modules, as a file with a **`.tftest.hcl`** extension

- Tests execute with temporary in-memory state and have no impact on live resources
- Automatic cleanup of test resources

```
# example.tftest.hcl

run "valid_string_concat"  {
  command = plan

  assert {
    condition     = aws_s3_bucket .this.bucket  == "test-bucket"
    error_message  = "Name did not match expected"
  }
}
```

Test Types

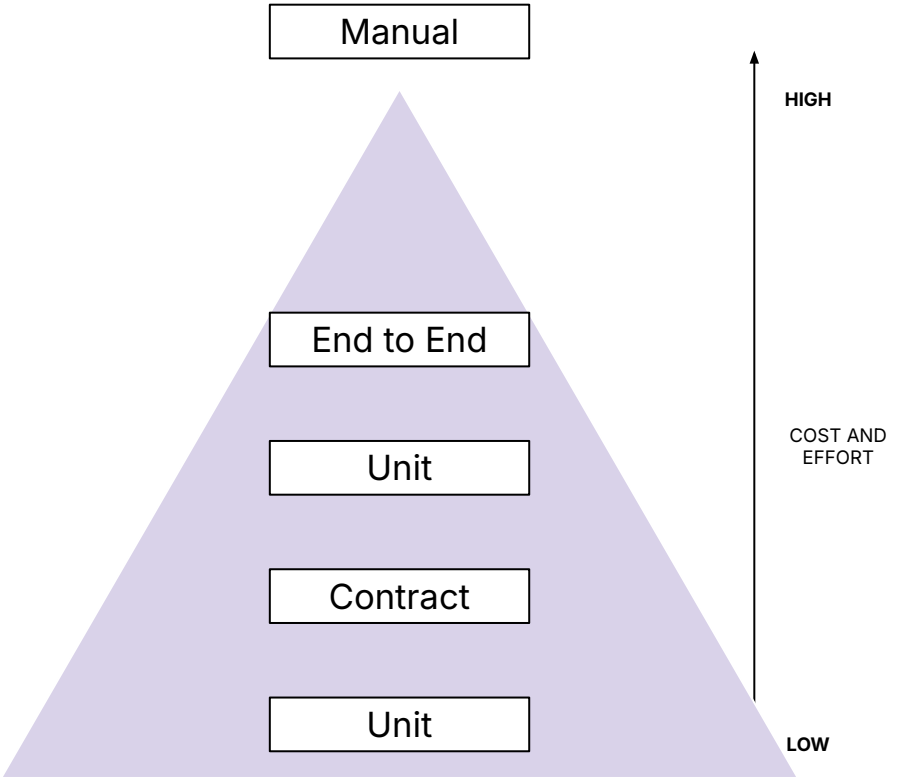
Testing Hierarchy

Branch-based instead of tag-based

Moves the control of publishing from Git tags to the private registry

Greater flexibility for module publishing

Option to switch existing modules from tag-based to branch-based



Unit Testing

Values Known at Plan Time

- Model different combinations of input values
- Are conditionals / logic working as expected?
- Validate string interpolations
- Unintended consequences of refactoring

Things That *Should* Fail

- Add **`expect_failures`** attribute to the run
- Examples:
  - Input variable validations
  - Preconditions
  - Checks

Integration Testing

Values Known *After* Apply

- Verify expected values for calculated attributes & outputs
- Apply-time errors like:
  - Missing permissions
  - Features or APIs not enabled in your account
  - Invalid YAML/JSON in a policy or job spec

Interoperability Testing

- Validate the impact of new:
  - Terraform versions
  - Provider versions
  - Child module versions
- Detect inconsistencies between cloud regions and partitions

Test Mocking

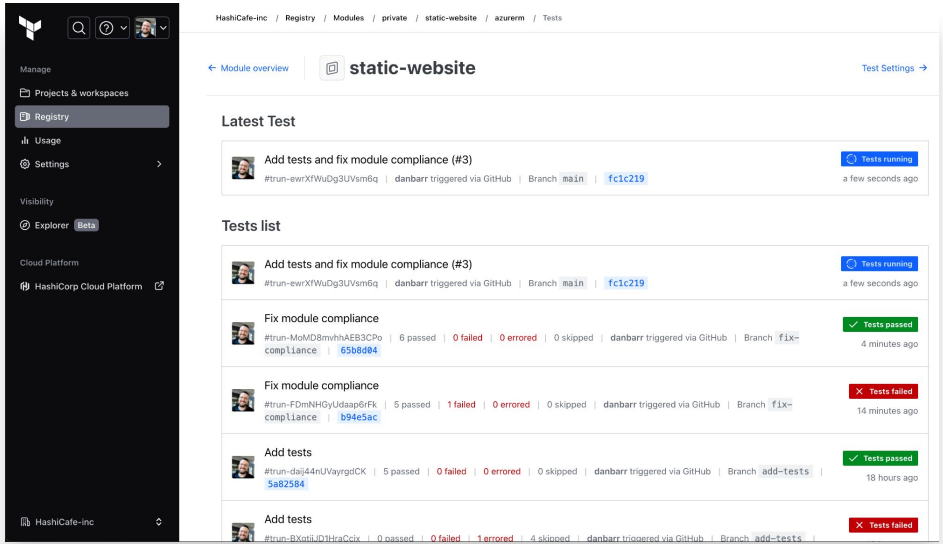
Improved test coverage & flexibility

- Mock provider API responses
  - Random data for calculated attributes
  - Optionally override with specific values
- Run tests without waiting for resource creation and without provider credentials
- Share mocks & overrides across tests with mock data files (`*.tfmock.hcl`)

```
# mocked.tftest.hcl

mock_provider "aws" {}

run "sets_correct_name" {
 variables {
   bucket_name = "my-bucket-name"
 }

 assert {
   condition     = aws_s3_bucket.my_bucket.bucket ==
"my-bucket-name"
   error_message = "incorrect bucket name"
 }
}
```

Test-Integrated Module Publishing

- **Verify modules before publishing**
- Track test results for each commit
- Remote execution of test runs
  - Improve security: module authors don't directly handle credentials
- Initiate tests via CLI or VCS integration
  - VCS: tests automatically run on PRs and commits to the module branch
  - CLI: `terraform test -cloud-run`

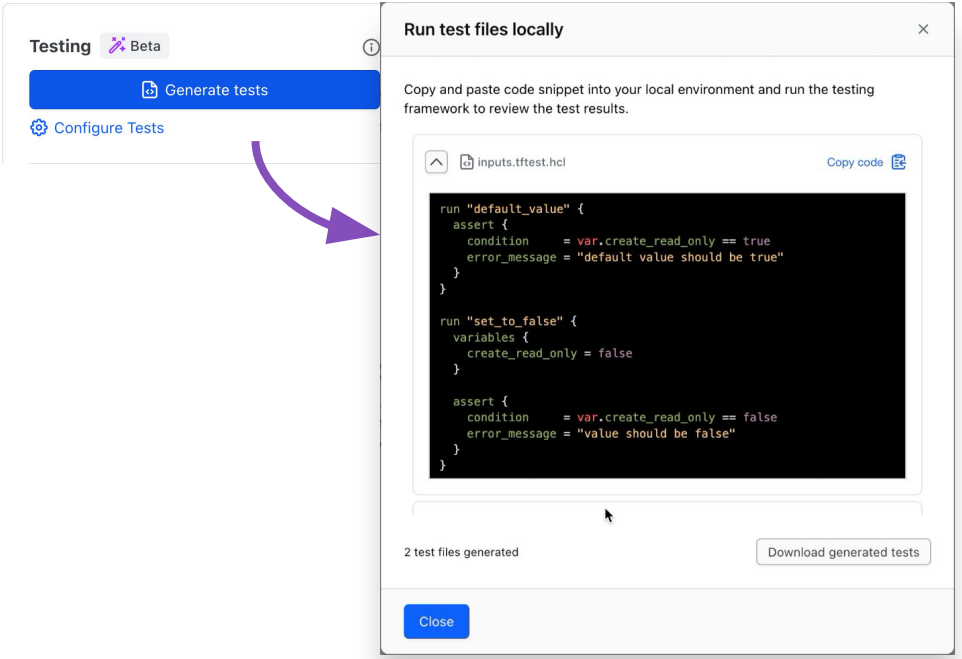


AI-Generated Module Tests

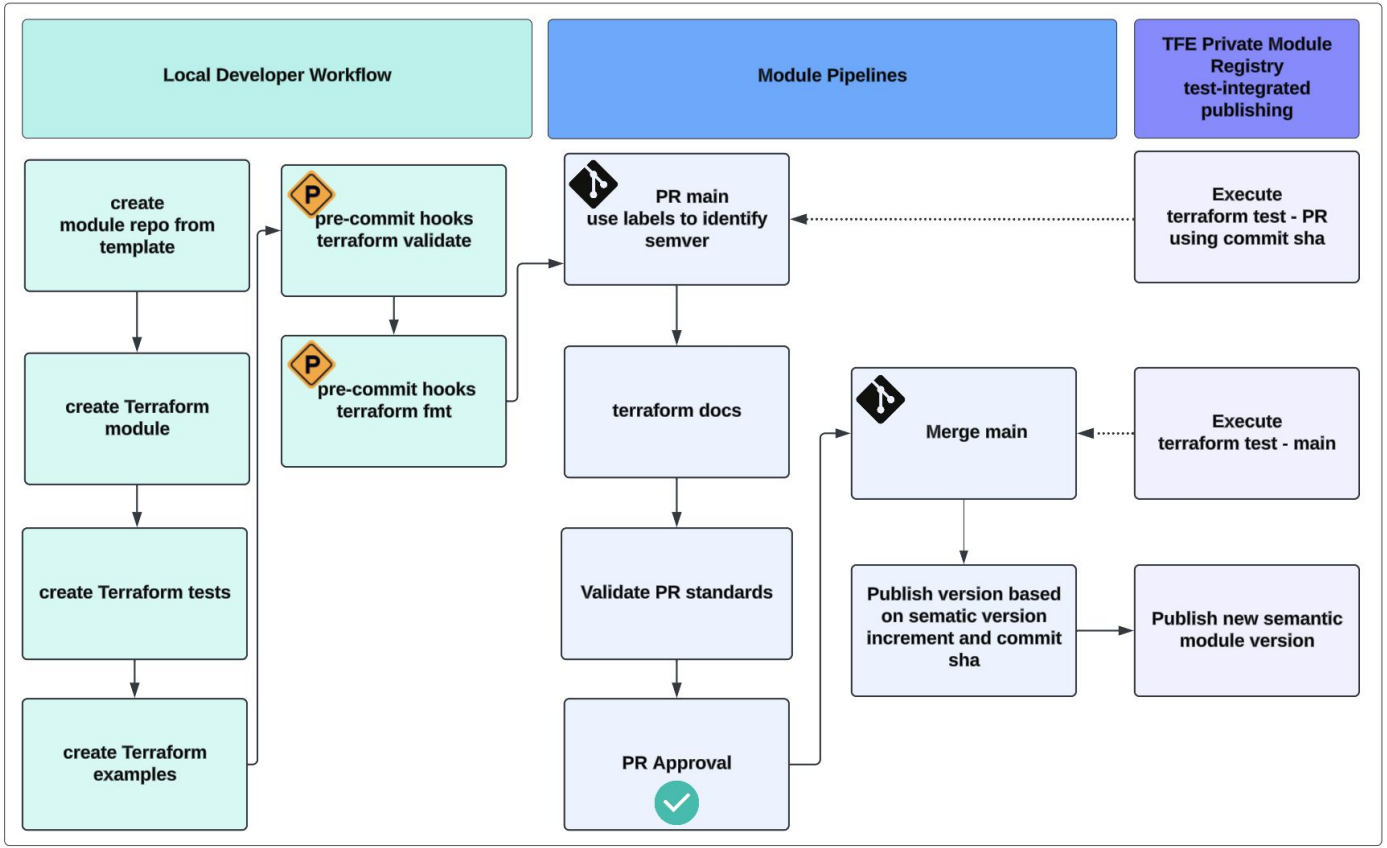
Accelerate test development

Leverages the power of generative AI

- Generate a test suite for any module in your private registry
- Uses a large language model engineered for HCL and the Terraform test framework
- Copy/paste code or download generated test files



Module Publishing Workflow



Terraform test demo

Registry and modules

Service catalogue: curating building blocks constituting large scale deployments

- Standardise the approach to module repository structure and SDLC
- Tagging
- Root module construction
- No-code modules
- Use of Sentinel to provide guardrails for security and compliance
- Discussion points
    - Code communities and significance of Git PR best practice
    - Use of dedicated TFE org for registry module sharing
    - Avoidance of nested child modules and submodules
    - Use of Terraform-docs

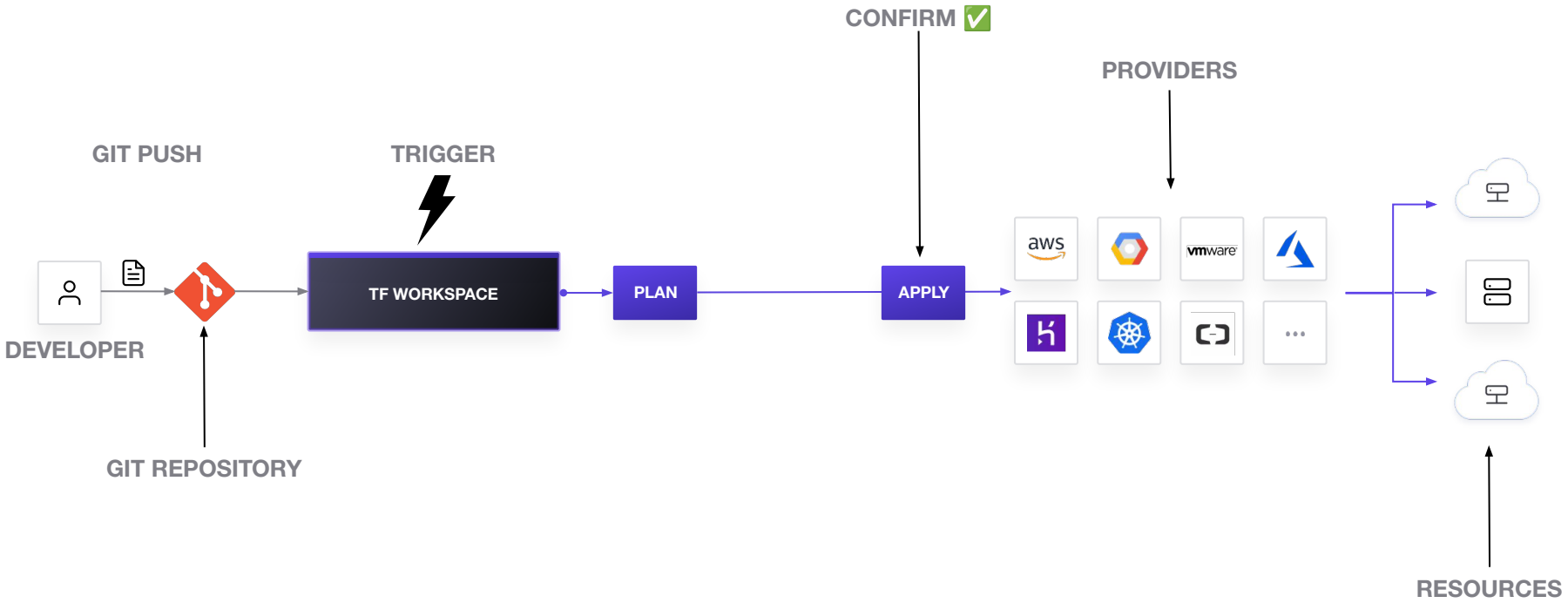

Terraform
HashiCorp
Terraform Cloud: Enforce Policy with Sentinel
Using Policy-as-Code to secure your infrastructure
Copyright © 2022 HashiCorp

Policy-as-Code

- Framework for implementing governance
- Treat policies as applications
- Store policies in version control (VCS)
- Automate policy enforcement & review
- Policies leverage conditional logic

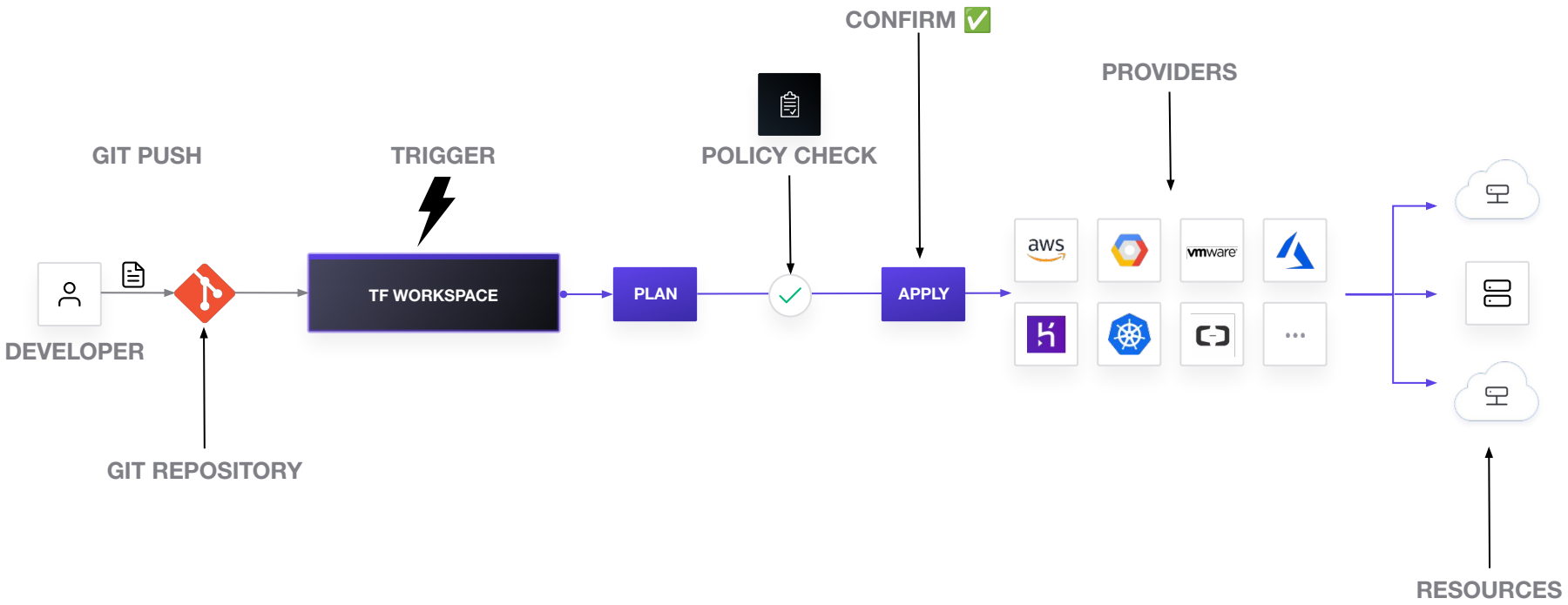
Terraform Workflow

Simple git trigger workflow without policy checks



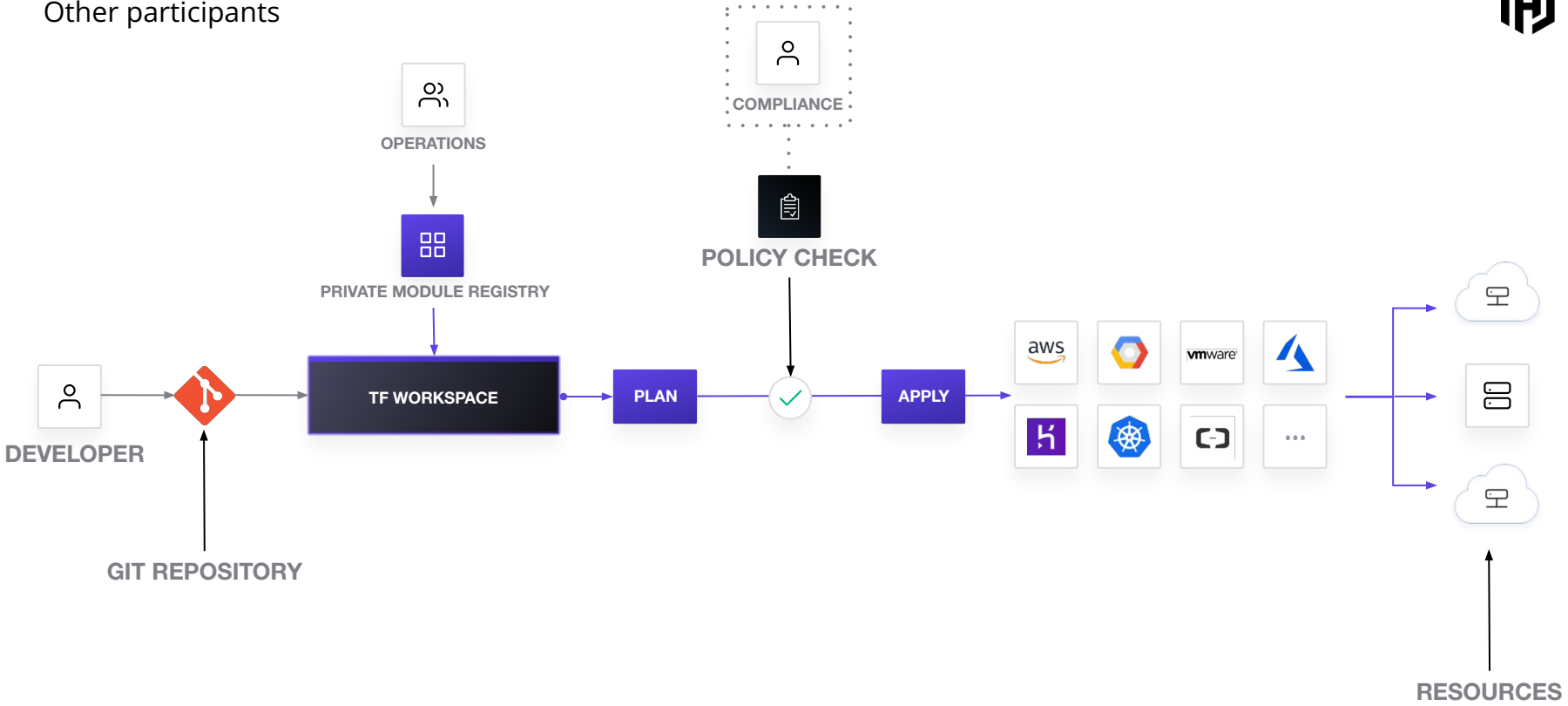
Terraform Workflow

Simple git trigger workflow with policy checks



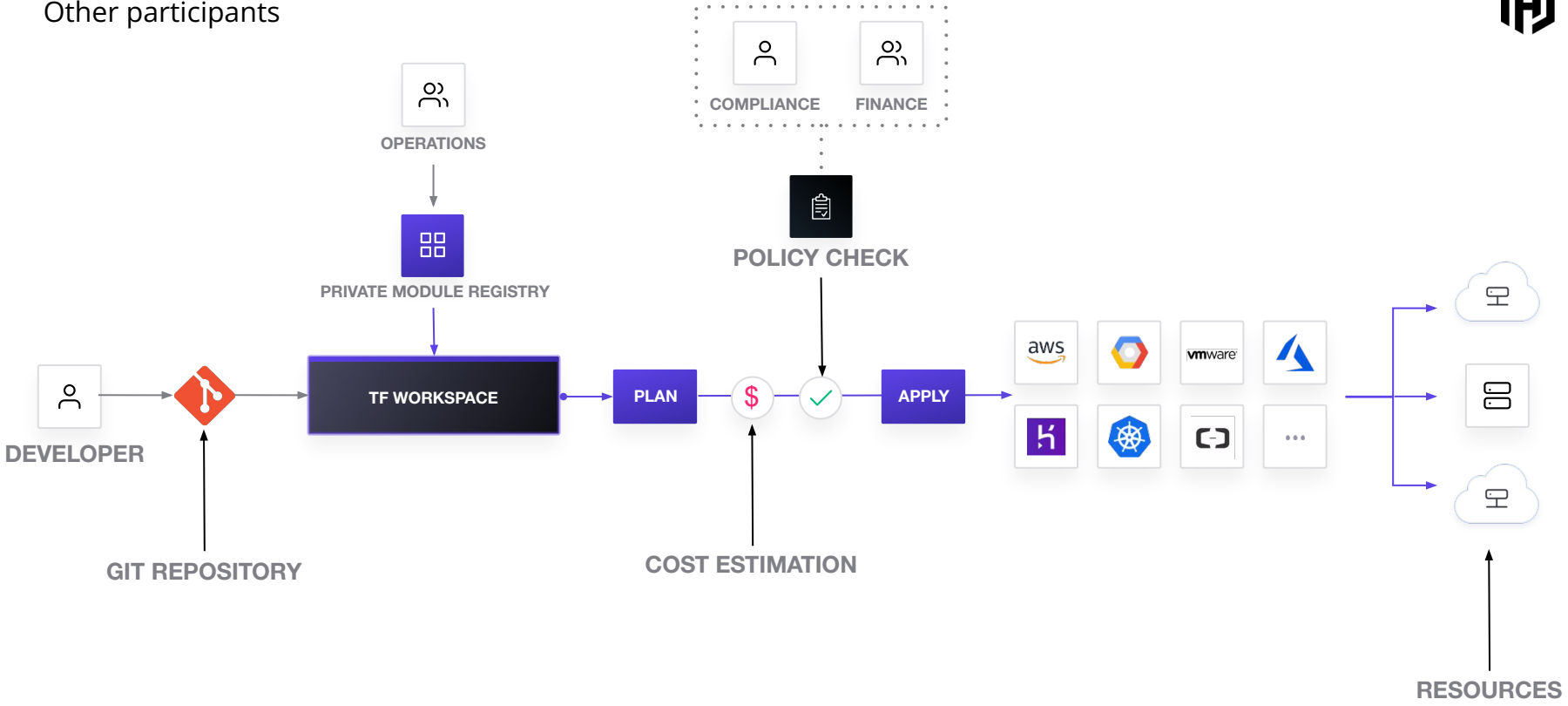
Terraform Workflow

Other participants



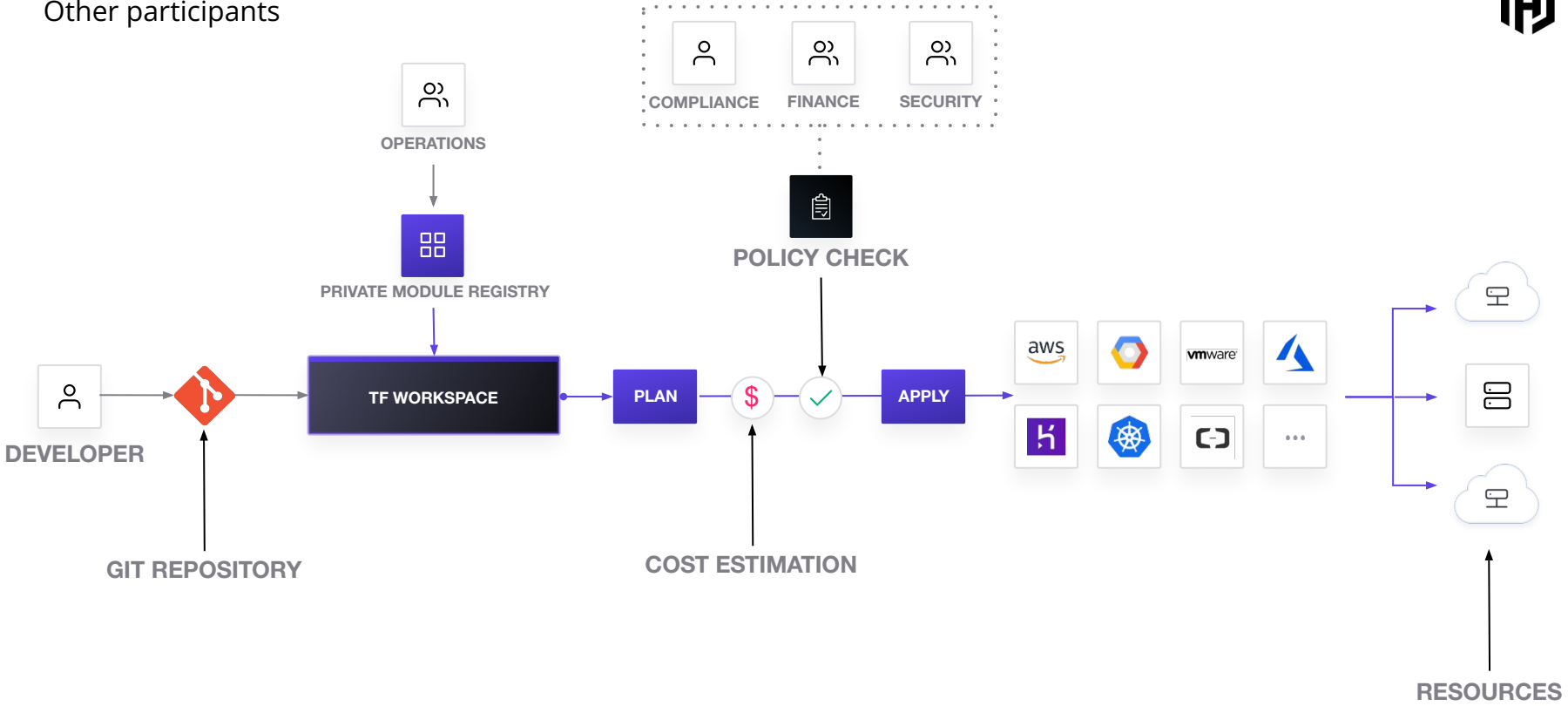
Terraform Workflow

Other participants



Terraform Workflow

Other participants



Security standards

Use cases

- Require all S3 buckets to use the private ACL
- Forbid or allow only certain resources, providers or data sources

Audit Tracking

Use cases

- ***Manage Overrides*** for specific teams
- Review an audit trail for Terraform Cloud operations

Resource Restriction

Use cases

- Limit the size of VMs and clusters for cost
- Enforce mandatory tagging on resources built with Terraform
- Restrict modules to your organizations Private Module Registry

Sentinel demo

2

Deployment environments

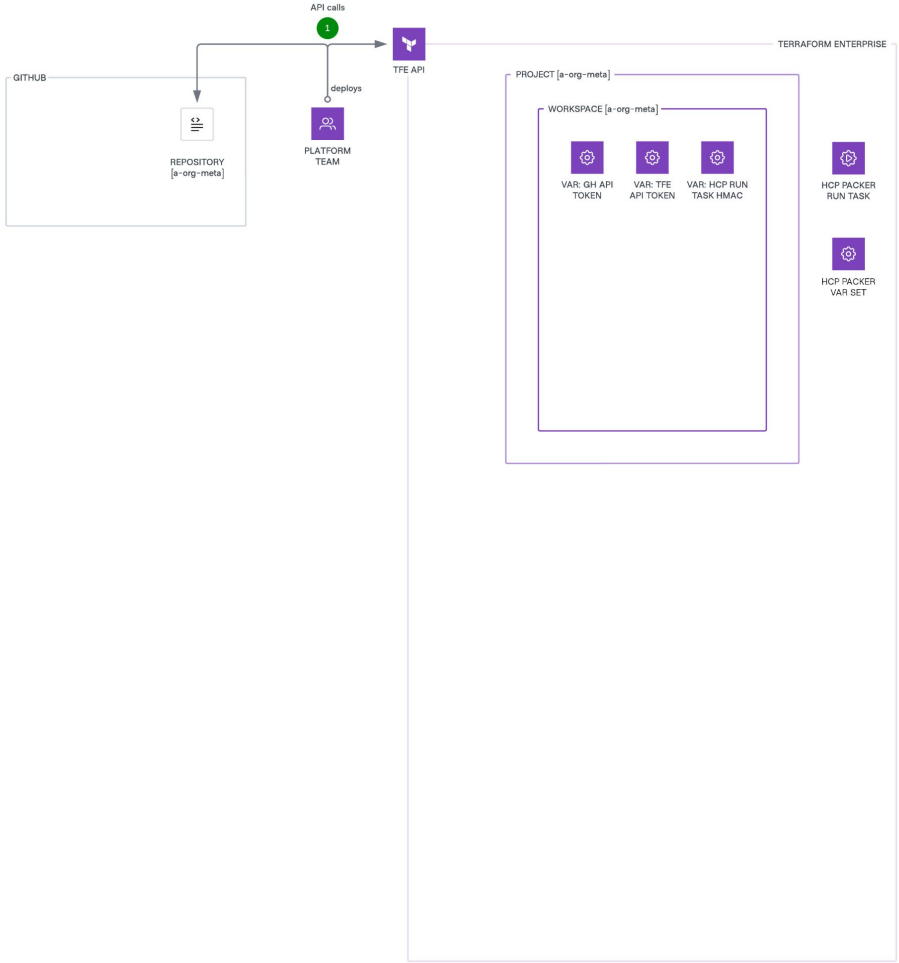
Deployment environments

Landing zones and best practices

- Demo env criteria
- Project planning implications
- TFE organisations and implications for concurrency
- Landing zones
- Traceability

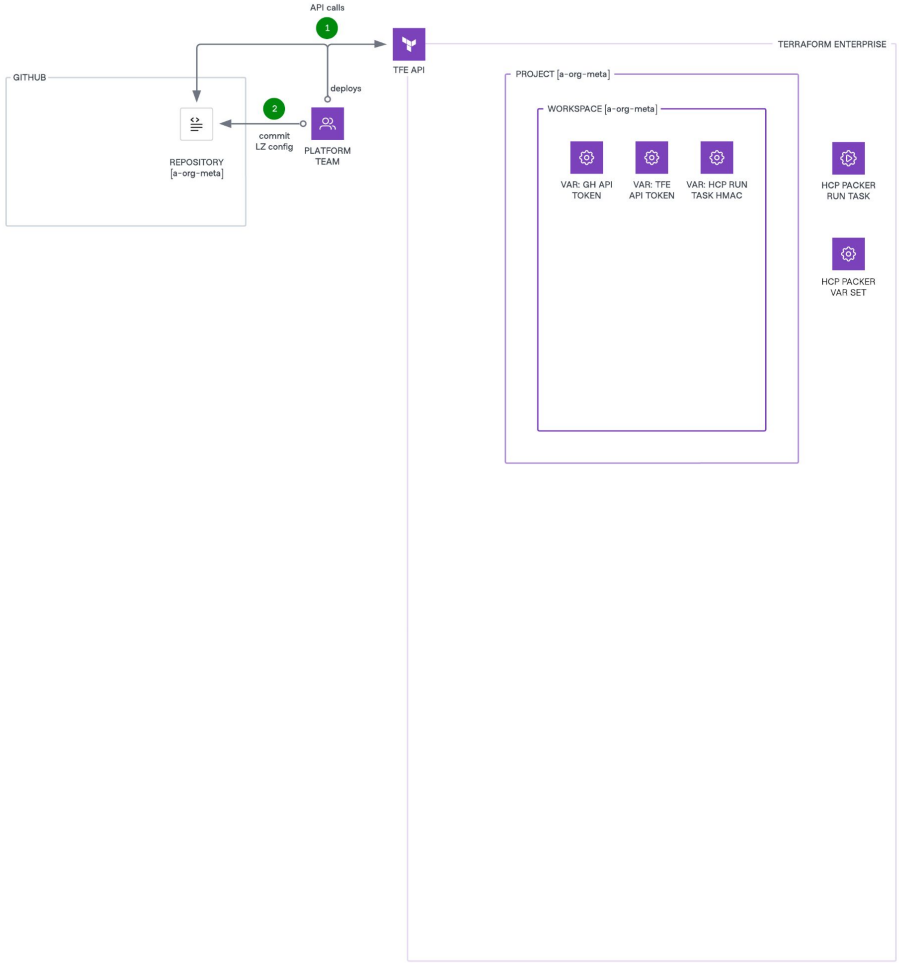
Landing Zone Workflow

1. Initialise



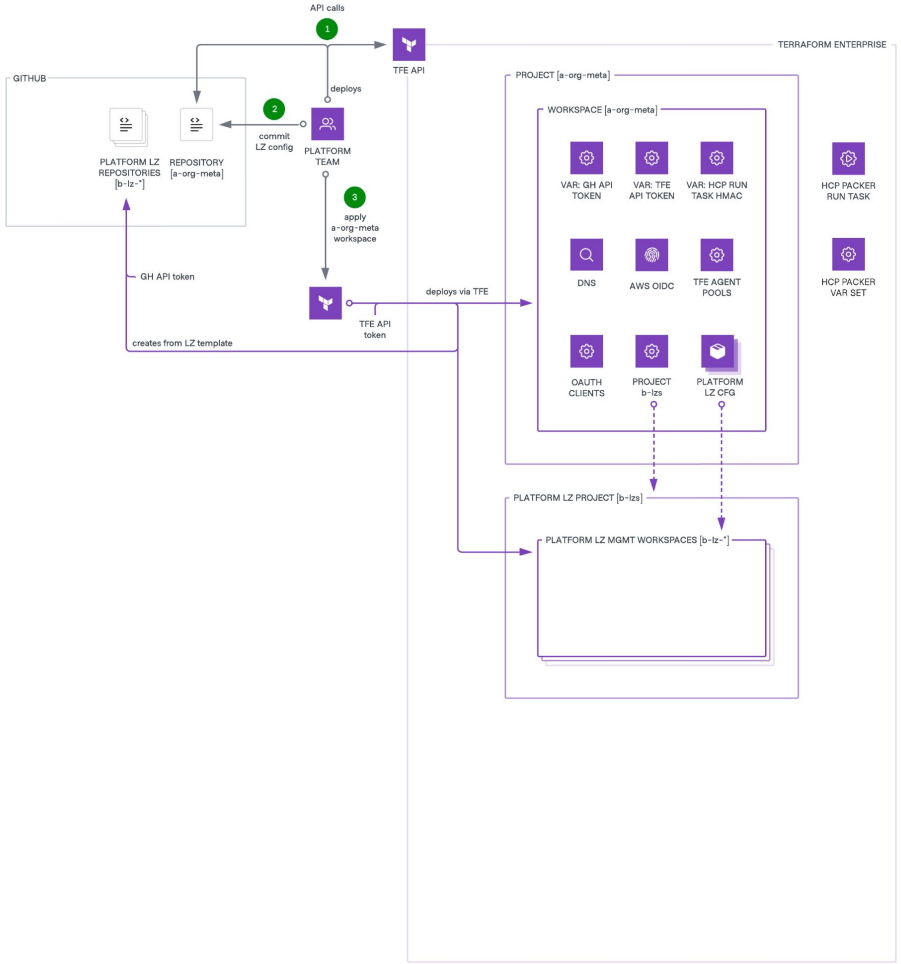
Landing Zone Workflow

1. Initialise
2. Commit IaC to top-level workspace



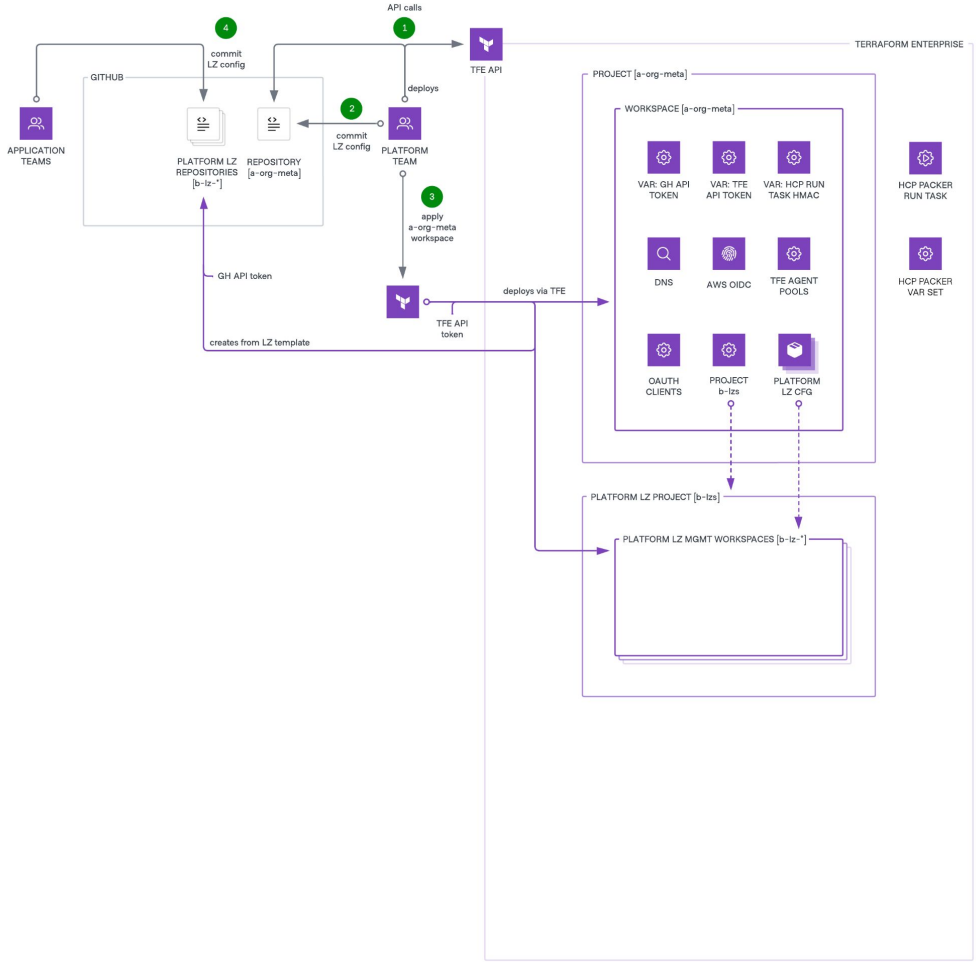
Landing Zone Workflow

1. Initialise
2. Commit IaC to top-level workspace
3. Run top-level workspace



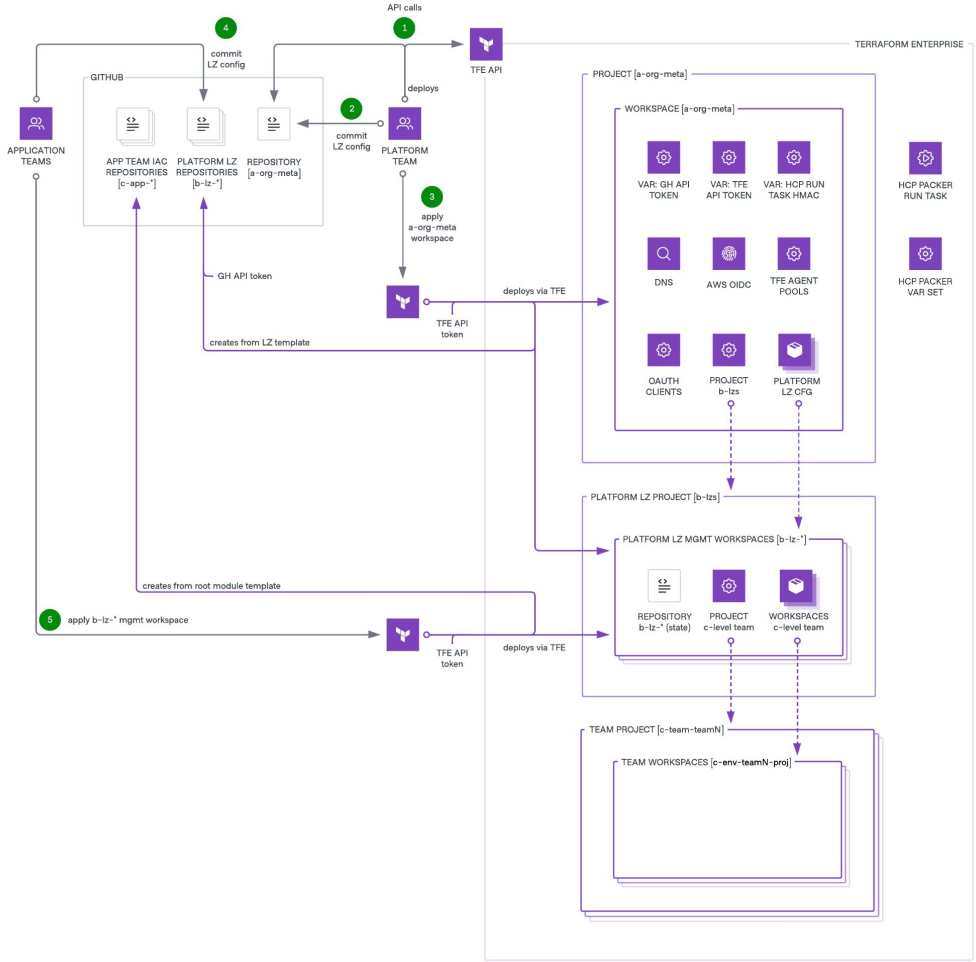
Landing Zone Workflow

1. Initialise
2. Commit IaC to top-level workspace
3. Run top-level workspace
4. Commits IaC to LZ repository



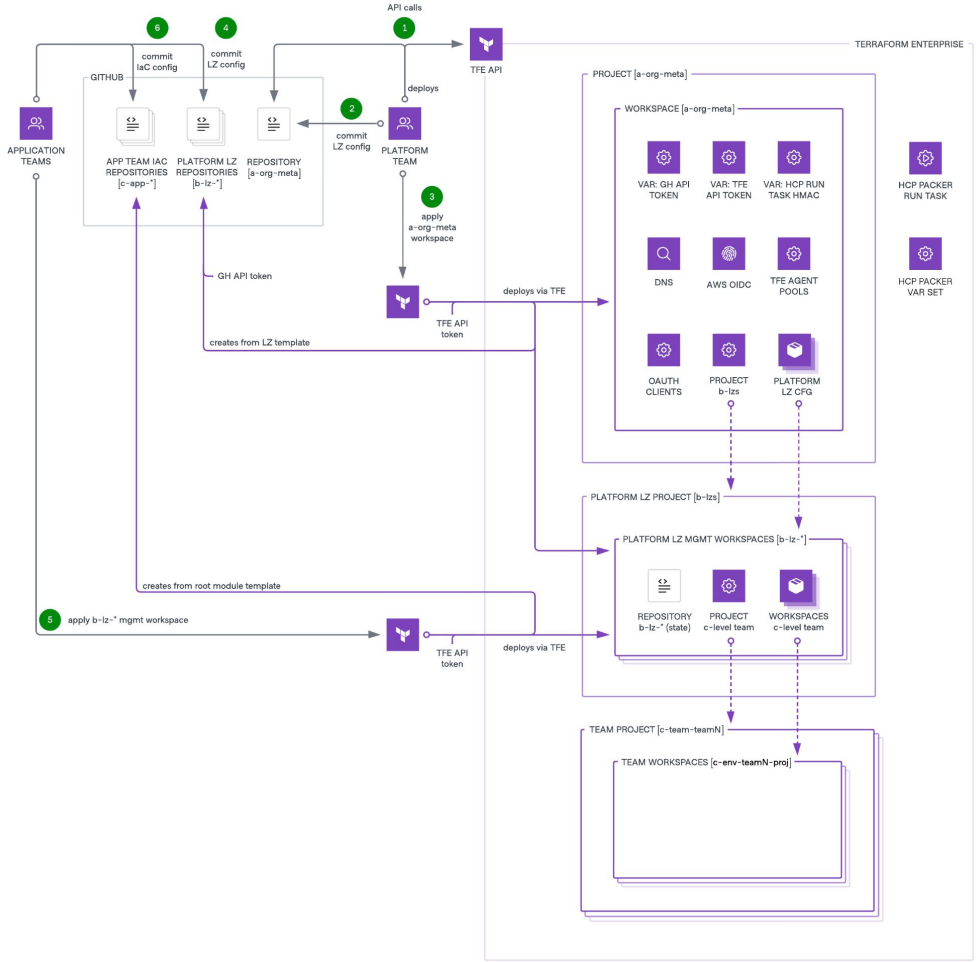
Landing Zone Workflow

1. Initialise
2. Commit IaC to top-level workspace
3. Run top-level workspace
4. Commits IaC to LZ repository
5. Run LZ workspace



Landing Zone Workflow

1. Initialise
2. Commit IaC to top-level workspace
3. Run top-level workspace
4. Commits IaC to LZ repository
5. Run LZ workspace
6. Commit IaC to app team repositories



Landing Zone Workflow

1. Initialise
2. Commit IaC to top-level workspace
3. Run top-level workspace
4. Commits IaC to LZ repository
5. Run LZ workspace
6. Commit IaC to app team repositories
7. Run app team workspaces