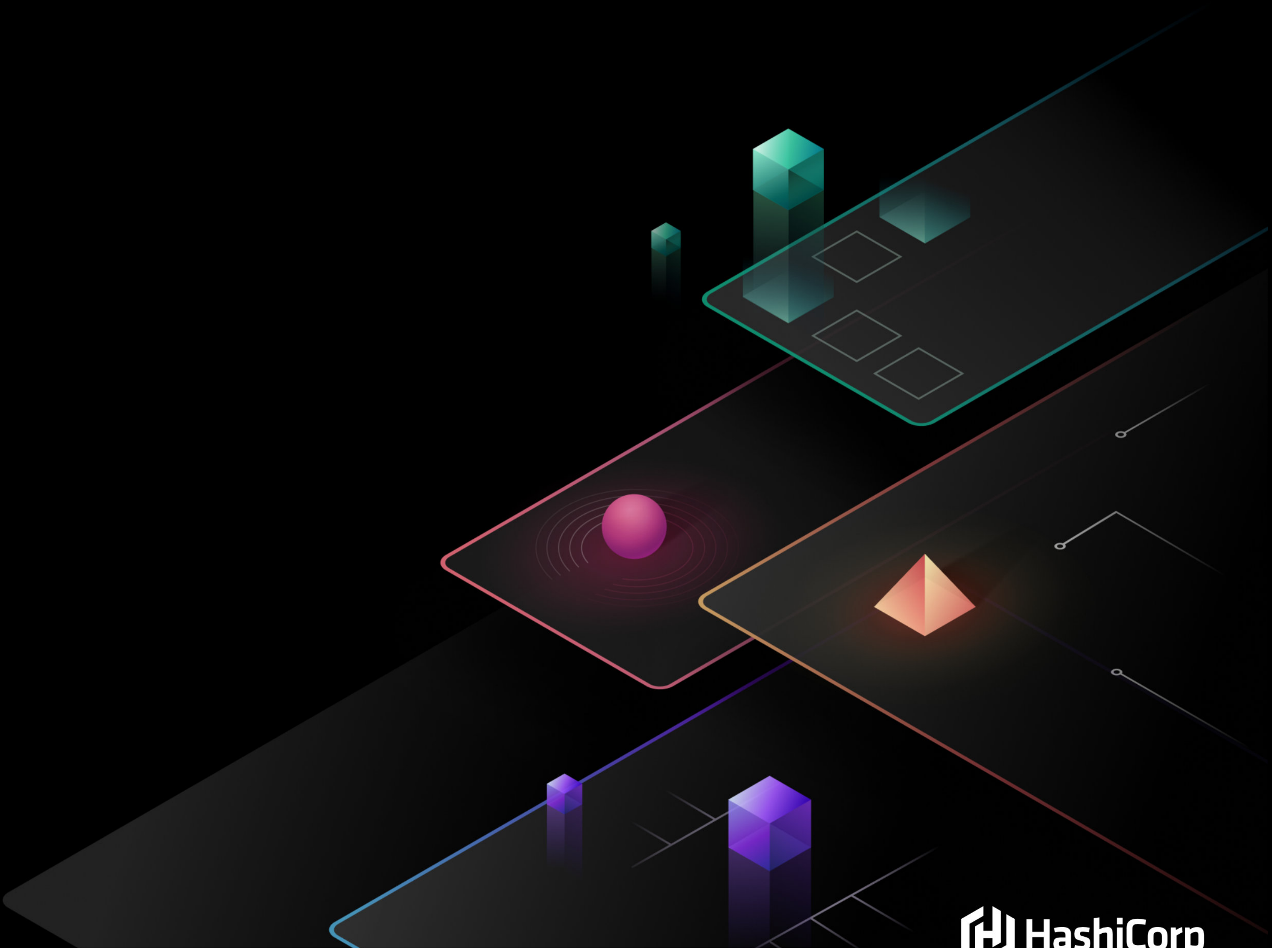
HashiCorp

Terraform Enterprise: Operating Guide for Adoption

HashiCorp | Validated Designs | Version v1.0.1

02 November 2023



Contents

1 Introduction

This document gives recommendations on how to implement Terraform infrastructure-as-code (IaC) as a shared service for your organization. Different organizations call the team responsible for this task different names, including the "Platform Team" or "Cloud Center of Excellence" (CCoE). No matter what your team is called, this document helps teams responsible for owning Terraform IaC in their organization.

Our objective in these HashiCorp Validated Designs (HVD) is to give you prescriptive guidance based on our experience partnering with hundreds of organizations who have implemented Terraform Cloud/Enterprise. We should acknowledge that our field is complex and the same solution can be implemented in many permutations. No matter what choices you make, what matters most is that you are able to safely provision and manage cloud resources at scale and experience the business benefits and value that automated Terraform Cloud/Enterprise workflows provide.

Unless specifically mentioned, concepts that apply to Terraform Cloud (TFC) also apply to its self-hosted version, Terraform Enterprise (TFE).

1.1 HashiCorp Validated Design maturity stages

While working with our customers, we have identified common patterns of maturity, allowing us to categorize customers into three main stages: Adopt, Standardize, and Scale. Each of the maturity stages is covered in a separate HVD document and assumes that the previous maturity stage is successfully implemented.

Adopt	Standardize	Scale
Infrastructure-as-code (IaC), cloud provisioning, secure variables, VCS integration/pipeline, RBAC (team collaboration), observability	Central registry development (policy-as-code, run tasks, modules), image management, day 2 resource management, network infrastructure management	Cost management and optimization, private DC provisioning, self-service workflows, event notification

Adopt: These customers have recently partnered with HashiCorp and are investing substantially in adopting infrastructure-as-code (IaC) for their enterprise. These customers lay the groundwork for growth by embracing fundamental use cases (as listed in the table above) facilitated by the Terraform Enterprise platform.

Standardize: These customers have completed the "Adopt" use cases and are now prioritizing the availability of the "Platform" as a shared service for the entire organization. At this stage of maturity, the platform team focuses on establishing guardrails by implementing policy-as-code and utilizing modules from the private registry and completing the development of an automated onboarding process for internal customers.

Scale: Once the platform is available to the wider organization and workloads are provisioned using an efficient, unified workflow, the platform team will need to address issues such as cost management, optimization, and other related use cases. These measures will ensure efficient scaling of operations for the organization over time.

1.2 Prerequisites

Review HashiCorp's cloud operating model which enables your organization to unlock the fastest path to value in a modern multi-cloud datacenter:

- Cloud Operating Model

If you are using Terraform Enterprise, this guide assumes that you have reviewed and implemented the following HVDs:

- Terraform: Solution Design Guide (Self-Managed)

1.3 HVD document structure

This document covers the "Adopt" phase of operating Terraform on the maturity scale and includes the following:

Section	Summary
Background	Provides an overview of the document
People and process	Recommendations on how to organize teams for optimum effectiveness for IaC provisioning
Configuration for first use	Prescriptive configuration of the deployed platform (Terraform Cloud/Enterprise) and making it ready for provisioning.
IaC and cloud provisioning	Discussion on infrastructure as code (IaC) concepts and best practices. Using the configured platform to provision resources in the cloud.
Terraform workflows	Discussion on the three main pipeline workflows (VCS, CLI, API) along with their pros and cons. Discussing best practices for branching strategy and other considerations for workflows.
Observability	Details on how to leverage audit logs to answer questions such as "who made what changes."

1.4 Objectives

You are implementing Terraform Cloud/Enterprise to achieve your company's business and functional objectives. Here, we list what we expect the goals you should realize after implementing the recommendations detailed in this guide.

1.4.1 Business objectives

1. **Reduce time to market:** This guide will assist you in establishing a robust standard workflow for provisioning, configuring, and managing the lifecycle of hybrid/multi cloud infrastructure. When implemented effectively, developers can provision infrastructure more efficiently, reducing the time it takes for your organization to introduce new products and features to the market.
2. **Mitigate risk:** By securing Terraform state, protecting cloud credentials, and implementing RBAC (role-based access control), you will significantly reduce the risk associated with your infrastructure.
3. **Consistent compliance**: Through policy-as-code, organizations will achieve compliant infrastructure, automate audit, address asset lineage concerns, manage against cost expectations, and respond proactively to regulatory change.
4. **Improve skills and retention**: Through infrastructure as code reuse, organizations reduce the cognitive load associated with onboarding new talent and retain that talent longer by improving productivity for team members.
5. **Optimize cloud cost**: By implementing a central shared service for provisioning, you will be able to optimize cloud spend and costs. This is achieved by standardizing and enforcing best practices on providing visibility into what is being provisioned across the organization. While details on how to do this and implement the necessary guardrails is covered in later HVDs, this guide is a necessary prerequisite towards that goal.

1.4.2 Functional objectives

1. Adopt a mature golden workflow for infrastructure provisioning
2. Enhance security posture
3. Improve traceability of actions and ensure audit readiness

1.5 Onboarding/adoption checklist

We recommend that the following tasks be accomplished for a successful onboarding and adoption of Terraform Cloud/Enterprise. The time it takes to complete this initial phase will vary depending on the complexity of your organization and the level of executive alignment. However, we have found that using HashiCorp implementation services or partner-provided services can significantly accelerate the process.

1.5.1 Project checklist

1. Identify key people from the Platform Team who will own and operate Terraform. In your organization, the Platform Team may own the architecture but the day-to-day operations may be delegated to a production services/support team who have 24/7 staffing arrangements. Both teams must be engaged at the outset.
2. Identify key executives sponsoring this project.
3. Establish cadences with the HashiCorp account team. We recommend the following:
   - Weekly/bi-weekly cadence
   - Quarterly business review with sponsoring executives
4. Enablement plan:
   - Platform Team enablement plan: We recommend that key platform team members attend HashiCorp Academy training. This training will also enable the Platform Team to be trained as trainers for the organization.
   - Application team enablement plan: We recommend that application teams be trained either by the Platform Team "trainers" or attend free hands-on workshops offered by HashiCorp solution engineers and architects.
5. Business unit onboarding schedule: Create a schedule for onboarding business units and/or application teams. We recommend for the "adopt" phase that you start with one or a handful of business units. (see note below)
6. Establish key milestones to track progress. We recommend the following key milestones:
   - Terraform Cloud/Enterprise onboarding
   - Platform team enablement
   - Application team early-adopter enablement
   - Application team early-adopter onboarding

Note on onboarding business units / app teams
HashiCorp recommends that you base the initial business unit / application team onboarding schedule on a representative set of early-adopter teams with one or more of the following characteristics:

- A high incentive to use infrastructure as code to achieve their goals,
- Have developed a DevOps skill set.

Those characteristics will increase the chances of early success.
The schedule should introduce approximately five teams in the first set, working with them from development to UAT. Then, the platform team will introduce a set of about twenty teams into development. This second set would benefit from the feedback the more experienced first set provides as part of pipeline refinement.
When the first set has reached the production environment and the second set has reached UAT, the platform team can introduce a third set of approximately 50-100 application teams into development, benefiting from the further refinement brought through feedback and collaboration with the first and second sets of teams. Through working with this third set, increased scale is both visible and demonstrable to senior management, making project success highly likely.

Onboarding checklist

1. Establish core integrations:
    1. VCS
    2. SSO/IdP
    3. Terraform Cloud (TFC) agents (optional)
    4. System logs and metrics (for Terraform Enterprise)
    5. TFC agent logs and metrics (optional)
    6. Audit logs
2. Establish a workflow to onboard application teams to Terraform Cloud/Enterprise:
    - Workflow vending
    - Cloud credentials for workspaces
3. Test of initial end-to-end CLI-driven run.

4. Test of end-to-end VCS-driven run.
5. Test SSO for the Platform Team and application teams.
6. Initial discussions with first set of early adopter teams regarding user onboarding experience and updates to the project backlog with next step improvements

We recommend that onboarding of Terraform be completed within 4-6 weeks of executing a new Terraform Cloud / Enterprise contract.

Adoption checklist

1. Determine the consumption model most suited for your organization.
2. Establish a GitOps-based workflow for application teams. This is likely to map to your existing organizational git repository standards.
3. Branching strategy: Decide on the branching strategy for managing environments.
4. Complete an adoption maturity assessment with a HashiCorp solutions architect.

We recommend scheduling a maturity assessment within 3-6 months after the completion of onboarding.

2 People & Process

Terraform isn't just a standalone tool: its effective use requires thoughtful allocation of staff resources and processes to achieve a notable return on investment.

Implementing quick changes to people and processes can be difficult due to cultural and political dynamics. As highlighted before, understanding the underlying concepts and the reasons ("why") is crucial. Being practical is vital since modifying an organizational structure is gradual, and adjustments will likely be needed to fit specific requirements.

This section delves into key organizational ideas that our customers have found valuable for deploying Terraform effectively and embracing the Cloud Operating Model (COM) discussed earlier.

2.1 Modern operations framework

Adopting the Cloud Operating Model is essential due to today's widespread use of cloud computing. Cloud computing was born in 2006 when Amazon's AWS introduced the Simple Storage Service (S3). Amazon launched AWS because it saw an unmet need to shorten development cycles by providing a new way to consume infrastructure: use an API to deploy and configure infrastructure available in a self-serve lease model instead of a buy model. This approach proved very successful, prompting competitors to enter the market: Google launched GCP in 2008, and Microsoft launched Azure in 2010.

The API model for consuming infrastructure required a new way to orchestrate the management of infrastructure components, and Infrastructure as Code (IaC) quickly emerged as an efficient new approach. Terraform (introduced in 2014) stood out as a best-of-breed tool in this area.

Competition between the cloud vendors increased the innovation rate, which meant more and more organizations started moving to the cloud. The use of Terraform furthermore facilitated this move because it made it easy to manage complex application infrastructures in the cloud.

Around 2015-2016, yearly cloud spending approached $100 billion, and many important revenue-generating workloads were running in the cloud. Companies realized they had to set up guardrails and control the infrastructure they provided in the cloud. At the same time, they had to offer cloud-enabled services to their customers quickly to keep their competitive advantage.

In this context, HashiCorp introduced the Cloud Operating Model, which allowed organizations to industrialize cloud adoption and reconcile those seemingly opposing goals.

2.1.1 Organizing for industrialized cloud adoption

This is how we see how many of our customers are orienting to adopting an industrialized modern IT operations framework.

The diagram below is tuned to our current context of Infrastructure provisioning.

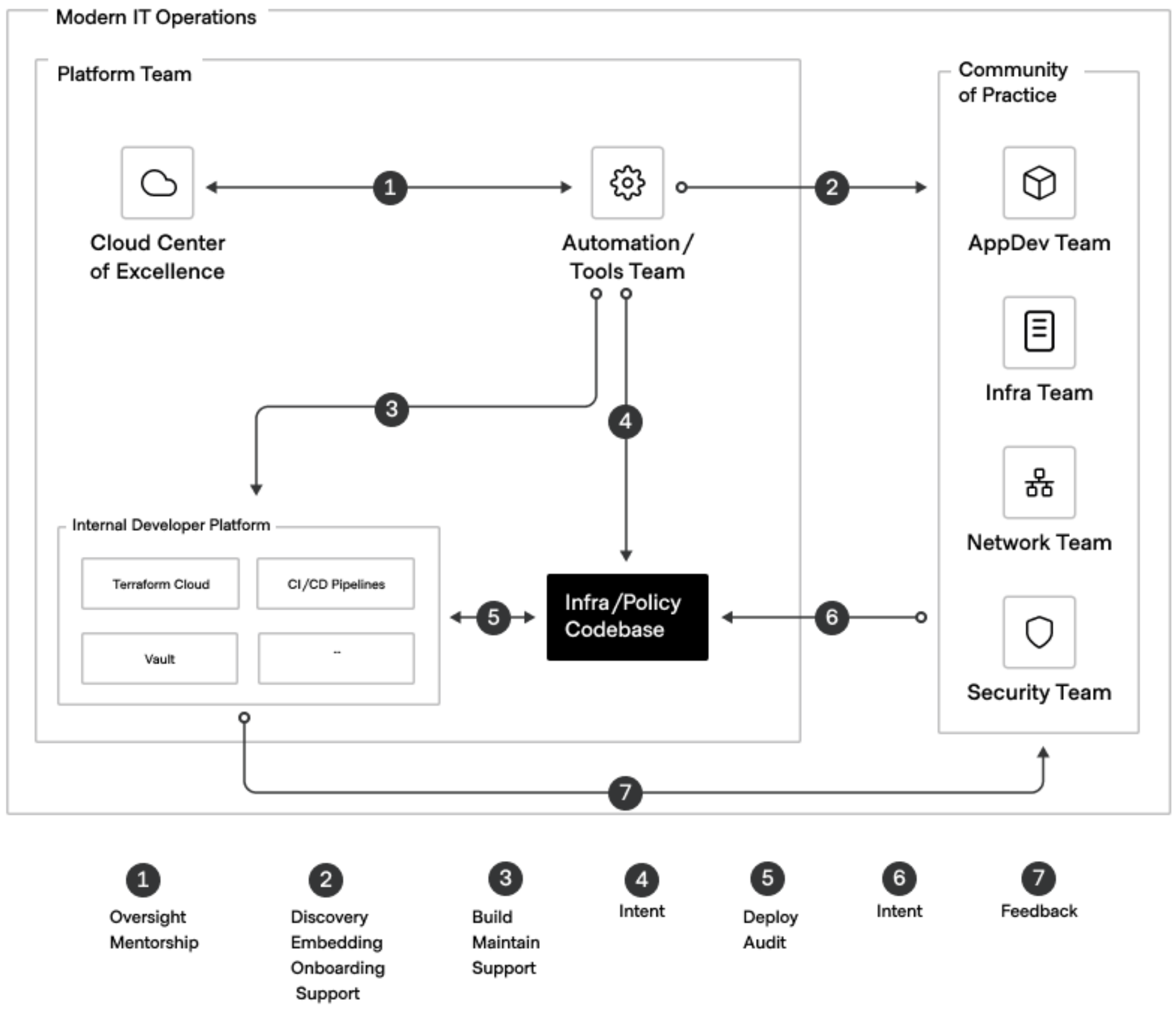


Figure 1: Overview of the Platform team and how it relates to the rest of the organization

1. The Cloud Center of Excellence (CCoE - discussed in more detail later) oversees the automation/tooling team. The relationship is bi-directional, with the automation/tooling team providing feedback and data used by the CCoE to refine its process and adjust its governance based on observer behavior. We discuss CCoE in more detail, but in a nutshell, its primary goal is to set the strategy.
2. The automation/tooling team conducts discovery and gathers requirements from the various consumers of the internal developer platform to ensure the shared services it provides meet their needs. Some of the categories that this falls under include:

    - Infrastructure stack requirements: The various teams, including the application development, networking teams will provide information on the kinds of stacks they need to host their application. For example, the need for PostgreSQL DB, Redis, step functions, load balancer, etc. The tools team will use this information to create standardized modules in the private registry that can be reused by the various teams.
    - Security requirements: The security team will provide policy requirements, for example, that you should encrypt S3 buckets at rest, that you should not expose port 22 to the public internet, etc.
    - The relationship between automation/tooling and security is critical, as the security team will understand the security requirements relevant to the business vertical but will likely not have the right mix of skills needed to implement the requirements relevant to this type of technical change.

3. The automation/tooling team is responsible for installing (in the case of self-hosted tooling), configuring and operating the tools that are part of the application estate. We discuss this team in more detail below, but establishing this team allows for much-reduced time-to-market. It is likely that this team already exists within your central technology office.
4. The automation/tooling team sets the intent for the IaC and policy-as-code, working with the security team as needed. For example, the automation/tooling team might establish a requirement that PostgreSQL be used in the organization but should be only deployed via a module that it maintains. They will then ensure that the Terraform module is available and then establish the policy that anytime the module is deployed, it is using this module. This provides standardization and scale simultaneously. Terraform modules and policy-as-code are discussed below.
5. The IaC/policy code base is stored in Git, and it needs to be deployed and audited. Terraform Cloud/Enterprise allows you to deploy IaC and policy and emits audit events that can be

consumed by the organization's preferred log management tool.

6. The Platform team gets intent from various stakeholders, including security, on what controls need to be established, including policy-as-code.
7. The tools provide the community with feedback regarding the status of the infrastructure, drift and many other aspects, including policy compliance.

2.1.2 Internal Developer Platform (IDP)

The Internal Developer Platform (IDP) has emerged to address the challenge of reducing development cycles and organizational complexity, especially as cloud computing and IaC have drastically cut down infrastructure provisioning times.

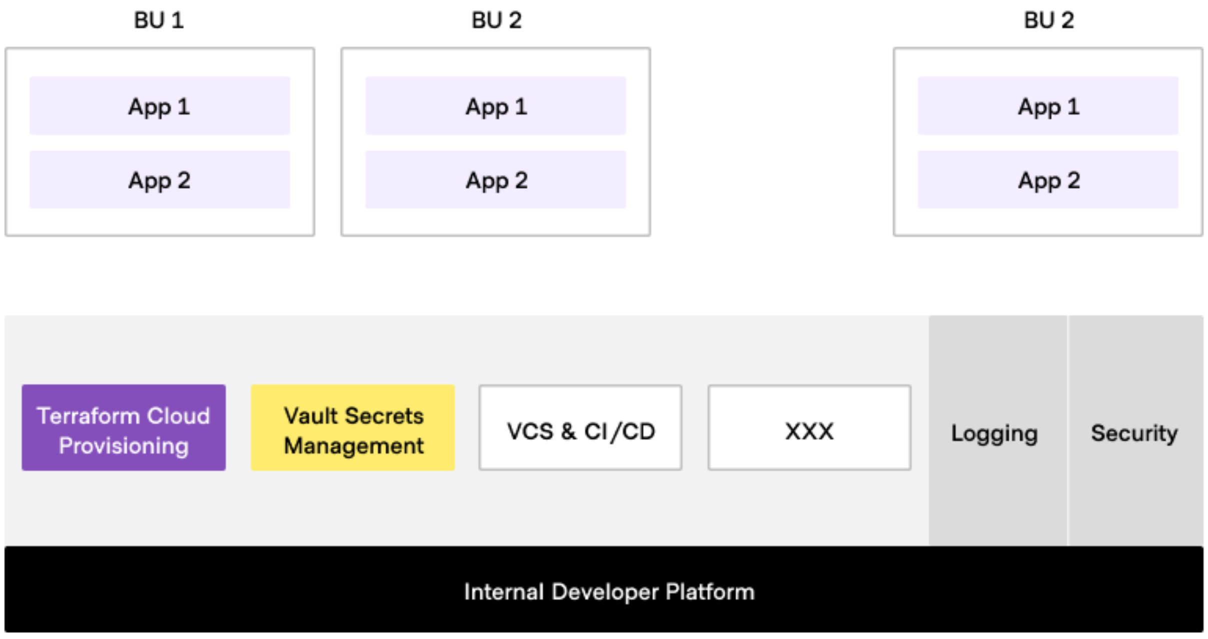


Figure 2: The Internal Developer Platform

The IDP builds on the time-tested shared service model improved by adding a Product Management approach. In this model, a Platform team works with developers to build golden paths that development teams can consume in a self-service mode.

Because developers use infrastructure, the IaC service is an integral component of any IDP. But it is also the foundation of the IDP service itself as the Platform team uses it to deploy and manage the infrastructure used to run the other IDP services such as monitoring, secrets management, log management, etc.

With this approach, two key roles are typically identified: producers and consumers. The terms "Producer" and "Consumer" are used to define the roles and responsibilities within a shared service or IDP provided by a "platform team".

While we talk of the "platform team" as a singular team, it is possible that the "platform team" is composed of multiple sub-teams with separate areas of responsibility. For e.g. Secrets Management might be owned by a separate team from Infrastructure provisioning. These sub-teams will however collaborate closely to ensure that developers in the various BU app teams have easy access to these shared services.

Producers

In a shared service model, the Producers have a role in configuring the system to meet the needs of the different consumers. Their responsibilities include:

- Providing a seamless onboarding process to the Terraform Enterprise/Cloud platform.
- Managing the Private Registry.
- Overseeing Policy as Code implementation.
- Offering necessary enablement to the consumers.

The Producers ensure that the shared services are readily accessible and efficiently utilized by the consumers, unlocking the benefits of the Terraform Cloud/Enterprise platform.

Consumers

In a shared service model, consumers refer to any team within the organization that utilizes the shared services provided by the Platform team. These teams have responsibilities that include:

- Initiating requests for access to the Terraform Cloud/Enterprise platform
- Writing Terraform code based on the available Private Registry modules and recommended practices established by the Platform team.

Consumers play a crucial role in leveraging the shared services to meet their specific needs and requirements while adhering to the guidelines and standards set by the Platform team.

2.2 Organizational big picture

The following diagram represents a bigger picture of where other organizations sit in relation to each other and how they interact with the Platform team. We do recognize that many companies may use different terms, and the hierarchy may differ, but the essence remains the same on how we should orient for modern IT delivery.

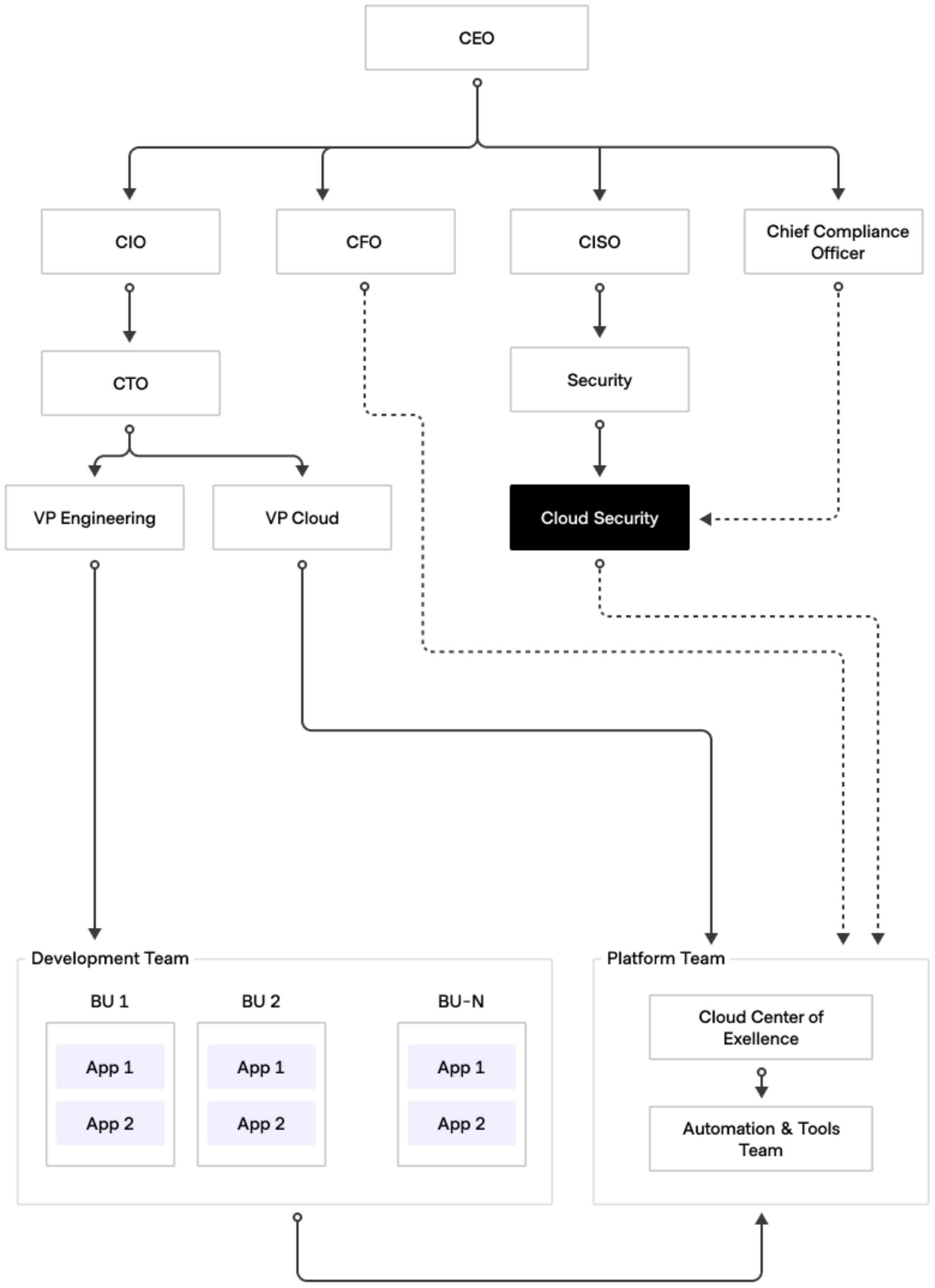


Figure 3: Organizational big picture

The hierarchy of application teams and platform teams can vary depending on the organization. In some organizations, such as those in the banking, telecommunications, and healthcare industries, application teams may fall under a completely different hierarchy than the Platform team. In other organizations, such as those that are focused on technology products, the application teams and platform teams may be part of the same hierarchy.

Other organizational functions, including finance and security, provide "governance" inputs to the Platform team and will request reporting and feedback to measure how effectively the governance is being applied.

2.2.1 Platform team

The platform team is responsible for overseeing the overall implementation & management of the shared tools and services to enable golden workflows, and driving its adoption within the organization. When a platform team matures they will tend toward providing their consumers an integrated user experience (that implements golden workflows) typically called Internal Developer Platform (IDP).

In the context of Terraform adoption, the Platform team ensures efficiency and standardization by establishing and enforcing IaC standards and best practices, promoting consistency across project teams and minimizing the risk of errors and inconsistencies that can disrupt operations. They also enable scaling and reusability by creating reusable Terraform templates and modules that streamline operations and reduce redundant efforts.

Additionally, the Platform team ensures governance and compliance by aligning the infrastructure defined in IaC with organizational policies and industry regulations. This ensures a secure and compliant environment. They provide valuable knowledge and expertise in Terraform and IaC, serving as a centralized resource for expert guidance. This reduces the learning curve for project teams and ensures consistent application of knowledge throughout the organization.

The Platform team facilitates automation and integration, automating common tasks using IaC and integrating it with other DevOps tools. This streamlines processes, increases speed, and reduces manual errors. They also address the security aspect by ensuring adherence to security best practices in infrastructure setup and maintenance, reducing vulnerabilities to cyber threats.

Moreover, the Platform team significantly reduces the time to market for new features and applications. With a mature IaC/Terraform service, they expedite the deployment of new infrastructure, enabling faster time to market.

Cloud Center of Excellence (CCoE)

CCoE is the strategic component of the Platform team. Also known as a Cloud Competency Center, it plays a pivotal role in driving cloud adoption within an organization. One of their key contributions is guiding the development and execution of an organization-wide cloud strategy. They work closely with stakeholders to ensure that the strategy aligns with the business goals and optimizes resources effectively.

They establish and maintain governance frameworks to govern cloud operations, ensuring that cloud resources and services are utilized efficiently and adhere to the organization's standards and regulations. They also ensure compliance with regulatory standards specific to cloud usage, mitigating any potential risks.

Knowledge sharing and promoting best practices are paramount for the CCoE team. They foster a culture of learning and collaboration, sharing valuable insights and expertise on cloud technologies throughout the organization. By doing so, they empower all teams to leverage cloud resources effectively and stay up-to-date with the latest advancements in the field.

They closely monitor and manage cloud costs, helping the organization make informed decisions and optimize resource utilization. By identifying opportunities for cost savings and efficiency improvements, they ensure that the organization gets the maximum value from its investment in cloud resources.

Automation/tools team

The Automation/Tools team is a tactical component of the Platform team. They manage and maintain the automation and tools required for systems and services to operate effectively. They implement *golden workflows* for consumers to leverage the shared services and tools that are part of the platform. As this team matures, they become the product owners and implementors of the IDP.

There could be multiple teams within the Automation/Tools team, each responsible for a different tool or group of tools. The CCoE provides overall strategic guidance for this team, and they collaborate with the consumer community to ensure that the services or workflows they enable meet their needs.

In simpler terms, the Automation/Tools team is responsible for making sure that the tools and automation that the Platform team uses are working properly. They also work with the consumer community to make sure that the services and workflows that are enabled by these tools meet their needs.

Platform team key roles and responsibilities

1. Platform Team Leader
    - Provides leadership for the overall Terraform deployment and operations.
    - Manages relationship with executive level sponsors (CTO, VP Eng etc).
    - Manages relationship with HashiCorp account team and executives.
    - Establishes / approves key goals and milestones for the projects.
    - Acts as an escalation point in case of blockers in implementation or ongoing operations.
    - Reviews progress at quarterly business reviews with the HashiCorp account team.
    - Manages the Enablement plan for training key Platform team engineers.
    - Manages the Enablement plan for the community including application developers, Security team etc.
2. Product Owner
    - Establishes requirements for terraform implementation.
    - Works with stakeholders to determine business unit / application team onboarding schedule.
    - In case of migrating from Terraform Community Edition, creates a schedule and milestones for the migration in collaboration with stakeholders.
    - Manages documentation for the project.
    - Owns the design of the “golden workflow” for deployment of infrastructure
3. Technical Lead
    - Manages the various teams supporting the deployment of Terraform Cloud / Enterprise.
    - Ensures all the integrations (VCS, SSO, logging etc) are configured correctly.
    - Manages the disaster recovery posture for the platform.
    - Ensures that governance requirements from Security and Finance are implemented and shares data and reports to help those functions fulfill their roles.
    - Manages the sprints for the deployment and adoption of Terraform across the enterprise.
    - Owns the implementation of the “golden workflow” for deployment of infrastructure
4. Tool Team - SRE (for TFE customers)

- Fulfills the role of TFE administrator
- Owns the installation and operations of the TFE software platform.
- Monitors the performance and usage of the TFE instances, database, Redis and block storage.
- Ensures that the TFE token is securely stored in a secrets management system.
- Maintains a test TFE environment to validate integrations & upgrades.
- Configures the various integrations (VCS, SSO, logging, etc.)
- Ensures that TFC agents are configured correctly
- Works with the Security team to ensure the TFE certificates are valid.
- Ensures that the instances are scanned to make sure they meet security policies.
- Follows guidance laid out in the Solution Design Guide.

5. Tool Team - SRE (for TFC customers)
    - Fulfills the role of TFC administrator
    - Configures the various integrations (VCS, SSO, audit logging, etc.)
    - Ensures that TFC agents are configured correctly
    - Is part of the Owners group and ensures that only key administrators have access to it.
6. Tool Team - Architect
    - Implements the various "golden workflows" for infrastructure provisioning.

2.2.2 Security team

The Security team collaborates with the Platform team by establishing governance policies as well as monitoring and validation tools to ensure the security of the provisioned cloud-based infrastructure and services. They work towards securing the cloud environment by implementing a range of security controls, including network segmentation, access controls, and encryption. These measures contribute to safeguarding against unauthorized access and data breaches, reducing the risk of security incidents.

A key contribution of the team is ensuring compliance with relevant regulations, industry standards, and internal policies specific to the cloud environment. They play a vital role in establishing and enforcing security policies, conducting audits, and supporting the organization in meeting compliance requirements. This could include establishing controls within CI/CD pipelines, for example, scanning code with SAST/DAST tools, checking images for vulnerabilities, etc. By ensuring adherence to these standards, the team helps maintain a secure and trustworthy cloud infrastructure.

Identity and access management (IAM) is an area where the Cloud Security Team makes a significant impact. They actively manage user identities, access controls, and permissions within the cloud, employing strong authentication mechanisms and following the principle of least privilege. These efforts contribute to mitigating the risk of unauthorized access and maintaining the confidentiality and integrity of cloud resources.

The team's contributions extend to data security as well. They implement encryption, data classification, and data loss prevention measures to protect sensitive data stored or processed in the cloud. By taking proactive steps to secure data, they help prevent data breaches and maintain the privacy of critical information.

In the realm of threat detection and incident response, the Cloud Security Team plays a vital role. They set up monitoring systems and intrusion detection tools, actively monitoring for potential security incidents. In the event of an incident, they collaborate with other teams to respond swiftly and effectively, minimizing the impact and ensuring a prompt resolution.

Cloud security team key responsibilities

The Cloud Security Team plays a crucial role in ensuring the security of the Terraform Cloud/Enterprise platform is in alignment with organizational security policies. We recommend that they perform the following roles/activities in support of the Platform team:

1. Subscribe to and monitor security updates & vulnerability alerts to stay proactive in addressing any potential vulnerabilities. In collaboration with the SRE team, they work diligently to implement recommended updates and patches according to HashiCorp's guidelines.
2. Ensure the validity and timely renewal of TLS certificates for TFE. These certificates establish a trusted relationship with various system integrations such as version control systems (VCS), single sign-on (SSO), run tasks, and platform certificate chains for authentication. Managing the CA_bundle that reflects changes to certificates on integrated third-party systems is essential, requiring code and artifact updates for TFE deployment and potential system restart or rebuild.
3. Conducts regular scans of TFE and agent instances to identify and address any vulnerabilities. Ensure that the SIEM tools are connected to Terraform Enterprise servers and agents to ensure logs are maintained for any audit events.
4. Provide governance inputs to CCoE (governance is a topic for the Terraform Standardize HVD)

- Governance of the "policy as code" guardrails initiative for Terraform, which involves defining and enforcing policies to ensure compliance and security. Maintain the repository where Sentinel/OPA policies are stored, facilitating easy access and management of these policies.
- Establish a list of standards, such as CIS benchmarks, that need to be enforced across the TFE platform. They ensure that the relevant policies are made available in the "Policy as Code" repository and diligently enforce them on appropriate workspaces. By fulfilling these responsibilities, the Cloud Security Team contributes to maintaining a secure and compliant environment for Terraform Enterprise.

In a nutshell, the Cloud Security team works closely with the Platform team, and a good collaborative partnership is vital in ensuring that the Terraform Cloud / Enterprise platform is secure. When we mention "Governance" we mean that Cloud security provides the oversight and the controls that groups within the Platform team will then implement. For example, Cloud Security will provide a list of controls to enforce using Policy as code, which engineers within the Platform team will then implement. Once this work is complete, the Cloud Security team will perform acceptance reviews and testing to ensure that what was implemented actually works. This process improves the security posture of the platform.

The IaC governance / Sentinel will be covered as part of a the "standardize" HVD.

2.2.3 Business unit / application development teams

The Platform team offers "golden workflows" primarily for application development teams within Business Units (BUs). Business Units are key components within many organizations, and they represent smaller segments of a larger entity, streamlining operations and enhancing management. For e.g., a bank may have a business unit for investments and one for consumer banking to manage distinct profit centers. Multiple AppDev teams can exist within a single business unit, for example, an application team that supports web applications, one for mobile applications and one for data integration.

The professionals in these application development teams – including developers, engineers, and scientists – are essential assets. The Platform team's objective is to optimize workflows for them, allowing them to concentrate on enhancing the applications they oversee.

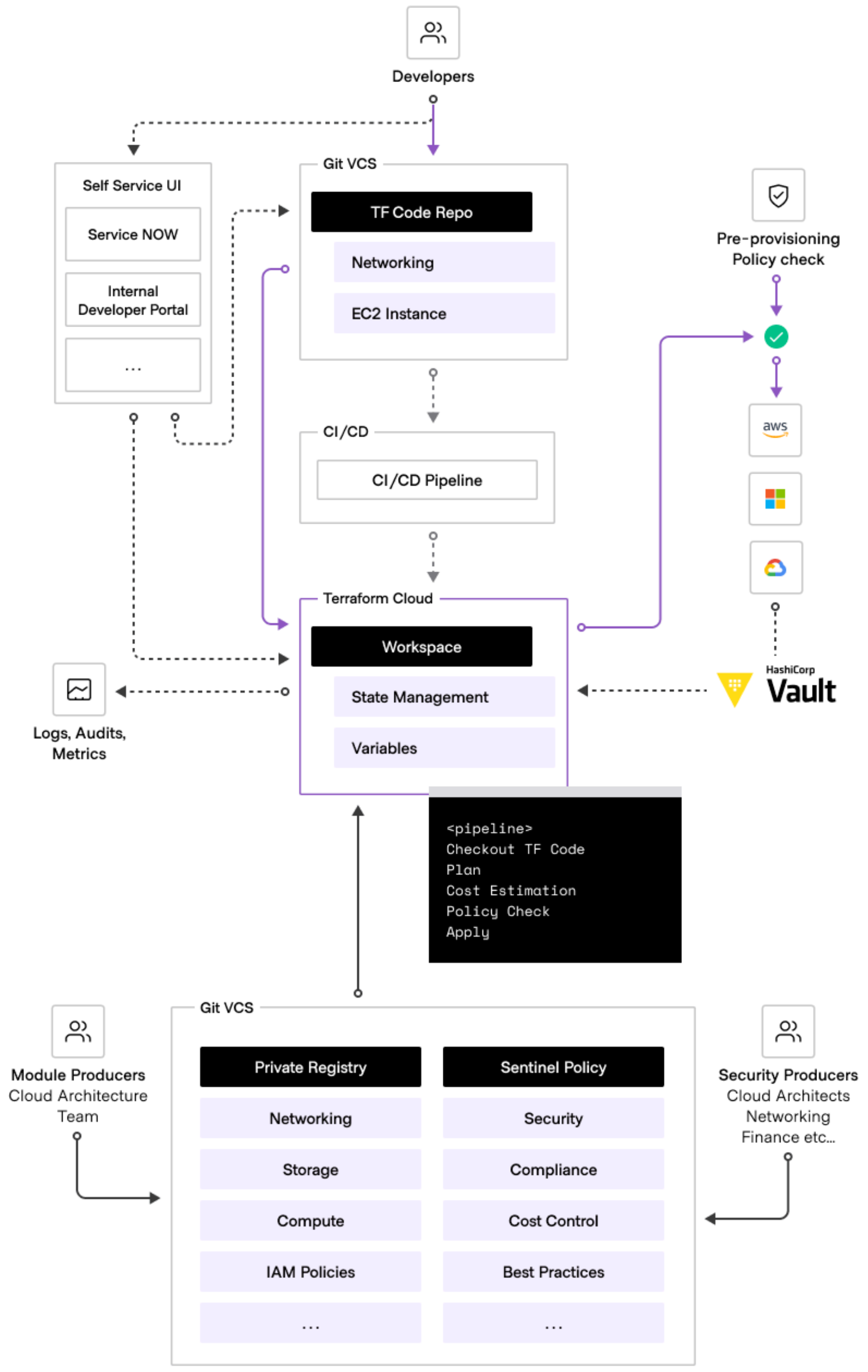
2.3 Golden IaC workflows

A golden workflow is a standardized, repeatable process for completing a specific task or achieving a specific outcome. It is typically well-documented and tested, and it is designed to be efficient and effective. Golden workflows are often used in software development, IT operations, and other business processes.

There are many benefits to using golden workflows. They can help to:

- Improve efficiency and productivity
- Reduce errors and mistakes
- Ensure consistency and quality
- Improve communication and collaboration
- Accelerate onboarding and training
- Facilitate automation

With Terraform, the Platform team enables Golden Infrastructure provisioning and management workflows. The Platform team empowers application development teams and other teams to independently provision infrastructure resources while maintaining centralized management and control. Terraform Cloud provides a user-friendly interface where teams can define and deploy their infrastructure using pre-approved Terraform configurations and curated modules. The Platform team oversees and governs these workflows, ensuring compliance with policies, security requirements, and best practices.



```
<pipeline>
Checkout TF Code
Plan
Cost Estimation
Policy Check
Apply
```

Figure 4: Infrastructure as Code golden workflow

In the next two sections we'll dive into each of the two main workflow categories, and shed light on the distinct workflows that must be implemented.

2.3.1 Producer workflows

These are workflows that are typically performed by members within the Platform team. More details about these workflows are covered as part of later HVDs.

1. Module Developer Workflow: In this workflow, module developers (referred to as module producers) create Terraform modules and register them with the private registry.
2. Policy-As-Code Developer Workflow: This workflow involves policy developers (referred to as policy producers) developing Sentinel/OPA code and registering them in Terraform Enterprise.

2.3.2 Consumer workflows

These workflows are enabled by the Platform team but used by various consumers including but not limited to various application/development teams.

1. Landing Zone Workflows: This foundational workflow involves provisioning cloud accounts and core Terraform configuration elements.
    - Cloud account & credentials
    - IaC VCS repository
    - Terraform project
    - Terraform workspace(s)
2. Developer Workflow: Once the landing zone is established, developers (referred to as Consumers) can create Terraform code and add it to the VCS repository. They can then utilize integrations such as VCS integration, CI/CD, or CLI to provision infrastructure to the cloud.

3 Consumption models

Looking at the evolution of IT infrastructure systems over the years, it's obvious to any observer that they have become increasingly sophisticated and capable, providing organizations with greater productivity gains. However, taking advantage of all those features also necessitated building specialized knowledge to deploy and operate these tools, resulting in functional silos. Over time, those functional silos caused an explosion of team-to-team handoffs and impeded agility.

Practices and technologies, such as DevOps, application-focused teams and cloud computing, emerged as a response to reduce team-to-team handoffs and increase business agility. They grew rapidly because they were successful at solving those problems and because the issue was widespread. You could now take advantage of those very sophisticated technologies without incurring the added team-to-team communication costs. And the early adopters of these practices and technologies gained a competitive advantage.

This evolution is leading to a situation where speed to market is more of a competitive advantage than ever. But for those organizations that made the jump or those that are seriously considering it, it also raises the question of how to adopt those practices and technologies for infrastructure provisioning and management.

Infrastructure as code (IaC) is a way of managing and defining data center components with code-based automation frameworks. It has been proven to be an effective pattern for provisioning and managing infrastructure in the DevOps model. When medium to large organizations begin using Terraform Enterprise and IaC, they typically offer two models of consumption to their IT customers:

1. Service Catalog Model: This model involves defining a core set of standard infrastructure stacks from which end customers can choose. It provides an accelerated or certified consumption path for infrastructure provisioning with limited customization options.
2. Infrastructure Franchise Model: Similar to real-life franchises, this model operates under a strict contract framework. With a robust policy model and supervision, end customers are empowered to build and run their infrastructure independently, with significant customization options.

Comparing the two consumption models of Infrastructure as Code, it is clear that implementing a digital transformation project of this scale requires careful alignment of resources. Both models can coexist, but you can only begin with one as it requires a lot of resources to create the initial catalog or set up necessary guardrails.

The diagram below illustrates when each model is preferable based on the target user group's technical knowledge level, and the group's need for customization.

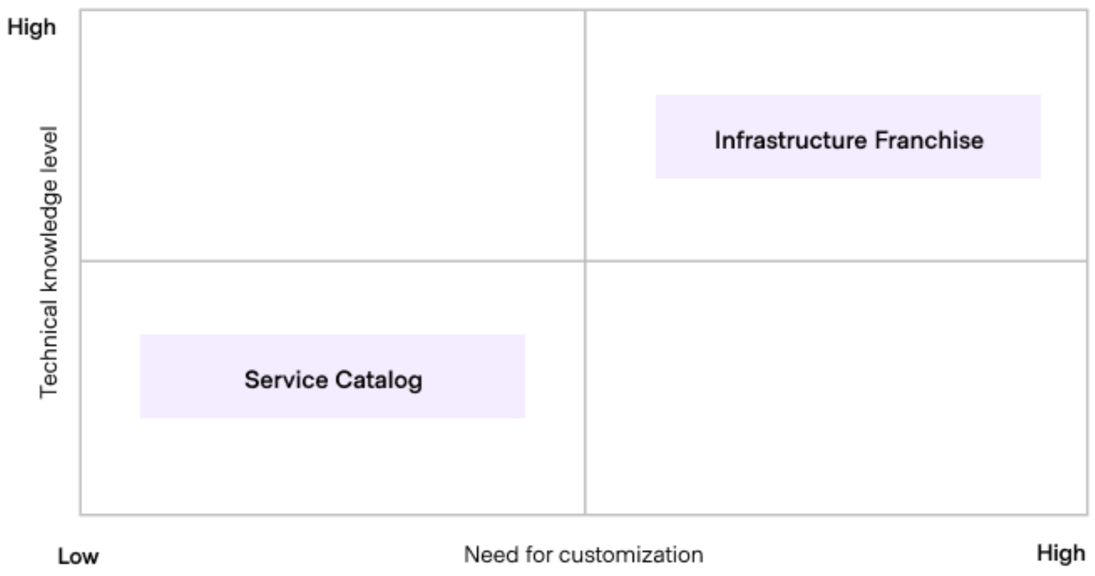


Figure 5: Choosing an IaC consumption model

	Service Catalog	Infrastructure Franchise
Primary UX	Vending Portal (UI, SNOW etc.)	Custom Workflow (Git, API/CLI, CI/CD etc.)
What can be built?	Standard modules	Anything via IaC within defined guardrails
Customization	Limited (via module parameters)	Unlimited
Primary Security model	Validated modules	Guardrails (policy-as-code etc.)

	Service Catalog	Infrastructure Franchise
Persona	Any - including business user/ project manager	Dev teams, SRE, etc.

Highlighting the functions and features that will be consumed in the Operating Models

3.1 Service catalog

A service catalog functions as a centralized vending portal offering heavily prescribed and mostly pre-configured infrastructure components. The primary objective of this model is to facilitate an accelerated consumption path for "certified" infrastructure elements. We'll go into more detail about the key aspects of this model which include:

- A vending portal
- A menu of pre-configured components
- Accelerated certified consumption path

3.1.1 Vending portal

The catalog is a central platform where users can view, request, and build infrastructure components that have been marked as "standard" by a core group of power users. The Platform team usually maintains this portal and its standard components, offering them as a service to end customers. The aim is to give users a self-service infrastructure experience, allowing them to audit requests and enforce chargeback policies.

3.1.2 Pre-configured components

The service catalog philosophy is based on the availability of pre-configured components. These components can range from entire application stacks to specific infrastructure aspects. The goal is to guarantee uniformity in infrastructure deployment. Organizations can use two methods to create and certify infrastructure:

1. Platform Team: A designated central team is authorized to create the organization's standardized patterns.
2. Contribution Pipeline: Establish an endorsement path and contribution pipeline to allow people outside the Platform Team to create endorsed patterns.

3.1.3 Accelerated certified consumption path

Successful organizations, like Meta (previously Facebook), Netflix, Wayfair, and others, attribute their success to defining consistent infrastructure. When starting projects, business units have two options:

1. Standard Infrastructure Stacks: Choose from existing standard infrastructure stacks to receive accelerated services and faster delivery.
2. Custom Stacks: Opt to define their own stack, leading to a slower delivery process as each component requires building and verification.

Naturally, this approach encourages a "highway" of success, with business units preferring existing trusted infrastructure building blocks for their projects. The following is a visualization of this model:

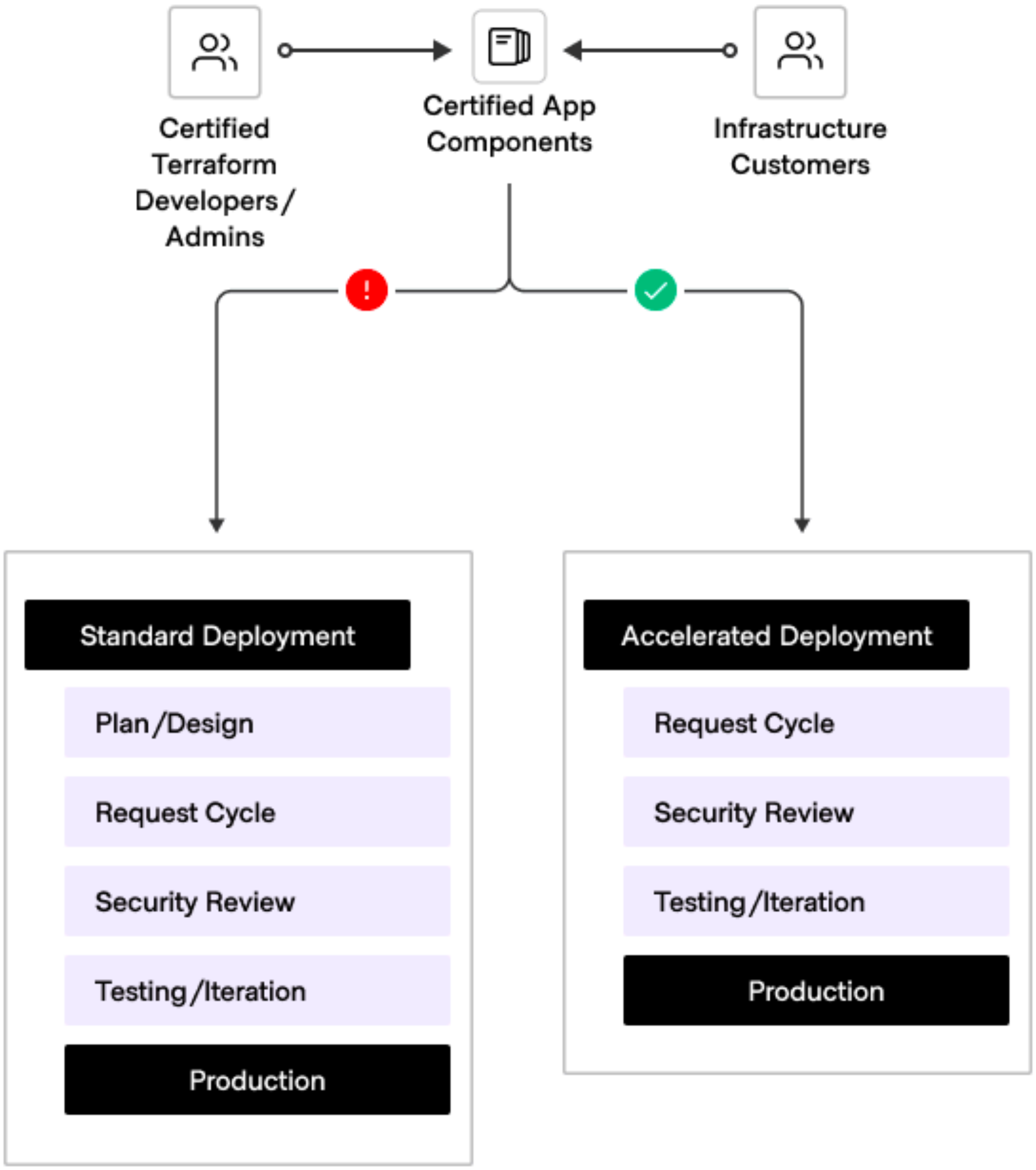


Figure 6: Service catalog workflow

3.1.4 Features and capabilities to support a service catalog model

Feature	Use Case
Private Registry	Terraform Cloud's private registry operates similarly to the public Terraform Registry, enabling you to share Terraform providers and modules within your organization. It offers support for versioning and provides a searchable list of available providers and modules.
Private Providers and Modules	Private providers and modules are hosted on your organization's private registry, restricting access to only members of your organization. In Terraform Enterprise, private modules are also accessible to other organizations configured to share modules with your organization. This allows for controlled access and sharing of custom-built modules across organizations within the Terraform Enterprise environment.
Run Tasks	Run Tasks in Terraform Cloud enable direct integration with third-party tools and services during specific stages of the run lifecycle. When triggered, a run task sends an API payload to the external service, which includes essential run-related details and a callback URL for the service to respond back to Terraform Cloud, indicating whether the task passed or failed. This feature allows seamless interaction with external tools, enhancing the flexibility and extensibility of your infrastructure automation workflows.

Feature	Use Case
3rd Party Certified Integrations	There is a full list of certified integrations available for Terraform Enterprise, and here are some of them: Aqua Security, Palo Alto Prisma Cloud, Firefly, Infracost, Kion, Lightlytics, Snyk. You may also build and customize your own integrations based on your specific requirements.

3.2 Infrastructure franchise model

The franchise model is a way of distributing products or services. It involves a platform team that sets up the company's best practices, services that can be used, and general instructions for using the infrastructure service. The main features of this model are:

1. The central control team
2. Consumption workflows and upfront governance
3. Controlled vending of specific components

3.2.1 Central control team

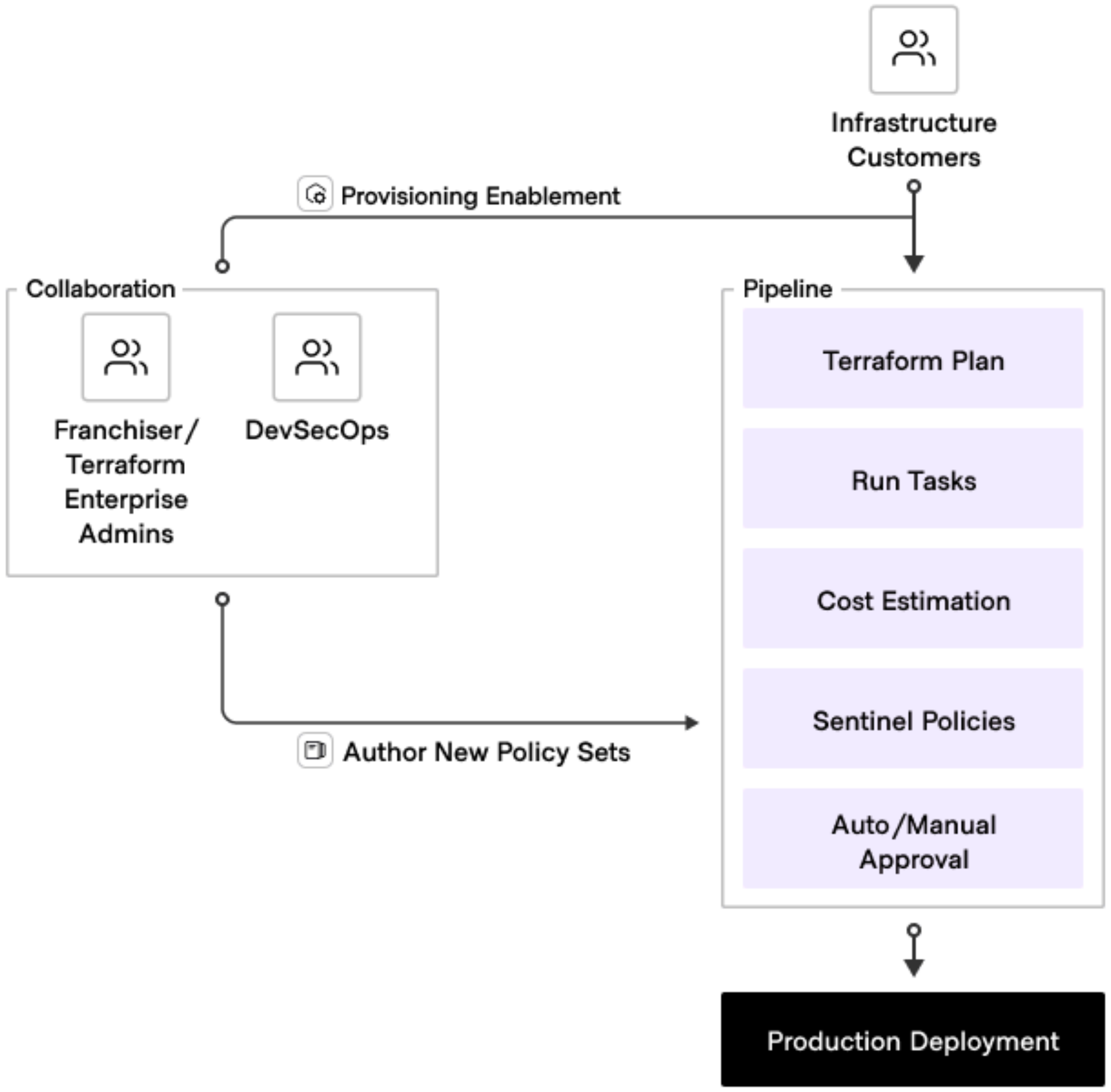
The franchise model has a platform team called the franchiser. This team provides franchisees with the workflow and resources they need. The franchiser is responsible for keeping the system running, setting rules, and adding extra capabilities to the workflow.

3.2.2 Consumption workflow and upfront governance

- **Consumption workflow**: Provide end users and customers with a way to access provisioning resources. This allows them to join the workflow and manage their own resource requirements.
- **Upfront governance**: Governance at the outset ensures that all customers and end users have the ability to provision resources while following legal, regulatory, and enterprise guidelines. Complying with regulations becomes easier and more cost-effective.

3.2.3 Controlled vending of specific components

The franchise model allows access to resources, but it also implements proactive controls to meet legal, regulatory, and enterprise standards. For existing resources, common controls are put in place to ensure the safety and security of the organization. Additionally, new resources go through testing, validation, and are controlled by basic policies. The following is a visualization of this model:



An overview for how a franchise model of infrastructure franchise can be supported.

3.2.4 Features and capabilities to support an infrastructure franchise model

An example of how to use each of these features to support services in a franchise model

Feature	Use Case
Terraform Workspaces	When working with Terraform, you'll be managing collections of infrastructure resources, and typically, organizations handle multiple collections. To facilitate this, Terraform employs a persistent working directory when run locally. This directory contains the configuration, state data, and variables for each collection of infrastructure. By organizing the configurations into separate directories, you can group infrastructure resources in a more meaningful and structured manner, allowing for better organization and management.
Terraform Workspace Projects	This feature enables the provisioning of self-managed portions of Terraform Enterprise. It allows end customers/users to use Terraform Enterprise while adhering to the same policies and controls as the root organization.
Sentinel Policies	Sentinel is a policy-as-code framework embedded within HashiCorp Enterprise products. It empowers organizations to make fine-grained, logic-based policy decisions and can be extended to utilize information from external sources.

3.3 Summary

Implementing and scaling Infrastructure as Code (IaC) is a major change for organizations that want to move away from imperative automation, manual provisioning, isolated IaC, or a combination of these models. We need to consider a practical starting point for IaC that multiple teams and developers can use.

In this section, we have compared two models of IaC adoption based on the company culture and level of maturity. The Service Catalog model focuses on providing standardized and reusable technology stacks. In contrast, the Infrastructure Franchise model allows teams to create their own Software Development Life Cycle (SDLC) workflows on an API-enabled platform. Both models have proven their value at scale, but each requires organizations to focus on different areas.

An organization adopting infrastructure as code will start with one of the models, but it will likely follow soon with implementing the other one. Not all internal customers are created equal, and this dual approach is necessary for organizations committed to meeting their users where they are.

4 Configuring for 1st use

This section will show you how to configure Terraform Cloud (or Enterprise) to meet your requirements for delivering Infrastructure as Code services to your internal customers.

Let's begin by taking a look at how Terraform Cloud organizes things and does access management.

4.1 Foundational organization concepts

4.1.1 Structure

Terraform Cloud helps you organize the infrastructure resources you want to manage with Terraform. It begins with the Organization concept, which usually corresponds to your entire organization. You can have one or more Projects under the Organization. Each Project can then manage one or more Workspaces. Workspaces are the containers to manage a set of related infrastructure resources.

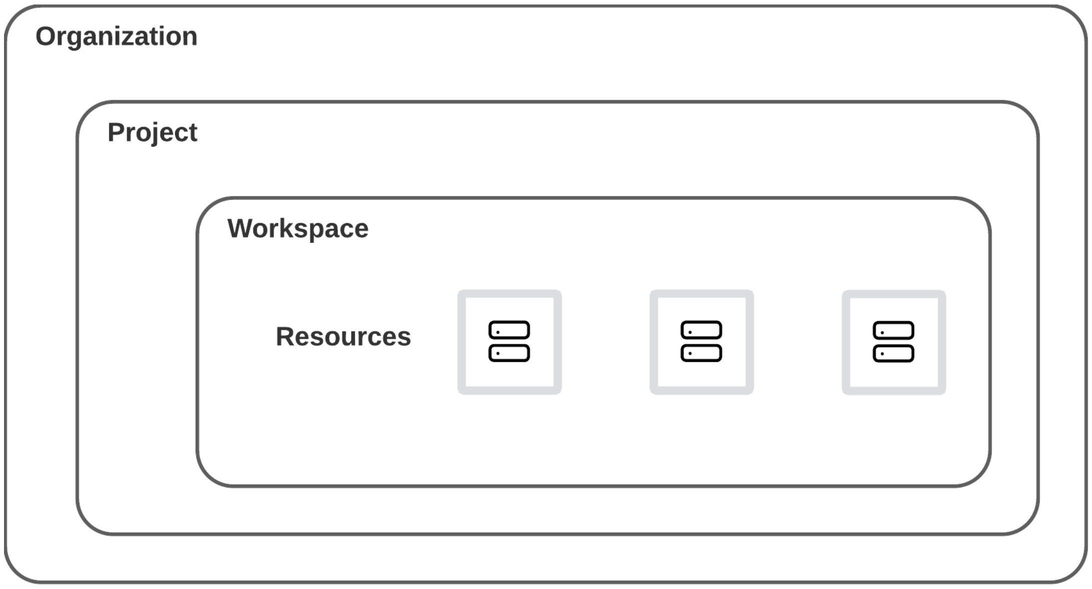


Figure 7: Terraform Cloud structure

4.1.2 Access management

Access management in Terraform Cloud is based on three components:

1. User accounts
2. Teams
3. Permissions

A user account represents a user of Terraform Cloud and is used to authenticate, authorize, personalize, and track user activities within Terraform Cloud.

A team is a container used to group related users together and helps to assign access rights easily. A user can be a member of one or more teams.

Finally, permissions are the specific rights and privileges granted to groups of users to perform certain actions or access particular resources within Terraform Cloud.

When managing your Terraform Cloud or Terraform Enterprise environment, it is recommended to implement SAML/SSO (Security Assertion Markup Language/Single Sign-On) for user management, in conjunction with RBAC (Role-Based Access Control). SAML/SSO provides a centralized and secure approach to user authentication, while RBAC allows you to define and manage granular access controls based on roles and permissions. By combining SAML/SSO and RBAC, you can streamline user onboarding and off-boarding processes, enforce strong authentication measures, and ensure that users have the appropriate level of access to resources within your Terraform environment. This integrated approach enhances security, simplifies user management, and provides a seamless experience for you and your team members.

4.1.3 Delegate access

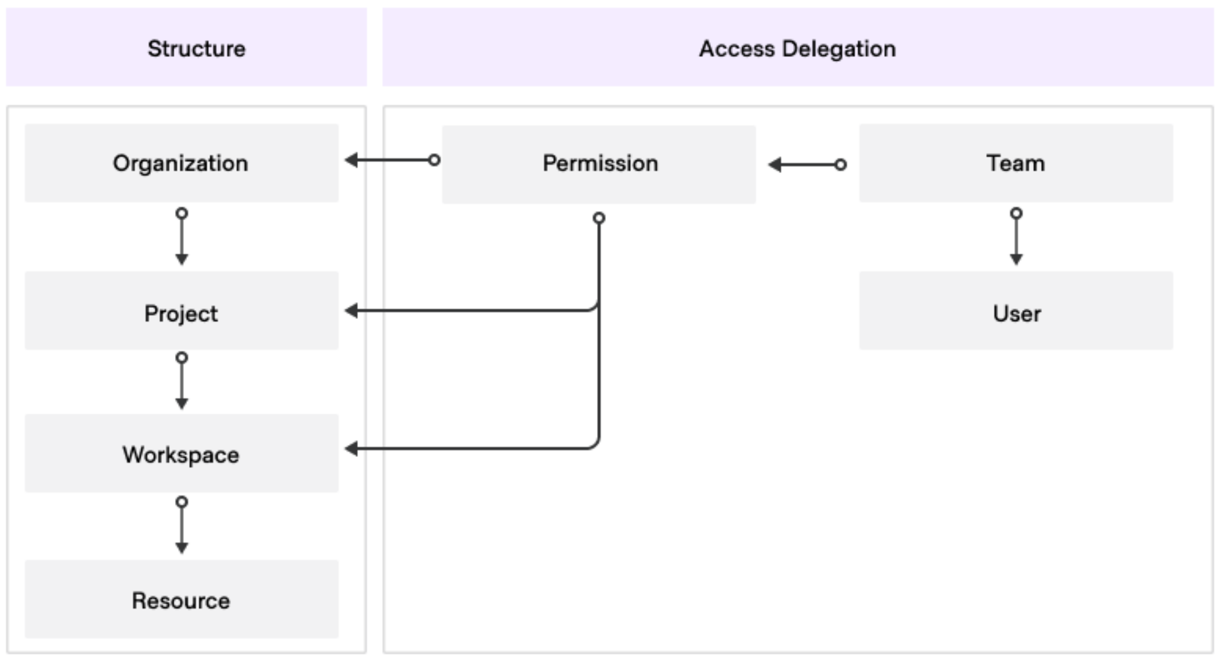


Figure 8: Terraform Cloud access delegation

In Terraform Cloud it is possible to assign team permissions at the following levels:

1. Organization
2. Project
3. Workspace

The level at which a permission is set defines the scope of this permission. For example:

- A permission assigned at the workspace level applies only to that workspace and not to any other workspace.
- A permission assigned at the project level applies to the project and the workspaces it manages, but not to workspaces managed by another project.
- A permission assigned at the organization level applies everywhere in that organization.

4.1.4 A word on naming conventions

As you configure your Terraform Cloud instance, you will need to create and name a number of entities: teams, projects, workspaces, variable sets, etc. We recommend that you create and maintain a naming convention document.

This document should include the naming conventions for each entity, as well as any other conventions you decide to use. For example, you may decide to use a specific prefix for all teams, or a specific suffix for all workspaces.

This document will help ensure that your TFC instance is organized and easy to navigate. Additionally, it will help you quickly identify and locate any entity you need.

4.2 Closer look at organization, projects and workspaces

4.2.1 Organization

Organization in Terraform Enterprise serves as the highest-level construct, encompassing various components and configurations. These include the following:

- Access Management
  - Single Sign-On (SSO)
  - Users and Teams
  - Organization API Tokens
- Projects and Workspaces
- SSH Keys
- Private Registry
- VCS Providers
- Variable Sets
- Policies
- Agent pools

We recommend centralizing core provisioning work within a single Organization in Terraform Cloud and Terraform Enterprise to ensure efficient provisioning and management.

This approach is compatible with a multi-tenant environment where you have to support multiple business units (BUs) within the same organization. Our robust role-based access control (RBAC) system combined with projects ensures that each BU has its own set of permissions and access levels. Projects enable you to divide workspaces or teams within the organization. We will discuss them in more detail in the following section.

By consolidating your provisioning work in one organization, you can benefit from streamlined integration with VCS systems, RBAC, and SSO. This simplifies the management and administration of your infrastructure as code service.

There is one reason justifying having a second organization: It is recommended that you maintain a separate test organization, to enable integration testing of various dependencies. This can save you time and money in the long run, as you won't have to go back and fix things that could have been prevented with proper testing. A test organization is the norm for large enterprises that have internal policies to test changes in isolated environments before promoting the change to the production environment.

Naming convention

We recommend the following naming convention for organizations:

- `<customer-name>-prod-org`: This organization serves as the main organization.
- `<customer-name>-test-org`: This organization is dedicated to integration testing and special Proof of Concepts (POCs) conducted by your team.

Access management and permissions

In this section we'll cover:

1. The Owners team
2. The organization-level permissions
3. Recommended approach to setting up organization-level access

The owners team

When an organization is established, an Owners team is created automatically. It is a fundamental team, and because of its role cannot be deleted.

As a member of the Owners team, you have comprehensive access to all aspects of the Organization, including Policies, Projects, Workspaces, VCS Providers, Teams, API Tokens, Authentication, SSH Keys, and Cost Estimation. A number of team-related tasks are reserved to the Owners team and cannot be delegated: Creating and deleting teams, managing organization-level permissions granted to teams, and viewing the full list of teams (including teams marked as “secret”).

Additionally, you can generate an Organization-level API token to manage teams, team membership, and workspaces. It’s important to note that this API token does not have permission to perform plans and applies in workspaces. Before assigning a workspace to a team, you can use either a member of the organization owners group or the organization API token to carry out some initial setup.

Organization-level permissions

Below are some of the key Organization-level RBAC permissions and their respective purposes.

- Manage Policies: Allows you to create, edit, read, list, and delete the organization’s Sentinel policies. This permission also grants you access to read runs on all workspaces for policy enforcement purposes, which is necessary to set enforcement of policy sets.
- Manage Run Tasks: Enables you to create, edit, and delete run tasks within the organization.
- Manage Policy Overrides: Permits you to override soft-mandatory policy checks. This permission also includes access to read runs on all workspaces for policy override purposes.
- Manage VCS Settings: Allows you to manage the VCS providers and SSH keys available within the organization.
- Manage Private Registry: Grants you the ability to publish and delete providers, modules, or both in the organization’s private registry. Normally, these permissions are only available to organization owners.
- Manage Membership: Enables you to invite users to the organization, remove users from the organization, and add or remove users from teams within the organization. This permission allows you to view the list of users in the organization and their organization access. However, it does not provide the ability to create teams, modify team settings, or view secret teams.

As you review the permissions above, it becomes clear that many of them are not required for typical users. Therefore, we recommend granting these organization-wide permissions to a select group of your team members, reserving them for the most senior members on your team and restricting access for members of the development team. This approach helps prevent unintended modifications or disruptions to your organization’s configuration.

It is crucial to treat the Organization Token as a secret and limit access to only a privileged few to maintain the security and integrity of your organization.

Additionally, you have the flexibility to create additional teams within your organization and grant them specific Organization-level permissions to suit your needs and ensure proper access control.

Setting up organization-level access

Because of the broad elevated privileges that the Owners team has, we recommend to limit membership to a very small group of trusted users (the Terraform Cloud core team), and create additional teams for a number of organization-level permissions. The table below illustrates the recommendation.

Role	TF core team	Security team	TF configurator	Registry team
Owners	X			
Policy Management		X		
Run Task Management			X	
VCS Provider Management			X	
Private Registry Management				X

You may need to make adjustments to this model based on the breakdown of roles and responsibilities in your organization. If you do, update this matrix based on your context and make sure you update your Terraform Cloud deployment documentation.

4.2.2 Project

In Terraform Cloud projects are a container for one or more workspaces. You can attach a number of configuration elements at the project level, which are then automatically shared with all workspaces managed by this project:

1. Team permissions
2. Variable set(s)
3. Policy set(s)

Because of that inheritance mechanism, projects are the main organization tool to delegate the configuration and management responsibilities in a multi-tenant setup. The two standard project-level permissions that are used to achieve this delegation are:

1. Admin - provides full control over the project (including deleting the project).
2. Maintain - provides full control of everything in the project, except the project itself.

Variations on the proposed delegation model are possible by creating your own custom permission sets and assigning those to the appropriate teams in your organization.

Benefits

Increased agility with workspace organization:

- Related workspaces can be added to projects to simplify and organize a team's workspace view.
- Teams can create and manage infrastructure in their designated project without requesting admin access at the organization level.

Reduced risk with centralized control:

- Project permissions allow teams to have administrative access to a subset of workspaces.
- This helps application teams safely manage their workspaces without interfering with other teams' infrastructure and allow organization owners to maintain the principle of least privilege.

Best efficiency with self-service:

- No-code provisioning is integrated with projects, which means teams with project-level admin permissions can provision no-code modules directly into their project without requiring organization-wide workspace management privileges.

Project logical boundaries

When deploying Terraform Enterprise at scale, organizing your projects effectively ensures streamlined workflows and improved efficiency. Defining logical boundaries for projects can help teams manage infrastructure resources more efficiently, adhere to the principle of least privilege, and simplify the setup of dynamic credentials. Here are some key considerations for defining project logical boundaries:

- Provider boundaries: Consider defining projects per cloud account to simplify the setup of dynamic credentials without the need for provider aliases or multiple "landing zones" in a project.
- Least privilege: Utilize project-wide teams if your project contains workspaces of similar criticality. For example, separate a Production networking workspace from a Non-Production compute workspace.
- Variable set usage: Leverage project-wide variable sets for default tags such as cost-codes, owners, and support contacts. Additionally, use dynamic credentials in a variable set for workspaces within a project.
- Practitioner efficiency: Evaluate whether it is efficient for a practitioner to traverse multiple projects to complete a deployment of an application stack. Consider consolidating related workspaces into a single project for better efficiency.

Project designs

With the above logical boundaries in mind, the proposed design below illustrates the organization of projects and workspaces:

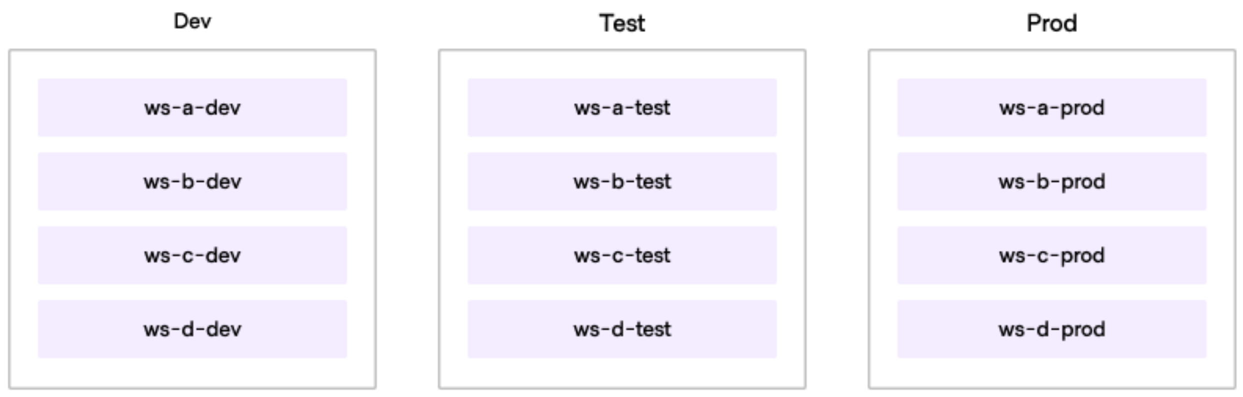


Figure 9: Project design

In this design, the grey boxes represent projects, and the purple boxes represent workspaces. We have defined the boundary as the respective environment, separating Dev, Test, and Prod workspaces into their own projects. This approach offers several benefits:

- Clear RBAC Definition: By segregating environments into separate projects, you can more

clearly define the role-based access control (RBAC) for practitioners, ensuring that each team has appropriate permissions only within their designated environment.
- Project-wide variable sets: Applying project-wide variable sets allows you to maintain consistency in variables such as cost codes, support emails, severity codes, and other configuration settings across all production infrastructure.

Adhering to this design enhances infrastructure management, improves team collaboration, and ensures a well-structured and efficient workspace organization within Terraform Cloud or Terraform Enterprise. More insights into workspaces will be provided in the next section.

Naming convention

We recommend assigning separate projects for each application within a business unit. The naming convention for projects is: <`business-unit-name`>-<`application-name`>.

Access management and permissions

Teams can be created without requiring any of the aforementioned Organization-level permissions. Instead, you can explicitly grant them permissions at the Project and Workspace levels. We recommend setting permissions at the Project level as it offers better scalability and allows teams to have the flexibility to create their own workspaces when necessary. By granting permissions at the Project level, you can effectively manage access control while providing teams with the necessary privileges for their specific projects and workspaces. Below are the permissions you can grant and their respective functionalities:

- **Read**: Users with this access level can view any information on the workspace, including state versions, runs, configuration versions, and variables. However, they cannot perform any actions that modify the state of these elements.
- **Plan**: Users with this access level have all the capabilities of the Read access level, and in addition, they can create runs.
- **Write**: Users with this access level have all the capabilities of the Plan access level. They can also execute functions that modify the state of the workspace models, approve runs, edit variables on the workspace, and lock/unlock the workspace.
- **Admin**: Users with this access level have all the capabilities of the Write access level. Additionally, they can delete the workspace, add and remove teams from the workspace at any

access level, and have read and write access to workspace settings such as VCS configuration.

Variations on the standard roles are possible by creating your own custom permission sets and assigning those to the appropriate teams in your project.

4.2.3 Workspace

A workspace is the most granular container concept in Terraform Cloud. They serve as isolated environments where you can independently apply and manage your infrastructure code. Each workspace maintains its own state file, which tracks the resources provisioned within that specific environment.

This separation allows for distinct configurations, variables, and provider settings for each workspace. By organizing your workspaces under their respective projects, you establish a clear and hierarchical relationship among different infrastructure components. This approach enhances organization, collaboration, and control over your deployments.

In the next few paragraphs, we will illustrate the decomposition of a typical application into multiple modules and lifecycle environments. This application is composed of three infrastructure layers:

1. A network layer
2. A data layer
3. An application layer, with a core component and a module or plugin component.

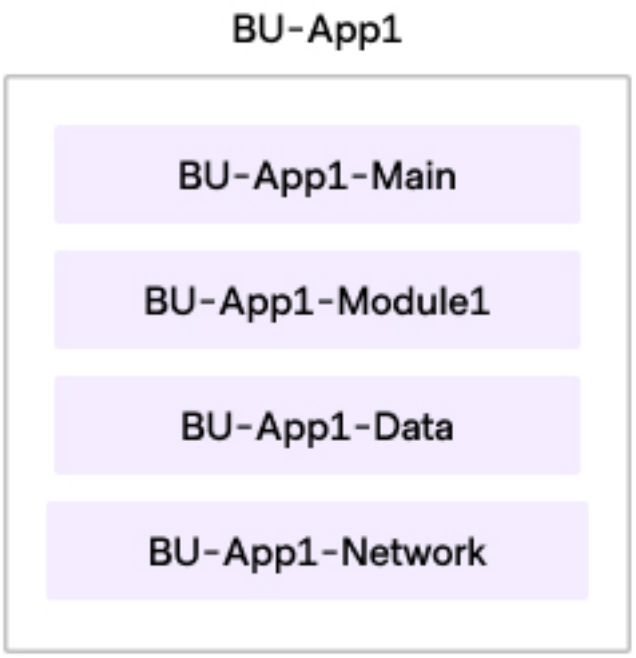


Figure 10: BU has an App called App1 stack broken into distinct workspaces

In Figure 10, the application needs network resources such as VPC, subnets, load balancers, etc., which are provisioned in the `BU-App1-Network` workspace. Any data elements, such as RDS and Elasticache, are in the `BU-App1-Data` workspace. The core of the application, which could include EC2 instances and security groups, will be provisioned in the `BU-App1-Main` workspace. If the application has submodules that need to be managed separately, they could go into an additional module workspace, such as `BU-App1-Module1`.

The above picture is not complete, as we did not take into account the application's environment landscape. Proper change promotion hygiene requires that updates be tested in a lower environment before being deployed in a higher environment. In our example, the environment landscape is composed of three distinct environments:

1. Dev
2. Test
3. Prod

To reflect this environment landscape, the workspaces will look like Figure 11 below:

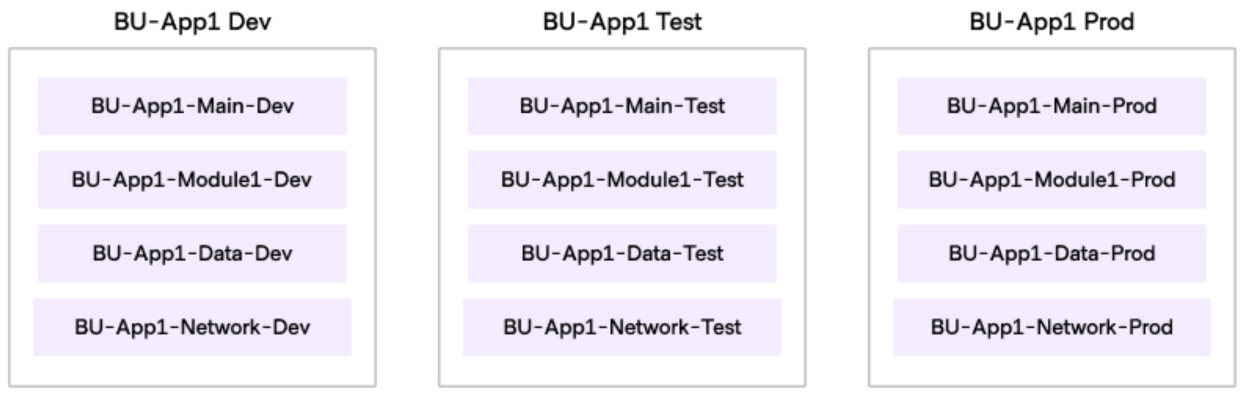


Figure 11: BU's App1 in its 3 distinct environments

Naming convention

We recommend following a naming convention for workspaces to maintain consistency and clarity. The recommended format, with a maximum of 90 characters, is as follows: `<business-unit>-<app-name>-<layer>-<env>`

By incorporating the business unit, application name, layer, and environment into the workspace name, you can easily identify and associate workspaces with specific components of your infrastructure.

Workspace logical boundaries

Above we have shown a design where we have chosen to break down an application stack into smaller individual workspaces. Below, we will discuss the main criteria when deciding how to break down one large workspace into smaller workspaces or potentially grouping together smaller workspaces into a larger workspace.

- **Volatility**: Volatility refers to the general rate of change of resources or modules being used within a workspace. It's essential to avoid adding volatility to unrelated resources. For example, if you have a workspace that creates an AWS VPC and Subnets along with a database instance or EC2 instances, it's likely that the Database or EC2 instances will see more frequent changes and applies than the VPC and Subnets. To maintain stability, we recommend making clear and logical lines between your infrastructure and deciding whether they share common volatility.

- **Privileges**: In Terraform Enterprise, you have the concept of users and teams. Splitting up your workspaces allows for finer access control refinement for different users of your Terraform Enterprise deployment. For instance, if you have one Terraform configuration file for all development databases, you'd want to protect this important configuration from any individual application teams. However, if you choose to split up each database by application or internal team, you can grant relevant access to developers in those teams to interact with their workspace without accidentally breaking another team's infrastructure.
- **Convenience vs Flexibility**: Grouping similar infrastructures together in one workspace can offer some benefits, especially when multiple teams are leveraging the same module or similar infrastructure. However, doing so may limit flexibility when calling modules at specific versions. If you need more flexibility, you can separate configurations or module calls and gain the ability to call modules at specific versions. This flexibility allows for easier upgrades and maintenance of different module versions for various teams.
- **Workspace Efficiency and Performance Considerations**: When working with Terraform, it's essential to optimize workspace efficiency and performance to ensure smooth operations. One crucial aspect to consider is managing the size and complexity of your workspaces.

  Large Terraform Plans and Applies: If you package multiple resources or modules within a single workspace, it can result in a larger graph produced during plan or apply runs. In such cases, you may encounter memory-related performance issues, such as poor performance or Out of Memory (OOM) errors. To mitigate these challenges, consider breaking down the workspace into smaller, more manageable ones. Additionally, having a large number of resources in one workspace can lead to network timeouts or API limits being reached. In such scenarios, it's advisable to create smaller workspaces or adjust the parallelism configuration variable in the Terraform CLI.

  Furthermore, in subsequent plan and apply runs, Terraform performs a refresh command, which checks every resource and link within the state to determine if it's up to date with the real world. With larger states, these refresh operations can take longer than expected, impacting the overall plan, refresh, and apply performance. The Terraform CLI's parallelism limit controls how many resources can be checked in parallel during these operations. More information on this can be found in the Terraform documentation here.
- **Workspace Concurrency vs Terraform Parallelism**: These are critical considerations when managing multiple workspaces. Workspace concurrency refers to the number of local worker agents that can run in parallel within your Terraform Enterprise instance. The default value

is typically set to 10, allowing ten separate worker agents to execute individual workspaces concurrently. For active/active configurations, the number of concurrent workspaces is multiplied by the number of active nodes (up to a maximum of five nodes in active/active setups).

On the other hand, Terraform parallelism is an internal concurrency limit within the Terraform CLI. It governs the number of resources that can be checked in parallel during plan, refresh, or apply commands. The default parallelism value is also set to 10, meaning that if you have 100 resources in a state file, ten will be checked initially, and additional checks will proceed sequentially. If you intend to increase parallelism, be mindful of the increased CPU demand on the docker containers, which may lead to performance issues. Resource-intensive workspaces, especially those with a large number of databases or machines, may require additional CPU cycles. Profiling the workspaces for resource-hungry behavior and considering breaking down configurations into smaller, more manageable units can improve efficiency and performance.

By carefully considering the above criteria and taking appropriate actions, you can design well-structured workspaces that enhance performance, security, and collaboration within Terraform Enterprise or Terraform Cloud. Proper workspace management ensures smooth execution and effective management of your infrastructure.

4.3 Managing Terraform Cloud configuration as code

In the previous section, you explored how Terraform organizations, projects, and workspaces fit together, and you learned about their permissions and best practices. Now, as you consider managing multiple BUs with multiple applications and tens of teams, the question arises: How can you establish seamless workflows at scale? HashiCorp recommends adopting the same workflow used for infrastructure provisioning to manage Terraform Cloud and Terraform Enterprise efficiently.

Control workspaces are a powerful concept in Terraform Cloud and Terraform Enterprise, allowing you to automate the provisioning and management of the platform's configuration. Terraform Cloud exposes an API and HashiCorp maintains a Terraform provider that allows you to tap into this API to effectively manage TFC/TFE configuration as code. A control workspace is a workspace that hosts Terraform code used to configure your TFC/TFE organization, including the provisioning of projects, workspaces, teams, and more. While we will cover essential aspects related to Control Workspaces, such as VCS integration and git-flow workflows in this section, further details will be discussed in subsequent sections to provide a comprehensive understanding.

4.3.1 Scaling Terraform Cloud onboarding with multiple business units

In the rest of this section, we will go over a scenario where the concept of control workspaces is used to delegate management of infrastructure to multiple business units in the organization.

The role of the platform team will be to onboard business units on Terraform Cloud, and in turn, the role of each business unit will be to onboard the application teams working for that BU on Terraform Cloud, carving out the necessary workspaces. This approach decentralizes the responsibility of onboarding new application teams, taking it away from the platform team and giving it to the Business Unit (BU). The main objective of this decentralization is to ensure that the platform team does not become a bottleneck.

4.3.2 Platform team control project & workspace

As a platform team administrator, your responsibility includes managing the `platform-control-ws` control workspace. This control workspace is linked to a version control repository (`platform-control-repo`), which hosts the Terraform code used for automating the provisioning and configuration of the BU control workspace.

We recommend creating a `Platform` project and moving the `platform-control-ws` under that project. Should you need in the future an additional workspace to manage another aspect of Terraform Cloud, you'll be able to leverage the variable set(s), RBAC, and more already configured in the project.

The code below shows how to set up the project, assuming that the organization name is passed as a workspace variable (`var.organization_name`).

```
resource "tfe_project" "platform" {
    name         = "Platform"
    organization = var.organization_name
}
```

When a BU (Business Unit) wants to be onboarded, they initiate a pull request to the Platform team's repository. This action triggers automation for the following tasks:

- Creation of the BU1-Control repository scaffolding.
- Linking of the BU1 control workspace to the control repo.
- Generation of a token for the BU1 control workspace.

- Creation of the projects required by the BU for their app team workspaces.
- Configuration of cloud credentials for the child workspaces requested by the App teams (details about these child workspaces will be covered in more depth during the landing zone discussion).

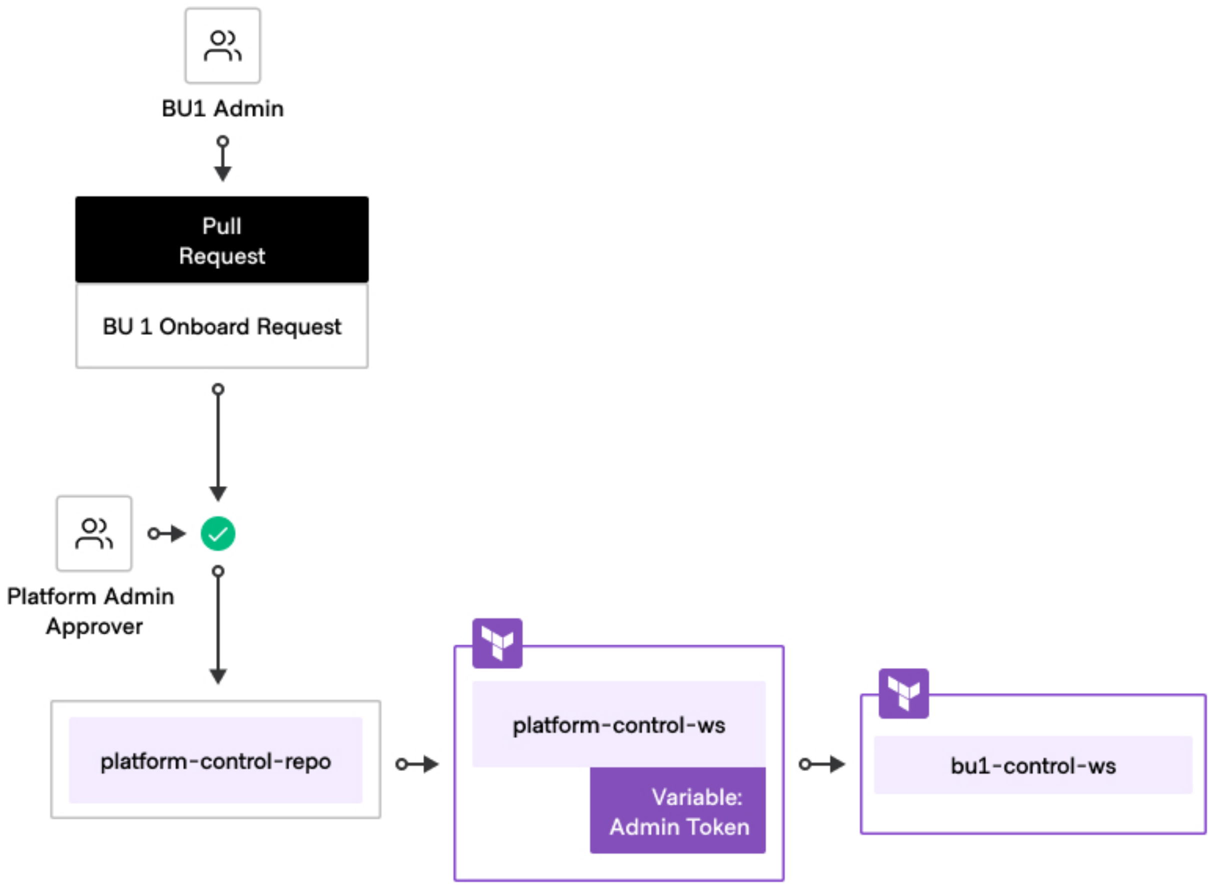


Figure 12: Control workspace workflow to provision BU's admin control workspace

4.3.3 Business unit control workspace

The `bu1-control-ws` control workspace is managed by the business team administrators who have to approve any requests made by the application team. This control workspace is linked to a `bu1-control-repo` which hosts Terraform code used to provision and configure requests from the application team.

The terraform projects are created in the `platform-control-repo` during the BU1 onboarding (further projects can be added if necessary by making additional pull requests to the repo) process. and within the `bu1-control-ws` workspaces are created for the App Team . The App Team has full flexibility to create workspaces within the confines of the various projects available for them.

For example, when a member of the `application1` team wants to onboard their project, they will make a request for the following items:

- BU application team repo scaffolding.
- Associated branches.
- Branch protections.
- Folder structure.
- BU application team workspaces.
- Environment-specific workspaces.
- Cloud credentials in the workspaces.
- Connect workspace to the repository.

The diagram below illustrates the workflow used by application teams to provision objects in Terraform Cloud and Terraform Enterprise.

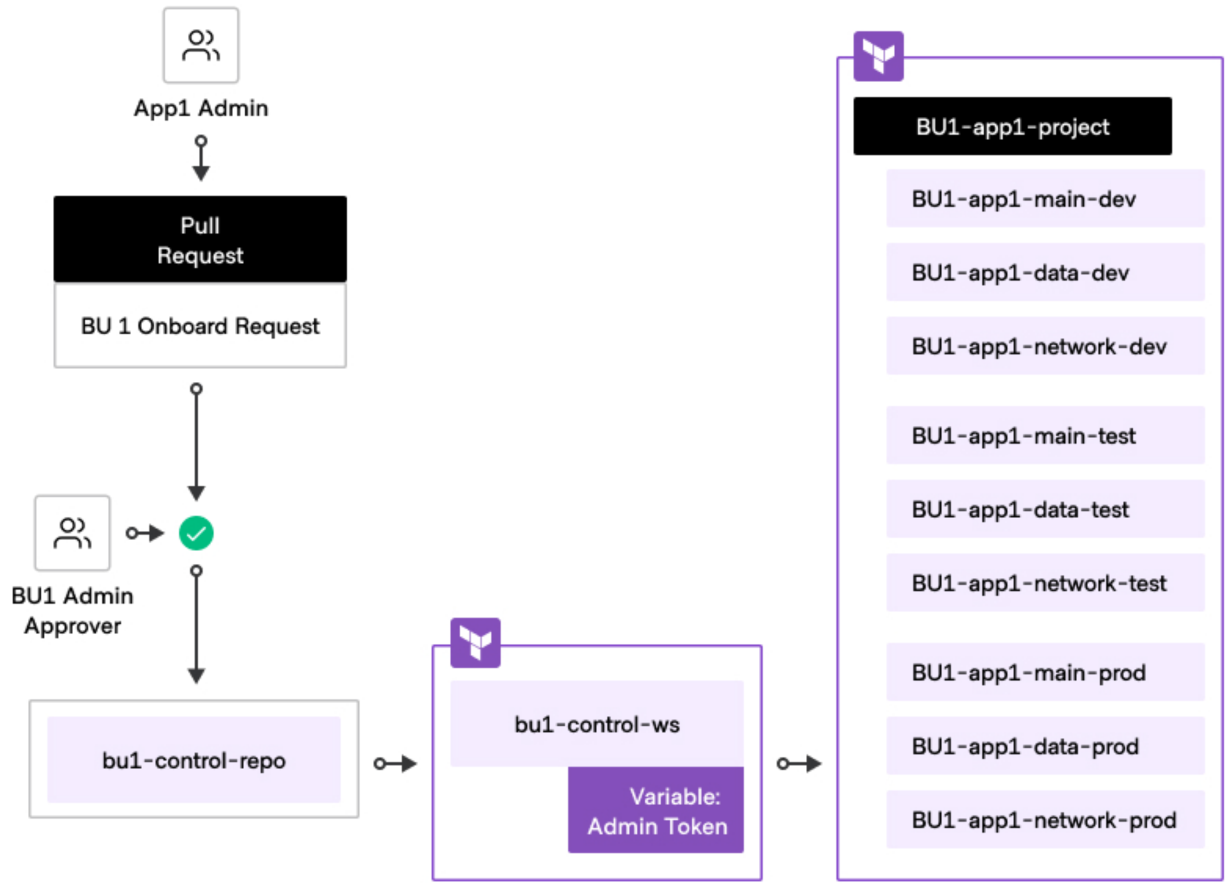


Figure 13: Provisioning workspaces for application team using control workspace

Implementing control workspaces can offer robust automation concepts for onboarding, yet Terraform Cloud/Enterprise also have a powerful API which can be utilized to achieve the same results.

4.4 Terraform Enterprise components

4.4.1 Post-installation initialization

Automated installation of Terraform Enterprise officially ends at the creation of the Initial Admin Token Creation (IATC), but many initialization tasks carry over after installation is completed. Once the IATC is created and usable, the solution can be used to initiate runs via the API. However, it's considered best practice to proceed further in the configuration to get the most value out of the solution. There are two viable options for post-provisioning configuration in Terraform Enterprise: API scripts or using the Terraform Provider. While both methods work, it's generally best practice to use the Terraform provider to codify the objects in Terraform, gaining all the same benefits of Infrastructure as Code (IaC) that you are leveraging Terraform for. All of the objects, definitions, and primitives below can be accomplished via the API, so if you wish to script using bash, translation is fairly straightforward.

4.4.2 SSO and teams

The setup of SSO for Terraform Cloud is not covered in detail in this document, as configurations can vary depending on the SAML 2.0-compliant provider used.

- You can find specific instructions for popular identity providers here.
- For general SAML 2.0 documentation, here.

We recommend automating the creation of teams alongside projects and workspaces using the Terraform Enterprise Provider within Terraform. After creating the teams, you can automate the addition of users within an Identity Provider via SSO. By adding a Team Attribute Name (default: MemberOf) attribute within your IdP, you can automatically assign users to the created groups in the SAML configuration. You can find further details here.

Note: For user accounts used by pipelines these should be flagged in SAML using IsServiceAccount set to true. See isserviceaccount.

4.4.3 GitHub

When your organization utilizes GitHub for version control, you have the option to create a GitHub App using a site admin account. This GitHub App can then be used across all organizations established in Terraform Enterprise. For detailed instructions on setting up the GitHub App, please refer to the following, here.

4.4.4 Logs, metrics and audit logs

This is for configuring logs/metrics/audit only See the audit/logging section at the end of this document.

4.4.5 Audits

In Terraform Cloud (TFC), the Audit Trails API provides access to a stream of audit events that detail changes made to application entities, such as workspaces, runs, and more. On the other hand, Terraform Enterprise (TFE) lacks a specific Audit Trials endpoint but does offer the capability to forward logs for monitoring and auditing purposes. Here are the key differences between Terraform Cloud and Terraform Enterprise:

Terraform Cloud	Terraform Enterprise
What?	
Terraform Cloud has an endpoint for pull based Audit Trails which is a Terraform Cloud Plus feature.	Terraform Enterprise does not have any Audit Trails endpoint, it does however have the ability to forward logs.
Who?	
Infrastructure Architect / Devops engineer / Devops admin	System Admins / Devops Admin
How?	
Responsibility of the practitioner is to build a custom pull method or use a third party application like splunk app to create a dashboard view.	Log forwarding is handled within the Terraform Enterprise architecture by a Fluent Bit container. The administrators of the deployment need to use Fluent Bit configuration. Audit Logs are a part of the log-stream (alongside the other application logs), the administrators would need to use filtering on [Audit Log].
Reference	
Audit Trail API	Terraform Enterprise Log Forwarding

4.4.6 Terraform Cloud (TFC) agents

We recommend deploying TFC agent and configuring your workspaces to use the Agent execution mode. This will result in the agents running all Terraform runs (plan and apply). Both Terraform Cloud and Terraform Enterprise can leverage TFC agents.

The same TFC agent binary can be used with both Terraform Cloud and Terraform Enterprise.

Scalability and availability

Terraform Cloud Agents allow Terraform Enterprise to communicate with isolated, private, or on-premises infrastructure. By deploying lightweight agents within a specific network segment, you can establish a simple connection between your environment and Terraform Enterprise which allows for provisioning operations and management.

TFC agent instances are grouped in agent pools. Configure a workspace to use a particular pool, and any agent in the associated agent pool can complete the run. By adjusting the number of agents in the pool and their location, you can address your scalability and availability requirements.

You may choose to run multiple agents within your network, up to the organization's purchased agent limit. Each agent process runs a single Terraform run at a time. Multiple agent processes can be concurrently run on a single instance, license limit permitting.

Compatibility

Agents currently only support x86_64 bit Linux operating systems. You can also run the agent within Docker using our official Terraform Cloud Agent Docker container.

Agents support Terraform versions 0.12 and above. Workspaces configured to use Terraform versions below 0.12 cannot select the agent-based execution mode.

When running TFE, consult your TFE version release notes for TFC agent version restrictions for your target TFE version.

Rolling out a new TFC agent version

When HashiCorp releases a new version of TFE, we test it against the most current TFC agent version.

Should you need to upgrade your TFC agent version, we recommend testing the new version in a separate agent pool before rolling it out to your main agent pool(s).

TFC agent follows the semantic versioning standard, and the version number change indicates the extent of the modifications.

If you're considering upgrading to a TFC agent version released after your currently running TFE version, testing the new TFC agent is not a step that should be skipped, and the extent of the testing effort must match the extent of the change. Patches and minor changes require little testing, while major changes require more extensive tests.

The diagram below illustrates our recommended rollout approach for a new TFC agent version:

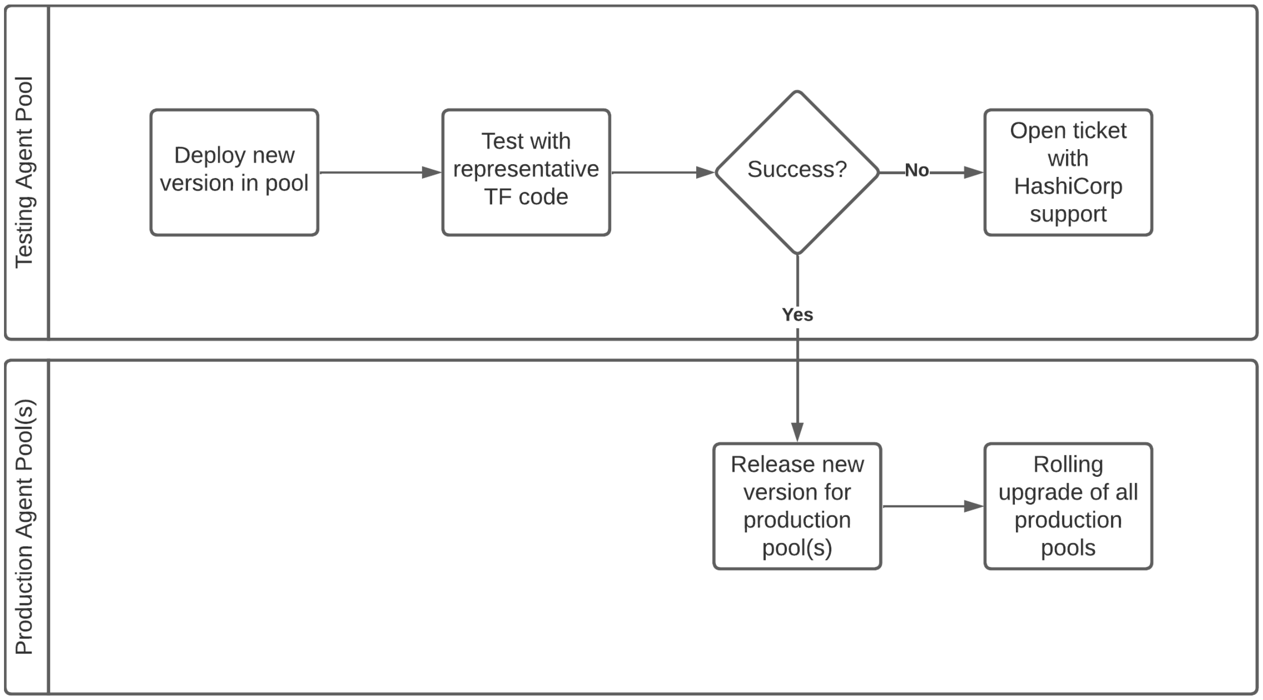

Testing Agent Pool
Deploy new version in pool
Test with representative TF code
Success?
No
Open ticket with HashiCorp support
Yes
Production Agent Pool(s)
Release new version for production pool(s)
Rolling upgrade of all production pools

Figure 14: Rolling out a new version

Resource requirements

The host running the agent has varying resource requirements depending on the workspace. A host can be a dedicated or shared cloud instance, virtual machine, bare metal server, or a container. You should monitor and adjust memory, CPU, and disk space based on each workspace's usage and performance. The name of your instance type may vary depending on your deployment environment.

Use the following specifications as a reference:

- At least 4 Gb of free disk space. Each run requires the agent to temporarily store local copies of the tarred repository, extracted repository, state file, any providers or modules, and the Terraform binary itself.
- At least 2 Gb of system memory

Network requirement

Agents must be able to make outbound requests over HTTPS (TCP port 443) to the Terraform Enterprise application APIs. These requests may require perimeter networking as well as container host networking changes, depending on your environment.

Additionally, the agent must also be able to communicate with any services required by the Terraform code it is executing. This includes the Terraform releases distribution service, `releases.hashicorp.com` (supported by Fastly), as well as any provider APIs.

The following services run on these IP ranges:

- **Hostname**: Terraform Enterprise FQDN or load balancer FQDN
  - **Port and protocol**: tcp/443, HTTPS
  - **Directionality**: Outbound
  - **Purpose**: Polling for new workloads, providing status updates, and downloading private modules from Terraform Enterprise's Private Registry. Set the FQDN via the `-address` CLI flag or `TFC_ADDRESS` environment variable.
- **Hostname**: `registry.terraform.io`
  - **Port and protocol**: tcp/443, HTTTPS
  - **Directionality**: Outbound
  - **Purpose**: Downloading public modules from the Terraform Registry
- **Hostname**: `releases.hashicorp.com`
  - **Port and protocol**: tcp/443, HTTPS

  - **Directionality**: Outbound
  - **Purpose**: Updating agent components and downloading Terraform binaries

- **Hostname**: `archivist.terraform.io`
  - **Port and protocol**: tcp/443, HTTPS
  - **Directionality**: Outbound
  - **Purpose**: Blob storage

Operational considerations

Agents do not guarantee a clean working environment per Terraform execution. Each execution occurs in its temporary directory with a clean environment, but references to absolute file paths or other machine states may cause interference between Terraform executions. We strongly recommend writing your Terraform code to be stateless and idempotent. Consider using single-execution mode to ensure your agent only runs a single workload.

The agent runs in the foreground as a long-running process that continuously polls for workloads from Terraform Enterprise. An agent process may terminate unexpectedly due to a variety of reasons. We strongly recommend pairing the agent with a process supervisor to ensure that it automatically restarts in case of an error. On a Linux host, this can be done using a Systemd unit file like the following:

```
[Unit]
Description="HashiCorp Terraform Cloud Agent"
Documentation="https://www.terraform.io/cloud-docs/agents"
Requires=network-online.target
After=network-online.target

[Service]
Type=Simple
User=tfcagent
Group=tfcagent
ExecStart=/usr/local/bin/tfc-agent
KillMode=process
KillSignal=SIGINT
Restart=on-failure
RestartSec=5
EnvironmentFile=-/etc/sysconfig/tfc-agent

[Install]
WantedBy=multi-user.target
```

Download and install the agent

1. Download the latest agent release, the associated checksum file (`.SHA256sums`), and the checksum signature (`.sig`).
2. Verify the integrity of the downloaded archive, as well as the signature of the `SHA256SUMS` file using the instructions available on HashiCorp's security page.
3. Extract the release archive. The unzip utility is available on most Linux distributions, and you can invoke it by running `unzip <archive file>`. The unzip command extracts two individual binaries (`tfc-agent` and `tfc-agent-core`). These binaries must reside in the same directory for the agent to function properly.

Start the agent

To start the agent and connect it to a Terraform Cloud agent pool:

1. Retrieve the token from the Terraform Cloud agent pool you want to use.
2. Set the `TFC_AGENT_TOKEN` environment variable.
3. (Optional) Set the `TFC_AGENT_NAME` environment variable. This name is for your reference only. The agent ID appears in logs and API requests.

```
export TFC_AGENT_TOKEN=your-token
export TFC_AGENT_NAME=your-agent-name
./tfc-agent
```

Once complete, your agent and its status appear on the Agents page in the Terraform Enterprise UI. Workspaces can now use this agent pool for runs.

Optional configuration: Run an agent using Docker

Alternatively, you can use our official agent Docker container to run the agent.

```
docker pull hashicorp/tfc-agent:latest
docker run \
  -e TFC_AGENT_TOKEN=your-token \
  -e TFC_AGENT_NAME=your-agent-name hashicorp/tfc-agent
```

This Docker image executes the `tfc-agent` process as the non-root `tfc-agent` user. For some workflows, such as workflows requiring the ability to install software using `apt-get` during `local`

`-exec` scripts, you may need to build a customized version of the agent Docker image for your internal use.

```
FROM hashicorp/tfc-agent:latest
USER root
# Install sudo. The container runs as a non-root user, but people may rely on
# the ability to apt-get install things.
RUN apt-get -y install sudo
# Permit tfc-agent to use sudo apt-get commands.
RUN echo 'tfc-agent ALL=NOPASSWD: /usr/bin/apt-get , /usr/bin/apt' >> /etc/sudoers.d
    /50-tfc-agent
USER tfc-agent
```

An image customized in this way permits the installation of additional software via `sudo apt-get`
.

Optional configuration: Single-execution mode

You can also configure the agent to run in single-execution mode, which ensures that the agent only runs a single workload, then terminates. You can use this configuration in combination with Docker and a process supervisor to ensure a clean working environment for every Terraform run.

To use single-execution mode, start the agent with the `-single` command line argument.

Stop the agent

We strongly recommend that you only terminate the agent using one of these methods. Abruptly terminating an agent by forcefully stopping the process or power cycling the host does not let the agent deregister and results in an Unknown agent status. Abrupt termination may cause further capacity issues. Refer to capacity issues for details.

The agent maintains a registration and a liveness indicator within Terraform Cloud during the entire course of its runtime. When an agent retires, it must deregister itself from Terraform Cloud. The agent de-registers automatically as part of its shutdown procedure in the following scenarios:

- You enter `Ctrl-C` in an interactive terminal.
- The agent process ID receives one of `SIGINT`, `SIGTERM`, or `SIGQUIT`. It is important to send only one signal. The agent interprets a second signal as a forceful termination signal and exits immediately.

After initiating a graceful shutdown by either of these methods, the terminal user or parent program should wait for the agent to exit. The amount of time this exit takes depends on the agent's current workload. The agent waits for any current operations to complete before deregistering and exiting.

Agents on Terraform Enterprise

- **No restriction on Agent Count**: Terraform Enterprise does not place a limitation on the number of Agents that can be registered per organization.
- **Hostname Registration**: Terraform Cloud Agents registering with a Terraform Enterprise instance must define the Terraform Enterprise hostname via the -address CLI flag or TFC_ADDRESS environment variable when running tfc-agent. By default, tfc-agent will attempt to connect to Terraform Cloud, so the value must be explicitly defined when registering with a Terraform Enterprise entry point, or the Load Balancer FQDN.
- **Custom Bundle Support**: Terraform Cloud Agents on Terraform Enterprise support custom Terraform bundles. Custom bundles are created and defined within the Terraform Enterprise application; Agents will download the custom bundle based on the Terraform version information.
- **Network Access Requirements**: Terraform Cloud Agents on Terraform Enterprise must be able to communicate with the Terraform Enterprise instance via HTTPS. Additionally, the agent must also be able to communicate with any services required by the Terraform code it is executing. This includes the Terraform releases distribution service, releases.hashicorp.com, as well as the Terraform provider registry. Agents executing in a workspace that leverages a Terraform version that provides a custom Terraform bundle with pre-existing provider binaries do not need access to these resources.
- **Agent Version Compatibility**: Terraform Enterprise places restrictions on what versions of Terraform Cloud Agents can be registered. This is to prevent an incompatible agent from registering with a Terraform Enterprise instance and attempting to execute a Terraform operation in an undefined way. Compatible versions of Terraform Cloud Agents on Terraform Enterprise will vary based on the specific Terraform Enterprise release sequence; any changes to compatible Terraform Cloud Agents versions will be noted in the Terraform Enterprise release notes.

5 Infrastructure-as-code and cloud provisioning

Terraform is an Infrastructure as Code (IaC) tool that enables you to manage and build your infrastructure using codified definitions instead of manual processes. With Terraform, you can interact with various APIs through Providers, which are plugins that communicate instructions to create entities. Common examples of Providers include Azure, AWS, Google Cloud Platform, Kubernetes, and Oracle, while more unique examples include Akamai, Cloudflare, Cobbler/Foreman (for bare metal provisioning), and PrismaCloud.

As a coding language/DSL, Terraform offers flexibility, but it also benefits from prescriptive patterns that have emerged over time. This composition section presents dogmatic patterns that promote straightforward, scalable, and value-oriented consumption of Terraform. By following these patterns, you can leverage the full potential of Terraform and position yourself, your team, and your organization to adopt and derive value from Terraform Cloud or Terraform Enterprise.

The purpose of this section is to provide you with a foundational understanding of the fundamental concepts and terminology. Familiarizing yourself with these concepts is crucial before diving into the composition recommendations and guidance.

5.1 Accelerated basics and definitions

Terraform infrastructure is defined as code written in HashiCorp Configuration Language (HCL) built on top of JSON. As a user, you author Terraform code in a text editor on your local machine and then use the Terraform binary to plan and apply that code to the target system.Terraform code is typically written in text files with the extension ".tf". One commonly used file is "main.tf," which serves as a central location to define all the entities you want to create. This file resides in a non-recursive directory, which acts as the fundamental container for your code moving forward.

When starting to write Terraform code, the first step is to instantiate one or more Providers. Providers are responsible for interacting with APIs and managing the lifecycle of the associated resources.The next step is to determine which Resources you want to codify with Terraform. A Resource Block describes one or more infrastructure objects, such as virtual networks, compute instances, or higher-level components like DNS records. By defining Resource Blocks in your code, you can precisely specify the desired state of your infrastructure.

While the file structure and items mentioned above are straightforward, personal preferences can influence decision-making. To ensure reusability, clarity, and standardization, consider the following guidelines:

- Naming `.tf` files: Determine a consistent naming convention for your `.tf` files.
- Resource ordering: Establish a preferred order for writing resources in your configuration files.
- Inputs and outputs: Define inputs and outputs in a clear and structured manner.
- Code reusability: Explore the use of modules to facilitate code reuse.
- Code sharing: Determine the process for sharing your code with others.
- Consumer usage: Document guidelines on how other consumers can effectively utilize your code.

When transitioning to a more mature development model, utilizing a Version Control Solution (VCS) becomes essential. A VCS enables source code management, collaboration, and streamlined review processes.This topic is covered below in this section. As the usage of Terraform expands to involve more developers, contributors, and reviewers, standardization becomes crucial. Fortunately, Terraform addresses these concerns inherently.

Terraform leverages a Directed Acyclic Graph (DAG) to establish resource relationships, allowing for parallel processing and simplified code structure. Consequently, the order of code execution becomes irrelevant to the underlying system. This approach offers numerous benefits:

1. Resource instantiation order: Terraform handles the sequencing of resource instantiation.
2. File organization: Terraform combines all `.tf` files within a directory, enabling the division of large Terraform files into smaller, more manageable logical files. For instance:
   - `main.tf`: Central location for Providers and custom logic.
   - `frontend.tf`: Contains frontend resource configurations and definitions.
   - `storage.tf`: Houses storage-related configurations.

Additionally, Terraform provides built-in support for input and output management through two conventionally named files:

1. `variables.tf`: Contains variable blocks in alphabetical order.
2. `output.tf`: Includes output blocks in alphabetical order.

5.2 Organizing and structuring Terraform

5.2.1 Overview

"Cloud computing provides new opportunities to deploy scalable applications in an efficient way, allowing enterprise applications to dynamically adjust their computing resources on demand. In this paper we analyze and test the microservice architecture pattern, used during the last years by large Internet companies like Amazon, Netflix and LinkedIn to deploy large applications in the cloud as a set of small services that can be developed, tested, deployed, scaled, operated and upgraded independently, allowing these companies to gain agility, reduce complexity and scale their applications in the cloud in a more efficient way."

The above citation is an abstract from an early research paper on the transition from monolithic enterprise application stacks to microservices. Terraform, as an Infrastructure as Code (IaC) tool, follows this paradigm shift towards microservices. When writing Terraform code, it is important to adhere to the same principles: highly scoped components, simplicity, composability, and pragmatism. This approach improves velocity, reduces impact, and enhances overall consumption. It aligns with the functions and features of Terraform Cloud and Terraform Enterprise, maximizing the value of these solutions.

In the context of Terraform repositories, it is advisable to avoid monolithic repositories. Instead, follow a modular structure. An example of a single Terraform repository is provided below, where customizable naming conventions are denoted with <`example`>:

- <`MyAppStack`>
  - `main.tf`: Contains provider definitions for the entire scope of the Terraform repository and calls to child modules or submodules.
  - <`component`>`.tf`: Individual files for components or resource types, often separated for better organization.
  - `variables.tf`: An alphabetical list of variables, including sensible defaults and descriptions.
  - `output.tf`: An alphabetical list of outputs that provide information for other utilities and Terraform consumption, similar to return values in programming languages.
  - `README.md`: A critical file that introduces the module's purpose and provides a comprehensive understanding of the Terraform code, usage, and variable descriptions.

  - `CHANGELOG.md`: A log file documenting changes made to the Terraform code. Refer to https://keepachangelog.com/ for recommendations on maintaining a changelog.
  - `backend.tf`: Defines where Terraform stores its state data files, which are used to track managed resources. Non-trivial configurations integrate with Terraform Cloud or use a backend to store state remotely, enabling collaborative work on infrastructure resources.
  - `terraform.tf`: Describes the Terraform and provider versions compatible with each module.
  - `/modules`: Directory for holding submodules.
  - `/examples`: Directory showcasing working examples of how to use the module.
  - `/tests`: Directory for containing tests for Terraform modules.

When organizing Terraform code files, it is recommended to follow a relatively linear order that aligns with the logical flow of the provisioning process. Although Terraform itself can execute in parallel, it's important to structure the code in a way that is easily understandable for humans. Here are some general best practices to consider:

- Define dependent resources before those that depend on them to ensure proper ordering.
- Place provider configurations at the top of the file to set the context for the resources.
- Include globally used items, such as randomly generated strings for naming or globally useful data sources, after the providers.
- Utilize locals to define and manage variables that are used within the file.
- Arrange the Terraform code files in a way that logically represents the building process.
- Group similar resources together, such as IAM policies for the same resources, while keeping distinct sections separate.

By organizing the code in a structured and logical manner, you can enhance readability and maintainability, making it easier to understand and manage your Terraform configurations.

5.2.2 Code formatting

The logical structuring of Terraform code is closely tied to adhering to Terraform's idiomatic style conventions, which promote consistency and readability. To ensure consistency across configuration files and modules, it is recommended to use automatic source code formatting tools like `terraform fmt`. These tools apply the style conventions automatically, enabling you to maintain a clean and organized codebase. Below are the key style conventions for formatting your Terraform code:

- Indentation: Use two spaces for each nesting level.
- Curly Brace Placement: Place the closing brace of non-empty, multiline curly braces or square braces on its own line.
- Snake Case: Use Snake Case for terraform resource/datasource/local names.
- Alignment: Align equals signs for single-line arguments at the same nesting level.

Example:

```
resource "aws_instance" "example" {
  count = 2 # Meta-argument first

  ami           = "abc123"
  instance_type = "t2.micro"

  network_interface {
    # ...
  }

  lifecycle { # Meta-argument block last
    create_before_destroy = true
  }
}
```

- Block Separation: Separate top-level blocks with one blank line. Separate nested blocks with blank lines, except when grouping related blocks of the same type.
- Block Grouping: Avoid separating multiple blocks of the same type with different blocks, unless they form a defined family.

Comments are crucial for understanding complex code but should not be overly verbose. Terraform itself serves as documentation, so it's recommended to follow concise and clear comment guidelines. Below are the recommended guidelines for using comments:

- Consider the audience when writing comments in your Terraform code. It is recommended to write code that is understandable to someone with a solid understanding of Terraform, but not necessarily complete mastery. Avoid sacrificing information in the comments by catering sole-ly to the least experienced readers.

- Comments should provide additional details, such as the "why" behind certain decisions, rather than restating information already evident from the resource itself.

- Maintain consistency in the types of comment delimiters used. The hash symbol (#) is preferred over double slashes (//). Multiline comments (/**/) should be used sparingly, primarily for longer comments or when commenting out resources.

- Metadata often requires comments to provide additional context. For example, lifecycle or count statements may require an explanation of their purpose.
- Complicated dynamic code, such as loops and counts, almost always benefits from comments to aid understanding.
- A well-written Terraform code should be understandable within a minute by someone proficient in Terraform but unfamiliar with the specific codebase.
- Variables should be in a separate file, provide a sane default value and validate input with informative error messages

5.3 Introducing modules

To scale and optimize the consumption of Terraform, the necessary next step is to introduce 'components' of infrastructure called modules. Modules in Terraform serve a similar purpose to library functions in other programming languages. They provide consistent and standardized results for a specific component, ensuring uniformity across infrastructure deployments. Modules also contribute to efficiency by promoting standardization and enabling resource creation with predictable outcomes. At the same time, modules establish boundaries for customization, allowing users to control inputs and outputs within predefined limits. This approach strikes a balance between flexibility and stability, ensuring that infrastructure can be customized while maintaining overall consistency.

To ensure efficient management and reuse of resource configurations in Terraform, modules are used as containers for multiple resources that are designed to be used together. Understanding the different types of modules is essential. Please review the definitions below:

- **Module**: A module consists of `.tf` or `.tf.json` files organized within a directory and intended to be managed in version control. The directory may also contain subfolders for related artifacts such as `*.ci`, `.gitignore`, and `*.md` files, used for Terraform code parsing, version control, or workflow.
- **Root Module**: The root module is constructed from the configuration files found in the current working directory when Terraform is executed. This module may reference child modules located in other directories or sourced remotely, allowing for the creation of a hierarchical structure.

- **Child Module**: A child module is not typically run directly from the Terraform CLI but is instead consumed by another module. It focuses on completing a specific function to create a "component" used within the broader infrastructure configuration.
- **Generalized Child Module**: A specialized type of child module designed for broad consumption. Well-written Terraform Public Registry modules often fall into this category. Generalized child modules provide a wider range of inputs and outputs and incorporate switching logic within the code to accommodate various use cases for the specific component.

In the world of programming, maturity can be compared to a large fast food franchise. Initially, a concept or small experiment, it eventually demands standardization and repeatability for scalability. Franchisees adhere to a strict contract model with robust supervision, aiming to serve customers the same product globally. Terraform naturally aligns with this pattern, providing exponential value. To structure your code and modules effectively, consider the following key questions:

- How are the boundaries of responsibility defined within your organization for owning specific components?
- Are there opportunities to break down barriers and build collaborative co-authoring teams for cross-functional requirements? For example, can a compute team, storage team, and web team come together to build a "full-stack" module?
- What opportunities exist in your organization for standardization that you can enforce through code? -Can you develop an optimized or express path to encourage further standardization?
- Which exceptions fall outside the scope of automation or are unnecessary to consider?
- What are the significant automation opportunities in your organization, and what are the small opportunities? Automating a "full-stack" deployment used by more than 30% of your company holds exponentially more value than automating a single application's unique stack.

5.3.1 Module best practices

The defining characteristic of a module is its encapsulation within a folder. However, it's important for you to evaluate the value and usefulness of creating a module, rather than doing so out of convenience. Consider the following fundamental principles of modules:

- Define a Distinct Architectural Concept: Modules raise the level of abstraction by describing a new concept in your architecture, constructed from resource types offered by providers. For example, "acmecorp_standard_application_stack" could represent a suitable module concept.

- Contains complex or business-specific requirements: When creating modules, they should cover multiple resources while staying focused and proportionate in scope. If there are complex or business-specific requirements, it's recommended to create modules for standardization. For example, an "aws_db_module" can address various tasks like managing backups, configuring CloudWatch monitoring, setting up alerts, log ingestion, and enabling Performance Insights. However, for cases where standardization is unnecessary, you can handle the requirements through policy implementation. Modules representing application stacks such as LAMP Stack, Landing Zone or Project Factory are good examples of useful modules. Conversely, modules encompassing the entire Quality Assurance environment, including Github accounts, Ingress/Egress rules, and all compute-related aspects, are considered excessive.
- Embody Best Practices: Modules serve as a platform for enforcing naming conventions for cloud resources. They utilize variables to retrieve information required for constructing cloud resource names (or tags) in accordance with the naming convention. While best practices may be nebulous, subject to interpretation, and subject of debate, HashiCorp Validated Designs recommend specific patterns.

The primary focus of modules is to cater to the needs of the customer. It aims to provide them with the value of lightweight abstractions, considering factors such as impact radius, usability, and convenience. From the customer's perspective, they should only be concerned with providing the necessary inputs and receiving the desired outputs for operations, without having to worry about the internal workings of the module. On the other hand, as writers or operators of modules, you need to ensure that the module follows standard consumption practices, guarantees security and correct deployment of entities. As a business objective, modules should:

- Enable consumers to easily provide inputs and obtain desired outputs related to compute, storage, and networking access.
- Establish a standard and consumable entity for producers, auditors, and security.
- Instill confidence in consumers by offering a standardized component within the enterprise.
- Facilitate rapid development of new shared best practices and infrastructure.
- Separate production and consumption maintenance cycles, allowing for the creation of newer versions while maintaining a versioned object for consumers to utilize.

5.3.2 Module production

The Private Registry in Terraform Cloud and Terraform Enterprise is a crucial component that allows organizations to create, manage, and share reusable modules within their environment. As we've previously discussed the hierarchical structure of organizations, projects, and workspaces, it's essential to understand how the private registry fits into this setup.

Within the confines of a single Terraform Cloud and Terraform Enterprise Organization, the private registry enables teams to publish and consume modules. Organizations serve as the highest-level entity, encompassing multiple projects and their respective workspaces. Projects, on the other hand, provide a structured approach to organizing workspaces and controlling access to resources. Workspaces act as isolated environments where infrastructure code can be applied and managed independently.

The private registry comes into play within this organizational hierarchy, providing a way to share modules securely within an organization. As of now, the private registry's scope is limited to a single organization, meaning that modules can be shared between workspaces within the same organization. However, Terraform Cloud does not support private registry module sharing across different organizations. For Terraform Enterprise users, administrators have the option to configure module sharing across organizations in the Enterprise version. This means that if you have multiple Terraform Cloud and Terraform Enterprise Organizations and wish to leverage the private registry, a separate private registry will need to be created for each Organization, and each shared module must be individually published to each private registry.

While workspaces within a single Terraform Cloud and Terraform Enterprise Organization can cross-reference or source modules from a private registry within the same organization, this process becomes streamlined if you have a supported Version Control System (VCS) connected to Terraform Cloud and Terraform Enterprise. When module source code resides in a singular VCS repository, it becomes easier to manage and update the modules across private registry's.

In such a scenario, each module must be individually published into every private registry as a separate operation initially. Once this is done and provided that the VCS repositories are correctly tagged, the private registry's will automatically identify new versions of the module. However, if Terraform Cloud and Terraform Enterprise is being used with an unsupported VCS, the publishing and updating of all private registry modules must be performed manually using the Terraform Cloud and Terraform Enterprise API, which involves separate operations. This process introduces additional operational overhead.

5.3.3 Module production overview

In this section, we will establish the fundamental principles for module production within an enterprise. Determining the precise boundaries of modules can be challenging, and it is important to base this decision on business and organizational factors rather than solely on Terraform capabilities. To ensure clarity and manageability, Terraform code should have defined owners and lifecycles. When different parts of the organization handle related but distinct tasks, such as creating underlying networking versus compute resources, it is often beneficial to use separate modules. Below is a recap the two key considerations before proceeding:

- Terraform consolidates all dependent folders, resources, and entities, creating an implicit connection within the codebase. This becomes especially evident when modules call other modules.
- Code segmentation should align with an organization's separation of duties and prioritize readability and usability above all else.

To put the previously discussed best practices into action, you will explore a sample software stack called the Acme Application. Through this stack, you'll gain insights into optimizing various components following Terraform's best practices. The Acme Application consists of the following components:

- Virtual Private Cloud (VPC) with ingress and egress rules.
- Frontend compute infrastructure for hosting the web configuration.
- Middleware running on a compute instance.
- Backend utilizing a hosted database provided by a Cloud Service Provider (CSP).

As you review the components of the Acme Application, you can begin to apply your Terraform knowledge to identify areas that can be optimized and transformed into modules by taking into the following steps presented below.

The first component to evaluate is the Virtual Private Cloud (VPC):

1. Determine relevant background information: At Acme Corp, the architecture operates within a single VPC. As of now, there is no need to segregate applications into separate VPCs. Customization is achieved using variables for quality assurance, staging, and production environments. If other entities or components require information or data from the VPC (e.g., allocated IPs), they can access them through Remote State.

2. Ask the question(s): Does this VPC contain complex or business-driven requirements?
3. Make a decision: No, this VPC does not contain complex or business-driven requirements, and there is no need to modularize it.

The second component to evaluate is the compute for Acme Applications' frontend and middleware:

1. Determine relevant background information: At Acme Corp, the compute is customized primarily at runtime and uses a standardized golden image. Most of this configuration is done post-provisioning using a configuration management software. The underlying compute is typically modified via an inlet form from the customer requesting it (for example if someone requests an application, the standard Quality Assurance/Staging environment gets 4G of RAM and 2 CPUs, but Production gets double).
2. Ask the question(s): Does this component represent a distinct architectural concept?
3. Make a decision: No, these components do not represent a distinct enough requirement to modularize and change from a basic resource definition.

The final component to evaluate is the hosted data storage layer:

1. Determine relevant background information: At Acme Corp, their applications operate in a highly regulated and complex data-driven environment. The databases require stringent security measures and regular backups. Additionally, when new features are added to the Acme Application, staging data is copied to a lower environment for thorough developer testing before considering further deployments.
2. Ask the question(s): Does this component represent a distinct architectural concept? Does it encapsulate complex or business requirements? Does it embody best practices?
3. Make a decision: Yes, maybe. This component has rigorous requirements, including complex business needs, but it could potentially be standardized on a single resource. However, considering the constant reprovisioning into a testing environment and the self-service nature of Acme Corp's cloud requirements, modularizing this component could provide benefits.

5.4 Public module registry

If your goal is to understand the basics and kickstart your automation journey, this section will guide you through the final steps to achieve initial success. Terraform benefits from a thriving community with extensive support and resources. Currently, there are over 3000 published providers and 13,000 modules available. Leveraging existing modules and development patterns can provide a valuable shortcut to jumpstart the provisioning process for your organization. The Terraform Registry serves as a public platform where contributions from various users are shared and reviewed. It's worth noting that some modules in the marketplace have been utilized over 50,000,000 times. While it's important to exercise caution and possess a certain level of understanding when using community-contributed resources, they serve as an excellent starting point for organizations venturing into Infrastructure as Code and are highly recommended for initiating your automation journey.

To curate and mirror existing Terraform code from the public Terraform Registry, it is essential to analyze the quality of the code and verify its intended purpose. Follow these guidelines to evaluate whether the code meets basic quality standards and aligns with the desired outcomes.

1. Evaluate the author:
   - Check the "Partner content" checkbox in the Module Registry to filter for stricter requirements and maintenance cycles.
   - Review the author's profile (e.g., Anton) and consider the number of downloads as an indicator of usage and quality.
   - Take note of the author's affiliation and association (e.g., Eric) for certifications or involvement with reputable companies.
2. Evaluate the module: Assess if the module follows a consistent update cadence and adheres to proper semantic versioning.
   - Review the number of downloads for the module, both within the current week, month, year, and overall. Compare these statistics with other modules of the same type.
   - Look for a detailed and exhaustive Readme. A good Readme should provide clear documentation on the module's usage, inputs, outputs, dependencies, and resources, offering comprehensive guidance to end users.
   - Check if the module includes examples. Examples provide detailed demonstrations on how to use the module effectively.

Once you've identified a suitable module for use, you can mirror the module internally with a private registry (more details on Terraform private registry can be found in the forthcoming Terraform Standardize HVD). This method of validating external modules and curating them for internal use can be an extremely useful way to start building a library of objects for consumption in your organization. While it doesn't develop the knowledge base in the same way as learning the rest of the composition best practices, it has to be acknowledged as one of the fastest ways to productivity using open source programming languages. For the lifecycle management of these modules and further best practices, please refer to the rest of this guide.

5.5 Terraform versioning concepts

Semantic versioning is a crucial concept in Terraform that was briefly introduced in the previous section. It provides a structured approach to versioning, ensuring compatibility and predictability when making changes to your Terraform configurations. By following semantic versioning principles, you can communicate the impact of your changes through version numbers:

- MAJOR version when making incompatible API changes
- MINOR version when adding functionality in a backwards compatible manner
- PATCH version when making backwards compatible bug fixes

Terraform supports semantic versioning for the binary, providers, and modules. It is crucial to evaluate compatibility between the Terraform binary and individual providers by referring to their compatibility matrix. For example, versions 4.0.0 and 4.x.x of the AWS Provider will be the last versions compatible with Terraform 0.12-0.15.

When upgrading the Terraform binary, provider versions, and module versions, it is important to avoid making code changes to the codebase (except for changes required to support the upgrade). Isolating changes to individual variables is necessary to prevent cross-contamination of issues. Additionally, it is recommended to review the changes notes while validating your upgrade strategy.

In general, it is advisable to use version constraints on major versions, which restrict large breaking changes to occur only in major versions. The compatibility hierarchy should follow the order of Terraform Version, Provider Version, and then Module Version. The visual representation is below:

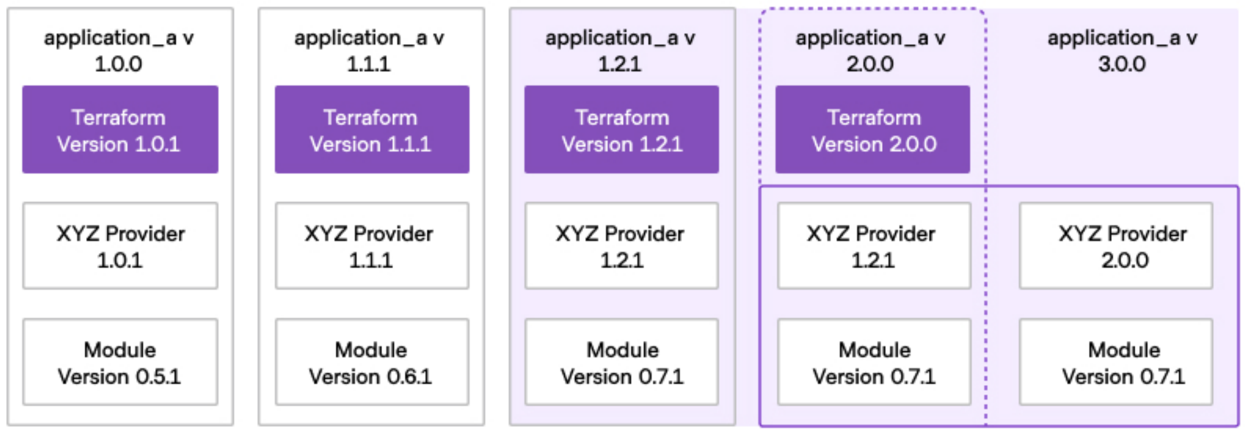


Figure 15: Using version constraints

The practice of branching in Version Control Solutions aligns well with this model. Additionally, speculative plans take advantage of this capability by allowing you to perform plan-only runs against specific branches. For example, you can create a new branch called `application_a_v_newtfversion_xyz` and conduct a speculative run with the next version you intend to introduce.

Utilizing the Dependency Lock File extensively is crucial for successful large-scale upgrades of Terraform Providers. It is important to be aware of certain limitations regarding version constraints, which can be found here.

During the installation of required providers for a configuration, the `terraform init` command considers both the version constraints specified in the configuration and the version selections recorded in the lock file.

By combining the dependency lock file with a standard branching solution in Version Control, it becomes safer and more straightforward to introduce major changes.

5.6 Cloud provisioning

With the foundations of Infrastructure as Code (IaC) established, you can now proceed to the next stage of this operating guide, which focuses on establishing a workflow for cloud provisioning.

Terraform Enterprise provides you with a robust and efficient "golden cloud provisioning" workflow. To successfully provision resources in the cloud, follow these steps:

1. Implement the "Landing Zone Provisioning Workflow": You will implement this workflow to ensure that infrastructure is provisioned in the appropriate "landing zones."
2. Set up a Terraform Workspace with Cloud Credentials: Create a Terraform Workspace that includes the necessary cloud credentials. Alternatively, you can integrate with Vault using Workload Identity to dynamically manage cloud credentials.
3. Store your Terraform Code in a Version Control System (VCS): Use a Version Control System repository to store your Terraform code. In the upcoming Pipeline section, we will discuss the two main alternatives for executing Terraform code.

The "Landing Zone Provisioning Workflow" is a crucial process that you implement for the various business units to ensure infrastructure provisioning in the appropriate "landing zones." Each cloud platform has its own recommendations for implementing these landing zones, and for Kubernetes-based teams, a Kubernetes namespace can also serve as a component of a landing zone. Key outputs of the Landing Zone workflow process include:

1. VCS Repo - Used to store Terraform code
2. Workspace
3. Cloud Credentials stored in Workspace Variables

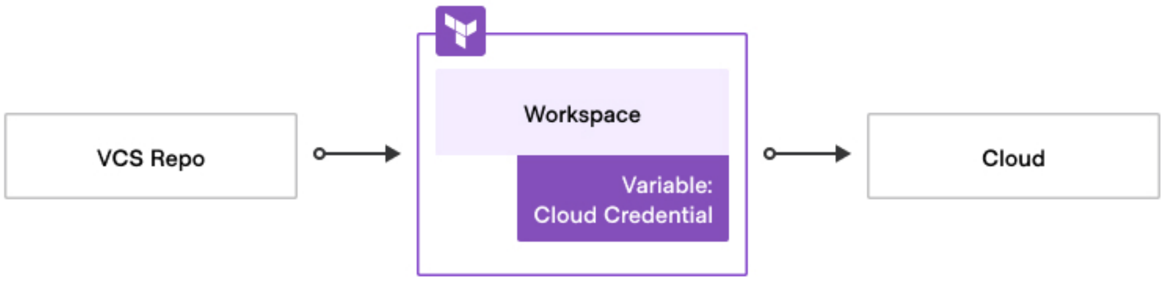


Figure 16: The cloud landing zone workspace

To achieve end-to-end cloud provisioning, developers will commit Terraform code to the designated repository, triggering a plan in both Terraform Cloud and Terraform Enterprise workspaces. The workspaces will utilize cloud credentials, either stored directly within the workspace or integrated with Vault, to perform the Terraform plan. Once approved, the workspaces will apply the plan, provisioning the desired resources in the cloud.

However, before cloud provisioning can commence and Terraform workspaces can be provisioned with cloud credentials, it is crucial that the "Landing Zone Workflow" mentioned in earlier sections is fully implemented.

The Landing Zone Workflow is implemented by you, the platform team, in collaboration with partners in networking, security, and finance, adhering to the guidance provided by the cloud vendors' Well-Architected Framework. The Landing Zone must be implemented by your team before any application teams can initiate infrastructure provisioning in the cloud.

5.7 Cloud landing zones

A cloud landing zone is a foundational concept in cloud computing. It refers to an environment or infrastructure that is set up in the cloud for an organization to facilitate secure, scalable, and efficient cloud operations.

A cloud landing zone provides a secure and compliant environment for an organization's cloud resources, including networking components, identity and access management, security measures, and baseline configurations. This forms a blueprint for setting up workloads in the cloud and serves as a launching pad for broader cloud adoption. Cloud landing zones address several important areas:

- Networking: Configured with the network resources, such as Virtual Private Clouds (VPCs), subnets, and connectivity settings needed for operating workloads in the cloud. This includes setting up appropriate firewall rules, network access control lists, and routes to control inbound and outbound traffic.
- Identity and Access Management (IAM): Helps to enforce IAM policies, role-based access control (RBAC), and other permissions settings. This allows an organization to control who can access their cloud resources and what actions they can perform.
- Security and Compliance: Security configurations like encryption settings, security group rules, and logging settings. This helps ensure that an organization's cloud environment complies with internal policies and external regulations.
- Operations: Includes operational tools and services, such as monitoring, logging, and automation tools. These help to manage and optimize cloud resources effectively.

- Cost Management: Enables cost monitoring and optimization by facilitating tagging policies, budget alerts, and cost reporting tools.

Major cloud providers like Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP) offer solutions to help organizations create and manage landing zones. For instance, AWS Control Tower for creating a landing zone in AWS, Azure Landing Zone Accelerator, and Cloud Foundation Toolkit for GCP.

In essence, a cloud landing zone is a best practice strategy that ensures an organization's cloud journey starts from a secure, manageable, and well-governed foundation.

5.8 Secure variable storage

An important remark about the use of variables:

> Terraform Workspaces contain a high level of privileged information and these are either in the state file, variables or their ability to access sensitive information from external secret management tools. It is possible for these sensitive information to be exfiltrated (by accident or otherwise) either by changing debug log level or using outputs. We recommend that Platform Teams use stringent RBAC controls to control this access.

5.8.1 Importance of variables

One of the the most obvious capabilities to explore when it comes to introducing variables in a secure fashion is Terraform Variable properties and precedence. Terraform Enterprise supports configuring Variables in many different ways (API, CLI, GUI) following many different security policies.

Variables are how terraform users can modify what terraform configuration code produces without having to make underlying changes to the terraform provider itself. Variable Sets are groups of variables created in Terraform that can contain both terraform and environment categories of variables. A variable set can be configured as global for an organization, that applies to all of the workspaces in an organization. Variable sets can also be reused by specified workspaces.

Secure Variables are crucial for storing access credentials used in the Terraform plan/apply lifecycle. These credentials include Cloud Credentials for provisioning, Admin passwords for initializing VMs and databases, and API keys/Tokens for third-party tools.

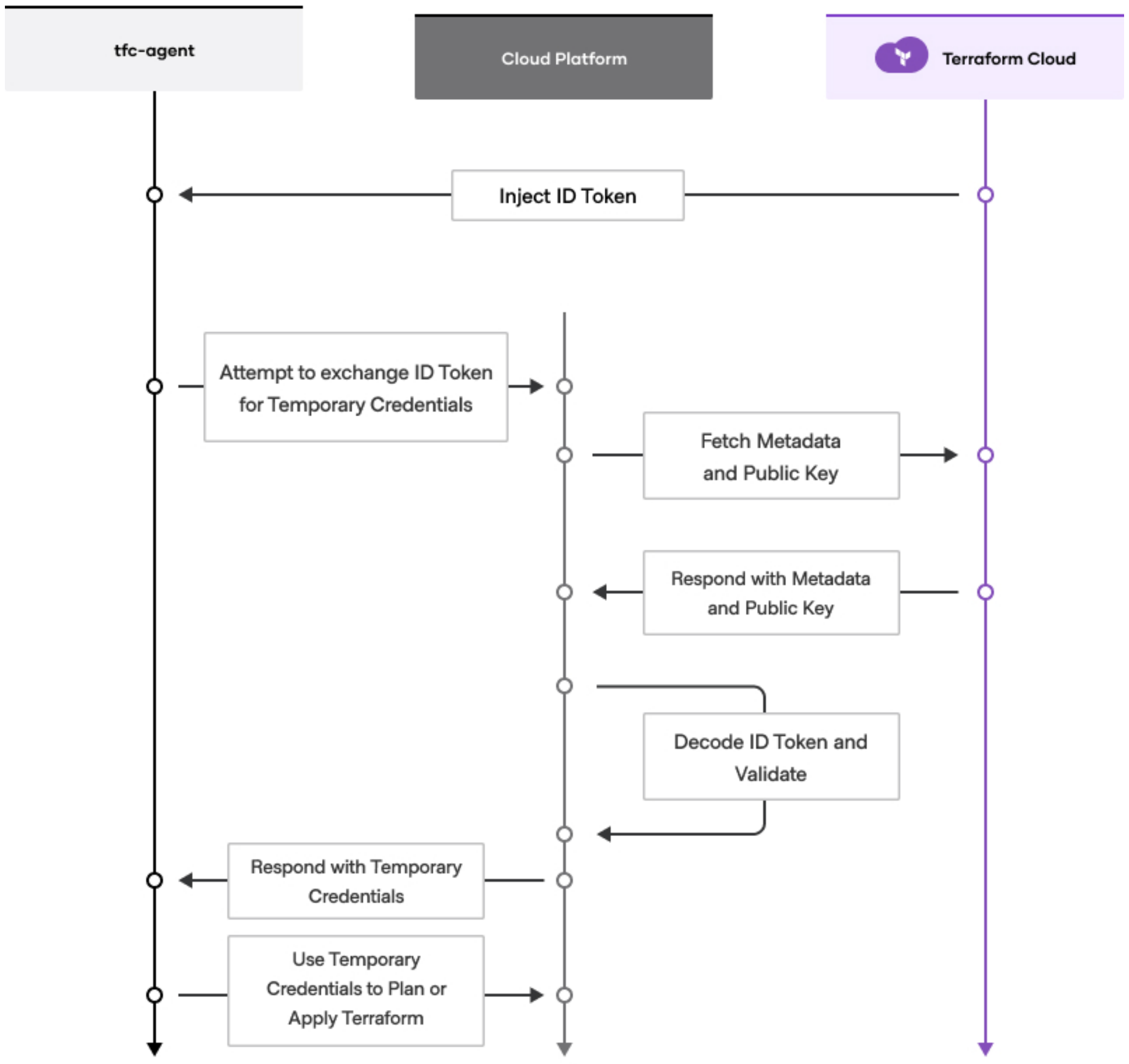
Considerations

- Consider using Variable sets Project level in Terraform Cloud. Benefits are,
  - Projects have granular access to who can view and manage the variable sets.
  - Ease of administration instead maintaining long lists of variables defined individually selected workspaces.
  - Sensitive variables can be defined at the project level and can be inherited to the all or specific workspaces if required.
- Leverage integration with Vault using "Workspace Identity" to enhance security and protection of sensitive data.

5.8.2 Workspace identity

With Dynamic Credentials, users are able to authenticate their Vault providers using OpenID Connect (OIDC) compliant identity tokens to provide just-in-time credentials and eliminate the need to store long lived secrets on Terraform Cloud/Enterprise. Some of the benefits of Workspace Identity include;

- No more "cloud credentials": Removes the need to store long-lived Cloud credentials in Terraform Cloud/Enterprise. Temporary cloud credentials are instead retrieved from a Vault on the fly.
- No more secret rotation logic: Credentials issued by cloud platform are temporary and short-lived, removing the need to rotate secrets.
- Granular permissions control: Allows for using a cloud platform's authentication and authorization tools to scope permissions based on Terraform Cloud metadata such as a run's phase, its workspace, or its organization.


tfc-agent
Cloud Platform
Terraform Cloud
Inject ID Token
Attempt to exchange ID Token for Temporary Credentials
Fetch Metadata and Public Key
Respond with Metadata and Public Key
Decode ID Token and Validate
Respond with Temporary Credentials
Use Temporary Credentials to Plan or Apply Terraform

6 Terraform workflow

In the previous sections, we discussed the significance of IaC and how it allows you to manage infrastructure as easily as software code. By adopting IaC practices, organizations have gained agility, efficiency, and reliability in deploying cloud-native resources. Now we will dive deeper into the topic of integrating Version Control Systems (VCS) and building efficient pipelines for Terraform.

6.1 Streamlining IaC

As you automate Infrastructure as Code (IaC) provisioning and management, you can significantly streamline the process, similar to automating source code build and deployment with Continuous Integration/Continuous Deployment (CI/CD) pipelines. To simplify this automation and integrate with Version Control Systems (VCS) repositories, Terraform Cloud and Terraform Enterprise offer powerful features.

You can take advantage of the *VCS integration* feature in Terraform Cloud and Terraform Enterprise, which uses webhooks to directly integrates with VCS repositories. This allows you to seamlessly implement a gitops workflow without the need for custom pipelines. We highly recommend using this method as it offers a straightforward and efficient approach.

In situations where you prefer to utilize CI/CD pipelines, Terraform Cloud and Terraform Enterprise provide pipeline templates to facilitate the integration. These templates are designed to make it easier to incorporate IaC code from VCS repositories into Terraform Cloud and Terraform Enterprise. Consider using CI/CD pipelines when the VCS integration might not entirely meet your specific requirements. We currently provide templates for GitHub and GitLab, but you can use the underlying workflow tooling to build an integration for other CI/CD solutions.

6.1.1 Overview

Terraform Cloud and Terraform Enterprise (TFE) play a vital role as CI/CD platforms for Infrastructure as Code (IaC), empowering organizations to manage their infrastructure efficiently. They support CI tools by running plans and policy checks to ensure changes meet functional and compliance requirements. They further handle CD by applying changes and conducting continuous validation and drift detection, making them ideal CI/CD tools for infrastructure management.

Embracing Infrastructure as Code allows you to define your infrastructure using source code, akin to traditional software systems. Storing your configurations in version control grants you the ability to conduct audits, testing, and continuous delivery, addressing the complexities of deploying cloud-native resources effectively.

In a constantly evolving infrastructure landscape, responsiveness and reliability are key. When selecting a workflow, consider deployment frequency, code change frequency, and the level of automation required, while adhering to established guardrails.

While our primary focus here is on designing a pipeline workflow using the VCS-driven approach with Terraform Cloud and Terraform Enterprise, we will also briefly touch on CLI-driven and API-driven workflows to provide a comprehensive understanding of Terraform's capabilities. However, the VCS-driven workflow is the recommended path for most organizations due to its seamless integration and collaborative benefits.

6.1.2 Personas

When constructing the workflow, it is advisable to involve only a minimal number of contributors in the pipeline development process. Depending on the size and scale of your organization, certain roles may be combined to take on additional responsibilities.

Persona	Role Description	Roles
Developer	Responsible for developing the Infrastructure and application code.	Software Engineer, Application Developer, Consultant
Lead Developer	Responsible for helping the efforts of the product developers / teams.	Development Lead, Technical Team Lead, Head of Development, Technical Manager
Release Engineer	Responsible for the coordination of a deployment to production (usually using automation)	Release Engineer, Release Manager

Persona	Role Description	Roles
Platform Engineer	Responsibilities include writing pipeline definitions and enabling developers to use the Pipelines.	DevOps Engineer, Operations Engineer, Systems Engineer, IT Consultant
Infrastructure Operator	Responsibilities include maintenance, configuration and administration activities	Systems Engineer, Infrastructure Engineer, Site Availability Engineer, Site Reliability Engineer, System Administrator

6.2 VCS-driven workflow

When it comes to managing your infrastructure as code, Terraform offers a VCS-driven workflow that streamlines the process and enhances collaboration among your team members. This approach automatically triggers runs based on changes made in your VCS repositories, making it easier to keep track of modifications and ensuring that your infrastructure configuration stays up-to-date.

Embracing Terraform's VCS-driven workflow brings with it a host of valuable benefits. By establishing shared repositories as the definitive source of truth, your team can work concurrently on the same infrastructure code, promoting seamless collaboration. Additionally, this workflow eliminates the need for additional Continuous Integration (CI) tooling, simplifying the process and making it more accessible to all team members. Moreover, you can effortlessly associate this workflow with webhooks on your preferred VCS providers, further streamlining the process of triggering runs and ensuring efficient synchronization between code changes and infrastructure updates.

6.2.1 Prerequisite(s)

Before proceeding with the VCS-driven workflow, ensure you complete the following pre-work for each category:

1. VCS repository:

- Set up a Version Control System (VCS) repository that contains the source code for the deployment.

2. VCS authentication:
    - Establish VCS authentication to enable Terraform's access to the repository securely.
3. VCS permissions:
    - Define the required VCS permissions for individuals, specifying appropriate access levels such as read-only, merge, etc.
4. Workspace:
    - Define workspace naming conventions to maintain consistency.
    - Define workspace permissions, ensuring that the right users have the necessary access for collaboration.
    - Configure workspace settings, including enabling speculative plans, auto-apply, and other relevant options.
    - Decide on a specific git tag format for auto-apply to streamline the deployment process.

6.2.2 Workflow overview

These are the high level steps one would take to implement a successful VCS-driven workflow in Terraform.

1. Configure VCS Integration: Establish a VCS integration for your organization to connect Terraform Cloud and Terraform Enterprise with your Version Control System.
2. Connect Workspace to Repository Branch: Link your workspace to the desired branch in your VCS repository, enabling automatic synchronization of code changes as they are pushed.
3. Adapt Branching Strategy: Adopt a suitable branching strategy, such as git flow, to effectively manage different stages of development and deployment within your VCS repository.
4. Enable Speculative Plan Runs: Automate speculative plan runs for each branch, allowing you to preview changes and validate configurations without impacting the live infrastructure.
5. Define PR Process: Establish a Pull Request (PR) process (which essentially means that branch protection is enabled and any changes will need to come in via a PR) with the necessary approvers to ensure thorough code review and approval before deploying changes.
6. Configure Automatic Run Triggers: Set up automatic run triggers for the plan phase based on git tag pushes, streamlining the deployment process and enhancing efficiency.

7. Deploy with 'Action': Utilize an 'Action' configuration to facilitate deployment based on the git tag push. You can configure this 'Action' for either plan or apply mode, depending on your specific requirements.

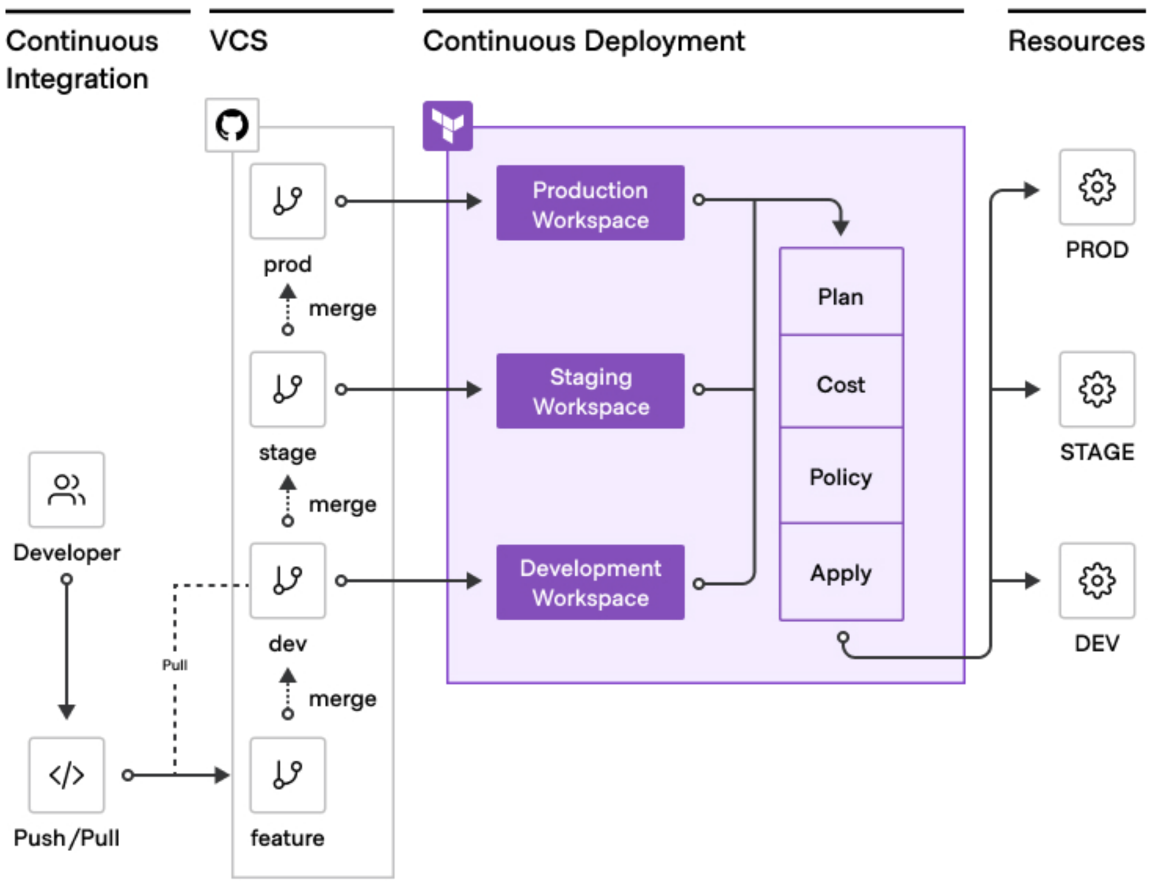


Figure 17: Workflow overview

These essential steps for setting up a VCS-driven workflow will be discussed in more depth in the subsequent sections. We'll explore each step thoroughly, providing detailed insights and practical guidance to help you effectively implement and streamline the workflow within your organization.

6.2.3 VCS integration

Incorporating the Version Control System (VCS) provider is crucial to establishing an automated workflow that predicts, plans, and initiates runs seamlessly. In the VCS-driven approach, each workspace is linked to a specific VCS repository, ensuring a clear association between branches and workspaces. To access configurations from the VCS, Terraform requires permissions for the following tasks:

- Access a list of repositories: This enables you to search for repositories when creating new workspaces. Register webhooks with your VCS provider: Webhooks keep Terraform Cloud informed about new commits to your chosen branch.
- Download repository contents at a specific commit: This action allows Terraform to execute with the specified code. Terraform Cloud uses webhooks to monitor new commits, pull requests and tags. When a new commit is pushed to the repository branch, it triggers an automatic run based on the configured trigger. Similarly, initiating a pull request or merge request to the branch will activate speculative plans.

6.2.4 VCS connection

Integrating the VCS provider is essential for the workflow to automatically predict, plan, and initiate runs. Terraform Cloud uses the OAuth protocol to authenticate with VCS providers, and each provider may have distinct methods and steps to configure this connection. For comprehensive details on setting up the connection with your specific VCS provider, please refer to the respective guide.

Connecting your Terraform workspace to the desired branch in your VCS repository is a key step in the VCS-driven workflow. By linking the workspace and the branch, you enable automatic synchronization of code changes as they are pushed to the repository. This means that every time a new commit is made to the associated branch, Terraform Cloud will automatically trigger a run based on the configured trigger.

Terraform Cloud provides three main options to specify which changes trigger runs in your repository:

1. When any changes are pushed to a specific branch
2. When changes are made to particular files within a specified branch (using the trigger patterns)

3. Any tags to a repo matching a pattern (which is what is described in option 1). Branches are irrelevant for this option. A single change in a repository can cause changes in multiple workspaces if customers are mapping a given repo-branch combination to multiple workspaces: This option instructs Terraform to begin new runs only for changes that have a specific tag format. You can choose between various tag format options:

   - Semantic Versioning: It matches tags in the popular semantic versioning format. For example, 0.4.2.
   - Version contains a prefix: It matches tags which have an additional prefix before the semantic versioning format. For example, version-0.4.2.
   - Version contains a suffix: It matches tags which have an additional suffix after the semantic versioning format. For example, 0.4.2-alpha.
   - Custom Regular Expression: You can define your own regex for Terraform Cloud to match against tags.

When managing your workspace with the `hashicorp/tfe` provider and triggering runs through matching git tags, remember to include an additional \ to escape the regex pattern.

Tag Format	Regex Pattern	Regex Pattern (Escaped)
Semantic Versioning	`^\d+.\d+.\d+$`	`^\\d+.\\d+.\\d+$`
Version contains a prefix	`\d+.\d+.\d+$`	`\\d+.\\d+.\\d+$`
Version contains a suffix	`^\d+.\d+.\d+`	`^\\d+.\\d+.\\d+`

6.3 Auto apply

Terraform offers an “auto apply” option, which will apply changes from successful plans without prompting for approval (this is irrespective of way the plan is triggered - VCS, API etc). This feature is especially useful in non-interactive environments like continuous deployment workflows, where you want to ensure that no one can change your infrastructure outside of your automated build pipeline. Enabling auto apply reduces the risk of configuration drift and unexpected changes to the infrastructure.

6.4 Branching strategy

Now that we've discussed how to configure the VCS integration and establish the connection between Terraform Cloud and your Version Control System, let's explore how to effectively manage your workspaces by utilizing a defined branching strategy. Having a well-defined branching strategy is important as it allows for effective management of different stages of development and deployment, providing clarity and control over the code changes introduced at each phase. As an overview, here are workspace configuration guidelines that will assist you in maintaining a well-organized, scalable, and efficient Terraform workflow:

- Unique Repository+Branch Mapping: The recommended pattern is to assign each workspace to a unique combination of a repository and branch. Whenever changes are made to the associated repository and branch, a Terraform run will automatically trigger for that specific workspace.
- Subfolders within Repository: While subfolders can exist within the repository, it's important to ensure that all resources in the repository are managed by the same workspace. This practice promotes isolation and organization, making it easier to manage and maintain Terraform configurations.
- Branches for Related Workspaces: Utilize different branches to represent related workspaces. For example, you can have a branch named `main` for production and another branch named `dev` for development. This approach helps to separate and manage configurations for different environments efficiently.
- Pull Request (PR) Workflow: When a PR is opened towards a specific workspace branch, the PR must only contain information about that particular workspace. This focused approach allows for clearer review and approval of changes specific to the intended workspace.
- Release Workflow: The workflow for releasing changes follows standard development practices. Developers make changes on a feature branch, which is later merged into a long-lived branch like `dev`. Finally, the changes from the `dev` branch are merged into the `main` workspace, enabling a controlled and streamlined release process.

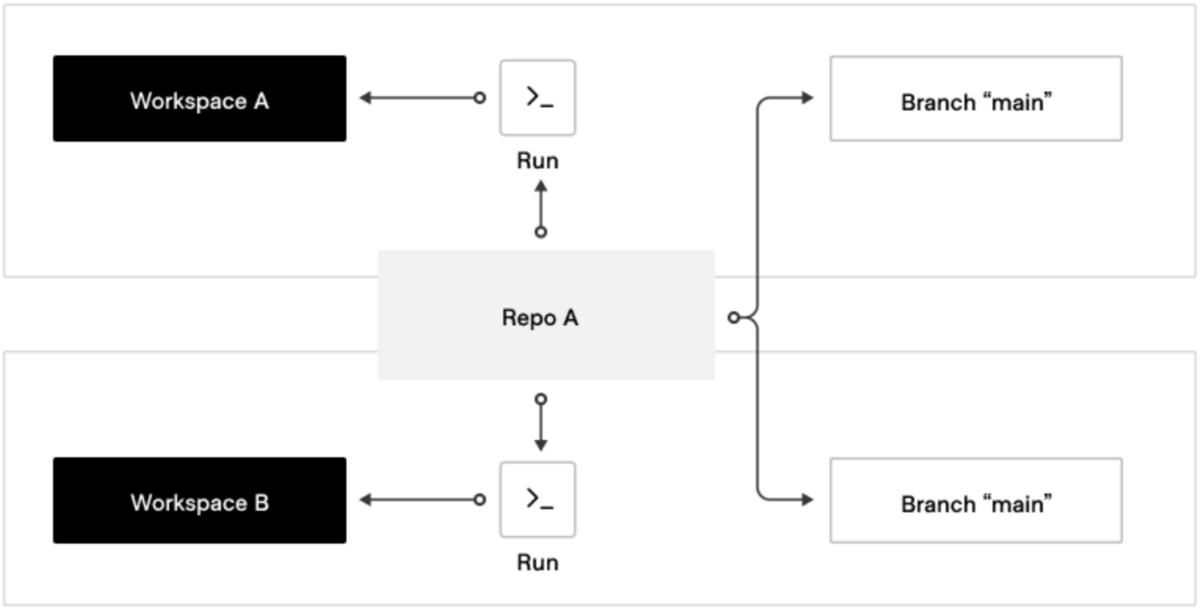


Figure 18: Branching strategy

Recommendation

Consider adopting a branching strategy similar to GitHub Flow, especially if GitHub is your primary version control tool. This model is based on trunking but utilizes short-lived branches instead of feature tags. With this approach, you create a branch from the main branch, make your changes (in our Terraform use case, these changes would be within the environment's specified workspace folder), commit the changes, push a pull request, merge the changes into the main branch, and then delete the branch. The main objective of this model is to have short-lived branches, ensuring that the main branch always reflects the latest features or changes at any given time.

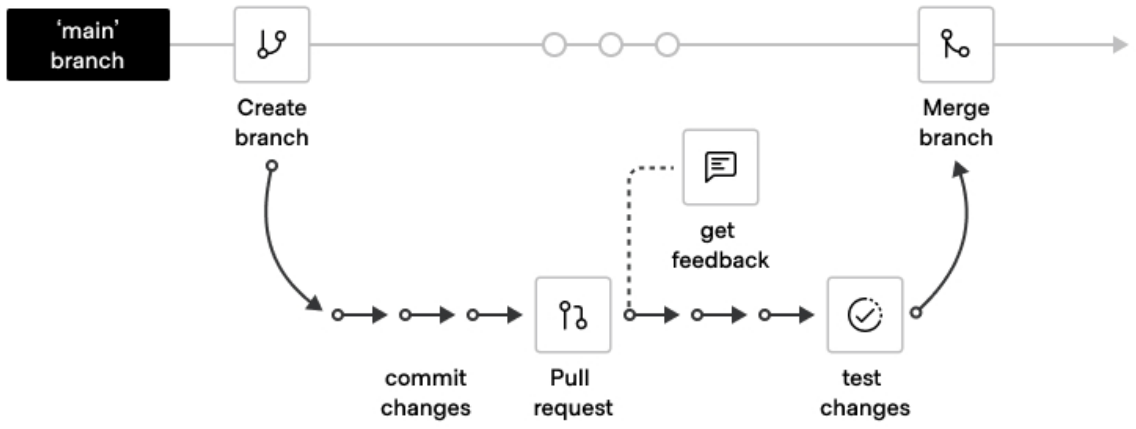


Figure 19: Using short-lived branches

As a general approach, we suggest aligning the `dev`, `stage`, and `production` branches with their respective workspaces for applications. This alignment promotes a clear and structured workflow, enabling smooth transitions between different development stages and deployments.

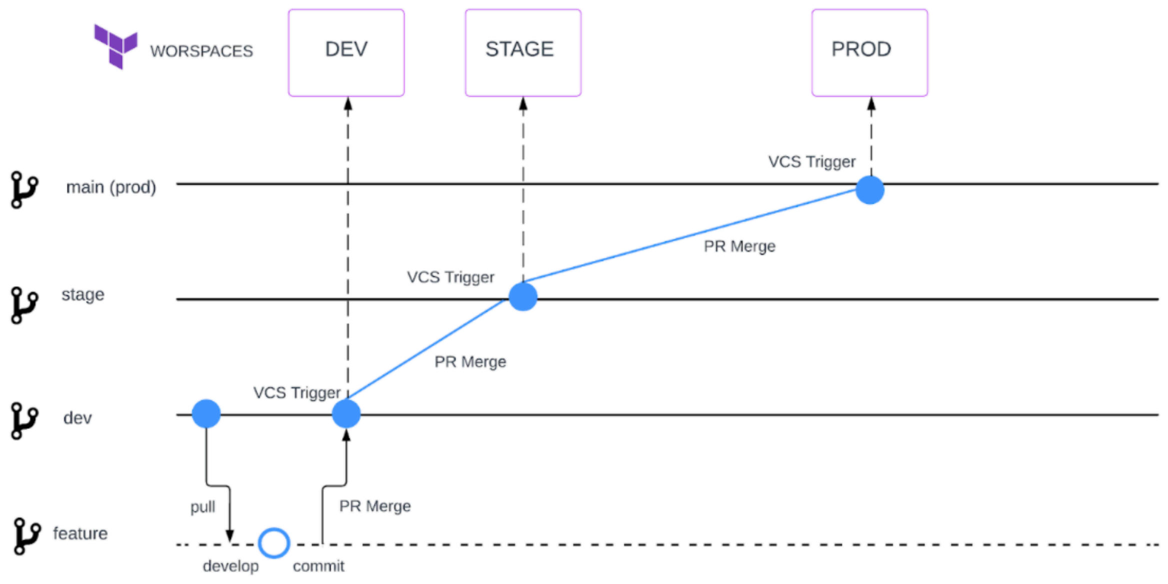


Figure 20: Changes promotion path

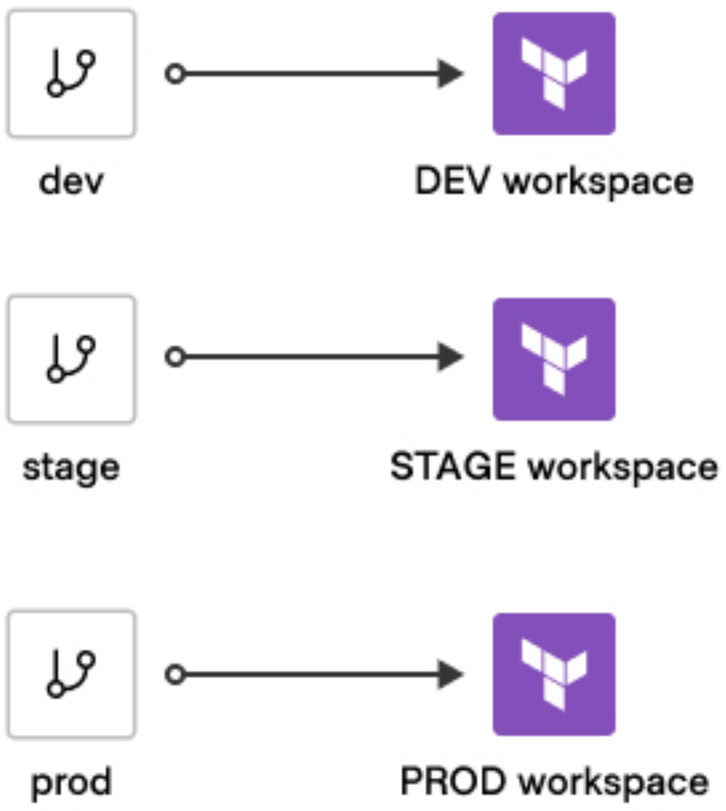


Figure 21: Mapping of branches to workspaces

Here are the steps that align with the GitHub Flow Git Branching Model:

1. Start by creating a repository with three branches - `dev`, `stage`, and `prod`. Commit your initial code in the dev branch.
2. For each feature development, create a separate feature branch off the `dev` branch. For example: `feature/app1-web-tier`.
3. Make your changes and commit them to your branch. Ensure that each commit contains an isolated and complete change for easy reversion and feedback.
4. Push your changes to your feature branch and create a pull request.
5. Address any comments, issues, tasks, or questions from reviewers during the pull request

review process.

6. The reviewer will merge your pull request, incorporating your feature branch into the `dev` branch.
7. The final reviewer will perform a squash commits and merge, streamlining the changes into the `dev` branch.
8. The feature branch can be deleted after merging to maintain only long-lived branches for each environment (dev, stage, and prod).

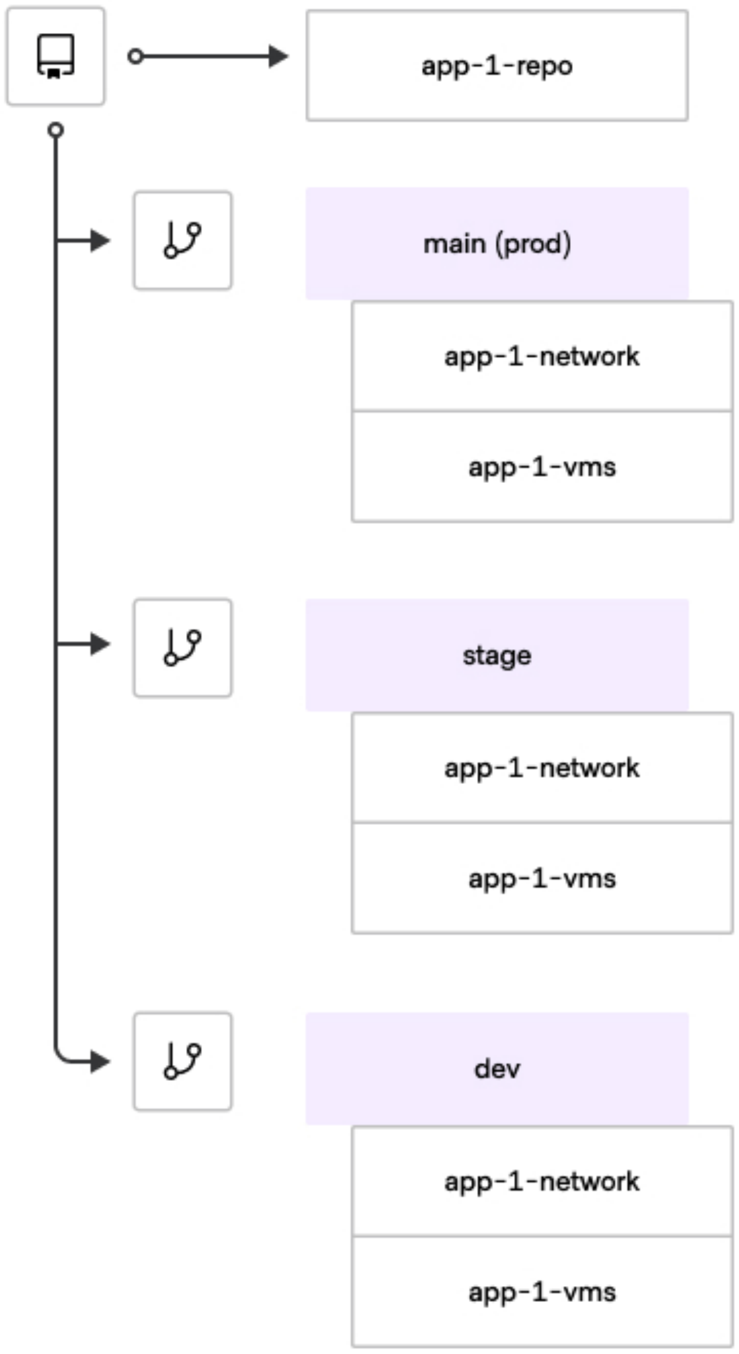


Figure 22: Github flow git branching model

Speculative plan

Now that we have looked at the recommended branching strategy, let's explore the code update workflow that triggers plans or runs. In this process, speculative plans play a crucial role as they allow you to preview and validate changes before merging them into the main branches. This ensures that your infrastructure remains stable and reliable. Here are the steps for the code update workflow, taking advantage of the recommended branching strategy and utilizing speculative plans:

1. Create a *feature branch* from the `dev` branch to work on specific changes.
2. Test and validate the features in the feature branch.
3. Open the PR. This will trigger a speculative run.
4. Review, approve and merge the changes into the `dev` branch.
5. Repeat the steps 3 and 4 for `stage` branch and then the `main` production branch.
6. Delete the feature branch to avoid the maintenance of long-lived feature branches.

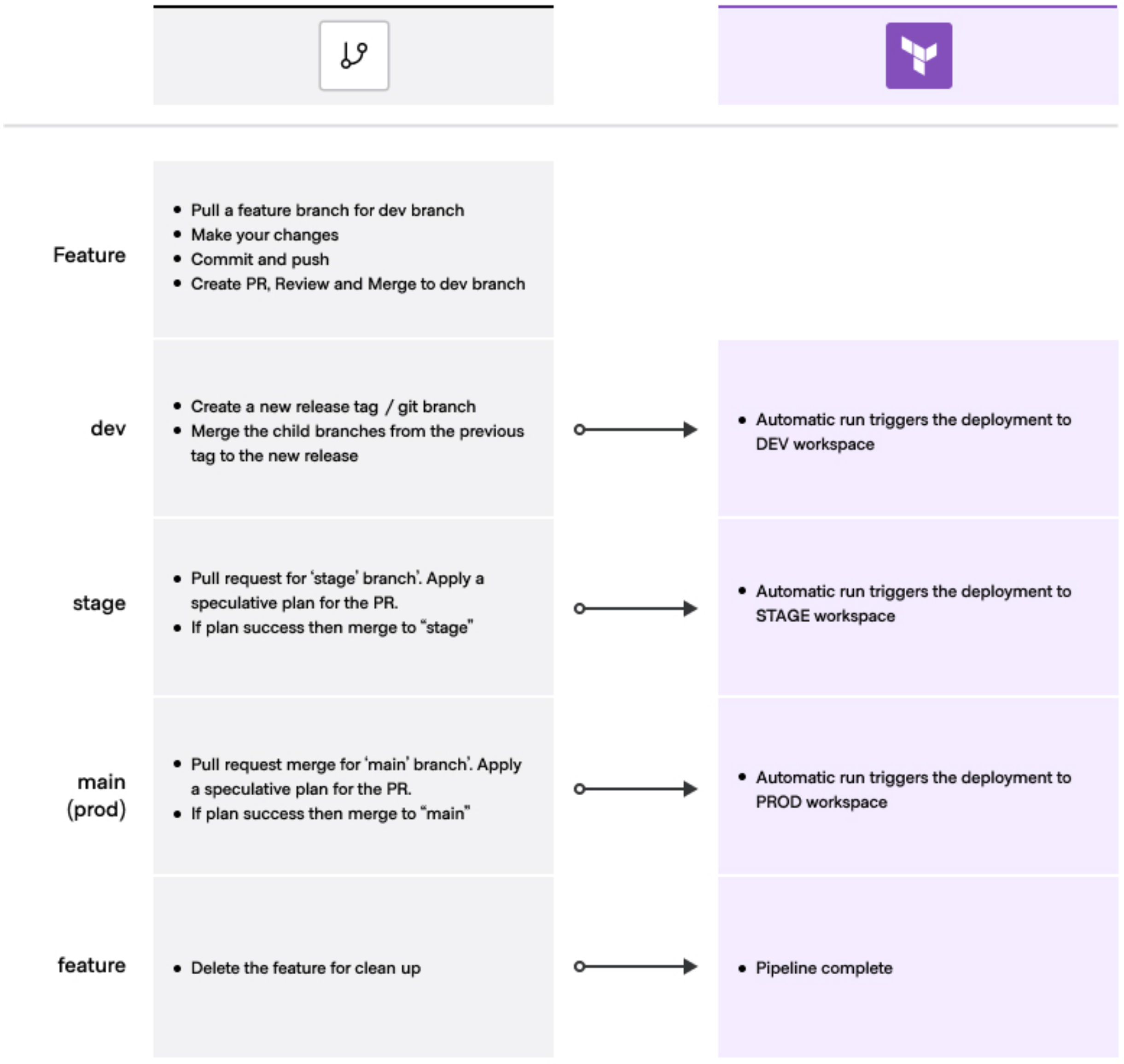

Feature
Pull a feature branch for dev branch
Make your changes
Commit and push
Create PR, Review and Merge to dev branch
dev
Create a new release tag / git branch
Merge the child branches from the previous tag to the new release
Automatic run triggers the deployment to DEV workspace
stage
Pull request for 'stage' branch'. Apply a speculative plan for the PR.
If plan success then merge to "stage"
Automatic run triggers the deployment to STAGE workspace
main (prod)
Pull request merge for 'main' branch'. Apply a speculative plan for the PR.
If plan success then merge to "main"
Automatic run triggers the deployment to PROD workspace
feature
Delete the feature for clean up
Pipeline complete

Figure 23: Speculative plans and code deployment

Speculative plans are a valuable feature for reviewing proposed changes. Terraform can automatically run speculative plans for pull requests or merge requests. You can view speculative plans in a workspace's list of normal runs. Additionally, if you are using GitHub,Terraform adds a link to the run in the pull request itself, along with an indicator of the run's status. When you update a pull request, Terraform performs new speculative plans and updates the link.

Although any contributor to the repository can see the status indicators for pull request plans, only members of your Terraform organization with permission to read runs for the affected workspaces can click through and view the complete plan output.

Whenever a pull request is created or updated, Terraform checks whether it should run speculative plans in workspaces connected to that repository, based on the following rules:

- Only pull requests that originate from within the same repository can trigger speculative plans.
- Pull requests can only trigger runs in workspaces where automatic speculative plans are allowed. You can disable automatic speculative plans in a workspace's VCS settings.
- A pull request will only trigger speculative plans in workspaces that are connected to that pull request's destination branch. In order to trigger an apply run, the PR must be closed and merged into the branch, which means statuses are no longer being registered.

Terraform Cloud does not update the status checks on a pull request with the status of an associated apply. This means that a commit with a successful plan but an errored apply will still show the passing commit status from the plan.

6.5 Additional workflow options

While the recommended VCS-driven workflow offers seamless collaboration and automation through integration with your Version Control System, it's good to consider alternative approaches that might better suit your organization's specific needs. The VCS-driven workflow provides unparalleled flexibility and enables teams to leverage their existing CI/CD pipelines effectively. However, in certain scenarios, you might find it beneficial to explore other options such as the API-driven and CLI-driven workflows. These alternative approaches have their advantages and can be valuable additions to your Terraform workflow.

6.5.1 API-driven workflow

When working with Terraform Cloud and Terraform Enterprise, the API-driven workflow offers complete control over the IaC process, making it highly compatible with custom CI/CD pipelines utilizing the Terraform Cloud and Terraform Enterprise API. If your requirement involves seamless integration with external CI tools, an API-driven workflow can be the most suitable approach, providing flexibility to utilize your current CI tools for tracking state and maintaining continuous linkage between Terraform and your code. As the most advanced approach, this workflow is often adopted by large organizations with complex requirements, either directly or as an evolution from the CLI-driven workflow.

The API-driven workflow offers distinct advantages over the recommended VCS approach, including unparalleled flexibility for integration with existing CI/CD pipelines, additional testing capabilities, and full integration with third-party platforms. It also enables centralized code management, retains current CI/CD configurations, and leverages existing programming languages for a more tailored Terraform workflow.

API workflow overview

The high-level API-driven workflow utilizing Terraform Cloud and Terraform Enterprise API endpoints includes the following steps:

1. Define variables for organization, workspace, and projects using API tokens, usually team tokens.
2. Prepare the configuration file for upload using the configuration version API.
3. Solicit the workspace ID using the configuration version API.
4. Create a configuration version using the configuration version API.
5. Upload the configuration version using the run apply API.

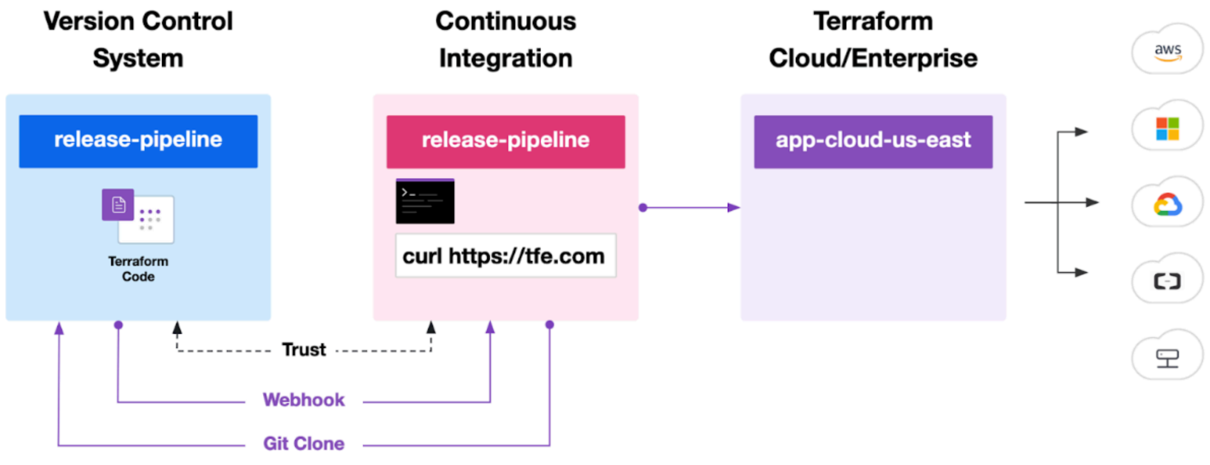


Figure 24: The VCS workflow

6.5.2 Pipeline templates

The API-driven workflow can be further enhanced by leveraging pipeline templates, which bring numerous benefits to integrating Terraform Cloud and Terraform Enterprise with various CI/CD platforms. These templates provide clear and predefined examples, making the integration process straightforward and less time-consuming. With a quick-start approach, organizations can streamline their integration process, accelerating the deployment of Terraform in their CI/CD workflows while ensuring consistency across the pipelines. By utilizing existing templates, organizations save effort and resources that would otherwise be spent on developing custom API-driven workflows. Moreover, these prescriptive templates are designed to cater to various CI/CD platforms, ensuring seamless compatibility and smooth integration with existing tools. As an added advantage, the templates are continuously maintained and updated by the Terraform community, allowing organizations to easily benefit from the latest improvements and best practices, keeping their Terraform workflow up-to-date and efficient.

When integrating Terraform into your existing CI/CD pipelines, consider the convenience and efficiency of leveraging prescriptive pipeline templates. These templates provide explicit examples of how to integrate with Terraform Cloud and Terraform Enterprise on various CI/CD platforms, offering a quick and straightforward way to get started without the need to develop templates from scratch. By following these steps, you can smoothly incorporate Terraform into your CI/CD workflows:

1. Choose a pipeline template that aligns with your CI/CD platform and meets your specific integration needs with Terraform Cloud and Terraform Enterprise.
2. Tailor the template to suit your organization's requirements by adjusting configurations, adding or removing steps, and integrating it with other tools in your CI/CD pipeline.
3. Integrate the customized template into your CI/CD pipeline and thoroughly test its functionality to ensure it aligns with your desired workflows.

Examples

HashiCorp offers GitHub Actions that integrate with Terraform Cloud and Terraform Enterprise APIs, empowering you to create customized CI/CD workflows tailored to your organization's specific needs.

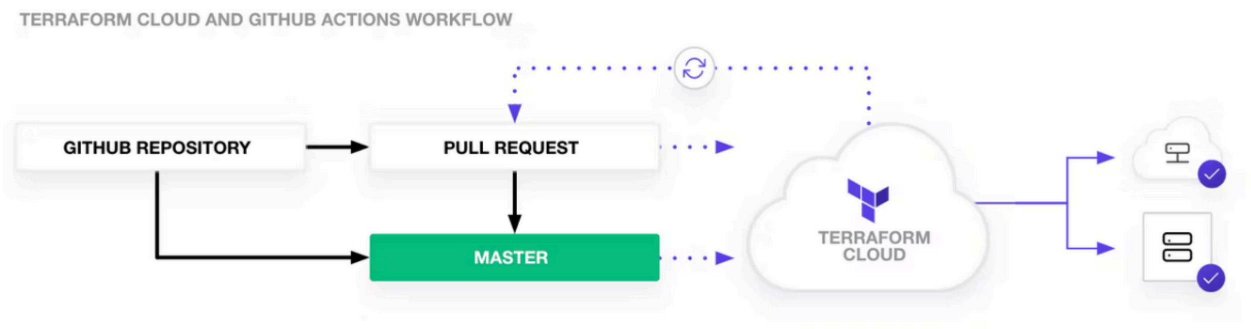


Figure 25: Github integration

A typical GitHub Action workflow includes generating a plan for each commit to a pull request branch, which can be reviewed in Terraform Cloud or Terraform Enterprise. Once you update the VCS branch, the configuration is automatically applied.

Upon configuring the GitHub Action, you can create and merge a pull request to test the workflow. Terraform Cloud's built-in support for webhooks facilitates this generic workflow.

Moreover, by leveraging HashiCorp's Terraform GitHub Actions, you can design a custom workflow with additional steps before or after your Terraform operations, enhancing your automation capabilities.

For additional information and resources, you can refer to the following documentation:

- Announcement
- Tutorial

6.5.3 CLI-driven workflow

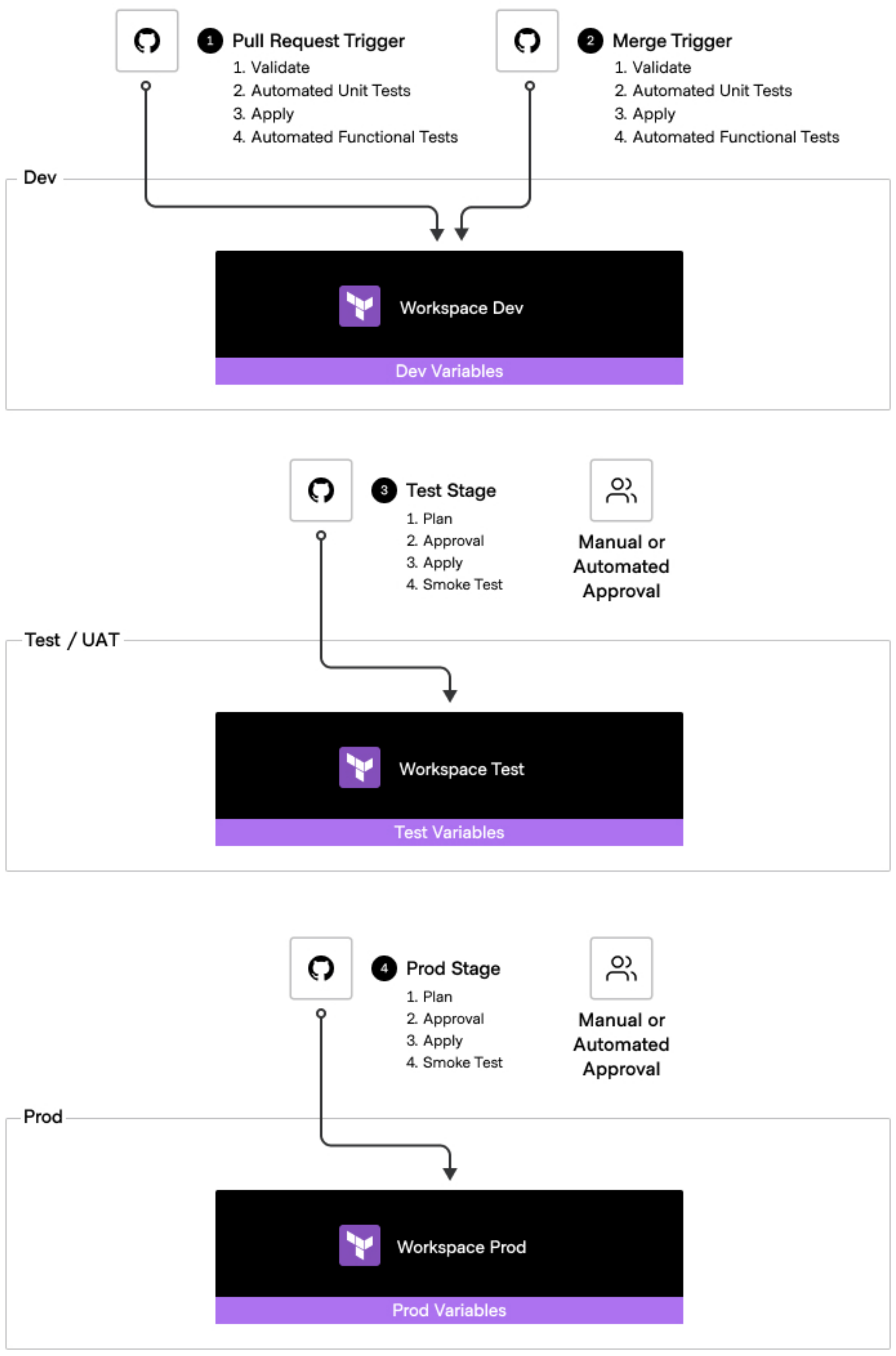
With Terraform's CLI-driven workflow, you can transition from using OSS CLI Commands to Terraform Cloud Agents for triggering the build process. This workflow offers the flexibility to iterate quickly on your configuration and work locally, allowing for efficient development and testing. Moreover, it simplifies integration with CI/CD tools, enabling the addition of testing steps and enhancing deployment reliability, streamlining the process compared to the VCS-driven approach. Additionally, the CLI workflow promotes centralized code management, fostering effective collaboration and version control, which is especially valuable for organizations with complex requirements. Furthermore, it permits convenient local development of workspaces, minimizing the need for frequent code commits and enhancing the overall development experience. For organizations with existing CI/CD configurations, transitioning to the CLI workflow can be straightforward, allowing them to retain their current setups and reducing the need for major adjustments. Lastly, the CLI workflow provides faster feedback to practitioners, enabling greater flexibility in deploying changes across different environments, making it an attractive alternative to the VCS-driven approach for those who require rapid iterations and deployments.

Workflow overview

This high-level CLI-driven workflow includes the following steps:

1. Generate a CLI token for use as a CI/CD tool or local user.
2. Create a workspace and update the backend configuration block.
3. Construct CI/CD pipelines to include pre-plan checks like linting, security scanning, and unit testing.
4. Trigger the plan using the terraform plan, apply, or destroy CLI command, whether invoked by a human user or a CI/CD script.
5. Configure your CI/CD tool to automatically trigger plans based on branch or directory changes.
6. Set up different pipelines for pull requests, pushes to specific branches, etc. After making and validating code changes, merge pull requests to the main branch.

CD Tool


1 Pull Request Trigger
1. Validate
2. Automated Unit Tests
3. Apply
4. Automated Functional Tests
2 Merge Trigger
1. Validate
2. Automated Unit Tests
3. Apply
4. Automated Functional Tests
Dev
Workspace Dev
Dev Variables
3 Test Stage
1. Plan
2. Approval
3. Apply
4. Smoke Test
Manual or Automated Approval
Test / UAT
Workspace Test
Test Variables
4 Prod Stage
1. Plan
2. Approval
3. Apply
4. Smoke Test
Manual or Automated Approval
Prod
Workspace Prod
Prod Variables

7 Observability

7.1 Observability and monitoring

Observability is about being able to look at the live outputs of a system running in production and answer questions about what is going on. It is often associated to monitoring, but both concepts are distinct:

- **Observability** is about setting up a system so that we have access to real-time data about what the system is doing
- **Monitoring** is about capturing this data and analyzing it to make decisions. Example of monitoring goals are: Detecting failures or performance degradation, capacity planning, intruder detection.

7.2 Observability in Terraform Cloud and Terraform Enterprise

Observability features play a critical role in TFC and TFE for maintaining a secure and compliant infrastructure-as-code workflow. These features provide detailed visibility into user actions, changes to infrastructure, and system events, creating a comprehensive record of all activities within the platform.

By capturing this information, organizations can effectively monitor and analyze activities, identify potential security threats, track configuration changes, and troubleshoot issues promptly. Audit and logging help meet regulatory requirements, bolster accountability, and support incident response efforts.

Terraform Cloud (TFC) and Terraform Enterprise (TFE) offer several features to support observability:

Observability feature	Terraform Cloud	Terraform Enterprise
Operational logs	No	Yes
Audit trail	Yes	Yes
Metrics	No	Yes
Terraform Cloud agent logs	Yes	Yes

Operational logs track the performance and behavior of the system. They provide information about the system's functioning, such as error messages, warnings, and other events that can help troubleshoot issues and identify performance bottlenecks. SRE teams typically use operational logs to monitor and maintain the service.

On the other hand, *audit trail* logs focus on tracking security-related events and activities within a system. They capture information about login attempts, access control changes, suspicious activities, and other security events. Audit trail logs are crucial for detecting and investigating security incidents, as they record who did what in a system. Security analysts and incident response teams often use these logs to identify and respond to potential threats.

Metrics measure application component performance and usage and are the raw data used to detect service quality issues or inform a capacity planning exercise.

TFE and TFC differ regarding operational logs and metrics, which stems from the shared responsibility model associated with TFC. With TFC, HashiCorp's SRE team tracks the operational logs and metrics as part of the service's operation. With TFE, your SRE team in charge of running the internal infrastructure-as-code service tracks those operation logs and metrics.

If you're using TFC agents, including those logs in the list of logs you collect and analyze is important. This will help ensure that you have a complete picture of all activities and can identify any issues or errors that may arise. By analyzing your logs together, you'll be better equipped to make informed decisions about optimizing and improving your TFE/TFC deployment. Remember always to stay vigilant and monitor your logs to ensure your systems run smoothly and securely.

7.3 Configuring data collection

In this section we'll cover how to set the data collection for:

1. Collecting metrics and logs (including audit trail logs) on TFE
2. Audit trail logs on TFC
3. Collecting metrics and logs on TFC agents (applicable for TFC and TFE)

7.3.1 TFE metrics and logs

Configuring metrics collection on TFE

By default TFE does not collect metrics, consequently you need to explicitely enable it. You can enable metrics collection by setting the `TFE_METRICS_ENABLE` parameter to **`true`**.

Once the metrics collection is turned on, you will need to configure your monitoring tool of choice to periodically query the TFE metrics endpoint to collect and store this information.

Configuration parameter	Description
TFE_METRICS_HTTP_PORT	The HTTP port that metrics will be exposed on.
TFE_METRICS_HTTPS_PORT	The HTTPS port that metrics will be exposed on.

Configuration parameter	Default Value
TFE_METRICS_HTTP_PORT	`9090`
TFE_METRICS_HTTPS_PORT	`9091`

Metrics can be captured in two formats, via the metrics endpoint URL. The table below provides the available options as well as the URL to use. We recommend capturing metrics over an encrypted connection.

Metrics Endpoint URL	Metrics format
`https://<tfe_instance>:9091/metrics`	JSON
`https://<tfe_instance>:9091/metrics?format=prometheus`	Prometheus

TFE will compute an aggregate metric value from a 5 seconds sample. This value will be kept in memory for 15 seconds before being flushed. This means that if the monitoring tool pooling frequency is greater than 15 seconds (for example every 60 seconds), you may be missing information necessary to detect short-lived issues.

If you are running multiple TFE instances, you must collect the metrics from each deployed TFE instance and aggregate the information to have a global view of the TFE service.

If you are using the `terraform-aws-tfe` module to deploy TFE, note that by default, it will enable metrics collection (see the `enable_metrics_collection` input variable).

Configuring log collection on TFE

TFE will emit logs to standard output and standard error. We recommend to collect TFE logs in a central location, preferably using a specialized tool that provides searching and alerting capabilities, althought we support sending logs to object storage for example.

There is a limited number of supported log destinations:

Category	Supported log destinations
AWS	AWS S3, AWS CloudWatch
Microsoft Azure	Azure Blob Storage, Azure Log Analytics
Google Cloud Platform	Google Cloud Platform Cloud Logging
Specialized SaaS	Datadog, Splunk Enterprise HTTP Event Collector (HEC)
Other	Syslog, Fluent Bit or Fluentd instance

If you are using the `terraform-aws-tfe` module to deploy TFE, note that by default, it not will enable log forwarding (see the `log_forwarding_enabled` input variable).
The module has support for:

1. Forwarding logs to AWS CloudWatch (using a built-in configuration template).
2. Forwarding logs to AWS S3 (using a built-in configuration template).
3. Forwarding logs to a custom destination.

Implementing audit trail on TFE

TFE will generate audit trail logs along with the application logs. If you need to forward audit trail logs to a specialized system, such as a Security Information and Event Management solution (SIEM), we recommend configuring a filter that will intercept all logs containing the string `[Audit Log]` and forward them to the SIEM system.

7.3.2 TFC audit trail logs

Terraform Cloud features an Audit Log API endpoint that you must use to collect the audit events and store them in the appropriate system. To implement this solution, you will need:

- A method to schedule and automate the audit events collection,
- A secure storage solution to store the audit events, and
- A data lifecycle solution to correctly dispose of the audit events once they are no longer required.

> Terraform Cloud will keep audit trail records for 14 days. After that delay, the information will be discarded.
> You should consider that information when designing and configuring your audit trail collection solution for TFC.

If you use a Security Information and Event Management system (SIEM), this must be the destination for those audit events. Suppose you are not using a SIEM but instead are using a centralized log management solution (Datadog, New Relic, Elastic, etc.): In that case, you must send those audit events to your centralized log management system. If neither of these solutions is available, you should still collect those audit events and store them securely using an object storage solution, such as AWS S3.

7.3.3 Metrics and logs on TFC agents

Configuring TFC agent metrics collection

The TFC agent binary exposes telemetry data using the OpenTelemetry protocol. This behavior allows the user to use a standard OpenTelemetry collector to push the metrics to a monitoring solution that supports the protocol, such as Prometheus or Datadog.

Because of that, to collect telemetry data from the agent, you need to have:

- A way to deploy and operate OpenTelemetry collector(s)
- A monitoring system that can integrate with OpenTelemetry collectors

Details about the selection of such a monitoring system or the operations of OpenTelemetry collectors are beyond the scope of this document. However, we will provide some guidelines regarding integrating OpenTelemetry collectors with TFC agents.

> If you are a Datadog customer, instead of deploying OpenTelemetry collectors, we recommend that you configure the Datadog agent to accept gRPC OTLP connections and then configure your TFC agent to use the Datadog agent as metrics destination.

Because OpenTelemetry is a push system, you must start the collector before the TFC agent. Conversely, you should shut down the collector after you have stopped all TFC agents using that collector. We recommend having a one to one ratio of TFC agent instance and OpenTelemetry collector for long running TFC agents as this will simplify management.

The OpenTelemetry integration will tag metrics with a number of useful fields, including the agent pool ID (`agent_pool_id`) and the agent name (`agent_name`). If you don't already have a naming convention for your TFC agents, then we recommend building one, as it will help you organize your dashboard with the metrics collected from TFC agents. You can then set the agent's name at startup time using the `TFC_AGENT_NAME` environment variable or the `-name` command line option.

Configuring TFC agent log collection

If you are using TFC agents in your TFE deployment then you will need to configure log collection for agents.

7.4 Using observability data

We've discussed how to configure TFE and TFC to provide observability data. We'll now pivot and go over an approach to make use of all this data for monitoring, covering:

- Data collection and aggregation
- Alerting and notification

7.4.1 Data collection and aggregation

As you collect and aggregate metrics and logs, you must consider the following:

- Volume (including granularity and frequency)
- Data retention
- Security and privacy

As you collect metrics, you must consider the *volume of information* relative to the value of individual metrics. This will impact the range of metrics collected and the collection frequency, and decisions in this area have clear tradeoffs.

Collecting a wide range of metrics at a high frequency will yield the most data and potentially a more accurate view of the system's health and load. But it comes at a higher cost because you must collect, store, analyze, alert and report on all this information.

One approach to control costs without reducing the range of metrics collected is to have clear *data retention rules* and leverage metrics roll-up.

We recommend keeping highly detailed metrics for shorter periods (two weeks usually works), discarding metrics after two weeks, or rolling up metrics (aggregating 60-second resolution data to 5 minutes, hourly or daily). If your monitoring platform supports rolling up data, you may have the choice of implementing staged roll-up:

- Full resolution for the last two weeks
- 5 minutes roll-up for metrics older than two weeks, up to a month
- Hourly roll-up for metrics over a month
- Discard metrics older than three months
- Etc.

For logs, we recommend keeping application logs for a short period (two weeks usually works) and discarding them after that. For audit trail logs, retention is typically longer, and you will need to align with the policy of your security organization.

When it comes to logs, you should be mindful that they may contain sensitive information. We recommend putting proper security measures in place to protect the collected metrics and logs. This includes implementing access controls, encryption (at rest and in transit), and monitoring mechanisms to safeguard sensitive information.

You should mark variables as sensitive in TFE or TFC when they contain value that must be confidential (such as credentials). This will cause the Terraform binary to redact this information in logs and in general take additional measures to prevent accidental data leaks.

7.4.2 Alerting and notification

Logging and audit capabilities in Terraform and Terraform Enterprise are essential for capturing and monitoring all events within the infrastructure management solution. Terraform Open Source mainly focuses on individual resource communication and API responses, while Terraform Enterprise provides comprehensive logs, including interactions with the solution, security events, and various inter-communication logs.

Terraform Enterprise offers different event streams and two types of logs: application logs and audit logs. Application logs provide information about the services that make up Terraform Enterprise, while audit logs record changes made to any resource managed by the platform. To ensure effective monitoring, notable log events contain a list of recommended events to track. These events fall into several categories, below are each of the categories and a non-comprehensive list of what events may fall under each:

1. Security Driven Events:
   - Requests to Authentication Tokens
   - Requests to Configuration Versions
   - Changes in policy set assignments
   - Changes in team permissions and user assignments
2. Login Events:
   - Login and Logout
   - Failed login attempts
   - Accessing, editing, and/or removing policies
3. Configuration Events:
   - Project and workspace operations
   - Variable set operations
4. Usage and Consumption Driven Events:
   - Execution or access of Terraform
   - Creation of a run in Terraform
   - Starting a plan in Terraform
   - Initiating an apply in Terraform
   - Policy overriden

5. Performance Driven Events:

   - Monitoring Terraform System/Health-check Endpoint
   - Tracking the number of active workers/agents
   - Monitoring host resource utilization

The intent of presenting various events is to demonstrate different categories and approaches to separating duties. Some events may appear in multiple categories, and their classification might vary based on your organization's perspective. For instance, log-ins might be more informative to a Security or Authentication team and can be enabled through AD/LDAP, while performance-driven events could be managed by your team hosting TFE. The main focus is to highlight the diverse types of events and offer a logical approach for handling them. This topic can be further delved into by reviewing Medium material.

8 Support readiness

8.1 Checklist

To ensure smooth communication and efficient support, make sure that these guidelines are followed:

- ☐ Each team member has an account created through our support portal.
- ☐ Each team member that should be permitted to create support tickets on behalf of your company must be configured as an Authorized Technical Contact in the support portal. Please provide a list of applicable team members to your company's assigned Solutions Engineer so that they can configure this.
- ☐ Have your team familiarize themselves with our documentation on how to open a support ticket with HashiCorp Support and generate/upload a support bundle, whenever applicable.
- ☐ Be aware of the support plan level your company has purchased to manage expectations of response times. Also, understand the definitions for each severity level when opening a support ticket through our documentation.

9 Appendix

9.1 Terraform DAG - Under the hood concept

The following section offers valuable insights into the computer science concepts utilized in Terraform. Understanding these concepts can greatly assist in the design and implementation of Terraform Infrastructure as Code (IaC). In this section, we will provide a brief overview of these concepts.

Terraform leverages Graph Theory, which enables it to model various types of relationships and processes. Specifically, Terraform utilizes a Directed Acyclic Graph (DAG) structure. In this context, a DAG consists of vertices and edges (also known as arcs), where each edge is directed from one vertex to another. Importantly, this ensures that following the directions of the edges will never form a closed loop.

9.2 Notable log events

We've assembled examples of notable log events that you can use to configure your monitoring tool(s).

9.2.1 Local user successful login

Container: tfe-atlas

```
"message": "2023-06-26 15:06:15 [INFO] [c3024dc4-0616-4eb1-b676-03ba5ec70788] [dd.
    service=atlas dd.trace_id=000 dd.span_id=000] {\"message\":\"Login Meta\",\"
    ip_address\":\"192.168.1.10, 10.0.2.117\",\"id\":\"user-PvcyWJWiVf1PQSR6\",\"
    success\":true}",

"message": "2023-06-26 15:06:15 [INFO] [c3024dc4-0616-4eb1-b676-03ba5ec70788] [dd.
    service=atlas dd.trace_id=000 dd.span_id=000] {\"method\":\"POST\",\"path\":\"/
    session\",\"format\":\"html\",\"status\":302,\"duration\":146.91,\"view\":0.0,\"
    db\":22.68,\"location\":\"https://src-tfe.abasista.sbx.hashidemos.io/app\",\"dd
    \":{\"trace_id\":\"961064219257453666\",\"span_id\":\"000\",\"env\":\"\",\"
    service\":\"atlas\",\"version\":\"\"},\"ddsource\":[\"ruby\"],\"uuid\":\"c3024dc4
    -0616-4eb1-b676-03ba5ec70788\",\"remote_ip\":\"192.168.1.10\",\"request_id\":\"
    c3024dc4-0616-4eb1-b676-03ba5ec70788\",\"user_agent\":\"Mozilla/5.0 (Macintosh;
    Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.0.0
    Safari/537.36\",\"user\":null,\"auth_source\":null}",
```

9.2.2 Local user failed login

Container: tfe-atlas

```
"message": "2023-06-26 15:22:28 [INFO] [121ab95c-9408-4d9c-bf28-2d802696cafa] [dd.
    service=atlas dd.trace_id=000 dd.span_id=000] {\"message\":\"Login Meta\",\"
    ip_address\":\"192.168.1.10, 10.0.2.117\",\"id\":\"user-PvcyWJWiVf1PQSR6\",\"
    success\":false}",
```

9.2.3 SAML/SSO user successful login

Container: tfe-atlas

```
"message": "2023-06-26 17:47:34 [INFO] [2931915a-eda6-406a-8cb6-9cd3187be4e1] [dd.
    service=atlas dd.trace_id=000 dd.span_id=000] [Audit Log] {\"resource\":\"user
    \",\"action\":\"update\",\"resource_id\":\"jmccollum\",\"organization\":null,\"
    organization_id\":null,\"actor\":\"SAML\",\"timestamp\":\"2023-06-26T17:47:34Z
    \",\"actor_ip\":null}",

"message": "2023-06-26 17:47:34 [INFO] [2931915a-eda6-406a-8cb6-9cd3187be4e1] [dd.
    service=atlas dd.trace_id=000 dd.span_id=000] {\"message\":\"Login Meta\",\"
    ip_address\":\"192.168.1.10, 10.0.2.117\",\"id\":\"user-T6WA1nH3jZfojtN5\",\"
    success\":true}",

"message": "2023-06-26 17:47:34 [INFO] [2931915a-eda6-406a-8cb6-9cd3187be4e1] [dd.
    service=atlas dd.trace_id=000 dd.span_id=000] {\"method\":\"POST\",\"path\":\"/
    users/saml/auth\",\"format\":\"html\",\"status\":302,\"duration\":86.28,\"view
    \":0.0,\"db\":14.19,\"location\":\"https://src-tfe.abasista.sbx.hashidemos.io/app
    \",\"dd\":{\"trace_id\":\"2591016944362387312\",\"span_id
    \":\"2649027049369611898\",\"env\":\"\",\"service\":\"atlas\",\"version
    \":\"\"},\"ddsource\":[\"ruby\"],\"uuid\":\"2931915a-eda6-406a-8cb6-9cd3187be4e1
    \",\"remote_ip\":\"192.168.1.10\",\"request_id\":\"2931915a-eda6-406a-8cb6-9
    cd3187be4e1\",\"user_agent\":\"Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv
    :109.0) Gecko/20100101 Firefox/114.0\",\"user\":\"jmccollum\",\"auth_source\":
    null}",

"message": "172.19.0.10 - - [26/Jun/2023:17:47:34 +0000] \"GET /app HTTP/1.0\" 200
    798629 \"https://login.microsoftonline.com/\" \"Mozilla/5.0 (Macintosh; Intel Mac
     OS X 10.15; rv:109.0) Gecko/20100101 Firefox/114.0\"",
```

9.2.4 SAML/SSO user failed login

Container: tfe-atlas N/A - trouble with AzureAD but should be similar to local. The IdP will log the rest.

9.2.5 Update scope of policy set - add all workspaces (global)

Container: tfe-atlas

```
"message": "2023-06-26 17:22:43 [INFO] [87525318-9ac1-49da-8e85-7a0dcacfd873] [dd.
    service=atlas dd.trace_id=000 dd.span_id=000] [Audit Log] {\"resource\":\"
    policy_set\",\"action\":\"update\",\"resource_id\":\"polset-74BF6Nzgd4tURgSF\",\"
    organization\":\"hello\",\"organization_id\":\"org-Ur11wxfYrZ77Zqjb\",\"actor
    \":\"admin\",\"timestamp\":\"2023-06-26T17:22:43Z\",\"actor_ip
    \":\"192.168.1.10\"}",

"message": "2023-06-26 17:22:43 [INFO] [87525318-9ac1-49da-8e85-7a0dcacfd873] [dd.
    service=atlas dd.trace_id=000 dd.span_id=000] {\"method\":\"PATCH\",\"path\":\"/
    api/v2/policy-sets/polset-74BF6Nzgd4tURgSF\",\"format\":\"jsonapi\",\"status
    \":200,\"duration\":31.15,\"view\":7.35,\"db\":9.65,\"dd\":{\"trace_id
    \":\"1471845680221587695\",\"span_id\":\"638927242808815250\",\"env\":\"\",\"
    service\":\"atlas\",\"version\":\"\"},\"ddsource\":[\"ruby\"],\"uuid
    \":\"87525318-9ac1-49da-8e85-7a0dcacfd873\",\"remote_ip\":\"192.168.1.10\",\"
    request_id\":\"87525318-9ac1-49da-8e85-7a0dcacfd873\",\"user_agent\":\"Mozilla
    /5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko)
    Chrome/114.0.0.0 Safari/537.36\",\"user\":\"admin\",\"organization\":\"hello\",\"
    auth_source\":\"ui\"}",
```

9.2.6 Update scope of policy set - remove all workspaces (selected)

Container: tfe-atlas

```
"message": "2023-06-26 17:14:35 [INFO] [c1d341c4-15fc-468a-9466-c759ed7784b4] [dd.
    service=atlas dd.trace_id=000 dd.span_id=000] [Audit Log] {\"resource\":\"
    policy_set\",\"action\":\"update\",\"resource_id\":\"polset-74BF6Nzgd4tURgSF\",\"
    organization\":\"hello\",\"organization_id\":\"org-Ur11wxfYrZ77Zqjb\",\"actor
    \":\"admin\",\"timestamp\":\"2023-06-26T17:14:35Z\",\"actor_ip
    \":\"192.168.1.10\"}",

"message": "2023-06-26 17:14:35 [INFO] [c1d341c4-15fc-468a-9466-c759ed7784b4] [dd.
    service=atlas dd.trace_id=000 dd.span_id=000] {\"method\":\"PATCH\",\"path\":\"/
    api/v2/policy-sets/polset-74BF6Nzgd4tURgSF\",\"format\":\"jsonapi\",\"status
    \":200,\"duration\":33.28,\"view\":8.67,\"db\":10.06,\"dd\":{\"trace_id
    \":\"4054904197741214413\",\"span_id\":\"3188071217353501350\",\"env\":\"\",\"
    service\":\"atlas\",\"version\":\"\"},\"ddsource\":[\"ruby\"],\"uuid\":\"c1d341c4
    -15fc-468a-9466-c759ed7784b4\",\"remote_ip\":\"192.168.1.10\",\"request_id\":\"
    c1d341c4-15fc-468a-9466-c759ed7784b4\",\"user_agent\":\"Mozilla/5.0 (Macintosh;
    Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.0.0
    Safari/537.36\",\"user\":\"admin\",\"organization\":\"hello\",\"auth_source\":\"
    ui\"}"

"message": "10.0.2.117 - - [26/Jun/2023:17:14:35 +0000] \"PATCH /api/v2/policy-sets/
    polset-74BF6Nzgd4tURgSF HTTP/1.1\" 200 528 \"https://src-tfe.abasista.sbx.
    hashidemos.io/app/hello/settings/policy-sets/polset-74BF6Nzgd4tURgSF/edit\" \"
    Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like
    Gecko) Chrome/114.0.0.0 Safari/537.36\"",
```

9.2.7 Delete policy set

Container: tfe-atlas

```
"message": "2023-06-26 17:40:02 [INFO] [18e3fb5e-22b7-44c0-a307-ac7f5b052d52] [dd.
    service=atlas dd.trace_id=000 dd.span_id=000] [Audit Log] {\"resource\":\"
    policy_set\",\"action\":\"destroy\",\"resource_id\":\"polset-74BF6Nzgd4tURgSF
    \",\"organization\":\"hello\",\"organization_id\":\"org-Ur11wxfYrZ77Zqjb\",\"
    actor\":\"admin\",\"timestamp\":\"2023-06-26T17:40:02Z\",\"actor_ip
    \":\"192.168.1.10\"}",

"message": "2023-06-26 17:40:02 [INFO] [18e3fb5e-22b7-44c0-a307-ac7f5b052d52] [dd.
    service=atlas dd.trace_id=000 dd.span_id=000] {\"method\":\"DELETE\",\"path\":\"/
    api/v2/policy-sets/polset-74BF6Nzgd4tURgSF\",\"format\":\"jsonapi\",\"status
    \":204,\"duration\":22.85,\"view\":0.71,\"db\":6.7,\"dd\":{\"trace_id
    \":\"3491109508228992074\",\"span_id\":\"4544634447247653806\",\"env\":\"\",\"
    service\":\"atlas\",\"version\":\"\"},\"ddsource\":[\"ruby\"],\"uuid\":\"18e3fb5e
    -22b7-44c0-a307-ac7f5b052d52\",\"remote_ip\":\"192.168.1.10\",\"request_id\":\"18
    e3fb5e-22b7-44c0-a307-ac7f5b052d52\",\"user_agent\":\"Mozilla/5.0 (Macintosh;
    Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.0.0
    Safari/537.36\",\"user\":\"admin\",\"auth_source\":\"ui\"}",
```

9.2.8 Create policy set (VCS-backed)

Container: tfe-atlas

```
"message": "2023-06-26 18:25:20 [INFO] [8c3c21b3-0778-4716-9255-c26a2dc922fc] [dd.
    service=atlas dd.trace_id=000 dd.span_id=000] [Audit Log] {\"resource\":\"
    policy_set\",\"action\":\"create\",\"resource_id\":\"polset-3iR6D1monBhL8aV8\",\"
    organization\":\"hello\",\"organization_id\":\"org-Ur11wxfYrZ77Zqjb\",\"actor
    \":\"admin\",\"timestamp\":\"2023-06-26T18:25:20Z\",\"actor_ip
    \":\"192.168.1.10\"}",

"message": "2023-06-26 18:25:20 [INFO] [8c3c21b3-0778-4716-9255-c26a2dc922fc] [dd.
    service=atlas dd.trace_id=000 dd.span_id=000] [Audit Log] {\"resource\":\"vcs-
    repo\",\"action\":\"create\",\"resource_id\":\"vcrepo-LfMrYsHniKNEtS3h\",\"
    organization\":\"hello\",\"organization_id\":\"org-Ur11wxfYrZ77Zqjb\",\"actor
    \":\"admin\",\"timestamp\":\"2023-06-26T18:25:20Z\",\"actor_ip
    \":\"192.168.1.10\"}",


"message": "2023-06-26 18:25:20 [INFO] [8c3c21b3-0778-4716-9255-c26a2dc922fc] [dd.
    service=atlas dd.trace_id=000 dd.span_id=000] [Audit Log] {\"resource\":\"vcs-
    repo\",\"action\":\"create_webhook\",\"resource_id\":\"vcrepo-LfMrYsHniKNEtS3h
    \",\"organization\":\"hello\",\"organization_id\":\"org-Ur11wxfYrZ77Zqjb\",\"
    actor\":\"admin\",\"timestamp\":\"2023-06-26T18:25:20Z\",\"actor_ip
    \":\"192.168.1.10\"}",

"message": "2023-06-26 18:25:20 [INFO] [8c3c21b3-0778-4716-9255-c26a2dc922fc] [dd.
    service=atlas dd.trace_id=000 dd.span_id=000] [Audit Log] {\"resource\":\"
    policy_set_version\",\"action\":\"create\",\"resource_id\":\"polsetver-
    NCTDgRCHgHcaUP54\",\"organization\":\"hello\",\"organization_id\":\"org-
    Ur11wxfYrZ77Zqjb\",\"actor\":\"admin\",\"timestamp\":\"2023-06-26T18:25:20Z\",\"
    policy_set\":\"sentinel-template-terraform\",\"actor_ip\":\"192.168.1.10\"}",

"message": "2023-06-26 18:25:20 [INFO] [8c3c21b3-0778-4716-9255-c26a2dc922fc] [dd.
    service=atlas dd.trace_id=000 dd.span_id=000] {\"method\":\"POST\",\"path\":\"/
    api/v2/organizations/hello/policy-sets\",\"format\":\"jsonapi\",\"status\":201,\"
    duration\":1442.77,\"view\":11.47,\"db\":41.39,\"dd\":{\"trace_id
    \":\"1505106906833100313\",\"span_id\":\"2625248489776788340\",\"env\":\"\",\"
    service\":\"atlas\",\"version\":\"\"},\"ddsource\":[\"ruby\"],\"uuid\":\"8c3c21b3
    -0778-4716-9255-c26a2dc922fc\",\"remote_ip\":\"192.168.1.10\",\"request_id\":\"8
    c3c21b3-0778-4716-9255-c26a2dc922fc\",\"user_agent\":\"Mozilla/5.0 (Macintosh;
    Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.0.0
    Safari/537.36\",\"user\":\"admin\",\"organization\":\"hello\",\"auth_source\":\"
    ui\"}",
```

9.2.9 Create run (VCS-driven)

Container: tfe-atlas

```
"message": "2023-06-26 19:11:39 [INFO] [bb5a9c16-1918-48bf-80bf-c8f00215497b] [dd.
    service=atlas dd.trace_id=000 dd.span_id=000] [Audit Log] {\"resource\":\"run
    \",\"action\":\"create\",\"resource_id\":\"run-Y1eFSGFuf3hJNnvG\",\"organization
    \":\"hello\",\"organization_id\":\"org-Ur11wxfYrZ77Zqjb\",\"actor\":\"admin\",\"
    timestamp\":\"2023-06-26T19:11:39Z\",\"actor_ip\":\"192.168.1.10\"}",

"message": "2023-06-26 19:11:39 [INFO] [bb5a9c16-1918-48bf-80bf-c8f00215497b] [dd.
    service=atlas dd.trace_id=000 dd.span_id=000] {\"method\":\"POST\",\"path\":\"/
    api/v2/runs\",\"format\":\"jsonapi\",\"status\":201,\"duration\":506.24,\"view
    \":19.37,\"db\":104.85,\"dd\":{\"trace_id\":\"4317749147917384263\",\"span_id
    \":\"2796771949324418788\",\"env\":\"\",\"service\":\"atlas\",\"version
    \":\"\"},\"ddsource\":[\"ruby\"],\"uuid\":\"bb5a9c16-1918-48bf-80bf-c8f00215497b
    \",\"remote_ip\":\"192.168.1.10\",\"request_id\":\"bb5a9c16-1918-48bf-80bf-
    c8f00215497b\",\"user_agent\":\"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)
    AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.0.0 Safari/537.36\",\"user
    \":\"admin\",\"organization\":\"hello\",\"auth_source\":\"ui\"}",
```

9.2.10 Cancel run (VCS-driven)

Container: tfe-atlas

```
"message": "2023-06-26 19:24:28 [INFO] [88bad7ed-4935-45b9-8ee5-937e9d25cf36] [dd.
    service=atlas dd.trace_id=000 dd.span_id=000] [Audit Log] {\"resource\":\"run
    \",\"action\":\"cancel\",\"resource_id\":\"run-HmBo3jdJsruuLneB\",\"organization
    \":\"hello\",\"organization_id\":\"org-Ur11wxfYrZ77Zqjb\",\"actor\":\"admin\",\"
    timestamp\":\"2023-06-26T19:24:28Z\",\"actor_ip\":\"192.168.1.10\"}",

"message": "2023-06-26 19:24:28 [INFO] [88bad7ed-4935-45b9-8ee5-937e9d25cf36] [dd.
    service=atlas dd.trace_id=000 dd.span_id=000] {\"method\":\"POST\",\"path\":\"/
    api/v2/runs/run-HmBo3jdJsruuLneB/actions/cancel\",\"format\":\"jsonapi\",\"status
    \":202,\"duration\":47.33,\"view\":1.79,\"db\":18.19,\"dd\":{\"trace_id
    \":\"1347663011854903801\",\"span_id\":\"1980572047794731214\",\"env\":\"\",\"
    service\":\"atlas\",\"version\":\"\"},\"ddsource\":[\"ruby\"],\"uuid\":\"88bad7ed
    -4935-45b9-8ee5-937e9d25cf36\",\"remote_ip\":\"192.168.1.10\",\"request_id\":\"88
    bad7ed-4935-45b9-8ee5-937e9d25cf36\",\"user_agent\":\"Mozilla/5.0 (Macintosh;
    Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.0.0
    Safari/537.36\",\"user\":\"admin\",\"auth_source\":\"ui\"}",
```

9.2.11 Discard run (VCS-driven)

Container: tfe-atlas

```
"message": "2023-06-26 19:52:59 [INFO] [ddac4d15-866c-459a-af9f-a0d4ef18e15f] [dd.
    service=atlas dd.trace_id=000 dd.span_id=000] [Audit Log] {\"resource\":\"run
    \",\"action\":\"discard\",\"resource_id\":\"run-gRViC3ERQ2f819B3\",\"organization
    \":\"hello\",\"organization_id\":\"org-Ur11wxfYrZ77Zqjb\",\"actor\":\"admin\",\"
    timestamp\":\"2023-06-26T19:52:59Z\",\"actor_ip\":\"192.168.1.10\"}",

"message": "2023-06-26 19:52:59 [INFO] [ddac4d15-866c-459a-af9f-a0d4ef18e15f] [dd.
    service=atlas dd.trace_id=000 dd.span_id=000] {\"method\":\"POST\",\"path\":\"/
    api/v2/runs/run-gRViC3ERQ2f819B3/actions/discard\",\"format\":\"jsonapi\",\"
    status\":202,\"duration\":196.72,\"view\":0.63,\"db\":77.13,\"dd\":{\"trace_id
    \":\"2710790823539404705\",\"span_id\":\"3887876551463909594\",\"env\":\"\",\"
    service\":\"atlas\",\"version\":\"\"},\"ddsource\":[\"ruby\"],\"uuid\":\"ddac4d15
    -866c-459a-af9f-a0d4ef18e15f\",\"remote_ip\":\"192.168.1.10\",\"request_id\":\"
    ddac4d15-866c-459a-af9f-a0d4ef18e15f\",\"user_agent\":\"Mozilla/5.0 (Macintosh;
    Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.0.0
    Safari/537.36\",\"user\":\"admin\",\"auth_source\":\"ui\"}",
```

9.2.12 Create organization token (same as `regenerate token`)

Container: tfe-atlas

```
"message": "2023-06-26 20:02:29 [INFO] [473be8b1-d7be-4da5-9bf0-7a20fec2deab] [dd.
    service=atlas dd.trace_id=000 dd.span_id=000] [Audit Log] {\"resource\":\"
    authentication_token\",\"action\":\"create\",\"resource_id\":\"at-
    uYbCrYEf5c4bUEUp\",\"organization\":\"hello\",\"organization_id\":\"org-
    Ur11wxfYrZ77Zqjb\",\"actor\":\"admin\",\"timestamp\":\"2023-06-26T20:02:29Z\",\"
    actor_ip\":\"192.168.1.10\"}",

"message": "2023-06-26 20:02:29 [INFO] [473be8b1-d7be-4da5-9bf0-7a20fec2deab] [dd.
    service=atlas dd.trace_id=000 dd.span_id=000] {\"method\":\"POST\",\"path\":\"/
    api/v2/organizations/hello/authentication-token\",\"format\":\"jsonapi\",\"status
    \":201,\"duration\":20.77,\"view\":1.62,\"db\":6.36,\"dd\":{\"trace_id
    \":\"1935707383909070806\",\"span_id\":\"4362189586206037735\",\"env\":\"\",\"
    service\":\"atlas\",\"version\":\"\"},\"ddsource\":[\"ruby\"],\"uuid\":\"473be8b1
    -d7be-4da5-9bf0-7a20fec2deab\",\"remote_ip\":\"192.168.1.10\",\"request_id
    \":\"473be8b1-d7be-4da5-9bf0-7a20fec2deab\",\"user_agent\":\"Mozilla/5.0 (
    Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome
    /114.0.0.0 Safari/537.36\",\"user\":\"admin\",\"auth_source\":\"ui\"}",
```

9.2.13 Create team token (same as `regenerate token`)

Container: tfe-atlas

```
"message": "2023-06-26 20:08:17 [INFO] [56eb4dae-eb21-428d-a0af-4d6d969bac4c] [dd.
    service=atlas dd.trace_id=000 dd.span_id=000] [Audit Log] {\"resource\":\"
    authentication_token\",\"action\":\"create\",\"resource_id\":\"at-
    bJVtGvG5rAL5ovPf\",\"organization\":\"hello\",\"organization_id\":\"org-
    Ur11wxfYrZ77Zqjb\",\"actor\":\"admin\",\"timestamp\":\"2023-06-26T20:08:17Z\",\"
    actor_ip\":\"192.168.1.10\"}",

"message": "2023-06-26 20:08:17 [INFO] [56eb4dae-eb21-428d-a0af-4d6d969bac4c] [dd.
    service=atlas dd.trace_id=000 dd.span_id=000] {\"method\":\"POST\",\"path\":\"/
    api/v2/teams/team-ywMigAWcApmCoKFf/authentication-token\",\"format\":\"jsonapi
    \",\"status\":201,\"duration\":21.14,\"view\":1.59,\"db\":5.4,\"dd\":{\"trace_id
    \":\"1256712621247484220\",\"span_id\":\"967436859127944885\",\"env\":\"\",\"
    service\":\"atlas\",\"version\":\"\"},\"ddsource\":[\"ruby\"],\"uuid\":\"56eb4dae
    -eb21-428d-a0af-4d6d969bac4c\",\"remote_ip\":\"192.168.1.10\",\"request_id\":\"56
    eb4dae-eb21-428d-a0af-4d6d969bac4c\",\"user_agent\":\"Mozilla/5.0 (Macintosh;
    Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.0.0
    Safari/537.36\",\"user\":\"admin\",\"auth_source\":\"ui\"}",
```

9.2.14 Create user token

Container: tfe-atlas

```
"message": "2023-06-26 20:14:26 [INFO] [5398c137-6c27-4182-b2e7-c7f0d94e5b23] [dd.
    service=atlas dd.trace_id=000 dd.span_id=000] [Audit Log] {\"resource\":\"
    authentication_token\",\"action\":\"create\",\"resource_id\":\"at-
    v1TKfBE8Chqf2ecx\",\"organization\":null,\"organization_id\":null,\"actor\":\"
    admin\",\"timestamp\":\"2023-06-26T20:14:26Z\",\"actor_ip\":\"192.168.1.10\"}",

"message": "2023-06-26 20:14:26 [INFO] [5398c137-6c27-4182-b2e7-c7f0d94e5b23] [dd.
    service=atlas dd.trace_id=000 dd.span_id=000] {\"method\":\"POST\",\"path\":\"/
    api/v2/users/user-PvcyWJWiVf1PQSR6/authentication-tokens\",\"format\":\"jsonapi
    \",\"status\":201,\"duration\":12.36,\"view\":2.03,\"db\":3.24,\"dd\":{\"trace_id
    \":\"2254994533781079943\",\"span_id\":\"1975888909477939647\",\"env\":\"\",\"
    service\":\"atlas\",\"version\":\"\"},\"ddsource\":[\"ruby\"],\"uuid\":\"5398c137
    -6c27-4182-b2e7-c7f0d94e5b23\",\"remote_ip\":\"192.168.1.10\",\"request_id
    \":\"5398c137-6c27-4182-b2e7-c7f0d94e5b23\",\"user_agent\":\"Mozilla/5.0 (
    Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome
    /114.0.0.0 Safari/537.36\",\"user\":\"admin\",\"auth_source\":\"ui\"}",
```

9.2.15 Publish module to private registry

Container: tfe-atlas

```
"message": "2023-06-26 20:32:00 [INFO] [0f3587c4-fe47-46df-9356-3f6a7c801242] [dd.
    service=atlas dd.trace_id=000 dd.span_id=000] [Audit Log] {\"resource\":\"
    registry_module\",\"action\":\"create\",\"resource_id\":\"mod-YhV7w55B7LH9mF91
    \",\"organization\":\"hello\",\"organization_id\":\"org-Ur11wxfYrZ77Zqjb\",\"
    actor\":\"admin\",\"timestamp\":\"2023-06-26T20:32:00Z\",\"actor_ip
    \":\"192.168.1.10\"}",
```

9.2.16 Publish provider to private registry

Container: tfe-atlas

```
"message": "2023-06-27 18:50:05 [INFO] [1e8d08b9-e36e-4196-9116-307c633926d6] [dd.
    service=atlas dd.trace_id=000 dd.span_id=000] [Audit Log] {\"resource\":\"
    registry_provider\",\"action\":\"create\",\"resource_id\":\"prov-n7e9MG9D1s8PUokT
    \",\"organization\":\"hello\",\"organization_id\":\"org-Ur11wxfYrZ77Zqjb\",\"
    actor\":\"tom_straub\",\"timestamp\":\"2023-06-27T18:50:05Z\",\"actor_ip
    \":\"192.168.1.10\"}",

"message": "2023-06-27 18:50:05 [INFO] [1e8d08b9-e36e-4196-9116-307c633926d6] [dd.
    service=atlas dd.trace_id=000 dd.span_id=000] {\"method\":\"POST\",\"path\":\"/
    api/v2/organizations/hello/registry-providers\",\"format\":\"jsonapi\",\"status
    \":201,\"duration\":44.24,\"view\":7.77,\"db\":12.28,\"dd\":{\"trace_id
    \":\"2100068035448008241\",\"span_id\":\"2539145137689530220\",\"env\":\"\",\"
    service\":\"atlas\",\"version\":\"\"},\"ddsource\":[\"ruby\"],\"uuid\":\"1e8d08b9
    -e36e-4196-9116-307c633926d6\",\"remote_ip\":\"192.168.1.10\",\"request_id\":\"1
    e8d08b9-e36e-4196-9116-307c633926d6\",\"user_agent\":\"go-tfe\",\"user\":\"
    tom_straub\",\"organization\":\"hello\",\"auth_source\":\"api\"}",

"message": "2023-06-27 18:50:39 [INFO] [a81eb08e-2a74-4298-8d16-c8d5f1753504] [dd.
    service=atlas dd.trace_id=000 dd.span_id=000] [Audit Log] {\"resource\":\"
    registry_provider_version\",\"action\":\"create\",\"resource_id\":\"provver-
    TC9rmugLV9BU7fzk\",\"organization\":\"hello\",\"organization_id\":\"org-
    Ur11wxfYrZ77Zqjb\",\"actor\":\"tom_straub\",\"timestamp\":\"2023-06-27T18:50:39Z
    \",\"actor_ip\":\"192.168.1.10\"}",
"message": "2023-06-27 18:50:39 [INFO] [a81eb08e-2a74-4298-8d16-c8d5f1753504] [dd.
    service=atlas dd.trace_id=000 dd.span_id=000] {\"method\":\"POST\",\"path\":\"/
    api/v2/organizations/hello/registry-providers/private/hello/random/versions\",\"
    format\":\"jsonapi\",\"status\":201,\"duration\":97.77,\"view\":24.9,\"db
    \":27.78,\"dd\":{\"trace_id\":\"20698542872263433\",\"span_id
    \":\"1826464858575608622\",\"env\":\"\",\"service\":\"atlas\",\"version
    \":\"\"},\"ddsource\":[\"ruby\"],\"uuid\":\"a81eb08e-2a74-4298-8d16-c8d5f1753504
    \",\"remote_ip\":\"192.168.1.10\",\"request_id\":\"a81eb08e-2a74-4298-8d16-
    c8d5f1753504\",\"user_agent\":\"go-tfe\",\"user\":\"tom_straub\",\"organization
    \":\"hello\",\"auth_source\":\"api\"}",

"message": "2023-06-27 19:06:39 [INFO] [624fa6ee-c27d-4df7-8b4d-6a7cf2dd1267] [dd.
    service=atlas dd.trace_id=000 dd.span_id=000] [Audit Log] {\"resource\":\"
    registry_provider_platform\",\"action\":\"create\",\"resource_id\":\"provpltfrm-5
    mHyfPky1ge86pFm\",\"organization\":\"hello\",\"organization_id\":\"org-
    Ur11wxfYrZ77Zqjb\",\"actor\":\"tom_straub\",\"timestamp\":\"2023-06-27T19:06:39Z
    \",\"actor_ip\":\"192.168.1.10\"}",

"message": "2023-06-27 19:06:39 [INFO] [624fa6ee-c27d-4df7-8b4d-6a7cf2dd1267] [dd.
    service=atlas dd.trace_id=000 dd.span_id=000] {\"method\":\"POST\",\"path\":\"/
    api/v2/organizations/hello/registry-providers/private/hello/random/versions
    /3.1.0/platforms\",\"format\":\"jsonapi\",\"status\":201,\"duration\":72.94,\"
    view\":10.2,\"db\":20.67,\"dd\":{\"trace_id\":\"3854175609003771184\",\"span_id
    \":\"4004712127517839897\",\"env\":\"\",\"service\":\"atlas\",\"version
    \":\"\"},\"ddsource\":[\"ruby\"],\"uuid\":\"624fa6ee-c27d-4df7-8b4d-6a7cf2dd1267
    \",\"remote_ip\":\"192.168.1.10\",\"request_id\":\"624fa6ee-c27d-4df7-8b4d-6
    a7cf2dd1267\",\"user_agent\":\"go-tfe\",\"user\":\"tom_straub\",\"organization
    \":\"hello\",\"auth_source\":\"api\"}",
```

10 Change Log

- [Nov/02/2023] - v1.00.1 Changes to Credits file
- [Oct/26/2023] - v1.00.0 GA Release

11 Credits

This document owes its existence to the invaluable contributions of the individuals mentioned below. Hailing from various functional areas and representing diverse departments within HashiCorp, these experienced professionals have played pivotal roles in writing, editing, reviewing, and ensuring the completion of this material. Their collective efforts have been instrumental in shaping the final output.

- Adam Cavaliere
- Adeel Ahmad
- Alex Basista
- Amy Brown
- Anna Chen
- Brandon Ferguson
- Chris Steinmeyer
- Chris Stella
- CJ Obermaier
- Emmanuel Rousselle
- Fraser Pollock
- Jenna Degaust
- Judith Malnick
- Kalen Arndt
- Kranthi Katepalli
- Kyle Seneker
- Michelle Roberts
- Prakash Manglanathan
- Purna Mehta
- Randy Keener
- Richard Russell
- Salil Subbakrishna
- Sean Doyle
- Sean Walker
- Simon Lynch
- Yogeshprabhu Jagadeesan
- Zeeshan Ratani