Terraform-ception: Creating Terraform Configurations with Terraform

Rob Jackson
Dec 3, 2020 · 7 min read

If you're reading this article, I'm making the assumption that you know something about Terraform. Not just that it's a great product by an incredible company, and not just that it could mean formation of the earth, but that it can be used to deploy infrastructure across multiple clouds, even private clouds, using a consistent workflow/methodology. When we contemplate this statement, infrastructure can have multiple meanings. When I first started using Terraform I primarily thought about infrastructure as just compute and connectivity. However, now I realize it is more…oh so much more.

So Much More Than Compute and Connectivity

In my time with HashiCorp, I've seen more people use providers outside of the primary/major cloud resources. Of course Terraform providers include other awesome HashiCorp products, but we're seeing people use Terraform for Cisco ACI, Heroku, Nutanix, F5, Palo Alto, Datadog, and more. But have you ever thought about using Terraform, for provisioning and managing Terraform itself? You might have to spin the top to determine which reality you're in, but if you think about it, we use computers to design and build computers, so this isn't much different. Until of course the machines rise up and battle the dolphins to determine which will triumph over the earth. I mean seriously, if that were to happen, this is the year.

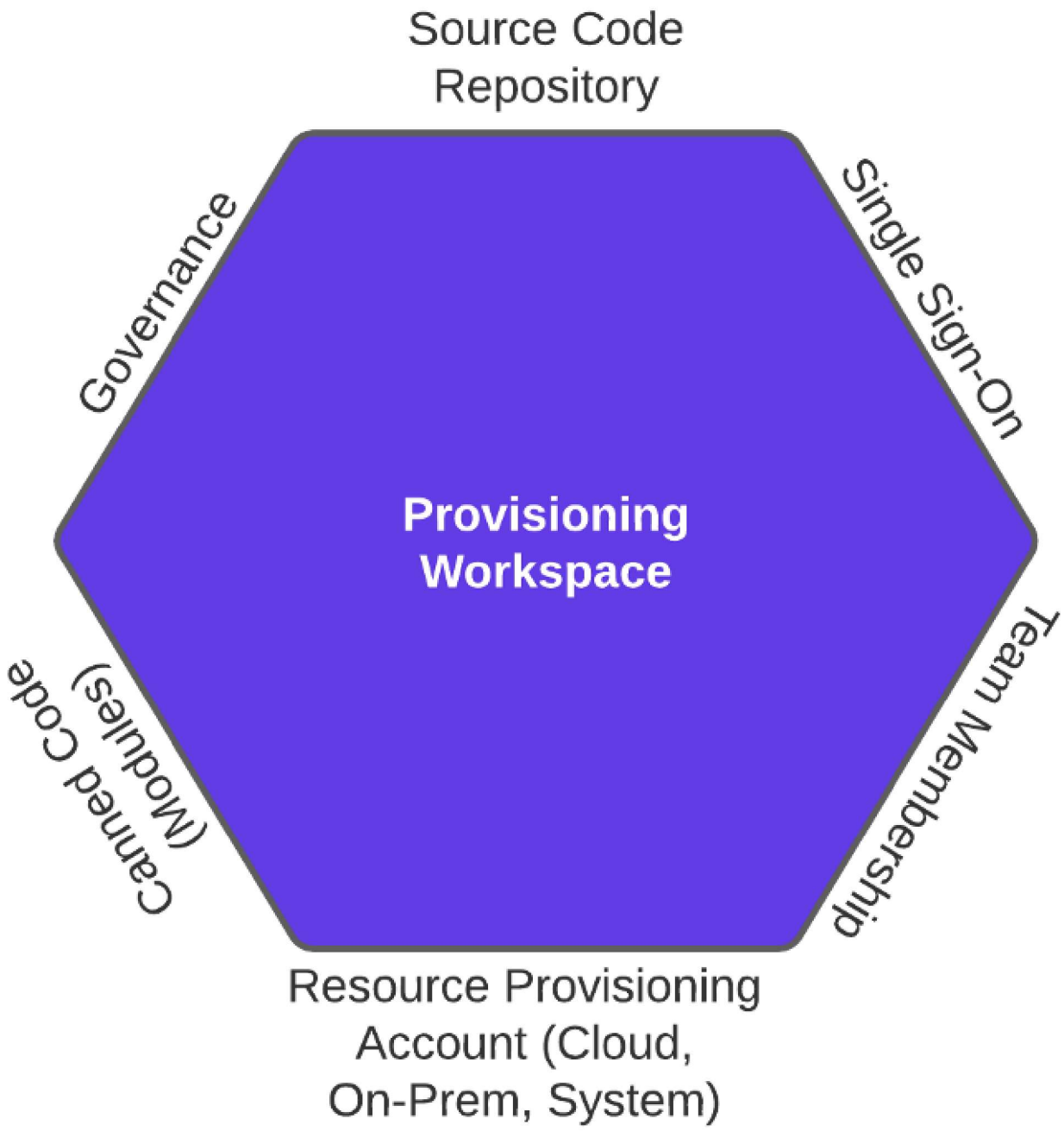
Is this a Dream, or Reality?

So Rob, what do you mean “use Terraform for Terraform?” Well, this is the path one of my customers has traveled, and since then I’m seeing more people interested in the concept. Terraform provides an Infrastructure as Code platform for teams of developers (and pseudo-developers) to create infrastructure of all kinds. Setting up a workspace (an environment) within Terraform isn’t difficult, but it is a process that involves humans, and therefore, is open to mistakes. So, let’s look at how we would automate that process, much like we automate the process of spinning up and spinning down resources for various teams and applications. First we’re going to look at the requirements (as every project should start with understanding the requirements), then look at the variables in our example, and finally, the process flow for the automation.

Requirements for Terraforming Terraform

What does it mean to spin up a new team on Terraform? Well, we need a repository of course, somewhere to store our code. We may or may not need a new organization, but we will need a provisioning workspace that is tied to that repository. We probably want users tied into some single sign-on process, and perhaps a cloud account for the infrastructure. Perhaps we need to create a new Team within Terraform, and create a new user for that team. Canned code, or modules, can help accelerate adoption and compliance, so we’ll want that as well. Don’t forget governance…whatever group we are creating, we’ll need to have some sort of protection setup to ensure they aren’t opening their infrastructure to the world (unless intended of course), and they aren’t

setting up some massive 96 core machine for a small development application. Funny how the six main ingredients can resemble that of a hexagon, as if the most industrious of creatures were constructing the first cell of a giant honeycomb!



The Six Main Ingredients to Terraform Terraform

Let's walk through that process, starting with the variables necessary to create the environment. The variables have been classified into two sets. One set of variables should be defined by those offering the Terraform development environment, the system administrators. I'm calling these 'Terraform System Variables'. The other set of

variables, read from a simple JSON file, are specific to the human, cyborg, or group requesting the environment. I'm calling these 'Consumer Variables.'

Terraform System Variables

These variables are necessary for communicating with Terraform. Of course there is the Terraform hostname (app.terraform.io if using Terraform Cloud), and the corresponding user token. Be sure that the user token is either an environment variable in your terminal, a sensitive (hidden) variable within the Terraform workspace, or in a file tracked within your `.gitignore` setting. Ideally the token can be pulled from HashiCorp Vault.

```
# Terraform System Variables
variable "hostname" {
   type = string
   description = "Hostname of the TFE/C Server"
}

variable "token" {
   type = string
   description = "Privileged User Token for TFE/C Server"
}

# Organization Variables
variable "new_organization" {
   type = string
   description = "Name for new organization to be created"
}

variable "new_org_email" {
   type = string
   description = "Email address for Organization admin"
}

# Workspace Variables
variable "workspace" {
   type = string
   description = "Workspace to be added to Organization"
}
```

Consumer Variables

The following variables are collected from the person or organization requesting the Terraform environment. These can be collected from an internal ticketing system, or even some simple front end website. I've organized these variables into three distinct categories; Organizational Variables, Workspace Variables, and GitHub Variables. Note

that several additional variables are available within the Terraform Enterprise provider, but the focus of this paper is on the base functionality.

For the purposes of this demonstration, the Organizational and Workspace variables are simply names for these resources. The reader is strongly encouraged to review the documentation to discover the plethora of additional variables available.

```
# Organization Variables
variable "new_organization" {
   type = string
   description = "Name for new organization to be created"
}

variable "new_org_email" {
   type = string
   description = "Email address for Organization admin"
}

# Workspace Variables
variable "new_workspace" {
   type = string
   description = "Workspace to be added to Organization"
}
```

The GitHub related variables are more specific, and are used to create the repository within the appropriate GitHub organization. The GitHub token, in particular, should be closely guarded, and really shouldn't be entered into a file as I've done here. The preferred flow would be to have the GitHub Personal Access Token for each user stored within a secure Vault, and have Terraform use Vault's Key/Value store as a 'generic secret' data source.

```
# GitHub Variables
variable "github_token" {
   type = string
   description = "GitHub token with authorization levels to create
repository"
}

variable "github_organization" {
   type = string
   description = "GitHub organization in which repository lies"
}
```

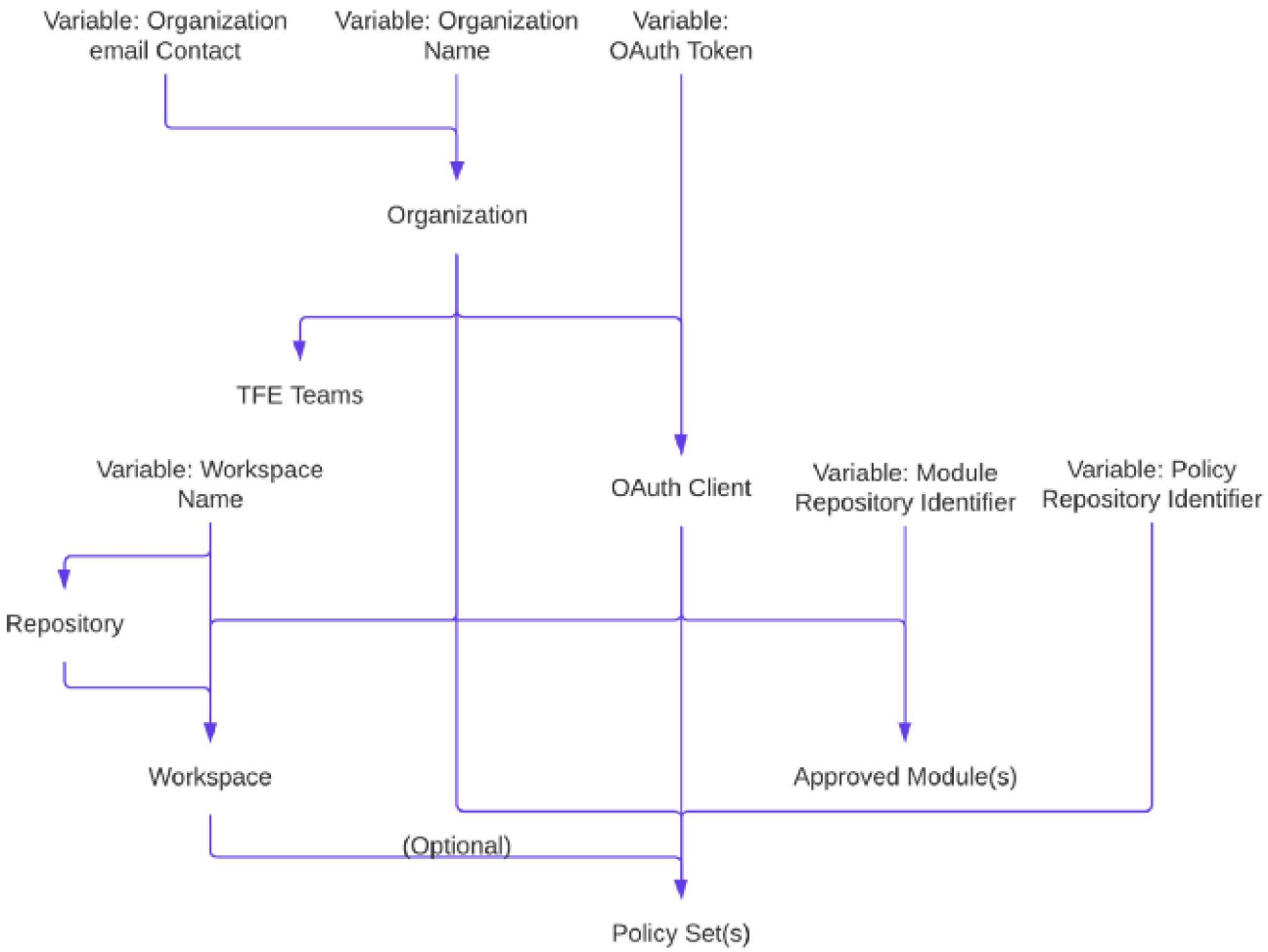
Lastly, We are creating variable maps for the repositories for both the Modules, as well as the Policy Sets, to be available to the Organization being created.

```
# Repository maps for modules and policies
variable "modules" {
   type = map
   description = "Map of name and identifier, for the VCS repo for
modules"
}

variable "policy_sets" {
   type = map
   description = "Map of name and identifier, for VCS repo for
policy sets"
}
```

Terraform Flow

Now that we've defined the variables that are necessary, let's look at how Terraform iterates through the creation of these resources. There are several dependencies that Terraform is able to address as part of the processing, identified in the following diagram.



Terraform Resource Creation

The diagram looks convoluted at first, so we'll walk through it.

1. Everything hinges on the creation of the Organization, which only requires a couple variables.
2. Once the Organization is created, we can create the Team and the OAuth Client associated with that Organization. This OAuth Client enables the connection to the Version Control System (VCS), providing several important references to the structure.
3. Using the OAuth Client Token ID, Terraform is able to pull repositories from the VCS to build the Private Module Registry for the Organization.
4. Once the GitHub repository is created (utilizing the Workspace name as a variable in the Repository name), the Workspace is now created. The Workspace name, Organization name, and the newly created Repository all are parts of the new Workspace creation.
5. Finally, the Organization and Workspaces feed into the Policy Set creation (which again, requires the OAuth Client Token ID).

The code for this creation is available in a public GitHub repository, but please note that this code has been written by a hack and is not guaranteed, under warranty, supported, etc., by HashiCorp.

Are you ready to Terraform Terraform with Terraform?

In this relatively simple set of Terraform code, we have been able to create an entire Terraform development environment, utilizing Terraform, a couple secret variables, and a handful of System and Consumer variables. I've "listified" a few parameters, but certainly more can be done. In fact, many of these variables can be fed as a map variable, with a request made via a GitHub Pull Request. Once the pull request has been approved, Terraform will create the new development environment with the new parameters. Alternatively, a simple front-end website could be created to collect the variables necessary for the environment creation. Regardless of how the few variables are delivered, this process can help apply your Infrastructure as Code methodology, to your Infrastructure as Code tools. So let me ask you, how many Terraforms can you Terraform using Terraform?

Terraform Infrastructure As Code Hashicorp Terraform Infrastructure Automation DevOps