TFE Technical Enablement

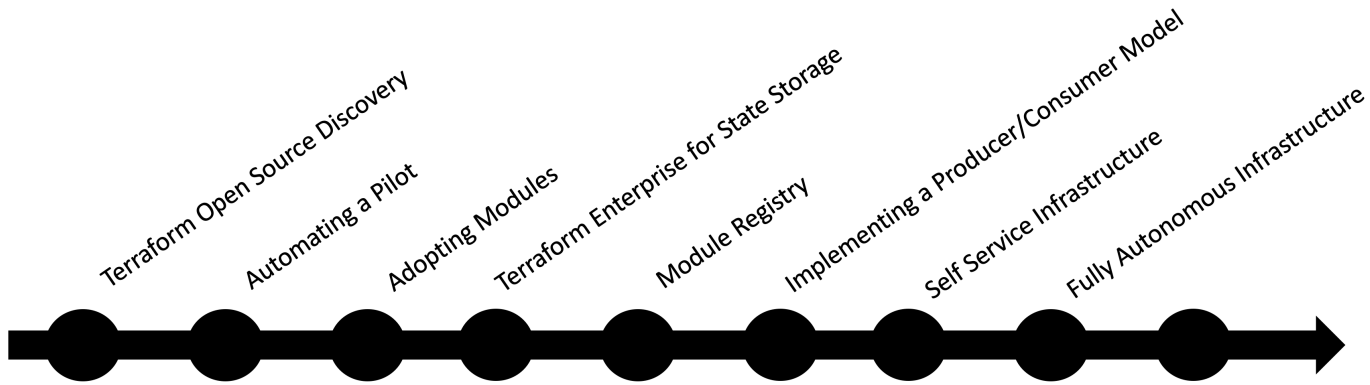
TFE Adoption and Self-Service

HashiCorp

Table of Contents

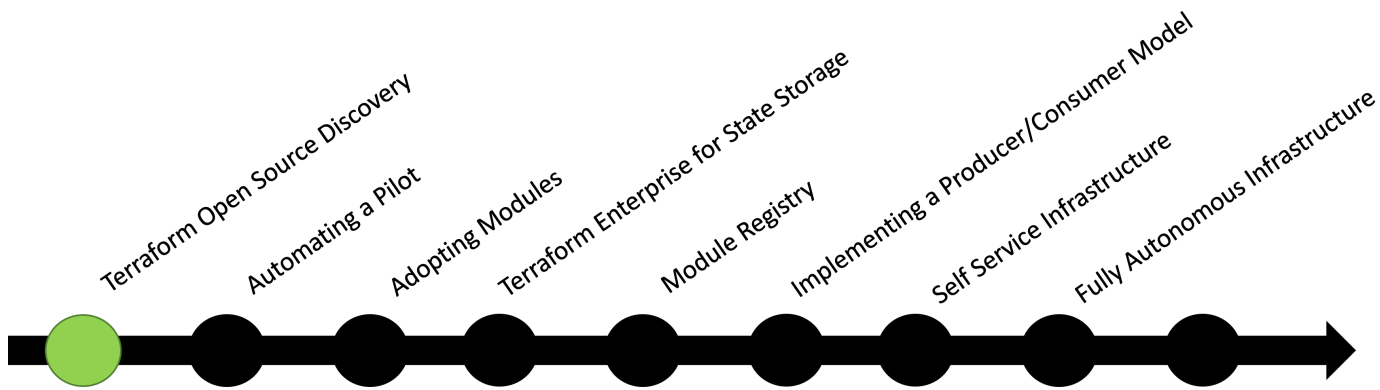
Terraform Adoption Milestones

As individuals, teams and organizations adopt Terraform Enterprise there are some predictable milestones that you may encounter.



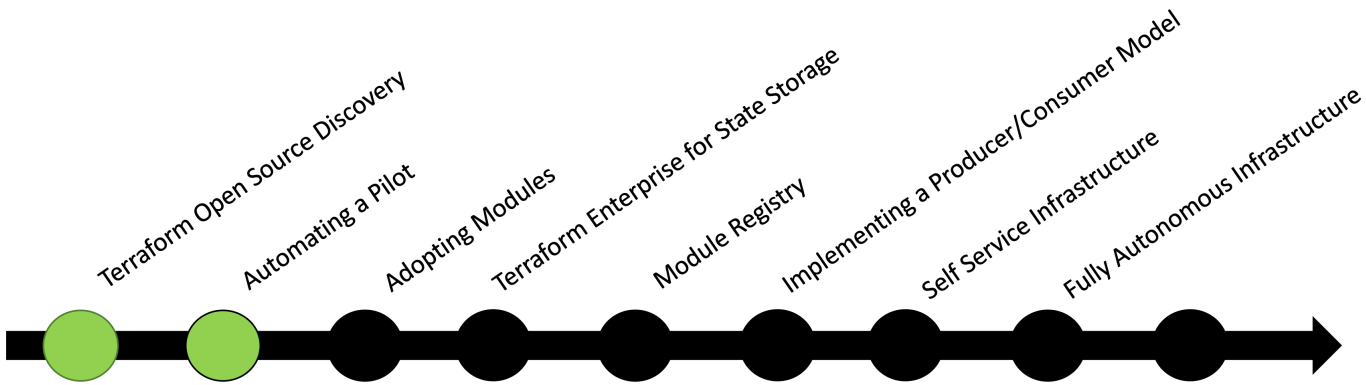
Terraform Adoption Milestones

The first milestone is discovering the value of Infrastructure As Code (IAC) and using a universal provisioning language. This leads individuals to Terraform Open Source.



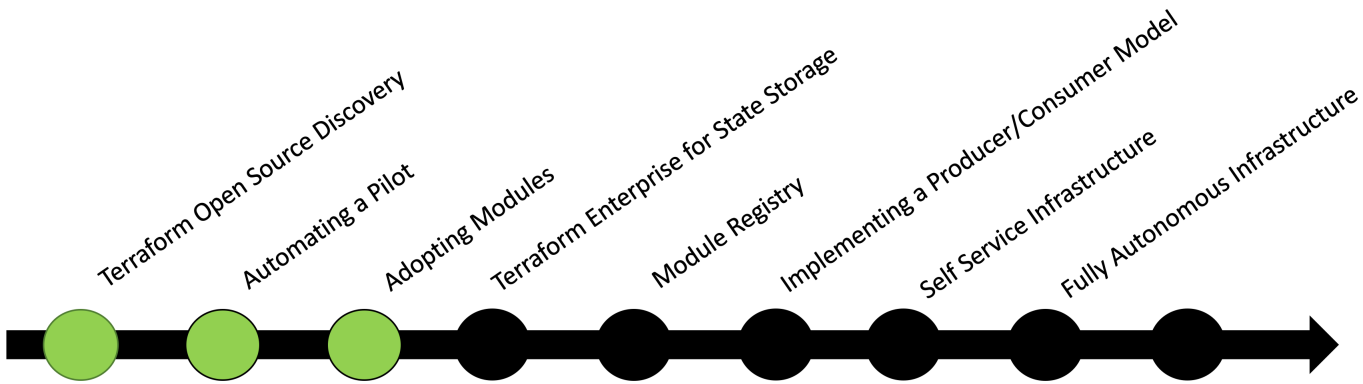
Terraform Adoption Milestones

Consumers typically evaluate Terraform Open Source against an existing pilot set of infrastructure, or as their first foray into a public cloud environment.



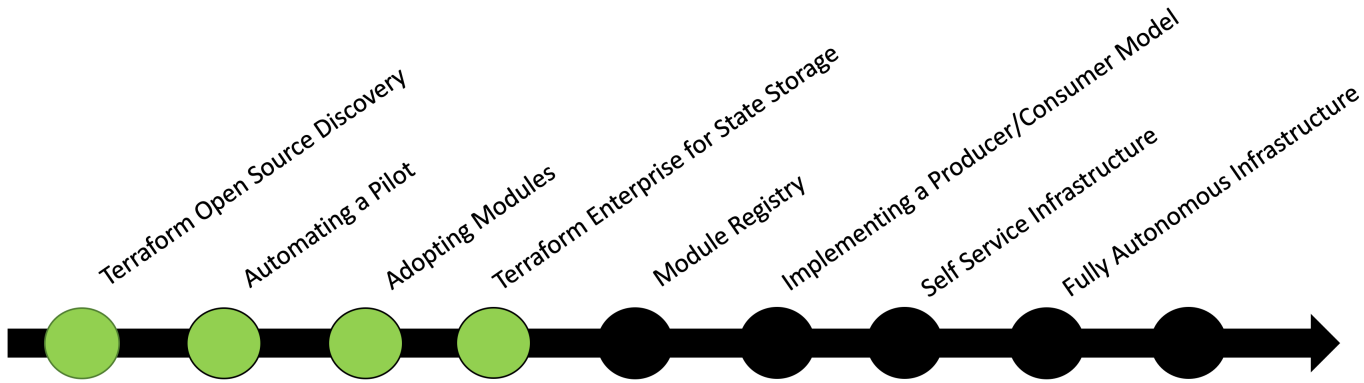
Terraform Adoption Milestones

Once initial testing of Terraform Open Source is completed, the next goal is to start maturing the usage and adoption of standards. This results in the consumption of modules.



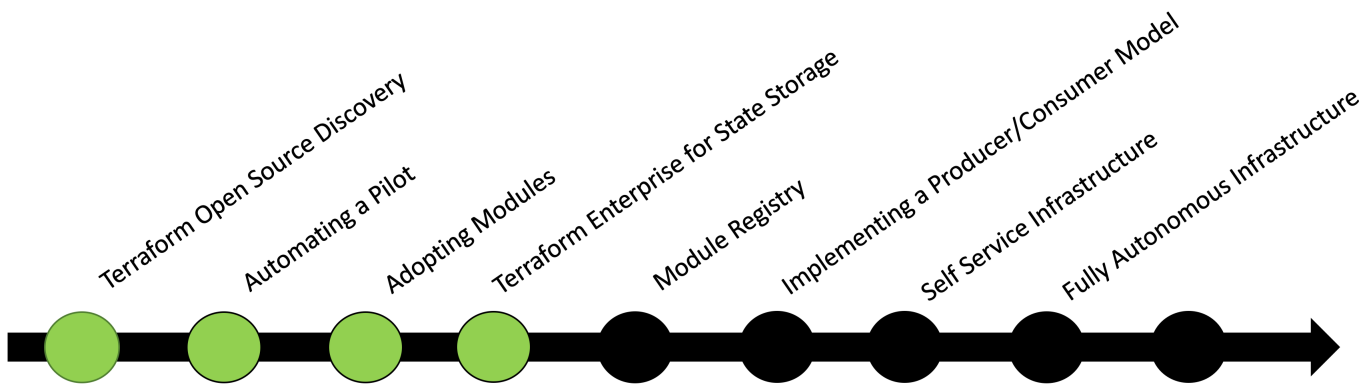
Terraform Adoption Milestones

At this point, after several users in org. have adopted Terraform Open Source, the consumption curve and utilization of Terraform in an organization typically leads to the exploration of our enterprise offering both for features and support.



Terraform Adoption Milestones

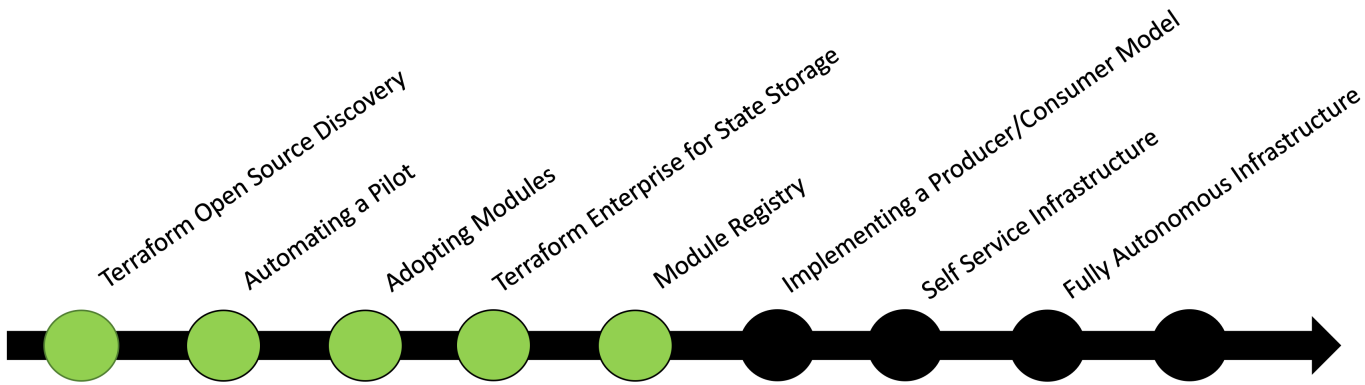
One of the most common first features to consume is remote state storage.



Terraform Adoption Milestones

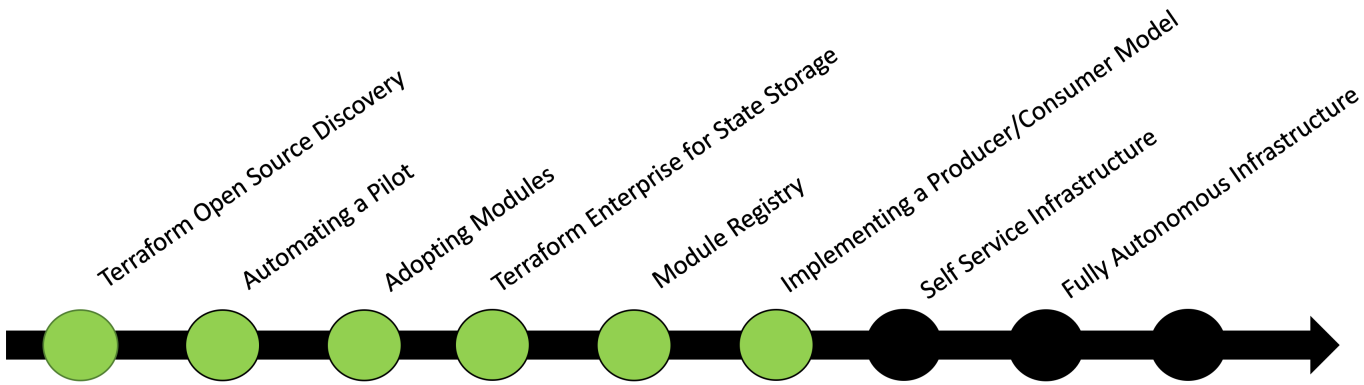
Increasing operator productivity and developer agility via **self-service** is a key driver for enterprise wide Terraform adoption.

The first building block of this is the module registry.



Terraform Adoption Milestones

Once the module registry is populated organizations typically follow a producer and consumer model. The producers are the core Terraform code creation team, and anyone else who needs infrastructure becomes a consumer in some form.



Interlude

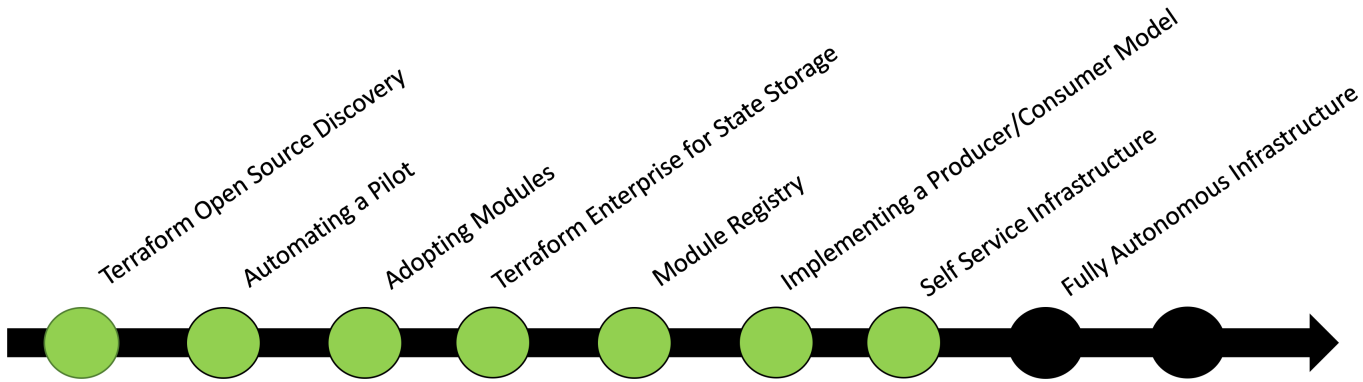
Consumer of Terraform is a broad term, there are multiple ways consumers could interface with Terraform Enterprise such as;

- Writing and contributing their own code and modules for review by a centralized team
- Writing "wrapping code" to consume existing modules and infrastructure from a another team
- Going through a self service process such as ServiceNow (discussed later in this chapter)

It all depends on the skillset of the consumer and level of interaction and customization required.

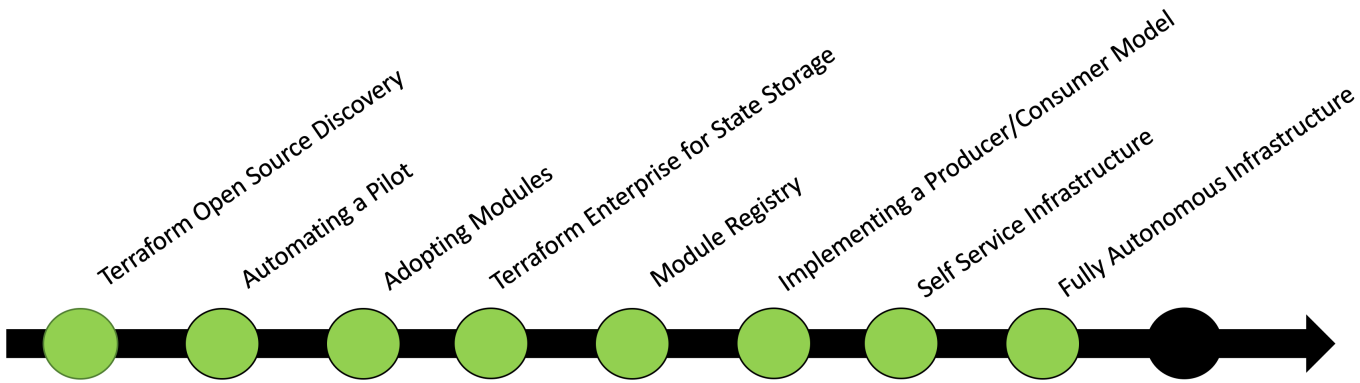
Terraform Adoption Milestones

Enabling the producer and consumer model allows for self service infrastructure with little to no oversight from a centralized provisioning team.



Terraform Adoption Milestones

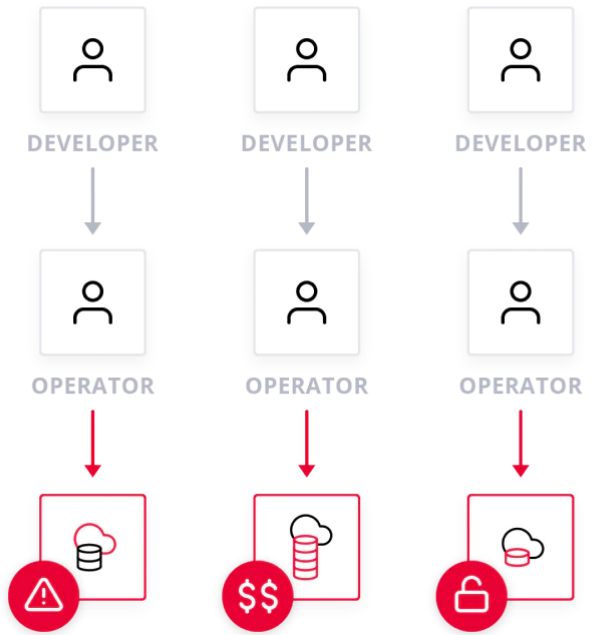
An end state goal would be fully autonomous infrastructure that requires little to no input from consumers and is ready "at the press of button". Most organizations provide this via a combination of tickets, PaaS services and Operator effort.



The Challenge

Developers have to wait for operators to provision and assign dedicated infrastructure.

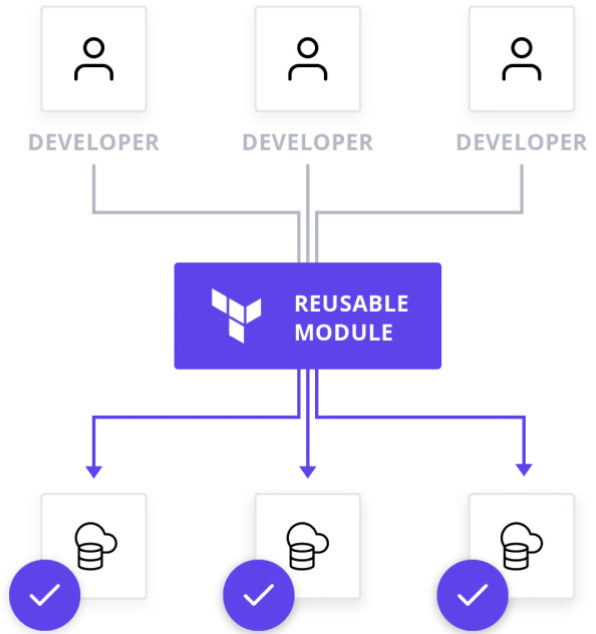
- **Increased costs** from over-provisioning and unused/orphaned infrastructure
- **Reduced productivity** from Developers waiting on operations to provision/scale infrastructure
- **Increased risk** from many users provisioning infrastructure and manual, error-prone processes



The Solution

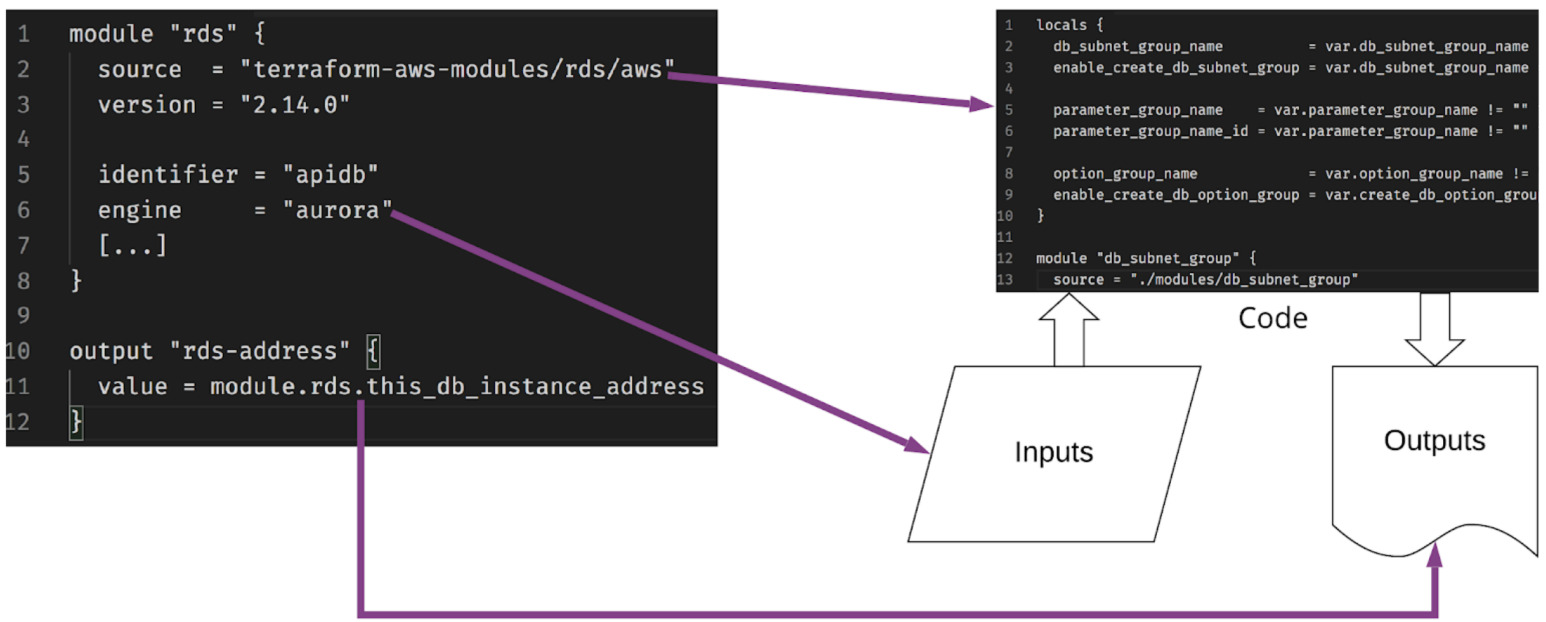
Library of versioned and validated infrastructure templates to be consumed for on-demand provisioning.

- **Increase developer agility** by allowing them to provision their own self-service infrastructure without an operator bottleneck
- **Increase operator productivity** by allowing them to serve more infrastructure request with predefined modules
- **Reduce risk** with a single workflow to secure, govern, and audit regardless who provisions



Producer View

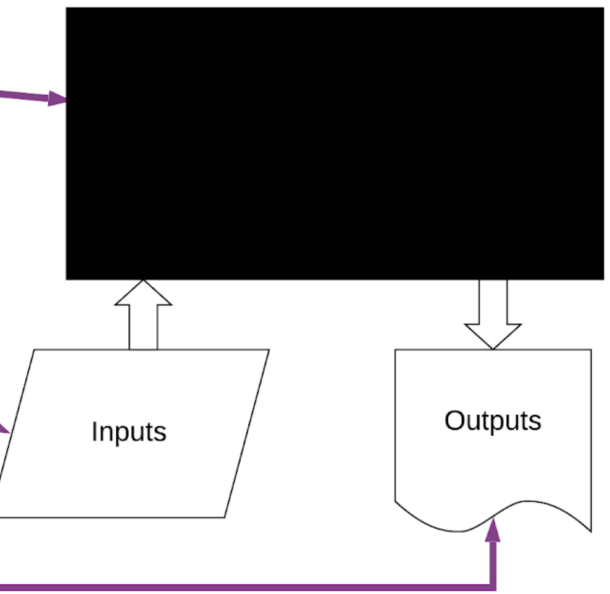
The goal of a collection of modules and a self service portal is to provide a simple process or "menu" for customers to "order" infrastructure. Producers decide what options are offered and how they are handled internally.



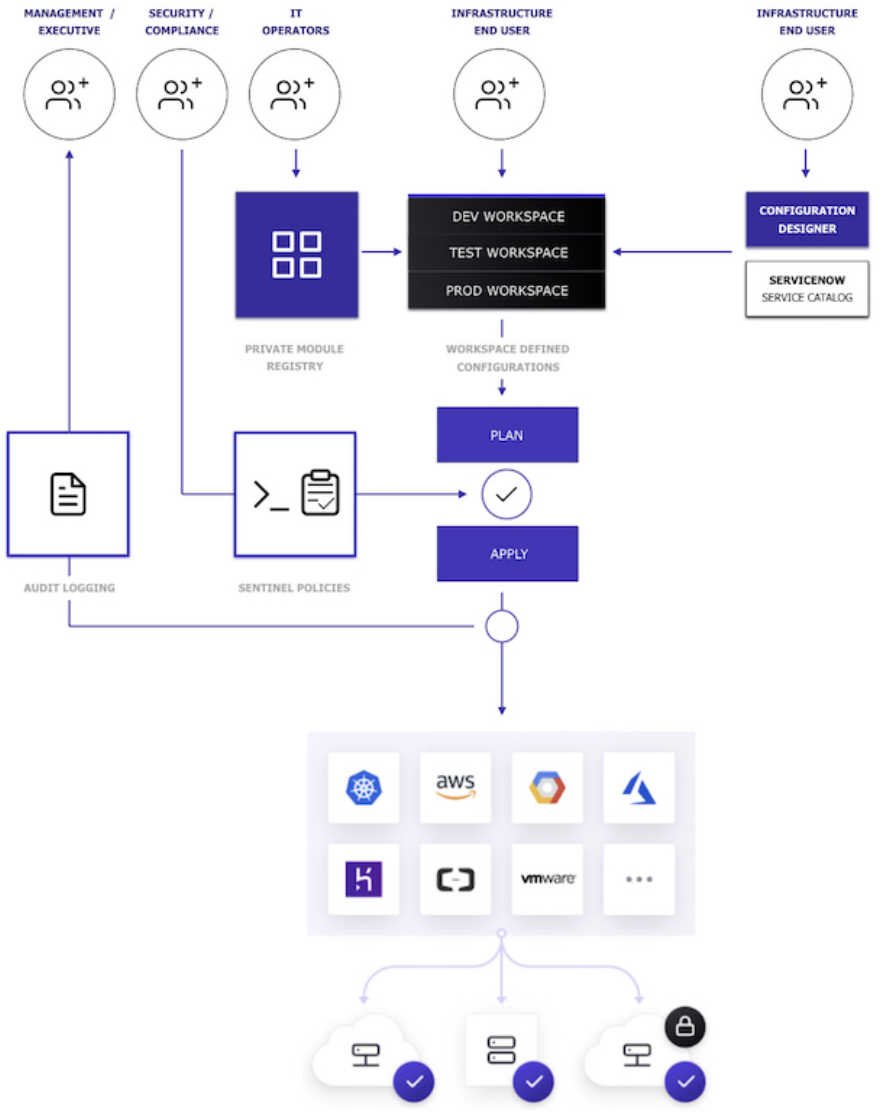
Consumer View

Consumers "order" infrastructure. They specify the inputs but need no visibility into the code.

```
1  module "rds" {
2    source  = "terraform-aws-modules/rds/aws"
3    version = "2.14.0"
4
5    identifier = "apidb"
6    engine     = "aurora"
7    [...]
8  }
9
10 output "rds-address" {
11   value = module.rds.this_db_instance_address
12 }
```



Terraform Cloud/Enterprise Workflow



TFE Technical Enablement

Self-service Workflows

HashiCorp

Service Now Integration

Terraform Enterprise offers organizations an infrastructure as code approach to multi-cloud provisioning, compliance, and management.

The integration with ServiceNow extends this capability so that any end-user can request infrastructure from the ServiceNow Service Catalog and Terraform Enterprise can provide an automated way to service those requests.

Service Catalog

A native integration for ServiceNow Service Catalog makes it simple for IT Operations to use Terraform Enterprise for provisioning, policy enforcement, and logging of all infrastructure requests. Anyone in the organization can use ServiceNow Service Catalog to request infrastructure.

ServiceNow integration is ideal for "cookie cutter" environments which do not need ongoing updates.

Developer Reference

The Terraform ServiceNow integration can be customized by ServiceNow developers using the Service Catalog Integration Developer Reference

The Terraform/ServiceNow Integration codebase includes ServiceNow Script, which includes classes that are used to interface with Terraform Cloud. The codebase also includes example catalog items and flows that implement the interface to the Terraform Cloud API.

API Endpoints

If you don't have ServiceNow or wish to customize your own pipeline/self service infrastructure, Terraform Enterprise has a VAST API that is exposed for consumption. There are a few notable points you will need to consume;

- API Driven Workflows: www.terraform.io/docs/cloud/run/api.html
- Module Registry API: www.terraform.io/docs/cloud/registry/index.html
- Workspace Status and API: www.terraform.io/docs/cloud/api/workspaces.html

API Driven Workflow

In the API-driven workflow, workspaces are not directly associated with a VCS repo, and runs are not driven by webhooks on your VCS provider.

Instead, one of your organization's other tools is in charge of deciding when configuration has changed and a run should occur.

Usually this is something like a CI system, or something else capable of monitoring changes to your Terraform code and performing actions in response.

Advanced Use Cases

Using the API we can deliver some more advanced use cases with Terraform Enterprise such as:

- Code Promotion
  - Running a deployment after deploying and testing changes in a workspace
  - **However** if downstream changes are purely Terraform based, `Run Triggers` offer a much easier approach
- Blue/Green Application Deployment
  - Having an orchestration tool deploy test versions of the application infrastructure and coe to various beta environments, automatically validate and promote releases

Chapter Summary

- The end state of any organization should be to run Terraform as a Service and enable Self Service
- Modules are a pre-requisites to going self service
- ServiceNow integration is provided as a native integration
- Terraform Enterprise provides a robust and expansive API

Reference links

- Self Service Infrastructure
- Self Service Webinar
- ServiceNow and TFE
- ServiceNow, DevOps and TFE