ActualTech MEDIA

upbound

The Future of Infrastructure as Code

Dan Sullivan

CONTENTS

IN THIS PAPER

Infrastructure as Code (IaC) is an essential feature of modern infrastructure management that enables organizations to treat deploying infrastructure a lot like deploying. There are two primary approaches to using IaC: Using it as a blueprint for resource deployment, or using it to specify a full deployment and what should persist over time. The former suffers from uncontrolled changes to configurations and collaboration challenges while the latter minimizes the need for human intervention to detect and address changes. Control planes, and Crossplane in particular, are the foundation for deploying the latter model.

Highlights include:

- IaC can be used as a guide for setting up infrastructure that's then manually managed
- A better approach is to adopt techniques used in Kubernetes to monitor state and apply changes as needed to maintain the desired state
- Crossplane extends Kubernetes to allow for declarative, adaptive management of IaC

Infrastructure as Code (IaC) is an essential feature of modern infrastructure management. It allows admins to focus on the "what" instead of worrying about the "how." A key advantage of this is that IaC lets organizations treat deploying infrastructure a lot like deploying software, and enterprises have become quite good at deploying software. So we can see the broad similarities between software engineering practices and infrastructure management, but what does this mean for enterprises looking to adopt these technologies and practices?

The future of IaC lies with control planes

IaC has a lot of promise, at first glance it sounds like it lets us describe what we want in our infrastructure and then we get it. Of course, it isn't that simple. Some IaC approaches are very good at standing up infrastructure according to our specifications; however, things change. We work in organizations that have constantly changing priorities and we live in a universe subject to entropy. What we deploy today as infrastructure may not be functioning as we expect tomorrow. Not only do we need a way to deploy infrastructure, but we need a self-correcting mechanism to address unexpected and unwanted changes to our infrastructure.

IaC is a declarative specification for what infrastructure to deploy. For example, we can declare that we want to deploy a VM in a cloud with 4 CPUs, 256GB of memory, 2 local SSD drives with 500GB of storage each, and that that VM should have access to a particular subnet in our virtual private cloud.

> Not only do we need a way to deploy infrastructure, but we need a self-correcting mechanism to address unexpected and unwanted changes to our infrastructure.

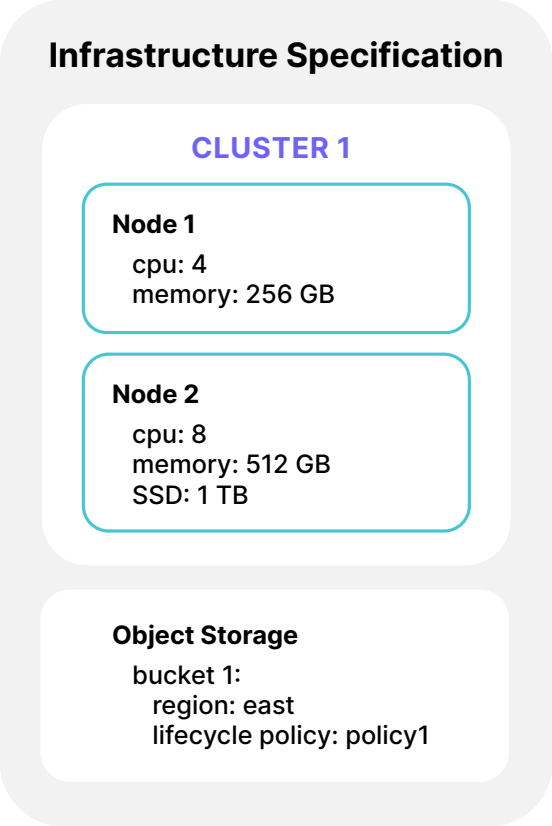

Figure 1: IT resources become something that can be configured in software with IaC

The entire specification is just about what the VM should be, but no instructions about how to actually implement what we want. IaC enables automatic implementation of your configuration, sidestepping a manual approach. This is the essence of IaC (see **Figure 1**).

There are two primary approaches to using IaC: Using it as a blueprint for resource deployment, or using it to specify a full deployment and what should persist over time.

The first option is like an architectural drawing for building a house. It lays out the initial setup of the building, but once the house is built, owners may make changes as they see fit. Over time, those first blueprints will no longer accurately describe the state of the house—e.g., walls have been added and removed, carpet has been replaced with hardwood, and so on. This method of using infrastructure as code lets organizations deploy an initial infrastructure state efficiently, while still allowing you to alter it at will moving forward.

The second approach is to use IaC as both an initial blueprint, and a method of managing infrastructure configurations over time. This option is more analogous to a flight path. It describes a course the plane will fly, and can be used during the flight to adjust course as needed. It's an extended use of IaC to help manage infrastructure, instead of just a starting point. This option works well in dynamic environments where changes are likely to occur, like environments where resources like VMs or containers might fail and must be replaced automatically.

Common problems with using IaC as a one-time blueprint

UNCONTROLLED CHANGES TO CONFIGURATIONS

IaC has many advantages over manually managing infrastructure or using imperative scripts to create infrastructure by executing a fixed set of steps. Manual changes to infrastructure are error-prone and difficult to scale, which is why many system administrators have opted to write scripts that execute specific steps to deploy infrastructure.

While an improvement over manually changing your infrastructure, these scripts tend to be brittle because they typically assume a known starting state. For example, a script might deploy a set of VMs using a particular operating system, and assign specific IP addresses to those VMs. This works well when all the VMs have to be deployed at once, and none of the VMs are currently deployed.

But consider the problem of a single VM becoming unhealthy. The script is designed to deploy all the VMs, not detect when one has failed. In this case, a system administrator could manually deploy a new VM or copy and edit the script to deploy just a single instance. There's no way to automatically detect the changes in the state of your infrastructure and correct for that change. This is known as "configuration drift," and is a significant obstacle to smooth-running infrastructure.

Configuration drift isn't just the result of unhealthy VMs or containers. DevOps engineers and developers may decide they need additional infrastructure and deploy a new VM or create a Kubernetes cluster. This may be acceptable in development environments, but in most enterprises, we need to have more control over our infrastructure.

COLLABORATION CHALLENGES

A hallmark of modern software engineering is collaboration. We build and work with complex systems that can entail an array of technologies, including custom software applications, databases, and networks.

Creating a production environment can require contributions from a team of engineers with different areas of expertise. One person might know the details of deploying an application, while another member of the team knows how to configure data pipelines to send data from the application to a data warehouse. A network engineer knows how to configure virtual private clouds, set up subnets, and control traffic between subnets using firewall rules.

Like other areas of software development, modularization is used to manage complexity. The problem, though, is that modularization of IaC is difficult to get right. There may be subtle dependencies between modules that are difficult to discern.

As a result, engineers may have to spend a lot of time refactoring their IaC modules. This in turn can be disruptive on development teams if they cannot count on having stable development environments. It can also lead to delays in deploying production environments where it's important to get things right the first time.

When we do use modules, which are like software libraries, developers may have to learn a new configuration language. The effort can be worth it because using modules raises the level of abstraction for application developers. Unfortunately, it doesn't necessarily raise the level of control abstraction.

EXAMPLE DEPLOYMENT

Let's consider how we would deploy three VM instances in a public cloud using a tool like HashiCorp's Terraform.

We start with creating a file that specifies the resources we want to deploy. This requires that we tell Terraform what cloud provider we're working with, as well as the region or other location information that may be required.

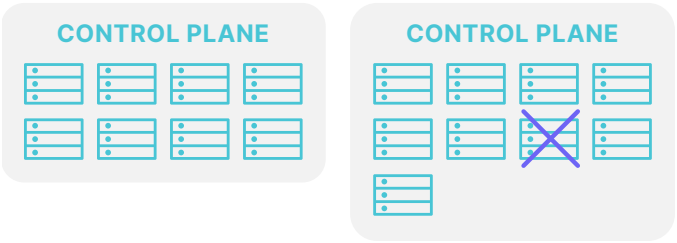

Figure 2: Control planes are used to maintain the desired state of infrastructure

We then list our resources we want. In the case of VMs, we can specify the machine type, operating system, and other specifications.

Next, we use the command line or a CI/CD pipeline to have the infrastructure deployed by Terraform. The IaC tool will inspect that state of infrastructure and build an execution plan that details the steps needed to have the deployed infrastructure match the description in the Terraform specification file. After the execution plan runs, the infrastructure will be in the desired state.

A significant shortcoming of this approach is that something about the state of infrastructure can change without someone detecting it. Our resources could be misconfigured for an extended period of time until someone notices the difference between the actual state of infrastructure and the desired state (see **Figure 2**). In a worst-case scenario, a service or application can be down, and the infrastructure problem is discovered only after users complain about a service outage.

> IaC brings new levels of flexibility, controls, and reproducibility to enterprises deploying infrastructure on-premises and in the cloud.

This is a similar problem to one we find in container orchestration, where a container may fail or there's some other change to the deployed state of a service. Kubernetes addresses this problem by monitoring the state of services and compares it to the desired state: If there are differences, Kubernetes executes actions to bring the existing state back to the desired state. Ideally, we would have a similar tool for managing IaC.

Crossplane: an open source universal control plane

Crossplane is an open source framework for building control planes. Crossplane extends the capabilities of Kubernetes and combines the power of IaC with the built-in monitoring and state enforcement capabilities of a control plane. With Crossplane installed, platform teams enable their Kubernetes clusters to manage infrastructure resources which exist outside of their clusters, constantly reconciling resource state based on the definitions in the cluster.

Crossplane connects to resources outside of a Kubernetes cluster via "providers," which are essentially plugins. Providers install custom resource definitions, or CRDs, for infrastructure to be managed. These CRDs are deployed into the cluster Crossplane is installed into, enabling Kubernetes tools and functions to operate on any piece of infrastructure or cloud resource, from Amazon S3 buckets and networking to bare metal servers hosted on-premises. Providers bridge the gap between Kubernetes APIs and operations and any third-party resource creating an end-to-end, API-first infrastructure.

A MORE ADAPTIVE AND DYNAMIC IAC

IaC brings new levels of flexibility, controls, and reproducibility to enterprises deploying infrastructure on-premises and in the cloud. IaC systems alone are not enough, though. IT environments are dynamic, and we need automated tools to maintain those environments. Control planes are the key to implementing a more adaptive and dynamic form of IaC.

To learn more about IaC, control planes, and how they can help manage increasingly complex infrastructure, schedule a demo with Upound today.