ActualTech Media

PLATFORM9

10 Considerations for Running Kubernetes at Scale

CONTENTS

IN THIS PAPER

Kubernetes deployments can scale to support services and applications that run in a broad array of environments, from multi-cloud deployments to resource-constrained edge computing environments. But as businesses look to save money by optimizing infrastructural resources, the process of deploying thousands of microservices over geographically distributed servers can become increasingly complicated.

If your company is considering a large-scale Kubernetes deployment, the 10 considerations detailed in this paper is where you need to begin.

INTRODUCTION

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 doesn't manifest the issues you're likely to encounter when you start to run hundreds of nodes in a cluster.

> Manual intervention to scale resources isn't a viable option when working with large deployments and dynamic workloads.

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

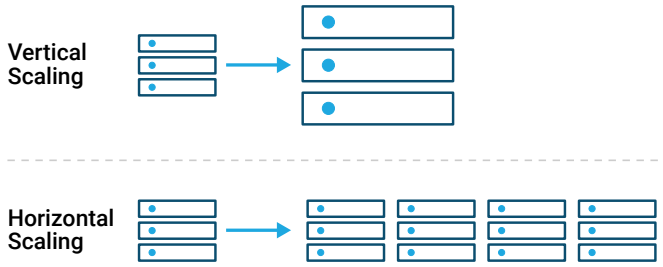

Figure 1: Horizontal vs. Vertical scaling of nodes

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.

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

AVAILABILITY

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

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

The repeatable processes for developer environments may be different from the repeatable processes used in production environments. Site reliability engineers (SREs), for example, may have a specific set of design principles they apply to production environments. For example, there may be different levels of health checking, monitoring, and alerting. Service-level agreements (SLAs) will likely be different, as well. Also, developers will likely have different needs from SREs. For example, developers shouldn't have administrative access to a production cluster, but they should have administrative privileges to a cluster in their development environment, rather than depend on others to configure and maintain it.

UPGRADABILITY

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

Consider a typical scenario of how an enterprise might start using Kubernetes. A group of developers and a business sponsor decide to develop a proof of concept (PoC) system on a small cluster. The developers want to show results as fast as possible, so they choose the easiest installation method to get Kubernetes up and running. Next, they incrementally add other services, such as a database, which increases complexity to the overall system. Wanting to show a realistic use case, the developers then deploy an application. The PoC is well received and decision-makers agree to make the service available in production.

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

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.

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.

Naturally, 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 downtimes.
- Use a container-friendly OS (like Alpine or CoreOS) so that they're 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.

> 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.

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 in order to use Kubernetes effectively, including Promoetheus, Fluentd, Istio, and Jeager.

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.

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

> As the size of Kubernetes clusters grows, it becomes imperative to define policies that allow you to meet regulatory requirements with minimal manual intervention.

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.

PLANNING FOR KUBERNETES AT SCALE

Kubernetes deployments can scale to support services and applications that run in a broad array of environments, from multi-cloud deployments to resource-constrained edge computing environments. As you plan for deploying Kubernetes at scale, keep in mind the 10 considerations outlined here to help improve the capabilities of your deployments without sacrificing long-term scalability.