Kubernetes for Operators

VMware Tanzu

Table of Contents

Containers and Kubernetes: Why Operators Should Care

If you're part of an operations team, you know that your company's success depends on your team's ability to operate digital services reliably at scale. Enterprises are increasingly turning to cloud native technologies, including containers and Kubernetes, to achieve this goal.

If you are contemplating this transition, you are likely under pressure from above and below. Executives are asking you to deliver more reliable services at a lower price point, while development teams want containerized infrastructure on which to create and deliver new applications.

In the traditional approach, your team has to manage a large number of dependencies with custom configurations for each application. Kubernetes is a way to commoditize the way you deliver infrastructure, simplifying management and leveling the playing field for operations of all sizes. Containers enable much greater application portability.

For operators, containers and Kubernetes have many benefits:

Eliminate the need to manage application dependencies at the infrastructure level

Deliver infrastructure in a more cloud-like way

Run an application on almost any infrastructure—on-premises or in the cloud

Move applications easily between environments in response to need or cost

Cloud native technologies are new and evolving fast. A whole ecosystem of solutions and services is emerging to address a wide variety of use cases and needs. There's a lot to learn. This eBook will help you map your company's journey to containers and Kubernetes, including important questions your team should ask itself and an exploration of the most common missteps.

Who Should Read This Book?

No two organizations are alike, and titles can vary widely from one to the next. However, this book is targeted to people that focus on platform and infrastructure operations.

Infrastructure engineers, systems engineers, and site reliability engineers (SREs)—anyone responsible for the infrastructure on which Kubernetes will run—can benefit.

Why Containers?

One of the big challenges that operations teams face is the complexity of managing highly custom applications and the difficulty of moving applications from one environment to another. By encapsulating all of an application's dependencies, containers make applications much more portable—and therefore make an operator's job simpler. A container can move from development to QA to production—or from one cloud environment to another—without requiring any changes to the container—and with no hardware and software reconfigurations in the target environment.

While many organizations are moving existing legacy applications into containers more or less as is, new application development makes use of containers in conjunction with a microservices architecture that breaks down an application into component services. From an operational standpoint, there are two critical things to understand about the microservices approach: applications scale out instead of scaling up, and there are many more application components to manage

A Kubernetes Primer

Container environments change more rapidly than VM environments. Having a way to manage containerized applications effectively is an essential element of cloud native and microservices architecture. Kubernetes has emerged as the leading solution for orchestrating and managing containerized applications.

The components of Kubernetes "play off" each other to coordinate activities and react to events like musicians playing jazz. At its core, Kubernetes is a database with some interesting features layered on top of it. Kubernetes uses a set of controllers that each implement specific capabilities and work together to produce the end result. Kubernetes controllers can be ripped out and replaced to extend the system and adapt it to new requirements and environments.

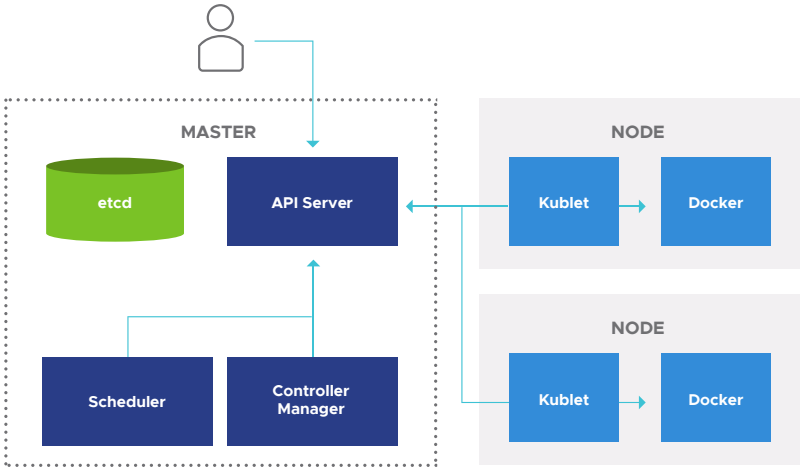
The diagram below shows the parts of a typical Kubernetes system. The core of the system is the database, etcd. The state of the cluster is stored there (and only there). In front of etcd is the API Server. Nothing else in Kubernetes talks to etcd directly. The API Server exposes a RESTful interface and provides the services necessary in a distributed system.

ALL interactions with Kubernetes are mediated via APIs. This approach can be a big change for both operations teams and developers.

Containers encapsulate an application in a form that's portable and easy to deploy. Containers can run on any compatible system—in any cloud—without changes. Containers consume resources efficiently, enabling high density and utilization.

Kubernetes makes it possible to deploy and run complex applications requiring multiple containers by clustering physical or virtual resources for application hosting. Kubernetes is extensible, self-healing, scales applications automatically, and is inherently multi-cloud.

Microservices architecture breaks down an application into multiple component services, enabling greater parallelism during both development and execution.



The Scheduler and the *Controller Manager* implement most of the orchestration logic of Kubernetes. Together, etcd, the API Server, Scheduler, and Controller Manager make up the Kubernetes control plane; they can run on a single node or across multiple nodes for availability.

Worker nodes make up the data plane of Kubernetes; each worker node runs the container runtime (Docker in the diagram) and a local daemon called the *Kubelet* that communicates with the API Server.

Kubernetes Controllers

Kubernetes Controllers ensure that the observed state of the cluster is as close as possible to your desired state. Each controller monitors its configuration, stored in the API Server. It then looks at the state of the world and tries to make the state of the world match its configuration. If a controller can't fully achieve the desired state, it retries. Controllers are both patient and diligent, resulting in a very stable distributed system pattern that is self-healing. If something goes wrong, a controller will work to fix it. If the desired state changes while a controller is working, it changes course and works toward the new desired state. Controllers react to each other very quickly, making Kubernetes extremely responsive. The actions of the system adapt to the state of the world in real time.

How Kubernetes Helps Operators

For platform operators, infrastructure resources in Kubernetes are clustered and can be consumed and released elastically, enabling seamless scaling and higher resource utilization. Kubernetes eliminates many of the manual provisioning and other tasks of conventional enterprise IT.

Kubernetes clusters deployed in different private and public clouds provide a uniform (or very similar) management environment and identical principles of operation, reducing the learning curve associated with managing a multi-cloud environment and minimizing the risk of operator errors.

Portability

Run Kubenetes everywhere.

Leverage a common upstream framework to run workloads on-premises, in public clouds or hybrid cloud

Integration

Build on your SDDC infrastructure.

Apply Kubernetes as a practical path to public cloud adoption

Community

Move in lockstep with the community.

Work with a partner that demonstrates leadership in the open source community.

Two Elements of Kubernetes that All Operators Should Know About

If you are new to Kubernetes, there are two emerging elements of the environment that you should know about. These two things are rapidly changing the way that operators interact with Kubernetes:

Operators. Custom Kubernetes controllers that implement domain-specific logic for an application

Cluster API. Declarative APIs that facilitate cluster creation, configuration, and management in the Kubernetes ecosystem

Operators Tailor Kubernetes to Application Needs

Kubernetes Custom Resource Definitions or CRDs extend the resources the API Server can manage. CRDs are often paired with a custom controller called an Operator. By automating application-specific tasks that otherwise have to be done manually, Operators allow you to more easily deploy and manage applications on Kubernetes.

Operators encapsulate domain-specific knowledge for a specific application. You can think of this as embodying the knowledge and logic that might traditionally be captured in run books. The open source Operator Framework provides the necessary tools to facilitate Operator creation.

Cluster API Streamlines Kubernetes Provisioning and Management

Infrastructure-level management is an area of rapid evolution for Kubernetes. Cluster API is a Kubernetes project to bring declarative, Kubernetes-style APIs to cluster creation, configuration, and management. It provides additive functionality on top of core Kubernetes. VMware is actively contributing to the development of the Kubernetes Cluster API.

One of the goals is to provide a way to let operators declaratively define what a cluster should look like. For example, using the Cluster API you might declare that you want seven servers running in your Kubernetes cluster. If you later decide that you want eight servers running, you use the Cluster API to change the number to eight and it will automatically spin up another virtual machine, applying Kubernetes logic to infrastructure. If one of those servers fails, Cluster API will automatically spin up a new one to take its place, using self-healing to return to the desired state.

To make Cluster API work for a particular type of environment, you need a provider for that environment. Provider implementations are already available for major public clouds as well as VMware vSphere. GitHub has a list of many of the available providers.

Where to Learn More

The What and the Why of the Cluster API [blog]

Cluster API Gitbooks

First Principles for Operators

There are a few basic principles that will help your organization move forward with Kubernetes.

You don't have to decide everything upfront. Because Kubernetes is flexible and extensible, decisions you make today don't necessarily lock you in for the rest of time. For example, you can choose NGINX ingress controller now, and change to something else like Contour in six months or a year as your needs change.

Everything doesn't have to be greenfield. You don't have to start from scratch and deploy all new tools. Look for integration points with the tools you are already using like Jenkins.

Enable your stakeholders. Don't ignore the needs of others that will be using the platform. If developers are used to being able to access system logs for debugging, make sure they still have access and know how to take advantage of any new tools.

Top Six Questions to Answer Upfront

While you don't have to answer every design question upfront, there are a few questions you should consider carefully before you start deploying Kubernetes. In this section, we look at the top six questions. While the decisions are ultimately up to your team, we've tried to provide guidance based on experience with many teams adopting Kubernetes.

Question 1: Where should I put my Kubernetes cluster?

The first question that teams grapple with when they approach Kubernetes is where to deploy it. Should they deploy on-premises? In the cloud? There are a range of factors you need to consider:

On-premises. If you deploy Kubernetes on-premises, you'll have complete control and more flexibility to tailor the deployment to your business needs. On the other hand, Kubernetes deployment requires some expertise, and there are a lot more decisions you'll have to make—and a lot more things you'll ultimately be responsible for managing. As you'll learn shortly, there are a few decisions you have to get right from the beginning.

Cloud. Cloud-based Kubernetes services like Amazon Elastic Kubernetes Service (Amazon EKS), Google Kubernetes Engine (GKE), and Azure Kubernetes Service (AKS) provide key primitives that can make it easier to get started. However, you may be trading future flexibility for speed and ease of initial deployment.

Question 2: How should I build my Kubernetes cluster?

There are two dimensions to this question: What infrastructure environment should you choose? And, what software should you install in that environment? If you choose a cloud service, those decisions are largely decided for you.

If you're deploying on-premises, will the infrastructure be virtualized or bare-metal? Both can work, so it may come down to what you have and what you're most comfortable with. The major cloud providers deploy Kubernetes on top of virtual machines, as do many enterprises with running VM environments.

Will you purchase new hardware for your deployment or repurpose existing hardware? This largely depends on whether you have up-to-date hardware available for the purpose.

When it comes to software, the biggest decision is whether to choose:

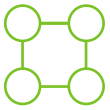
- A hosted cloud service
- A packaged software distribution
- A full do-it-yourself (DIY) installation from open source

This topic is discussed in more detail in the following chapter. Our general guidance is to stay as close to upstream Kubernetes as possible no matter which option you choose. Compatibility ensures that you can move operations between environments with no or minimal changes. Kubernetes is evolving quickly so you want to ensure compatibility with the latest versions and avoid wandering into any walled gardens.

Question 3: How many clusters should I build?

A question that comes up all the time is whether or not to have everything in a single large cluster. You should think of Kubernetes as a multi-cluster solution and cultivate a multi-cluster mindset. Having multiple clusters reduces the size of each failure domain and provides you with greater flexibility going forward. For example, you can stand up a new cluster with a new feature and migrate services to that cluster to take advantage of the feature.

As a purely practical matter, the maximum "comfortable" size of a Kubernetes cluster is around 500 nodes. Beyond that, you will have to start tuning Kubernetes itself to continue scaling; that's an arduous process..



Question 4: What about underlying infrastructure?

There are several infrastructure "plumbing" considerations that you definitely need to think about and understand before you begin to deploy. These may be difficult to change after the fact:

Container networking. Kubernetes gives you flexibility regarding networking through Container Network Interface (CNI) plugins. There are a variety of CNI plugins that support various approaches to software-defined networking (SDN) and various network options. If you have to have a particular network capability (for example multicast) you'll need a CNI that supports that feature. You can view a list of available CNI plugins on GitHub.

Persistent storage. A surprising number of teams get fairly far along in the Kubernetes planning process without thinking about storage. Similar to networking, Kubernetes provides storage flexibility through drivers that conform to the Container Storage Interface (CSI). Many storage vendors offer drivers for compatibility with Kubernetes. You can view a list of available CSI drivers on GitHub.

Connectivity. Plan ahead to make sure you have the appropriate infrastructure surrounding Kubernetes to support your application needs. This may include things like load balancers and ingress controllers.

Security. A final item that's hard to bolt-on to Kubernetes after the fact is security. Be sure and involve your security team in Kubernetes planning.

Question 5: What about security?

The security model is an area that people often overlook during Kubernetes planning phases:

Multi-team vs. Multi-tenant. This distinction can be useful to think about during security planning:

Multi-team. You need infrastructure to support different teams within the same organization so there is a certain level of trust.

Multi-tenant. You need infrastructure to support separate organizations where there is no trust relationship.

This distinction affects your approach to security. Kubernetes does not have a hard multi-tenancy design. Multi-tenancy can be enabled, but it doesn't come right out of the box.

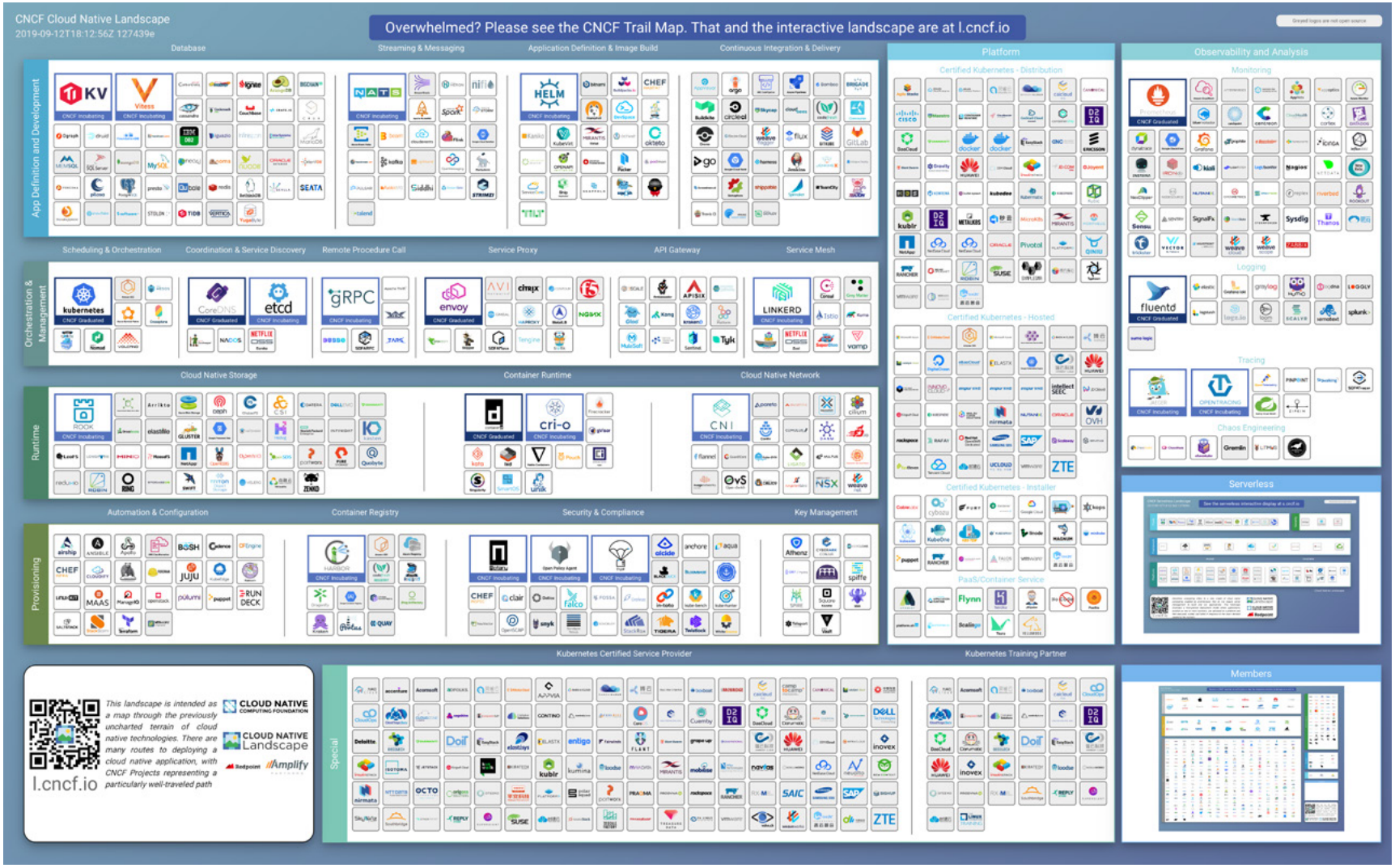
Authentication/authorization. Use something like OIDC and connect your existing authentication system to Kubernetes from the beginning. It's simple to do and worth the effort.

Policies. People often deploy Kubernetes and forget to configure things like resource quotas and pod security policies. These are essential both to secure your cluster and to achieve high levels of utilization.

Backup and restore. Having a regular backup policy is essential to protect your Kubernetes environment. It enables you to move applications to another cluster in a disaster situation and is also useful for moving a collection of resources from one cluster to another. Velero is a useful tool, providing backup and migration of Kubernetes applications and their persistent volumes.

Question 6: How can I extend Kubernetes to address specific operational needs?

Unlike other platforms like OpenStack, Kubernetes is highly extensible. If you have a scenario that Kubernetes doesn't address, you can extend Kubernetes without having to worry about the upgrade path going forward. You should approach Kubernetes with that mindset and be prepared to take advantage of existing Kubernetes extensions where possible—and to develop your own when necessary. The Cloud Native Computing Foundation (CNCF) maintains a landscape that shows many of the active Kubernetes projects in areas ranging from database to key management to observability, making it a little easier to identify the ones that you want to either adopt immediately or track for possible future use. Be warned, however, that the number of items in the landscape is already a little overwhelming.


CNCF Cloud Native Landscape
2019-09-12T18:12:56Z 127439e
Overwhelmed? Please see the CNCF Trail Map. That and the interactive landscape are at l.cncf.io
Database
Streaming & Messaging
Application Definition & Image Build
Continuous Integration & Delivery
App Definition and Development
Scheduling & Orchestration
Coordination & Service Discovery
Remote Procedure Call
Service Proxy
API Gateway
Service Mesh
Orchestration & Management
Cloud Native Storage
Container Runtime
Cloud Native Network
Runtime
Automation & Configuration
Container Registry
Security & Compliance
Key Management
Provisioning
Kubernetes Certified Service Provider
Kubernetes Training Partner
Platform
Certified Kubernetes - Distribution
Certified Kubernetes - Hosted
Certified Kubernetes - Installer
PaaS/Container Service
Observability and Analysis
Monitoring
Logging
Tracing
Chaos Engineering
Serverless
Members
Special
This landscape is intended as a map through the previously uncharted terrain of cloud native technologies. There are many routes to deploying a cloud native application, with CNCF Projects representing a particularly well-traveled path
l.cncf.io
CLOUD NATIVE COMPUTING FOUNDATION
CLOUD NATIVE Landscape
Amplify

Which Flavor of Kubernetes Should You Choose?

Earlier you learned that there are three basic flavors of Kubernetes:

Managed Cloud Service

Kubernetes distribution

Do it yourself

One of your first decisions, as noted earlier, is to decide which one(s) to choose. The good news is that if you get six months into deployment and decide for whatever reason that you made the wrong initial decision, you can still switch strategies and recover. Whatever work you've done in one environment should be largely compatible with the other environments. After all, that's one of the key strengths of Kubernetes and containers.

Managed Kubernetes

The major cloud providers all offer Kubernetes platforms that provide an easy, turnkey solution that you can use to get up and running with Kubernetes in a managed environment. The hosted cloud solutions have all committed to maintaining compatibility with upstream Kubernetes. However, if portability is important to your operations, you still need to be careful to avoid incorporating other services that are only available in a particular cloud.

Kubernetes Distributions

A variety of vendors—Red Hat, Canonical, and many others—offer packaged Kubernetes distributions. For anyone who has used Linux, the packaged distribution model will seem familiar. However, there's an important distinction: Kubernetes is young and changing extremely quickly. Because they may limit future compatibility and flexibility, avoid deep forks that diverge significantly from upstream Kubernetes.

Distributions provide greater control than managed Kubernetes, including full access to the control plane. They can work well if your operations stay within the confines of what the distribution offers. However, if you need to extend beyond the boundaries created by the distribution, you could experience issues.

Do It Yourself

The final approach is to take a complete, do-it-yourself approach and build Kubernetes from source, either on your own or with the help of a partner. This approach requires the most technical expertise and more personnel. You'll need to pay more attention to where Kubernetes is headed and create a cluster lifecycle strategy to keep up with changes, and you'll be more dependent on the community for support.

However, this is the option that gives you the most flexibility and the greatest control. Our opinion is that it's not as hard as people think and you shouldn't simply rule it out. We continue to believe that this is the best option for many organizations.

Avoiding Missteps

The following table describes some of the most common missteps that we see organizations make as they move to adopt Kubernetes—along with tips on how to avoid them.

Common Kubernetes Missteps
Misstep 1: Going all-in on managed Kubernetes It's awesome to no longer have to manage your control pane. However, there are downsides associated with not having full control. Some limitations may be showstoppers. with tips on how to avoid them.
Misstep 2: Solving problems you don't yet have With platforms that aren't extensible, if you don't start out with a capability, it's hard to add later. Kubernetes turns this paradigm on its head. Designing solutions for problems you think may arise in the future just adds complexity and delays deployment.
Misstep 3: Letting perfect be the enemy of done This misstep is a corollary to the previous one. You can waste a lot of cycles guessing about future requirements and trying to build an ideal solution. If you narrow your scope, you can finish deployment more quickly and start gaining operational experience.

Common Kubernetes Missteps

Misstep 4:
Trading “battle-tested” for “cutting-edge”

If you look at the CNCF landscape, there are a lot of interesting technologies. Resist the temptation to bet your business on technology that’s brand new. Track interesting projects and give them time to mature.

Misstep 5:
Open source is not free

The burden to support open source software falls on you. Be diligent in evaluating a project before adopting it: How many stars does it have? How healthy is its community? Are people responsive? Are pull requests reviewed and merged? Are there guidelines for contributors?

Misstep 6:
Mixing too many workloads on one cluster

It is easy to add workloads to an existing cluster, but new apps may require changes that have undesired effects elsewhere, even compromising security. It’s better to have multiple smaller clusters. Managed Kubernetes makes it fast and easy to add new clusters for unique requirements.

Misstep 7:
Using available tools for federation

Federation is a hard problem, and current tools likely won’t meet your needs. Instead, ensure you have copies of application containers and data where they can be used for DR. A modest substitution of human effort for fancy federation insulates you from outages that span zones.

Misstep 8:
Not implementing resource quotas

To increase resource utilization, you must implement resource quotas that ensure that no one person or application consumes too many Kubernetes resources. This allows you to do tighter bin packing on each node.

Misstep 9:
Not using operators

Operators are becoming the preferred way for managing domain-specific knowledge and simplifying application management in Kubernetes. Take advantage of existing operators when they are available and modify or build your own operators when necessary.

What Should I Do Next?

If you're a part of an infrastructure team starting out with Kubernetes, the most important thing is to get started. Follow links to learn more, watch videos, and engage with your peers. In addition to connecting online, you may find local Kubernetes meetups in your area, and there are annual KubeCon + CloudNativeCon conferences in North America, Europe, and China. In addition, VMware has a variety of resources for everyone on the Kubernetes journey:

TGIK

Every Friday at 1PM Pacific Time, VMware holds an informal hangout session focusing on a specific Kubernetes-related topic. You can see the archive of past sessions on YouTube and subscribe to view the live sessions.

Cloud Native Apps Blog

Read our regular blog to find out the latest. Posts cover diverse topics and new blogs are added regularly.

Watch a webinar on Cluster API

Learn about Cluster API, how it works, its current state, and why it's crucial for the future of Kubernetes.

And be sure and follow @VMwareTanzu on Twitter to keep up with all the latest cloud native developments.

Kubernetes Academy

KubeAcademy provides an accessible learning path to advance your skill set, regardless of where you are on your Kubernetes journey. Courses are designed and delivered by Kubernetes experts, for free.