LUMEN® | vmware®

Lumen® Edge Private Cloud with VMware Tanzu™

For Developers

Contents

Containers and Kubernetes: Why developers should care

Your company's success increasingly depends on the ability of development teams to deliver digital services and software more quickly and with higher quality—and on the ability to operate applications and services reliably at scale. You have likely already discovered that traditional methods don't deliver new applications quickly enough or achieve the necessary velocity. For most teams, the answer is agile software development methods (sometimes called DevOps) and cloud native technologies, including containers and Kubernetes.

If you're a software developer, it's likely that you're already somewhat familiar with these technologies. You might have kicked the tires, you might have done your first project, or you might just be trying to figure out how to learn enough to get started.

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 for everyone to learn, and it's likely to stay that way for quite a while.

The journey won't always be an easy one. Kubernetes introduces additional complexity to development and production environments. But the cost of moving to Kubernetes is less than the value you get from it. That's the whole reason that Kubernetes has grown in popularity so quickly. The new tooling and the learning curve are worthwhile because the efforts you make now will save you time down the road.

There's no lack of resources out there. You might even say that's part of the problem. This book will help you think more clearly about software development for Kubernetes, whether you're porting existing applications or doing new cloud native development.

When we talk to software and application engineers, there's frustration that there's no short reference that pulls together everything they need to know to get started. This eBook is intended to fill that gap, helping you map your journey to containers and Kubernetes.

Who should read this book?

No two development organizations are alike, and titles can vary widely from one to the next. However, this book is targeted to people that focus on application development. If that sounds like you, then read on.

However, we also think that infrastructure engineers, systems engineers, and site reliability engineers (SRE)—anyone responsible for the infrastructure on which Kubernetes will run—can benefit, too.

Why containers?

One of the big challenges that enterprise software teams face is difficulty moving applications from one environment to another. You may need to move an enterprise application from your data center to the public cloud, but the effort necessary to refactor the application can make it impossible to justify. This is where containers come into play. By encapsulating all of an application's dependencies, containers make applications much more portable.

A container can move from a developer's laptop to QA to production—or from one cloud environment to another—without requiring any changes to the container, and without hardware and software reconfigurations in the target environment. For developers, this translates to greater agility and efficiency with less effort. You spend less time reworking existing code to run in new environments and more time on new applications and features.

In large part, the shift to containers is being driven by developers at a grassroots level, and containers are becoming an essential part of cloud native development, microservices architecture and DevOps.

A Kubernetes Primer

Container environments tend to 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. These features enable a set of Controllers that each implement specific capabilities and work together to produce the end result. Kubernetes components can be ripped out and replaced to extend the system and adapt it to new requirements and environments.

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.

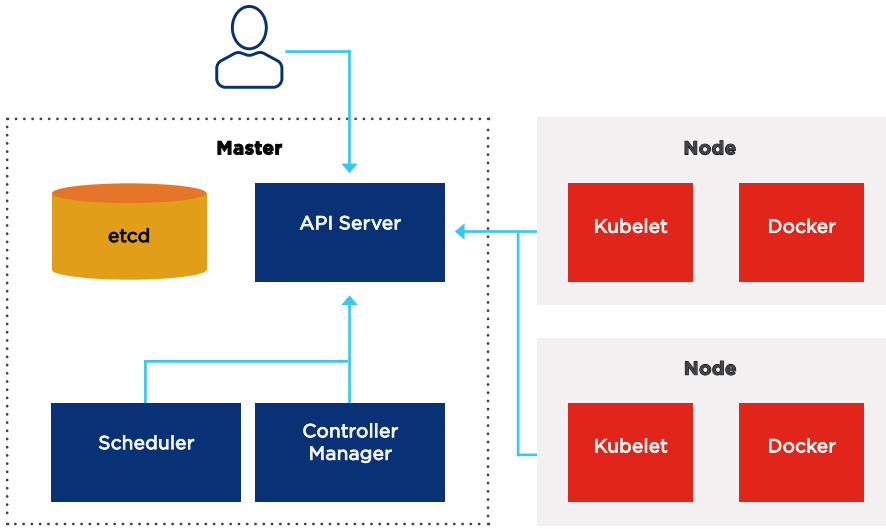


Figure 1. The parts of a typical Kubernetes system

Figure 1 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.

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 the desired state. Each Controller monitors its configuration, stored as a resource 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 selfhealing. 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.

Operators tailor Kubernetes to application needs

Kubernetes Custom Resource Definitions (CRD) provide a way to extend the resources the API Server can manage. CRD is usually paired with a custom Controller called an Operator.

Operators allow you to 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. By automating application-specific tasks that otherwise have to be done manually, Operators allow you to more easily deploy and manage applications on Kubernetes. The opensource Operator Framework provides the necessary tools to facilitate Operator creation.

Why do developers like Kubernetes?

Resilience. Kubernetes is designed to be inherently resilient. You declare the desired state, and Kubernetes works in the background to maintain that state and recover from failures.

Efficiency. Kubernetes makes your team more efficient. Once you get through your first project, building and piecing together applications will become comparatively easy for your team, enabling you to learn through trial and error.

Repeatability. With containers and Kubernetes, you can ensure that the application configuration running in one cluster is identical to the configuration running in another. Kubernetes makes it possible to run instances of the same application across multiple environments with minimum effort.

Flexibility. Almost any type of application can be run inside a container regardless of the language it's written in. It's easy to switch between programming frameworks and deployment platforms.

Visibility. You can increase observability and gain greater insight into how an application can be improved.

Building block approach. With Kubernetes, you can package, platforms, systems, and applications into reusable building blocks. It's an easy solution for making development, testing, and production environments consistent.

Kubernetes: A developer's checklist

If you're a developer learning about Kubernetes, there are three things you're probably most concerned with. The API Server exposes a RESTful interface and provides the services necessary in a distributed system.

- How do I develop for Kubernetes?
- How do I create a build pipeline?
- How do I manage a Kubernetes cluster once my application is in production?

This section serves as a checklist for how to get started in these "big three" areas.

How to develop for Kubernetes

Developing for Kubernetes is conceptually simple: You need a way to develop your application, containerize it and then run it on Kubernetes. Each of these steps is easy to understand, and if you're motivated, you can accomplish all of the steps yourself by writing a Bash script, creating a makefile, etc. Many organizations have done it this way.

However, there are a number of open-source tools that make developing for Kubernetes easier and more efficient. These tools automate the local development workflow so that you can code and test applications more quickly, either on your local machine or, in some cases, on a remote development cluster. Consider choosing one of these tools as your entry point for developing on Kubernetes.

These are not the only tools available for Kubernetes development, but these three are a good place to start.

Draft	**Website**	**Docs**	**GitHub**	Developer: Azure
Skaffold	**Website**	**Docs**	**GitHub**	Developer: Google
Garden	**Website**	**Docs**	**GitHub**	Developer: Garden

Kubernetes advantages for developers

- More productive and happier development teams
- Fewer impediments to development and deployment
- Greater velocity

Learn More

The Ultimate Guide for Local Development on Kubernetes: Draft vs. Skaffold vs. Garden.io

Guide to Cloud Native DevOps

How to create and build pipeline

If you've been a software developer for a while, especially as part of an agile development team, you're probably familiar with the concepts of continuous integration and continuous delivery (CI/CD). The basic idea is that CI/CD software creates an automated pipeline to integrate, test and deploy code changes.

A recent survey of developers highlighted the importance of automation and CI/CD for organizations as they move to a cloud native, microservices architecture. However, the survey also found a very low level of automation for the CI/CD process in most companies. Almost 40% of respondents reported that less than 10% of the process was automated and lack of automation was a major inhibitor to faster code deployment.

CI/CD tools are a key part of developing software to run on Kubernetes. Existing CI/CD tools like Jenkins can also be used in Kubernetes environments, and there are newer tools emerging specifically for Kubernetes.

Kubernetes advantages for developers

Jenkins: The most widely deployed CI/CD tool; a Kubernetes plugin supports CI/CD for Kubernetes	Website Docs Download
Bazel: A general-purpose, open-source tool for building and testing software, developed and used by Google internally	Website Docs GitHub

Kubernetes advantages for developers

Jenkins X: A more opinionated version of Jenkins specifically for Kubernetes	Website Docs GitHub
Travis CI: Simple, cloud-based CI/CD	Website Docs
Flux: CI/CD specifically for **git** version control repositories	Website Docs GitHub

How to manage a Kubernetes cluster in production

A final thing that software engineers need to understand is how to exert control over an application running on a Kubernetes cluster in production. The importance of this can depend somewhat on how your team is organized, but as a rule, developers should understand the basics and be able to define how an application is deployed and managed.

As you saw earlier, Kubernetes enables you to create custom Operators that encapsulate the logic necessary for important application management operations, like deploying application instances, scaling the application, upgrading and so on. Depending on the application, an Operator is something you may want to consider investing effort in. You may also find that Operators have already been created for services you may use in conjunction with your application, such as service meshes and databases.

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 optional, additive functionality on top of core Kubernetes. VMware is actively contributing to the development of a Kubernetes Cluster API.

One of the goals is to provide a way to let developers 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, Kubernetes automatically spins 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**.

The right amount of visibility and control

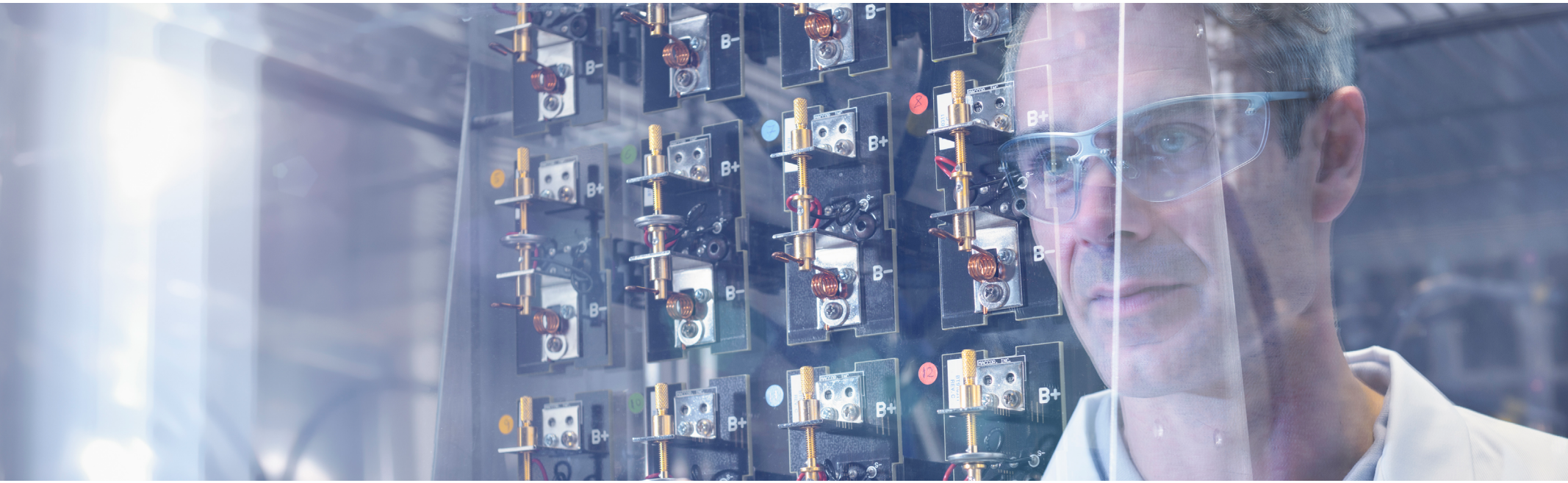
If your organization is going cloud native, visibility and control are essential. VMware surveyed almost 400 IT and technology decision makers and found:

58% wanted greater visibility into performance[1]

65% needed better access to audit logs and forensics[1]

The ability to orchestrate efforts across multiple clouds is the number one driver for Kubernetes and that requires visibility and control across teams and clusters.

What parts of your application environment are currently opaque? Keep visibility in mind as you choose K8s tooling.



[1] VMware customer data

Migrating applications to Kubernetes

For many of us, one of the first projects we're asked to take on is moving a large enterprise application to Kubernetes. Every enterprise has monolithic applications, often written in Java or PHP, that are hard to maintain and manage. The first question you have to ask yourself is whether to move the application at all. (See sidebar.) Assuming the answer is yes, the process proceeds through a number of stages: application audit, containerizing your app, breaking your app into pieces, migration and management.

Application audit

Once you've made the decision to migrate, the first step is a careful audit of the application to make sure you know what you're dealing with. (You may learn things that make you rethink your migration decision.)

Here are the important questions to ask:

- **What are the dependencies for this application?** The list of things that will break your monolithic application if they suddenly go away can often be quite surprising. Can all the necessary pieces be migrated?
- **Does the application have configuration files?** Where are they kept? Are they stored on the system? How do they get changed?
- **Does your application make assumptions about the OS and hardware it's running on?** Your existing software may include hardcoded assumptions about underlying hardware and software that aren't documented.

Containerizing your application

There are a lot of tools that can help with the process of containerizing an existing application. Several of these were described in the Checklist Section. Tools to create a CI/CD pipeline are described there as well.

Many monolithic apps include Java code. Before Java SE 10, which came out in 2018, containerizing Java code was problematic. Running Java inside a container is no longer a problem—one less thing to worry about.

Should I move my application to Kubernetes?

The decision to move an application comes down to three factors: value, risk and time.

The **value** of Kubernetes for development teams is the subject of this book, but it's up to you to evaluate each application to understand the potential benefits for that application.

Migrating an application to a new platform always entails technical **risk** and cost. Kubernetes is young as IT technologies go, as are the associated tools.

The migration effort can take significant **time** on the part of developers and operators.

Before you jump in, look for ways to concretely measure the gained value, understand the amount of risk and determine how much time you can afford to spend.

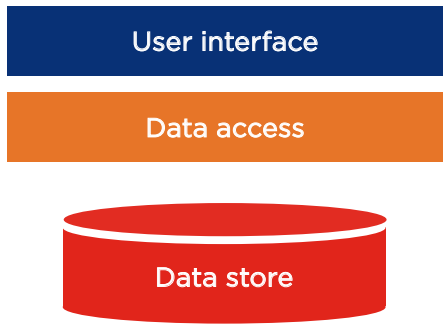
Breaking your application into pieces

Of course, it may be possible to run your monolithic application on Kubernetes in a single container, and you may start out that way, but the real goal is usually to decompose your application into logical pieces (microservices) that fit nicely into Kubernetes and allow you to parallelize continuing development activities with more clearly defined areas of responsibility.

Every application is different, and only you can decide where the dividing lines should be drawn. One guideline is to consider the network as the new application interface. Any time an application transfers large, complete data structures, that's a good indicator of a possible break point.

If you're just not sure where to begin breaking up your application, the big three application components—user interface, data access layer, and data store—are always good starting points.

If your application relies on a back-end database as its data store—maybe SQL Server or Oracle running in a VM or on bare metal—these databases can now be containerized. There is also a variety of open-source alternatives that are amenable to containerization.



Migration

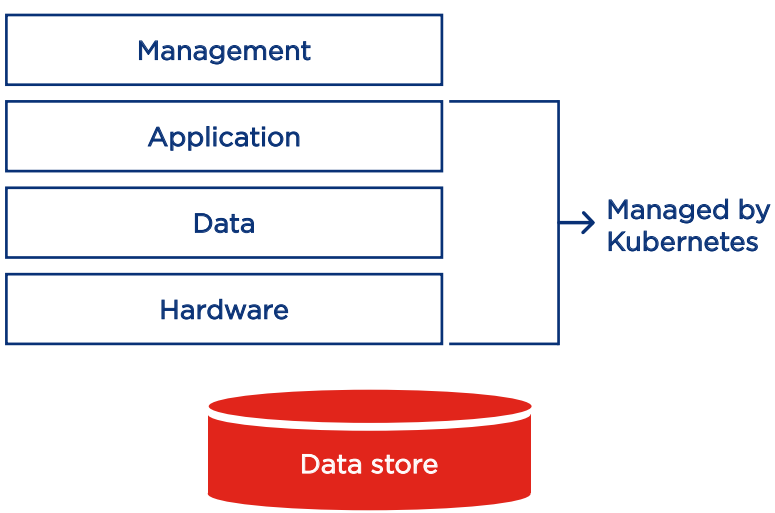
Migrating an application to Kubernetes is in many respects the same as every other application migration you've ever been involved with. You have to consider how much downtime will occur during the migration and protect against data loss. There are almost always unforeseen problems that arise. Then you have to decide what the whole migration process will look like. Do you do an A/B deployment? A DNS rollover? All those decisions have to be made and orchestrated by humans.

One of the biggest concerns when you're migrating a monolithic application is migrating state information. One option is to leave the state information alone. If you're dealing with a back-end database, you may decide to leave it as is. The drawback with that is it can leave you straddling two worlds, leaving you with a fragmented system. An alternative, as mentioned in the previous section, is to containerize the database as well so that everything you depend on runs under Kubernetes. Naturally, you may decide to migrate the front-end application first and tackle the database later.

Management

A final important consideration is how you will manage your application stack once it's running under Kubernetes. As you've already learned, Operators are becoming a preferred way for managing domain-specific knowledge to simplify managing your application under Kubernetes, but that may represent a significant effort in terms of learning and coding.

Whether you use an Operator or not, it's important to note that the Kubernetes environment is likely to be significantly different than the environment your application came from. That means things like backup, logging and monitoring are all going to be difficult. Plenty of open-source solutions exist in all these areas, and Kubernetes has the added advantage of being built from the core for resiliency with **liveness and readiness probes** and other health checks.



How to get started

The good news is, if you're migrating a monolithic application today, lots of people have gone before you. Do everything you can to learn from their mistakes—instead of making them yourself. There are plenty of good writeups on ways teams have succeeded—and failed. For example, **GitHub has a fairly detailed blog** on its approach to migrating the applications that run github.com and api.github.com. A little searching is likely to turn up examples that are relevant to your situation.

Best Practices for Cloud Native Apps

The great thing about taking on the migration of a legacy application as one of your first Kubernetes projects is that it can really clarify your thinking about what not to do when it comes to developing cloud native applications from scratch. That hard-won knowledge may serve you well. Either way, here are some additional guidelines to help you get your greenfield efforts off the ground with minimal pain.

Find the right tools

Taking the time to identify the right tools will help you get started on the right foot. The earlier Developer's Checklist introduced three Kubernetes development tools: Draft, Skaffold, and Garden. Choosing one of these or a similar Kubernetes-focused tool will help you start out on the right foot and avoid wasting too much time while you gain familiarity with Kubernetes.

Use the patterns

Kubernetes application design patterns are different than for other environments. If you're architecting applications for Kubernetes, it's extremely useful to understand and use established patterns whenever possible. This **Usenix paper** from Kubernetes co-founder Brendan Burns introduces many of the common patterns.

Take full advantage of operators

If the application you're building is stateful, you should strongly consider whether you need to create an operator. To refresh your memory, Operators encapsulate the application-specific logic necessary for data protection, high availability, and other management functions. As you think about Operators, remember that open-source Operators already exist for common databases and other services you may be using with your application. You may be able to use these as is or modify them to suit your needs..

Engage with the community

If you're only going to follow one best practice, it should be this one: **engage with the Kubernetes community**. One of the things that differentiates Kubernetes from many other open-source projects is the vibrancy of the community around it. No matter what problem you're trying to solve, chances are good that there is someone out there who's either already solved it or is working to solve the same challenge.

What Should I Do Next?

If you're a developer starting out with Kubernetes, the most important thing is to get started. Use the resources links in this eBook to learn more, watch videos, and **engage with your peers** online. For in-person opportunities to connect, find local **Kubernetes meetups** in your area.

Consider downloading one of the development tools mentioned earlier and kick the tires, or install **minikube** on your local machine to get more familiar with Kubernetes. In addition, there are a variety of resources for everyone on the Kubernetes journey:

Cloud Native Apps Blog
Read our regular **blog** to find out the latest. Posts cover diverse topics and new blogs are posted 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.

Kube 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.

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

Need help deciding how Tanzu Kubernetes is the right solution for you?

Explore Lumen solutions or schedule an expert consultation at **https://www.lumen.com/en-us/hybrid-it-cloud/private-cloud/vmware-tanzu.html**