HashiCorp Terraform Cloud for Business

Test plan for provisioning with Terraform Cloud for Business

Proof of Value
HealthFirst

Contents

Overview

HashiCorp Terraform is the de-facto standard for hybrid provisioning of infrastructure and applications. The benefits of using Terraform Cloud for Business will be:

1) Increase throughput in provisioning additional/new, updating and deleting instances and save IT team time and money replicating (dev, qa, prod) environments.
2) Deploy faster, safer and more consistently to keep up with existing and future workloads. Perform cost estimation before initial deployment and before changes.
3) Ability to manage multiple software environments (dev, qa, prod) as well as collaborate on infrastructure in those environments in order to increase quality and prevent mistakes.
4) Developers and infrastructure groups are able to provision infrastructure and collaborate using a common language. Infrastructure deployments are consistent, safe, and reliably deployed through policy guardrails.

The goal of this Proof of Value is to demonstrate the value of HashiCorp Terraform Cloud for Business features implementing some critical use cases.

This test plan for Terraform Cloud supports the following objectives:

- Functional Capabilities
    - Infrastructure as code
    - Cloud management
- Operational Aspects
    - Defining infrastructure as code
    - Linking Terraform Cloud to a version control system
    - Single Sign On
    - Managing infrastructure on cloud platforms
    - Utilizing Role Based Access Control
    - Creating and consuming Terraform modules in Terraform Cloud
    - Creating and consuming Terraform “no-code provisioning” modules in Terraform Cloud
    - Governance and compliance through Sentinel Policies

Test Plan

Test Items

- Terraform Cloud for Business - TFC4B

Features To Be Tested

- Creation and Management of infrastructure template files that can be integrated with version control software providers such as Github.
- Runs initiated from Terraform Cloud's UI, Github Actions, and the Terraform Cloud Github VCS integration.
- The management of users via SSO, and teams and access to Workspaces through RBAC.
- The production and consumption of Terraform modules that are stored in the Terraform Cloud private module registry, as well as "no-code provisioning" modules.
- The configuration and application of sentinel policies demonstrating what can be accomplished with the compliance features of Terraform Cloud.

Approach

Testing will be performed by one or all members of the client team. Members may opt out at the customer's discretion, but each use case will be completed by at least one member of the customer team.

Preparation

This POV requires a number of preparations to be successfully run. Here are the requirements for a successful POV:

- Subscription accounts and access to the cloud providers to be tested.
    - AWS: a subscription with an access id and a secret key
- Administrative access to Github or the VCS in use at the customer
- Creation of a Terraform Cloud account and organization
- Admin access to SSO provider
- Web Console access to these cloud provider environments to verify the results of provisioning.
- CLI access to these cloud provider environments in the event there is a need to verify the results from the command line.
- Network connectivity and access to the cloud environments
- A good text editor to view and insert your keys and information as needed

Pass/Fail Criteria

All functionality should function as expected and outlined in the individual test cases.

Suspension Criteria

Testing should be postponed if connectivity or any other issues are present.

Test Deliverables

The following documents will be created and updated as a result of these testing activities:

- Test Plan (this document)
- Test Environment (Appendix A)
- Test Case Specifications (Appendix B)
- Test logs
- Test summary report

Testing Tasks

The following activities must be completed:

- Prepare Test Plan (this document)
- Prepare Test Case Specifications (Appendix A)
- Prepare test environment
- Perform the tests and capture test logs
- Prepare test summary report

Environment Needs

The following environment must be created:

- Each participant needs access to a terminal client in which they can run a shell.
- All the prerequisites should be available for all test cases before we start the use cases.
- A git-based version control system (GitHub, BitBucket, Gitlab) will be required for testing.

Schedule

Once the test environment is ready, test execution is expected to take 2-10 days. The timeline will be highly dependent on use cases and team proficiency with tools, subject to the prerequisites being met prior to the start of this Proof-of-Value (PoV) engagement. Any dates and duration of specific tasks are based on estimates only. Refer to the 'HashiCorp Proof of Value Scope' document for specific agreed upon timelines.

Risks And Contingencies

The following scenarios are potential risks and contingencies:

- Inability to access Terraform Cloud for Business due to network or organizational constraints or problems.
- Inability to provide any environment prerequisites listed in the 'Environment Needs' section.
- Inability to engage with a private or public cloud platform due to network constraints or organizational policies.

Approvals

The customer and vendor must agree to the Test Plan (this document) and Test Case Specifications (Appendix A).

Hashicorp	HealthFirst
Company	*Company*
Sam Terrell	Ahmad Bacchus
Name	*Name*
Signature	*Signature*
Regional Vice President - NY Metro	
Title	*Title*
Date	*Date*

Appendix A

Test Case Environment

Terraform Cloud

- A Terraform Cloud account and organization
- Each participant will have an account created for them by the lead customer resource within the Terraform Cloud organization

Version Control System

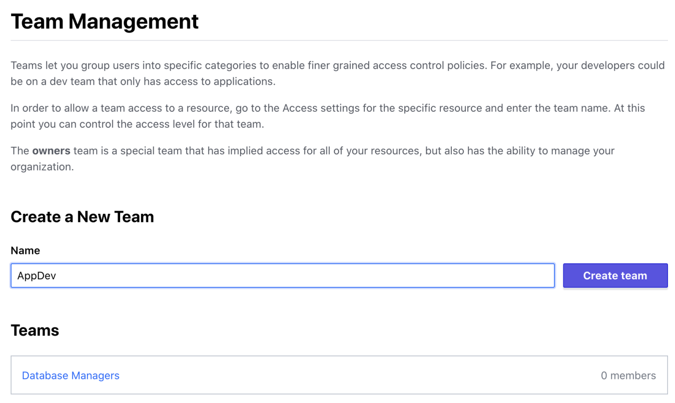
- Proper VCS access and integration level permissions for the VCS (Github.com)

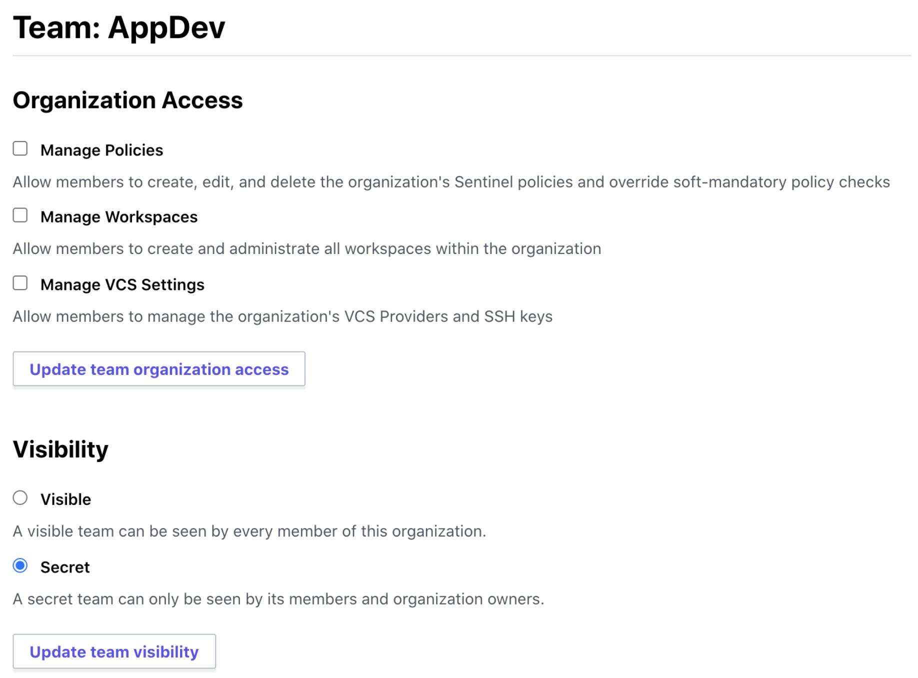
Amazon Web Services

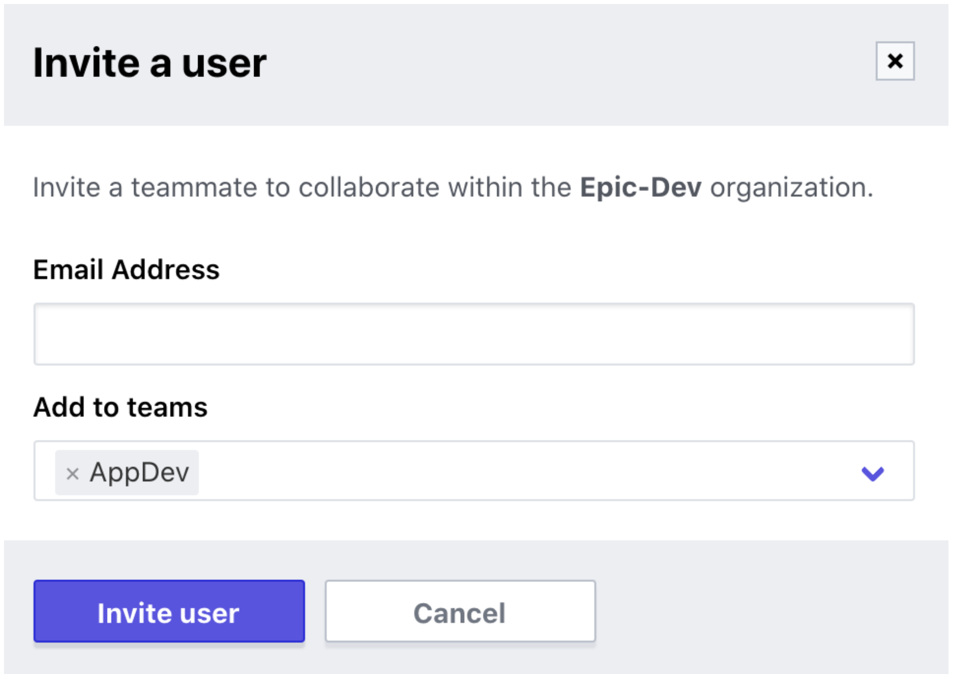
- Credentials to deploy / destroy the planned resources for each success criteria
- Optional: TFC Agents to execute Terraform applies from within an AWS VPC.

Appendix B

Use Case Specifications

Test Case	**TF-UC-1: Demonstrate Team management and access to workspaces through Organizations and Access control**
Description	**Create Teams and Invite Users** As an Administrator, I'd like to set up an initial Organization after the system setup has been completed. I want to be able to create a number of teams to manage infrastructure in workspaces and invite users into those teams
Setup	● Prerequisite - Be sure to complete this section of the SSO instructions
Command/Input	1. Under the Org settings menu, go to Teams and create a new Team called "AppDev"  **Team Management** Teams let you group users into specific categories to enable finer grained access control policies. For example, your developers could be on a dev team that only has access to applications. In order to allow a team access to a resource, go to the Access settings for the specific resource and enter the team name. At this point you can control the access level for that team. The **owners** team is a special team that has implied access for all of your resources, but also has the ability to manage your organization. **Create a New Team** Name AppDev Create team **Teams** Database Managers 0 members 2. Set permissions to allow the team to manage the workspace and VCS repositories

	 **Team: AppDev** **Organization Access** ☐ Manage Policies — Allow members to create, edit, and delete the organization's Sentinel policies and override soft-mandatory policy checks ☐ Manage Workspaces — Allow members to create and administrate all workspaces within the organization ☐ Manage VCS Settings — Allow members to manage the organization's VCS Providers and SSH keys **Visibility** ○ Visible — A visible team can be seen by every member of this organization. ● Secret — A secret team can only be seen by its members and organization owners. 3. Repeat Steps 2 and 3 create a Operators Team with only Policy Management access 4. Invite HealthFirst users into the teams from the Organization's Settings-> Users menu. Add an email address and the user should receive an invite to join. They will have an opportunity to create their user accounts and access workspaces they are aligned to.

	 **Invite a user** Invite a teammate to collaborate within the **Epic-Dev** organization. **Email Address** **Add to teams** × AppDev
Expected Results	● The organization, teams and users should be set up with 2 new teams: AppDev and Operators. ● Users are invited and part of the teams
Pass (Y/N)	
Actual Results	

Test Case	**TF-UC-2: Configure VCS Integration of Terraform Cloud for Business with Github Enterprise**
Description	As an Admin, I would like to configure VCS Integration of Terraform Cloud Business with Github Enterprise
Setup	● Ensure accessibility of Github Enterprise from Terraform Cloud IP ranges ● Ensure you have created an organization in TFCB for use with the VCS integration

Command/Input	https://developer.hashicorp.com/terraform/cloud-docs/vcs/github-enterprise
Expected Results	You have set up a VCS integration of Github Enterprise in TFCB for any workspace you'd like to link to any repository. All future integrations to that repository can now automatically trigger a new plan if changes are committed to that repository path.
Pass (Y/N)	
Actual Results	

Test Case	**TF-UC-3: Initiate a Terraform Run through GitHub Actions (CI/CD Pipeline)**
Description	As an administrator or Terraform user, you should be able to initiate a plan(and apply) from a Github Actions CI/CD triggering event
Setup	● A GitHub account ● A Terraform Cloud account ● An AWS account and AWS Access Credentials
Command/Input	https://developer.hashicorp.com/terraform/tutorials/automation/github-actions
Expected Results	You have successfully initiated a Terraform apply from a Github Actions CI/CD pipeline step and the desired resources have been successfully provisioned
Pass (Y/N)	
Actual Results	

Test Case	**TF-UC-4: Initiate a Terraform Run from the Terraform Cloud interface**

Description	As an administrator or Terraform user, you should be able to initiate a plan (and apply) from the Terraform Cloud UI
Setup	• Ensure you have the proper permissions in TFCB • Ensure you've created a workspace that you have access to that is tied to a VCS repository
Command/Input	https://developer.hashicorp.com/terraform/cloud-docs/run/ui#manually-starting-runs
Expected Results	You have successfully initiated a Terraform apply from the Terraform Cloud User Interface and the desired resources have been successfully provisioned.
Pass (Y/N)	
Actual Results	

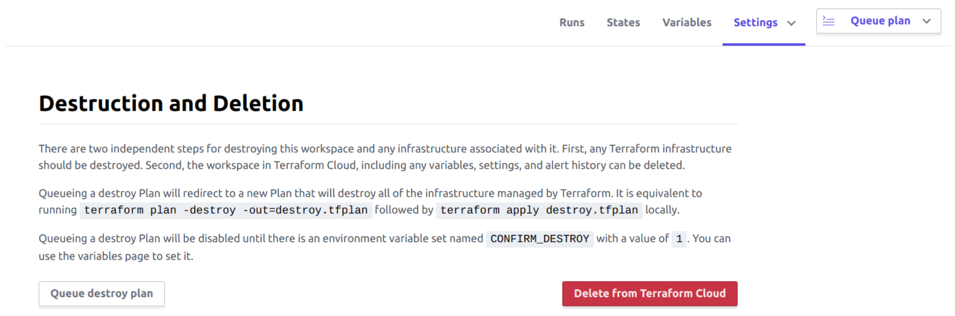
Test Case	**TF-UC-5: Build and configure a Terraform Module to be stored in the private module registry**
Description	As a producer of Terraform Code, you should be able to create a Terraform module and store said module in the private module registry.
Setup	• TFCB Connection to VCS provider • TFCB Admin level access for managing modules
Command/Input	There are multiple parts to this test case. Once you complete the initial creation of the module, you will move onto the publishing of the module into the private module registry 1. https://developer.hashicorp.com/terraform/tutorials/modules/module-create 2. https://developer.hashicorp.com/terraform/tutorials/modules/module-private-registry-share
Expected Results	A Terraform module is created and then published to VCS. That module is then added to the private module registry.

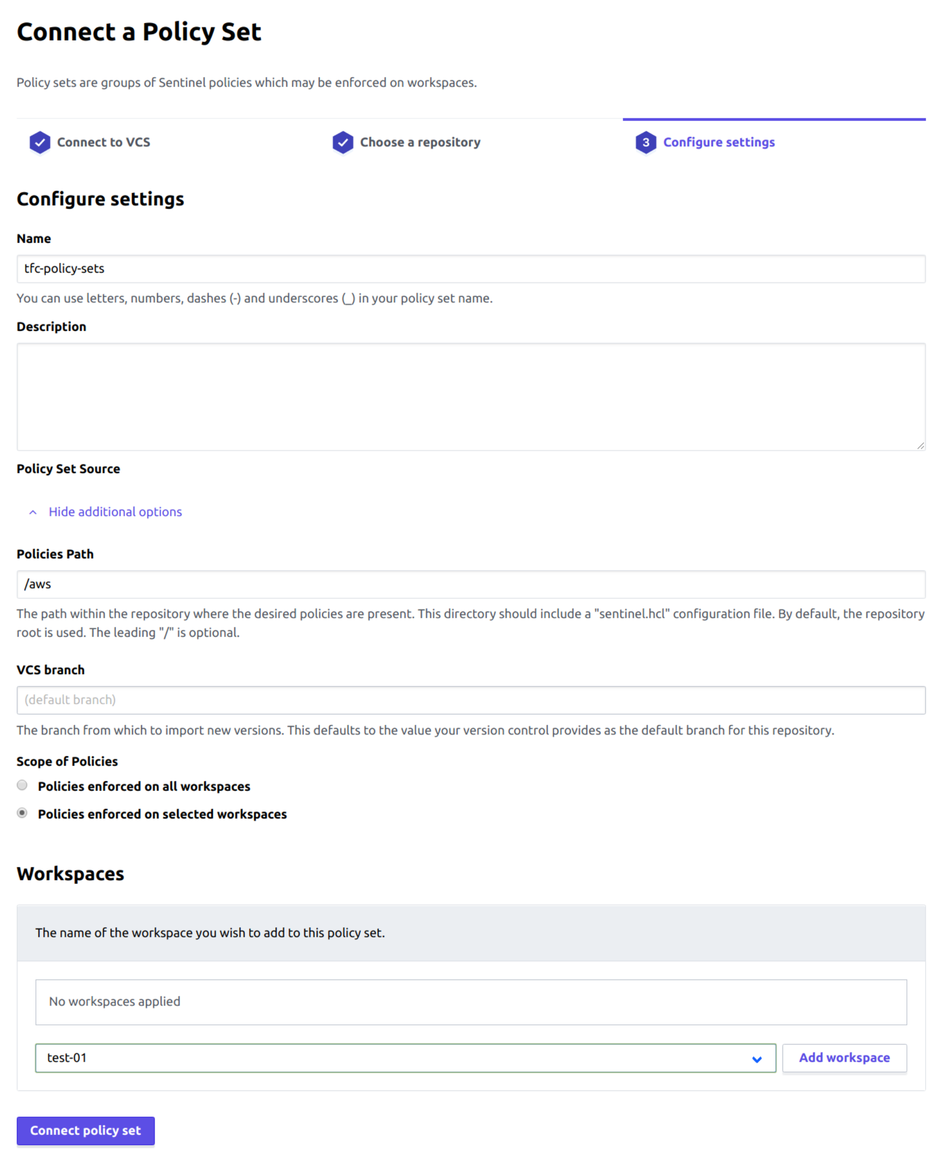
Pass (Y/N)	
Actual Results	

Test Case	**TF-UC-6: Consume created private module in a new Terraform Cloud for Business workspace**
Description	As a consumer of Terraform code, you should be able to utilize modules from the private module registry.
Setup	● Successful completion of TF-UC-5
Command/Input	https://developer.hashicorp.com/terraform/tutorials/modules/module-private-registry-share#create-a-configuration-that-uses-the-module
Expected Results	A new workspace with a root Terraform configuration is created, and the Terraform code consumes a module from the private module registry.
Pass (Y/N)	
Actual Results	

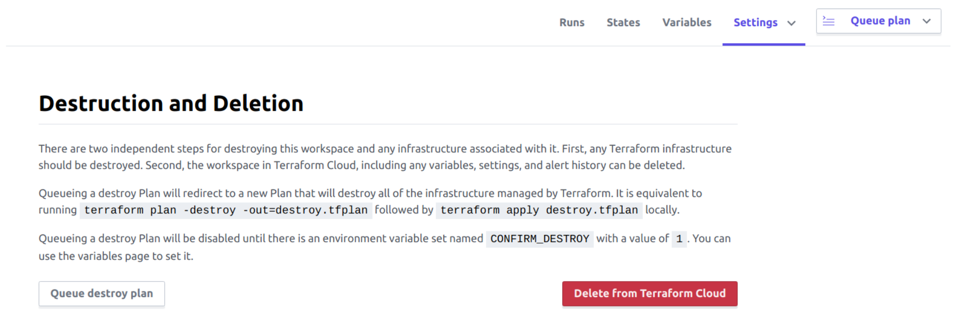
Test Case	**TF-UC-7: Build and configure a “no code provisioning” ready module to be stored in the private registry, then consume said “no code provisioning” module**
Description	As a producer of Terraform code you should be able to create a no code provisioning module and then use that no code provisioning module to create a new workspace containing Terraform resources from the aforementioned module.
Setup	● TFCB Admin account with ability to manipulate modules ● AWS Account ● TFCB Variable Set containing AWS credentials for deploying infrastructure

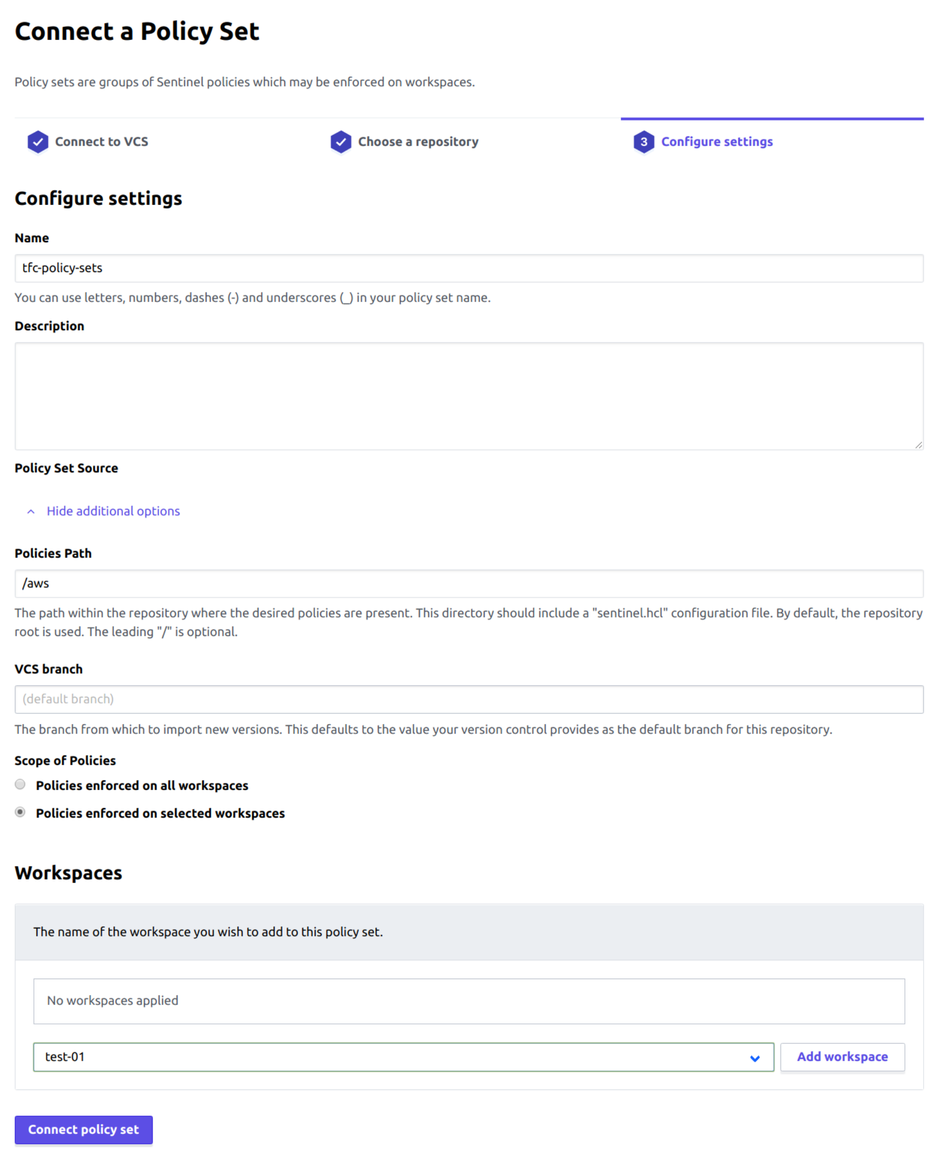
Command/Input	https://developer.hashicorp.com/terraform/tutorials/modules/no-code-provisioning
Expected Results	A no code provisioning module has been created, and then subsequently consumed to create a new workspace containing the Terraform resources from the no code provisioning module.
Pass (Y/N)	
Actual Results	

Test Case	**TF-UC-8: Configure a Sentinel policy to restrict instance size**
Description	**Enforce Policy with Sentinel to restrict expenditure above a particular dollar amount** As an engineer, I want to enforce corporate governance policies
Setup	● Have prepared a configuration for deploying infrastructure in the cloud, such as AWS EC2 Instance ● Queue Destroy Plan in the workspace created earlier.
Command/Input	1. Delete the resources from the prior test cases by clicking on “Queue destroy plan” in the workspace (if required).  **Destruction and Deletion** There are two independent steps for destroying this workspace and any infrastructure associated with it. First, any Terraform infrastructure should be destroyed. Second, the workspace in Terraform Cloud, including any variables, settings, and alert history can be deleted. Queueing a destroy Plan will redirect to a new Plan that will destroy all of the infrastructure managed by Terraform. It is equivalent to running `terraform plan -destroy -out=destroy.tfplan` followed by `terraform apply destroy.tfplan` locally. Queueing a destroy Plan will be disabled until there is an environment variable set named `CONFIRM_DESTROY` with a value of `1`. You can use the variables page to set it. 2. Fork the repo hashicorp/terraform-guides into your VCS if not done already. 3. In the TFE UI, (purple menu bar at top of screen), go to **Settings** -> **Policy Sets** and click on **Connect a Policy Set.**

	 **Connect a Policy Set** Policy sets are groups of Sentinel policies which may be enforced on workspaces. ✓ Connect to VCS ✓ Choose a repository 3 Configure settings **Configure settings** **Name** tfc-policy-sets You can use letters, numbers, dashes (-) and underscores (_) in your policy set name. **Description** **Policy Set Source** Hide additional options **Policies Path** /aws The path within the repository where the desired policies are present. This directory should include a "sentinel.hcl" configuration file. By default, the repository root is used. The leading "/" is optional. **VCS branch** (default branch) The branch from which to import new versions. This defaults to the value your version control provides as the default branch for this repository. **Scope of Policies** Policies enforced on all workspaces [✓ Policies enforced on selected workspaces] **Workspaces** The name of the workspace you wish to add to this policy set. No workspaces applied test-01 Add workspace Connect policy set 4. Choose the newly created repo from your VCS. Click the 'more options' link and set the Policies Path to '`/governance/third-generation/aws`'. Set the scope of that policy to 'All workspaces'. Click the 'Connect policy set' to complete. 5. Queue a Plan on the workspace to recreate the infra based on the module again.
Expected Results	Policy is enforced during a run and checks resources being applied to EC2 virtual machines

Pass (Y/N)	
Actual Results	

Test Case	**TF-UC-9: Configure a Sentinel policy to limit dollar spend per apply**
Description	**Enforce Policy with Sentinel to restrict expenditure above a particular dollar amount** As an engineer, I want to enforce corporate governance policies
Setup	● Have prepared a configuration for deploying infrastructure in the cloud, such as AWS EC2 Instance ● Queue Destroy Plan in the workspace created earlier.
Command/Input	1. Delete the resources from the prior test cases by clicking on "Queue destroy plan" in the workspace (if desired/required).  Runs States Variables Settings Queue plan Queue destroy plan Delete from Terraform Cloud **Destruction and Deletion** There are two independent steps for destroying this workspace and any infrastructure associated with it. First, any Terraform infrastructure should be destroyed. Second, the workspace in Terraform Cloud, including any variables, settings, and alert history can be deleted. Queueing a destroy Plan will redirect to a new Plan that will destroy all of the infrastructure managed by Terraform. It is equivalent to running `terraform plan -destroy -out=destroy.tfplan` followed by `terraform apply destroy.tfplan` locally. Queueing a destroy Plan will be disabled until there is an environment variable set named `CONFIRM_DESTROY` with a value of `1`. You can use the variables page to set it. 2. Fork the repo hashicorp/terraform-guides into your VCS. 3. In the TFCB UI, (purple menu bar at top of screen), go to **Settings** -> **Policy Sets** and click on **Connect a Policy Set.**

	 **Connect a Policy Set** Policy sets are groups of Sentinel policies which may be enforced on workspaces. Connect to VCS · Choose a repository · 3 Configure settings **Configure settings** **Name** tfc-policy-sets You can use letters, numbers, dashes (-) and underscores (_) in your policy set name. **Description** **Policy Set Source** Hide additional options **Policies Path** /aws The path within the repository where the desired policies are present. This directory should include a "sentinel.hcl" configuration file. By default, the repository root is used. The leading "/" is optional. **VCS branch** (default branch) The branch from which to import new versions. This defaults to the value your version control provides as the default branch for this repository. **Scope of Policies** Policies enforced on all workspaces [✓ Policies enforced on selected workspaces] **Workspaces** The name of the workspace you wish to add to this policy set. No workspaces applied test-01 · Add workspace Connect policy set 4. Choose the newly created repo from your VCS. Click the 'more options' link and set the Policies Path to '`/governance/third-generation/cloud-agnostic`'. Set the scope of that policy to 'All workspaces'. Click the 'Connect policy set' to complete. 5. Queue a Plan on the workspace to recreate the infra based on the module again.
Expected Results	Policy is enforced during a run and checks for costs being within a threshold with a message:

	`Proposed monthly cost xxx is under the limit: 1000`
Pass (Y/N)	
Actual Results	

Summary

You have successfully learned the basics of using the Terraform Cloud for Business application. These examples, although fairly simple, provide powerful models that make your infrastructure much more flexible, agile and knowable. Some of the things that this models provides to teams and enterprises are:

- The ability to create, update or destroy infrastructure quickly, automatically and within continuous delivery pipelines, if desired.
- The ability to provide a self-service infrastructure portal that's flexible and provides highly customized infrastructure.
- The ability to provide developers and others with infrastructure with little to no time investment from operators.
- The ability to provide infrastructure that adheres to even the strictest security and compliance requirements in an automated way, without creating a security review bottleneck.

Terraform and Sentinel are powerful tools that make even the largest enterprises able to adapt quickly. We hope this proof of value session was helpful and solidified your interest in moving forward regarding a commercial partnership with HashiCorp.