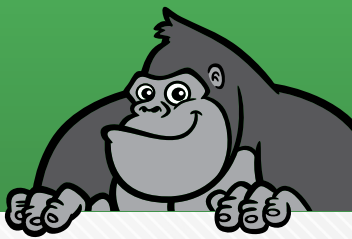
ActualTech Media

PLATFORM9

The Gorilla Guide to...®

Express Edition



Production-Grade Kubernetes

Inside the Guide

- Deploy Kubernetes the Right Way
- Leveraging Kubernetes for Great DevOps
- Best Practices for Using Kubernetes

THE GORILLA GUIDE TO...

Production-Grade Kubernetes

Express Edition

ACTUALTECH MEDIA

6650 Rivers Ave Ste 105 #22489
North Charleston, SC 29406-4829
www.actualtechmedia.com

PUBLISHER'S ACKNOWLEDGEMENTS

EDITOR
Keith Ward, ActualTech Media

PROJECT MANAGER
Wendy Hernandez, ActualTech Media

EXECUTIVE EDITOR
James Green, ActualTech Media

LAYOUT AND DESIGN
Olivia Thomson, ActualTech Media

WITH SPECIAL CONTRIBUTIONS FROM
Kamesh Pemmaraju, Bic Le, Roopak Parikh

TABLE OF CONTENTS

CALLOUTS USED IN THIS BOOK

The Gorilla is the professorial sort that enjoys helping people learn. In the School House callout, you'll gain insight into topics that may be outside the main subject but are still important.

This is a special place where you can learn a bit more about ancillary topics presented in the book.

When we have a great thought, we express them through a series of grunts in the Bright Idea section.

Takes you into the deep, dark depths of a particular topic.

Discusses items of strategic interest to business leaders.

ICONS USED IN THIS BOOK

DEFINITION

Defines a word, phrase, or concept.

KNOWLEDGE CHECK

Tests your knowledge of what you've read.

PAY ATTENTION

We want to make sure you see this!

GPS

We'll help you navigate your knowledge to the right place.

WATCH OUT!

Make sure you read this so you don't make a critical error!

TIP

A helpful piece of advice based on what you've read.

Getting Hold of the Kubernetes Curve

In the early days of containers, it became immediately obvious that some kind of orchestration tool would be necessary. Given how easily containers could be created and destroyed, and how many of them would be used, management of this process was key.

Many contenders came and went, but eventually one technology emerged from the fray and was adopted by the industry: Kubernetes.

But just because Kubernetes became the standard, it doesn't mean that it's a simple plug-and-play tool. While powerful (and continuing to grow in scope and power), it has a steep learning curve.

This Gorilla Guide To...® (Express Edition) Production-Grade Kubernetes can be an important part of that curve. In this book, you'll get a solid introduction to many of the aspects of deploying and managing Kubernetes in production.

The content in this book is geared toward the practical. There's information and advice for getting the most out of Kubernetes, including best practices, and what you need to know to scale up, keep it secure, and more. We skip most of the theoretical aspects of Kubernetes in favor of how to use it day to day.

We start with some pre-Kubernetes considerations, those things you'll have to decide on before deploying it in your environment. Let's dive in!

CHAPTER 1

Considerations for Distributed Kubernetes—from the Data Center to the Edge

Kubernetes is widely recognized as the platform of choice for running efficient, distributed, containerized applications. It's also common to think of Kubernetes in terms of a single, large cluster or set of clusters running in a data center. This is certainly a common deployment approach, but it's not the only one.

Variety of Deployment Models

Kubernetes can be deployed in many kinds of environments. The platform is well-suited to run in micro data centers that are closer to the edge. A branch office may only need a small cluster to support the remote operations of an office. This kind of use case can typically run components that fit into a single rack. Kubernetes can also run at point-of-presence sites. For example, retailers may deploy Kubernetes clusters to physical stores and distribution centers to run applications, store data locally, and coordinate operations with centralized processes.

Kubernetes may also run at edge locations to support Internet of Things (IoT) systems. A manufacturer may deploy Kubernetes in multiple locations within a manufacturing facility to collect IoT data and perform preliminary processing and analysis. This kind of processing close to the environment can help compensate for unreliable networks and long latencies that can reduce the effectiveness of highly centralized processing.

Platform9, a leading managed Kubernetes vendor, has produced a webinar[1] that provides an overview of Kubernetes use cases that includes cloud-native apps, hybrid clouds, and edge computing scenarios.

[1] https://platform9.com/resource/scaling-kubernetes-reliably-at-the-edge/

It's clear there's a spectrum of cluster deployments. When you're considering and planning your Kubernetes strategy, it's important to understand where your deployment falls on that spectrum because there are requirements particular to each. A data center cluster, for example, may have ample resources to scale up the number of pods in a deployment, while a micro data center is more constrained.

In the case of Kubernetes deployed at the edge, you should consider how continuous integration/continuous deployment (CI/CD) will work with potentially unreliable networking. The number of sites can quickly become a factor you need to consider. Updating a single cluster in a data center is challenging enough—updating hundreds of point-of-presence sites is even more difficult.

Network Issues and Multiple Kubernetes Sites

When deploying Kubernetes clusters to multiple data centers and remote sites, the quality and capacity of network infrastructure can impact the overall performance of the platform.

Data centers typically have high-bandwidth connectivity. Clusters are composed of servers with high-speed network connections between them and run in an environment with multiple racks. The combination of high-bandwidth networking and the ability to distribute pods over multiple racks provides the optimal environment for performant and reliable Kubernetes clusters.

That level of network capacity extends beyond single data centers, too. Hybrid clouds composed of resources in a data center and in one or more public clouds can

have high bandwidth dedicated direct connections between sites.

Micro data centers and point-of-presence deployments typically won't have the same network bandwidth available in data centers and within hybrid clouds. Edge processors and IoT devices are even more constrained in terms of bandwidth. This is one of the reasons it's advantageous to deploy Kubernetes to multiple locations—with remotely deployed clusters, the processing is brought close to where the data is being generated. Local processing reduces the amount of data that must be sent to the data center and gives local sites the ability to function autonomously in the event of a network outage.

This highlights another factor to consider when planning your Kubernetes strategy: There may be periods of extended outage. Short outages in well-architected deployments won't significantly adversely affect operations. Longer outages, however, will cause different clusters to get out of sync. Changes to data will accumulate in the clusters that are isolated by the network outage and when connectivity is restored, recovery can begin and data can be synced. Depending on the duration of the network outage, the recovery may be long enough to impact performance and service delivery.

Local Data Processing

The ability to process data locally is a key advantage of having multiple Kubernetes deployments. This approach, however, does make it more difficult to deploy services to multiple clusters. Consider, for example, the various use cases for distributed Kubernetes.

A retailer may deploy Kubernetes to a centralized server in a store, as well as to point-of-sale systems. The centralized server could collect data from point-of-sale systems, generate real-time reports and dashboards for local managers, as well as coordinate services running in a corporate data center or cloud.

5G is changing how businesses deliver services and collect data. With significantly more bandwidth than previous generation networks, 5G enables more data-intensive applications. To achieve the higher bandwidths, 5G networks use higher frequency signals. The disadvantage of this is that 5G networks need more cell towers because the signal degrades over long distances. Those cell towers all have to run networking services, so managing the deployment of software is a significant challenge for carriers. Distributed Kubernetes can help here, as well. Networking services can be deployed in containers and updated as needed from a central location (see **Figure 1**).

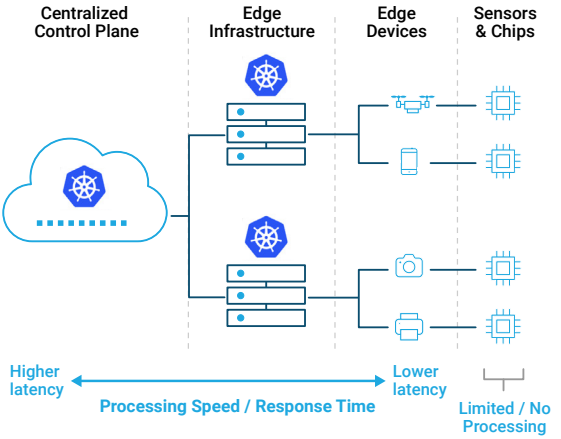


Figure 1: The central management of data centers and edge locations

IoT devices can also require local data processing. Machine learning models for analyzing images or controlling autonomous vehicles are best run locally to avoid unnecessary latency introduced by centralized processing. Distributed Kubernetes can again mitigate the challenges of managing software deployed on thousands of geographically distributed devices.

These three examples share some common requirements. They all need consistent and reliable methods to update software. In addition, these update methods

have to be essentially zero-touch and automated in order to scale.

Stateful services, such as databases, bring another set of challenges to managing multiple Kubernetes clusters. These services need persistent storage, so you'll need to understand how to architect the cluster to deliver the needed read and write performance. To enable some level of autonomy within the cluster, plan for graceful degradation of services when the network is down.

For example, data stored on a remote cluster could be cached locally so that data is available to local processes. When the network is available, the databases can sync and caches can refresh.

Security Considerations

Security operations need to be coordinated across all environments—especially encryption for data at rest and key management.

Encryption at rest is required to comply with a wide variety of regulations, especially when personally identifying information (PII) or other sensitive data is stored. There may be multiple levels of encryption, starting with the storage device.

Middleware, such as databases, may also provide for encryption. For example, some relational database management systems allow data modelers to specify that particular columns of data should be encrypted. Applications can also provide for their own encryption policies and methods. Regardless of the combination of encryption options you may employ, they need to be coordinated across all Kubernetes environments.

Key management is another security process that will need to be managed across environments. Key management services can provide all the required functionality, but you'll still need to define policies and monitor operations. For example, you'll want to define policies for key rotation and be able to verify the operation occurs.

Centralized Management of Multiple Environments

Multiple environments can be a challenge to manage, especially as the number of sites grows. Some clusters will be in the data center and can be managed to some degree with existing tools. Clusters at point-of-presence sites and on the edge need to be monitored and managed to maintain the necessary quality of service.

Fortunately, Kubernetes has auto-healing capabilities that reduce the need for human intervention. Unhealthy pods are replaced automatically without requiring a DevOps engineer to log into a cluster, identify the failing pods, and replace them. Auto-healing also promotes autonomy—if the network is down, the cluster can continue to function and correct for some failure within the system.

Focus on Your Core Business Objectives

Kubernetes is moving beyond the data center to micro data centers, point-of-presence facilities, and even the edge. Managing Kubernetes is difficult when it's isolated to a data center, but multiple deployments in different environments compound your management challenges.

How to respond to those challenges is the subject of Chapter 2.

CHAPTER 2

Creating an Optimal DevOps Experience with Distributed Kubernetes

Kubernetes is widely recognized as a platform that enables highly efficient use of infrastructure, but organizations need to understand those benefits are maximized when the developer experience itself is optimized.

Developers are increasingly assuming responsibilities for systems operations. In the past, it was common to have a separate team of systems administrators responsible for deploying applications, monitoring resource use, and responding to incidents that disrupted services. Developers who use agile methodologies are more likely to employ practices that include responsibility for ensuring their software operates efficiently and reliably.

This is understandable, since one aspect of agile engineering practices is the frequent release of new versions of services. Rather than hold up the release of an

update so that multiple features can be included, it's more efficient to release small changes continually.

CI/CD pipelines, coupled with version control platforms that promote collaboration, enable this kind of rapid release of new features. It also means that the developers who are working in the code and revising it are in the best position to understand the cause of performance or reliability problems.

Platform Engineering Optimal Experience

Developers depend on a stable environment to work. This entails high uptime, reliability, and performance. Platform engineering teams should treat the platform as a product. They provide this platform for developers, enabling them to create services for their customers.

This includes building teams, processes, and a culture that continually improves—not just sustains—the platform. Using agile approaches, developers can deliver initial applications on a platform they manage—but expect to have platform engineers take over responsibility for the platform.

Kubernetes can help deliver the optimal engineering experience. It's designed to automate and orchestrate reliable computing resources for containerized

applications. One important aspect of Kubernetes is that it can be deployed in multiple environments, including:

- Centralized data centers, either on-premises or co-located in a third-party data center
- Micro data centers used in remote offices
- Point-of-presence locations such as retail stores
- Edge computing settings for Internet of Things (IoT) deployments

The ability to deploy Kubernetes to a wide variety of environments is a significant advantage over deploying customized, case-specific servers. With a single, common platform for executing workloads, developers can spend less time on operational issues with the help of tooling that supports the Kubernetes platform.

Consistent Policies, Practices, and Tools

Consider the challenges of complying with regulations and policies while maintaining an agile, rapid-feature-delivery engineering environment. There are

multiple dimensions of compliance that must be attended to.

For example, developers, who are also operations managers, need tools to help ensure authentication mechanisms are in place. In many cases, authentication and identity management services are provided by a centralized service that needs to be accessible from various Kubernetes deployments.

Highly distributed systems like Kubernetes are constantly generating, storing, and transmitting data. Many regulations governing privacy and the control of sensitive information have rules about protecting the confidentiality of data. To meet these requirements, it's a best practice to employ encryption for data in motion and for data at rest.

Kubernetes environments should be deployed in ways that provide these encryption services by default. Application developers shouldn't have to learn the intricacies of configuring full disk encryption or setting up TLS connections between nodes. Role based access controls (RBACs) are essential for securing the platform. Given the large number of services and tenants, this can be a difficult task and requires tooling to support and maintain proper RBAC configurations.

Kubernetes should be deployed with controls in place to support other governance requirements. For example, security scans should be configured to run reliably on all clusters. Again, this is a necessary capability, but not one that should require significant developer time.

Tooling should be in place to help with capacity planning and cost control. Kubernetes is designed to allocate resources to workloads that need them. Those resource demands can, and often do, change over time, so it's important to monitor resource utilization and growth rates in workloads. If a cluster has insufficient resources, developers may be forced to limit features or find other workarounds to deal with the lack of capacity. Poor capacity planning can introduce significant friction in the development process and slow the creation of new services.

Organizations are increasingly adopting multi-cloud platforms, so you'll need to consider integration of different systems. Legacy on-premises applications and servers may be used alongside servers running in a public cloud, for instance. Kubernetes is well suited to these kinds of deployment models, but there must be tooling in place to maintain the reliability of these systems.

The Negative Impact of Shadow IT

When appropriate tooling isn't in place and there's insufficient centralized support, developers will likely develop their own solutions to operational challenges. For example, when platform tools like CI/CD pipelines aren't centrally standardized, departments or teams of engineers may implement their own solutions.

"Shadow IT" refers to any IT asset—hardware, software, applications—that a user downloads and uses without the organization's knowledge. It's becoming more prevalent in the cloud era, as there is more access to more technology.

Some of the biggest problems this causes for IT include security and compliance issues.

Some estimates of the damage that Shadow IT causes put it in the trillions of dollars every year. That makes it crucial for companies to control and eliminate as much Shadow IT as possible.

This is problematic for several reasons. For one, it's inefficient to have multiple teams duplicating work. It

also means that individual teams are responsible for maintaining tools and ensuring they're deployed in compliance with policies and regulations. They also become responsible for ensuring that all service-level agreements (SLAs) are being met.

These kinds of shadow IT practices lead to inconsistent management practices. Instead of a common operations model, organizations are left with a fractured DevOps situation that makes it more difficult for teams to collaborate. Teams will develop different procedures and use different tools, and this often means each team takes on learning on its own and may not benefit from what others have experienced.

Clearly, a consistent set of policies, practices, and tools across an organization is essential to maintaining an optimal developer experience. It's also important to consider what might be required for an application owner's optimal experience.

The Application Owner's Optimal Experience

Application owners have an obvious stake in ensuring an optimal developer environment. Key considerations from their perspective include:

- Ensuring developers have needed resources

- Standardizing on commonly needed resources and middleware
- Using tools to streamline package management within Kubernetes deployments

Key Resources for Developers

Key resources for developers span the development cycle. There should be support for full stack development. UI developers typically work with frameworks for creating complex Web interfaces, while back-end developers are more likely to need tools to help optimize high-performance code.

Tooling should also include support for version control and CI/CD. These tools are becoming more feature-rich and integrated so that as soon as changes are checked into a repository, they can trigger a build, with testing and eventual release to follow.

Service discovery and application catalogs are important for ensuring developers know the kinds of services available in the environment. These tools can foster the sharing of services and reduce redundant code.

Standardizing Commonly Needed Resources

Application owners should also consider standardizing commonly needed resources and middleware. For example, multiple services may need a relational database back end. There are many high-quality options to choose from, including both open source and commercial products.

While different relational databases have distinct features and capabilities, application owners must ask if the cost of supporting two or more databases is outweighed by the benefit of those specialized features. In many cases, the economics favor standardizing on a single kind of database.

It's also important to make shared components available in a central catalog that are available for developers to easily deploy with a few clicks. This provides the governance that the operations teams need and the self-service agile experience that developers crave.

Streamlining Package Management

Similarly, organizations should standardize on load balancers and monitoring tools. While different load balancers may have different features, the core job of a load balancer isn't likely to vary much among services running in the same environment.

A single, consolidated monitoring tool should be selected as well, with performance metrics collected in a single tool. This allows for more comprehensive analysis of performance monitoring data than if the data were spread across multiple tools.

Logging and distributed tracing tools are also important for understanding the state of your systems, identifying bottlenecks, and understanding the root causes of performance problems.

Service meshes, like Istio, provide additional services on top of Kubernetes (see **Figure 2**). Standardizing on a single service mesh across all deployments of

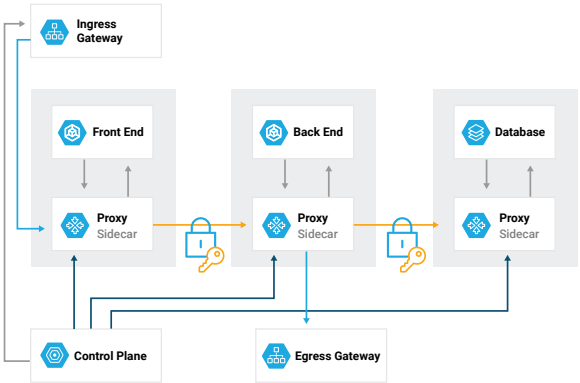


Figure 2: Service mesh traffic overview

Kubernetes will also improve the overall utility of Kubernetes from a developer and application owner perspective.

For all of the benefits of Kubernetes, there are some challenges to using the platform. Within a single cluster, dozens of packages may be deployed, all of which must be monitored and maintained along with other applications. It can be a challenge to keep track of packages and their state in a single cluster, but the workload is multiplied when you include Kubernetes deployments in distributed and edge computing environments.

Automation is required to support package management. Fortunately, Helm and Kustomize are two such package managers that can streamline package management.

The promise of Kubernetes to more efficiently employ computing and storage resources is best realized when you take into account how Kubernetes is used and maintained by developers. Kubernetes is complex, and as responsibility for managing clusters moves from a small number of clusters in a single data center to hundreds or thousands of distributed clusters, there's a risk of not knowing how to run such a distributed platform optimally. That's where Chapter 3 comes in.

CHAPTER 3

10 Considerations for Running Kubernetes at Scale

Kubernetes is designed to scale to distributed computing platforms far larger than the systems many enterprises use. Moreover, when you deploy thousands of microservices over a large number of geographically distributed servers that need to be available virtually all of the time, operating that platform becomes increasingly complicated. Before you enter the world of large-scale Kubernetes deployments, here are 10 considerations to keep in mind as you plan your system.

Scalability

Many engineers start working with Kubernetes by using small clusters. A set of five nodes is sufficient to work with Kubernetes services, get to know the commands, and practice basic operations, like deploying new versions of services and creating persistent storage volumes.

While this scale is well-suited for learning about Kubernetes or supporting a small set of applications, it

won't reveal the issues you're likely to encounter when you start running hundreds of nodes in a cluster.

One of the issues with large deployments is scaling the number of pods in a deployment or nodes in a cluster. In the case of a small five-node cluster, if the workload increases by 20%, you can manually add another node to the cluster. You could keep the cluster at the increased size or reduce the number of servers sometime in the future when the load decreases. The disadvantage of this approach is obvious. Manual intervention to scale resources isn't a viable option when working with large deployments and dynamic workloads.

Kubernetes employs autoscaling to adjust the number of nodes in a cluster. As the demands for computing resources change, the autoscaler can increase or decrease the number of nodes. When nodes in the cluster are running at high CPU utilization for extended periods, the autoscaler will add nodes. Similarly, if nodes become idle for some period of time, they're removed from the cluster. Adjusting the number of nodes in a cluster is referred to as horizontal scaling.

Another way to scale is to use servers with more resources. For example, instead of deploying nodes with 16 CPUs and 96GB of memory, you could use nodes with 64 CPUs and 400GB of memory. This is called vertical scaling (see **Figure 3**).

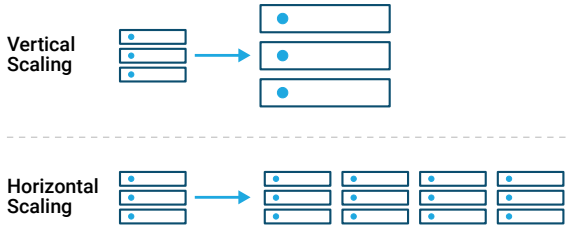


Figure 3: Horizontal vs. Vertical scaling of nodes

Scaling is an important consideration because it directly impacts the availability of services. A resource-constrained cluster doesn't have the capacity to process additional workloads. Over-provisioning is an option, but it's a costly one. A better approach is to ensure you've instrumented the cluster so you can collect metrics about its state and automatically respond to changing workloads.

Availability

The formal definition of availability is the percentage of time a system is ready for use. This way of thinking about availability is useful when working with service-level agreements (SLAs). It's also an appropriate way to think of availability from a user's perspective—a system is available if they can use it. A developer's perspective is slightly different.

Developers have a more expansive view of availability. It includes ensuring a production environment is functioning and able to meet the workload on the system at any time. Developers also depend on development and test environments being available to do their work. To ensure developers have the necessary environments available to them, it's important to create repeatable processes for deploying clusters and services.

The repeatable processes for developer environments may be different from the repeatable processes used in production environments. Site reliability engineers (SREs), for example, may have a specific set of design principles they apply to production environments.

For example, there may be different levels of health checking, monitoring, and alerting. Service-level agreements (SLAs) will likely be different, as well. Also, developers will likely have different needs from SREs. For example, developers shouldn't have administrative access to a production cluster, but they should have administrative privileges to a cluster in their development environment, rather than depend on others to configure and maintain it.

Upgradability

Kubernetes is under active development. To ensure you have access to the latest features, you need to plan for upgrading clusters. It's easy to begin working with Kubernetes and even run production workloads without thinking about how you'll upgrade the cluster.

Consider a typical scenario of how an enterprise might start using Kubernetes. A group of developers and a business sponsor decide to develop a proof of concept (PoC) system on a small cluster. The developers want to show results as fast as possible, so they choose the easiest installation method to get Kubernetes up and running.

Next, they incrementally add other services, such as a database, which increases complexity to the overall system. Wanting to show a realistic use case, the developers then deploy an application. The PoC is well received and decision-makers agree to make the service available in production.

Now, the developers of the PoC are faced with operationalizing a system that wasn't designed for the demands of a production environment. They'll have to install monitoring and logging tools. Of course, the business application running in the cluster will need to be updated, so they'll have to integrate with a CI/CD

platform. As you can see, decisions from choosing an installation method to integrating with a CI/CD platform can't be made in isolation.

To ensure upgradability, plan for it from the start of a Kubernetes project. Often upgrades can lead to downtime if not planned carefully. For high SLA and mission-critical applications, upgrades need to be designed to avoid downtime, which is exceptionally difficult without appropriate safeguards. Platform9 has a beneficial blog post[1] that can help pave the way to successful Kubernetes upgrades.

[1] https://platform9.com/blog/kubernetes-upgrade-the-definitive-guide-to-do-it-yourself/

This process continues with even more tools added to the cluster, which essentially grows organically and incrementally according to emerging requirements. This is unfortunate. To ensure a cluster is upgradeable, organizations should plan for full lifecycle development. This is challenging when working with Kubernetes, however, because most organizations don't have teams of experts, and often have far fewer Kubernetes experts than needed. As a result, production systems are

difficult to upgrade and, rather than risk disrupting services because of an issue updating the platform, enterprises continue to run older versions of Kubernetes.

With a properly established CD pipeline that allows for rollbacks, Blue-Green and Canary deployments, enterprises can be more confident in upgrading to newer versions more frequently. This helps avoid running significantly out-of-date versions of the platform.

Observability

The more complex a system becomes, the more important it is to be able to determine the state of that system at any time. Observability is the term for this. Usually, when developers talk about observability, they're referring to collecting metrics, logs, and distributed traces from servers and processes. These types of information are essential for diagnosing and correcting problems.

For example, a pod in a Kubernetes cluster may be constantly restarting. How would someone go about troubleshooting this? They might look into problems with the cluster, like the loss of a quorum or a problem on a single node, such as no free disk space—and this problem is compounded when dealing with multiple clusters running in different locations and clouds.

There are many possible contributing factors to problems with cluster operations. Curated dashboards showing key metrics can help developers and SREs focus on the most important pieces of information.

Given the overwhelming number of metrics and logs that could be observed, it helps to have experts identify which to include in your dashboard. In fact, this principle applies to all of the considerations outlined here.

Performance

When planning for Kubernetes at scale, consider how you'll maintain appropriate levels of performance. Specifically, is your system able to meet compute, storage, and network needs at any point in time? Think about performance at both an application and a cluster level.

At the application level, deployments should be performant. Deployments consist of multiple pods, so pods need to be performant for the deployment to be performant. Of course, with a sufficient number of pods, the deployment can continue to meet the needs of workloads even if some small number are not functioning as expected.

At the cluster level, you should consider how to maintain the overall performance of a cluster. This is largely

a factor of how performant the nodes are, but other cluster-level properties, such as how fast a cluster can autoscale, can impact the overall performance of the system.

Working with Images

It's best to use container-optimized images so that Kubernetes can pull them faster and run them more efficiently.

What's meant by being optimized is that they:

- Only contain one application or do one thing
- Have small images, since big images aren't so portable over the network
- Have endpoints for health and readiness checks so that Kubernetes can take action in case of downtime
- Use a container-friendly OS (like Alpine or CoreOS) that make them more resistant to misconfigurations
- Use multistage builds so that only the compiled application (and not the dev sources that come with it) is deployed

Lots of tools and services let you scan and optimize images on the fly. It's important to keep them up to date and security-assessed at all times.

The geographic location of the cluster nodes that Kubernetes manages is closely related to the latency that clients experience. For example, nodes that host pods located in Europe will have faster DNS resolve times and lower latencies for customers in that region.

Reliability

Reliability is a property of a system that's closely related to availability. The formal definition of reliability is a measure that takes into account the mean time between failures and the mean time to recovery.

Reliability in Kubernetes is determined by the ability of the system to provide resources when needed and the ability of systems software to function as expected. The ability to scale resources up is especially important to reliability. Being able to observe the state of a cluster and respond to problems is also a significant factor for maintaining highly reliable clusters and services.

Supportability

Kubernetes clusters, like any complex system, require sufficient support to be maintained properly. Supportability is a measure of how much effort is required to keep clusters and services functioning.

Systems can be available and reliable, but only with human intervention. Kubernetes is designed to minimize the need for that intervention. For example, Kubernetes monitors the status of pods and replaces them automatically when they fail health checks.

In addition to the core Kubernetes components, supportability encompasses other components that may be deployed in a cluster. For example, a cluster that supports the training and use of machine learning models may support Kubeflow, a deployment manager for machine learning. Supportability also needs to extend to services that users will need to use Kubernetes effectively, including Prometheus, Fluentd, Istio, and Jaeger.

When scaling up a Kubernetes cluster, consider how you'll continue to support existing services, as well as additional services that may be needed in the future.

Security

Security is always a consideration when deploying services. As an administrator of Kubernetes clusters, you'll need to attend to multiple security mechanisms, including access controls, encryption, and managing secrets.

Access controls depend on identity management. There must be a way to represent users and service accounts within the cluster. To streamline identity management, users should be assigned to roles or groups that have permissions assigned to them.

You should also consider how you'll enforce the principle of least privilege—granting only the permissions a user needs to perform their job and no more. In addition to these authorization considerations, you'll also need to deploy authentication methods that support the way users employ the cluster.

Platform9 has a useful blog entry that discusses Kubernetes-related security issues in-depth: Kubernetes Security: What (and What Not) to Expect.[1] It includes:

- An architectural overview of components
- Built-in features
- What is *not* secured by default
- Securing at scale

[1] https://platform9.com/blog/kubernetes-security-what-and-what-not-to-expect/

Also, plan for how and when you'll use encryption. Sensitive and confidential information should be encrypted at rest, as well as in transit.

You should plan to provide a mechanism for storing secrets, such as database passwords and API keys. Developers may be used to storing secrets in configuration files and setting environment variables with those secret values, but a centralized repository for managing secrets is more secure.

Compliance

Closely related to security is compliance. As the size of Kubernetes clusters grows, it becomes imperative to define policies that allow you to meet regulatory requirements with minimal manual intervention. Pay particular attention to audit policies and how they're used to demonstrate compliance.

Also, consider where Kubernetes is deployed. Clusters aren't constrained by political boundaries. Know which regulations you must comply with if a cluster is deployed in multiple countries or in states with applicable regulations, such as GDPR in the European Union and the California Consumer Privacy Act in the United States.

Deployability

Kubernetes runs in a variety of environments. Some clusters are deployed on-premises in a data center while others are in a public cloud. Hybrid clouds are common, as well. At the other end of the spectrum, Kubernetes may be deployed to a point-of-presence system, such as those in retail stores. IoT systems can benefit from computing resources at the edge, which can be delivered using Kubernetes.

Consider how you'd deploy updates to Kubernetes in these various environments. Is the process automatable? How much human intervention is required to deploy Kubernetes? If you plan to scale Kubernetes, you should strive for zero-touch automated procedures.

With these considerations in mind, it's time to move onto the more practical aspects of running Kubernetes in production. The next chapter is full of best practices that will help you get the most out of your containerized environment.

CHAPTER 4

Production-Grade Kubernetes: Best Practices Checklist

Kubernetes is rapidly becoming a key element of enterprise IT infrastructure. As with other enterprise platforms, there's a broad array of requirements to keep Kubernetes clusters functioning and running efficiently. Here are several best practices to employ when running Kubernetes in production.

Deployment Best Practices

Kubernetes environments are highly dynamic. Services are deployed and updated frequently. Nodes are added and removed from clusters. Clusters are spun up and down according to workload.

Kubernetes handles much of the management of this lifecycle, but when it comes to deploying services, much of the responsibility rests with IT professionals. To streamline the ability to deploy and maintain services, keep in mind deployment best practices—these ones in particular:

- Ensure open and flexible environments

- Standardize the container build process
- Ensure self-service
- Manage applications and storage

Ensuring Open and Flexible Environments

Kubernetes runs on a variety of computing infrastructure, including commodity servers. Existing hardware can be redeployed to run Kubernetes along with newly procured servers. You have your choice of running Kubernetes on bare metal, virtual machines (VMs), or in public clouds.

If you already have an established VM environment, running Kubernetes in that environment can be a logical choice. If you would rather not maintain physical infrastructure, then running Kubernetes in a public cloud is a good option and one with low barriers to entry.

Kubernetes has also prompted the development of additional open source software that runs on the platform. Tools like Helm for deployment and Istio for service management are open source tools that extend the capabilities of the Kubernetes environment.

Platform9 has an informative blog post[1] about running Kubernetes on-premises. It includes the challenges, opportunities and benefits, and considerations for running Kubernetes on bare metal. It also details infrastructure requirements and best practices for on-premises DIY Kubernetes implementations.

[1] https://platform9.com/blog/kubernetes-on-premises-why-and-how/

Standardize the Container Build Process

Containers are a key building block of a microservice architecture, and how they're managed directly impacts the efficiency, reliability, and availability of services running in Kubernetes clusters.

The container build process should be automated using a CI/CD system. Open source tools such as Jenkins are widely used CI/CD tools. Major public cloud providers also offer CI/CD services. These tools reduce the workload on developers when deploying a service. They also allow for automated testing prior to deployment, and can support rollback operations when needed.

Container images should be stored in an image repository. This centralized store should also support image scanning to check for security vulnerabilities. By providing an image repository, you can promote the consistent use of approved images. This reduces the chance of deploying a misconfigured container. Developers have a range of container registry options, including Docker Hub,[1] JFrog Container Registry,[2] and JFrog Artifactory.[3]

For example, a Docker image may require that several tools be installed, and those tools may require different versions of the same library. Someone unaware of the potential conflict might fail to properly install the library's multiple versions. This could lead to deploying an image that will fail in production and require a team of DevOps engineers to diagnose in production.

A standardized image build process helps remediate failed deployments. For example, the CI/CD pipeline can be configured to perform a canary deployment, in which a small amount of traffic is routed to a newly deployed service. If there's a problem, only a small number of users are adversely affected.

[1] https://hub.docker.com/

[2] https://jfrog.com/container-registry/

[3] https://jfrog.com/artifactory/

Alternatively, the CI/CD process could employ a rolling deployment in which pods are replaced one by one, allowing for an incremental transition to a new version of a service. Of course both canary and rolling deployments could be done manually, but that would be more time-consuming and error-prone (see **Figure 4**).

As part of the image build process, be sure to include a monitoring mechanism. Some monitoring tools use

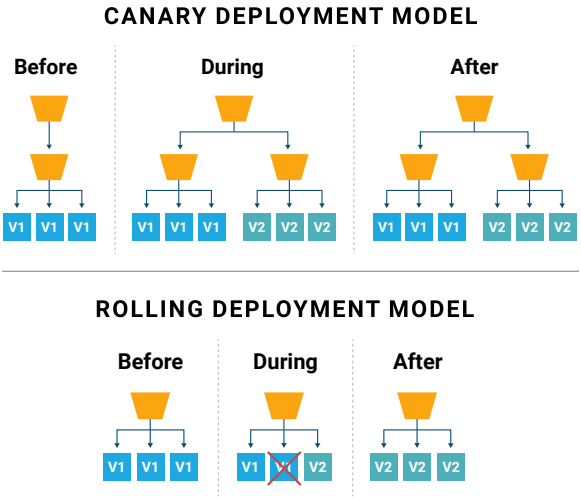


Figure 4: Deployment models like rolling and canary deployments help identify any issues with a new deployment before it causes any significant impact to production.

agents to collect server and application performance data and send it to a centralized data store for reporting and alerting. It's important to have visibility into the performance of services so you can correct issues that can't be addressed directly by Kubernetes.

Ensure Self-Service

Developers should not have to coordinate with IT administrators to deploy and monitor services running in Kubernetes clusters. To ensure developers can manage deployments, it's important to provide tools for deploying and scanning applications.

For example, Helm is a package manager for Kubernetes and supports defining, deploying, and upgrading applications, which can streamline the management of applications. Security scanning tools should be in place as well, to help developers identify vulnerabilities in applications before they're deployed.

Applications and Storage

With developers and admins alike working across multiple environments, it's also important to have policies in place to enable efficient use of resources. Consider RBAC policies and limits to ensure resources are used

fairly, and that no single deployment consumes an excessive amount of resources.

Operations Best Practices

In addition to employing deployment best practices, there are several operations best practices you should strive to implement, including:

- Single pane of glass visibility
- Scaling best practices
- Governance and security
- Upgrading

Together, these best practices can help reduce the operational overhead associated with maintaining Kubernetes clusters.

Cluster Observability

The idea behind single pane of glass visibility is that all information needed to understand and diagnose the current state of the cluster, deployments, and other components should be available from a single tool.

For example, from a single application, administrators should be able to configure monitoring, analyze monitoring data, and specify alerts triggered by that

monitoring data. Plan to use a standardized set of monitoring tools for collecting, storing, analyzing, and visualizing performance monitoring data.

This monitoring functionality can also be used to monitor compliance with SLAs. Another advantage of standardizing is that you can define templates to promote reusability.

Scaling Best Practices

When scaling with Kubernetes, you have the option of scaling the size of a cluster or increasing the number of clusters. When workloads vary widely, the Horizontal Pod Autoscaler can be used to adjust the number of nodes in a deployment. Kubernetes also has a Vertical Pod Autoscaler, but that's currently in beta release and shouldn't be used in production.

One scaling question you'll face is whether to run one cluster or multiple clusters. Kubernetes can scale to thousands of nodes and hundreds of thousands of pods, so a single cluster can meet many use cases.

There are, however, some advantages of using multiple clusters. One is reliability. In the event of a cluster failure, all workloads are affected in a single cluster environment. Also, with multiple clusters different development teams can manage their own clusters—and

Platform9: Your Trusted Kubernetes Partner

Kubernetes is a powerful tool, but that power comes at the price of complexity. And the more you scale up and out, the more complex it gets.

That's why it can make sense to call in the cavalry. Bringing in a partner who specializes in Kubernetes can get you up and running much more quickly, and help you manage the new infrastructure more efficiently.

Platform9,[1] for example, helps automate Kubernetes, freeing you from the significant burden of a DIY approach. This is what the company does, and it's an expert at it.

Its managed Kubernetes platform, known as PMK, provides the ability to easily run Kubernetes at scale. It allows you to leverage your existing environments, with no operational burden on your IT staff. You get the power of Kubernetes without the hassle of managing the complexity.

When you're ready to do more with your Kubernetes, be sure to check out Platform9's free sandbox and freedom plan,[2] which users can try out at no charge.

[1] https://platform9.com/

[2] https://platform9.com/sandbox/kubernetes/

that can increase the velocity of each team, if it doesn't need to coordinate cluster changes with other teams.

Governance and Security

As with any enterprise platform, you'll need to consider governance and security. To start with, plan to implement granular RBAC. These can be used to implement the principle of "least privilege," which states that users should have only permissions they need to perform tasks assigned to them and no more.

Use audit trails to track security-related changes in the system. For example, operations, like adding a user and changing permissions, should be logged. Also, use encryption to secure communications both within and outside of the cluster.

It's a best practice to scan applications for vulnerabilities. In a similar way, you should review vulnerabilities in Kubernetes software and patch as necessary. This is a situation in which it may be beneficial to have multiple clusters, because a patch can be deployed to a single cluster and evaluated before rolling it out to others.

Upgrading

Kubernetes is under active development, which provides for new functionality and improved reliability.

As part of your Kubernetes management strategy, plan to upgrade while supporting production workloads. For example, the master will need to be upgraded before nodes. To avoid disruption, you can run multiple master nodes and upgrade one master at a time or use rolling upgrades to ensure zero downtime during the upgrade process.

Similarly, nodes can be upgraded incrementally. Also plan to patch and upgrade operating systems running on nodes. Be sure to maintain up-to-date backups. Backups are an important insurance measure for recovering from a failed upgrade.

They're 'Best' Practices for a Reason

Kubernetes is a complex platform that provides for highly scalable, efficient use of computing and storage resources. But it can also be highly problematic for companies that just try to "wing it" and figure out what to do as they go along.

Don't let that be you. Following the best practices outlined here will help to ensure that you realize the optimal benefit of your Kubernetes investment.

Work Your Kubernetes Plan

As you've seen through this Gorilla Guide, Kubernetes is a powerful way to orchestrate your container environment. That's why it's become the de facto method for the IT industry. But that power comes at the price of complexity—and as your operations scale up, it becomes more difficult.

It's a challenge, to be sure, to properly deploy and run Kubernetes. But the sizable advantages that come along with it make the effort well worth it. The key is to understand what you want to do with containers *before* deploying your first pod. Spinning them up without a well-thought-out plan can be inviting disaster.

Don't let that be you.

ABOUT PLATFORM9

Platform9[1] enables freedom in cloud computing for enterprises that need the ability to run private, edge, or hybrid clouds. Our SaaS-managed cloud platform makes it easy to operate and scale clouds based on open source standards such as Kubernetes and OpenStack, while supporting any infrastructure running on-premises or at the edge.

[1] https://platform9.com/

ABOUT ACTUALTECH MEDIA

ActualTech Media is a B2B tech marketing company that connects enterprise IT vendors with IT buyers through innovative lead generation programs and compelling custom content services.

ActualTech Media's team speaks to the enterprise IT audience because we've been the enterprise IT audience.

Our leadership team is stacked with former CIOs, IT managers, architects, subject matter experts and marketing professionals that help our clients spend less time explaining what their technology does and more time creating strategies that drive results.

For more information, visit
www.actualtechmedia.com