THE KUBERNETES BOOK

2024 Edition

Nigel Poulton
& Pushkar Joglekar

The Kubernetes Book

Nigel Poulton

This book is for sale at http://leanpub.com/thekubernetesbook

This version was published on 2024-02-03

ISBN 9781916585195

Tweet This Book!

Please help Nigel Poulton by spreading the word about this book on Twitter!

The suggested tweet for this book is:

I just bought The Kubernetes Book from @nigelpoulton and can't wait to get into this!

The suggested hashtag for this book is #kubernetes.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

#kubernetes

Education is about inspiring and creating opportunities. I hope this book, and my video training courses, inspire you and create lots of opportunities!

A huge thanks to my family for putting up with me. I'm a geek who thinks he's software running on midrange biological hardware. I know it's not easy living with me.

Thanks to everyone who watches my Pluralsight and A Cloud Guru training videos. I love connecting with you and appreciate all the feedback I've had over the years. This feedback is what inspired me to write this book. I think you'll love it, and I hope it helps drive your career forward.

@nigelpoulton

Contents

0: Preface

Kubernetes is developing fast, so I update the book every year. And when I say *update*, I mean real updates — I review every word and every concept, and test every example against the latest versions of Kubernetes. I'm 100% committed to making this the best Kubernetes book in the world.

As an author, I'd love to write a book and never touch it again for five years. Unfortunately, a two-year-old book on Kubernetes could be dangerously out of date.

Editions Paperbacks, hardbacks, eBooks, audio, and translations

The following editions of the book are available:

- **Paperback:** English, Simplified Chinese, Spanish, Portuguese
- **Hardback:** English
- **eBook:** English, Russian, Spanish, Portuguese

eBook copies are available on Kindle and from Leanpub.

The following collector's editions are available. Each has a themed front cover, but the content is exactly the same as the regular English-language edition.

- Klingon paperback
- Borg hardback
- Sterfleet paperback

The sample app and GitHub repo

There's a GitHub repo with all the YAML and code used throughout the book.

You can clone it with the following command. You'll need **`git`** installed. This will create a new folder in your current working directory called **`TheK8sBook`** with all the files you need to follow the examples.

```
$ git clone https://github.com/nigelpoulton/TheK8sBook.git
```

Don't stress if you've never used git. The book walks you through everything you need to do.

Windows users

Almost all of the commands in the hands-on sections work on Linux, Mac, and Windows. However, a small number require slight changes to work on Windows. Whenever this is the case, I explain what you need to do to make them work on Windows.

However, to prevent myself from repeating the same thing too often, I don't always tell Windows users to replace backslashes with backticks for linebreaks. With this in mind, Windows users should do one of the following every time the book splits a command over multiple lines using backslashes:

- Remove the backslash and run the command on a single line
- Replace the backslash with a backtick

All other changes are explained in full every time.

Responsible language

The book follows *Inclusive Naming Initiative* (inclusivenaming.org) guidelines, which attempt to avoid harmful terms and promote responsible language.

1: Kubernetes primer

This chapter gets you up-to-speed with the basics and background of Kubernetes and is divided as follows:

- Important Kubernetes background
- Kubernetes: the Operating System of the cloud

Important Kubernetes background

Kubernetes is an orchestrator of containerized cloud-native microservices apps.

That's a lot of jargon, so let's explain things.

Orchestration

An *orchestrator* is a system or platform that deploys applications and dynamically responds to changes. For example, Kubernetes can:

- Deploy applications
- Scale them up and down based on demand
- Self-heal them when things break
- Perform zero-downtime rolling updates and rollbacks
- Lots more

The best part is that it does all of this without **you** having to get involved. You need to configure a few things in the first place, but once you've done that, you sit back and let Kubernetes work its magic.

Containerization

Containerization is the process of packaging an application and dependencies as an image and then running it as a container.

It can be useful to think of containers as the next generation of virtual machines (VM). Both are ways of packaging and running applications, but containers are smaller, faster, and more portable.

Despite these advantages, containers haven't replaced VMs, and it's common for them to run side-by-side in most cloud-native environments. However, containers are the first-choice solution for most new applications.

Cloud native

Cloud-native applications possess cloud-like features such as *auto-scaling, self-healing, automated updates, rollbacks,* and more.

Simply running a regular application in the public cloud **does not** make it *cloud-native.*

Microservices

Microservices applications are built from many small, specialized, independent parts that work together to form a useful application.

Consider an e-commerce app with the following six features:

- Web front-end
- Catalog
- Shopping cart
- Authentication
- Logging
- Store

To make this a *microservices app*, you design, develop, deploy, and manage each feature as its own small application. We call each of these small apps a *microservice*, meaning this app will have six microservices.

This design brings huge flexibility by allowing all six microservices to have their own small development teams and their own release cycles. It also lets you scale and update each one independently.

The most common pattern is to deploy each microservice as its own container. This means one or more web front-end containers, one or more catalog containers, one or more shopping cart containers, etc. Scaling any part of the app is as simple as adding or removing containers.

Now that we've explained a few things, let's re-write that jargon-filled sentence from the start of the chapter.

The original sentence read; *"Kubernetes is an orchestrator of containerized cloud-native microservices apps."* We now know this means: *Kubernetes deploys and manages applications that are packaged as containers and can easily scale, self-heal, and be updated.*

That should clarify some of the main industry jargon. But don't worry if some of it still needs to be clarified; we'll cover everything again in much more detail throughout the book.

Where did Kubernetes come from

Kubernetes was developed by a group of Google engineers partly in response to Amazon Web Services (AWS) and Docker.

AWS changed the world when it invented modern cloud computing, and everyone needed to catch up.

One of the companies catching up was Google. They'd built their own cloud but needed a way to abstract the value of AWS **and** make it as easy as possible for customers to get off AWS and onto their cloud. They also ran production apps, such as *Search* and *Gmail*, on billions of containers per week.

At the same time, Docker was taking the world by storm, and users needed help managing explosive container growth.

While all this was happening, a group of Google engineers took the lessons they'd learned using their internal container management tools and created a new tool called ***Kubernetes***. In 2014, they open-sourced Kubernetes and donated it to the newly formed *Cloud Native Computing Foundation (CNCF)*[1].

At the time of writing, Kubernetes is ~10 years old and has experienced incredible growth and adoption. However, at its core, it still does the two things Google and the rest of the industry need:

1. It abstracts infrastructure (such as AWS)
2. It simplifies moving applications between clouds

These are two of the biggest reasons Kubernetes is important to the industry.

[1] https://www.cncf.io

Kubernetes and Docker

All of the early versions of Kubernetes shipped with Docker and used it as its runtime. This means Kubernetes used Docker for low-level tasks such as creating, starting, and stopping containers. However, two things happened:

1. Docker got bloated
2. People created lots of Docker alternatives

As a result, the Kubernetes project created the *container runtime interface (CRI)* to make the runtime layer *pluggable*. This means you can pick and choose the best runtimes for your needs. For example, some runtimes provide better isolation, whereas others provide better performance.

Kubernetes 1.24 finally removed support for Docker as a runtime as it was bloated and overkill for what Kubernetes needed. Since then, most new Kubernetes clusters ship with *containerd* (pronounced *"container dee"*) as the default runtime. Fortunately, containerd is a stripped-down version of Docker optimized for Kubernetes, and it fully supports applications containerized by Docker. In fact, Docker, containerd, and Kubernetes all work with images and containers that implement the *Open Container Initiative (OCI)*[2] standards.

Figure 1.2 shows a four-node cluster running multiple container runtimes.

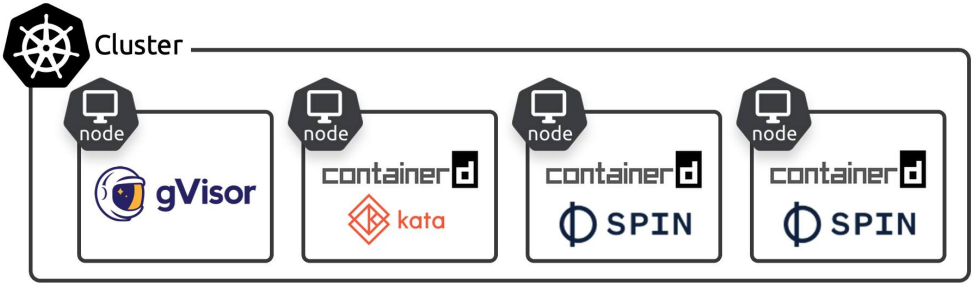

Figure 1.2

Notice how some of the nodes have multiple runtimes. Configurations like this are fully supported and increasingly common. You'll work with a configuration like this in Chapter 14 when you deploy a WebAssembly (Wasm) app to Kubernetes.

[2] https://opencontainers.org

What about Docker Swarm

In 2016 and 2017, Docker Swarm, Mesosphere DCOS, and Kubernetes competed to become the industry standard container orchestrator. Kubernetes won.

However, Docker Swarm remains under active development and is popular with small companies wanting a simple alternative to Kubernetes.

Kubernetes and Borg: Resistance is futile!

We already said that Google has been running containers at massive scale for a very long time. Well, orchestrating these billions of containers were two in-house tools called *Borg* and *Omega*. So, it's easy to make the connection with Kubernetes — all three orchestrate containers at scale, and all three are related to Google.

However, Kubernetes is **not** an open-source version of Borg or Omega. It's more like Kubernetes shares its DNA and family history with them.

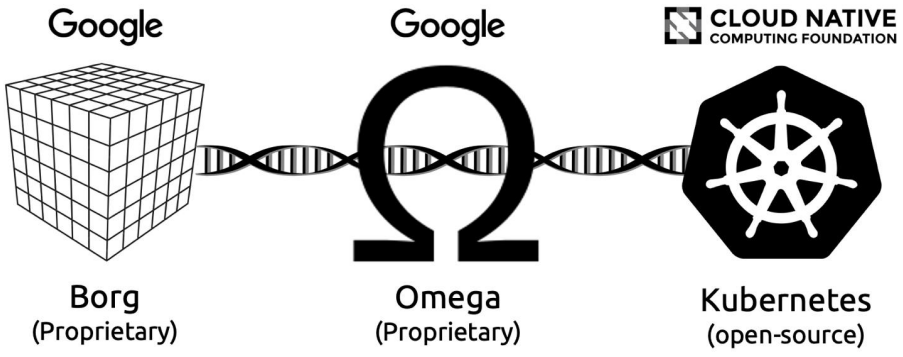

Figure 1.3 - Shared DNA

As things stand, Kubernetes is an open-source project owned by the CNCF. It's licensed under the Apache 2.0 license, version 1.0 shipped way back in July 2015, and at the time of writing, we're already at version 1.29 and averaging three new releases per year.

Kubernetes — what's in the name

Most people pronounce Kubernetes as *"koo-ber-net-eez"*, but the community is very friendly, and people won't mind if you pronounce it differently.

The word Kubernetes originates from the Greek word for *helmsman* or the person who steers a ship. You can see this in the logo, which is a ship's wheel.

Figure 1.4 - The Kubernetes logo

Some of the original engineers wanted to call Kubernetes *Seven of Nine* after the famous *Borg* drone from the TV series Star Trek Voyager. Copyright laws wouldn't allow this, so they gave the logo *seven* spokes as a subtle reference to Seven of Nine.

One last thing about the name. You'll often see it shortened to *K8s* and pronounced as "kates". The number 8 replaces the eight characters between the "K" and the "s".

Kubernetes: the operating system of the cloud

Kubernetes is the de facto platform for cloud-native applications, and we sometimes call it *the operating system (OS) of the cloud.* This is because Kubernetes abstracts the differences between cloud platforms the same way that operating systems like Linux and Windows abstract the differences between servers:

- Linux and Windows *abstract* server resources and *schedule* application processes
- Kubernetes *abstracts* cloud resources and *schedules* application microservices

As a quick example, you can schedule applications to Kubernetes without caring if it's running on AWS, Azure, Civo Cloud, GCP, or your on-premises datacenter. This makes Kubernetes a key enabler for:

- Hybrid cloud
- Multi-cloud
- Cloud migrations

In summary, Kubernetes makes it easier to deploy to one cloud today and migrate to another cloud tomorrow.

Application scheduling

One of the main things an OS does is simplify the scheduling of work tasks.

Computers are complex collections of hardware resources such as CPU, memory, storage, and networking. Thankfully, modern operating systems hide most of this and make the world of application development a far friendlier place. For example, how many developers need to care which CPU core, memory DIMM, or flash chip their code uses? Most of the time, we leave it up to the OS.

Kubernetes does a similar thing with clouds and datacenters.

At a high level, a cloud or datacenter is a complex collection of resources and services. Kubernetes can abstract a lot of these and make them easier to consume. Again, how often do you need to care about which compute node, which failure zone, or which storage volume your app uses? Most of the time, we're happy to let Kubernetes decide.

Chapter summary

Kubernetes was created by Google engineers based on lessons learned running containers at hyper-scale for many years. It was donated to the community as an open-source project and is now the industry standard platform for deploying and managing cloud-native applications. It runs on any cloud or on-premises datacenter and abstracts the underlying infrastructure. This allows you to build hybrid clouds, as well as migrate on, off, and between different clouds. It's open-sourced under the Apache 2.0 license and is owned and managed by the Cloud Native Computing Foundation (CNCF).

Don't be afraid of all the new terminology. I'm here to help, and you can reach me at any of the following:

- Twitter: @nigelpoulton
- LinkedIn: linkedin.com/in/nigelpoulton/
- Mastodon: @nigelpoulton@hachyderm.io
- Web: nigelpoulton.com
- Email: tkb@nigelpoulton.com

2: Kubernetes principles of operation

This chapter introduces you to major Kubernetes technologies and prepares you for upcoming chapters. You're not expected to be an expert at the end of this chapter.

We'll cover all of the following:

- Kubernetes from 40K feet
- Control plane nodes and worker nodes
- Packaging apps for Kubernetes
- The declarative model and desired state
- Pods
- Deployments
- Services

Kubernetes from 40K feet

Kubernetes is both of the following:

- A cluster
- An orchestrator

Kubernetes: Cluster

A *Kubernetes cluster* is one or more *nodes* providing CPU, memory, and other resources for use by applications.

Kubernetes supports two node types:

- Control plane nodes
- Worker nodes

Both types can be physical servers, virtual machines, or cloud instances, and both can run on ARM and AMD64/x86-64. Control plane nodes must be Linux, but worker nodes can be Linux or Windows.

Control plane nodes implement the Kubernetes intelligence, and every cluster needs at least one. However, you should have three or five for high availability (HA).

Every control plane node runs every control plane service. These include the API server, the scheduler, and the controllers that implement cloud-native features such as self-healing, autoscaling, and rollouts.

Worker nodes are for running user applications.

Figure 2.1 shows a cluster with three control plane nodes and three workers.

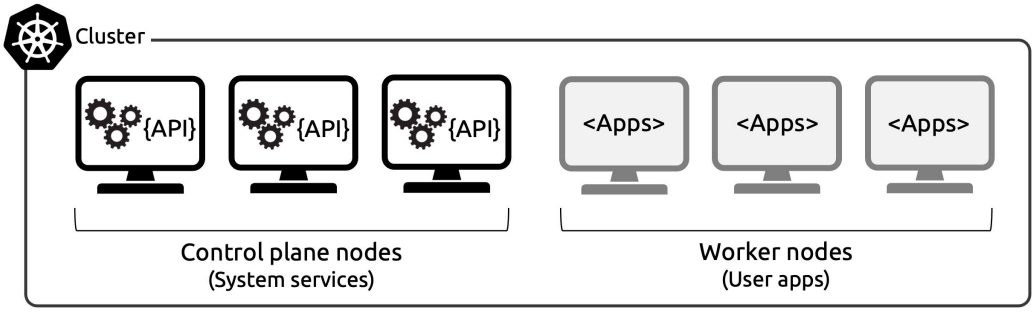

Figure 2.1

It's common to run user applications on control plan nodes in development and test environments. However, many production environments restrict user applications to worker nodes so that control plane nodes can focus entirely on cluster operations.

Control plane nodes can also run user applications, but you should probably force user applications to run on worker nodes in production environments. Doing this allows control plane nodes to focus on managing the cluster.

Kubernetes: Orchestrator

Orchestrator is jargon for a system that deploys and manages applications.

Kubernetes is the industry-standard orchestrator and can intelligently deploy applications across nodes and failure zones for optimal performance and availability. It can also fix them when they break, scale them when demand changes, and manage zero-downtime rolling updates.

That's the big picture. Let's dig a bit deeper.

Control plane and worker nodes

We already said a Kubernetes cluster is one or more *control plane nodes* and *worker nodes*.

Control plane nodes have to be Linux, but workers can be Linux or Windows.

Almost all cloud-native apps are Linux and will run on Linux worker nodes. However, you'll need one or more worker nodes running Windows if you have cloud-native Windows apps. Fortunately, a single Kubernetes cluster can have a mix of Linux and Windows worker nodes, and Kubernetes is intelligent enough to schedule apps to the correct nodes.

The control plane

The *control plane* is a collection of system services that implement the brains of Kubernetes. It exposes the API, schedules tasks, implements self-healing, manages scaling operations, and more.

The simplest setups run a single control plane node and are best suited for labs and testing. However, as previously mentioned, you should run three or five control plane nodes in production environments and spread them across availability zones for high availability, as shown in Figure 2.2

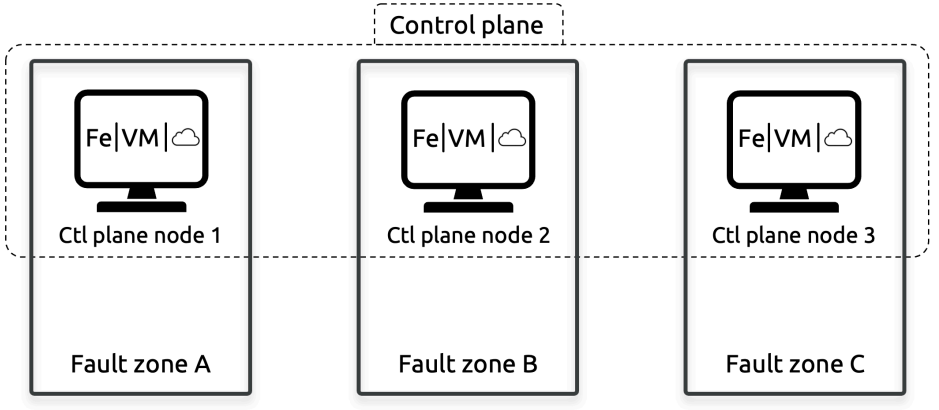

Figure 2.2 Control plane high availability

As previously mentioned, it's sometimes considered a production best practice to run all user apps on worker nodes, allowing control plane nodes to allocate all resources to cluster-related operations.

Most clusters run every control plane service on every control plane node for HA.