ActualTech MEDIA

upbound

From Configuring to Controlling: Transforming Cloud Operations

Dan Sullivan

CONTENTS

IN THIS PAPER

While there have been great leaps made in the individual realms of software and hardware, the way businesses organize their IT resources utilizing new software and hardware configurations has drastically shifted infrastructure design philosophies.

In the current era, the most recent method of segmentation pushing infrastructure design forward is containers.

Highlights include:

- How Kubernetes has changed the way organizations manage their infrastructure
- An essential feature of modern infrastructure management is Infrastructure as Code (IaC)
- Learn how Upbound's fully managed control planes enable you to scale your infrastructure while minimizing system administration overhead

Some of the largest advancements in information technology over the past 30 years have been behind the scenes in infrastructure management. While there have been great leaps made in the individual realms of software and hardware, the way businesses organize their IT resources utilizing new software and hardware configurations has drastically shifted infrastructure management philosophies.

Data centers were once home to a single mainframe running multiple applications, but the advent of server technology shifted the common way of deploying applications. A large part of the infrastructure was dedicated to servers each running a single application.

> The most recent method of segmentation pushing infrastructure design forward is with containers.

Then virtual machines (VMs) shifted infrastructure design again. VMs could increase efficiency by allowing multiple applications to run in isolated environments, but on a single server. Each step forward has been driven by the ability to further segment and streamline infrastructure.

The most recent method of segmentation pushing infrastructure design forward is with containers.

The way we utilize infrastructure is increasingly similar to the way large cloud providers manage their infrastructure businesses

Today, containers extend the efficiency of running applications beyond VMs, and software engineers are taking advantage of the benefits of containerization to build more scalable applications.

The downside is that each new container adds to the complexity of the overall infrastructure, and an organization utilizing containers will have greater management challenges than one running a small set of servers. Kubernetes has changed the way that organizations manage their containerized infrastructure—but what is Kubernetes?

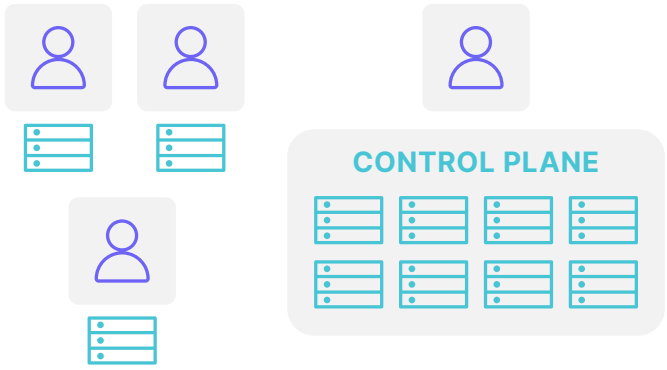

Figure 1: Automated orchestration enables more efficient and scalable use of resources

Simply put, Kubernetes, sometimes abbreviated as "K8s," is an open source platform for managing containerized applications and services. Announced by Google in 2014, K8s was influenced by Borg, Google's in-house cluster manager, although the company worked with The Linux Foundation and the Cloud Native Computing Foundation (CNCF) to open source the platform.

The name itself is from the Greek term for helmsman or pilot, with the thing it "pilots," or orchestrates, being containerized applications (as shown in **Figure 1**). This may seem like just a fun fact, but the etymology of the name can be useful in understanding the benefits of Kubernetes.

BENEFITS OF KUBERNETES

Kubernetes has changed the way organizations deploy and manage infrastructure. It provides service discovery to make applications and services easily accessible by their network. It also provides load balancing, so as workloads increase or decrease, K8s will distribute the load and deploy or remove compute resources as needed, and Kubernetes supports storage orchestration using both local and cloud-based storage services. These features cover a lot of ground, but Kubernetes deployments often impact almost every aspect of an organization's infrastructure.

The foundation of any K8s environment is a description of a desired state. The user first describes the desired functionality and connectivity of their Kubernetes

environment, and K8s monitors the environment for deviations from that desired state. If that state of the cluster deviates from the desired state then Kubernetes will work to bring it back to its expected configuration and functionality. This process involves the monitoring and manipulation of many aspects of an infrastructure.

K8s works to optimize the use of compute and storage resources to run containers efficiently. For instance, if a cluster is underutilizing resources or found to be redundant, K8s, in certain configurations, can remove resources, nodes, or entire clusters to maintain efficiency.

> Kubernetes has changed the way organizations deploy and manage infrastructure.

Similarly, Kubernetes can monitor the health of containers, as well as the K8s abstractions called "pods," and will replace any that are failing or unhealthy. K8s can also help manage configurations and store secrets, such as SSH keys and OAuth tokens. While Kubernetes comes with many benefits by being flexible and powerful, there are many challenges associated with adopting and deploying Kubernetes to production.

KUBERNETES CHALLENGES

First, Kubernetes is complex. It's designed to orchestrate many different types of resources, including compute, storage, and networking resources. It also provides many types of functionalities, like load balancing and health monitoring.

What this adds up to is a platform that can facilitate many complex interactions. For example, K8s makes it simple to increase the number of container deployments, but this can put a heavy load on servers or nodes. This would lead to multiple containers experiencing a spike in workload and cause degraded performance until the issue is resolved.

As Kubernetes impacts so much of an organization's infrastructure, administrators must utilize the available tools to avoid problems such as this. Of course, many aspects of operations are automated, and that requires complex policy making and configurations to specify how each service or resource should be managed. There's also a growing ecosystem of tools to support K8s management, but each of those have their own elements of complexity. One method of meeting the challenges presented by Kubernetes deployments is using Infrastructure as Code tools.

INFRASTRUCTURE AS CODE

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." For example, it can enable admins to specify what specific elements of an infrastructure should be deployed, such as the number of servers and CPUs, the amount of memory, and the number and size of storage disks. IaC then removes some configuration barriers.

The reduced focus on how infrastructure is deployed means organizations don't need to know platform-specific commands for deploying VMs, or how to configure network rules to access them. Instead, IaC lets you specify which networks or subnets should be accessible, then lets the Kubernetes platform configure it for you. This means deployment is automated while the decisions that require human creative thinking, such as configuring infrastructure for a specific use case, can still be done manually.

> While Kubernetes comes with many benefits by being flexible and powerful, there are many challenges associated with adopting and deploying Kubernetes to production.

In the end, IaC lets organizations treat deploying infrastructure a lot like deploying software. Infrastructure specifications can be treated as code, meaning version control, code reviews, and deployment pipelines all end up specified in written code. This all adds up to what looks like the first iteration of a cloud-like data center operations philosophy. But what comes next?

The future of IaC and how control planes are the new solution

First, let's dive a little deeper into IaC. 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.

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 2**).

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.

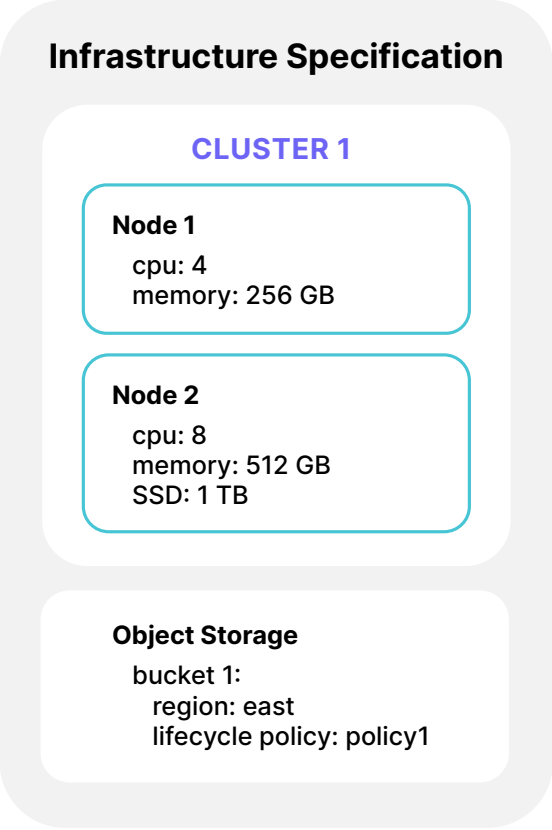

Figure 2: IT resources become something that can be configured in software with IaC

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.

Using the first option of utilizing IaC as an initial blueprint, container replacement would need to be done manually, allowing room for someone to deploy a VM in a way that violates a policy. In the end, this second option lends itself to a more robust method of self-correction and promotes automated infrastructure management.

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 infrastructure as code 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. 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.

Having a tool that can deploy infrastructure to our specification is a valuable resource. If there's little to no chance of configuration drift or unhealthy resources, this approach to IaC may be sufficient. Engineers and system administrators can use the command-line utility or CI/CD pipeline to run it again whenever they think it may be needed.

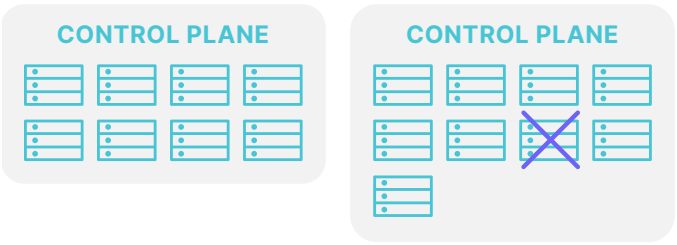

Figure 3: Control planes are used to maintain the desired state of infrastructure

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 3**). 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.

This is a similar problem to one we find in container orchestration, where a container may fail or there is 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.

Open source Crossplane provides a universal control plane based on Kubernetes

Crossplane is an open source framework for building control planes. Crossplane extends the capabilities of Kubernetes and provides the power of IaC with the built-in monitoring and state enforcement capabilities of a control plane. Terraform takes a different approach, relying on a custom "domain specific language" and its own methods to verify the state of the infrastructure.

Crossplane was donated to the CNCF in 2020 and became an incubating project in 2021. The Crossplane community is rapidly expanding with 8% monthly growth in their Slack channel, over 5,000 stars on GitHub, and 60-plus open source providers.

Crossplane relies on "providers" to allow native 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.

Beyond providers, Crossplane introduces the concepts of "claims" and "compositions." These allow infrastructure operators to build their own customized clouds while simplifying how developers consume these cloud resources. Claims are custom APIs defined by the infrastructure owners that only expose the things users care about. For example, a claim may only allow a developer to ask for a "small" or "medium" database. This custom API then works in tandem with Crossplane compositions. Compositions live on the other side of the API line and define all the components required to provide a resource to a developer. Things like which cloud provider, how much disk, adding resources to monitoring systems, and the required networking and security policies. These things are unnecessary details to the developer but critically important to infrastructure operators.

> Crossplane extends the capabilities of Kubernetes and provides the power of IaC with the built-in monitoring and state enforcement capabilities of a control plane.

By extending Kubernetes, all the Crossplane components are REST API-enabled, including providers, claims, and compositions. Everything is just another element within the Kubernetes cluster. A VM running in Azure and a database in AWS are both managed through the same Kubernetes APIs and tools that manage native pods and nodes.

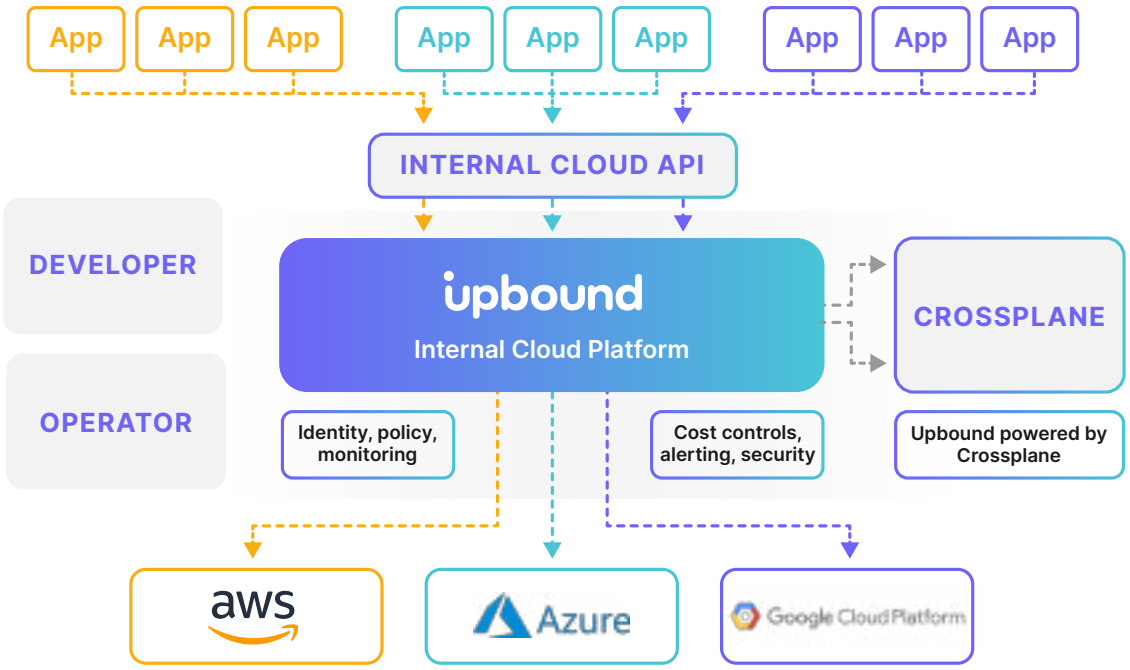

Figure 4: Upbound brings fully managed control plane services to the enterprise

Why Upbound is the key to taking control planes to production

Crossplane enables enterprises to manage their infrastructure the way hyperscaling cloud providers do. If you don't want to manage your own Crossplane clusters and expend time and staff on learning to optimize control planes, Upbound's fully managed control planes will enable you to scale your infrastructure while minimizing system administration overhead (see **Figure 4**).

Upbound offers a turnkey experience. Control planes provided by Upbound are designed for high performance and security, while still supporting multi-tenancy. Upbound also provides access to a component marketplace for rapidly integrating additional tools into your control plane.

Customers future-proof their infrastructure because Crossplane can be adapted to support new types of infrastructure. Since Upbound is designed around Crossplane, it provides a control plane-driven approach for customers to build internal cloud platforms on top of the services and infrastructure they already have. Customers also benefit from reduced costs and increase efficiency from automating the management of complex infrastructures.

Upbound democratized control plane technology with the Crossplane project. Upbound's collective expertise in Crossplane is difficult to match, and it has used its experience and knowledge of complex distributed systems to build a platform for customizable control planes.

Learn more about Upbound

To learn more about Crossplane and how Upbound's managed control plane services can help you realize more efficient and reliable service delivery, see Upbound's documentation for details on Universal Crossplane, Upbound Cloud, Upbound Marketplace, and Upbound Enterprise. Also, see the Upbound blog for details about new functionality and ways of using control plane technologies and join the Upbound Slack channel to learn more about how Upbound is being used in enterprises like yours.