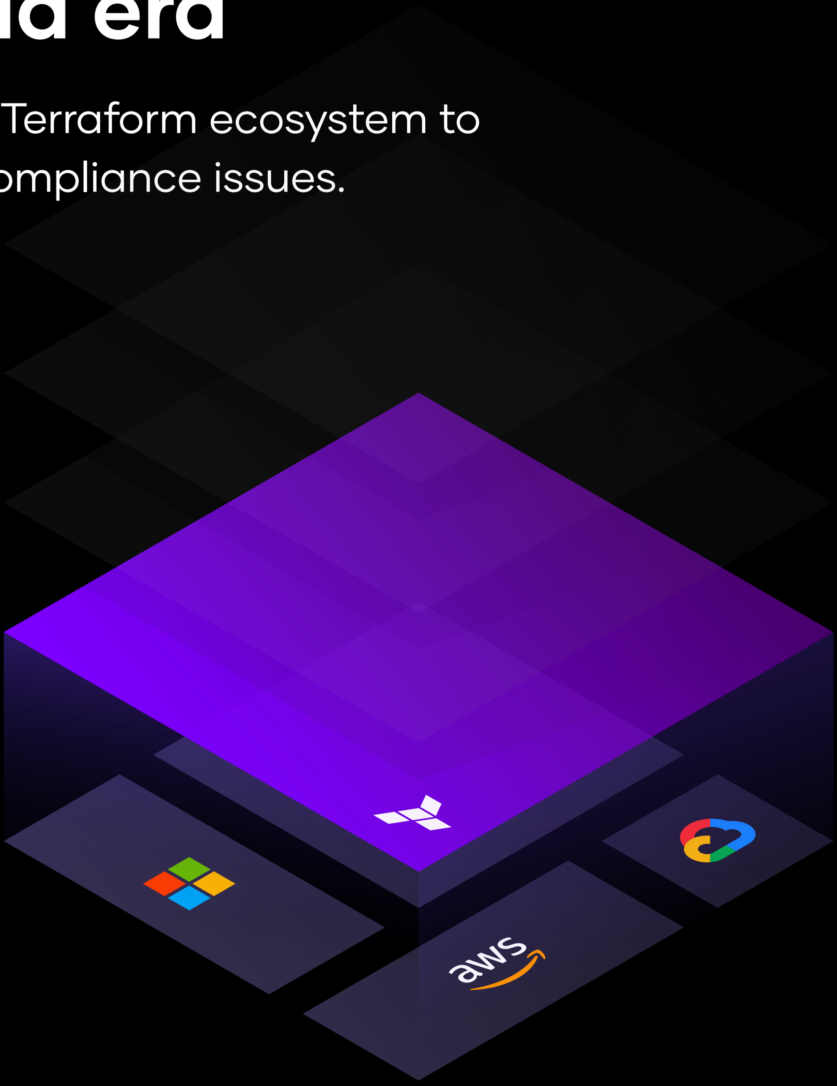
HashiCorp

Building secure and compliant infrastructure in the multi-cloud era

How to leverage the Terraform ecosystem to solve security and compliance issues.



WHITE PAPER

Contents

Executive summary

The way organizations provision infrastructure has significantly changed with the move from dedicated servers to capacity on-demand in the cloud. Homogeneous blueprints of infrastructure owned by IT have grown inefficient and outdated. In the cloud, infrastructure resources must be readily available across a variety of providers.

While infrastructure automation underpins the move to the cloud and the overall modernization of application delivery, this shift requires organizations to address potential security shortcomings in their traditional provisioning processes, including:

- Slow, error-prone, manual workflows and ticketing systems
- A lack of built-in security controls or secure templates for infrastructure code reuse
- Inconsistent or non-existent policy enforcement processes
- No system to detect non-compliant or drifted infrastructure
- Insufficient auditing and observability

To successfully tackle these issues, teams must first think about how infrastructure and applications interact with two core audiences with very different priorities:

- **Developers and application teams** who consume the infrastructure to deploy and manage their applications. Their priority is to work with infrastructure in an easily consumable way that makes the deployment process easier.
- **Operators or platform teams** who provide the infrastructure in a self-service way for their end developers. These teams solve problems such as, “How do I make the provisioning process repeatable?” “How do I implement proper policy guardrails?” and “How do I ensure security and compliance for the duration of the resource lifecycle?”

This white paper will discuss how organizations can leverage infrastructure as code (IaC) to help solve many of the security and compliance challenges they encounter. This is done by implementing six fundamental practices for your HashiCorp Terraform workflow that can help ensure secure infrastructure from the first provision to months and years in the future.

Introduction

The cloud has changed the way we provision infrastructure for the better, from dedicated servers to capacity on-demand. Previous homogeneous blueprints of infrastructure owned by IT have transitioned to capacity on demand and in many cases developers can now provision the infrastructure resources they need across a variety of providers.

In fact, infrastructure automation underpins organizations' move to cloud and the modernization of application delivery. But the cloud also opens up new security challenges. Maintaining a strong security posture is critical — it seems like there are new breaches reported every day. And for many organizations, security is not just a nice-to-have enhancement, but a mandate that must be followed due to regulatory requirements like FISMA, GDPR, Executive Cyber Security orders, and HIPAA.

As organizations make their move to the cloud, how do they make sure their developers provision infrastructure with security and compliance in mind? With the transition to cloud we must think about how infrastructure and applications interact for two core audiences:

1. **Operators or platform teams**, whose job is to provide the infrastructure in a way that is self-service organization developers. The challenge they must solve is that there are many developers, so how do orgs make the provisioning process repeatable and with the proper guardrails in place to ensure the security of systems and resources for the duration of their lifecycle?

2. **Developers and application teams** use the tools and infrastructure to deploy and manage their applications. Their priorities are, "How do I work with this infrastructure in a consumable way that makes the deployment process easier?" As a result, they are less knowledgeable and less focused on the security and compliance aspect of their infrastructure, so it must be baked in.

The challenge of secure infrastructure

Infrastructure automation underpins organizations' move to cloud and the overall modernization of application delivery. Unfortunately, the cloud's benefits don't come without challenges and cloud usage opens a litany of security issues that organizations must address.

Up to 70% of cloud related security incidents are due to resource misconfiguration and over 20% are from resources IT did not previously know about. According to the 2023 HashiCorp State of Cloud Strategy Survey, more than 31% of organizations believe they lack adequate controls and oversight to prevent cloud misconfiguration mistakes, directly leading to these negative outcomes. In fact, Gartner estimates that by the end of 2024, the majority of organizations will continue to struggle with measuring cloud risk, often significantly underestimating it. As a result, Gartner predicts that through 2025, 90% of organizations that fail to properly secure their public cloud use will inadvertently share sensitive data — with 99% of these cloud-related security failures being self inflicted.

To prevent this, as part of the provisioning-automation process, central IT, security, compliance, and finance teams must overcome:

- Non-compliance with regulatory requirements
- Slow, error prone, manual workflows and ticketing systems
- Lack of consistent policy enforcements processes
- Different policy engines and considerations for different infrastructure providers

Any cloud misconfiguration can be a potential entry point for attackers, and there are thousands of policies that must be adhered to in a cloud-native environment. Errors like overlooked encryption or firewall rules can compromise an organization's infrastructure in potentially disastrous ways. Inconsistently setup infrastructure or drift, not to mention simple human error, can easily introduce vulnerabilities, push infrastructure out of compliance, or expose critical systems and data. Cloud infrastructure also has to be vetted for data privacy policies and reporting for federal government requirements, GDPR, and HIPAA. This brings in thousands of other considerations for provisioning infrastructure that must be addressed.

Infrastructure as code (IaC) tools like HashiCorp Terraform increase provisioning velocity, but do not change this reality alone. To ensure security at all times, HashiCorp's commercial products — Terraform Cloud and Terraform Enterprise — provide organizations with better workflows and tools to identify and respond to these threats. Codifying infrastructure makes it scannable, allowing it to be cross-checked across security and compliance benchmarks using policy as code. By codifying thousands of policies and then running them against infrastructure as code configurations, operations teams can maintain provisioning velocity while reducing required code review for security teams.

Here are key HashiCorp's learnings on how best to leverage the Terraform ecosystem to solve these security and compliance issues.

Security and compliance remains a manual process for many teams

According to a survey by the developer security company Snyk, 95% of cloud-native companies use infrastructure automation and 50% use infrastructure as code. Public Terraform modules available in the Terraform Registry provide an easier means for provisioning infrastructure via automation. But just as any open source library must be reviewed prior to shipping in production, Terraform modules must be vetted for security and compliance. The need for code review could explain why only 33% of the survey respondents fully automate their provisioning pipeline.

Many teams, particularly teams that build custom solutions around Terraform Community, turn to a manual process for reviewing Terraform configurations. When writing Terraform code, the workflow around the community edition generally incorporates four steps:

1. First, practitioners write new configurations or make a change to an existing configuration. Then they check this configuration into version control.
2. A security and compliance person reviews the code in version control. Depending on their workload, this can create a bottleneck for implementing infrastructure changes.
3. Upon review, a developer executes a Terraform plan. Terraform creates a preview of the outlined infrastructure changes.
4. If everything is as expected, a team member can then safely apply infrastructure changes using the Terraform command line interface (CLI).

The problem is that this process doesn't scale easily as organizations increase their cloud footprint and adopt a multi-cloud strategy.

Code review is more arduous than it should be

Often teams that build custom solutions around Terraform Community lack a standardized set of Terraform modules that can be used in configurations. This introduces an even more critical need for manual security and compliance review. Multiple Terraform operators create "snowflake" modules, individual modules that provision similar infrastructure. In addition to wasting effort re-inventing the wheel, not all teams may be familiar with all of the security and compliance requirements for their organization. The following trade-offs must then be made:

- **Velocity over security:** Should DevOps managers continue to allow their teams to write their own modules without manual review from security personnel, to ensure they are able to move quickly?
- **Security over velocity:** Should DevOps insist on a manual code review process by team members familiar with compliance benchmarks like CIS, as well as all current security and compliance requirements set by the organization?

Some organizations may not see provisioning velocity as a critical need compared to security. But in an outage or security breach, manual processes can rob operations teams of critical hours that can mean the difference between a minor incident and a major problem that makes headline news. HashiCorp has established the best practice steps we suggest organizations adopt in order to establish secure, compliant infrastructure built on consistent workflows and policies while being scalable to today's modern cloud environments.

Six key practices for secure infrastructure

1. **Address the provisioning skills gap**
   Many organizations are encountering increased complexity and skills gaps across their DevOps teams. This results in longer development times, more errors due to misconfigurations, and increased risk. To combat this, organizations must find a unified multi-cloud provisioning solution to reduce inconsistent workflows and create a golden provisioning workflow for any type of infrastructure.

2. **Reuse infrastructure for consistency**
   Once you have a unified provisioning solution and begin creating your workflows, it is necessary to codify infrastructure for reuse. This enables consistent provisioning of infrastructure to remove misconfigurations. It also lets platform teams enable self-service development while limiting security risks. This is done by creating golden configurations that have been reviewed, tested, validated against misconfigurations, and published by central IT to a centralized library. From there they are easily discoverable by end users.

3. **Create guardrails**
   With reusable infrastructure in place, the next step is to ensure your team is consistently and compliantly leveraging it. This can be done by defining policies and guardrails through policy as code for all infrastructure, which ensures security and compliance while reducing costs.

4. **Enforce guardrails at the time of provision**
   Creating policies and guardrails is the start, but traditionally these approaches still had to be validated through time-consuming manual processes. To avoid this, organizations must establish and enforce the ability to create guardrails and have them automatically enacted during the provisioning process, which automatically validates the code against these policies and blocks the provisioning of non-compliant resources.

5. **Enforce guardrails continuously post Day 1 provisioning**
   Once an organization is provisioning consistent, compliant resources from the start, it also needs to

ensure those resources aren't improperly updated or changed on Day 2 and beyond. At scale, this can be done through automation tools that can range from alerting operators to potentially unwanted changes to preventing them from taking effect at all.

6. **Add observability and visibility**
   Finally, organizations, and especially those in highly regulated industries, need observability into this process to ensure they are remaining compliant and secure. When properly established, observability also enables required auditing and reporting on systems and infrastructure. That makes it easy to share this information with internal stakeholders and external regulators to ensure the security of systems and key data as well as continued regulatory compliance.

Critically, Terraform can help you achieve all six of these steps to establish secure infrastructure for your organization.

1. Address the provisioning skills gap

As noted in the 2023 HashiCorp State of Cloud Strategy Survey, organizations rank skills gaps as the most common barrier to multi-cloud adoption. In addition, skills-related factors issues such as siloed teams, lack of training, manual operations, and budget-constrained headcounts also complicate organizations' ability to operationalize multi-cloud. This lack of resources can inflate costs and push developers into roles they aren't prepared for.

As a result, skills and resources gaps make it harder to properly provision infrastructure, resulting in errors due to misconfigurations and poor security configurations, which increases risk. Organizations can address this problem by employing a unified provisioning solution for multi-cloud, such as Terraform, to decrease the number of workflows by creating unified golden provisioning workflows for any type of infrastructure. This allows operators and developers to learn one process and tooling, while limiting misconfiguration from lack of expertise.

Establishing this workflow requires features that allow your teams to collaborate across all your infrastructure and ensure compliance and consistency. Terraform provides for this while enabling DevOps teams to establish and manage infrastructure in their preferred manner:

- Infrastructure as code with HashiCorp Configuration Language (HCL) and Cloud Development Kit for Terraform (CDKTF)
- The ability to connect to version control systems (VCS)
- Remote operations
- Dynamic provider credentials

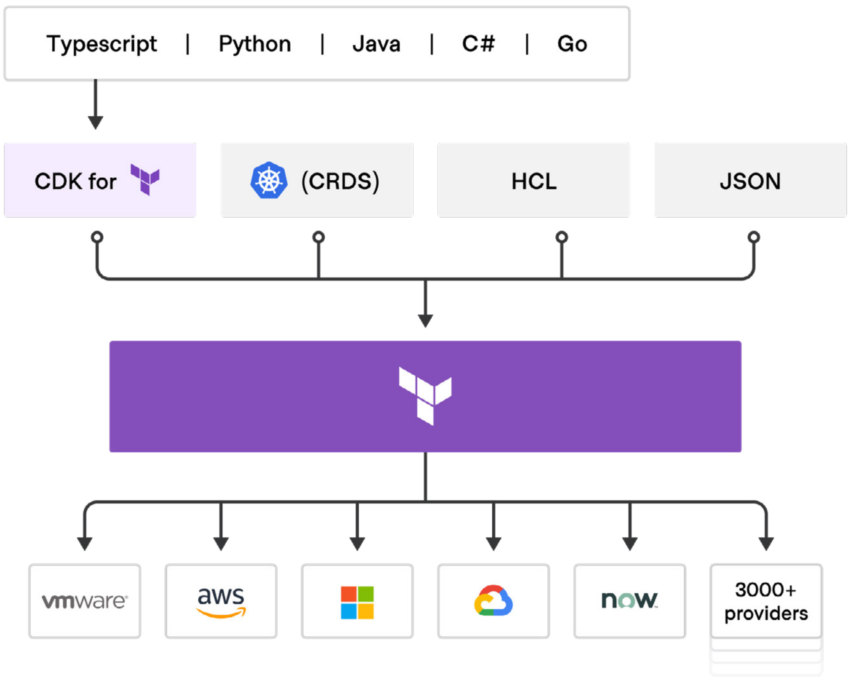
- Vault-backed dynamic credentials
- No-code provisioning

Terraform also lets you provision any infrastructure and leverage security tools you already use with a vast ecosystem of more than 3,000 providers and more than 250 technology partners and integrations.

Infrastructure as code with HCL and CDKTF

Terraform is built to give you the options to best define your infrastructure as code (IaC). The first way is to leverage HCL, which is a simple syntax that allows you to better leverage Terraform's capabilities. It gives you a significant degree of control over your infrastructure in a way that's more 'human-readable' than other configuration languages such as YAML and JSON. As a result it is easier to review and understand what your code is saying and doing.

That said, some developers have preferred languages they like to work with, or they may not want to learn a new configuration language. That's where the Cloud Development Kit for Terraform comes in. CDKTF lets you use familiar programming languages to define and provision your infrastructure.



With CDKTF, developers can set up their infrastructure as code without context switching from their familiar programming language, using the same tooling and syntax to provision infrastructure resources that they use to define the application's business logic. Teams can collaborate in a familiar syntax while still leveraging the Terraform ecosystem and deploying their infrastructure configurations via established Terraform provisioning pipelines.

CDK for Terraform allows you to manage complexity and reduce code duplication by creating custom abstraction layers, referred to as constructs. Constructs let developers reuse existing resource configurations written in their preferred programming language rather than defining resources by hand, which simplifies development and speeds delivery of new features and services.

Adding further flexibility, the choice of whether to work with Terraform in HashiCorp Configuration Language (HCL) or one of CDKTF's supported programming languages can be made on a team-by-team or project-by-project basis. This is because CDKTF interoperates with all existing Terraform providers and modules, and the JSON configuration file that is synthesized from your application code can be deployed with Terraform directly. These options help mitigate the skills gap issues that arise with a cloud transformation while enabling greater reuse of code to promote greater security and compliance.

Connect to VCS

Another way to ensure collaboration and compliance across your team is through the use of version control systems. In addition to the CLI-driven workflow, Terraform offers a VCS-driven workflow that automatically triggers runs based on changes to your VCS repositories. The CLI-driven workflow allows you to quickly iterate on your configuration and work locally, while the VCS-driven workflow enables collaboration within teams by establishing your shared repositories as the source of truth for infrastructure configuration. VCS workflows also allow for stronger version control and code reviews to ensure continued compliance and prevent errors. This is a key step in setting up secure and compliant workflows across your entire cloud stack.

Remote operations

Terraform Cloud is designed as an execution platform for Terraform, and can perform Terraform runs on its own disposable virtual machines. This provides a consistent and reliable run environment, and enables advanced features like Sentinel policy enforcement (more on that below), notifications, version-control integration, and more.

Terraform runs managed by Terraform Cloud are called remote operations. Remote runs can be initiated by webhooks from your VCS provider, by UI controls within Terraform Cloud, by API calls, or via the Terraform CLI.

To secure these remote operations, Terraform Cloud Agents is a paid feature that can allow Terraform to communicate with isolated, private, or on-premises infrastructure. The agent polls Terraform Cloud or Terraform Enterprise for any changes to your configuration and executes the changes locally, so you do not need to allow public ingress traffic to your resources. Agents allow you to control infrastructure in private environments without modifying your network perimeter, increasing security.

Terraform Cloud Agents also support running custom programs, called hooks, during strategic points of a Terraform run. For example, you may create a hook to dynamically download software required by the Terraform run or send an HTTP request to a system to kick off an external workflow.

Dynamic provider credentials

Storing long-lived static credentials in Terraform presents severe security risks, as cloud credentials are highly sensitive and can grant powerful privileges. Terraform Cloud's dynamic provider credentials provide short-lived just-in-time (JIT) access to HashiCorp Vault and official cloud providers (AWS, Microsoft Azure, and Google Cloud). This authentication method is enabled by implementing the OpenID Connect (OIDC) standard using a workload identity token generated by the Terraform Cloud platform.

Using dynamic provider credentials removes the need to store long-lived credentials in Terraform and significantly limits the impact of accidental credential exposure and reuse. You also gain more granular permissions control over your Terraform operations. You can scope privileges down to the run phase, workspace, project, or organization, enabling the least privilege principle.

Vault-backed dynamic credentials

Vault-backed dynamic credentials combine dynamic provider credentials with Vault secrets engines to offer a consolidated workflow. This approach authenticates Terraform runs to Vault using workload identity tokens generated by Terraform Cloud, then uses Vault secrets engines to generate dynamic credentials for the AWS, Azure, and Google Cloud providers. Vault-backed dynamic credentials represent a significant enhancement for customers already using Vault for on-demand cloud access and for any organization seeking to reduce the risks of managing credentials.

No-code provisioning

No-code provisioning allows you to provision infrastructure with modules already built and approved by your organization. This gives developers with limited infrastructure knowledge a way to effectively deploy and manage resources. Organizations can use this approach to adopt a self-service model where your Terraform experts can write configurations using HCL, and others can consume and reuse the modules they create. No-code provisioning enables collaboration among developers and paves the way for infrastructure as code at scale. Terraform projects help enable guardrails around self-service provisioning by allowing team permissions, variable sets, and policies to be scoped at the project level for all existing and new workspaces.

Now that you have the tools and structure to create consistent workflows across your DevOps team, the next step is to establish the processes for reuse of this infrastructure.

2. Reuse infrastructure for consistency

Once you have a unified provisioning solution and begin creating your workflows, it is necessary to codify this infrastructure for reuse. To do this, operators create golden configurations that enable developers while limiting security risks.

Terraform enables this with an automated, repeatable secure framework for your infrastructure.
It provides features to create a producer and consumer model with the ability to build a library of approved infrastructure as code that can then be leveraged by your developers. When following this methodology, many organizations find that they can actually publish 60% or more of their resources for reuse, reducing development times and costs. There are two primary paths for distributing your reusable code: leveraging the Terraform public registry or building out your own private registry of modules, providers, and policies.

The Terraform public registry

The public Terraform Registry is an official repository enabling teams to quickly and easily discover Terraform providers that power all of Terraform's resource types, find modules for quickly deploying common infrastructure configurations, and leverage a library of policies that can be used within Terraform to accelerate your adoption of policy as code. Here you can find official providers and modules owned and maintained by HashiCorp, owned and maintained by our approved technology partners, or developed by third parties and distributed to the larger Terraform community. The registry is a great starting point to build and share proven code that can address your organization's key use cases.

Private registries

Many organizations use modules, providers, or policies that cannot or do not need to be publicly available, but must still be easily accessible across their teams. In these instances, teams could load private modules directly from version control systems and other sources, but those methods do not support version constraints or a browsable marketplace, both of which help enable the producer-consumer content model essential in larger organizations. The answer is the creation of private registries to give your teams access to a common set of providers and modules.

Terraform Cloud includes a private registry that is available to all accounts. Unlike the public registry, the private registry can import modules from your private VCS repositories on any of Terraform Cloud's supported VCS providers. It also lets you upload and manage private, custom providers through the Terraform Cloud API and curate a list of commonly used public providers and modules. You can seamlessly reference private modules and providers in your Terraform configurations, and the Terraform Cloud UI includes a searchable marketplace to help users find the components they need.

Leveraging the appropriate registry system gives you a set of reusable code that is easily accessible across your teams. The next step to securing your infrastructure is to establish guardrails and policies that ensure everything is implemented in a secure and compliant manner.

Integration with HCP Packer

Terraform's pre-written modules let teams dynamically provision multi-cloud self-service infrastructure, but organizations also need a way to manage their provisioning pipelines at the image level. HCP Packer helps platform teams ensure all image artifacts provisioned are secure, compliant, up-to-date, and easily trackable. With the HCP provider, Terraform can access the Packer data source to query image channels in Terraform configuration files rather than hard-coding them. This allows users to track image metadata and storage location, and automatically provides the correct image to developers in Terraform. This close integration of Terraform and HCP Packer enables users to unify their infrastructure workflows and create a successful golden image pipeline.

3. Create guardrails

Once you create reusable infrastructure, you need to ensure your team is consistently and compliantly leveraging it. Provisioning doesn't happen in a vacuum. Different business units — like finance, security, and compliance, for example — must ensure that their goals are met any time a change is introduced to infrastructure. Historically, this necessitated a pass-off to other teams for code review, which slows down DevOps teams.

Policy as code speeds this review process by enabling organizations to define requirements from these teams as code, which enables automated testing and automated deployment. This can be done by defining policies and guardrails that enforce security, maintain compliance, and monitor costs. The best way to ensure secure and compliant infrastructure is to integrate policy as code across all layers of your infrastructure. Policy as code reduces human error and misconfigurations, preventing some of the easiest attack vectors for security breaches. Terraform has a number of features to help implement this:

- **HashiCorp Sentinel**: Allows you to write custom policies that are automatically enforced in the provisioning workflow.
- **Native Open Policy Agent (OPA) integration**: Leverage your organization's standard OPA across infrastructure managed by Terraform
- **Run tasks**: Implement pre-written policies from third-party partners to ensure continued security and compliance beyond Terraform

HashiCorp Sentinel

Sentinel is HashiCorp's policy as code framework, which is based on a powerful declarative policy language to enable fine-grained, logic-based policies. It enables guardrails to be put in place on automation while allowing the codification and automatic enforcement of business requirements in critical areas of your infrastructure. Historically, these types of guardrails often required human enforcement through time-consuming and error-prone ticketing systems. In addition, these manual processes don't scale well in the cloud. By leveraging Sentinel policies in your Terraform workflows, you can ensure secure and create compliant infrastructure across all your teams.

You can leverage pre-defined policies to expedite development and overcome skills gaps or develop your own policies internally to fit your particular business and security needs. These policies are then grouped into reusable policy sets that are applied globally or at a project and workspace level, boosting consistency across your organization.

Native Open Policy Agent (OPA) integration

Sometimes organizations use a more generalized policy engine like Open Policy Agent (OPA) for Terraform and other tools. OPA is a general-purpose policy engine that unifies policy enforcement across the stack and has native integration into Terraform Cloud. OPA uses a high-level declarative language called Rego that, much like Sentinel, is used to enforce controls using policy as code. Organizations that have already invested in OPA may want to leverage those existing skills and policies. Native OPA integration allows customers to migrate their existing Rego-based policies into Terraform Cloud with minimal engineering effort. As with Terraform and infrastructure as code, OPA lets users rely on a single language for policy as code for different configurations and resources. This makes it easy to create and share reusable policies with the rest of the organization.

Teams can choose their desired policy framework and collaborate seamlessly across their organization. Native OPA support also allows Terraform Cloud users to take advantage of the platform's mature policy workflows. These workflows help simplify policy enforcement and free users from the burden of using custom or third-party tools to connect OPA and Terraform Cloud.

Native OPA support in Terraform Cloud gives teams an additional framework choice when adopting a policy as code approach. In some organizations, multiple teams are responsible for different policies, such as compliance or operational best practices. Limiting these teams to a single policy framework can lead to suboptimal policy enforcement processes. OPA and Sentinel can operate side-by-side so each team can choose its preferred framework and work together seamlessly.

Sentinel and native OPA support in Terraform Cloud both include:

- **Individually managed or VCS-connected policies**: Edit policy code directly in the UI, or use the recommended approach of connecting to a version control repository containing your policies.
- **Policy sets**: Group policies and enforce them on projects and workspaces.
- **Enforcement levels**: Define behavior for when policies fail as part of a policy check.
- **Policy overrides**: Enable authorized users to override a failed policy check and continue the execution of a run.
- **Role-based access control (RBAC)**: Grant team permissions to manage policies and override policy failures.
- **Auditing**: Policy results are included in audit logs, enabling centralized visibility.

Run tasks

In addition to your integrated Terraform workflow, run tasks are a simple but flexible method for integrating outside tools into your provisioning workflow. Third-party partner integrations are discoverable in the Terraform Registry, are configured once in a Terraform Cloud organization, and can be attached to any number of workspaces.

Depending on partner support, run tasks execute in the pre-plan, post-plan, or pre-apply phase of a Terraform run. Pre-plan tasks are designed to evaluate Terraform code, while post-plan/pre-apply run tasks gain the additional context of the computed plan including all resources to be provisioned.

Run tasks help your teams add additional security and cost conditions into every Terraform run. This provides the additional functionality of these applications without required custom code and having to ensure they are complying with the internal policies crucial to your organization. Relevant use cases for run task integrations include:

Security and compliance: Scan for common compliance benchmark rules

There are thousands of basic security and compliance rules for provisioning in the cloud, defined by frameworks such as CIS and NIST. You can avoid provisioning misconfigurations that cause security issues using tools like Prisma Cloud Code Security, Snyk, Tenable, Moderne, Sophos, Aqua Security, Firefly, and Lightlytics.

Additionally, run tasks help ensure compliance with regulations like HIPAA, GDPR, and PCI-DSS. Run tasks enables your teams to scan for security and compliance threats across all of your workspaces before any changes are ever applied.

Productivity: Increase provisioning velocity

Ensuring that any infrastructure change adheres to security, cost, and compliance benchmarks is critical for any business. By shifting these concerns left into the Terraform workflow, your organization can reduce manual review and increase productivity. The HCP Packer run task helps ensure only approved images are provisioned, blocking the use of revoked versions.

Cost savings: Provide visibility and thresholds for costs

Operators often have no idea how much the infrastructure they provision costs. You can gain visibility into the costs of infrastructure prior to any change using Infracost, Vantage, or Kion to help your organization proactively work to reduce cloud spend.

Discover integrations in the growing run task ecosystem on the Terraform Registry. You can also build your own custom Run Task integrations.

Terraform was built to allow for multiple policy creation and enforcement methods to best fit the needs of your organization and development teams. Once you have built your policies and made them accessible across your org, the next step is to ensure they are being enforced during provisioning.

4. Enforce guardrails at the time of provisioning

Creating policies and guardrails is the first step. Traditionally, these types of policies had to be validated and enforced through manual processes. But that approach doesn't scale. In the cloud, the policies you created must instead be deployed and enforced through automated provisioning processes so they are automatically validated against your confirmed requirements.

For example policies might validate that an end user is consuming approved modules rather than custom code, make sure that your infrastructure resources are tagged for visibility, enforce the location of data storage for GDPR compliance, or check that storage buckets are not publicly accessible. Terraform gives your organization the tools to integrate these types of security into automated provisioning workflows while allowing you to establish the sophistication and methods of enforcement.

Automatic policy enforcement

Different types of policies require different levels of enforcement. The automation of your enforcement must match the use case. In some cases it might be OK to deploy non-compliant code, perhaps because of a necessary exception, or to fix a larger issue that could cause downtime or greater risks to your systems. On the other hand, certain types of deployments can never be allowed, like posting critical security information on a publically accessible node or changes that will break your network infrastructure. Terraform allows platform and security teams to define how to deal with different types of policies and infrastructure to fit the larger needs of your organization. These flexible enforcement levels are:

- **Advisory**: Warns when a policy breaks, but doesn't stop the code from being run.
- **Soft mandatory**: Warns of the broken policy and won't allow it to run without a manual override that explicitly gives permission for the policy to be broken and recording the incident.
- **Hard mandatory**: When a policy is broken, the code will not be allowed to run and the apply will be halted until it is brought into compliance.

5. Enforce guardrails continuously post Day 1 provisioning

Once an organization is provisioning consistent, compliant resources from the start, they must deal with the challenges of remaining in compliance. This means ensuring that nothing is improperly updated or changed on Day 2 and beyond. Changes made outside of the normal provisioning process, or health events that occur during operations, can bring your infrastructure out of compliance or compromise its security. As a result, you need tools to watch your infrastructure and look for potential infrastructure drift to inform you when these resources unexpectedly change from their initial state.

Drift detection and continuous validation

Here's why drift is so important to manage. As cloud adoption grows, organizational resources and processes become increasingly complex, which can create inconsistencies around the state of the infrastructure. Without standard procedures, notifications, and guidelines for adjustments, even temporary changes or the smallest tweaks to infrastructure can have significant impacts on the business, including unplanned downtime, security incidents, and unused resources.

Most importantly, unrecognized infrastructure drift creates multiple risks that need to be addressed before they become real problems. Drift can dramatically increase the probability of critical data exposures, perhaps due to mission-critical systems mistakenly left open to public access or unknown resources being left unsecured.

Additionally, development teams unaware of production environment changes not reflected in the IaC systems will almost certainly have to contend with applications suddenly crashing and deployment projects that unexpectedly fail.

The challenge is that once infrastructure is provisioned, how do you ensure the actual state of infrastructure reflects the known and recorded infrastructure state? If changes occur, notifications or alerts need to be sent and action must be taken to correct the detected infrastructure drift. To help you do this, Terraform's drift detection functionality runs continuous checks against the infrastructure to validate that it matches the last known Terraform state. It detects when resources have changed from what Terraform has reflected in the state file and sends an alert to the team so they can take action. Drift detection is a key tool to ensure the continued security and compliance of your infrastructure, minimizing risk, downtime, and costs.

Centralized visibility is critical for drift detection

Ultimately, security teams must be concerned with risks inherent in infrastructure drift. Terraform can solve this by functioning as a development team's all-in-one automated provisioning and central management system, enabling development teams to continuously monitor the infrastructure state to detect changes. Operating from a consolidated environment, the system should be able to send immediate notifications to the appropriate teams so they can take specific corrective actions any time a resource is altered.

For CISOs concerned with narrowing security gaps — both the kind they know about and the previously undetectable ones created by infrastructure drift — this type of solution can help strengthen the organization's overall security posture without adding undue operational burdens.

Continuous validation

Continuous validation, the next step in Terraform's investment in infrastructure management, expands checks on your infrastructure beyond just configuration drift. This feature provides long-term visibility and checks on the health of your infrastructure to ensure it is working as expected. Users can add custom assertions to a Terraform configuration and/or modules via check blocks. Terraform Cloud then continuously validates these assertions and notifies operators when a check fails.

This paints a more comprehensive picture of your infrastructure health beyond a single point-in-time validation of the run prior to an apply. This helps organizations standardize their usage of Terraform for Day 1 provisioning and Day 2 infrastructure management, minimizing risk, downtime, and costs.

Once you have tools in place to establish and maintain your guardrails as well as drift detection to warn you of unwanted or unexpected changes, the final step is to establish auditing and observability systems to fully understand and report on your compliant infrastructure to key regulatory bodies and internal stakeholders.

6. Add observability and visibility

All organizations, and especially those in highly regulated industries, need observability into their infrastructure to ensure they are remaining compliant and secure. They also need to audit and report on their infrastructure, whether for internal stakeholders or external regulators. The tools you leverage to create and manage your infrastructure should be built with this in mind, with turnkey auditing and reporting so that both technical and non-technical team members can leverage and share it. This helps limit time-consuming manual processes burdening a small group of your DevOps team and ensures that new issues are quickly alerted to the larger team for remediation. Terraform has a number of key integrations to help with both the observability and remediation of issues with your infrastructure as well as the auditing and reporting of relevant information:

Observability of your infrastructure

As the system of record for all types of infrastructure, Terraform provides inherent visibility into your cloud estate. Workspaces in Terraform Cloud collect and isolate everything Terraform needs to run: the configuration, variable values, and state data. State represents Terraform's mapping of real world resources to your configuration. Terraform Cloud provides encrypted and versioned state storage with role-based access controls to prevent unauthorized access to sensitive state data. State versioning allows teams to gain insight into how a workspace's resources have changed over time, and in extreme cases, an entire state can be rolled back to respond to mistakes.

The projects and workspaces view gives organizations visibility across all Terraform configurations. Workspaces can be filtered by project, run status, health state, and tags. Tags are custom metadata applied to workspaces to bring additional contextual information and assist with filtering and reporting on workspaces. Tags are also made available to the Sentinel policy as code framework, so you can dynamically apply policies based on whether or not a specific tag has been applied to a workspace.

Explorer for Terraform Cloud provides an even wider range of valuable workspace data across your organization. This includes information on providers, modules, and Terraform versions in use. Explorer also provides a consolidated view of health checks from drift detection and continuous validation. These views help teams ensure their environments have the necessary up-to-date versions of Terraform, modules, and providers while tracking health status to ensure security, reliability, and compliance.

Audit logs and sharing

Another key requirement for maintaining compliant infrastructure is the ability to audit and compile reports that can be reviewed by internal security and compliance teams as well as shared externally to regulators and other interested parties. This can be challenging, as improperly managed cloud resources can become a black hole once they are provisioned and deployed. To avoid this, Terraform offers a rich audit logging feature set for organizations that need to oversee retrospective activities on their infrastructure over time. Audit logs emit information whenever any resource managed by Terraform is changed, so teams can understand what changes were made and by whom. Audit logs are exposed in Terraform Cloud via the audit trails API, and Terraform Enterprise provides log forwarding.

Here are examples of Terraform audit logging use cases leveraged by HashiCorp Terraform customers:

- **Monitor provisioning activities:** Audit logs track all activity happening in the platform including user logins, changes to settings and variable values, and provisioning runs.
- **Policy check overrides:** If a soft mandatory policy fails, users with permission to override policies will be presented with an Override & Continue button in the run. With access to these logs, organizations can see who overrode a policy and when.
- **Policy-related events help you see the effects of new policies or policy changes:** If an organization notices a lot of policy failures or overrides, it can educate developers or update the policy evaluation itself.
- **Keeping track of destroyed infrastructure:** Through the audit trail, you can identify destructive actions like the execution of a destroy plan or deletion of a workspace, see which user or team was responsible, and apply measures to prevent a recurrence.

Audit logs help capture events that can show who did what activity and how the system behaved. With audit logs, administrators, developers, and site reliability engineers (SREs) can get a complete picture of normal and abnormal events on the organizational level. Additionally, these logs can be collected by one or more external systems to provide increased observability, assistance complying with log-retention requirements, and information gathering and sharing during troubleshooting.

The way forward

The transition to cloud infrastructure can be challenging, but also presents an opportunity for a fresh start. As outlined in this paper, proven tools and best practices can help organizations establish and maintain secure infrastructure. HashiCorp and its cloud infrastructure automation tools, including Terraform, were created to help organizations successfully deliver business outcomes at scale across multiple clouds.

Establishing secure workflows and infrastructure guardrails is just the beginning of the larger cloud security journey. After setting up secure and compliant infrastructure, the next step is to establish zero trust security best practices across your organization. HashiCorp's approach to identity-based security and access provides a solid foundation for companies to safely migrate and secure their infrastructure, applications, networks, and data. We are working hard to support and accelerate the adoption of zero trust thinking across all aspects of your cloud infrastructure. Learn more about next steps and how we can help on our Zero Trust Security page.

HashiCorp

www.hashicorp.com