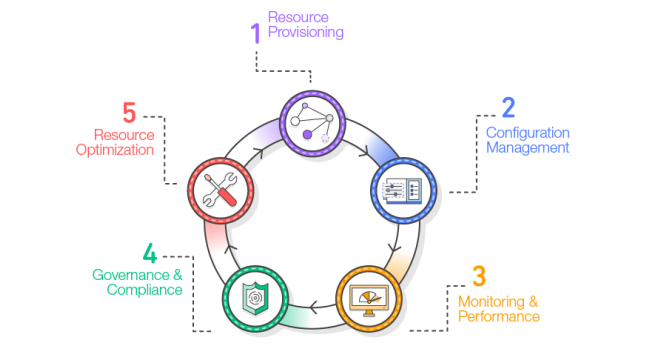
**One of Terraform’s key features for enterprise customers is its integration with [Hashicorp Sentinel](https://www.hashicorp.com/sentinel" \t "_blank). Sentinel is a**[**Policy-as-Code**](https://docs.hashicorp.com/sentinel/concepts/policy-as-code)**(PaC) framework that makes it possible to define organizational policies as code and enforce them as part of a Terraform provisioning workflow.**

**One of Terraform’s key features for enterprise customers is its integration with [Hashicorp Sentinel](https://www.hashicorp.com/sentinel" \t "_blank). Sentinel is a**[**Policy-as-Code**](https://docs.hashicorp.com/sentinel/concepts/policy-as-code)**(PaC) framework that makes it possible to define organizational policies as code and enforce them as part of a Terraform provisioning workflow.**

****

**Figure 1: Typical Infrastructure Resource Lifecycle from**[**AWS Whitepaper**](https://d1.awsstatic.com/whitepapers/DevOps/infrastructure-as-code.pdf)

**Examining the infrastructure resource lifecycle shown in Figure 1, many popular policy enforcement tools ([DivvyCloud](https://divvycloud.com/" \t "_blank),**[**Cloud Custodian**](https://cloudcustodian.io/)**,**[**AWS Config**](https://aws.amazon.com/config/)**, etc.) are designed to operate at Step 4 (green). They are reactive in nature, identifying infrastructure policy violations in real-time and executing user-defined remediation steps upon detection. Taking an alternate approach, Hashicorp Sentinel offers a proactive solution, allowing organizations to shift-left on governance and ensure policies are adhered to as a prerequisite to infrastructure creation. Using Figure 1 to help draw comparisons, imagine Sentinel existing as Step 0 in the lifecycle.**

**Armed with an understanding of Sentinel’s purpose, let’s examine some of the key concepts and features of Sentinel:**

**Policy-As-Code**

* **Briefly discussed above,**[**Policy-As-Code**](https://docs.hashicorp.com/sentinel/concepts/policy-as-code)**(PaC) refers to the ability to define organizational policies in areas such as security, compliance, or operational best practices as code. These policies can then be evaluated as part of an automated infrastructure provisioning workflow**

**Policy Language**

* **Sentinel Policies are constructed using a**[**high-level, purpose-built programming language**](https://docs.hashicorp.com/sentinel/language/spec#sentinel-language-specification)**. Athe intent was for this language to be useable by those who may not come from a traditional development background. As a best practice, isolating policy writers (e.g. InfoSec) from those doing infrastructure deployments (e.g. Application Teams) is highly desirable. Sentinel policy repositories should exist separately from any infrastructure configurations.**

**Imports**

* **Like any other programming language, the Sentinel language includes the ability to employ reusable libraries through the use of imports. I’ve found several of the standard imports to be extremely useful when crafting policies, for cases such as string manipulation (**[**strings import**](https://docs.hashicorp.com/sentinel/imports/strings)**) or contacting external API endpoints (**[**http import**](https://docs.hashicorp.com/sentinel/imports/http)**) to retrieve information from internal corporate systems. In addition to the standard imports, Sentinel can be extended through the creation of custom modules. However, I feel that this is an area that needs investment from Hashicorp to make it more accessible for end-users looking to utilize highly custom imports.**

**Enforcement Level**

**Sentinel’s**[**enforcement level concept**](https://docs.hashicorp.com/sentinel/concepts/enforcement-levels)**replaces the typical binary pass/fail nature of policy evaluation with 3 classification levels:**

* **‘Hard-Mandatory’ — policies must pass all of the time in order for infrastructure creation to proceed**
* **‘Soft-Mandatory’ — if a policy fails, the result can be overridden by those with correct permissions in order for infrastructure creation to proceed**
* **‘Advisory’ — if a policy fails, a warning message is printed/logged but it will not block infrastructure creation**

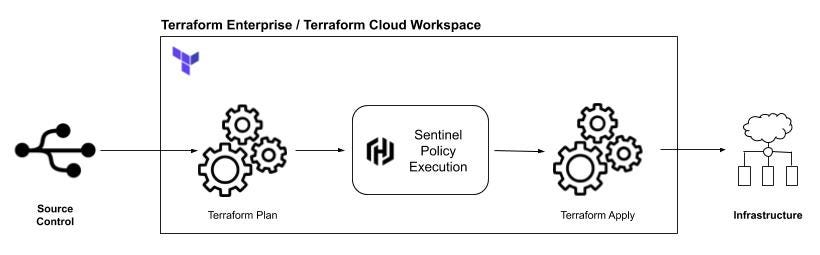
**Enforcement levels are implemented by detaching a policy's logic from its assigned enforcement level. The assignment of enforcement levels to policies is stored in a [sentinel.hcl](https://www.terraform.io/docs/cloud/sentinel/manage-policies.html" \l "the-sentinel-hcl-configuration-file" \t "_blank)**[**configuration file**](https://docs.hashicorp.com/sentinel/configuration)**, which lives in the same repository that hosts the policies.**

**Sentinel Policy Validation Workflow**

**While I’ll focus on the integration of Sentinel with Terraform in the remainder of this post, it’s useful to know that it’s embedded within all of Hashicorp’s enterprise products:**

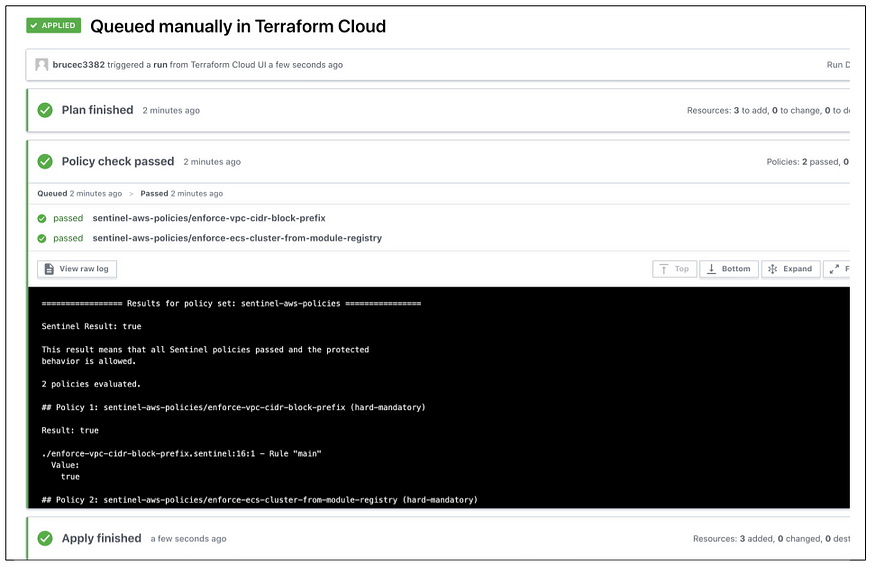
* **Terraform Enterprise/Cloud**
* [**Vault**](https://www.hashicorp.com/products/vault)**Enterprise**
* [**Consul**](https://www.consul.io/)**Enterprise**
* [**Nomad**](https://www.nomadproject.io/)**Enterprise**

**As part of an infrastructure provisioning workflow in Terraform, Sentinel executes all configured policies between the Plan and Apply stages, as illustrated in Figure 2:**

****

**Figure 2: How Sentinel Fits into the Terraform Infrastructure Provisioning Workflow**

**Figure 3 illustrates how the workflow depicted in Figure 2 is represented within the Terraform UI:**

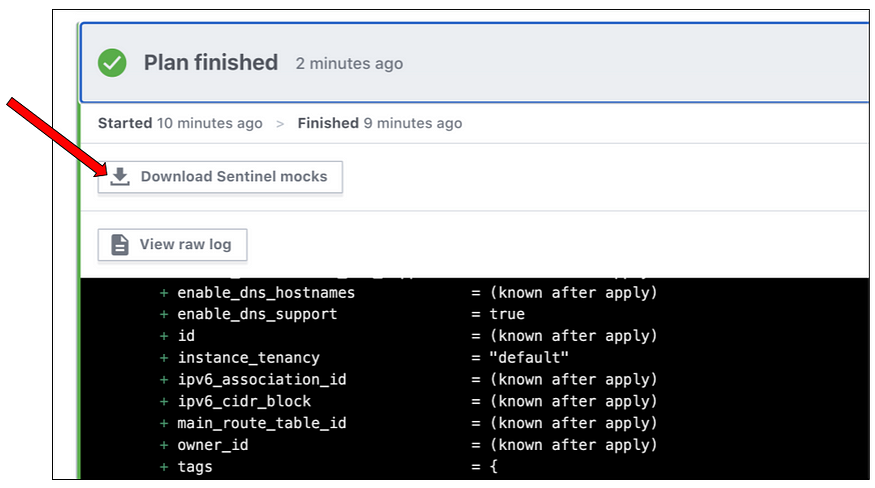
****

**Figure 3: Terraform Plan, Sentinel Policy Check, and Terraform Apply**

**Sentinel Policy Development and Testing Workflow**

**An exhaustive discussion of Sentinel Policy development is beyond the scope of this post. However, detailed instructions on how to get started can be found in the [Hashicorp Sentinel documentation](https://docs.hashicorp.com/sentinel/writing" \t "_blank).**

**Writing Sentinel Policies that validate the compliance of cloud infrastructure requires the use of realistic mock test data. Terraform provides the ability to generate mock data from existing Terraform configurations executed in a Terraform workspace. This is easily achieved from the Terraform UI by using the ‘Download Sentinel mocks’ button as shown in Figure 4:**

****

**Figure 4: Downloading Mock Files following Terraform Plan**

**Clicking the button downloads a compressed TAR file to an end user’s local machine. This TAR file contains the following 4 file types:**

* **mock-tfplan-v2.sentinel = Terraform Plan document generated from the workspace run**
* **mock-tfstate-v2.sentinel = Terraform State at the time of the Terraform Plan**
* **mock-tfconfig-v2.sentinel = Terraform Config at the time of the Terraform Plan**
* **mock-tfrun.sentinel = Terraform Run document**

**During the development of a Sentinel policy, the above “mock\*.sentinel” files can serve as mock inputs to help validate expected policy behavior. Each policy should be thoroughly tested with mock data containing pass and fail scenarios before being deployed to Terraform via a**[**policy set**](https://www.terraform.io/docs/cloud/sentinel/manage-policies.html#managing-policy-sets)**. The**[**Sentinel CLI tool**](https://docs.hashicorp.com/sentinel/intro/getting-started/install)**contains a ‘test’ command that will find and execute all test cases for a given policy, provided the correct directory structure exists at test/{sentinel-policy-name} within the repository.**

**Figure 5 illustrates an example of the folder structure for a policy that validates the prefix of a VPC CIDR block (discussed more in Example 1 in the next section). The directory at test/enforce-vpc-cidr-block-prefix is required in order for tests to be discovered and executed as intended for the enforce-vpc-cidr-block-prefix.sentinel policy.**

****

**Figure 5: Example Sentinel Policy Directory Structure**

**Specific test case scenarios are defined using**[**JSON or HCL configuration files**](https://docs.hashicorp.com/sentinel/configuration)**(Label #4 in Figure 5). Each HCL/JSON file in the test directory references mock files to use within that specific test case scenario, as shown in the contents of pass.json (Figure 6):**

****

**Figure 6: Contents of the pass.json file from Example 1**

**The common modules referenced using label #1 in Figure 6 are a collection of reusable functions that perform common actions in Sentinel policies, such as searching Terraform plans for specific resource types or validating AWS resource attributes values. The functions referenced in the examples were written by Roger Berlind from Hashicorp and sourced from his excellent**[**terraform-guides**](https://github.com/hashicorp/terraform-guides)**repository. I highly recommend taking a look through the**[**third-generation governance directory**](https://github.com/hashicorp/terraform-guides/tree/master/governance/third-generation)**for an example of Sentinel policy repository structure and many useful Sentinel Policy examples.**

**Sentinel Policy Examples**

**Having discussed a little bit about how Sentinel works, within this section I’ll provide two simple scenarios and examples of Sentinel Policies that can be used to enforce them. The repository containing these simple examples can be found**[**here**](https://github.com/BruceCutler/sentinel-aws-policies)**— it may help to have the files open as you read through the policy explanations below.**

**Example 1: Validating that AWS VPC CIDR Block Addresses start with “10.”:**

**This first example, written to validate the prefix of a VPC CIDR block, highlights the benefits of utilizing reusable common functions within your policies.**

**After importing the common tfplan-functions module (Line 1), we can easily identify if there are any AWS VPC resources in our Terraform Plan document using the [find\_resources](https://github.com/hashicorp/terraform-guides/blob/master/governance/third-generation/common-functions/tfplan-functions/docs/find_resources.md" \t "_blank) module function (Line 4).**

**With that information in hand, we can then leverage the [filter\_attribute\_does\_not\_have\_prefix](https://github.com/hashicorp/terraform-guides/blob/master/governance/third-generation/common-functions/tfplan-functions/docs/filter_attribute_does_not_have_prefix.md" \t "_blank) function to ensure that all AWS VPC resources have a prefix that matches our required value of “10.”.**

**Example 2: AWS ECS Clusters must be created using the ECS Cluster Module from the Private Module Registry:**

**This second example demonstrates more functionality of the Sentinel language, using functions and loops to help validate that any AWS ECS Cluster resources were created using a specific Terraform Module hosted in my**[**Terraform Private Module Registry**](https://www.terraform.io/docs/cloud/registry/index.html)**.**

**One of the first things to notice is that this function has additional imports, adding [tfconfig-functions](https://github.com/hashicorp/terraform-guides/tree/master/governance/third-generation/common-functions/tfconfig-functions" \t "_blank), the tfconfig file, and the built-in**[**strings**](https://docs.hashicorp.com/sentinel/imports/strings)**import. These are required as this policy examines resources within the Terraform Plan and investigates the Terraform Config to identify specific module calls.**

**Lines 11–16 and Lines 22–42 demonstrate user-defined functions, conditional statements, and string manipulation using the**[**strings**](https://docs.hashicorp.com/sentinel/imports/strings)**import.**

**Beneath the function definitions, the policy logic executes between Lines 44–64. After searching the Terraform Plan for any AWS ECS Clusters (Line 45), we loop (Line 48) through any ECS Cluster resource results to determine whether they were:**

1. **Created using a module**
2. **Whether that module matches the expected module source from the Private Module Registry in my Terraform Organization**

**If any of the discovered cluster resources fail either of these conditions, they are appended to a list. The success/failure of the policy is based on the length of that list at the conclusion of the policy execution (Lines 66–68).**

**Final Thoughts**

**Organizations leveraging the cloud at scale require company policies tied to security, compliance, and operational best practices to be adhered to at all times. Hashicorp Sentinel makes it possible for organizations who have invested in commercial offerings of Terraform to shift-left on the enforcement of such policies within their infrastructure provisioning workflows.**

**As shown in the provided examples, the Sentinel Policy language is straightforward to read and designed for ease of use. Leveraging reusable functions from common modules can help to greatly simplify your policy code. These examples barely scratch the surface of Sentinel’s functionality but do offer an insight into how conditional logic and looping constructs can be used to build more detailed policies.**

**In my opinion, the use of Sentinel doesn’t obviate the need for real-time policy enforcement strategies using tools such as DivvyCloud or Cloud Custodian. Instead, Sentinel and traditional governance enforcement tools should be used together. Doing so will ensure that organizational policies are enforced while providing fast-feedback and fostering the level of innovation that we are fortunate to enjoy in the public cloud today.**