PLATFORM9 PRESENTS

THE GORILLA GUIDE TO...®

Scaling Kubernetes for the Enterprise

Joep Piscaer

INSIDE THE GUIDE:

- Creating an Optimal DevOps Experience with Distributed Kubernetes
- 10 Considerations for Running Kubernetes at Scale
- SaaS Managed Kubernetes: The Effective DIY Alternative

HELPING YOU NAVIGATE THE TECHNOLOGY JUNGLE!

ActualTech Media
www.actualtechmedia.com

In Partnership With

PLATFORM9

THE GORILLA GUIDE TO...®

Scaling Kubernetes for the Enterprise

By Joep Piscaer

ACTUALTECH MEDIA

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

PUBLISHER'S ACKNOWLEDGEMENTS

EDITORIAL DIRECTOR

Keith Ward

DIRECTOR OF CONTENT DELIVERY

Wendy Hernandez

CREATIVE DIRECTOR

Olivia Thomson

SENIOR DIRECTOR OF CONTENT

Katie Mohr

PARTNER AND VP OF CONTENT

James Green

WITH SPECIAL CONTRIBUTIONS FROM PLATFORM9

Kamesh Pemmaraju, Bic Le, Roopak Parikh

ABOUT THE AUTHOR

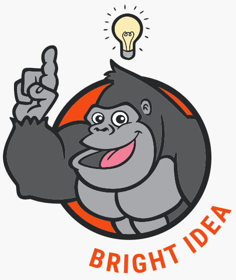
Joep Piscaer is a seasoned IT professional, with 10-plus years experience as a CTO, head of IaaS and infrastructure, (enterprise) architect, and technical consultant. His specialization is in infrastructure, cloud, and way-of-work (DevOp, Infrastructure-as-Code). He has built Infrastructure-as-Code toolchains, IaaS platforms, transformed (infrastructure-focused) organizations to DevOps and Infrastructure-as-Code ways of work.

ENTERING THE JUNGLE

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.

The title is a bit odd. The theme is that Kubernetes is a common underlying "fabric" that can bring together diverse and distributed infrastructure: multi-clouds, edge, and on-premises

INTRODUCTION

The Conductor of Online Transformation

Welcome to The Gorilla Guide To...® Scaling Kubernetes for the Enterprise. If you're reading this, it likely means you're either thinking about modernizing your infrastructure, or you're in the midst of doing just that, and need to take things even further.

In a relatively short time, Kubernetes has become the leading platform for enabling much of the cloud computing revolution. If you're doing cloud-native development, you're probably doing it with Kubernetes.

It's the same thing if you're planning to run applications on-premises, in multiple clouds, or at the edge ~~moving on-premises applications and data to the cloud~~. Kubernetes is the conductor of this online transformation, and understanding how to squeeze the most out of it is crucial.

This Guide can be an important part of that process. 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

In This Chapter:

- The different types of Kubernetes deployment models
- Advantages of local data processing
- Security considerations and the importance of coordinating across all environments

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**).

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.

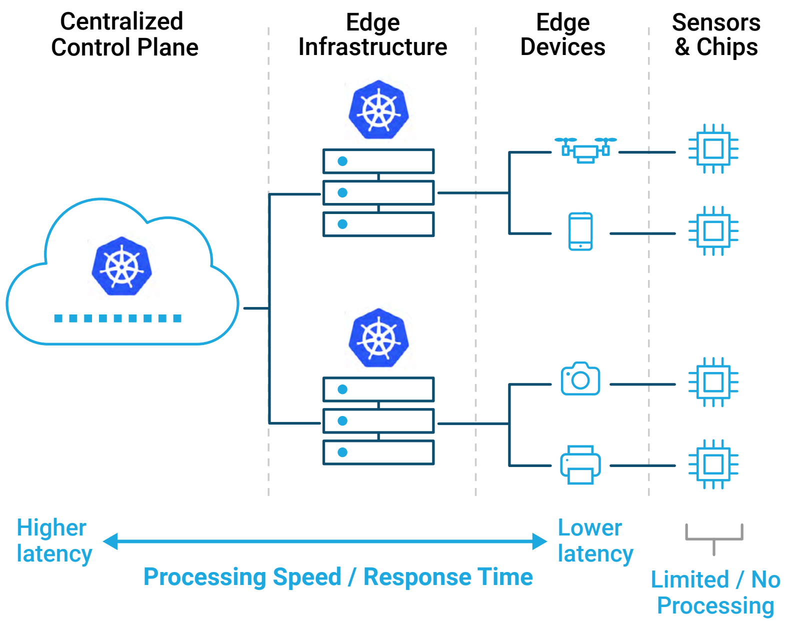


Figure 1: The central management of data centers and edge locations

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

In This Chapter:

- Platform engineering teams should treat the platform as a product
- The importance of consistent policies, practices, and tools
- How application owners can ensure an optimal developer environment

As mentioned in Chapter 1, 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 colocated 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 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.

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.

"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 … for IT include security and complia…

Seems like an overstatement without a reference. I would remove this unless backed by a reference link.

Some estimates of th… adow 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.

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

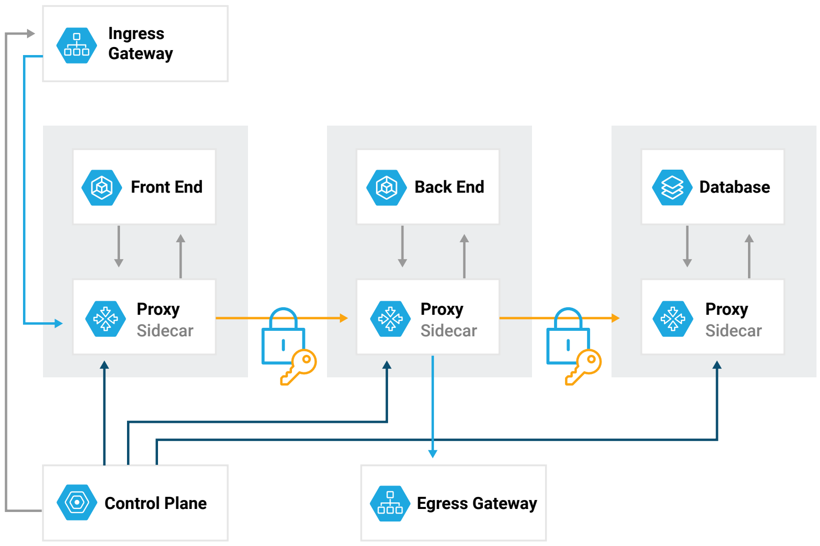


Figure 2: Service mesh traffic overview

across all deployments of 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

In This Chapter:

- The developer's more expansive view of availability
- The more complex a system becomes, the more important it is to be able to determine the state of that system at any time
- Maintain appropriate levels of performance

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**).

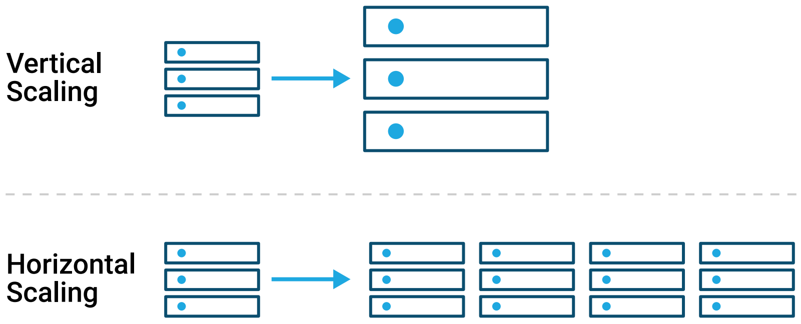


Figure 3: Horizontal vs. Vertical scaling of nodes

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 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. 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.

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 post1 that can help pave the way to successful Kubernetes upgrades.

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

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.

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.

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.

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.

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

In This Chapter:

- Deployment best practices
- Operations best practices
- Platform9 Managed Kubernetes

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.

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/

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.

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

1 https://hub.docker.com/

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

3 https://jfrog.com/artifactory/

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.

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 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.

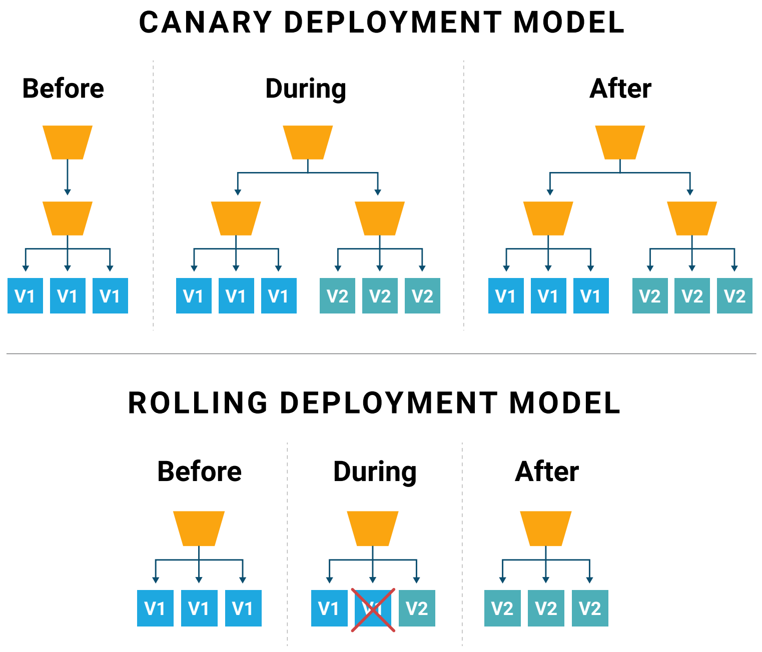


Figure 4: Deployment models like rolling and canary deployments help identify any issues with a new deployment before it causes any significant impact to production

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

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/

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 these first four chapters, 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.

The next four chapters focus on Kubernetes operations, which include monitoring, upgrading, restoring, troubleshooting, security patching, logging, alerting, load balancing, exception handling, DNS services, and more. We'll focus on three major areas of your infrastructure that require standardization and consolidation:

- Monitoring and observability
- A service mesh for communication
- Automated management and updates of multiple distributed locations

First, let's look at SaaS managed Kubernetes: the effective DIY alternative.

CHAPTER 5

SaaS Managed Kubernetes: The Effective DIY Alternative

In This Chapter:

- Outsourcing Kubernetes to a managed service provider
- Avoiding vendor lock-in
- Focus on business outcomes

The operational complexity of Kubernetes is a major hurdle for many organizations, creating a barrier to entry that's hard to solve: qualified engineers are expensive, and DIY solutions have a long lead time and are very complex.

This means organizations are missing out on the advantages that Kubernetes provides for development teams, like accelerating software releases with more control over the infrastructure to optimize performance and cost.

This chapter helps you understand how to remove complexity and decrease lead time for Kubernetes using a cloud-agnostic, managed solution approach.

Kubernetes is a powerful infrastructure platform for developers. Its self-service nature allows developers to take control of releasing software to production without the direct involvement of Ops teams. This helps development teams increase their velocity, enabling them to release more often and more quickly, with more control over the infrastructure than ever to optimize cost, performance, and resilience.

The downside is increased complexity. With all of its advantages when it's up and running, Kubernetes is notoriously hard to deploy and manage. Its open, pluggable architecture is complicated and can be overwhelming for those new to Kubernetes.

This creates high barriers to entry for Ops teams wanting to design and implement production-grade clusters that provide high resilience and good performance at a reasonable cost.

In the architecture diagram in **Figure 5**, it becomes clear that Kubernetes is a complex solution with many moving and interchangeable parts.

Even after the initial learning curve is conquered, new challenges await. Kubernetes clusters tend to have a shorter lifespan than virtual infrastructure clusters, and are often built for a very specific function, such as a single application. This is especially true in environments with ephemeral compute needs, like cloud computing. Here, clusters here are constantly spun down, recycled, and replaced by new clusters.

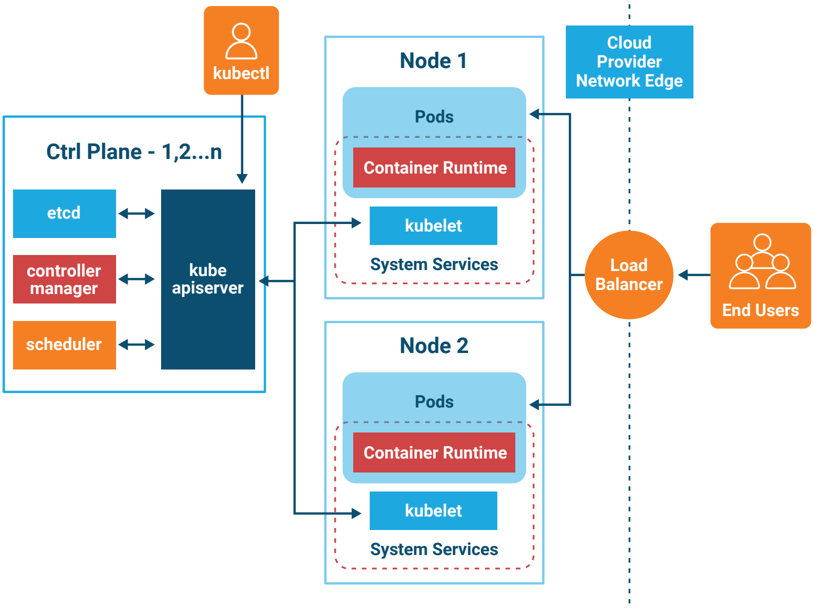


Figure 5: Kubernetes architecture is difficult to understand and master

Kubernetes, Microservices, and Modern Development

The powerful benefit offered by container technologies such as Kubernetes is also the source of the difficulties that containers present. They're part of an architecture for computer applications that combines:

- Multiple services, such as databases, that communicate by exposing APIs
- A microservice architecture, which is a modern form of the classic programming practice of breaking programs into independent modules
- Cloud deployment, which makes it easy to scale up or down in response to demand

This type of architecture is becoming increasingly popular for several reasons First, it allows a company to spin up new instances—virtual representations of a physical computer—as demand increases, such as if it advertises a sale and thousands of people suddenly visit its web site. The company saves money by terminating those instances when the spike ends.

The architecture is also fault-tolerant. If a physical computer blows a fuse, you have to power up a new computer. In a virtual or containerized environment, you just press a button on the cloud provider's graphical interface—or even better, run a monitoring system to detect failures and to automatically launch new instances through the cloud provider's API.

Less obvious, but equally important, the containerized architecture supports a faster, more distributed development cycle. Programmers in different places can work on different microservice components and indulge in their own release cycles, without having to communicate frequently with the developers of other components. Because many new features require changes to only a single component, developers can keep the users happy with fast-changing stream of new features, or can react quickly to a change in the business model.

This poses major challenges in the way Ops teams work, requiring new operational processes. It also requires new technical skills, which use up team resources that could have been better spent on other, more business-focused initiatives.

Indeed, chances are that Ops teams are already stretched thin, and putting in the time to master the day-to-day chores of managing a cluster's lifecycle is simply another added burden. Ops is already busy maintaining existing infrastructure, from WAN and LAN networking, to virtual compute infrastructure, client management systems, and many others.

Outsourcing Makes Sense—on the Surface

It is therefore logical to outsource Kubernetes to a managed services provider. Luckily, there are many options to choose from, across a spectrum of different types of solutions.

On one end of this spectrum are software distributions that provide a framework, but leave you alone to figure out how it works. While better than rolling out your own Kubernetes distribution, these options still require you to do all of the heavy lifting yourself, and don't actually solve the problems outlined earlier.

The hyperscale public cloud vendors go one step further, by offering these software distributions as an easy-to-consume service. These services take care of initial cluster deployment, using their own design best practices and implementation tooling.

While saving massively on initial deployment lead time, these solutions have a number of downsides. Public cloud vendors' service portfolios are designed to lower the barriers for developers to start using additional services, often without an upfront cost. It's an elegant, integrated portfolio of services that developers love.

Lock-in Blues

It's not without its drawbacks, though—the biggest one being vendor lock-in. These hyperscale public cloud vendors have many services, and they'll try to persuade you to use their other services, as well. Lock-in increases steadily, drawing you in as a customer with each step.

Lock-in

Software vendors—and now cloud providers—have devised many barriers over the years to leaving their offerings. Some of the dangers of lock-in are:

- The vendor going out of business, which means stopping updates and bug fixes or (in the case of cloud offerings) terminating all services. Customer data can also be lost.
- More subtly, a change in the vendor's plans and priorities may take the product in a direction that doesn't help some of its own clients. They may find that the product no longer offers the original value they found in it, but that their data and processes are trapped in it.
- Similarly, the vendor may decide to remove a feature that the client relies on. Perhaps the feature isn't important to the clients the vendor wants to attract. Often, the vendor sees the feature as a hindrance to selling some other services, and removes it to force clients to spend more for a new service.
- Bugs of high priority to the client may be of low priority to the vendor, so they may go unfixed.
- Vendors have been known to raise prices so much that they make the offering unaffordable to many users. Sometimes they suddenly start charging high prices, without warning, for offerings that were previously free of charge.
- Thus, professional administrators and operators are wary of lock-in.

While, operationally, the level of integration between services and products is naturally very high, the cost and the operational and strategic risk increase exponentially.

The public cloud provider may lock you into using their authentication, monitoring, or data services (like databases and object storage). For instance, to use their managed Kubernetes services, you have no alternative but to use their compute instances, block storage service, and monitoring services, as well as their authentication service. This lack of choice increases the lock-in.

In the Kubernetes realm, another lock-in is more obvious: the managed Kubernetes service often dictates what compute instances can be used, in the sense that you can only use their compute nodes. This can prevent you from using a third-party service or bringing your own compute.

And while technically there's nothing wrong with using their compute nodes as Kubernetes worker nodes, they do make up the vast majority of cloud costs. And what happens if they alter their terms or hike prices? If you're locked in, you may feel you have to accept these unwanted changes.

But maybe not. And that leads to an often-hidden, but frustrating and expensive, issue—breaking the lock and finding another provider can be incredibly frustrating. The time it takes to move away from a specific public cloud service when locked into that ecosystem can take many months, and may have a significant impact on your projects and budgets. Sometimes providing your own infrastructure is cheaper and offers more agility.

While lock-in is often associated with the risk of cost increases, the strategic risk of not being able to move and adapt to changing circumstances in your business could also be crippling, especially when the services are used for customer-facing digital transformation projects.

This means a loss of agility in the marketplace, since you're no longer able to adjust requirements in response to changing circumstances. That increases your Total Cost of Ownership, or TCO, when using the public cloud vendor's entire service landscape.

Mitigating Risk

Intelligent planning demands that organizations consider solutions that don't have economic and operational lock-in, while still offering Kubernetes as a service. The value proposition is clear: the enterprise receives all the benefits, without the downside.

With SaaS, you're essentially hiring the best consultants to assist with the architecture design, configuration, and operational processes to optimize your Kubernetes environments for availability, resilience, security, and cost. But you don't pay the high price of a specialized consultant. Instead, the SaaS provider creates the automated workflows and the back-end automation that allow hands-off initial deployment, upgrades, monitoring, alerting and more. Since it's all done with software instead of labor-intensive manual processes, it scales elegantly.

Smaller organizations simply can't justify the cost of a dedicated Operations team with the appropriate Kubernetes knowledge and experience, which requires expensive staff—nor can they run the risk of being dependent on a single worker or a small number of employees for their specific knowledge.

Instead, cloud-agnostic, managed Kubernetes services, like Platform9 SaaS Managed Kubernetes service (PMK), are indifferent to the location of your Kubernetes cluster: on-premises, in a private or hosted cloud, across any of the public clouds, or in a combination of all of these.

Workloads are moving increasingly to the network edge. This dynamic creates hundreds or even thousands of new locations. In that scenario, operational overhead ramps up massively, leading to huge management nightmares. But SaaS provides central management of those widely distributed clusters with the simplicity of a single pane of glass console. That means formerly labor-intensive operations like software updates are as easy as the click of a button.

PMK offers quick onboarding to Kubernetes for developers, allowing them to use the service without any re-training, and includes many of the moving parts that usually accompany Kubernetes for monitoring, logging, networking, and storage.

Focus on Business Outcomes

Managed Kubernetes services are invaluable in other ways, too. Not only do they remove the operational complexity of designing, implementing, and operating Kubernetes, they allow organizations to focus their staff's time on things that directly impact their bottom line.

Instead of ITOps staff focusing on daily IT operations, they will have time to spend on business projects, which increasingly have an IT or tech component.

Thus, a managed Kubernetes platform has two key advantages. First, your organization will have a proper Kubernetes infrastructure, which is a driver for many digital transformation, digitization, and e-commerce projects. It allows organizations to quickly develop, release, and iteratively improve customer-facing applications.

Second, freeing up IT staff from their day-to-day task will accelerate those projects by adding invaluable tech skills and experience into the mix, without having to risk being pulled back into yet another operational fire that needs their immediate and undivided attention.

Many companies underestimate this latter aspect of using a managed Kubernetes service. Freeing up technical staff that know the organization's technology stack and all of its subtleties, technical debt, and quirks can have a massive impact on the quality of the software delivered as part of those innovative projects.

Improving Time-to-Value

In addition, using a managed Kubernetes service allows organizations to hit the ground running. Instead of slowing down a transformation project to hire the right staff, design, install, and configure a Kubernetes environment, a managed Kubernetes service helps speed up projects by decreasing lead time for the technical aspects of building a Kubernetes environment.

This newly unencumbered IT staff can be the difference between a successful digital transformation and a failed one. IT staff have a crucial role in non-functional aspects. While functional characteristics define specific behavior and functionality, non-functional aspects define qualitative aspects of a system, including stability, availability, resilience, security, performance, manageability, upgradeability, cost, and more. With IT staff safeguarding those attributes, these projects will deliver a better end result, and more quickly.

Now that we've examined the value of a dedicated service to manage your containers, we'll look at what such a service offers. The first of such a service is to monitor what's going on so you know when there are problems. Thus, monitoring and observability are the topics of the next chapter.

CHAPTER 6

Tackling Observability in Your Kubernetes Environment

In This Chapter:

- Types of observability and their value
- The different layers of application monitoring
- Making observability work for your business

We're on our way to discovering a robust and vendor-neutral way to manage Kubernetes instances. In this chapter we look at the foundation of management: monitoring.

There are many tools in the cloud-native and microservices tool chest. Kubernetes is the go-to for container management, giving organizations superpowers for running container applications at scale. However, running an enterprise-grade, production-level Kubernetes deployment is more than running just Kubernetes by itself.

Because containers are ephemeral and transient, monitoring, security, and data protection are fundamentally different from their counterparts in virtualized or bare metal applications. Optimizing the tooling that supports a Kubernetes deployment is not a trivial task. In many cases, this means that tooling aimed at virtualized environments doesn't translate well into containerized platforms. Replacing these tools may be better than retrofitting legacy tooling.

In fact, more modern tooling created to support container environments may help you get the most out of your container platform. In this chapter, we'll look at how to optimize observability in your Kubernetes environment. We'll define the types of observability, offer a path for starting and expanding the process, and describe the advantages of cloud-based or Software as a Service (SaaS) monitoring.

Types of Observability and Their Value

Let's go back to basics first. Looking at the *observability* space for container-based microservices landscapes, we can distinguish three separate types of tooling:

1. Monitoring (or metrics): collecting operational telemetry about applications, application services, middleware, databases, operating systems, and virtual or physical machines
2. Logging: collecting error messages, debug or stack traces, and more detailed messages
3. Tracing: collecting user transactions and performance data across a single or distributed system

KUBERNETES

Watch the Platform9 webinar[1] on how Kubernetes has transformed monitoring. Another great resource to check out is the Platform9 blog,[2] "Logging & Monitoring of Kubernetes Applications: Requirements & Recommended Toolset."

[1] https://platform9.com/resource/how-has-kubernetes-transformed-monitoring/thank-you/

[2] https://platform9.com/blog/logging-monitoring-of-kubernetes-applications-requirements-recommended-toolset/

In a DevOps or SRE world, these three disciplines collectively make up observability.

Each discipline provides valuable insights in all layers of the layer cake that make up the increasingly complex application and infrastructure landscape of containers. DevOps engineers and SREs use the insights from these tools to improve resilience and performance, as well as triage errors, fix bugs, and improve availability and reliability.

Finally, they use these tools to gauge how users are interacting with the system. The tools help figure out which functionality visitors use or don't use, and where performance bottlenecks lie.

As application landscapes expand due to digital transformation, the number of microservices and individual containers explodes, making it harder to see the inner workings of systems. So, it shouldn't be a surprise that executing a good observability strategy is one of the deciding factors of a successful Kubernetes deployment.

While enterprise IT is more important than ever, digital transformation has led many more organizations to create digital and online applications. IT has often become a critical business function, vital for the survival of your business.

Layers of Monitoring

A good place to start with monitoring is by collecting metrics and operational telemetry of the Kubernetes constructs like clusters and pods, as well as collecting metrics on resource usage like CPU, memory, networking, and storage. Starting with the bottom two layers (see **Figure 6**) for monitoring is relatively easy and a good way of becoming comfortable with observability tooling.

Infrastructure monitoring and logging are key capabilities because it's important to know the activities of your physical infrastructure. A substantial amount of your application's performance and resilience comes from correctly functioning servers and networking.

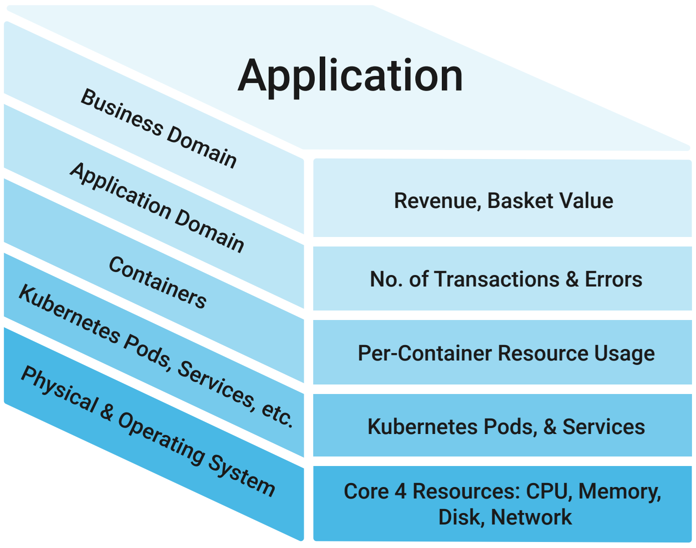

Application
Business Domain
Application Domain
Containers
Kubernetes Pods, Services, etc.
Physical & Operating System
Revenue, Basket Value
No. of Transactions & Errors
Per-Container Resource Usage
Kubernetes Pods, & Services
Core 4 Resources: CPU, Memory, Disk, Network

Figure 6: An application layer cake with monitoring examples

As the application landscape expands, a well-executed infrastructure monitoring and logging strategy also builds a shared understanding of application performance across teams, preventing miscommunication between application development, cloud platform, and other teams.

Visibility into infrastructure and the shared understanding it builds is crucial, but of course doesn't give the entire picture. For that, you need to move up the stack, and start with application performance monitoring (APM). For many organizations, the application monitoring journey starts with monitoring (or metrics collection) and logging containerized workloads. For Kubernetes-based environments, there are natural combinations to start with, like the open source Fluentd and Prometheus, which make it easier to run monitoring and logging.

Making Observability Work for Your Business

To gain benefits from observability, you need to think about your business requirements over the next few years and choose a platform that meets your needs. This section describes how to think about requirements, tools, and the IT organization.

Align Monitoring to Business Objectives

This chapter has presented a natural progression of how teams use observability, starting with the infrastructure basics, working their way up the stack into the realm of applications, and even tracing users across the application landscape, monitoring their behavior and transactions.

This journey up the stack is an opportunity to align monitoring, logging, and tracing to business objectives, mining more insights from the increased visibility. It allows teams to gain visibility into more than just technical metrics, generating business-oriented metrics, too.

By measuring business-oriented metrics (such as the dollar value of the shopping basket, the number of abandoned baskets, and metrics on popular or even disused features), product owners can align development priorities to what their users really want, optimize performance in areas where it actually matters, and fix technical debt to accommodate further growth. Naturally, these insights fuel business growth and revenue.

When tooling is aligned to the business and customer experience, the tools can be used by more than just IT teams, allowing business teams to gain insights into their applications and its users.

Think Mid-Term to Long-Term

The tools you choose for observability should serve your needs for several years. This requires you to think about how your business is changing and how that will change your observability requirements in the long-term.

The cost of migrating to a new, more capable APM platform can be significant, but won't immediately give you additional functionality. This additional functionality requires additional engineering and implementation before these capabilities are fully unlocked.

And let's not forget that moving to another APM platform requires you to retrain staff and needs time to regain confidence in the metrics and insights, all of which reduce the value the APM platform brings in the short term. So, it makes sense to choose your tooling wisely from the start, keeping the long-term goals in mind.

In other words, while you won't need the most complex or feature-rich solution now, look at what features you'll need to support evolving requirements in the future. Invest in your team and people and start with the APM capabilities you need now. You don't need to enable, implement, and incorporate every feature the tooling provides from the get-go. It's OK to start simple, build up confidence along the way, continuously evolve your knowledge of the tool, and expand its use in-sync with changing requirements.

Cloud-Based Observability (SaaS)

If your teams are busy setting up and managing servers, they'll be less effective at their real job: improving the usage of APM across the organization. By adopting a SaaS offering, the observability platform team can focus on the functional side of observability instead of the day-to-day toil of managing and operating the platform.

The Observability Platform Team

As your organization grows and your use of metrics becomes commonplace, it makes sense to create a dedicated team focused on the continual improvement of the observability platform, and help application development teams gather the metrics that matter to them.

The dedicated team owns the platform and executes an observability strategy across many applications and teams. The observability platform team's expertise speeds up troubleshooting, helps with application architecture optimization and can help teams to quickly pinpoint and solve bottlenecks and fragile areas prone to failure.

With their expertise and knowledge of the environment, they can "travel" from application team to application team to grow each team's knowledge and expertise more efficiently, preventing re-inventing the wheel and other local optimizations within each team, instead letting all teams learn from the collective knowledge, driving up maturity more quickly.

With this team structure, the application teams can focus on gathering metrics, refining the metrics they collect, and improving the signal-to-noise ratio inherent in gathering metrics, so the telemetry gathered is optimally serving business objectives and the team isn't spending any time on managing or operating the observability platform.

That means they're not bogged down with installing yet another security patch or forced to think about scaling the observability platform. Instead, they have more time to help application development teams with their observability challenges or to improve the platform itself.

SaaS also helps an organization get started with an observability platform. Instead of having to make the hard design choices at the start of a project (with little to no experience), you can use the choices offered by the SaaS vendor, confident that its experts have vetted the choices.

Using a SaaS service lets the vendor take on the operational burden of upgrades, scalability, and performance for the APM software, freeing up your team to work on broader and deeper implementation of APM across application teams.

This will speed up the pace at which your observability platform matures, allowing teams to gain a deeper and more user-oriented understanding of the application landscape more quickly. This increases the value of the investments made in the observability space in two ways: You spend less time installing, configuring, and getting started with the tooling, and the insights gained from the tooling can be applied more quickly to optimizing revenue and fueling growth.

In effect, you can leapfrog your APM journey, skipping the traditionally hard first steps in getting started with monitoring.

This chapter laid out the essential role of monitoring in the management of Kubernetes clusters. In the next chapter, we turn to the networks on which you run your Kubernetes containers.

CHAPTER 7

Best Practices for Selecting and Implementing Your Service Mesh

In This Chapter:

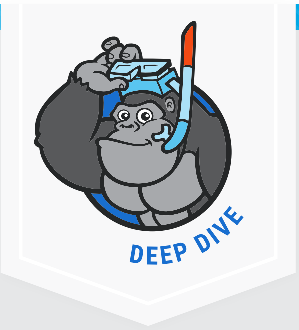
- Reducing service mesh complexity
- The service mesh catch-22
- The three leading, mature service mesh options available in the Kubernetes ecosystem: Consul Connect, Istio, and Linkerd

As applications are being broken down from monoliths into microservices, the number of services making up an application increases exponentially. And as anyone in IT knows, managing a very large number of entities is no trivial task.

Service meshes solve challenges caused by container and service sprawl in a microservices architecture by standardizing and automating communication between services. A service mesh standardizes and automates security (authentication, authorization, and end-to-end encryption), service discovery and traffic routing, load balancing, service failure recovery, and observability.[4] Just as virtualization abstracted the hardware layer of computer systems and containers abstracted the operating system, a service mesh abstracts away communication within the network.

[4] https://platform9.com/resource/tackling-observability-in-your-kubernetes-environment/

Why a Service Mesh?



As monoliths are pulled apart into their smallest constituents, the resulting microservices are usually distributed across multiple systems and communicate over HTTPS, so they become heavily dependent on network communications.

A service mesh manages the network communications by setting up standards and automating their implementation. It frees developers from defining and implementing the communications for every service, over and over again (see **Figure A**).

This is much more scalable, more automated, and less error-prone. The service mesh also improves security and reliability by standardizing the interface between services. The service mesh acts like an automatic walled garden for each service on the network.

This is done by making sure other services know the service exists (called "service discovery"), managing authorization and authentication between services, taking care of load balancing, and adding security policies for communication to and from the service by the outside world.

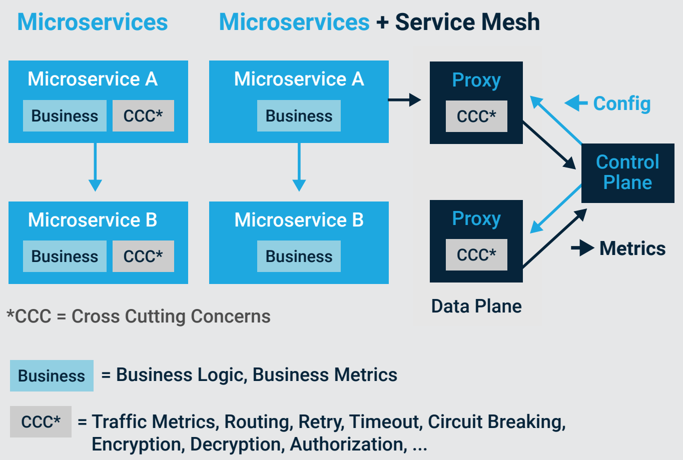

Microservices
Microservices + Service Mesh
Microservice A
Business
CCC*
Microservice B
Business
CCC*
Microservice A
Business
Microservice B
Business
Proxy
CCC*
Proxy
CCC*
Config
Control Plane
Metrics
Data Plane
*CCC = Cross Cutting Concerns
Business = Business Logic, Business Metrics
CCC* = Traffic Metrics, Routing, Retry, Timeout, Circuit Breaking, Encryption, Decryption, Authorization, ...

Figure A: Service mesh architecture

Because a service mesh has control over the network communication between all services in the mesh, it unlocks some advanced deployment and release strategies, such as canary releases, blue/green releases, and rolling upgrades.

This improves the reliability of the services in production. In some cases, the service mesh can react to changes in the traffic patterns, adding circuit breakers and rate limiters between services to prevent cascading failures. In order for teams to gauge the performance and quality of each release, a service mesh often has observability tooling (for collecting telemetry and metrics, as well as building in distributed tracing capabilities).

In short, a service mesh acts like an operational mission control to determine the behavior of microservices at scale, making sure the landscape of microservices is communicating securely, and monitoring performance and service quality. It removes much of the manual work from the developer's plate, so they need to focus only on the business logic, not the network, security, and communication plumbing.

The result is not only higher quality in business logic code, but also a reduction in variations and human errors in the plumbing, by standardizing and automating much of that work.

Reducing Service Mesh Complexity

Although a service mesh is very useful to development teams, implementing the service mesh itself still takes some work. Because there are many moving parts, a service mesh leaves a lot of flexibility and room to customize it to your specific needs. As always, flexibility comes at the cost of complexity.

Balancing the features, functionality, and value of a service mesh with its inherent complexity it is highly challenging, and requires expertise, but is well worth the effort. With an experienced team in place, organizations can overcome the complexity associated with running a service mesh at scale.

The best way to start developing the necessary skills and experience is no different from any other technology: start early, and start simple. You don't need to accelerate from 0 to 60 miles per hour instantly. Instead, start small, and incrementally add more features and functionality as you build trust in the service mesh.

It's recommended to start developing service mesh skills in tandem with your microservices architecture, because adding service mesh features to a relatively simple microservices architecture is much easier than when it's already complex and large. Let the service mesh grow organically alongside your ever-evolving microservices architecture. This keeps services secure and compliant, and helps maintain visibility.

The Service Mesh Team

As your organization grows and your use of the service mesh increases, it makes sense to create a dedicated team focused on the continual improvement of the service mesh, as well as helping application development teams make the most of the features and functionality it offers.

The dedicated team owns the service mesh platform and is responsible for the adoption of the service mesh across application teams and the entire microservices landscape. With this team structure, application development teams can focus on building business logic and microservices.

As you'll see in the following sections, having a dedicated team keep tabs on service mesh use cases (like multi-cloud and heterogenous workloads) may save you from an expensive, intrusive, and complex migration project because reality got in the way.

The Service Mesh Catch-22

Choosing the right service mesh technology, and nailing the implementation details, are crucial factors in your service mesh success. But how do you make the right decisions and do the right things when you don't have the right knowledge and experience yet? This is the catch-22 for the initial deployment and configuration of every new technology, including a service mesh.

This is a common pitfall for organizations, as engineers start designing and implementing a new technology enthusiastically. The inefficiencies and sub-optimal decisions due to lack of experience don't immediately come to light, but often surface only weeks, months, or even years later, when it's too late to drastically change anything.

How do you prevent these mistakes? And how do you kickstart the learning process without the associated risk and possibly massive impact down the road? Turning to a simpler, less feature-rich alternative carries its own risk, as you introduce a future point in time where your own maturity outpaces the limited feature set, forcing you to do a forklift upgrade of the mesh, introducing a migration not only of the mesh itself, but a migration of all the microservices in the mesh, too.

Instead, choosing the right mesh technology with the end-goal in sight makes more sense. Currently, there are three leading, mature options available in the Kubernetes ecosystem: Consul Connect, Istio, and Linkerd.

While there are differences, all three are battle-tested, production-ready, and enterprise-grade solutions. It's a matter of finding the right one given your unique context, requirements, and goals.

Istio has the most functionality and flexibility, but is also the most complex, making the first steps harder. Linkerd is Kubernetes-only, making it easier to implement and use. If you need to support virtual machines (VMs) alongside Kubernetes, Consul is a good choice.

The paradox here is knowing which level of flexibility you need a few years down the line when you have zero experience and expertise to make that decision now. Let's dive into an overview of these three options to start building a picture of which one is right for your organization. This will help you make the right decision and prevent obvious pitfalls as you build trust and increase your service mesh proficiency.

HashiCorp Consul Connect

Connect is Consul's service mesh feature. It provides service-to-service networking and security (authorization, encryption). As seen in **Figure 7**, applications can use a sidecar proxy deployment model.

These sidecars handle the inbound and outbound TLS connections, with the application completely agnostic of Consul. Consul also has a native integration deployment model. In Kubernetes environments, Consul uses a per-host DaemonSet agent and Envoy sidecar proxies per application that handles application traffic. Consul applies a zero-trust security model, is platform agnostic, and supports multi-cluster deployments.

As with other HashiCorp tools, Consul Connect is easy to get started with. Its deployment and initial configuration are a little less daunting than other options, making it a good solution for those very new to the service mesh space.

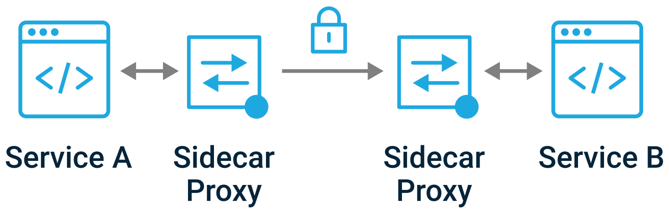


Figure 7: Consul Connect in a sidecar proxy model

Istio

Istio is the darling of the cloud-native space. Like many projects before it, it was open sourced by an end-user company (Lyft, in Istio's case), as they built a solution to handle complexity and scale.

Istio has seen massive adoption, especially as the basis of various public cloud offerings.

Istio's complexity is its downside for newcomers to the field, but also what makes it so powerful; one example is the addition of telemetry and analytics. As **Figure 8** shows, its architecture is much like Consul Connect.

A notable fact about Istio is that it is not part of the Cloud Native Computing Foundation

(CNCF) landscape map, even though it's the most popular service mesh option for the CNCF's Kubernetes ecosystem.

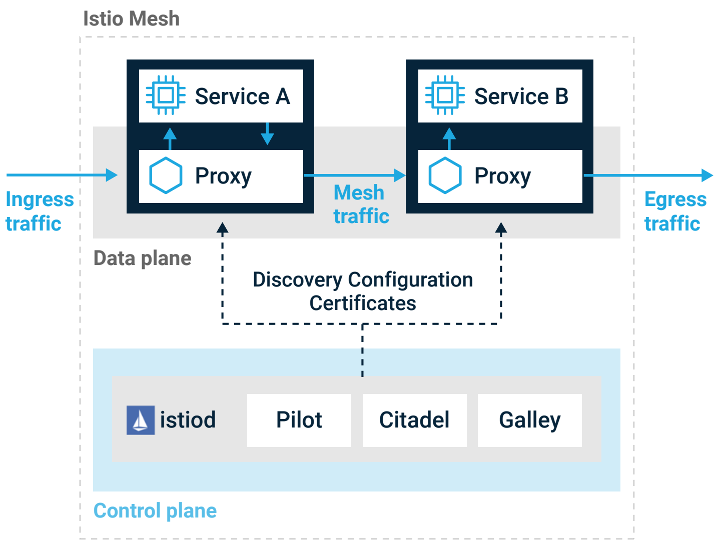


Figure 8: The Istio architecture

Linkerd

Linkerd is the CNCF answer to a service mesh (see **Figure 9**). Its v2 architecture mimics Istio, but favors simplicity over features and flexibility.

Where Consul and Istio work with Kubernetes and VMs, Linkerd exclusively works on Kubernetes. This means its architecture has fewer moving parts and fits into the Kubernetes architecture more seamlessly, with deeper integration into many other CNCF projects like Prometheus.

To get the full details, the Platform9 blog[5] has a post called "Kubernetes Service Mesh: A Comparison of Istio, Linkerd and Consul." It compares these three feature-by-feature.

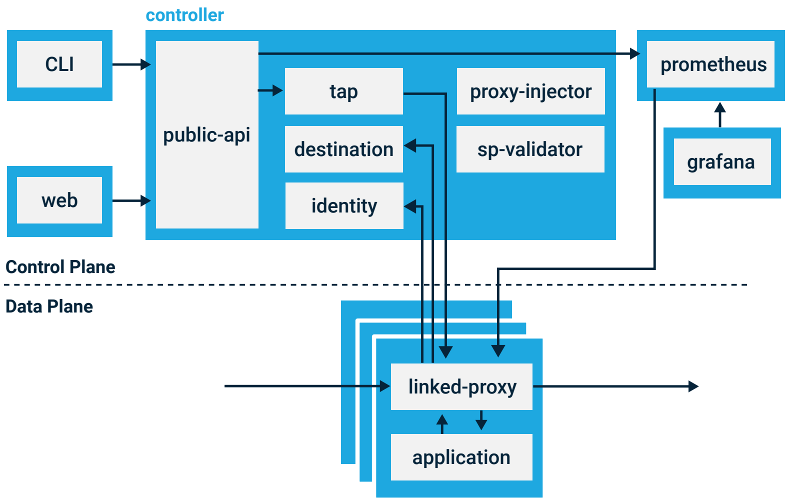


Figure 9: The Linkerd architecture

[5] https://platform9.com/blog/kubernetes-service-mesh-a-comparison-of-istio-linkerd-and-consul/

A Service Mesh Choice Is Not Forever

Even though you should now have the knowledge to make an initial choice, remember that requirements and circumstances change, so your service mesh will need to evolve, catering to those changes.

In some cases, a different technology is needed. If you're using the sidecar deployment model, applications and microservices running as part of the mesh are not aware of the mesh, nor do they have any special customization or integration with any specific mesh. The sidecar model makes it easier to migrate between technologies.

For more deeply integrated service mesh approaches, the Service Mesh Interface,[6] or SMI for short, may prove useful. SMI offers a set of common, portable APIs that provide developers with interoperability across different service mesh technologies including Istio, Linkerd, and Consul Connect.

Conquering Multi-Cloud

Reality is messy, and IT is no different. Migration from old technologies to new ones is always happening, whether from VMs to containers, from on-premises to public cloud, or from one public cloud to another. What use is a service mesh that helps you control traffic, security, permissions, and observability when it works for only a subset of workloads in just one environment?

Multi-cloud in a service mesh context means more than just multiple public clouds. It also needs to support on-premises deployments and support VMs. Last, the service mesh should span all these environments and have multi-cluster support.

This multi-cloud reality is often not explicitly designed by the organization, but "just happens." For instance, a group of developers starts using yet another public cloud, because it has the specific functionality

[6] https://cloudblogs.microsoft.com/opensource/2019/05/21/service-mesh-interface-smi-release/

they need to do their work. Whatever the cause, making sure your service mesh can handle this guarantees you can take a proactive approach to supporting the endless variety of multi-cloud scenarios in production. It gives you the piece of mind that you're in control of security in the untrusted world of public cloud, and have visibility into the entire microservices landscape.

In other words, if chosen correctly, a service mesh can serve as an abstraction layer on top of the public cloud, abstracting away the cloud and giving back control over traffic, security, permissions, and observability in a multi-cloud reality.

Looking at the three options shows that while Linkerd's simplicity sounds great on paper, reality may get in the way, requiring you to use a service mesh technology that works across containers and VMs. And again, SMI may help you migrate service mesh technologies if you need to—accepting and acknowledging that reality is messy may save you from a painful service mesh migration project.

With the tools we've described in the Chapters 5 and 6 of this guide, you can set up a robust architecture for your applications. But where should you actually run those systems? In the final chapter we'll look at a promising new environment for highly responsive containerized applications.

CHAPTER 8

Distributed Edge with Managed Kubernetes

In This Chapter:

- Bandwidth and compute power in a distributed architecture
- Why central management is still necessary
- Applying the distributed Kubernetes architecture to Retail, Manufacturing, and SaaS

This guide has laid out a strategy that can run in any modern environment, including multiple third-party clouds. This chapter presents a high-performance environment that has evolved recently, part of a movement known as *edge computing*.

EDGE COMPUTING

As Internet speeds grew and SaaS services became popular, applications moved a lot of their intelligence into the cloud. The popularity of mobile devices with limited storage and compute power accelerates this trend toward centralizing most processing in a remote hub. But for performance reasons, application developers are trying to do more processing with local data in the device itself, or in a cell or local system located near the device. This is called edge computing.

Each generation of networking presents unique challenges that are met by unique solutions. Current application architectures connect small computers at the edge—mobile apps, IoT devices, retail point-of-sale systems, and so forth—with hubs in the cloud. These environments are characterized by:

- Compute-intensive requests sent by mobile devices into the cloud, where a vendor is expected to handle the requests and return results quickly
- Cellular networks to handle most requests
- Modular, scalable worker nodes that handle requests and are controlled by Kubernetes, the most popular tool currently for distributing requests across worker nodes

This network architecture calls on companies hosting applications to place worker sites on network endpoints, so they can quickly accept requests from mobile device users, calculate the answers to the requests, and return them to the mobile devices.

RUNNING COMPUTATIONS IN CELL TOWERS

Don't cell towers contain just radios and antennae? Nowadays, towers are [illegible] compute power for their clients, including C[illegible]s compute power so that applications [illegible]ients instead of in far-away data cen[illegible] network costs.

We have a good webinar that talks about this: I would use that reference instead: https://platform9.com/resource/networking-and-kubernetes-in-the-world-of-5g-edge/

[1] https://ubuntu.com/b[illegible]g/edge-ai-in-a-5g-world-part-1-how-smart-cell-towers-will-change-our-lives

A New Architecture for Responding to Compute-Intensive Applications

During the past 10 to 15 years, companies have gotten used to accepting data from mobile users or edge devices into a centralized data center. This data center run by the company owning the app determines the proper response and returns it to the edge.

But this simple model has turned out to be inadequate for delivering the performance needed as the complexity of calculations grows. Cellular networks, especially those employing some form of 5G, have alleviated bandwidth limitations between the cellular tower and the edge, whether the edge is an end user on a mobile device or an IoT device reporting conditions in the field. But a significant bottleneck remains between the cell tower and the company's systems.

A new architecture has therefore developed, based on distributing the work to intelligent systems at the collection points provided by enterprises. Much of the information and work is never seen by the sites run by app developers. Instead, enterprise companies launch local containers to handle requests. Kubernetes instances are also launched at the endpoints to start up and monitor worker processes.

This chapter describes the new architecture and how to take advantage of it to offer applications that respond gracefully to user requests or reports from devices in the field.

Bandwidth and Compute Power in a Distributed Architecture

Edge computing brings compute power closer to the end users and their devices, essentially decentralizing some of the compute capabilities of centralized public cloud offerings.

Recent developments in connectivity, such as the increased bandwidth of 5G networks, have increased the opportunities for communication between edge locations and end users. This increase in connectivity allows an enormous growth of data and unlocks many new use cases, from image and video processing and voice recognition to running factories and retail locations over 5G instead of Wi-Fi. Fast response time is critical in the new applications. Mobile users are impatient, and IoT devices must make real-time decisions.

While the connection between the end user and the edge location enjoys ample bandwidth, responses from the centralized cloud or data center can't keep up in speed. Edge locations in many cases are remote and have limited connectivity, but require complex processing for huge amounts of locally generated data. That in turn creates an architectural challenge to bring compute resources where they can exploit the increases in bandwidth at the edge.

A Distributed Architecture Tailored to Kubernetes for the Edge

Transporting data from the edge to the central cloud or core data center for processing doesn't make sense, especially if there's a large amount of data to be transferred and fast response times are required. Processing at the edge is more cost-effective. In this architecture an edge location, such as a 5G radio tower, runs one or more clusters of worker nodes. The edge location sends only processed data that's useful for business-related analytics to the central repository.

This new architecture processes data close to where it's generated, with a few core data centers or cloud regions acting as the brains of the operation.

In this scenario, the edge locations themselves need sophisticated task management for thousands of simultaneous processes that are set up and torn down quickly. Kubernetes is the current industry standard for this kind of process management. That means starting up Kubernetes worker nodes at the edge locations to run the data processing applications locally. By running only the absolutely necessary workloads at the edge, companies can reduce costs associated with maintaining centralized data centers and transferring data from the edge.

Application providers are now dealing with hundreds to thousands of edge locations, or even more. With an architecture that requires less hardware at the edge, savings scale linearly with the number of edge locations.

Of course, this concept applies to more than just data-processing applications. 5G, as well as faster and more cost-effective endpoints (consumer and industrial IoT devices alike), are creating new use cases that generate far more data than ever before, such as video feeds from CCTV systems, telemetry information from industrial IoT devices, and interactions generated by apps and games on consumer phones.

Central Management Is Still Needed

Although the modern, distributed applications described in this chapter process data at the edge, application providers still need central control and visibility into the processing. Platform9 Managed Kubernetes (PMK) delivers Software-as-a-Service (SaaS) Kubernetes cluster management and simplicity of operations like native public cloud services but using upstream open source stacks that are deployed and operated on a wide range of on-premises (VMware, bare metal), public cloud(s) (AWS and Azure), and edge infrastructures.

The federated, distributed architecture of PMK provides a consistent experience across regions, while being centrally managed and resilient against connectivity and bandwidth issues. The architecture supports multiple regions in a hub-and-spoke model. **Figure 10** shows an overview of the federated architecture supported by PMK. Multiple regions work independently. Each management plane region acts as the central hub and brains of a region. The management plane defines

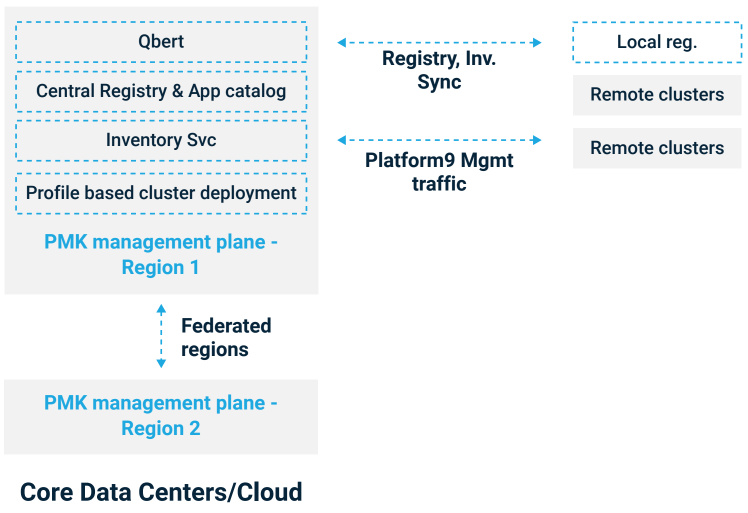


Figure 10: Distributed Edge Platform overview

policies for all the edge processors, with the federation of configuration templates and apps.

The management plane is where DevOps engineers manage the entire operation. There, they store container images and inventory caches of remote locations. Synchronization ensures eventual consistency to regional and edge locations automatically, regardless of the number of locations.

Profile-Based Management

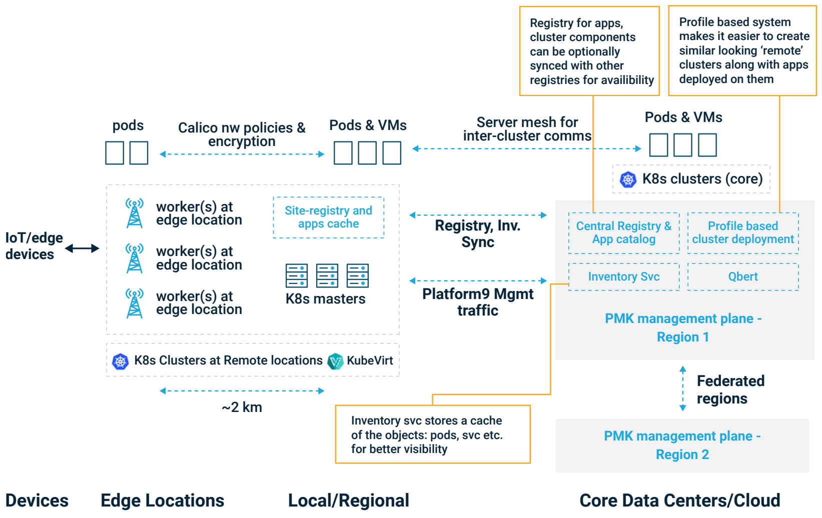
Platform9 facilitates the scaling of edge computing to thousands of edge data centers by grouping them so that similar data centers can be managed centrally through a single policy known as a *profile*. This relieves IT staff from managing each data center individually. Instead, the staff just defines a small number of profiles and indicates exceptions to policies where needed for a particular data center. Each edge location, such as radio towers, warehouses, and retail locations, runs its own worker nodes and containers.

Profile-based cluster management makes it easier to deploy identical remote clusters and configurations, from a single profile, instead of managing each remote cluster separately. That minimizes configuration drift, while still being able to apply unique configurations where needed.

This feature helps standardize application and container deployment across clusters, making the onboarding of new edge regions easy and consistent. This allows the day-to-day operations work to scale non-linearly, making the most of each engineer's time. The feature allows administrators to manage a large number of edge locations without additional work.

As you saw in **Figure 10**, the "core" registries and catalogues are synced with remote locations, which cache this information to reduce bandwidth and remove any connectivity dependencies they might have to the core data centers.

As a result, deploying and scaling applications at the edge isn't dependent on the core data centers, but can be handled locally while still receiving periodic policy-based changes to keep configurations consistent and in sync with the centrally defined policies.



Registry for apps, cluster components can be optionally synced with other registries for availibility
Profile based system makes it easier to create similar looking 'remote' clusters along with apps deployed on them
pods
Calico nw policies & encryption
Pods & VMs
Server mesh for inter-cluster comms
Pods & VMs
K8s clusters (core)
worker(s) at edge location
worker(s) at edge location
worker(s) at edge location
Site-registry and apps cache
K8s masters
IoT/edge devices
Registry, Inv. Sync
Platform9 Mgmt traffic
Central Registry & App catalog
Profile based cluster deployment
Inventory Svc
Qbert
PMK management plane - Region 1
K8s Clusters at Remote locations
KubeVirt
~2 km
Federated regions
Inventory svc stores a cache of the objects: pods, svc etc. for better visibility
PMK management plane - Region 2
Devices
Edge Locations
Local/Regional
Core Data Centers/Cloud

Figure 11: Distributed Edge Platform deep dive

The architecture deep dive in **Figure 11** shows the entire architecture, from endpoints to core data centers. We'll dive into each area of this diagram in the next sections.

Core Data Centers Are the Brains of the Operation

The core data centers run the management plane, central container registry, and App Catalog, as well as inventory services and policy-based deployment and configuration tools for cluster management.

The profile-based system makes it easier to create identical remote clusters and to deploy applications to remote clusters consistently, keeping configuration drift to a minimum and ensuring maximum application compatibility.

Core data centers are federated across regions for consistent deployment in large scenarios. A single management plane region is able to handle up to 100 Kubernetes clusters with up to 100 nodes per cluster. The central inventory service caches a representation of the remote sites for better visibility.

Consistent Networking and Granular Security

Networking and security policies are deployed consistently from core to edge, making sure application deployments are secure and compliant. Compliance can be enforced even in untrusted physical environments through service mesh and micro-segmentation technologies. A distributed Kubernetes architecture combines all of these locations—public cloud VPCs, core data centers, edge data centers, and edge locations—in a single, interconnected mesh.

Local Data Centers Ensure Independent Operation

Each local data center can deploy and scale applications independently of the core data centers, creating geographically separated "cells" that can run without a continuous connection to the core data centers. Each local or regional location runs one or more Kubernetes master cluster nodes, which manage worker nodes across that location. This way, each edge location runs only the absolute necessary hardware—often low-cost, low-power, and low-maintenance machines—while regional hubs coordinate application deployment and resilience across their local region.

Applying This Architecture to Retail, Manufacturing, and SaaS

A large number of use cases can benefit from the distributed Kubernetes architecture described in this chapter. Many enterprises see the same growing need for edge computing as their revenue streams become more and more digitally focused. The enterprises see more need for connecting cloud services to where their users are, regardless of whether those users are consumers, other businesses, or IoT-enabled devices.

Retailers are innovating and transforming the shopping experience to be seamless across online and in-store visits. This requires connecting cloud and on-premises locations to work together, offering buyers a consistent experience. The new architecture unlocks new revenue streams and increases existing value streams by accelerating the rollout of digital store concepts and by increasing in-store automation and the use of innovative retail software.

The distributed Kubernetes architecture helps retailers deploy new stores quickly and consistently. It reduces per-store operational IT costs both for onboarding new stores and for continuous operation, including lifecycle management and seamless software upgrades of running containers.

Manufacturers are replacing Wi-Fi and legacy wired networks with 5G wireless connectivity for factory floors and manufacturing plants to connect IoT and edge devices. That means they need to connect 5G endpoints with worker nodes for data processing, central management, and factory process engineering. Putting the compute power at the endpoints minimizes costs and operational burden.

Similarly, SaaS and independent software vendors (ISVs) are starting to use edge computing to improve their users' experience, decrease time to market, and reduce costs and operational burdens. The Platform9 Managed Kubernetes distributed architecture helps them deploy their software to edge locations, decreasing latency and bandwidth requirements, while consistently deploying the application to many locations simultaneously.

Especially with many single-tenant application deployments across edge locations (such as customer sites), upgrades and other operational and lifecycle tasks are automated and consistently executed across the board. This reduces support costs and effort, and allows developers to spend more time on delivering new features and software.

Solve Your Kubernetes-at-the-Edge Challenges

In this Gorilla Guide, you've seen how Platform9 Managed Kubernetes is suitable for any kind of application at the edge across telco, retail, manufacturing, enterprise, ISV, and SaaS use cases. Its ability to manage deployments on any infrastructure based on centralized policies is a huge time saver. It lowers time-to-fix when outages occur, lowers support costs, and improves customer satisfaction.

Make sure to look at Platform9 Managed Kubernetes and its distributed architecture to solve your Kubernetes-at-the-edge challenges. Try it for free[7] or download the updated buyer's guide.[8]

Thanks for reading, and stay safe out there!

[7] https://platform9.com/managed-kubernetes/

[8] https://platform9.com/resource/buyers-guide-to-enterprise-kubernetes-solutions-a-comparison-of-openshift-vs-vwware-tanzu-vs-google-anthos-vs-rancher-vs-platform9-managed-kubernetes/

ABOUT PLATFORM9

Platform9[1] enables freedom in cloud computing for enterprises that need the ability to run pri [illegible] hybrid clouds. Our SaaS-managed cloud platform [illegible] erate and scale clouds based on open source st [illegible] rnetes and OpenStack, while supporting any [illegible] ning on-premises or at the edge.

Replace with the latest boilerplate below. I will send an email since this is missing the links

Platform9 is the leading Managed Kubernetes provider for private, edge, and hybrid clouds. With a mission to enable freedom in cloud computing, the company delivers cloud-native technologies with SaaS simplicity that are easy to operate and scale, while supporting broad cloud capabilities that run on any infrastructure. The company's SaaS management model has been proven to eliminate management complexity and costs through automated cluster upgrades, security patches, and fully-managed monitoring and troubleshooting across multiple points of presence, edge sites, and data centers with guaranteed SLAs. Headquartered in Mountain View, CA, Platform9 clients include enterprises such as Kingfisher Retail, Redfin, Cloudera, Yext, and Juniper Networks. For additional information please visit Platform9.com, refer to the Platform9 blog and download the recent Forrester Wave™: MultiCloud Container Development Platforms, Q3 2020 Report. Follow Platform9 on LinkedIn, Twitter, and Facebook.

[1] https://platform9.com/

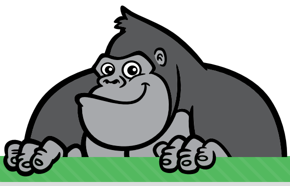
ABOUT ACTUALTECH MEDIA

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.



If you're an IT marketer and you'd like your own custom Gorilla Guide® title for your company, please visit https://www.gorilla.guide/custom-solutions/