What does Hashicorp do?

Infrastructure for managing your infrastructure

Justin
Aug 24, 2021

The TL;DR

Hashicorp sells open-core software that **helps developers manage their cloud infrastructure** via code-based configuration and access.

- Cloud infrastructure is **constantly evolving** – larger organizations need to create and modify their resources on the reg
- As organizations move towards multi-cloud and infrastructure breadth increases, **managing this stuff is getting more annoying**
- Hashicorp provides a **suite of open-core tools** like Terraform and Vault to make this process easier for developers
- The company makes money by **selling packaged solutions to large enterprises** (and a little via cloud)

Hashicorp is a beloved and respected (importantly) brand among developers. And with a $5B valuation as of March 2020, they are probably going to make somebody rich.

Why infrastructure needs managing

To understand Hashicorp, we'll first need to get a view of what it's like to manage cloud infrastructure, especially for larger companies. Let's put on our x-ray goggles and venture into the desk clump by the wall that gets no light where they put the software engineers.

If you're running your app(s) on cloud infrastructure today (and this also applies somewhat to home-grown stuff), your team runs up against **4 major problems**. Well, more than that, but 4 for the purposes of this post.

1. **Product overload**

AWS has literally hundreds of products, and it's not at all uncommon for companies to use 10 or even 20 of them at once. To quote from the original Technically post on AWS:

> *Even medium sized startups will often be using 10+ AWS services from the get go, and more established businesses can easily go past 100. Let's imagine we're a startup that sells technical literacy and education software to tech businesses (let's imagine). We've got a basic web application, and a little data warehouse for our Growth Lead to report basic company metrics. We might be using:*
>
> 1. ***EC2*** *to deploy our web app in a few Docker containers*
> 2. ***Lambda*** *to process form submissions on the marketing site*
> 3. ***EBS*** *for block storage connected to our EC2 instance(s)*
> 4. ***S3*** *to store backups and files for the app and marketing site*
> 5. ***Route53*** *to connect our domain name to our AWS servers*
> 6. ***RDS*** *(Postgres) as our managed database for our web app*
> 7. ***Cloudfront*** *as our CDN for serving assets quickly*
> 8. ***VPC*** *to isolate our resources into a private, secure network*
> 9. ***Backup*** *to back up our data across services*
> 10. ***Redshift*** *to store analytics data as our data warehouse*
>
> *These are just the AWS products that you'll be using - but there are other parts of the ecosystem that help support this product usage; sort of like the glue that keeps things together.*

As you might imagine, working with all of these products can get a bit hairy. It's hard to understand what's running, what state it's in, and what you should use for spinning up new apps or services.

2. **Config hell**

Each infrastructure product has different configurations. What region should your EC2 instance be in? What's the timeout for your Lambda function? What size is your EBS instance?

These configurations are essential to an efficient infrastructure setup, but creating and maintaining them is understandably complex. It doesn't help that the AWS console is notoriously difficult to work with.

3. **Permissions**

When you're a small shop with a few developers, it's OK for everyone to have master access to your infrastructure. But as the team grows, it can get dangerous to be loose with access controls – you'll want to start giving different developer groups different permission levels. Maybe your platform team has access to adjusting existing configurations, while your application team can only create *new* instances.

AWS – and other cloud providers – offer internal IAM (Identity & Access Management) utilities, but they, too, aren't particularly smooth or easy to use.

4. **Multi-cloud**

The overwhelming majority of cloud-native enterprises are using multiple cloud providers. Now what multi-cloud actually means in practice is beyond the scope of this piece (how diversified do you need to be?), but suffice it to say that large enterprises are using products from multiple cloud providers. That means multiple UIs and CLIs to manage this stuff, multiple billing portals, etc. This is not fun.

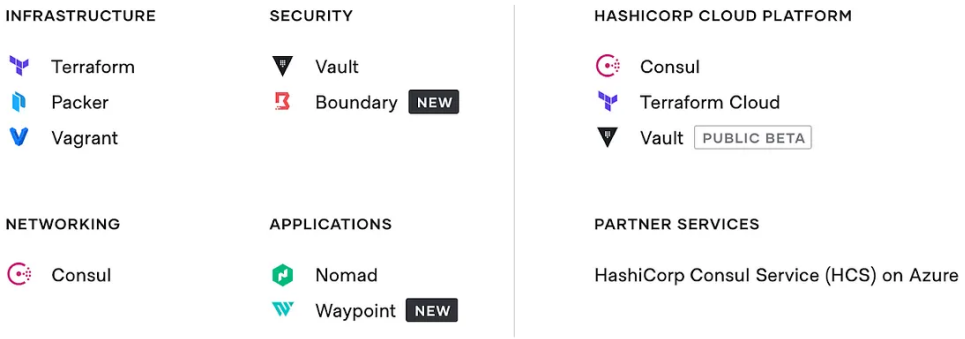
Hashicorp is a layer on top of your infrastructure

This is where Hashicorp comes in. You can think of them as a layer on top of your infrastructure, cloud or otherwise (like I literally just said in the headline), agnostic to who you're actually using – Amazon, Google, both of them, on-prem, whatever. And they help developers create and manage that infrastructure in a simple, scale-friendly way.

> 🚨 **Confusion Alert** 🚨
>
> "Infrastructure" can mean a lot of different things – a lot of people would call Hashicorp itself part of infrastructure. For the purposes of this post, think of

> it as anything you'd run an application on: a virtual machine, a Docker container, or any combination of those sorts of things.
>
> **🚨 Confusion Alert 🚨**

We'll start by running (jogging?) through some high level descriptions of Hashicorp's core products, and then dive deeper into their most popular one, called Terraform. And unlike AWS, the good people over at Hashicorp seem to be actually able to name their products nicely, which we love.



> **😰 Don't sweat the details 😰**
>
> Feel free to skip through this section if it's too in detail – there's a lot here. Just keep in mind that Hashicorp's goal is to be the one-stop-shop for filling in the gaps in your infrastructure – anything that AWS and co. don't do, plus making them all work together.
>
> **😰 Don't sweat the details 😰**

→ Infrastructure

These products help you manage your infrastructure primitives.

- **Terraform** – create, manage, and destroy infrastructure from any provider with an easy programmatic interface.
- **Packer** – service for building images that you can place on virtual machines.

- **Vagrant** – sort of a Docker alternative. Hashicorp's first product, back in 2010.

→ **Security**

These products help you manage your credentials and access controls.

- **Vault** – create and manage access to secrets (API Keys, access tokens, etc).
- **Boundary** – system for managing remote access (like SSH Keys or a VPN).

→ **Networking**

Consul is a server / service for running your networking infrastructure – think a centralized registry of all of your different infrastructure, and policies for how they can connect to each other.

→ **Applications**

These products help teams deploy their apps on existing infrastructure setups.

- **Nomad** – sort of a Kubernetes alternative. Lets you deploy apps on a pool of infrastructure, containerized or otherwise.
- **Waypoint** – Automates deploying your app on any infrastructure, like Kubernetes or EC2.

All of the above products are open source. But Hashicorp does make at least some money, and they do that by packaging a couple of these services into a cloud-based offering, mostly for enterprises (on-prem). The big two are Consul and Terraform Cloud, but Vault is now in a public beta as well. They're fully managed – i.e. you don't need to run your own Hashicorp servers).

Deeper dive: Terraform

To get a better sense of what you'd be doing with these products, let's dive a bit deeper into Terraform. It's the piece of Hashicorp that you'll see discussed most often on ~the web~. Terraform helps you manage infrastructure primitives – things like virtual machines, Docker containers, databases, really anything.

→ Why you'd use Terraform

You might use Terraform if:

- You're on a platform team and need to build an interface for other engineers at the company to work with your infrastructure securely
- You want a simpler way to work with the AWS API
- You need an easy way to create / manage / destroy infrastructure from multiple providers at the same time

The product is pretty broad, so you'll find users on smaller and larger teams (and companies) alike.

→ Core concepts

You can use Terraform to create, manage, and destroy infrastructure.

- **Create** – spin up new virtual machines, Docker containers, Kubernetes clusters, etc.
- **Manage** – change machine sizes, regions, rollover clusters, etc.
- **Destroy** – you can guess what this one means

Terraform's secret sauce (or at least some of it) is the concept of **state**. The product keeps a running log of exactly what your active infrastructure looks like at a given point in time – that helps developers debug and make changes without breaking things. You can even store that state remotely, which is part of why folks pay for Terraform Cloud.

→ Quick example

Everything in Terraform starts with a **provider**. You can use Hashicorp official ones, like the AWS provider, but there are hundreds built by the community themselves. Everything is available via their Registry site.

For our example, the provider we need is AWS. So we'd get started by declaring the provider and resources we'd want to use:

```
terraform {
  required_providers {
  aws = {
    source  = "hashicorp/aws"
    version = "~> 2.70"
    }
  }
}

provider "aws" {
  profile = "default"
  region  = "us-west-2"
}
```

Then we might add a resource that we want our fellow developers (or ourselves) at the company to be using:

```
resource "aws_instance" "example" {
  ami           = "ami-830c94e3"
  instance_type = "t2.micro"
}
```

These properties (AMI and Instance Type) are infinitely configurable, and there are hundreds to choose from based on your provider. The power of Terraform is that once you define them here, you can have users spin up new ones based on existing configurations.

If later on we wanted to update the machine image that this little resource is based on, Terraform gives us a nice change management utility:

```
resource "aws_instance" "example" {
- ami           = "ami-830c94e3"
+ ami           = "ami-08d70e59c07c61a3a"
  instance_type = "t2.micro"
}
```

This is a tiny piece of what you can do – read more in the docs here.

Comments

Write a comment...