HashiCorp Terraform

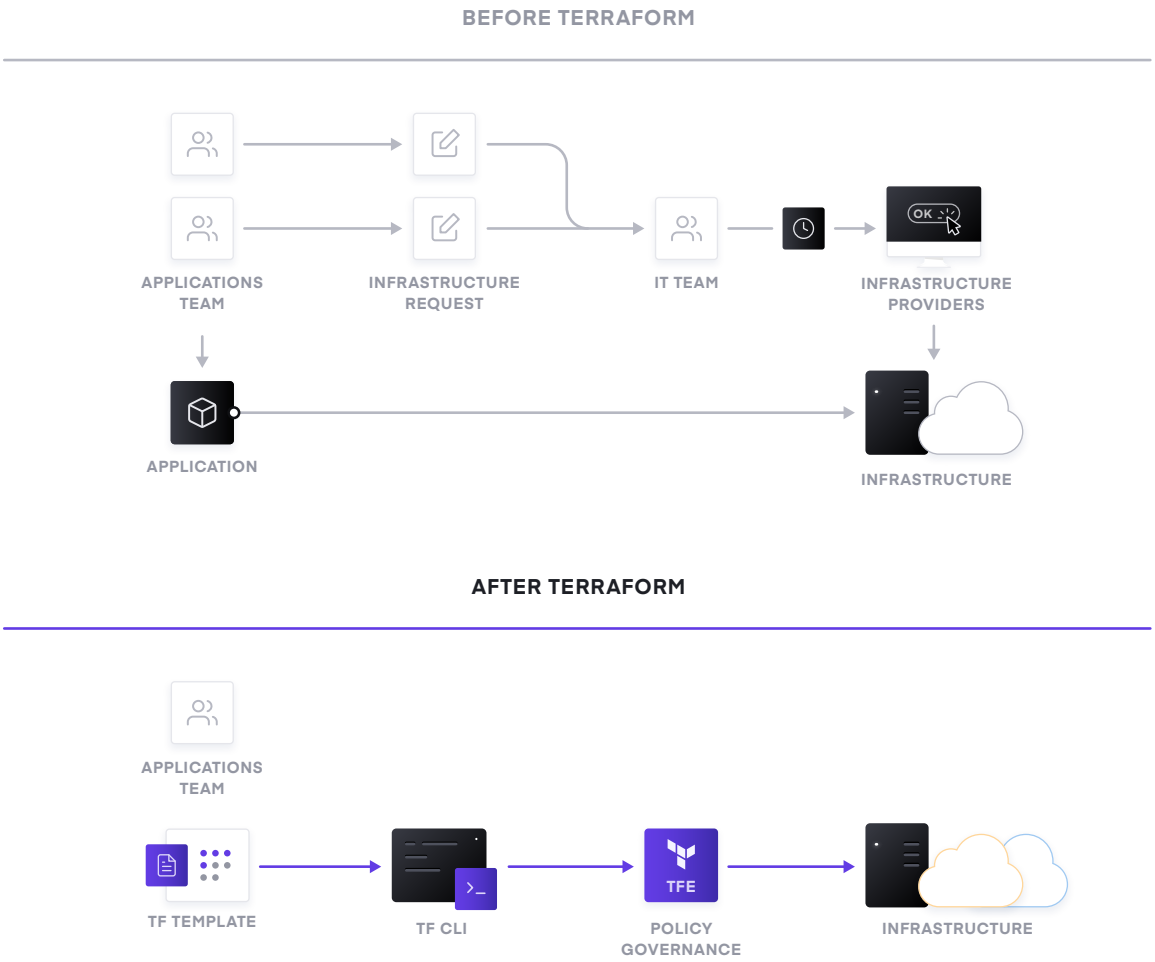
Terraform Operating Model

Contents

Multi-Cloud Infrastructure Provisioning

The foundation for adopting the cloud is infrastructure provisioning. HashiCorp Terraform is the world's most widely used cloud provisioning product and can be used to provision infrastructure for any application using an array of providers for any target platform.

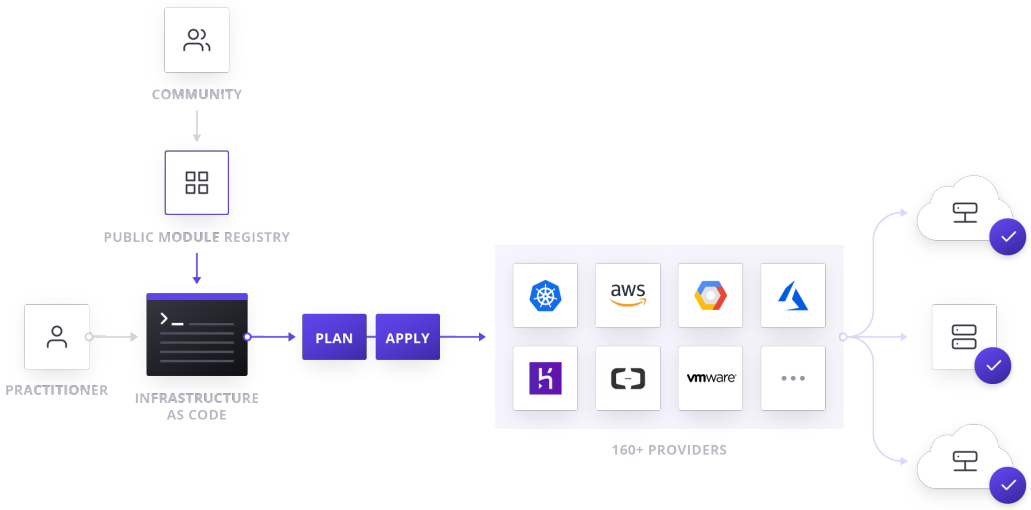
To achieve shared services for infrastructure provisioning, IT teams should start by implementing reproducible infrastructure as code practices, and then layering compliance and governance



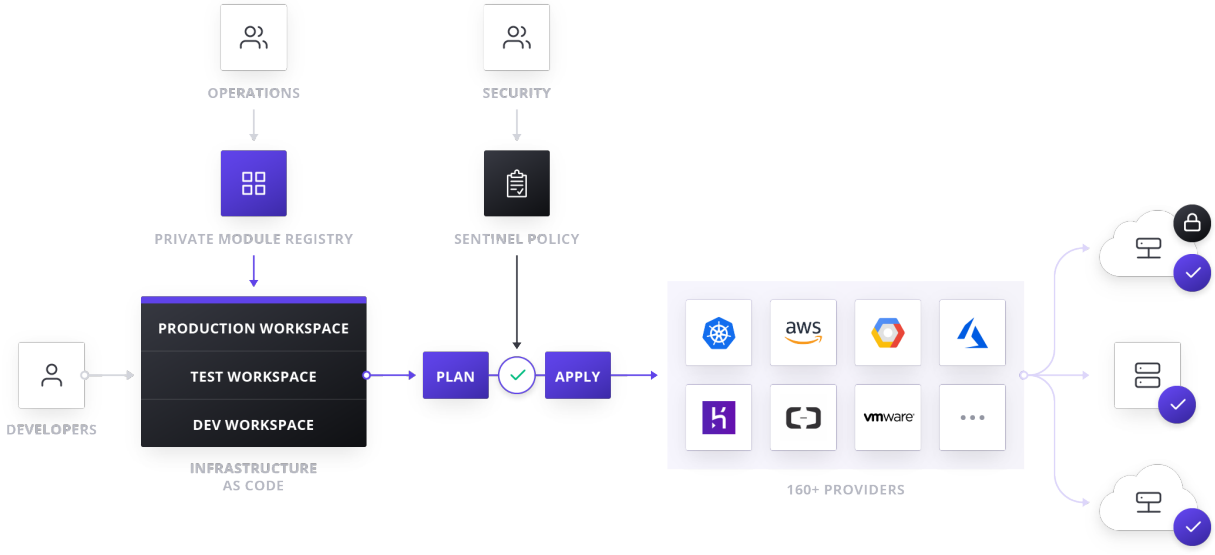
Terraform Enterprise

Terraform Enterprise automates the infrastructure provisioning workflow across an organization. This document outlines how organizations can setup Terraform Enterprise to build complex, interdependent infrastructure using modules and workspaces.

Organizations using Terraform Open Source should be familiar with the following workflow:



This guide will help organizations use the collaboration and governance features of Terraform Enterprise to establish the following workflow:



Goals and Assumptions

The goal of this model is to give organizations a clear path to supporting multiple teams provisioning infrastructure with Terraform Enterprise. Most organizations gradually adopt Terraform, often starting the adoption journey with a small pilot team that proves out and validates the use case. After this pilot, Terraform Enterprise is usually brought under the management of a central IT, operations, or cloud team. Different organizations will have different structures that best suit their needs, but generally the team managing Terraform Enterprise should be infrastructure experts, hereafter this team will be referred to as the infrastructure team. This infrastructure team facilitates and enables the larger organization to provision infrastructure.

Assuming the Terraform pilot is successful, the infrastructure team should have a working understanding of Terraform's features. For reference, here are a few key terms to be familiar with:

- **Infrastructure as Code** is the approach that Terraform uses to build infrastructure. Users define their desired infrastructure in HCL (HashiCorp Configuration Language) and Terraform builds, manages, and tracks the state of that infrastructure.

- **Workspaces** are the unit of isolation in Terraform containing a configuration file, variables, settings, and state. Similar to how monolithic applications are decomposed into microservices, monolithic infrastructure is decomposed into workspaces. Workspaces have separate owners, access controls, and scope. We recommend having one workspace per environment per configuration.

- **Modules** are re-usable infrastructure components. Similar to how libraries are reused across microservices, modules are reused across workspaces.

The next task for the team is to understand the relationships between the teams and applications they support. With that understanding, the infrastructure team can then enable the most secure and efficient infrastructure provisioning for those teams and applications.

Understanding Relationships Between Teams and Applications

A small team (or individual) using Terraform has the benefit of a tight feedback loop between key team members and responsibilities. For example, the same person might be responsible for building the infrastructure, writing application code, and securing that application. In this scenario, operating with a single, monolithic configuration file might be sufficient. However, as organizations build more complex applications running on top of more complex infrastructure, that feedback loop loosens dramatically— networking teams don't have perfect insight into the needs of development teams, security teams want to slow operations for auditing purposes, while operations teams want to provide speed and efficiency— motivating the need for the infrastructure team to create boundaries between components to harden the overall topology against breaks and allow work in parallel.

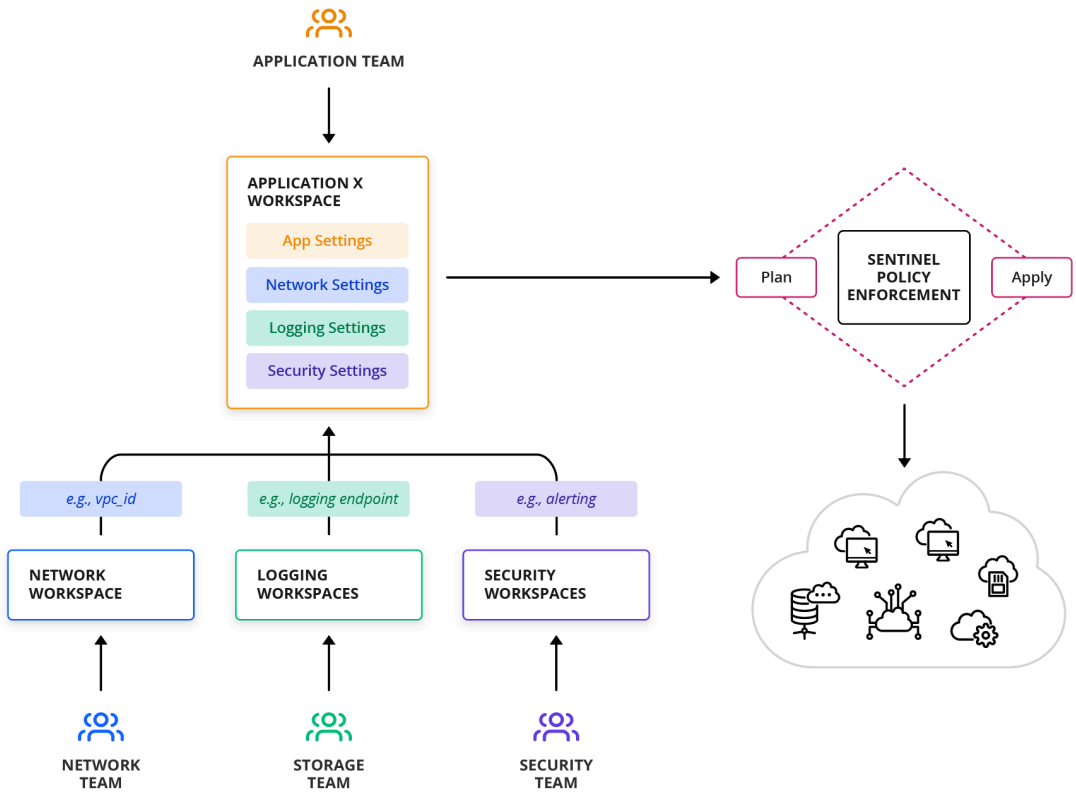
For the infrastructure team managing Terraform, it is important to understand relationships between teams and applications. They should work with teams to understand communication channels between teams, interdependencies between infrastructure components (such as network dependencies), and interdependencies between the services running on top of the infrastructure.

Other important considerations include both the rate of change of these components and the scale of those changes. Slower changing components such as VPC's can be treated differently than more rapidly changing components like compute instances. Likewise, by qualifying releases (major and minor at the least), teams can establish processes accordingly.

Determining Workspace Structure

After understanding the relationships between teams and applications, the infrastructure team should begin decomposing existing infrastructure to match those relationships and build a framework and process for future changes and additions. The infrastructure team should leverage workspaces to decompose infrastructure to allow teams to efficiently work in parallel, while maintaining proper security posture.

An example topology might include many applications and services existing in respective workspaces with appropriate access controls in place.



For example, organizations will have a networking workspace managed by the networking team the outputs of which can be consumed by components, services, and applications to be deployed on the network. The networking workspace should have write access for the networking team, but limited permissions for other teams[1]. Middleware such as security, logging, and monitoring tooling to be employed by many applications should each get respective workspaces, with “write”

1 Depending on the complexity of the network, subcomponents could have respective teams and workspaces as well. This is true for any infrastructure component as workspaces should provide logical, helpful decomposition where possible.

access available to the teams managing those services and “read” access for teams consuming those resources. This allows applications teams to quickly make use of these services without special knowledge or set up. Likewise, additional applications should get their own respective workspaces with similarly appropriate access control.

To make changes to components the infrastructure team should work with respective workspace owners to version and iterate infrastructure as code making use of a version control system. In some cases, the infrastructure team will work with component/application teams to write infrastructure as code to define needed components. In other cases, the Infrastructure team will vet and review infrastructure as code configuration files written by the respective component/application team.

It’s also important to workout releases amongst teams: minor releases might be isolated in impact and not require coordination with other workspaces or teams, while major releases might require informing and aligning with affected teams. For example, a minor change might create a new input for a configuration to allow more specific configuration, while providing the a default value to avoid a breaking change. A major change might be a refactor that results in different outputs which results in existing workflow failing if the changes aren’t planned for.

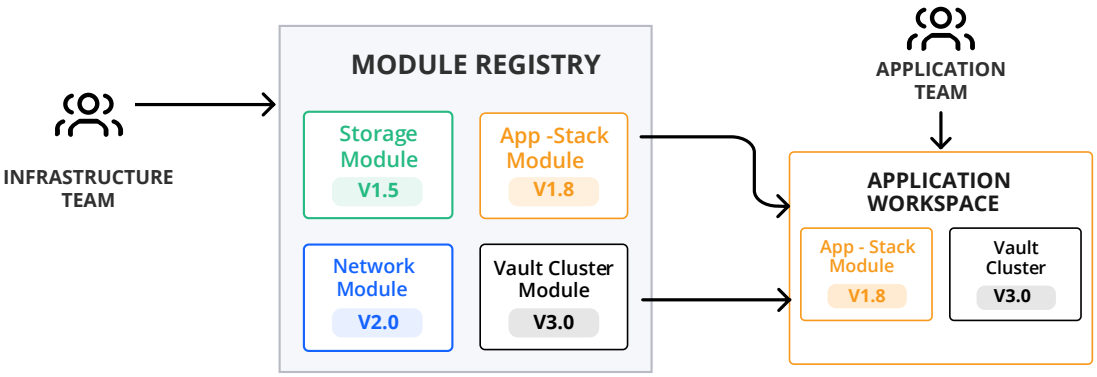
The infrastructure team should work with a security team (CIO, CISO, etc.) to ensure workspaces have proper access permissions and to create policy as code governance for infrastructure. The security team could write and implement their own policy as code, or work with the infrastructure team to develop and enforce policies. For example, many regulations require sensitive handling of data and where it is stored. To meet that requirement the security team could write a policy that prevents building infrastructure outside specified availability zones.

```
# Allowed availability zones
allowed_zones = [
  “us-east-1a”,
  “us-east-1b”,
  “us-east-1c”,
  “us-east-1d”,
  “us-east-1e”,
  “us-east-1f”,
]

# Rule to restrict availability zones and region
region_allowed = rule {
    all aws_instances as _, instances {
      all instances as index, r {
         r.applied.availability_zone in allowed_zones
      }
    }
}

# Main rule that requires other rules to be true
main = rule {
  (region_allowed) else true
}
```

This rule restricts AWS deployments to us-east availability zones.

Using Modules for Self-Service Infrastructure

Some organizations seek to establish self-service infrastructure for teams building applications. In this model the infrastructure team can develop modules of common infrastructure components that application teams can rapidly customize and deploy.



Many application teams will have knowledge and preference about the infrastructure they're deploying applications to, however, not every application team will have this knowledge. Terraform allows these types of teams to still rapidly deploy infrastructure with the module registry and configuration designer. Terraform Enterprise users have access to both the public registry filled with community built modules and a private module registry, where proprietary modules can be securely stored and shared internally. Teams with no infrastructure as code experience can select the modules they would like to use, customize them in configuration designer, and receive infrastructure as code to provision that infrastructure.

Conclusion

As companies continue to build complex applications on top of complex infrastructure, teams tasked with managing that infrastructure and facilitating it's provisioning can use Terraform Enterprise to decompose, automate, and secure that infrastructure. By understanding relationships and dependencies the infrastructure team empowers application and service teams to work in parallel developing and iterating infrastructure components (and respective applications) while isolating and minimizing impact from unexpected errors and bugs. Likewise, the infrastructure team can enable the larger organization to provision infrastructure in a self-service fashion through the module registry and configuration designer.


HashiCorp