weaveworks

Your Guide to a Production Ready Kubernetes Cluster

WHITEPAPER

Kubernetes is open source software for deploying, scaling and managing containerized applications. As an orchestrator, it handles the work of scheduling containers on a cluster and also manages the workloads to ensure they run as you intended.

Because Kubernetes was designed from the beginning with the idea of software development and operations working together, operations tasks and how they get performed is an integral component of the Kubernetes architecture and design. Almost everything in Kubernetes uses declarative constructs that describe how applications are composed, how they interact and how they are managed. This enables a significant increase in the operability and portability of modern software systems. It also allows developers to easily adopt GitOps workflows [1] into their development pipelines which can increase the velocity and reliability of feature deployments.

While the promise of increased productivity combined with a flexible, portable infrastructure is something to get excited about, most know that once you're in production some of these promises can fall short, especially if you are feeling your way through it all on your own.

So how do growing development and ops teams manage a production ready cluster? What does it even mean to be production ready and how do you know you are production ready?

FIVE STEPS TO PRODUCTION READINESS

Production readiness is a term you hear a lot, and depending on who you are talking to and what they are doing, it can mean different things. Carter Morgan[2], a Developer Advocate at Google sums it up as: "Your offering is production ready when it exceeds customer expectations in a way that allows for business growth."

Cluster production readiness is somewhat dependant on your use case and can be about making tradeoffs. Although a cluster can be production ready when it's good enough to serve traffic, many agree on a minimum set of requirements you need before you can safely declare your cluster ready for commercial traffic.

When creating a reliable production set-up, the following areas are important with some more important than others, depending on your use case.

1 Keep security vulnerabilities and attack surfaces to a minimum

Ensure that the applications you are running are secure and that the data you are storing, whether it's your own or your clients, is secured against attack. In addition to this, be aware of security breaches within your own development environments. And because Kubernetes is a rapidly growing open source project, you'll also need to be on top of the updates and patches so that they can be applied in a timely manner.

2 Maximize portability and scalability by tuning cluster performance

Running and managing applications anywhere is why Kubernetes is the number one orchestrator, as is its ability to self-heal nodes, autoscale infrastructure and adapt to your expanding business. Most want all of the benefits of self-healing and scalability without taking a performance hit.

3 Increase deployment velocity through automated CICD pipelines

With your infrastructure in place, an automated continuous deployment and delivery pipeline allows your development team to maximize their velocity and improve productivity through increased release frequency and repeatable deployments.

4 Recover faster from disaster with GitOps best practices

Ensure that you have high availability which means that if you have a failure your cluster can recover automatically. In the case of complete cluster meltdown with the adoption of GitOps best practices, your team should be able to recover it and any application data without any adverse effects.

5 Mitigate deployment risk with built in observability

Observability is not only about being able to monitor your system, but it's also about having a high-level view into your running services so that you can make decisions before you deploy. To achieve true observability you need the processes and tools in place to act on that monitoring through timely incident alerts.

MULTI-TEAM DISTRIBUTED APP DEVELOPMENT

Multiple development teams ≠ Multiple Kubernetes clusters

Kubernetes challenges the way we have traditionally thought about development environments. It has also changed the way we implement and share them with different teams. Working in teams no longer requires that everyone use the same language, or even that deployment pipelines be set up in a specific way. You may not need three isolated environments: Dev, QA and Staging that are all kept in sync with one another and with every developer on the team.

How many clusters do you need?

An obvious setup for most development shops is to create one cluster that runs production traffic and another one for staging and then possibly a third for QA - depending on your requirements.

Both the staging and the test clusters need to reflect production, and then of course there's your development environments which many think also need its own cluster. As you can see, this can quickly get out of hand, and if you are not careful, you can end up with a lot of clusters in your development environments. Multiple clusters not only adds a lot more overhead and maintenence, but they can also incur a large expense both monetarily and time-wise if you are not careful.

SHARING ENVIRONMENTS BETWEEN TEAMS

Kubernetes has a built-in feature that allows you to share a cluster among different environments and between different projects on separate teams. While you can split a single cluster into different environments and share them across your team, we're not necessarily advocating that you should only use one Kubernetes cluster.

There are many cases where it makes sense to run more than one cluster. It's important to note that with Kubernetes, multiple clusters doesn't have to be the default approach. You can start with at least one Kubernetes cluster, possibly two, and then share those clusters among different development teams all working in different environments.

To efficiently and securely develop your application on Kubernetes in a team, these are the features you need to know about:

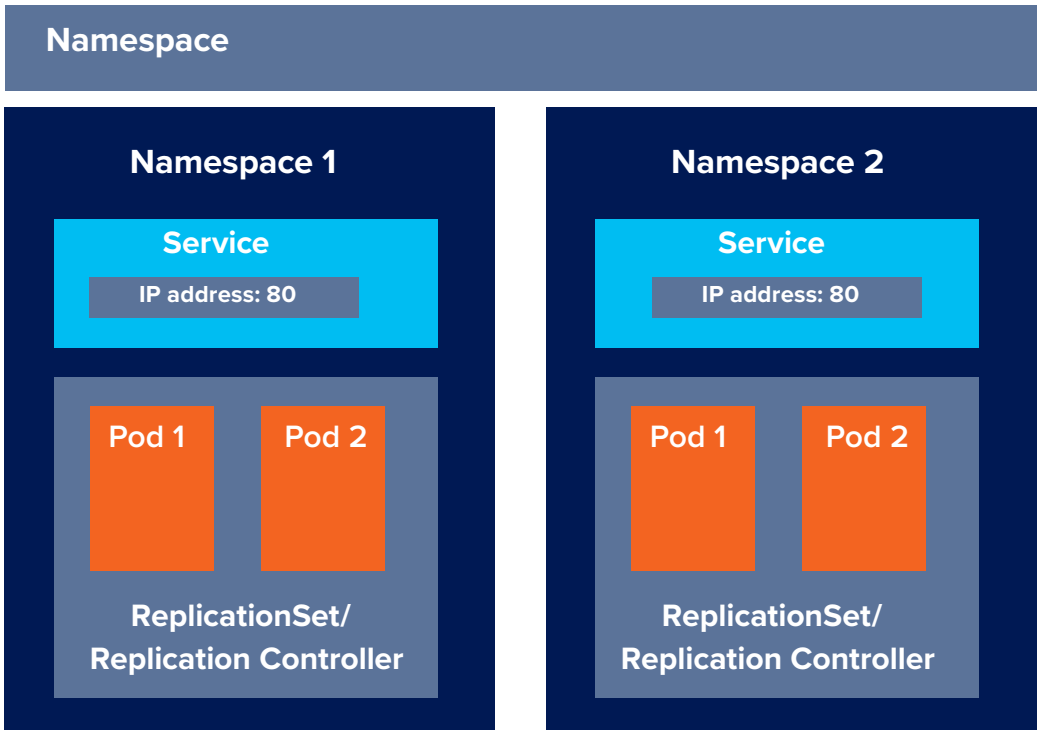
- Namespaces
- Role Based Access Control (RBAC)
- Network Policies

Namespaces

Namespaces[3] are a very basic concept in Kubernetes. They are simple to create and delete and are used to subdivide your cluster so that multiple teams can work on it. This not only saves you server costs, but it can also increase quality by providing a convenient platform for integration testing and other smoke tests before deploying to production.

See the How and Why of Namespaces[4] for information on creating namespaces.

When you create namespaces on your cluster it looks a little bit like this.



The two namespaces shown here provide a virtual separation where different services can run alongside each other on the same node. They can still communicate with one another if the service name is prefaced; where the service name will get resolved to the service that is local to the namespace[5].

A namespace provides a virtual separation only, and as mentioned, it is a great way for different teams to work together on one application[6].

For stronger security though, you will want to implement Role Based Access Control (RBAC) at the user level, and Network Policies for service to service permissions.

Role Based Access Control (RBAC)

RBAC allows even more fine-grained control over who sees what in a cluster. It provides an additional layer of security for running multiple environments across teams. With RBAC you can create generic roles like ‘developer’ or ‘admin’ and then assign permissions to them. This then allows you to specify what roles can access which clusters and whether they have read and write access to them.

Roles can also be applied to an entire namespace so that you can define who is allowed to create, read or write to pods within that namespace.

Network Policies

Applying network policies even further defines application security. A namespace in Kubernetes is simply a virtual separation and with only namespaces set, applications can still easily communicate with each other. But what if you need to restrict which services or even what namespaces can communicate with one another? To do this, you’ll need to implement a network policy.

Network policy is an add-on feature that is implemented by the Kubernetes CNI or pod networking, see Weave Net for NetworkPolicy[7].

Resource Constraints and Quotas

When sharing a cluster with multiple environments or multiple teams, another important consideration is resource allocation. Start early with setting and testing constraints and quotas. Without specifying these, everything in your cluster will still run properly, but you may get a big, unpleasant surprise when you get a sudden load on one of your containers and you haven’t set the correct resource quota for it[8] [9].

PROTECTING YOUR CLUSTER WHILE IN PRODUCTION

If you have even stronger security requirements, you can implement complete pod to pod SSL traffic. SSL implementation is not native in Kubernetes and will require planning and implementation time by your team. If you want to include SSL, you may consider implementing a secure service mesh like Istio to do this for you[10].

More on cluster security

Once you are running in production and accepting end-user traffic, you’ll need to fully lockdown and secure your cluster by doing the following:

- **Control access to the cluster API** - This means implementing TLS for all API traffic and also applying API authentication and authorization for all calls.
- **Control access to the kubelet** - Because the Kubelet provides https endpoints that grant control over pods and nodes, all production clusters should enable authentication and authorization over the kubelet.

- **Control user and workload capabilities** - This is where you need to limit the resource usage by setting quotas and limits, as well as specifying container privileges, restricting network access, API access and node access to pods.
- **Protect cluster components from compromise** - Lastly you need to restrict access to etcd, enable full audit logging, start a key rotating schedule, encrypt secrets and set up standards and review for third-party applications.

Kube-bench from Aqua Security[11] is a tool that can be run on your cluster to check for vulnerabilities and areas that need hardening.

There are many good topics on security to consult:

- Securing a cluster[12]
- Security vulnerability reports[13]

DATA STORES AND KUBERNETES

Many new to Kubernetes ask “How should I handle my data stores with Kubernetes?”

If you’re running a data store such as MongoDB or MySQL, you will want the data to be persistent, since containers lose their data when they restart. This of course is fine for stateless applications and services, but it is not adequate for a persistent data store.

For services that rely on state, Kubernetes has the concept of volumes that work with persistent data. With a volume, you can back a variety of implementations, including files on the host machines, AWS Elastic Block Store (EBS), and NFS.

This is a topic that requires some thought and planning, and is very much dependant on your application and use case.

You will need to read up on Kubernetes volumes and persistent volumes[14] and also be aware of the special circumstances around managing StatefulSets in Kubernetes and on how to handle database schemas during Kubernetes rollouts[15].

How to implement rollout strategies, readiness probes and liveness probes, and select a database migrations library needs to be addressed. Lastly applying some good engineering practices that can save the day for when something goes wrong and you need to roll things back are also important considerations when dealing with storage in Kubernetes.

DEFINING CICD PIPELINES FOR KUBERNETES

Solid engineering practices are a reason for why you need Continuous Integration/Continuous Delivery (CICD) pipelines. These are a must have in order to increase code quality as well as feature development velocity. It is the preferred method for deploying your services to Kubernetes and some would even argue that no software should be deployed into production without going through a CI/CD pipeline.

Deployment pipelines, if done correctly, also solves the problem of having to give your entire development team kubectl access on your cluster which as a general practice you shouldn't allow[16].

GitOps Workflows for Velocity and Reliability

Our management and operations platform for Kubernetes, Weave Cloud, is designed to help you automate the deployment of new features and increase velocity and reliability for development teams.

Challenges with Velocity

The most common problem faced by development teams is the risk of failure. Imagine that you are about to release a large deployment into production with a significant backlog of changes. What happens if you release that deployment into production, and it doesn't go as planned?

Chances are you'll end up in a whole world of pain as you try to sort through the changes in order to roll back. To mitigate risk, you could add some manual steps before you deploy, but then you risk a divergence between your development environments and your actual production environments. You can see how you can end up with too many processes and extra systems around deployments.

Ultimately, what you need is automation that drives the repeatability as well as the speed of your deployments.

Challenges with Reliability

When it comes to reliability, with Kubernetes you are a much better off than you were in the past. This is because Kubernetes helps you by delivering your services more reliably. Kubernetes understands the lifecycle of containers; and it can do things like autoscaling as well as reacting to the health of the system by self-healing.

Even though it can do all of these things, Kubernetes still lacks in some tooling. For example when you are running a large distributed system, external dependencies such as RDS databases or other services from public cloud providers can be problematic to monitor.

Also if you need to debug an issue, those tools need to be container and Kubernetes aware. Traditional debugging tools don't exactly understand how to follow a transaction or an API call through a distributed application. By implementing observability you can ensure reliability so that you can fully understand the problem and find a resolution.

INCREASE VELOCITY AND RELIABILITY WITH GITOPS AND WEAVE CLOUD

Almost everything in Kubernetes can be configured declaratively. What this means is that you can specify exactly what a workload is and how it's going to run in the cluster and then keep those declarations in a repo like Git.

With your declarative infrastructure in Git, Weave Cloud can compare Git with what's running in production. When there's a divergence, it automatically brings the system in accordance with production. Since deployments are automated you don't have to log in and make manual changes to your development or production clusters.

A second advantage is that you can use the development tools and workflows that you are familiar with. In essence, developers can do operations by pull request. At the same time, because this is all done through Git, it leaves you with a convenient audit trail to meet SOC2 compliance.

Confidently deploy reliable workloads

How can you be sure that your system is reliable and that new deployments made by your team aren't going to cause a cluster meltdown?

Solutions that can monitor Kubernetes need to manage more services and servers than ever before. Collecting data from an environment composed of so many moving parts is complex. The open source project, Prometheus was built specifically to monitor applications and microservices running in containers at scale. Most Kubernetes experts would agree that Prometheus is the de facto standard for monitoring and alerting on your Kubernetes workloads.

But Prometheus is capable of monitoring much more than your infrastructure; it can also analyze long-term trends in your software as it runs in production as well as debug performance in real-time[17].

With Weave Cloud, we not only host Prometheus for you, but we've also integrated its powerful monitoring and real-time performance analysis right into our platform. With Weave Cloud's real-time workload dashboards, you can increase reliability by running real-time performance checks against your cluster before a workload gets deployed.

Weave Cloud: an operations and management platform

Weave Cloud is an operations and management platform for development and DevOps teams. It minimizes the complexity of operating workloads in Kubernetes by providing automated continuous delivery pipelines, observability and monitoring. You can plug it into any Kubernetes installation on any of the public clouds. It can also integrate with and monitor external services such as RDS on Amazon EKS.

Production Ready Kubernetes with Weaveworks

We can help you accelerate your Kubernetes journey with our subscription service that supports installing production-grade Kubernetes on-premise, in AWS and GCP from development to production. For the past 3 years, Kubernetes has been powering Weave Cloud, our operations and management offering, so we're taking our knowledge and helping teams embrace the benefits of cloud native tooling. We've focused on creating GitOps workflows - our approach uses developer-centric tooling (e.g. Git) and a tested approach to help you install, set-up, operate and upgrade Kubernetes. Contact us for more details.

Learn more about our Production Grade Kubernetes professional services and sign up for a free Weave Cloud account to try it out for yourself.

REFERENCES AND FURTHER RESEARCH

1 GitOps What you need to know? Buehrle, A. (July 2018)

2 Carter Morgan on Production Ready

3 When to use Multiple Namespaces, Kubernetes

4 How and Why of Namespaces, Ajitesh, K. (January 2018)

5 Understanding Namespaces and DNS, Kubernetes

6 Share a Cluster with Namespaces, Kubernetes

7 Securing Microservices in Kubernetes , Harrison, A., (Nov., 2016)

8 Configure Memory and CPU Quotas for a Namespace, Kubernetes

9 Managing Compute Resources for Containers, Kubernetes

10 The rise of the Istio service mesh, Rettori, D. (May 2018)

11 The Kubernetes Bench for Security, Aquasecurity

12 Securing a Cluster, Kubernetes

13 Kubernetes Security and Disclosure Information, Kubernetes

14 Configure a Pod to Use a PersistentVolume for Storage, Kubernetes

15 How to Correctly Handle DB Schemas During Kubernetes Rollouts, Carre, M. (August 2018)

16 The GitOps Pipeline - Part 2, Richardson, A. (August 2017)

17 Monitoring Kubernetes with Prometheus, Buehrle, A. (September 2017)