O'REILLY®

Kubernetes and Cloud Native Associate (KCNA) Study Guide

In Depth Exam Prep and Practice

Early Release
RAW & UNEDITED

Adrian Gonzalez Sanchez
& Jorge Valenzuela Jiménez

Kubernetes and Cloud Native Associate (KCNA) Study Guide

In Depth Exam Prep and Practice

Adrian Gonzalez Sanchez and Jorge Valenzuela Jiménez

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

Kubernetes and Cloud Native Associate (KCNA) Study Guide
by Adrian Gonzalez Sanchez and Jorge Valenzuela Jiménez

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*https://oreilly.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Acquisitions Editor: John Devins
Development Editor: Sara Hunter
Production Editor: Gregory Hyman
Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Kate Dullea

April 2024:	First Edition

Revision History for the Early Release

2023-10-23:	First Release
2024-01-23:	Second Release
2024-03-25:	Third Release

See *http://oreilly.com/catalog/errata.csp?isbn=9781098138943* for release details.

978-1-098-13888-2

Table of Contents

CHAPTER 1

Introduction

A Note for Early Release Readers

With Early Release ebooks, you get books in their earliest form—the author's raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 1st chapter of the final book. Please note that the GitHub repo will be made active later on.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at *shunter@oreilly.com*.

KCNA. Interesting choice for a new technology certification acronym. Why? Well, when you search for this term for the first time via Google or Bing, most of the results will refer to a national news agency in Asia. But no, nothing to do with that. The Linux Foundation released the Kubernetes and Cloud Native Associate exam in 2021, and even if the acronym can be initially a bit confusing, the name is not at all random. The exam and certification name is a very clear and direct reference to the two main topics of study, and the starting point for millions of learners and adopters. This is, the increasingly rich cloud native ecosystem, and the central role of Kubernetes as key technology enabler.

It makes sense. Containerization is quickly becoming the new normal for big and small technology companies and developers around the world, and technologies such as Docker and Kubernetes are "nonofficial" standards for most of the adopters out there. A powerful movement that is not new. The Cloud Native Computing Foundation was created in 2015, but companies like Google have leveraged cloud native technologies since the early aughts. That said, it is a complex area of highly technical knowledge that requires a lot of practice, effort and upskilling. But it also requires the creation of new learning and training resources, easy-to-read documentation, and more simple ways for professionals to learn and then adopt cloud native tools. This exam is part of the equation for that to happen.

But hey, this is not really a hurdle. I still remember the first time I heard about cloud native. It was during a job interview, and I naively thought that my interviewer meant something like general cloud computing. We will discuss the difference (and relation) between these two terms later, but it is clear that I was missing something big. At that time, I was well versed on cloud technologies because of my data and AI projects, and I was of course aware of the existence of something called Kubernetes. But this old school telematics engineer had missed the first years of the container revolution and his main reference of virtualization techniques were the traditional VMWare-ish virtual machines. And new communities such as CNCF (Cloud Native Computing Foundation) were still unknown to me. A lot to catch up with…

However, the promise of something similar to virtual machines (VMs) but even "better", lighter, and super scalable was very tempting. In addition to that, my curious nature and willingness to learn helped me connect the cloud native dots with the building blocks of the data and AI ecosystem. "Do you mean that I can deploy resources as new iterations or different versions of AI models, whenever I need it, in a relatively automated way? Are you telling me that this is how big companies are doing it today?". I still remember the words from some friend in the videogames industry, who was leveraging containers to deploy AI models, several times a day, to release new features or to just test alternative model versions. Too new, too exciting, I couldn't let it go.

Fun was about to start, but I quickly realized that Kubernetes wasn't a simple topic. Technical knowledge of its architecture, networking, security, managed options from hyperscalers, different pieces from the same puzzle… This was the next step for my engineering degree in Spain, except that no professor explained this to us because it was just too early. Also, I had lost part of my command line ability, and abandoned my arduous exploration of Linux-related topics. Years before, I used to stay awake during the night to try to find a way to replicate in Linux Ubuntu some cool functionalities from Windows, participate in forums, try new open-source software, debug new tools… but most of these things were gone. Instead, I was just a cloud-enabled adopter looking forward to making sense of Kubernetes, Docker, and other topics.

I had to find a way to start my cloud-native journey (in parallel with a few other certification topics I have been "devouring" during the last years), so I took a pragmatic and kind of lazy approach to avoid the most technical parts, and to focus first on the cloud-native ecosystem and community. That decision led me to learn more about CNCF and its actors, the open source governance model, cool projects beyond the omnipresent Kubernetes, and the levels of maturity and incubation. Such a discovery. A community full of talent, willingness to teach and collaborate, existing resources, etc.

Besides that, my research helped me find amazing technical evangelists and industry experts, a great source of knowledge and an amazing way to learn from the best. I also tried to understand the history of things. Who created Kubernetes? How did this tool become so relevant and necessary everywhere? And why? (note: the official Kubernetes Documentary and its part 2 are a must, and a great option to get some initial answers). Then, I organically started to explore all technical topics and connect more dots. It was challenging, but feasible to learn with a good study plan and learning material. The initial version of the one you will see in this book.

Coming back to the KCNA exam, I remember being very excited when I first saw the scope and topics that the CNCF folks chose. But why was this exam necessary? Why would someone want to take it? Well, basically the KCNA is the best to demonstrate an initial knowledge of Kubernetes and cloud native topics, and a great opportunity for the community to create content for those who are willing to learn and evolve. Finally, a fundamentals, associate-level certification that would allow more people to join the movement, regardless of their previous background.

Up to then, the only available options for learning and certification required very advanced Kubernetes knowledge. Concretely, the Certified Kubernetes Administrator (CKA), Certified Kubernetes Security (CKS), Certified Kubernetes Application Development (CKAD). Quite a bit too long walk for people who are still trying to crawl. But the CNCF and Linux Foundation Certification teams nailed it, and they released a great exam that, even if it is not the easiest one, helps a lot to bring new folks and generate great career opportunities for fresh and experienced professionals.

Perfect timing, great opportunity. Some time after the beta exam release, we started to see more and more cool learning resources from cloud native professionals. Even myself, I prepared some 101-level KCNA exam prep sessions with O'Reilly Media, and the Open Source Summit LATAM. I consider myself a pretty good lecturer, but that was a fairly hard topic, and a difficult one to simplify. My main goal wasn't to deliver the perfect session, but to find a teaching approach that would get people closer to cloud native and Kubernetes, regardless of their professional background and technical ability. To be honest, I'm a bit biased because I have been teaching big data and AI for business-oriented professionals for a while. If I could explain those topics, I had to make it happen for this one!

Some months later, serendipity brought me to THE O'Reilly team working with Kubernetes-related authors. Concretely, the team that had managed previous book publications for other Kubernetes Certification topics, including the wonderful study guides that so many professionals had used before to prepare their CKA / CKAD / CKS certifications. And I was really willing to put some thought and time into designing something that may help KCNA candidates to 1) learn about Kubernetes and the cloud native ecosystem and landscape 2) pass the exam and leverage it to get amazing career opportunities. I wanted to prove that even the most challenging topics can be explained in a way that will help all sorts of professionals up and reskilling.

The rest is history. The wonderful O'Reilly team gave me the opportunity to sit and write the book. A relatively light guide for candidates to understand what to learn, in a pragmatic and simple way, but also as the starting point for their new cloud native journey. A pretty difficult task if we keep in mind that I do several things in my professional life (i.e. working at Microsoft, teaching, international speaking engagements, etc.). But all this was totally worth it. I like the topic a lot, and I love teaching and writing content. Being a book author is one of my main personal goals, and the KCNA is a highly relevant and scalable area of study and work. Let's see if this passion can be translated into a great and useful resource for young and experienced professionals out there. If this book helps just a few of them, I will feel very satisfied. Same as when I teach to a small group of students. Contributing to create a new generation of cloud-enabled professionals is very cool. Please enjoy it.

Adrian Gonzalez Sanchez

The Book Structure

This book has seven chapters that contains the end-to-end knowledge you are expected to have when you decide to take your KCNA exam. Concretely, very applied and descriptive chapters that will cover relevant exam topics, in addition to contextual information that will help you understand the full picture of the cloud native and kubernetes topics:

Chapter 1, Introduction to the KCNA Certification and Study Guide
: This first chapter is your way to begin your journey into the world of Kubernetes and cloud-native technologies with an in-depth introduction to the Kubernetes and Cloud Native Associate (KCNA) certification. Our main goal is that you understand its global relevance, the opportunities it opens up for professionals, and how this Study Guide is uniquely designed to cover every nuance of the syllabus, supported by real-world examples and expert insights.

Chapter 2, The KCNA Exam as the Starting Point
: The second chapter goes deeper into what the KCNA exam entails. This chapter not only touches upon the format and structure but also elaborates on the weightage of each domain, types of questions to expect, and the significance of this certification in one's professional journey. It serves as your personal roadmap to navigate the complexities of the exam.

Chapter 3, CNCF and the New Era of Cloud Native
: The third chapter includes a detailed overview of the Cloud Native Computing Foundation (CNCF), its mission, vision, and contributions to the cloud-native universe. You will learn how CNCF's initiatives and projects are redefining computing, and understand the nuances of cloud-native's transformative impact on businesses, from startups to big companies. We will also explore some relevant technologies beyond Kubernetes.

Chapter 4, Essential Concepts for Cloud Native Practitioners
: This fourth chapter is a descriptive exploration of cloud computing and cloud native foundational concepts. We will dive deep into topics like as-a-Service models, containerization, microservices, declarative APIs, etc. The idea is to equip you with additional information for your KCNA exam, but also that you make sense of the implications of these concepts on software development, deployment, and operations, and understand their interplay in building robust, scalable, and resilient systems.

Chapter 5, The Key Role of Kubernetes
: Kubernetes (also known as K8s) is more than just a buzzword; it's an ecosystem of very relevant technologies. You will understand its origins, its meteoric rise in the tech world, and why it stands out as the premier container orchestration tool. The chapter dissects its architecture, the philosophy behind its design principles, and its pivotal role in the successful implementation of cloud-native strategies.

Chapter 6, Technical Kubernetes Topics
: The sixth chapter is where (technical) things get serious. We will deep dive into the technical details of Kubernetes, from understanding the intricacies of pods, deployments, and services to unraveling more complex topics like ingress controllers, network policies, persistent storage, and stateful applications. Through detailed explanations and visuals, you gain a profound understanding of how Kubernetes works.

Chapter 7, Final Recommendations
: Before taking your KCNA exam, this chapter will equip you with the final set of tools and some pieces of advice. Benefit from a curated list of resources, and real-life tips from professionals who have successfully treated this path. Understand the common pitfalls, how to manage your time during the exam, and strategies to tackle challenging questions.

Summarizing, this book will cover the step-by-step learning blocks of your KCNA journey. Let's now see the typical audience for both the KCNA exam and the Study Guide.

Intended Book Audience

If you are reading this book, you likely have some preliminary knowledge or interest on cloud computing and cloud native topics. For example, before starting the preparation of the KCNA exam, you could benefit from some general knowledge of cloud computing offerings from the main hyperscalers, Linux fundamentals, command line experience, telematics and networking, etc. The perfect scenario would even include between six and twelve months of hands-on Kubernetes experience.

But we know that this is not always the case. It is understandable that if you are starting your cloud native journey with the KCNA certification, you can be missing some or all these knowledge blocks. Regardless of your profile, here are some hints for preliminary knowledge you should have before starting the certification journey:

Basic understanding of cloud computing
: This study guide will cover the fundamentals, but it would be good for you to know the public and private cloud fundamentals. For that, you can rely on existing O'Reilly resources such as books and video lessons. You could even pre-explore some cloud entry-level certification from the main hypescalers such as AWS Certified Cloud Practitioner, Microsoft Azure AZ-900, or Google Cloud Digital Leader. All of them include agnostic terms that will help you understand the cloud computing type of resources and capabilities, which you can easily apply by leveraging the amazing O'Reilly Cloud Labs resources.

Developer mindset
: This does not mean that you need to be a developer yourself, but you will probably need to understand the fundamentals of software development, the notion of microservices (which we will cover during the next chapters), and some basic software lifecycle terms. Same as before, you can definitively leverage other O'Reilly learning resources to explore the fundamentals of software development.

Industry knowledge
: Another piece we will cover here, but it will be good for you to have a clear overview of the different technology companies related to the cloud computing and cloud native industry. Names such as Google, Red Hat, Microsoft, AWS, etc. should already sound familiar to you, but if you know their solution offering, this process should be a bit easier for you. We will see managed Kubernetes services from different big companies in Chapter 6, so you don't need to worry too much about this.

Containers and Kubernetes
: This would be the best case scenario. If you have already used Kubernetes, Docker, and other related technologies, this will be a plus for you during the exam preparation. That said, we will cover the fundamentals required to pass the exam, some additional resources for you to continue learning, and some

technical aspects such as commands and architecture elements that may help you with some specific exam questions. Also, you can leverage interactive learning elements from O'Reilly, such as Kubernetes and Docker sandboxes to practice any kind of deployment and configuration, including the kubectl commands we will explore in Chapter 6.

In terms of roles, you may be just starting off in tech, or an experienced professional from another technical area, a technical manager or even a leader willing to better understand these topics. Or maybe you are just looking for new career opportunities. You can see in Figure 1-1 the typical exam candidate profiles for the KCNA certification.

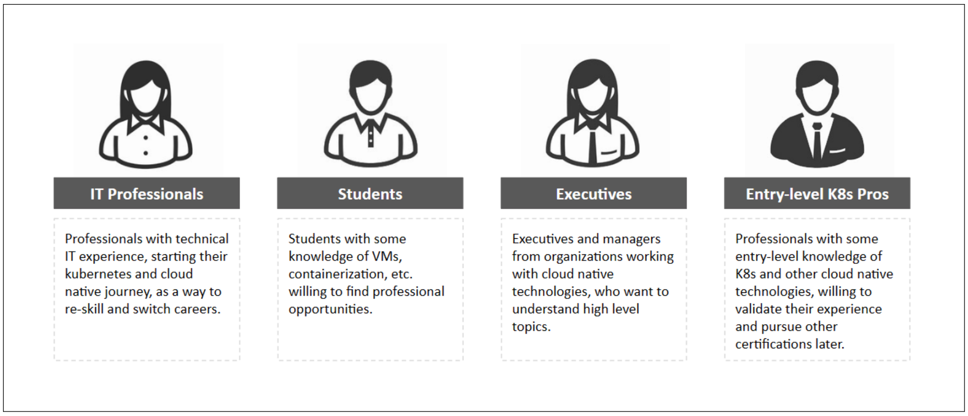

IT Professionals
Professionals with technical IT experience, starting their kubernetes and cloud native journey, as a way to re-skill and switch careers.
Students
Students with some knowledge of VMs, containerization, etc. willing to find professional opportunities.
Executives
Executives and managers from organizations working with cloud native technologies, who want to understand high level topics.
Entry-level K8s Pros
Professionals with some entry-level knowledge of K8s and other cloud native technologies, willing to validate their experience and pursue other certifications later.

Figure 1-1. Typical Roles for KCNA Exam Candidates

Regardless of your previous background, motivation is key to reach the actual goal of this study guide: learning the fundamentals of cloud native, including Kubernetes, and of course passing the KCNA exam, which is not a proof of knowledge or experience by itself, but for sure a great first step for anyone willing to join the cloud native industry, and a powerful portfolio asset for new career opportunities. By the end of the book we will recommend additional community and learning resources that will complement your upskilling journey.

Regardless of your current activities, here are some examples of cloud native roles that can inspire you to continue your cloud native learning path, based on the potential career options:

Software developers
: Responsible for designing, coding and testing the microservices that make up the cloud native applications. They use tools and languages suitable for the cloud environment and follow the best practices of DevOps and continuous delivery.

Operations engineers
: In charge of deploying, monitoring and managing the cloud native applications on the chosen platform. They use container technologies such as Docker or Kubernetes to orchestrate the microservices and ensure their availability and performance.

Solution architects
: They define the vision, scope and architecture of the cloud native applications. They make sure that the microservices are well defined, integrated and documented. They also assess the business needs and technical requirements to choose the best platform and cloud services.

Business analysts
: Folks who identify the opportunities, problems and needs of the customer or end user. They collaborate with developers, engineers and architects to define the functionalities, use cases and acceptance criteria of the cloud native applications.

Project managers
: To plan, coordinate and oversee the entire lifecycle of developing and implementing the cloud native applications. They manage the budget, schedule, scope and risks of the project. They also facilitate communication among all stakeholders involved in the project.

Cloud native engineer
: Responsible for designing, developing and deploying applications based on microservices, containers and cloud platforms.

Security engineer
: Dedicated to protecting cloud native applications from external and internal threats, applying security best practices from design to operation.

FinOps engineer
: Specialized in optimizing the use of cloud resources to reduce operational costs and improve energy efficiency.

QA engineer
: Focused on ensuring the quality and reliability of cloud native applications through automated testing, continuous monitoring and troubleshooting.

Business and sales professionals
: Non-technical folks working for cloud-enabled companies, and/or selling Software-as-a-service (SaaS) kinds of solutions.

: Any other role related to cloud native and cloud computing companies and activities.

We are just starting, but take some time to reflect on your personal goals, besides the KCNA exam and certification. What do you want to achieve, what is your envisioned role, what kind of skills you will require to make it happen, and which percentage of those is already covered in the Kubernetes and Cloud Native Associate certification.

Next let's analyze what this Study Guide will realistically bring to your learning and certification journey, keeping in mind that no book or resource will be a single source of learning for your upskilling activities.

What To Expect From This Study Guide

Before getting started, you should know that this KCNA Study Guide is not a regular book. Yes, you have text-based content, but we want it to be your living support for your cloud native upskilling journey, and obviously to help you prepare and pass the KCNA exam. Compared to other Kubernetes books and study guides, we are focusing not only on the exam content, but also on the contextual and additional aspects that may complete your learning experience. Also, we will refer to existing O'Reilly Learning resources, so you can make the most of them.

For that purpose, we are including exclusive expert insights and interviews from some of the key Kubernetes and KCNA actors, regular knowledge checks, links to rich and highly evolving community repositories, reference to relevant communities, and of course, a lot of applied knowledge. For this last point, we will be sharing examples on how companies are adopting cloud native technologies and generating value for their products and clients.

The next section includes contextual information about cloud native, and inspirational success stories from different adopter companies, and we will include other emblematic success stories in next chapters, that will illustrate the continuous technical and business value of Kubernetes and other cloud native projects.

Why Cloud Native and Why Now

This section will help you contextualize the information about the cloud native ecosystem, some intro level about the CNCF (that we will explore later in Chapter 3), and the intended audience for this KCNA Exam Guide.

The momentum of the cloud native movement is unstoppable. Not only because of the Kubernetes project (don't worry, if you don't know about it, this book will equip you with all relevant information), but also thanks to the exponential growth of the landscape and communities, and their level of contribution during the last years. Community-led ecosystems exist today for organizations, professionals, and even investors, and the Cloud Native Computing Foundation (CNCF) is playing the primary role as a catalyst. If you check the CNCF Landscape, there are more than 1200 tiles that represent the small and big companies involved in cloud native activities, about 820 company members, and a rich list of projects with an amazing level of community engagement.

But before starting… What's exactly "cloud native" and what does it mean for an adopter company? We will see the difference between general cloud computing and the cloud native term later, but even if the "cloud native" term has no one single definition, we can take CNCF's one as the main one:

"Cloud native technologies empower organizations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds. Containers, service meshes, microservices, immutable infrastructure, and declarative APIs exemplify this approach. These techniques enable loosely coupled systems that are resilient, manageable, and observable. Combined with robust automation, they allow engineers to make high-impact changes frequently and predictably with minimal toil." - CNCF Cloud Native Definition v1.0

This means having access to technical benefits typically related to cloud-enabled systems, regardless of the work environment. Let's see other definitions from big technology companies, which are obviously related but bring different perspectives, depending on the organization and their experience:

Google
: Cloud native is an approach to building and running scalable applications to take full advantage of cloud-based services and delivery models.

  https://cloud.google.com/learn/what-is-cloud-native

Red Hat
: Cloud native applications are a collection of small, independent, and loosely coupled services. They are designed to deliver well-recognized business value, like the ability to rapidly incorporate user feedback for continuous improvement.

  https://www.redhat.com/en/topics/cloud-native-apps

Microsoft
: Cloud native architecture and technologies are an approach to designing, constructing, and operating workloads that are built in the cloud and take full advantage of the cloud computing model.

  https://learn.microsoft.com/en-us/dotnet/architecture/cloud-native/definition

IBM
: A cloud native application consists of discrete, reusable components known as microservices that are designed to integrate into any cloud environment.

  https://www.ibm.com/topics/cloud-native

Oracle

The term cloud native refers to the concept of building and running applications to take advantage of the distributed computing offered by the cloud delivery model. Cloud native apps are designed and built to exploit the scale, elasticity, resiliency, and flexibility the cloud provides.

https://www.oracle.com/cloud/cloud-native/what-is-cloud-native

AWS

Cloud native applications are software programs that consist of multiple small, interdependent services called microservices. Traditionally, developers built monolithic applications with a single block structure containing all the required functionalities. By using the cloud-native approach, software developers break the functionalities into smaller microservices. This makes cloud-native applications more agile as these microservices work independently and take minimal computing resources to run.

https://aws.amazon.com/what-is/cloud-native

Summarizing, besides the fact that cloud native is a fundamental building block for all these companies' businesses, and regardless of the specific definitions, the important point of cloud native development is the ability to leverage capabilities from cloud computing, and to develop applications in a sustainable and scalable way by applying microservices, containerization, and other technologies and approaches.

But this level of adoption took some time. During the last two decades, we have seen the beginning of what I define as the new era of cloud native. New era because of the apparition of the CNCF, which has been growing since its 2015 creation, at both community and technology level (we will explore more details in Chapter 3). This generated not only a wide level adoption of cloud native technologies such as Kubernetes, but also the creation of a solid community of experts and adopters. Last but not least, the Foundation found its way, and the effervescence of events, training, resources, and new projects has been constant during the last years.

In parallel, the initial resistance of some public and private organizations to any cloud related topic, due in part to data residency and privacy concerns, is progressively disappearing. The cloud paradigm is becoming clearer for everyone, as well as its advantages for adopters. People are adopting and developing in ways that enable them to leverage the numerous cloud benefits. Bottom-up tool adoption from technical teams is guiding company-level overall maturity and making the decision process a bit easier, and the relevance of cloud economics and FinOps (i.e., financial operations oriented to optimize cloud investments) practices are helping companies to trust and control their cloud-related investments. Perfect combination, great timing.

But the best way to understand its success and advantages is by sharing tangible examples. Let's explore a few public success stories from all sorts of companies making the most of cloud-enabled technologies.

Inspirational Success Stories

Companies have progressively adopted cloud computing, and cloud native technologies during the last two decades. Initially, these powerful technologies were only available to a reduced group of technology-enabled companies such as Google, Amazon Web Services, etc., but other organizations have leveraged them and shown the way to adopt and transform their businesses. Here are a few examples from the public CNCF and Kubernetes.io case studies:

Boxed - *https://boxed.com (acquired by MSG Distributors in 2023)*

Boxed was an online wholesale retailer that offered direct delivery of bulk-sized packages. As their customer base grew, they began seeking more agile and scalable infrastructure solutions to handle their e-commerce operations. As the company began to experience increased traffic, it was clear that their existing infrastructure wouldn't suffice for the surges in demand, especially during sales or holiday seasons. Boxed faced the typical challenges of shifting to Kubernetes, including training their team, managing stateful services, and monitoring in a distributed environment. Boxed's journey with Kubernetes and cloud native emphasizes that even if a company was not a global giant, the principles of scalability, resilience, and agility provided by Kubernetes are a great proof of the democratizing power of open-source technologies, allowing companies of all sizes to harness top-tier tech solutions.

Airbnb - *https://airbnb.com*

Airbnb is a leading platform in the peer-to-peer service for people to list, discover, and book accommodations globally. As their platform rapidly grew, they found themselves dealing with infrastructure that couldn't easily scale with their needs.

As the platform grew and traffic increased, deployment with their existing monolithic architecture became riskier, slower, and more error-prone. To break away from the limitations of a monolithic structure, Airbnb looked towards a microservices architecture, where each function of their platform could operate as a distinct service. Airbnb began by containerizing individual services, progressively transitioning them onto Kubernetes. By starting with less critical services, they could ensure that any potential issues didn't critically impact their platform, allowing them to learn and adjust their approach. Airbnb's adoption of Kubernetes highlights how even platforms with millions of users and listings can leverage cloud-native technologies to achieve high levels of scalability, efficiency, and agility. Their journey serves as an inspiration to companies worldwide, emphasizing the importance of adaptable infrastructure in today's digital age.

Zalando - *https://www.zalando.com*

Zalando is one of Europe's leading online fashion platforms. With millions of customers, a vast inventory, and numerous daily transactions, they needed a robust and scalable technical infrastructure to support their operations. Zalando has actively embraced a mix of cloud-native technologies to bolster its technical infrastructure, including tools like Kubernetes, Helm, Argo, and Prometheus. For example, Zalando transitioned to Kubernetes for orchestrating their containerized applications, but they needed a way to manage Kubernetes applications effortlessly. Helm, a package manager for Kubernetes, was adopted to streamline deployments. Argo, a set of Kubernetes-native tools, was integrated to facilitate continuous delivery in their Kubernetes environment. Zalando could maintain and manage Kubernetes resources using declarative configurations, streamlining their deployment process. They also used Prometheus to monitor their services, set up alerts, and gain insights into their system's health in real-time. Zalando's journey illustrates how a blend of cloud-native technologies can supercharge a company's infrastructure.

The New York Times (NYT) - *https://www.nytimes.com*

The New York Times, one of the world's most renown newspapers, has been transitioning into the digital age, with a vast portion of its readership now accessing content online. With the rise in digital users and the delivery of news in real-time, there was a need to ensure their infrastructure could support the demand. They initially relied heavily on monolithic applications, which were becoming increasingly difficult to scale and maintain, especially with a growing global readership and the demand for faster content delivery. To better serve its readership, NYT wanted to break its monolithic application into microservices. Kubernetes was chosen as the orchestration platform due to its technical features and the community support and the growing ecosystem. The New York Times began its Kubernetes journey by containerizing its applications and then moving them to Kubernetes. The team decided to start with non-critical applications to understand Kubernetes better and to mitigate potential risks. Over time, as they gained confidence and expertise, more critical parts of their infrastructure were migrated. Thanks to this, the NYT handled reader traffic spikes more efficiently, and started to run their infrastructure in a more cost-effectively manner.

A.P. Moller - Maersk - *https://www.maersk.com*

Maersk is one of the world's largest shipping companies. As the global trade dynamics and customer expectations evolved, Maersk recognized the need to modernize its IT infrastructure to improve efficiency, reduce costs, and offer better digital solutions to its customers. Their legacy systems were disparate, slow, and often siloed, and they wanted to consolidate their IT operations and introduce more agility and scalability into their system. There was also an urgent need for real-time tracking, digital booking systems, and predictive analytics. Maersk decided to partner with Microsoft Azure to leverage its robust cloud infrastructure and services. Azure Kubernetes Service (AKS, Microsoft's managed Kubernetes service), became a core part of this transformation as the way to deploy, manage, and scale containerized applications using Kubernetes, without the complexity of managing Kubernetes clusters. Leveraging Azure's analytics and AI tools in tandem with their Kubernetes-deployed applications, Maersk could derive actionable insights from their data, leading to better decision-making. Maersk's story exemplifies how even century-old companies, traditionally not seen as tech innovators, can leverage modern cloud-native solutions like Azure and Kubernetes to revolutionize their operations and enhance customer experiences.

These examples are just a few cases illustrating the importance of cloud native technologies, and how companies are making the most of them. Keep in mind that the increasing technology demand has also a positive consequence: huge talent demand, which means a lot of professional opportunities for learners like you.

The next section (the last one of Chapter 1) starts with a brief self-assessment to allow you to analyze in an honest way those topics that will require more effort. This study guide was built for an audience with varying degrees of experience and knowledge; some readers will certainly know about some topics, and less about the rest. Once you determine your level of expertise, you can continue your exploration through this book, devoting more time to the topics you struggled with.

Self-questionnaire for New Cloud Native Practitioners

As a cloud native learner, you have a double mission here: to broaden your mind to the interesting and vast Kubernetes landscape and cloud native ecosystem, and (of course) to help you pass the KCNA exam. For that to happen, there are several areas of knowledge you will need to cover before even trying to take it.

This self-questionnaire will help you identify your current gaps and direct you to the areas you should focus on. Don't try to find the right answers; this is about evaluating your actual level of knowledge. If you are totally new to cloud native and/or Kubernetes and know very little, these questions will function as a preliminary structure for your study guide.

Concretely, we will have a series of ***45 multiple-choice questions*** for the following topics, relatively simple and with only one valid answer:

- 7 questions related to the ***CNCF Ecosystem***, mostly focused on high level details of the Foundation and its mission and mechanisms

- 8 questions for ***cloud native concepts***, as an initial validation of your understanding of cloud computing and cloud native
- 8 questions for ***Kubernetes and orchestration topics***, for to see start seeing the level of knowledge required for the KCNA certification
- 5 questions for **Kubernetes commands**, which are the kubectl instructions required to manage technical Kubernetes aspects. If you have never worked with Kubernetes at technical level, don't worry, we will explore these commands in Chapter 6.
- 8 questions related to ***Linux fundamentals***, which are technically NOT part of the official exam curriculum we will explain in Chapter 2, but that will help you not only understand some Kubernetes-related areas of knowledge, but also perform better at work.
- Last but not least, 9 questions for **other cloud native projects**, to explore the ecosystem of tools and projects (besides Kubernetes). We will describe them in Chapter 3, but it is good for you to know if you have some initial or high level knowledge of those tools.

Again, don't try to guess or find the good answers just to "pass" this test. Focus on choosing one based on your current level of knowledge, and take notes while you review the correct answers, especially for those areas you know less.

Test 1 - The CNCF Ecosystem

1. What's the primary role of the CNCF?
   a. To deploy new cloud native applications, as a service for anyone to use them
   b. ***To make cloud native universal and sustainable, by evangelizing and supporting projects***
   c. To fund Kubernetes-related projects that could benefit the rest of the community
   d. To convert existing closed projects into open source so everyone can adopt them and reuse their code
2. What does the second "C" of CNCF stand for?
   a. Containers
   b. Cloud
   c. ***Computing***
   d. Containerization
3. What does the term "cloud-native" primarily refer to?
   a. Applications hosted exclusively on public clouds

   b. Traditional monolithic applications migrated to cloud platforms
   c. Designing and building applications to run on cloud-based infrastructure
   d. Applications that use native APIs specific to a cloud provider
4. How does CNCF typically support its projects?
   a. By providing a monetary incentive to project maintainers
   b. By offering governance, marketing, and technical resources
   c. By mandating the direction and feature set of projects
   d. By taking ownership and commercializing open-source projects
5. What is the CNCF's stance regarding vendor neutrality?
   a. CNCF promotes only one vendor for each cloud-native technology
   b. CNCF endorses whichever vendor provides the most funding
   c. CNCF maintains a strict vendor-neutral position to ensure a level playing field
   d. CNCF is primarily tied to a single cloud provider
6. How are projects typically categorized within CNCF's ecosystem?
   a. By their popularity and number of users
   b. By their age and the duration they've been in the market
   c. By stages, such as Sandbox, Incubating, and Graduated
   d. By the programming languages they are written in
7. Why does CNCF emphasize end-user involvement and feedback in its community?
   a. To prioritize the commercialization of projects
   b. To understand real-world challenges and drive relevant innovation in its projects
   c. Solely for marketing and promotional activities
   d. To increase the sale of CNCF-branded merchandise

Ok, you may have missed some questions, but that's easy to solve as you will soon know the role and details of the CNCF (Cloud Native Computing Foundation). Explore Chapter 3 to learn more about this and other related topics.

Test 2 - General Cloud Native Concepts

1. What is the difference between monolithic and microservices architectures?
   a. I have no idea what you are talking about

   b. They are both different software design and architecture approaches, based on one vs. several development "blocks"
   c. The connection between data services from different pieces of the software solution
   d. They are similar, only front-end / UI differences
2. What are the main advantages of containers when compared to traditional virtual machines?
   a. Containers run the whole Operating System (OS)
   b. Containers are lighter and portable, and more efficient in terms of resources required
   c. Virtual machines are more agile and portable
   d. Virtual machines are cheaper and more scalable
3. Find the term that is not directly related to cloud native
   a. Infrastructure-as-Code (IaC)
   b. DevOps
   c. Distributed Ledger Technology (DLT)
   d. Elasticity (ability to scale units of workload up or down)
4. Which of these is an actual cloud as-a-Service model?
   a. Platform as a Service (PaaS)
   b. Back-end as a Service (BaaS)
   c. Deployment as a Service (DaaS)
   d. Front-end as a Service (FaaS)
5. Which of the following is a key advantage of a microservices architecture?
   a. It requires a single technology stack for all services
   b. It allows each service to be developed, deployed, and scaled independently
   c. It ensures that a failure in one service will cause the entire application to fail
   d. It simplifies the application as a single, indivisible unit
6. In a cloud-native environment, what is the main purpose of "containerization"?
   a. Increase the size of applications for better performance
   b. Package applications with their dependencies and configurations for consistent deployment
   c. Replace virtual machines entirely
   d. Store large volumes of data more efficiently

7. What is a primary advantage of continuous integration and continuous deployment (CI/CD) in cloud-native development?
   a. It requires manual testing after every change to ensure quality
   b. It allows for faster and more frequent release cycles
   c. It restricts developers from using new tools and technologies
   d. It eliminates the need for version control systems
8. In the context of cloud-native architectures, what does "observability" primarily refer to?
   a. The ability to observe team meetings and discussions
   b. Monitoring tools that provide visibility only when systems fail
   c. ***The ability to understand the internal state of a system from its external outputs***
   d. Tools that only offer static metrics and logs

These questions are easy to answer if you are familiar with cloud native content, but a bit challenging if you are just starting. Pay attention to Chapter 4, which discusses these and others cloud computing and cloud native terms.

Test 3 - Kubernetes Topics

1. Choose the correct compute concept hierarchy, from small to large
   a. Container, pod, node, cluster
   b. Cluster, container, node, pod
   c. Cluster, pod, container, node
   d. Orchestration, cluster, pod, container, node
2. Find the non-existent type of node in Kubernetes
   a. Master
   b. Worker
   c. Control-plane
   d. Task
3. Which of these topics is NOT related to Kubernetes?
   a. Observability
   b. Networking
   c. Data flows
   d. Policies

4. Which object is responsible for scaling and managing a set of replica Pods?
    a. ReplicaSet
    b. Deployment
    c. StatefulSet
    d. Pod
5. Which of the following is a Kubernetes service that is used to externally expose your Pod?
    a. ClusterIP
    b. ***NodePort***
    c. PodPort
    d. ExposePod
6. In a Kubernetes cluster, what is the main role of the "etcd" component?
    a. Scheduling pods on nodes
    b. Load balancing the traffic to services
    c. Storing configuration data in a key-value format
    d. Container runtime for executing the pods
7. What type of Kubernetes controller is best suited for managing stateful applications?
    a. ReplicaSet
    b. DaemonSet
    c. ***StatefulSet***
    d. Deployment
8. What is the primary purpose of a "ConfigMap"?
    a. To store secret data and passwords
    b. To define the desired state of a pod
    c. To store configuration data and parameters for pods to use
    d. To allocate CPU and memory resources for a pod

These Kubernetes topics are one of the core reasons and building blocks of the KCNA exam. Don't worry, if you don't have the answers for some of them, you will certainly get the required knowledge in Chapters 5 and 6.

Test 4 - Kubernetes Commands

1. Which kubectl command is used to view the detailed state of a specific resource?

   a. kubectl look
   b. kubectl show
   c. kubectl describe
   d. kubectl watch

2. If you want to view the logs of a particular pod, which kubectl command would you use?
   a. kubectl logs <POD_NAME>
   b. kubectl get logs <POD_NAME>
   c. kubectl describe logs <POD_NAME>
   d. kubectl show <POD_NAME>
3. Which of the following kubectl commands would you use to deploy a container using a YAML configuration file named deployment.yaml?
   a. kubectl create deployment.yaml
   b. kubectl apply -f deployment.yaml
   c. kubectl push deployment.yaml
   d. kubectl start -f deployment.yaml
4. If you want to get a list of all nodes in a Kubernetes cluster, which kubectl command would be appropriate?
   a. kubectl get nodes
   b. kubectl describe nodes
   c. kubectl list nodes
   d. kubectl show nodes
5. Which kubectl command allows you to enter the shell of a specific container?
   a. kubectl into <POD_NAME> -c <CONTAINER_NAME>
   b. kubectl exec -it <POD_NAME> -- /bin/sh
   c. kubectl shell <POD_NAME> -c <CONTAINER_NAME>
   d. kubectl run <POD_NAME> -c <CONTAINER_NAME>

The Kubernetes kubectl commands are one of the main areas of knowledge, not only for the exam but for your professional activities. We will include relevant information and additional resources for you to make sense of these technical instructions in Chapter 6.

Test 5 - Linux Fundamentals

1. Which command is used in Linux to view the contents of a directory?
    a. view
    b. watch
    c. dir
    d. ls
2. In Linux, what is the primary purpose of the chmod command?
    a. Change the ownership of a file
    b. Change the file's modification time
    c. ***Change the permissions of a file***
    d. Change the location of a file
3. Which of the following directories typically contains system configuration files?
    a. /bin
    b. /etc
    c. /usr
    d. /tmp
4. Which command in Linux is used to display your current working directory?
    a. cwd
    b. dir
    c. pwd
    d. locate
5. What is the primary role of the Linux kernel?
    a. Providing a graphical user interface for users
    b. Running shell commands and scripts
    c. Managing the system's hardware and resources
    d. Offering network services like DNS and SSH
6. Which command in Linux is used to kill a running process?
    a. terminate
    b. stop
    c. exit
    d. kill
7. Which file contains the system-wide environment variables?

   a. /etc/passwd
   b. /etc/shadow
   c. /etc/profile
   d. /etc/network
8. Which command do we use to view the end of a file as it grows in real-time?
   a. cat
   b. more
   c. tail -f
   d. head

Depending on your existing background and experience, you may have passed or failed most of these questions. In Chapter 3, we will point to a variety of Linux-related topics, as well as some existing O'Reilly and external resources that will help to catch up and gain some base knowledge, before taking the KCNA exam.

Test 6 - Other Related Projects

1. Identify the project that is NOT a CNCF one
   a. Prometheus
   b. ***Zeus***
   c. Litmus
   d. Helm
2. Which CNCF project focuses on cloud-native monitoring and alerting?
   a. Jaeger
   b. Helm
   c. ***Prometheus***
   d. Envoy
3. If you're looking for a CNCF project that serves as a package manager for Kubernetes, which would you choose?
   a. Helm
   b. TUF (The Update Framework)
   c. gRPC
   d. NATS
4. Which of the following projects is a service proxy designed to make network interactions for microservices more resilient and observable?

   a. Rook
   b. ***Linkerd***
   c. Vitess
   d. etcd
5. If you need a distributed logging project, which one would you likely opt for?
   a. OpenTracing
   b. CoreDNS
   c. CNI (Container Network Interface)
   d. ***Fluentd***
6. Find the CNCF project from all these open source initiatives
   a. Apache Atlas
   b. CRI-O
   c. Feathr
   d. Eclipse Data Connector
7. Which CNCF project provides a high-performance, lightweight service mesh that provides runtime debugging, observability, and security?
   a. Istio
   b. Jaeger
   c. Argo
   d. Linkerd
8. If you're interested in a CNCF project that offers distributed tracing to help troubleshoot latency problems in microservice architectures, which one would you refer to?
   a. CRI-O
   b. Vitess
   c. ***Jaeger***
   d. TIKV
9. Which CNCF project aims to provide a consistent and platform-agnostic way for plugins to handle network configuration of containers?
   a. Helm
   b. Linkerd
   c. CNI (Container Network Interface)

d. etcd

This part is relatively complex for new learners, and even if the goal is not to memorize all CNCF project names, you will need to be aware of the most important ones (especially graduated projects with relevant traction from developers and companies). The KCNA certainly includes questions related to them, and this Study Guide will help you develop this knowledge. As we can see in Figure 1-2, you will need to pay special attention to some parts of the books and other resources we will mention in this KCNA Study Guide, depending on your weakest areas of knowledge from the self-questionnaire, and your preliminary level of experience and knowledge:

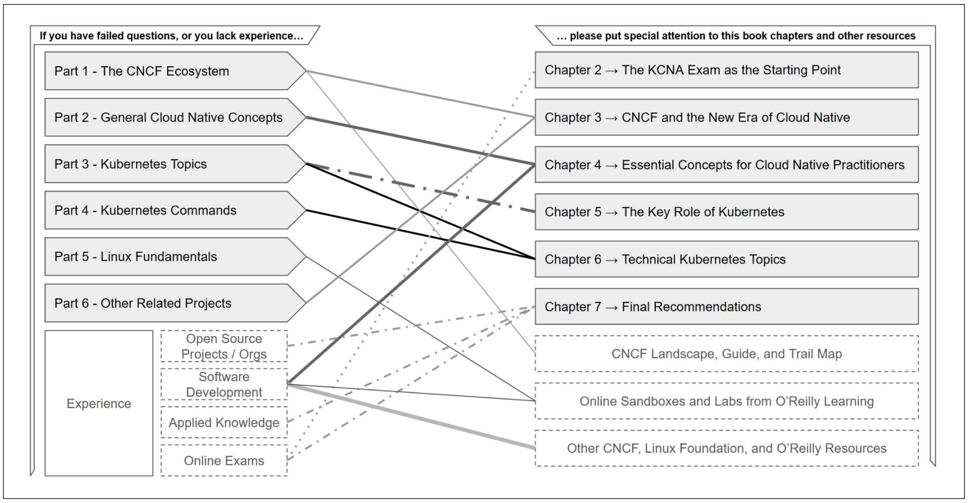

If you have failed questions, or you lack experience...
... please put special attention to this book chapters and other resources
Part 1 - The CNCF Ecosystem
Part 2 - General Cloud Native Concepts
Part 3 - Kubernetes Topics
Part 4 - Kubernetes Commands
Part 5 - Linux Fundamentals
Part 6 - Other Related Projects
Experience
Open Source Projects / Orgs
Software Development
Applied Knowledge
Online Exams
Chapter 2 → The KCNA Exam as the Starting Point
Chapter 3 → CNCF and the New Era of Cloud Native
Chapter 4 → Essential Concepts for Cloud Native Practitioners
Chapter 5 → The Key Role of Kubernetes
Chapter 6 → Technical Kubernetes Topics
Chapter 7 → Final Recommendations
CNCF Landscape, Guide, and Trail Map
Online Sandboxes and Labs from O'Reilly Learning
Other CNCF, Linux Foundation, and O'Reilly Resources

Figure 1-2. Action Plan from the Self-Questionnaire Results

Summarizing, these are relatively basic questions that cover only part of the KCNA exam scope. However, if you have found them difficult, and hesitated while answering some of them, you will need to carefully explore Chapters 2 to 7. Also, the KCNA exam won't go into highly practical Kubernetes details (compared to other Kubernetes exams, the KCNA does not include hands-on labs for live exercises), but you need to know the "ABC" of it. The key is to gain at the very least, basic knowledge for all these relevant areas.

PLACEHOLDER Expert Insights 1 - Walid Shaari

Note that the interview below was summarized. To access the video and hear the interview verbatim, please go to...

PLACEHOLDER EXPERT INSIGHTS

Interview with Walid Shaari

Adrian: Hi. Welcome to this episode of this series of expert insights for the KCNA exam. Today we have the pleasure to welcome Walid Shari who is an expert of the Cloud Native community.

Walid: Thank you for having me here.

Adrian: Many of our learners here are getting started on their Cloud Native journey so let's start with an introduction. Who is Walid and what's your relation with the Cloud Native ecosystem?

Walid: I'm currently working for Amazon Web Services for the public sector as a journalist and as a part of the containers community team, advocating for container services. Before this, I was leading the Ansible and the Docker community in Saudi Arabia where the adoption of containers and cloud native technology is still in its early stages. So when Docker took the world by surprise, I started the meetup and it was quite an eye opener. There was a lot of interest in containers, especially from developers. And it opened a lot of doors for me. In fact, my current career is a result of this community. So for the question of who's Walid, I see myself as a bridge between different cloud communities. I'm passionate about open source. And one thing about CNCF and about a cloud native community, you can see that it's the best community in terms of inclusiveness and knowledge sharing. This sort of culture is exhibited during KubeCon sessions and other events.

Adrian: Because everyone demystifies the fact that we cannot learn everything, the ecosystem is huge. It's not only about Docker and Kubernetes now, it's about all the projects around the community, all the certifications, all the training. So we are part of an ecosystem.

Walid: Exactly. When the CNCF, the Cloud Native Foundation, was established with the first project, Kubernetes, donated by Google at that time and helped by everybody else. They saw the gap between the enterprises, the organizations and the people. One of them being the skill gaps which they solved by providing curriculum, trainings and measures for the companies to check. Are these people qualified enough to be on my team or how can I upskill my team to be good enough? How can I have some kind of measures to qualify people within my team or even incentivize them to learn? Because of this, I'm very grateful to the CNCF in terms of finding these gaps and addressing them and bridging between us, the individuals and the enterprises that need us.

Adrian: You have been leading activities of CNCF on different journeys and events, even some local events. I think it was exactly related to this exam as well, right?

Walid: Right, I was a beta tester for the KCNA exam. It was really tough at that time because basically you don't have any content or anything that you can rely on except whatever is publicly available and slack channels. I was also a beta tester for the Prometheus observability exam. It's good to do the beta testing because basically you don't have content to learn from. You end up learning a lot because you don't have resources and you try to work with the community and create resources or find your way out.

Adrian: Right now, there is a lot of content out there. We are not missing content and documentation. In the book, we mentioned the glossary from the CNCF that is growing and improving. The official documentation from Kubernetes is amazing now. But that's the challenge when a new certification is coming out, right? We may not know the topics or maybe we know the topics because of a curriculum but it's not clear what kind of questions we will get. I remember you were leading some events to help prepare for this exam at some point. And there were some good video sessions that were very helpful, even for me, to get some additional knowledge.

Walid: Yes. So I worked with Sayem Batak who is a CNCF ambassador and a very well-known advocate when it comes to cloud native. His company Cebu Cloud also has the Kubernetes Academy, which are helping, which is one good resource for many certifications and many of the cloud native ecosystem. I participated with Sayem and Sysdig in doing events with the local communities. The challenge here is that you have to keep it fresh and to keep up to date. Exams like the CKA (Certified Kubernetes Administrator), CKD (Certified Kubernetes Developer) and CKS (Certified Kubernetes Security Specialist) change quite often. They get updated every six months or so with the release of Kubernetes. For the KCNA, however, I'm not aware of changes because it's the, it's supposed to be the high level. It's supposed to be the 10,000 feet level, but it's not really because it was written by engineers. So they have to, you'll still find command lines, you'll still find technical details. And these are the ones that actually might confuse you. When I did the exam for the first time, there were questions that stumped me. The answer choices look quite similar. For example, scheduling. If you want to protect the workload from running in a certain node, what do you use? Do you use labels or do you use affinity? Do you use annotations? The answers are very close to each other. The other challenge with this kind of exam is that because this exam is a multiple choice exam, it's not a hands-on exam. If it was hands-on and they know the answer, then absolutely it cannot be vague. The questions could be vague, but if the deliverables are clear, it's not an issue. And usually the exam questions are a paragraph. So there are enough details there to understand what is being asked. In the case of the KCNA, however, the questions were very, very concise. Most of them are a single line. I don't remember if I have seen a question that was a paragraph or a couple of lines long at that time.

Adrian: I agree, I have seen a good variety of the different questions of the exam. And they are focusing on specific details within a system. And this kind of example you provide was perfect, because this is the kind of detail that you can expect. That is normally unexpected when we talk about an associate level kind of certification. That's the reason why we need to go deeper in our preparation, even if it's an intro level certification. Because that's the kind of question we will get. And the other part is, all the command lines. You mentioned the command lines. If I ask you about the trickiest kind of questions that people can expect in a KCNA, what do you think will be the trickiest one?

Walid: The trickiest one for newcomers is they need to remember the command lines. If you are on the terminal, it will be very easy to get help but if you are a newcomer and you don't have enough hands-on experience, you might find it tricky to find the right answer. Let's say, for example, they are asking you about the CPU utilization across nodes for the pods, or the pods utilization or something like this. You need to be familiar with the Linux command line. In Linux command line, you usually use ps (process show) and aox, or wef. This concept is not in the Kubernetes world. In the Kubernetes world, there's kubectl top. You may be familiar with kubectl top, or kubectl top, or kubectl ps in Kubernetes. But I can try to confuse you with some Linux commands that look genuine, but they are not. It could be a Linux command mixed with kubectl commands.

Adrian: And if we add the prefixes, affixes and stuff, it gets tricky.

Walid: Yes. The other thing confusing is when you go into details. So I would say 60% of the exam is actually Kubernetes focused. The other 40% is the ecosystem. It's the CNCF organization and the ecosystem of data observability and application delivery, GitOps and things like that. So in this 60%, some of it really goes into details, like the scheduling. I remember they do really go into details. I remember I got one question wrong. I don't remember it exactly now, but I got it wrong because I didn't think about it at that time, and I didn't really prepare. So basically, for newcomers, who this exam is actually targeted for, they need to know their theory, and they need to have some hands-on knowledge. Andrew Brown has a free course, 14 hours, less than 20 hour course, with some hands-on follow-up. Basically, he walks you through some exercises. A Cloud Guru, they have also a course with hands-on training. I'm sure you also provide enough hands-on exercises and labs that people can go through. It's a pity that we lost some of the online available hands-on materials on Katacoda. Some vendors like Red Hat have some available training but it's very focused on their own distribution, OpenShift. So they have some tutorials that people can follow in GitOps. The other 40% of the test is more what I call level 100. For example, what is service mesh? And what are the prominent solutions out there? What about solutions that have been promoted or the ones that are not in the sandbox? What is the sandbox? What does it mean for a project to be graduated? The life cycle of a project in CNCF.

Adrian: You mentioned the ecosystem including activities and projects. We know that besides Kubernetes, there are other projects that are relevant for monitoring or for application delivery, et cetera. What are the projects that a new learner should be aware of, right? So taht they can get a very high level knowledge of these projecst, and can feel confident to answer these questions if they come up in the exam.

Walid: So the objective of the KCNA is to target newcomers and non-tech people that are in the Kubernetes world. Iif you think about Kubernetes as a platform to host other platforms, especially applications, the first thing one needs to do is to make sure that he can deploy an application. This is the first thing I ask myself. So deploying the applications from manifest, from YAML manifest, and then how to do it in the automated way. For example, packaging using Helm charts, templating using customize, automating the delivery using the GitOps. So basically, there is the Flux project, there is the Argo CD project. There are so many demos, there are so many solutions out there from cloud providers and from people that are supporting Argo CD that allow you to practice. There are also certifications as well. Say I have a cluster, Kubernetes, and this cluster could be managed in the cloud. What is the next thing I need to be aware of? Monitoring. Observability. In the past, we would consider logs, but now it's more than logs. It's the traces, because we are dealing with microservices. So which services talk to each other? Where is my bottlenecks? How's the latency? How's the performance? How can I troubleshoot real time? So observability is a must. Tools like Prometheus, Grafana, and Loki and their integration with each other to provide me with situational context. Maybe I have an error, and I can see the metrics. How can they relate to each other? For any newcomer, they shouldn't really focus on the technology. They should focus problem solving. So why did Kubernetes come into place? Why containers took the technology world and the business world by storm? The portability, the resiliency, the business value. And this is one thing, when we're discussing some chapters in the exam earlier on, I remember that we were discussing it in the comments. The value. Now, the FinOps is part of the exam, but it's not very obvious. The total cost of ownership, and the cloud adoption value. I'm not sure, maybe because I did the exam earlier, so I'm not sure, but maybe now, I believe, like after COVID, the last two years, I believe that this is an area where I see every customer is focused on the FinOps, on the cost, on how to optimize cost, and how to get better at chargeback and monitoring cost. Optimizing not just for cost, but also for sustainability. Because the more you optimize for cost, you optimize usually for sustainability at the same time. So it's a win-win situation. So this is one area, I'm not aware if the exam actually has been addressing it lately or not. It used to be a very weak area, but it's one of the areas that needs to be there. Back then, there was only one project, KubeCost. I think KubeCost was the primary open source project for monitoring FinOps.

Adrian: This is a good example given that this topic related to FinOps is a piece of knowledge that are not officially or in a very specific way a part of the official curriculum of the KCNA. However, it helps learners, because preliminary knowledge like this will help people perform better on the exam and even better understand the concepts that are related. For example, I'm a telematics engineer, so working with Kubernetes is more natural to me. Not saying that it's easy, it's very difficult, but it's more natural for me to understand because I have that background. With the variety of learners profiles that we get on this exam, we can assume that people will have stronger background in some specific areas and then weaker in others, so we are encouraging them to analyze where they need to improve based on their existing knowledge. What's the best way to practice today's exam topics? You have mentioned a couple of them.

Walid: Nowadays, I mean, if you want to practice, if someone wants to practice on his own laptop and learn about Kubernetes, there is Kind, K-I-N-D, Kubernetes and Docker. First, they need to cover the basics, especially with Docker, where there are good free courses out there. There is Kind, Minikube, and other services where you can have a managed cluster. There are lots of YouTube contents. There are books, this book, for example. There are meetups. I feel the meetups are good because they are interactive. And there are the conferences like KubeCon and Kubernetes Community Days. So CNCF saw that they cannot scale when it came to conference and that these conferences is getting really huge and really starting to be expensive. So one way to scale them up, doing Kubernetes Community Day around the world, and we see lots of them nowadays. Usually, there are workshops around them also. Around the big events, usually there are other events for specific technologies like GitOps, service mesh, networking, special distributions, and the operator framework from Rancher. They were doing this like every KubeCon, how to create an operator. The operator in Kubernetes is how to combine human knowledge, operator knowledge, and technology, especially for stateful applications, into a packaging. And this packaging becomes easy to install a software stack like Postgres cluster and how to update it, how to monitor it. So it takes you from day zero to day two and beyond, hopefully. There are many resources. The CNCF Slack is one resource. The CNCF events in terms of meetups, in terms of chapters, in terms of conferences. Books from O'reilly and from all other publishers. And the local events. There are some GitHub resources also where they aggregate and curate some content. And the best thing is actually to participate in projects. If someone wants to learn more about something and he has a business problem or university challenge or project, it's better to participate in this project and explore and ask the community. Just ping somebody from the community to ask, especially in Slack.

Adrian: In the book, we explain the notion of a contributor and maintainer, two different types of roles. Have you been a contributor to any project or a maintainer?

Walid: Unfortunately, not for a software project. I have been maintaining resources for the CKA and CKS resources. For projects, when I see a bug, when I see an issue, I usually raise an issue at the very least, if I cannot create a pull request. I'm trying to get involved with the documentation. The easiest thing, especially in the CNCF-related project, there are tags with the first-time contributors. Basically, these tags help you find the easiest issues and challenges where you can start. There are mentor programs. You can be part of the release team. They always publish that on Twitter or Slack. Every new release of Kubernetes, regardless of your experience, regardless of your past contributions, they always welcome new, fresh people, because of the diversity and inclusion, to shadow and to learn from them. That's how the project has been going. I mean, I haven't seen this in other projects.

Adrian: That's gold, because it offers beginners like many of our exam takers here with an opportunity to join projects. It doesn't mean that we need to develop a core functionality of Kubernetes but we can help with some documentation or checking bugs. It can be joining just the meetings to learn. And that's part of cloud-native upskilling. There are different ways to do it, but I totally agree.

Walid: So giving feedback and helping with documentation, helping with localization, these are the easiest way in.

Adrian: Perfect.I think we have covered more or less all the important topics for this exam preparation for exam takers, but are there any other recommendations or topics that you would like to highlight that you think will help people preparing their associate-level exam?

Walid: For the associate-level exam, I would focus on the KubeCon business tracks. They haven't been that many, but they have been recent. You can learn about the ecosystem well if you look at the keynotes from KubeCon and the business cases, especially for observability, for premieres, especially for the application delivery, GitOps and stuff like this. Other than that, Twitter is good. Follow James Barron, RawCode, David Flemingen. He has RawCode Academy. And every couple of weeks, he has a new topic, a new subject. He's discovering a new technology, discovering a new software stack or playing with it. We used to have TGIK. Unfortunately, we don't have it anymore. Thank God It's Kubernetes (TGIK) run by Heptio earlier and then later VMware. You can follow people in the community through social networks and LinkedIn. For me, I get my feeds from X (Twitter) and LinkedIn. I follow specific people in LinkedIn and I get the updates and news from them of what the new projects are. For me also, I found out that OpenShift TV or OpenShift Red Hat streaming, especially GitOps Guide to the Galaxy is very nice. They are more focused on OpenShift, in general. Containers from the Couch is another one. Brendan Burns from Microsoft, he has small short videos. VMware has the same short videos. I like these small nuggets, actually. Maximum 15 minutes.

Adrian: Yes, like a mini Netflix show. You mentioned Brendan Burns from Microsoft, Joe Baguley's videos from VMware. There are lots of material. We even recommend the documentary. You mentioned it once when we were reviewing the book, the documentary that is very illustrative.

Adrian: This is wonderful. If we had to summarize, I think that one of the words for people to prepare their excellent journey is awareness. Just being aware of the existence of things, the kind of projects that are there, the ways to contribute or to participate, the kind of questions that you can get. That's what we're trying to cover in this book as a study guide. We always mention, no book or study guide will be a single resource to prepare for Kubernetes or Cloud Native topics. You need to go and check all the resources and complement it and create your own mix. But this is very aligned with what we're trying to do there.

Walid: Yes. Different people have different tastes. Some people like books, some people like videos, some people like podcasts. Whatever floats your boat. But start with the Linux Foundation, start with the GitHub repo. CNCF have a GitHub rebel for their exam curriculum. They keep updating it. They actually mention some resources there.

Adrian: Yes. Very good selection. By the end of the book, we are also adding, let's say, the champions in the community, including yourself, your personal repositories, which you are recommending already, like a good structure for this exam. I remember other people doing the same. These are good resources that are helping learners around the world. Thank you.

Walid: Yes. Thank you.

Adrian: Well, I'm very happy we got some time to discuss these topics. I cannot wait to release everything, the material, and what you mentioned, not in the format of a video or a book. It's small pieces for people to continue improving their learning and to pass the exams. It's not necessarily easy, even if it's beginner level, but I think that this is the way.

Walid: This is the way. True. Perfect.

Adrian: Well, thank you very much again. Have a lovely day.

Walid: Thank you, Adrian. Thank you very much.

We hope you have enjoyed this expert discussion with Walid Shaari. He is certainly one of the top KCNA experts out there, and part of the original team who created the exam questions. Feel free to explore his valuable KCNA resources, such as his KCNA repository with relevant information to prepare the exam, and the amazing series of three video sessions (including part 1 and part 2, and a deep dive of container orchestration) with him and his colleague Brad McCoy.

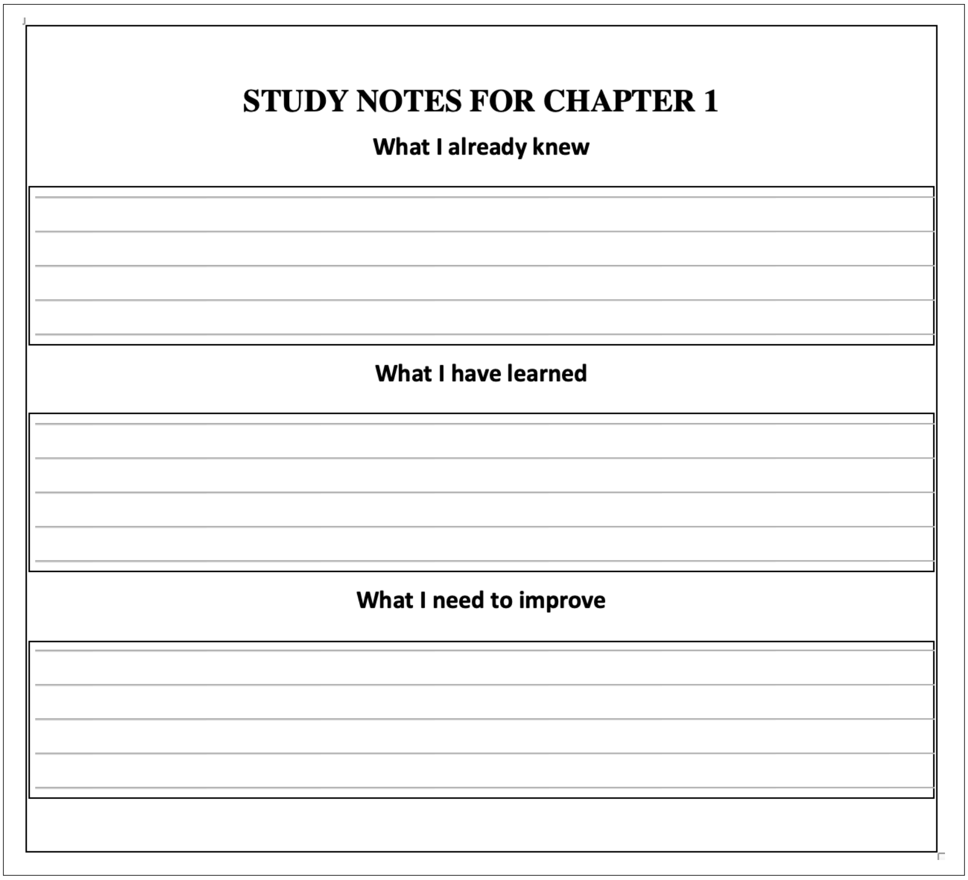
This is the first of a series of expert interviews that will bring you applied and exclusive insights, related to the KCNA exam and to the cloud native ecosystem and its endless possibilities for upskilling, collaboration, and professional opportunities.

PLACEHOLDER CONCLUSION - Chapter 1

We have now concluded the first of these seven chapters. At this point, you should have a good initial idea of what the KCNA exam is about, and the learning approach of this Study Guide. You have also gotten some exclusive insights on the KCNA exam creation process, and some recommendations to start working on it. If you feel like you are still missing some details, at both logistics and knowledge levels, don't worry. We will help you with everything.

Now, before moving to Chapter 2, please complete your study notes (if you are reading a hardcopy version of the book, feel free to use this page as a regular notebook) with information about what you already knew and you are now aware of, the new things you may have learned up to know, and those that you think you will need to improve.

This may look a bit old fashioned, but remember that writing down things will complement your reading and listening-based reading process. Even more, you will have the opportunity to practice some of the concepts with hands-on platforms.



STUDY NOTES FOR CHAPTER 1

What I already knew

What I have learned

What I need to improve

Figure 1-3. Study Notes - Chapter 1

CHAPTER 2

The KCNA Exam as the Starting Point

Learning Kubernetes and getting certified is not an easy task. Just a few years ago, available certifications on the topic were quite advanced for newcomers, but the KCNA exam and certification are now feasible learning goals, with a good balance between entry level elements and useful pieces of knowledge for Kubernetes practitioners. Also, it is an initial proof of value and knowledge for candidates looking for professional opportunities.

If you are reading this book, that means you want to get the KCNA certification from The Linux Foundation, but you may know less about Foundation itself (don't worry, we will cover it during this second chapter). Also, you should know that passing this exam demonstrates a strong understanding of the Kubernetes and cloud-native fundamentals, but it will also show that you have a grasp of the principles of cloud-native development, security, etc. Thus, having this certification under your belt prepares you for further advanced and specialized credentials as well as signaling to the industry of your relevant and up to date knowledge on the topic.

But preparing and passing an online exam, especially if it is the first time you do it, is not easy. In this chapter we will explain to you the exam details and logistics, in order to set us up for success. We will be reviewing its structure, topics covered and what to expect on game day.

Let's start by learning about the exam creation and its origins.

The KCNA Certification by Linux Foundation and CNCF

Let's start with the origins of the KCNA exam. Who created the KCNA exam? The answer is The Linux Foundation, and the Cloud Native Computing Foundation (CNCF), in partnership with Certiverse, an online platform for exam creation that enabled a group of 15 renown subject matter experts to create test content asynchronously.

As a learner and exam taker, this information is important for you for a couple of reasons: You will leverage The Linux Foundation's online exam platform (the one we will explain here, for you to get ready and not miss any important step). Also, the content of the exam and other resources relies heavily on specific knowledge from experts who often act as CNCF ambassadors or contributors, as we have seen with Katie Gamanji' intro, and Walid Shaari in Chapter 1.

As you begin learning about cloud native technologies, you will notice that both The Linux Foundation and CNCF names come up frequently. In Figure 2-1, for illustrative purposes and your general knowledge, you can see their foundation logos:

Figure 2-1. The Linux Foundation and CNCF Logos

The Linux Foundation was born in 2007 from the merger of two existing organizations, the Free Standards Group (FSG) and Open Source Development Labs (OSDL). It is a non-profit organization that *"provides a neutral, trusted hub for developers and organizations to code, manage, and scale open technology projects and ecosystems"*. It supports companies and developers to identify and contribute to projects that address the challenges of industry and technology for the benefit of society.

The Linux Foundation (LF) is the umbrella organization for other projects and sub-communities, one of which is the Cloud Native Computing Foundation. Actually, the CNCF is one of the Linux Foundation's largest sub-foundations but it has a unique role to play in the ecosystem in that it serves as a community responsible for fostering the growth and promoting open source cloud native computing technologies. You can see in Figure 2-2 a sample of the list of LF projects. Feel free to bookmark this URL and to explore its details to find other topics (e.g. databases, development, financial operations for the cloud) that will complement your cloud native upskilling journey.

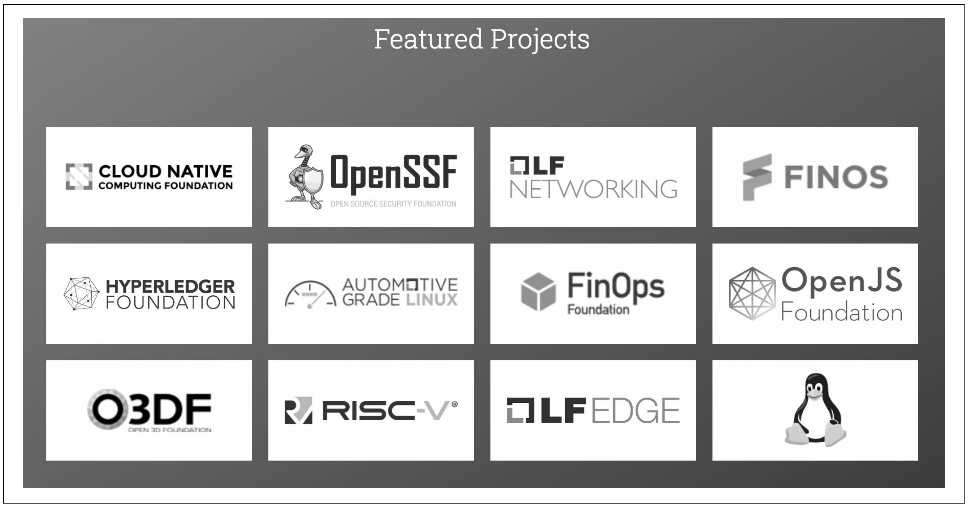


Figure 2-2. : The Linux Foundation Projects and Communities

Seeing that the CNCF is just one of the many organizations under The Linux Foundation umbrella, it is not surprising that it is a wealth of knowledge when it comes to online learning resources and documentation. The Linux Foundation works with all these open source communities in order to create training, resources, and industry standard certifications. It offers various learning paths composed of both self-paced and instructor-led courses and materials, as well as certifications organized per skills and technology groups.

The KCNA that is the subject of this book falls under The Linux Foundation's Cloud & Containers Learning Path. This and other learning paths include vast selection of system administration, Linux development, telecommunications networking, cybersecurity, DevOps, and other cloud topics. Based on the results of the self-questionnaire in Chapter 1, you may want to explore the different learning paths, based on the potential knowledge areas of improvement you have detected (i.e. those where you feel you need to improve a bit more). For example, if you have failed most of the Linux-related questions, you can go to the System Administration Learning

Path, and check beginner level courses such as the Introduction to Linux class, which is free and it's available in both English and Spanish.

Related to the core requirements of the KCNA exam, you may want to take a look at the Introduction to Cloud Infrastructure Technologies and the Introduction to Kubernetes free courses, as they can bring a complementary perspective to the existing catalog of O'Reilly Learning Kubernetes resources. Once again, depending on your level of experience and professional background, the success formula will combine this KCNA Study Guide as your main reference to prepare for the exam, but also different sources of knowledge.

Apart from the aforementioned role as industry knowledge hub, and same as the CNCF, the Linux Foundation international events (such as the Open Source Summits in North America, Europe, LATAM, and China) conducts open source research, and provides practical guidelines and support for open source collaboration. We recommend these and other pieces of knowledge, as they will complement your learning experience and help you connect the dots between technologies, communities, etc.

Now that you know the overall structure of The Linux Foundation, and its role as enabler for your professional up or reskilling adventure, let's come back to the KCNA exam and see the actual reasons for the creation of this certification, within the end-to-end ecosystem of cloud native exams.

Why the KCNA matters and Why to Get Certified

The Linux Foundation's Training and Certification team releases, in partnership with edX and Linux Foundation Research, an annual Open Source Jobs Report to outline the latest trends in open source careers and hiring demands, including those related to cloud native and Kubernetes topics. This is the contextual information you need to understand the ***actual value of the KCNA certification***, as it will give you a clear idea of the industry needs, and your market value as a cloud native professional.

If we take a look at the 2022 report and its previous editions, they showcase a clear trend and the obvious need to ***bridge the talent gap***, given the increasing demand for qualified open source and cloud native talent. In 2022, 93% of hiring managers reported difficulty finding sufficient talent with open source skills, while this same figure was 87% in 2020. Perhaps not surprising, but the report also found that hiring managers are more likely to hire someone with a certification. Concretely, summarizing the report insights that are relevant for you:

The Unwavering Reign of the Cloud –
: Skills in cloud and container technologies continue to be the most coveted, with 69% of employers on the lookout for such expertise. This demand is affirmed by 71% of open-source professionals.

Certifications Gain Prominence –

An overwhelming 90% of employers are ready to finance their employees' certification endeavors, and 81% of professionals aim to acquire additional certifications, underscoring their rising value.

Compensation Emerges as a Key Incentive -

While financial rewards, encompassing salaries and bonuses, stand as the primary retention strategy, an impressive two-thirds of open-source experts contend that an augmented salary would discourage them from job-hopping. As flexible schedules and remote working become commonplace, monetary benefits rise in significance, overshadowing lifestyle perks.

Increased Expenditure to Avoid Project Delays –

The prevailing approach to bridge skill deficiencies is through training, as indicated by 43% of hiring managers. However, hiring consultants to address these gaps is on the rise too.

Persistent Lack of Open Source Experts –

A striking 93% of employers face challenges in recruiting individuals proficient in open source. Moreover, 73% of these professionals believe they can effortlessly transition to a new position if desired.

Escalating Cyber Security Imperatives –

Ranking fourth in recruitment decision influences, cybersecurity competencies are highlighted by 40% of employers, only outpaced by cloud, Linux, and DevOps skills.

Moreover, The Linux Foundation's 2023 State of Tech Talent Report reiterates the importance of industry certifications, and showcases cloud native and containerization as one of the key areas of hiring priorities for most of the companies, along with AI and cybersecurity.

In parallel to this, the CNCF's 2022 Annual Report, indicates that the KCNA exam hit 4,000 registrations since its launch in November 2021, and that trend is similar to other advanced certifications such as CKA, CKAD, etc. Other CNCF publications such as the CNCF Annual Survey 2022, which contains a curated series of insights from the cloud native industry, highlights the unstoppable adoption of cloud native technologies (obviously, including Kubernetes), and reasons such as ***lack of talent or security concerns*** being the main blocker for a lot of companies out there.

Let's take a minute to reflect on this. What does it mean for you? Is the KCNA exam a good investment of time and effort? Based on the industry trends and the figures, it is clear that the KCNA will enable you to enter a promising industry, where certifications have a clear value for hiring managers, and where the mix of high demand and talent scarcity can give you the perfect scenario to shine and to continue learning and progressing.

Although there is no official statistics on hireability of candidates passing the KCNA, the benefit of this certification really lies in preparing you to get some initiative experience in the job market, as well as by completing further advanced certifications. Trust us for now, and let's now deep dive into the details of the KCNA exam and its official curriculum.

Exam Description and Curriculum

The Cloud Native Computing Foundation (CNCF) and the Linux Foundation define the KCNA exam as the way to demonstrate foundational knowledge and skills in Kubernetes and the wider cloud native ecosystem. Announced in 2021, the KCNA is the way to *"both prepare individuals for entry-level cloud roles and to sit for the KCNA exam"*. Yes, very similar to the goal of this KCNA Study Guide. Let's start by the key information of the exam:

Target audience

The KCNA serves as an introductory certification tailored for individuals aiming to progress to advanced stages, by showcasing their foundational proficiency in Kubernetes. It's perfectly suited for those diving into or eyeing roles that emphasize cloud-native methodologies.

Pre-professional profiles

Just because this is a beginner level exam does not mean that it is not suitable for those already familiar with cloud technology. For any level of experience, there is value in demonstrating solid understanding of cloud native topics which is what this certification aims to do.

Certification insights

Earning the KCNA title authenticates an individual's grasp of the overarching cloud-native realm, with an accent on Kubernetes. The examination for KCNA paves the way for individuals to immerse themselves in cloud-native tools and readies them for other CNCF accreditations, such as CKA, CKAD, and CKS.

Proficiency showcase

Securing the KCNA accentuates an individual's elementary acumen in Kubernetes and allied cloud-native tools. This includes executing deployments via rudimentary kubectl directives, discerning Kubernetes' structure (encompassing containers, pods, nodes, and clusters), recognizing the broader cloud-native project spectrum (spanning storage, networking, GitOps, service mesh), and assimilating the core tenets of cloud-native security.

Conceptual knowledge

Being a beginner level certification, the exam focuses on testing conceptual as opposed to technical knowledge. This is meant to guide the learners to spend more effort mastering the functions of the technology and its implementation rather than to prove their ability to execute it in a simulated environment.

Entire cloud native ecosystem

How big is the *entire* cloud native ecosystem? The answer is large and growing. However, the CNCF outlines the exact scope covered in this certification in their curriculum. More discussion in this chapter, and detail to be covered in Chapter 3.

About the format: Even if the general description of the exam above shows KCNA as an entry-level certification for practitioners who are starting out, the reality is that it can still be a bit challenging for beginners. It doesn't require practical knowledge such as typing commands or making configurations (other advanced certifications do have a working lab to test your knowledge), but the content being tested goes into a level of detail that can only be acquired by practicing, or by studying very targeted topics. In general, you will find multiple-choice questions (MCQ) with several possible answers, including specific applied cases (questions that include a specific company scenario, where you will need to choose a good option for their needs). Last but not least, it is not an open-book exam so you will be relying on your knowledge during the exam duration, with no access to this KCNA Study Guide or other documentation.

In summary, the KCNA exam covers all basic pieces related to Kubernetes and orchestration fundamentals, as well as cloud native orchestration, observability, and application delivery. Most of these topics may not be familiar yet, but you will get an introduction to them in the next chapters. Meanwhile, here is an overview of the official KCNA curriculum, always updated and available via CNCF's Github with a high-level introduction of each exam topic.

This section is only an overview of the topics, but we will include the details, descriptions, and explanations of each term within the main five exam areas in Chapters 3 to 6. Meanwhile, we are including links to the main documentation references:

- Kubernetes.io Documentation - https://kubernetes.io/docs
- CNCF Glossary - https://glossary.cncf.io

You can consider these two URLs as the main source of terms and descriptions, very useful for your cloud native upskilling, including the KCNA exam preparation (some questions focus on core terms that are described in these two pages).

Part 1 - Kubernetes Fundamentals

It counts for 46% of the total exam content, and we can refer to this block of the content as a Kubernetes 101 section. Maybe even 201, because some concepts such as technical commands or architectural elements are not trivial. Concretely:

Kubernetes resources
: One of the fundamental aspects of Kubernetes is its variety of K8s objects. Concretely, some terms for new kinds of resources that you will discover soon, such as pods, deployment, controllers, kubectl and its commands, ReplicaSets, etc. These are the resources necessary to build cloud native architectures with K8s.

Kubernetes architecture
: Given the available resources, this portion of the test refers to the architectural design and components of Kubernetes-enabled applications. How each component interacts with each other to carry out the function of the architecture.

Kubernetes API
: As a Kubernetes practitioner, you will be dealing hands-on with this component. The Kubernetes API is the interface that allows users to interact with Kubernetes components to adjust resources and perform settings and configurations.

Containers
: This portion deals directly with the notion of containerization, and its role for microservices implementation. The concept of containers is at the heart of Kubernetes. The Container Orchestration section below builds upon this concept.

Scheduling
: Scheduling in the context of Kubernetes implementation refers to the concept of automated assignment of resources, in order to distribute workloads in an optimal way, maximizing resource utilization and application performance.

Part 2 - Container Orchestration

This second block counts for 22% of the total exam content. Even if it is related to the first “Kubernetes Fundamentals” section, it focuses on the core notion of orchestration, and its management in production-level systems. It includes:

Container Orchestration Fundamentals –
: Given that Kubernetes is one of the many industry tools available for container orchestration, it’s natural that this portion makes up a big chunk of the exam. This section deals with basic notions related to automating deployment, scaling, and management of containerized applications.

Related topics –
Terms like runtime, cloud native security, networking, service mesh, and storage are peripheral, yet essential to the practitioner's ability to successfully orchestrate and maintain containers in Kubernetes.

Part 3 - Cloud Native Architecture

The architecture part punts for 16% of the total exam content. Besides being an important piece of the exam, it is also a great introduction to cloud native architecture and development topics, as well as the fundamentals of the CNCF and some industry standards.

Cloud Native Architecture Fundamentals –
This section dives into the general cloud native architecture and relevant concepts beyond Kubernetes. Touching upon but not limited to microservices, DevOps and Continuous Integration / Continuous Deployment pipelines (CI/CD). Topics related to general cloud native knowledge areas and autoscaling and serverless concepts.

Community, Governance, and Personas –
These are specific topics related to the Cloud Native Computing Foundation (CNCF) and their setup. Given that K8s is the first project under their umbrella, reinforcing open source culture and governance is an important factor for the exam.

Open Standards –
CNCF and the Linux Foundation supports implementation and adoption of international open source standards for cloud native, containerization, and orchestration technology development. It contains several acronyms that may be unknown for you right now, but that guide the standardization and alignment of the different projects and initiatives.

Part 4 - Cloud Native Observability

The first of the two sections that count for 8% of the total exam content and grade. It contains admin, maintenance, and support building blocks that will help you manage your cloud native systems.

Telemetry and Observability
These terms (that we will explore later in Chapter 6) bring a holistic view of cloud native systems based on quantitative information about their performance.

Prometheus

Similar to Kubernetes, Prometheus is one of the earliest CNCF projects, and one of the most relevant at cloud native level. It is a widely adopted tool for monitoring and alerts.

Cost Management

This section aims at having the practitioners demonstrate their understanding of cost implication of cloud utilization. It covers techniques and tools to control cloud native costs..

Part 5 - Cloud Native Application Delivery

The last of the five sections, which is oriented to application development topics, counts for 8% of the total exam content too. It goes from the fundamental capabilities of cloud native applications, to the methods and tools to move them to production level.

Application Delivery Fundamentals

Cloud native development is based on principles such as fluidity, resiliency, and scalability. These principles drive the implementation of cloud native systems.

GitOps

Git-enabled control of cloud infrastructure, from development to deployment, and app lifecycle. GitOps enables an end-to-end cloud infrastructure management framework for consistent development, making it easy to follow best practices and guidelines. This section includes existing CNCF projects such as ArgoCD and Flux.

CI/CD

It refers to the continuous Integration and Continuous Deployment for automated cloud native development, deployment, and testing, and it includes other CNCF projects like Flux and Argo.

All topics mentioned above will be covered in this study guide, and here is the high level mapping of the Study Guide chapters, with the KCNA curriculum:

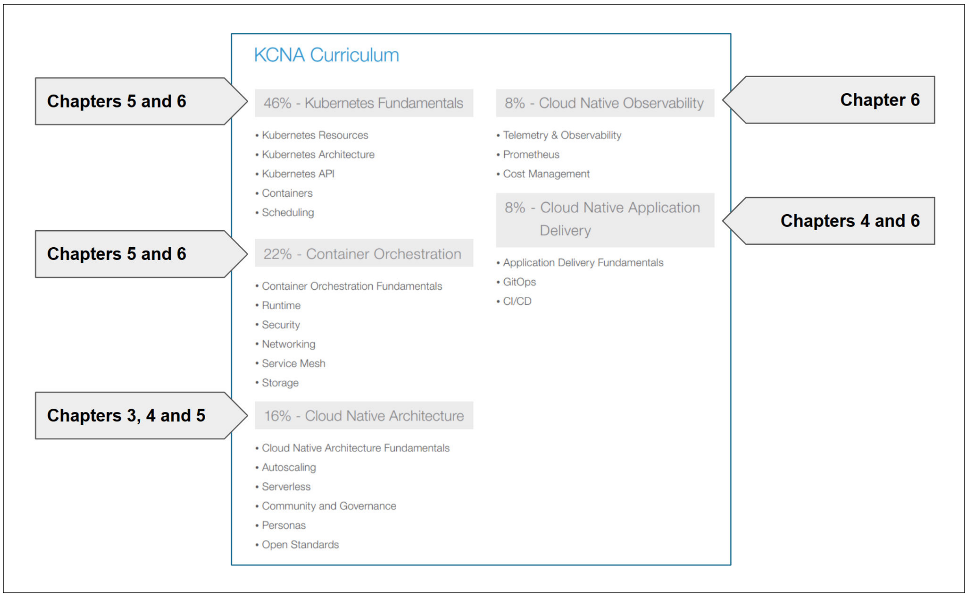

KCNA Curriculum
Chapters 5 and 6
46% - Kubernetes Fundamentals
• Kubernetes Resources
• Kubernetes Architecture
• Kubernetes API
• Containers
• Scheduling
Chapters 5 and 6
22% - Container Orchestration
• Container Orchestration Fundamentals
• Runtime
• Security
• Networking
• Service Mesh
• Storage
Chapters 3, 4 and 5
16% - Cloud Native Architecture
• Cloud Native Architecture Fundamentals
• Autoscaling
• Serverless
• Community and Governance
• Personas
• Open Standards
8% - Cloud Native Observability
Chapter 6
• Telemetry & Observability
• Prometheus
• Cost Management
8% - Cloud Native Application Delivery
Chapters 4 and 6
• Application Delivery Fundamentals
• GitOps
• CI/CD

Figure 2-3. Mapping of the KCNA Curriculum & The Study Guide Chapters

Now, even if the official KCNA curriculum is the main scope reference for this exam, you may find some additional pieces of knowledge that cannot be intuitively deduced from the curriculum itself. For example, you may get a mix of descriptive and command-related questions, which makes sense but maybe wasn't clear for newcomers when they read the official curriculum. Or the questions may rely on specific preliminary knowledge, and some previous experience. For that reason, we will add a few additional areas of knowledge that we believe will support both your KCNA journey, and the post-certification work activities you choose.

Additional Areas of Knowledge (outside of the official curriculum)

Once again, this section includes non-official topics, which means that they are not directly evaluated in the KCNA exam, but that will help you identify additional knowledge gaps depending on your background, and make passing this exam a bit easier for you. Here is the authors' selection, for your own analysis and preparation:

Part 6 - Kubernetes 101 Hands-on

Yes, the official curriculum includes Kubernetes fundamentals, and at this point it should be clear for you as a candidate that the exam won't include any interactive platform to test your Kubeneretes hands-on experience. But that does not mean that you won't need to know practical commands, or at least be able to identify them and know how to use them.

For this purpose, here are our recommendations:

- Bookmark, read, and understand the main kubectl commands. We will of course discuss them later in Chapter 6, but the official documentation includes those that you need to know, including the syntax and a very useful cheat sheet you can print and study.
- Play with the O'Reilly Kubernetes Sandbox. This and other sandboxes offer a one-click on demand workspace for you to start exploring and testing different commands and configurations.
- Leverage the existing interactive Kubernetes labs from O'Reilly. They are based on the amazing Katacoda platform, and they offer a variety of step-by-step scenarios (e.g. defining and deployment resources, launching nodes). A big part of them were developed by one of our industry experts, specifically Benjamin Muschko who will join us in Chapter 4.

Part 7 - Linux Fundamentals

You will get similar recommendations from our Chapter 3 expert, but knowing Linux and its terminal will make your exam and post-exam life easier. Let's see what you can do, in a pragmatic way, before taking your KCNA exam:

- Besides understanding Linux and its origins, you could focus on the command line, which is like the kubectl (a user interface without windows or icons, just text). For you to get some initial knowledge, you can leverage existing cheat sheets, or some applied tutorials for renown industry organizations.
- Same as before, you can leverage the O'Reilly Sandbox platform. Concretely, you have two options for a regular Linux command line, and a second version for Rocky Linux (in case you want to work with Red Hat-compatible environments). The O'Reilly Learning platform also includes specific Linux scenarios.
- Ideally, if you have time and you want to go deeper, you can always take the Linux Foundation Certified IT Associate (LFCA) exam, or at least take a look at the related intro level and free resources.

Part 8 - API-enabled Development

As a cloud native and Kubernetes professional, you will leverage Application Programming Interfaces (API) as a way to connect to remote services, that will include the Kubernetes API (a very important feature that we will explain later), but also other development-oriented APIs you may have as part of your end-to-end solution. The concept is pretty simple, but it can be a bit challenging or even abstract for newcomers. If you have never worked with APIs, we recommend you to spend some time, getting familiar with API-enabled environments by:

- Checking the vast list of API topics from the O'Reilly Learning platform.
- Leveraging didactic material from industry actors such as Postman (which is an API platform for building and using APIs), and have some 101 intro videos and documentation.

Part 9 - Telematics

You may not even know this, but there is an area of study called telematics that combines telecommunications, electrical engineering, and IT / computer science. One of the core key building blocks is what we call networking, which is the ability to connect diverse systems so they can operate interactively. For example, the internet is a network that connects your personal device with remote resources such as websites, and online platforms.

This area relies on relatively complex knowledge, traditionally related to engineering jobs. However, if you work with cloud native tools (including Kubernetes), the notion of networking will come to you at some moment. We will cover this topic in Chapter 6, but don't hesitate to take a look at the official K8s Networking documentation before.

Part 10 - Web Development

This one can be considered as a basic area of knowledge. Building web-based applications doesn't necessarily mean building regular client-facing websites. It can be used to create internal platforms that rely on the same kind of architecture and networking elements.

If you have never worked on this kind of project, you can leverage existing resources from both O'Reilly Learning, and other relevant actors, with special focus on the notions of client-server model, and web servers. Once again, these topics are not part of the evaluated knowledge of the KCNA exam, but understanding the fundamentals will help you conceptualize the technical details related to cloud native and Kubernetes.

Part 11 - Applied Usage

Last but not least, and as we saw in Chapter 1 with our inspirational success stories, one of the best ways to integrate the learnings from the different official and non-official areas of study of the KCNA exam is to understand how companies are using cloud native and Kubernetes, and their typical use cases. Here are a few examples, but you can continue exploring this in parallel with your regular exam study and preparation process:

Artificial Intelligence
: You can containerize a machine learning model, deploy it as a scalable API service for usage. The model can be deployed within a Kubernetes cluster, as this will help handle large volumes of requests, scale based on demand

Website Creation
: Imagine an e-commerce website with microservices like product catalog, user authentication, payment gateway, in which we containerize each microservice and deploy them in a Kubernetes cluster, so each service can work and scale independently, preventing features and adapting to peak demand moments

Cloud Computing
: Imagine a company working with two or three public clouds such as AWS, Azure, or GCP. If the company wants to avoid vendor lock-in and desires to utilize the same kind of resources via multiple cloud providers, or by combining on-premises and cloud, they can containerize applications, databases, etc.

Internal Development
: We can of course containerize our developed applications, and leverage tools we use as part of the development lifecycle. For example, if we want to implement a CI/CD pipeline, tools like Jenkins or GitLab CI will help us get the pushed code from the repository, then build, test and deploy a new container.

Edge Computing
: Same as cloud computing, imagine that you want to process data close to an IoT device (i.e. sensors) across different geographies. Lightweight Kubernetes distributions like K3s, you could deploy applications closer to the source, reduce latency, and ensure efficient data processing at the edge.

This concludes the first block of topics related to the KCNA exam, including its origins, structure, scope and format, key areas of knowledge, and some additional topics you may want to explore to complement your preliminary knowledge. Let's proceed now with your study plan and exam logistics.

Exam Study Plan

So now that you have an idea what the exam will cover and what is the weight for each section, how do you go about tackling it? The recommended study plan and checklist below is our plan.

This is what we recommend for an average test taker who may have some experience with Kubernetes and is looking at taking this exam for the first time. By no means is this a one-size fits all mandate. You can consider this a list of what we have found to work as a starting point to help you customize your learning plan for what works best for you.

Step 1 - Determine your Current Level of Knowledge

We have already discussed this between Chapter 1 and 2, it is all about being aware of our limitations, and adapting the study plan in consequence. How familiar are you with the topics covered in this exam? If you have never heard of Kubernetes until you picked up this book, then perhaps you have to spend a little more time studying for this exam than if you have been working with it for a year.

No matter what level you are at, we recommend you take a diagnostic of your level of knowledge and determine where your strengths and areas of focus are. The self-diagnostic questionnaire from Chapter 1 is a good place to start, but this is a continuous exercise, throughout your KCNA exam study journey.

If you are honest with yourself about what you know and don't know, how you do on this diagnostic test will showcase your current level of knowledge and determine your readiness to tackle the exam. So at this point, if you have not taken the test yet, we recommend that you bookmark the current page, go again to Chapter 1, and come back here with your results.

Step 2 - Prepare and Adopt an Study Schedule

Now you are equipped with knowing the areas you need to pay attention to, you can take the time you need to master these topics. We recommend creating a study plan that includes between 1-3 hours per day in the weeks leading up to the exam. Depending on your schedule and rigor, some students spend up to 15 hours per week for 4 weeks studying before the exam, but it really depends on your previous background, and ability to study and prepare for the exam. Concretely, ensure that you understand each topic section that is covered in the exam, and take the time to go over topics you are less familiar with. The ones outlined by your diagnostic test as areas of improvement, or those that you will learn from the first team in the next three chapters, are certainly what you should be focusing on.

Step 3 - Focus on the Right Material

This book is a comprehensive guide on the KCNA certification and exam. All the materials covered in this book are tailored towards laying the fundamentals to Kubernetes and preparing you for the KCNA exam. They go hand in hand with the candidate's handbook and other material published by the Linux Foundation. However, though we consider this book to be THE guide on the KCNA exam, we did not create the exam (and even if we did, the exam questions change regularly). The Linux Foundation and CNCF are the official sources of information regarding the exam, its rules and curriculum. Familiarizing yourself with the Cloud Native landscape through their online presence is helpful in getting an up to date understanding of this ever evolving open space.

Exam Logistics and What to Expect

The KCNA exam and its logistics preparation process is very specific to the The Linux Foundation online examination platform and rules. Especially for first time test takers, you will need to pay attention to these steps and get ready for the exam day. Note that this information is accurate as of the time this book is published, some details may change but the general process should be the same. Let's see the main steps:

Step 1 - Schedule Your Exam

All certifications and exams provided by The Linux Foundation can be scheduled on their online portal. The process is always the same, regardless of the exam:

1. Go to their Training and Certification platform https://training.linuxfoundation.org
2. Select My Training Portal, this is key to enter your own use space.

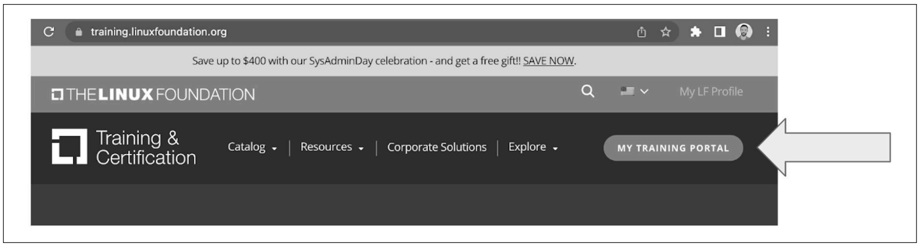


Figure 2-4. The Linux Foundation - My Training Portal

3. In the authentication page you can see in Figure 2-5, create an account (sign up) if you are a first time user, or Sign in with your credentials or SSO (i.e., existing Gmail. Facebook, Github, and LinkedIn accounts) if you already have one. If you're creating the account for the first time, a request to verify your email will be sent to the email address provided.

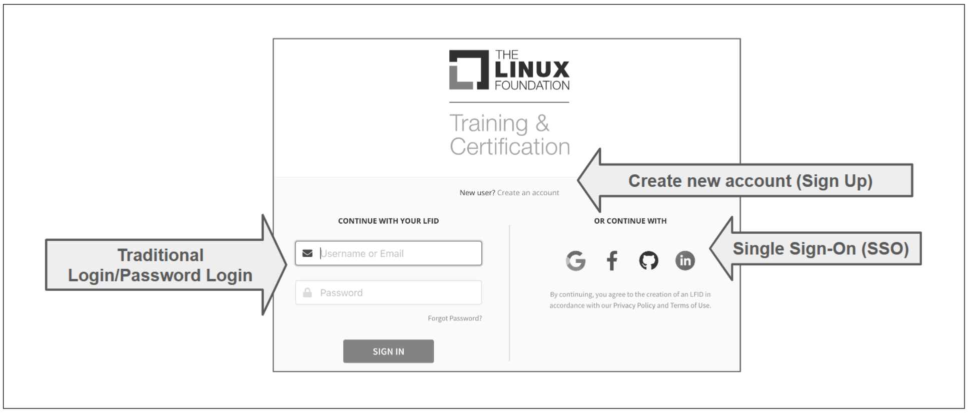
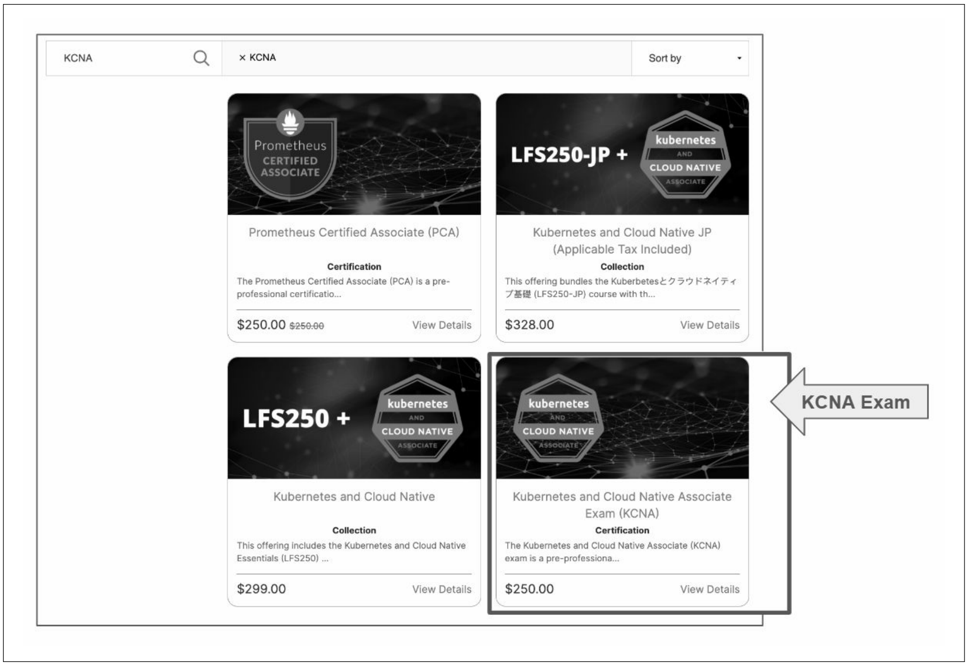

THE LINUX FOUNDATION
Training & Certification
New user? Create an account
Create new account (Sign Up)
CONTINUE WITH YOUR LFID
OR CONTINUE WITH
Traditional Login/Password Login
Single Sign-On (SSO)
Username or Email
Password
Forgot Password?
SIGN IN
By continuing, you agree to the creation of an LFID in accordance with our Privacy Policy and Terms of Use.

Figure 2-5. The Linux Foundation - Login and Sign-Up

1. From your new learning dashboard, scroll down and find the browse section. Once there, click on ***search for content***, and type "KCNA" to find the KCNA exam from the list of courses and exams available. As seen below in Figure 2-6, there are two options to register for the KCNA exam. You can register for the exam alone for $250, or opt for the collection that includes the exam and the Kubernetes and Cloud Native Essential Training (LFS250) for $299. As a test taker, you have the choice to take it in English or in Japanese.


KCNA
× KCNA
Sort by

Prometheus Certified Associate (PCA)
Certification
The Prometheus Certified Associate (PCA) is a pre-professional certificatio...
$250.00 ~~$250.00~~ View Details

Kubernetes and Cloud Native JP (Applicable Tax Included)
Collection
This offering bundles the Kuberbetesとクラウドネイティブ基礎 (LFS250-JP) course with th...
$328.00 View Details

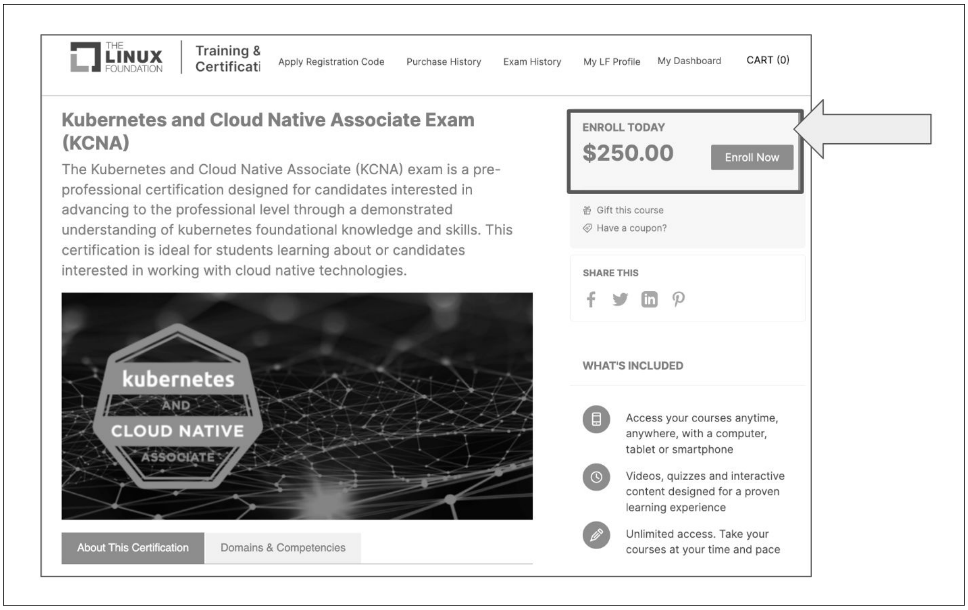
Kubernetes and Cloud Native
Collection
This offering includes the Kubernetes and Cloud Native Essentials (LFS250) ...
$299.00 View Details

Kubernetes and Cloud Native Associate Exam (KCNA)
Certification
The Kubernetes and Cloud Native Associate (KCNA) exam is a pre-professiona...
$250.00 View Details

Prometheus CERTIFIED ASSOCIATE
LFS250-JP +
kubernetes AND CLOUD NATIVE ASSOCIATE
LFS250 +
KCNA Exam

Figure 2-6. The Linux Foundation - KCNA Exam Options

1. After selecting the KCNA exam, from Figure 2-7 click ***Enroll Now*** and complete the checkout and payment process. You can make the payment online using a credit card, and/or add any coupon you may have from an ongoing campaign, or vouchers from your company's membership (it can be the case for CNCF and The Linux Foundation members).



Kubernetes and Cloud Native Associate Exam (KCNA)

The Kubernetes and Cloud Native Associate (KCNA) exam is a pre-professional certification designed for candidates interested in advancing to the professional level through a demonstrated understanding of kubernetes foundational knowledge and skills. This certification is ideal for students learning about or candidates interested in working with cloud native technologies.

$250.00

Access your courses anytime, anywhere, with a computer, tablet or smartphone

Videos, quizzes and interactive content designed for a proven learning experience

Unlimited access. Take your courses at your time and pace

Figure 2-7. The Linux Foundation - KCNA Enrollment

1. After you complete the process, you will be able to explore the checklist, and to read the exam handbook.
2. Finally, you can proceed to schedule your exam. This option is only available once you complete the preliminary steps. Pay attention to the date you prefer, and the timezone. Online exam booking platforms will adapt to your specific timezone, and let you choose among the available exam languages.

This series of steps conclude the initial logistics required for your KCNA exam. Now, let's proceed with the preparation details.

Step 2 - Exam Day Preparation

Prior to the day of the exam, you are highly encouraged to review the official candidate handbook to familiarize yourself with the requirements. All Linux Foundation Certification Exams including the KCNA are conducted online and monitored by a proctor during the exam session via streaming audio, video and screen sharing feeds.

Given the demanding nature of the exam, you want to make sure that your computer, the system and the internet bandwidth to work well during the period. Therefore, it is highly recommended that you take the Online Proctoring Compatibility Check and review the PSI Bridge Platform System Requirements to make sure your laptop meets the online system proctoring requirements. Note that the latest version of Google Chrome is recommended for the exam, and that you are not allowed to have other applications or browser windows running except the one on which the exam is being shown. You're allowed to use one active monitor, either built in or external. However, dual monitors are not allowed. It may seem like a lot of work just to make sure your computer is working fine during the exam, but it is better to make the necessary changes to your system when you have the time rather than stressing out at the last minute. The full list of system requirements is available here.

Given that your live audio and video will be taken (to proctor and review your KCNA exam) the candidates must consent for it, keeping in mind that the test session recordings are stored for no more than 90 days. Also, in order to be admitted to take the exam, you need to present an acceptable form of unexpired Government-issued original, physical identification (ID). Admissible forms of ID include passports, driver's license, and national identification. With the ID, you have to make sure that the room you're taking the test in is quiet, private and well-lit. You must be alone in the room without any notes or electronics other than your computer.

Similarly, the Linux Foundation mandates that the test takers comply with their examination rules. Some may be intuitive whereas the others are good to know beforehand. For example, your testing location must be clutter free with clear walls. No paper or print-outs hanging on the wall. The room must not be a public space such as a coffee shop or shared workspace. The candidate is not allowed to chew gum, eat or drink except for clear liquids from a clear, label-free bottle or glass. Once you sit down for the exam, you are not allowed to leave the desk or step out of the view of the webcam. You may be tempted to jot down an answer or a question on a piece of paper next to you, but note that the use of paper or electronic devices outside of the computer screen is not permitted during the exam. These misconducts are taken seriously. Certain violations of these rules may result in a warning from the proctor, while others can result in immediate termination of the exam.

To make it easier to prepare for the exam, the Linux Foundation published a checklist of steps the candidate must complete before the exam. The steps are available on your platform and the candidates must complete certain steps in order to be able to schedule the test or to take it. The My Portal platform is the one stop shop for all things you need to complete the certification exam. Apart from registering for the exam, you can also view your exam results there.

Finally, on exam day, after all the preparation you've done, it is time to do the exam. Ensure you show up and are ready to launch the exam within 30 from the appointment time otherwise, you may be considered no show and not able to reschedule.

You have 90 minutes to complete the KCNA exam. You need 75% or more to pass the exam. All Linux Foundation multiple choice question exams are conducted on a simple UI. You can review the screenshots here to familiarize yourself. To navigate between questions, you can click the previous or next button. You can also flag an item for later review which will be highlighted on the Review Screen. Once all the questions are answered, a prompt to click Review Exam. The exam will end when the timer completes or when you click Finish Exam.

Step 3 - Post-Exam Logistics

The results do not follow immediately after the exam is completed. Once available, you will receive a notification via the email you used to register for the exam. Only at that moment, you will be able to see your exam result on the My Portal platform.

If you pass, then congratulations, your certification is valid for three years. You will have the possibility to add the certification as a verifiable badge on LinkedIn and Credly, which is a platform that officially manages and recognizes your digital credentials. You can choose to share it with potential employers or anyone you wish. Once shared, they'll be able to see that your KCNA certification is truly issued by the provider and is valid. Add your certification to your resume and show it off, but keep in mind that you will need to create a Credly account with the same email you already used for The Linux Foundation access (otherwise you will be able to add and link different email addresses from your Credly Profile settings).

However, if you did not pass the exam on your first try, do not sweat. The exam comes with one retake opportunity (same as any other The Linux Foundation exams). If this happens, you will leverage your exam experience, and actual understanding of the kind of exam questions to continue study, focus on your weakest areas of knowledge, and take the exam again.

This is everything in terms of exam process and steps. Let's now see the potential learning and certification paths, beyond the official certification roadmaps, in which the authors of this KCNA Study Guide will provide you with an applied industry-side point of view. This means, what you can do before and after taking the KCNA exam to gain knowledge and experience, and to finally be part (if you are not still there) of the next annual reports that highlight success and great professional opportunities.

Potential Certification Paths

Although the KCNA exam is currently not a prerequisite for other professional advanced certificates to follow, the fact that it focuses on the functional concepts related to the technology makes it the go-to first step certification every practitioner should consider obtaining. This exam's focus on laying solid groundwork is why the CNCF recommends it as a good preparation for further advanced exams and certifications in this field. But there are also some preliminary steps that may help you achieve the goal.

In Figure 2-8, you can see the customized end-to-end view of your potential certification and learning paths as an KCNA candidate, before and after passing this exam, so you can contextualize where the KCNA stands and how you can both prepare it, and make the most of it for your future learning journeys:

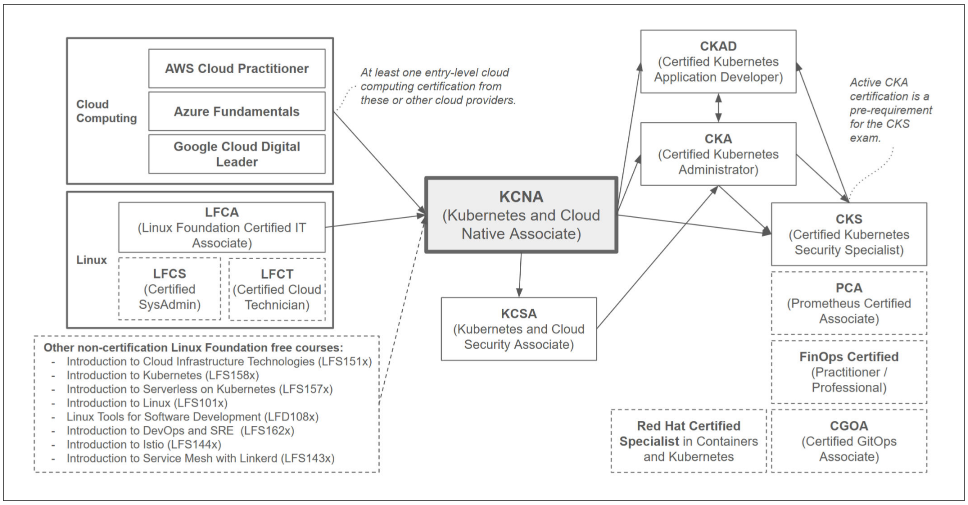

Cloud Computing
AWS Cloud Practitioner
Azure Fundamentals
Google Cloud Digital Leader
At least one entry-level cloud computing certification from these or other cloud providers.
Linux
LFCA (Linux Foundation Certified IT Associate)
LFCS (Certified SysAdmin)
LFCT (Certified Cloud Technician)
Other non-certification Linux Foundation free courses:
- Introduction to Cloud Infrastructure Technologies (LFS151x)
- Introduction to Kubernetes (LFS158x)
- Introduction to Serverless on Kubernetes (LFS157x)
- Introduction to Linux (LFS101x)
- Linux Tools for Software Development (LFD108x)
- Introduction to DevOps and SRE (LFS162x)
- Introduction to Istio (LFS144x)
- Introduction to Service Mesh with Linkerd (LFS143x)
KCNA (Kubernetes and Cloud Native Associate)
KCSA (Kubernetes and Cloud Security Associate)
CKAD (Certified Kubernetes Application Developer)
CKA (Certified Kubernetes Administrator)
CKS (Certified Kubernetes Security Specialist)
Active CKA certification is a pre-requirement for the CKS exam.
PCA (Prometheus Certified Associate)
FinOps Certified (Practitioner / Professional)
Red Hat Certified Specialist in Containers and Kubernetes
CGOA (Certified GitOps Associate)

Figure 2-8. Potential Certification Paths for KCNA Candidates

This overview contains some vendor-related certifications from different providers, mainly for illustrative purposes but also as a way to explore potential industry specializations. Also, once you conclude your KNCA journey, you will have a clear understanding of administration, development, monitoring, security, operations, and cost management topics that will help you choose other potential areas of certification. Last but not least, The Linux Foundation has an official IT Career Roadmap that includes the KCNA and other certifications, so you can take a look and prepare your very own certification steps.

Summarizing, passing the KCNA exam is a signal to potential employers that you've embarked on the journey of cloud native computing learning, but it is just the beginning. Here is our selection of exam that you may want to consider in your learning journey:

Kubernetes and Cloud Security Associate (KCSA)

The KCSA is a certification exam for Kubernetes and cloud-native security at associate level, same as the KCNA. It was initially released in 2023, and it can be seen as a complementary option, to continue diving into other Kubernetes-related topics, before going to hands on, advanced certifications. Basically, it will test your ability to configure, monitor and assess the security of Kubernetes clusters, and it covers topics such as security policies, controls, risks, vulnerabilities, incident response, forensics, and best practices. Our recommendations:

It is a good option to continue your certification path after the KCNA, as you will leverage your Kubernetes knowledge here too. It is a pre-professional exam that prepares you for the CKS we will explain in this same list.

Based on the industry trends from The Linux Foundation and the CNCF, cybersecurity and container security are some of the hottest topics out there, as there is a clear knowledge gap and a lot of demand. You may analyze if security-related topics are interesting to you.

Check the exam page to explore all details and pricing.

Certified Kubernetes Administrator (CKA)
: This online exam is especially tailored to learners who wish to demonstrate their technical knowledge on Kubernetes administration. It's a performance-based test that requires solving multiple tasks from a command line running Kubernetes. With 30% of the content being on hands-on troubleshooting intermediate level issues commonly faced by administrators, the exam is the next gateway to hands-on certifications. From a KCNA learner point of view, here are our recommendations:

The CKA is not an easy exam. The knowledge requirements difference between the KCNA and this certification goes from knowing the fundamentals of cloud native and Kubernetes, to actually having several years of experience in managing Kubernetes-enabled systems.

Based on the same trends that justify the value of the KCNA certification, obtaining the CKA is a great career advantage for you, but also for companies looking for talent and wanting to become a Kubernetes Certified Service Provider, as they will need at least three employees CKA certified to be eligible. That along with the scarcity of available CKA professional (around 40,000 CKAs worldwide in 2022), and the projections of Kubernetes adoption (e.g. Gartner predicts that by 2027, more than 90% of global organizations will be running containerized applications in production), it will guarantee you great professional opportunities and benefits.

The exam details, requirements, and the official exam guide are available via The Linux Foundation's exam page.

Certified Kubernetes Application Developer (CKAD)
: If you are planning to pursue a career as a Kubernetes-enabled application developer, then this certification is next on your list of training. This 2-hour exam tests practical knowledge on application design, build, deployment, all the way to maintaining its security policies. As opposed to the KCNA exam, the CKAD candidate is encouraged to have hands-on knowledge of container runtimes and microservice architectures, from a software developer perspective

If you decide to go from KCNA to these advanced certifications, the choice between the CKA and the CKAD may be your first big decision. In reality, it is pretty simple. Are you planning to leverage Kubernetes technologies, or do you prefer to play an admin role in K8s environments? Depending on your preference, you may choose one or the other.

Same as KCNA, CKA, the market demand is clearly justified. Now, if you choose the CKAD, you may leverage the skills for different kinds of activities: typical software development, creation of new AI-enabled solutions, website development and management, etc. The developer side of the CKAD will be beneficial for different kinds of profiles, check the official curriculum to see if it is your case. Based on statistics, the CKAD is one of the popular options.

The exam information and requirements is, as usual, available at The Linux Foundation exams website.

Certified Kubernetes Security Specialist (CKS)
: While the two certifications and the KCNA do not have a prerequisite, the CKS requires the candidate to have an active (non-expired) CKA certification. Does it make this an advanced certification? Well, the Linux Foundation marks this performance based exam as being suitable for intermediate level candidates, but in the big scheme of things, this one is pretty advanced. The certificate aims at testing competence on a broad range of best practices for securing container-based applications and kubernetes platforms during build, deployment and runtime. The setup for this test differs from the ones mentioned prior as the test is based on a real world environment through a cloud security simulation platform.

The CKS is for sure not your next step from the KCNA, but it can be your end-goal if you decide to go into the cloud native security road. The job market demand is clearly favorable for this type of role, however if it will take you a few certifications to get here.

Same as before, all information is available at The Linux Foundation's CKS website.

Summarizing, looking at these four certifications give us a glimpse into a large set of specializations that Kubernetes practitioners can develop for themselves. Note that these certifications are only the starting point to demonstrating your knowledge, as cloud native practitioners learn industry best practices for their respective needs through hands-on experience and industry specific training.

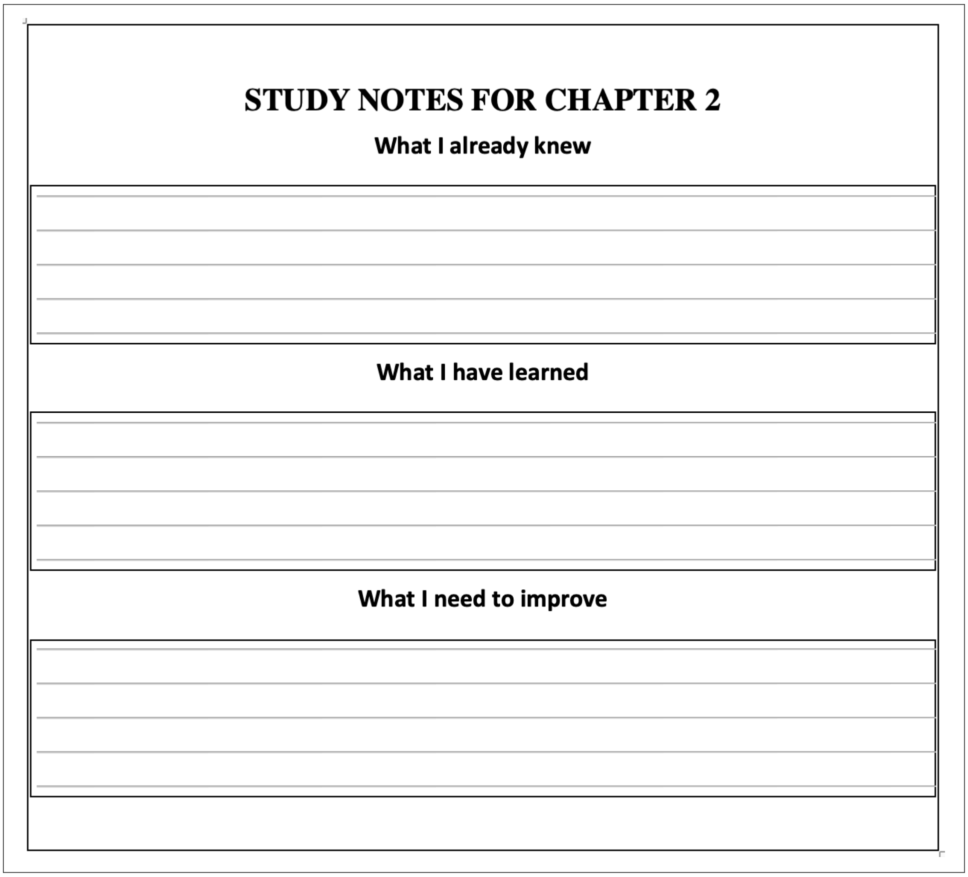
In the next section, we will discuss with another great industry expert. Concretely, Jim Zemlin, Executive Director at The Linux Foundation. We will explore the vast certification ecosystem related to the KCNA, the impact of the KCNA exam for the general talent generation process, and insider information to enable you to make the most of this upskilling journey.

PLACEHOLDER CONCLUSION - Chapter 2

We are now concluding our second Chapter. The main goal was to provide you with all required information to noy only pass the exam, but to also see the possibilities beyond the KCNA and how they will leverage the skills you develop in this KCNA journey.

Concretely, we have analyzed the origins and details of the KCNA exam, its end-to-end logistics, as well as the related learning opportunities and other certification paths. This should give you a good overview of the different options in front of you, as well as the specifics of the KCNA certification, its official curriculum, and the relation with all the preliminary topics we discussed in Chapter 1.

The next Chapter 3 will focus on the CNCF, its organizational details, some CNCF projects besides Kubernetes, and of course exclusive expert insights from a very relevant CNCF actor. Let's get started.


STUDY NOTES FOR CHAPTER 2

What I already knew

What I have learned

What I need to improve

Figure 2-9. Study Notes - Chapter 2

CHAPTER 3

CNCF and the New Era of Cloud Native

As you know already, the KCNA exam covers not only Kubernetes topics, but also other cloud native fundamentals. That includes the CNCF structure, which is related to how an open source community works, including its governance, project incubation processes, individual contribution, etc. We will cover all that in Chapter 3. As a KCNA candidate, this information is important for you because it is part of the scope of the exam questions, but it is also very relevant for your cloud native journey.

The CNCF Origins

To set the stage for the new era of cloud native, let's first examine the origins of the CNCF. The CNCF is a non-profit that was created in 2015 to push for the use of cloud native computing and to give cloud native technologies a place to call home. It was set up by a bunch of tech companies like Google, Red Hat, and IBM because they saw the need for an organization to speed up the adoption of cloud native technologies and best practices.

From CNCF's website: "*The Cloud Native Computing Foundation (CNCF) hosts critical components of the global technology infrastructure. We bring together the world's top developers, end users, and vendors and run the largest open source developer conferences. CNCF is part of the nonprofit Linux Foundation*".

Currently, the Foundation is supported by companies and individual members who are committed to advancing the cloud native landscape. The CNCF is governed by a board of directors and is overseen by the The Linux Foundation, the non-profit organization that promotes the use of open source software. The CNCF website itself is a great source of information and resources.

Reality is that the establishment of the CNCF in 2015 had a great impact on the cloud native industry, as it provided a central hub for the development and promotion of cloud native technologies and best practices. Before that, there was no single organization that was dedicated to advancing the cloud native landscape, and many different technologies and approaches were being developed in isolation.

The CNCF has played a key role in standardizing and consolidating the cloud native ecosystem, and has helped to drive the widespread adoption of cloud native technologies by providing a forum for collaboration and sharing of ideas and knowledge. The CNCF has also helped to establish a set of common practices and standards for cloud native development, which has made it easier for organizations to adopt cloud native approaches and technologies.

Overall, the birth of the CNCF in 2015 had a significant and lasting impact on the cloud native industry, and has helped to accelerate the adoption and development of cloud native technologies and practices, including Kubernetes but also other great cloud native projects we will explore in this chapter, as they will be relevant for your exam preparation.

Timeline of The Cloud Native Industry

Here are a series of events that describe the evolution of the cloud native industry before and after the creation of the CNCF, and that we can see later in Figure 3-1:

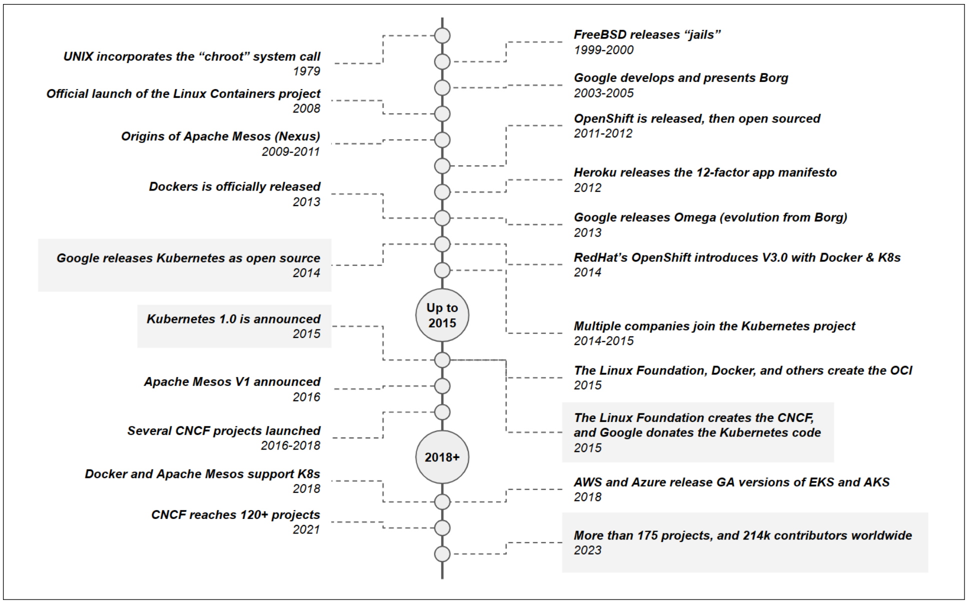

UNIX incorporates the "chroot" system call 1979
Official launch of the Linux Containers project 2008
Origins of Apache Mesos (Nexus) 2009-2011
Dockers is officially released 2013
Google releases Kubernetes as open source 2014
Kubernetes 1.0 is announced 2015
Apache Mesos V1 announced 2016
Several CNCF projects launched 2016-2018
Docker and Apache Mesos support K8s 2018
CNCF reaches 120+ projects 2021
Up to 2015
2018+
FreeBSD releases "jails" 1999-2000
Google develops and presents Borg 2003-2005
OpenShift is released, then open sourced 2011-2012
Heroku releases the 12-factor app manifesto 2012
Google releases Omega (evolution from Borg) 2013
RedHat's OpenShift introduces V3.0 with Docker & K8s 2014
Multiple companies join the Kubernetes project 2014-2015
The Linux Foundation, Docker, and others create the OCI 2015
The Linux Foundation creates the CNCF, and Google donates the Kubernetes code 2015
AWS and Azure release GA versions of EKS and AKS 2018
More than 175 projects, and 214k contributors worldwide 2023

Figure 3-1. Cloud Native Timeline

Concretely, there were several key events that led to what we understand as cloud native and containerization today. Starting from the old times:

The containerization origins

- 1979 - UNIX (a family of operating systems that derive from the original AT&T UNIX OS, developed by Bell Labs) releases the chroot system call in its version 7. chroot is a command in Unix and Linux that allows changing the root directory for a running process and its children, often used for sandboxing, testing, and recovery purposes. The idea was to provide an isolated disk space where processes think it's their root, but in reality, it's just a subdirectory of the real root filesystem. This can be seen as the predecessor of what modern containerization is, even if it had more limited features.
- 1999-2000 - FreeBSD (an operating system used to power servers, desktops, and embedded platforms) releases jails, which are an evolution from the chroot, and relatively similar to today's containers.
- 2003-2004 - Google develops the Borg system. Technically, Borg was a "cluster manager that runs hundreds of thousands of jobs, from many thousands of different applications, across a number of clusters each with up to tens of thousands of machines". Google released a paper that explains its technical details. Basically, Borg was the predecessor of Kubernetes, but it wasn't open source yet.
- 2008 - The Linux Containers (LXC) project is officially released, as an umbrella project behind initiatives such as LXC, LXCFS, distrobuilder, libresource and

lxcri, with the goal of offering distributions and vendor neutral environment for the development of Linux container technologies.

- 2012 - Heroku introduces the 12-Factor app methodology, which is a well known set of best practices designed to enable the creation of software-as-a-service (SaaS) applications. It provides guidelines for building web applications that can be easily deployed to the cloud, scale gracefully, and remain resilient in the face of changing infrastructure and environments.
- 2013 - Docker is released, setting a new industry reference for containers. It was introduced to the public by Solomon Hykes at a PyCon event. It was initially a project within dotCloud, a platform-as-a-service company. The introduction of Docker brought the concept of containers to the masses, making it simpler to create, deploy, and run applications using containerization. One of the keys of its success was the almost immediate release of its code as open source, weeks after the announcement. You can read more about its origins at Docker: Up & Running, 3rd Edition (O'Reilly).
- 2013 - Google presents Omega, a research project within the company, focused on the next generation of cluster management, which can be seen as an evolution of Borg.
- 2014 - Google releases Kubernetes as an open-source version of Borg, which means opening their code to all contributors and other companies. During the first months, Joe Beda did the first Github commit, and companies like IBM, RedHat, Microsoft, and Docker joined the Kubernetes community. It was the beginning of the Kubernetes era.

The rise of Kubernetes and the CNCF

- 2014 - As an example of the initial traction from the main industry actors, and after a series of collaborations with the Google team, RedHat decided to support container orchestration with Kubernetes for their OpenShift V3.0. This is very illustrative as the previous versions were based on Linux Containers and other in-house developments from RedHat, but both the community support and the specialized knowledge of the K8s Google team (thanks to their extensive experience with Borg) were key factors for other companies to join.
- 2015 - Google and the Linux Foundation establish the Cloud Native Computing Foundation (CNCF) and Google donates the Kubernetes code. The idea behind it was to have CNCF govern the future open source development of K8s and interoperability and performance for both public and private clouds, and on-premises.
- 2015 - Kubernetes 1.0 is released, with a total of 14,000 commits and more than 400 contributors. This version included improvements on App Services, networking and storage, cluster management, performance, and stability.

- 2015 - The Linux Foundation, with Docker and other organizations, launched the Open Container Initiative (OCI) to create open industry standards around container formats and runtimes. As part of the initiative, Docker donated its container format and the runC runtime (more information here).
- 2016 - The Pokemon GO video game uses Kubernetes via Google Cloud, becoming a very notorious success story because it was the largest K8s deployment on Google Container Engine at the moment.
- 2016-2018 - During this period, several cloud native projects were launched, including Helm (package manager), KOps (Kubernetes operations), or Minikube (for local implementations). Other projects like Istio came from the collaboration amongst Google, IBM and Lyft.
- 2018 - Kubernetes becomes the first CNCF graduated project, and both Apache Mesos and Docker announce support for it. The adoption trend at this point was clear.
- 2018 - At that point, all the main hyperscalers, such as Google, Amazon, Microsoft, and IBM had released cloud-enabled managed services to support Kubernetes.
- 2021-2023 - CNCF went from 120 to 170 projects, with more than 214,000 contributors around the world. The new era of the cloud native ecosystem is a reality.

These are just a few examples of how the cloud native industry, led by CNCF and their contributors, has evolved in just a few years, reaching alignment around containers-related formats and standards, with high levels of collaboration thanks to the open source dynamics. Let's now see the internal governance and way to work of the CNCF.

Community dynamics and governance

As an open-source organization, the CNCF plays a pivotal role in guiding the future of cloud computing and the technologies that underpin it. As a leading foundation, it acts as both a steward and advocate for cloud-native technologies by providing a structured environment, resources, and expertise, and it ensures that open-source projects can grow, mature, and thrive in a rapidly changing technological landscape. Some key aspects that explain their central role:

Multi-stakeholder structure

The CNCF relies on a series of groups with different roles, and a distributed structure that enables accountability and collaboration. We will see the different committees and groups.

Vendor Neutralism
One of the primary values of CNCF is to be vendor-neutral. This means that no single company has undue influence over a project, regardless of their participation and contribution to the project and its code. Instead, a community-driven approach ensures that projects remain open, free, and not tied to any single vendor's commercial interests.

Project hosting and lifecycle
CNCF hosts multiple open-source projects that align with its mission. Those projects go through a defined life cycle that includes Sandbox, Incubating, and Graduated stages. We will explore this later in this same chapter, but the idea behind this model is to adapt the support to the projects, at each phase of their growth.

Promotion and upskilling
CNCF organizes and sponsors events, providing forums for collaboration, networking, and knowledge sharing. They also support training and certification programs to bolster cloud-native skills in the industry.

Community and Collaboration
CNCF emphasizes building a collaborative community. It provides guidelines on governance, a code of conduct, and resources to ensure projects foster inclusive environments.

Continuous advocacy
As the umbrella organization, CNCF plays a crucial role in marketing and promoting its projects. Projects under the CNCF umbrella receive assistance in reaching a broader audience, which can be pivotal for open-source initiatives.

Regarding the structure, the CNCF is governed by a board of directors (officially called the *Governing Board*, or GB) and is overseen by the Linux Foundation. That board of directors is responsible for setting the overall direction and policies of the organization, as well as approving new projects and members. These directors are representatives from member companies, individual members, and the Linux Foundation.

Besides the board, as we can see in Figure 3-2, there are several committees and working groups that are responsible for specific areas of the organization, made up of volunteers from the CNCF member companies and individual members. They are responsible for things like project management, marketing, and events.

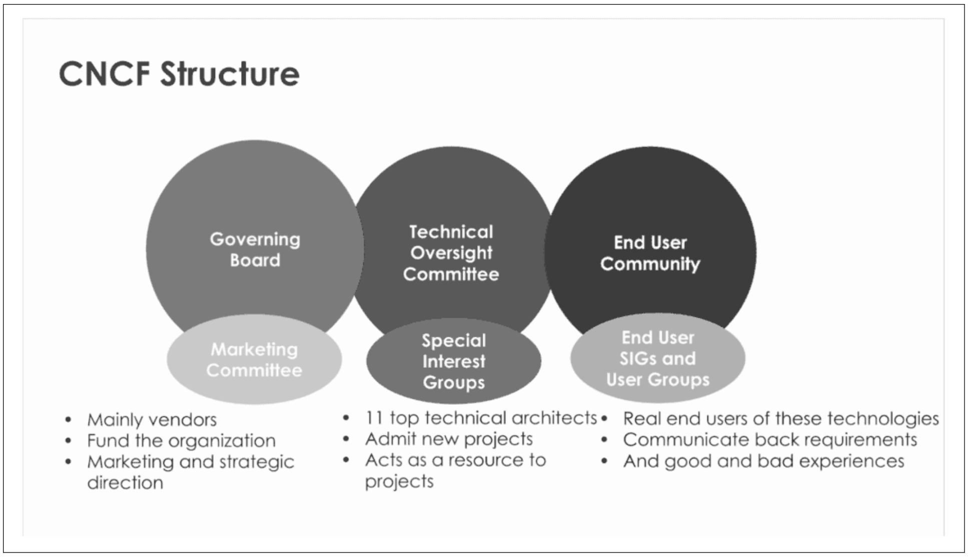

CNCF Structure
Governing Board
Technical Oversight Committee
End User Community
Marketing Committee
Special Interest Groups
End User SIGs and User Groups
• Mainly vendors
• Fund the organization
• Marketing and strategic direction
• 11 top technical architects
• Admit new projects
• Acts as a resource to projects
• Real end users of these technologies
• Communicate back requirements
• And good and bad experiences

Figure 3-2. CNCF Structure (source: CNCF)

Concretely, besides the governing board, there are other key elements for the CNCF structure that help make the community dynamics work:

Technical Oversight Committee (TOC)
: A group of technical experts who oversee the technical direction, provide neutral guidance, and evaluate the projects submitted to CNCF. The TOC plays a critical role in steering the foundation's technical vision and direction, and it's responsible for the technical leadership and decision-making in different areas such as project oversight, and connection with the rest of CNCF stakeholders.

End User Community
: The group of cloud native adopters that provides insights and feedback on the direction of cloud-native technologies, ensuring that projects remain relevant to real-world needs. From CNCF's website, the end user community is a group of "experienced practitioners who help power CNCF's end user-driven open source ecosystem, steering production experience and accelerating cloud native project growth". You can explore the full list of end user companies, including those from the previously mentioned success stories.

Special Interest Groups (SIG)
: These groups focus on special topics based on areas of interest or needs from the community, which are not directly related to one single project. Transversal topics such as contributor strategy, storage and security, network, and others are part of the existing group of SIGs.

Other Technical Groups

The CNCF hosts a number of technical advisory (TAG) and working (WG) groups. While TAGs are permanent and intended to support technical activities, WGs are meant to accomplish a specific goal and disband once that goal is achieved. As a community member, you can find existing TAGs and propose new WGs.

Overall, the CNCF governance structure is designed to ensure that the organization is transparent, accountable, and responsive to the needs of its members and the cloud native community. Part of their great mission is to create a series of resources that will be very useful for your KCNA journey, let's explore them.

Key CNCF resources

The CNCF has a growing set of resources for companies and individuals to learn about cloud native technologies, enabling you to interact with other actors and to join the cloud native practitioner community. We will highlight their usefulness for your KCNA upskilling and general learning. Some of the resources that the CNCF provides to the community include:

Community Forums

The CNCF boasts an active community that collaborates across various platforms and channels, by maintaining a number of online platforms where members can ask questions, share information, and collaborate on projects. Concretely:

Online discussion

The CNCF community has an active Slack workspace where members can discuss topics related to cloud-native technologies, ask questions, and share insights. Within the CNCF Slack, there are channels dedicated to specific CNCF projects like Kubernetes, Prometheus, and others. Recommendations:

- Join the cloud-native.slack.com workspace, and explore the different channels. For your KCNA journey, the #certification channel will help you connect with other exam candidates and post any questions.
- In addition to the CNCF Slack, you may want to take a look at the dedicated Kubernetes Slack workspace, which is not managed by CNCF but includes the #kcna-exam-prep and #kubernetes-novice channels that contain good advice and resources too.

Mailing Lists

CNCF uses mailing lists extensively for asynchronous communication, including general CNCF mailing lists and project-specific lists where discussions about development, issues, and updates related to that project take place. Recommendations:

- Join the general mail list to get updates from CNCF, as they usually include recommendations and opportunities that will support your KCNA and cloud native journey.
- Explore the other subgroups to join related mail lists, more specific to CNCF projects, working groups, etc. There is nothing specific to the KCNA but it will keep you updated with relevant community and project-related information.
- The CNCF mailing list system is similar to the one from The Linux Foundation. It basically relies on the groups.io platform to provide scalable email distribution and subscription. In your case, if you have never used it, you will need to create a free account before joining the general mail list and the subgroups.

Repositories

CNCF-related topics are mostly hosted on CNCF's GitHub. As any other repo, it contains code, issues, pull requests, and discussions related to that project. As part of the official repo, there is a feature called Github Discussions that includes some forum-kind of discussions for each subfolder of the repo. Major projects have their own repository, for example:

- Kubernetes - https://github.com/kubernetes/kubernetes
- Prometheus - https://github.com/prometheus/prometheus
- Helm - https://github.com/helm/helm
- We will explore these and other CNCF-related projects by the end of Chapter 3.

Community Meetings

Regular community meetings are held for CNCF projects, SIGs, and TAGs. These meetings often take place over Zoom and are open for anyone to join. Agendas are typically shared ahead of time, and recordings or minutes of the meetings are often made available for those who couldn't attend, and the calendar is publicly available for you to check the next meetings. As a KCNA candidate, it may be good for you to choose and join some of these meetings, to understand the work dynamics of these groups.

Social Channels

The CNCF has a YouTube channel where it uploads recordings of webinars, community meetings, conference talks, and other relevant content. This is a great resource for those looking to catch up on what's happening within the CNCF ecosystem, and you can leverage it to watch past events and to understand how the CNCF works. Also, the CNCF has a presence on platforms like Twitter (including a second account for the Student Community), LinkedIn, and others. These channels are used for announcements, sharing updates, and engaging with the broader tech community. Last but not least, the CNCF Blog is a great source of updated information, news, and applied cases.

In person and online events

The CNCF hosts a number of events throughout the year, which brings together experts and practitioners from across the cloud native ecosystem to share ideas and knowledge. The CNCF also supports regional events and meetups around the world (feel free to explore the upcoming ones by location). Concretely:

KubeCon + CloudNativeCon

This is the flagship conference of the CNCF and focuses on Kubernetes (thus "KubeCon") and other cloud-native technologies. The event typically happens multiple times a year, rotating between North America, Europe, and Asia Pacific regions. At KubeCon + CloudNativeCon, attendees can expect keynotes, technical sessions, workshops, and various networking opportunities.

As a cloud native and KCNA learning, you should pay attention to any scholarship or sponsorship opportunity to join these events. CNCF usually encourages diversity by bringing practitioners from all around the world.

Kubernetes Community Days

Organized by local Kubernetes communities, the K8s Community Days events focus specifically on Kubernetes and its ecosystem. They are meant to promote face-to-face collaboration and provide local communities an opportunity to network and learn from one another.

Webinars

Throughout the year, CNCF hosts various webinars that focus on specific projects, technologies, or trends within the cloud-native ecosystem. These webinars are usually free to attend and provide a great way for community members to stay updated without traveling to a physical event.

CNCF End User Events

These events are tailored for end users of cloud-native technologies, providing them a platform to share their experiences and learn from each other.

CloudNativeDays

These are smaller, localized events that might be seen as regional or more focused versions of KubeCon. CloudNativeDays events are typically organized in collaboration with local cloud-native communities.

Project-Specific Events

Given the wide range of projects under the CNCF umbrella, there are also project-specific events, meetups, and summits that take place. For example, there might be a Prometheus conference or a Fluentd summit.

Cloud Native Security Day, Cloud Native Storage Day, and others
: These are co-located events often held alongside KubeCon + CloudNativeCon, focusing on specific areas within the cloud-native ecosystem, like security or storage.

Educational resources

The CNCF offers a range of educational resources, including online courses, webinars, and tutorials, to help people learn about cloud native technologies and best practices. As a KCNA candidate, these resources are gold for you to understand cloud native topics, and to prepare your exam:

The CNCF Landscape,
: An interactive tool that allows you to get a single view of CNCF-related actors and projects, with filters by provider, CNCF project, industry, etc. It is (really) full of information so you will need to navigate it, but it is a very rich and always updated source of information. For purely illustrative purposes, because it cannot be shown in just a single page (or even a website), you can see in Figure 3-3 what the landscape looks like, or you can simply visit it by typing l.cncf.io:

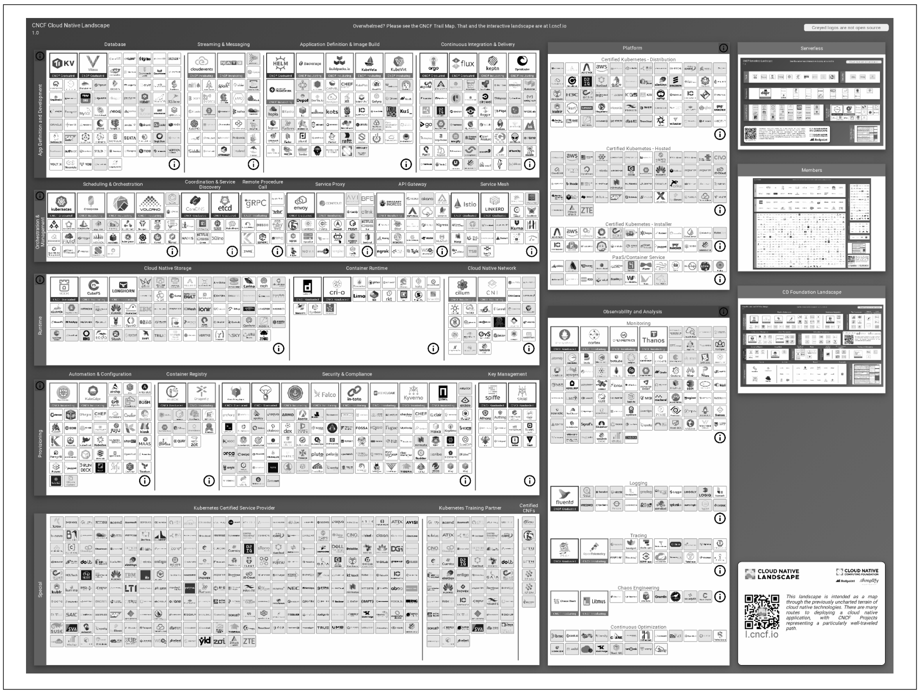


Figure 3-3. CNCF Landscape (source: CNCF)

The CNCF Guide

As an extension of the landscape, this extensive guide summarizes the key aspects of the CNCF community, highlighting the areas of knowledge (including technical 101 sections) and the most relevant projects, so you can see it as *one of the key source of information to prepare the KCNA exam* (actually, an amazing reference for you to save via Ctrl+P and keep it close to this KCNA Study Guide).

The official CNCF Glossary

This online glossary is the source of official descriptions for cloud native terms from the KCNA certification. It includes descriptions for most of the topics that are part of the exam, and the good news is that the community has evolved it a lot during the last years, including multi-language options (check the top-right menu of the glossary web) to see the terms in other languages such as Spanish, Portuguese, Italian, Hindi, Bengali, Dutch, etc. This, along with the official Kubernetes Documentation, are great complementary readings, and a clear reference for the kind of questions you will get on exam day.

The CNCF End User Tech Radars

These technology radars are great for you and the rest of the community to understand the level of adoption related to specific projects and tools. Basically, CNCF asks several end user companies about these tools, to understand how they are using them, and their recommendation. Concretely, the radars have three levels:

Adopt

The CNCF End User Community wholeheartedly endorses this technology. After extensive use across various teams over extended durations, it has consistently demonstrated reliability and value.

Trial

The CNCF End User Community has experienced positive outcomes with this technology. We suggest giving it a thorough evaluation.

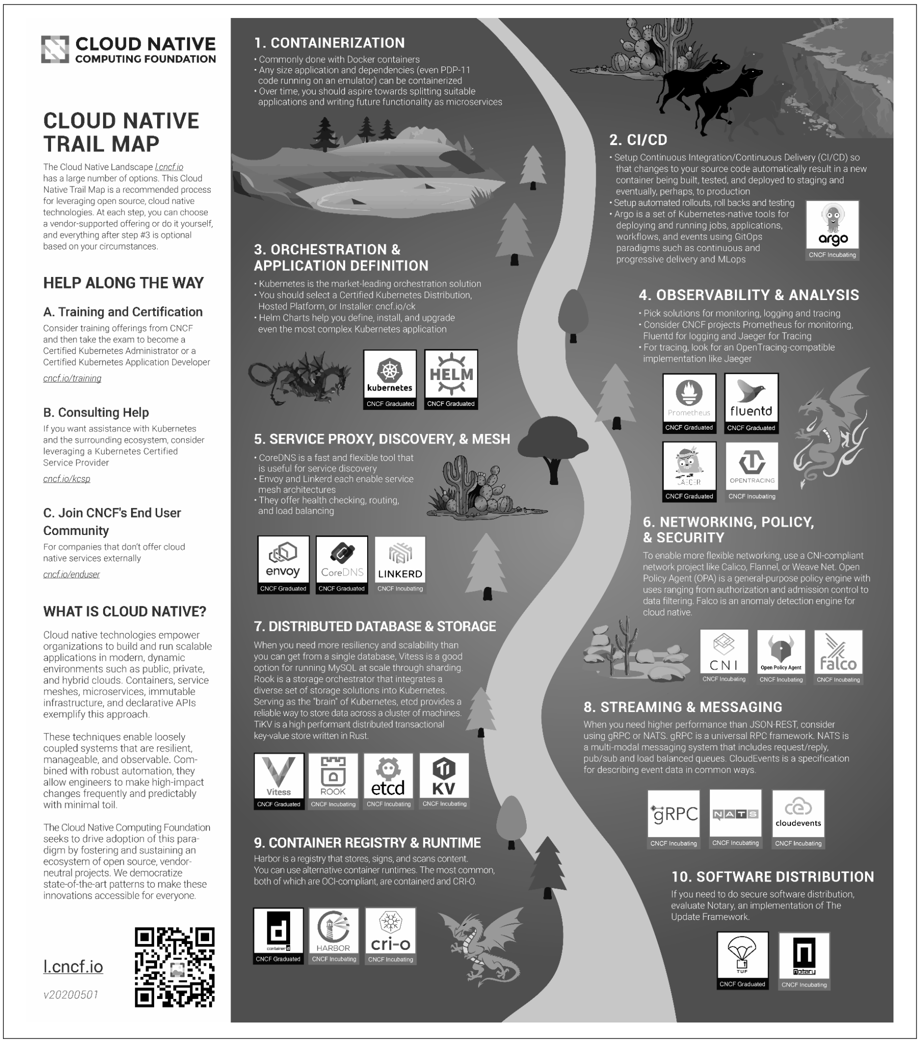
Assess

The CNCF End User Community has tested and sees potential in this technology. We advise considering it when it aligns with particular requirements in your project.

There are several themes for the technology radars (based on the recommendations from the community) including DevSecOps, multicluster and secret management, database storage, observability, and continuous delivery. All of these can be great indicators for KCNA learners like you to understand the current adoption trends, and the important projects and tools to learn about.

The Cloud Native Trail Map

Another key piece of knowledge for your KCNA exam preparation. The trail map is a visual description of how adopter enterprises can start their cloud native journey. This is relevant for you as a KCNA candidate not only because of the explanation of the different cloud native steps, but also the CNCF project examples of solutions available for each of the steps (e.g. Argo for CI/CD). We will explain all these projects later in Chapter 3.



CLOUD NATIVE COMPUTING FOUNDATION

CLOUD NATIVE TRAIL MAP

The Cloud Native Landscape l.cncf.io has a large number of options. This Cloud Native Trail Map is a recommended process for leveraging open source, cloud native technologies. At each step, you can choose a vendor-supported offering or do it yourself, and everything after step #3 is optional based on your circumstances.

HELP ALONG THE WAY

A. Training and Certification

Consider training offerings from CNCF and then take the exam to become a Certified Kubernetes Administrator or a Certified Kubernetes Application Developer

cncf.io/training

B. Consulting Help

If you want assistance with Kubernetes and the surrounding ecosystem, consider leveraging a Kubernetes Certified Service Provider

cncf.io/kcsp

C. Join CNCF's End User Community

For companies that don't offer cloud native services externally

cncf.io/enduser

WHAT IS CLOUD NATIVE?

Cloud native technologies empower organizations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds. Containers, service meshes, microservices, immutable infrastructure, and declarative APIs exemplify this approach.

These techniques enable loosely coupled systems that are resilient, manageable, and observable. Combined with robust automation, they allow engineers to make high-impact changes frequently and predictably with minimal toil.

The Cloud Native Computing Foundation seeks to drive adoption of this paradigm by fostering and sustaining an ecosystem of open source, vendor-neutral projects. We democratize state-of-the-art patterns to make these innovations accessible for everyone.

l.cncf.io

v20200501

1. CONTAINERIZATION

- Commonly done with Docker containers
- Any size application and dependencies (even PDP-11 code running on an emulator) can be containerized
- Over time, you should aspire towards splitting suitable applications and writing future functionality as microservices

2. CI/CD

- Setup Continuous Integration/Continuous Delivery (CI/CD) so that changes to your source code automatically result in a new container being built, tested, and deployed to staging and eventually, perhaps, to production
- Setup automated rollouts, roll backs and testing
- Argo is a set of Kubernetes-native tools for deploying and running jobs, applications, workflows, and events using GitOps paradigms such as continuous and progressive delivery and MLops

3. ORCHESTRATION & APPLICATION DEFINITION

- Kubernetes is the market-leading orchestration solution
- You should select a Certified Kubernetes Distribution, Hosted Platform, or Installer: cncf.io/ck
- Helm Charts help you define, install, and upgrade even the most complex Kubernetes application

4. OBSERVABILITY & ANALYSIS

- Pick solutions for monitoring, logging and tracing
- Consider CNCF projects Prometheus for monitoring, Fluentd for logging and Jaeger for Tracing
- For tracing, look for an OpenTracing-compatible implementation like Jaeger

5. SERVICE PROXY, DISCOVERY, & MESH

- CoreDNS is a fast and flexible tool that is useful for service discovery
- Envoy and Linkerd each enable service mesh architectures
- They offer health checking, routing, and load balancing

6. NETWORKING, POLICY, & SECURITY

To enable more flexible networking, use a CNI-compliant network project like Calico, Flannel, or Weave Net. Open Policy Agent (OPA) is a general-purpose policy engine with uses ranging from authorization and admission control to data filtering. Falco is an anomaly detection engine for cloud native.

7. DISTRIBUTED DATABASE & STORAGE

When you need more resiliency and scalability than you can get from a single database, Vitess is a good option for running MySQL at scale through sharding. Rook is a storage orchestrator that integrates a diverse set of storage solutions into Kubernetes. Serving as the "brain" of Kubernetes, etcd provides a reliable way to store data across a cluster of machines. TiKV is a high performant distributed transactional key-value store written in Rust.

8. STREAMING & MESSAGING

When you need higher performance than JSON-REST, consider using gRPC or NATS. gRPC is a universal RPC framework. NATS is a multi-modal messaging system that includes request/reply, pub/sub and load balanced queues. CloudEvents is a specification for describing event data in common ways.

9. CONTAINER REGISTRY & RUNTIME

Harbor is a registry that stores, signs, and scans content. You can use alternative container runtimes. The most common, both of which are OCI-compliant, are containerd and CRI-O.

10. SOFTWARE DISTRIBUTION

If you need to do secure software distribution, evaluate Notary, an implementation of The Update Framework.

Figure 3-4. Cloud Native Trail Map (source: CNCF)

In parallel with this cloud native journey, you can explore the cloud native maturity levels, which is a CNCF framework for adopter companies to understand how to move on, along five different maturity levels: Build, operate, scale, improve, and optimize. You can combine this and the cloud native trail map stages to understand how companies go ahead with their cloud native adoption.

The CNCF community is huge, so are their areas of interest. From kitchen cookbooks, to kids-style educational books for cloud native topics (don't get it wrong, they are highly visual and totally recommended for KCNA candidates!). Feel free to explore this and other assets, as they will contribute to your general culture and connections with other cloud native practitioners.

Mentorship and Ambassadors

Besides the educational resources, there are additional opportunities that can support your cloud native journey, after or during your KCNA exam preparation. Concretely, CNCF facilitates mentorship opportunities for people to develop new skills by helping in different CNCF projects, with remunerated roles. It is a good space for mentors and mentees to connect and discuss. There are even some public testimonials from previous mentees that explain their experience and the value of this program for the cloud native career.

Also, you may take a look at the CNCF Ambassadors program, which enables advocates for the cloud native ecosystem to support and represent the cloud native community with notable contributions such as event organization, content creation, etc.

CNCF projects and Maturity Levels

The CNCF hosts a variety of open source projects for cloud native technologies. Hosting a project at the CNCF means that the project becomes part of the CNCF's set of projects with a neutral home to grow and thrive. This is particularly appealing for projects looking to get credibility and adoption, garner wide-scale industry support, but also attracting a diverse set of contributors.

The projects also get access to critical resources, including cloud credits, funding for specific needs, infrastructure for CI/CD, and other tools necessary for the project's growth. Basically, everything required at technical, marketing, legal, training, and business development level.

Also, these projects follow the official levels of maturity of the CNCF. This means, they are put into different categories depending on how mature they are. That level of maturity is based on metrics such as community adoption, developer participation, etc. As we can see in Figure 3-5, the CNCF has four categories for maturity: incubating, graduated, sandbox, and retired. Concretely:

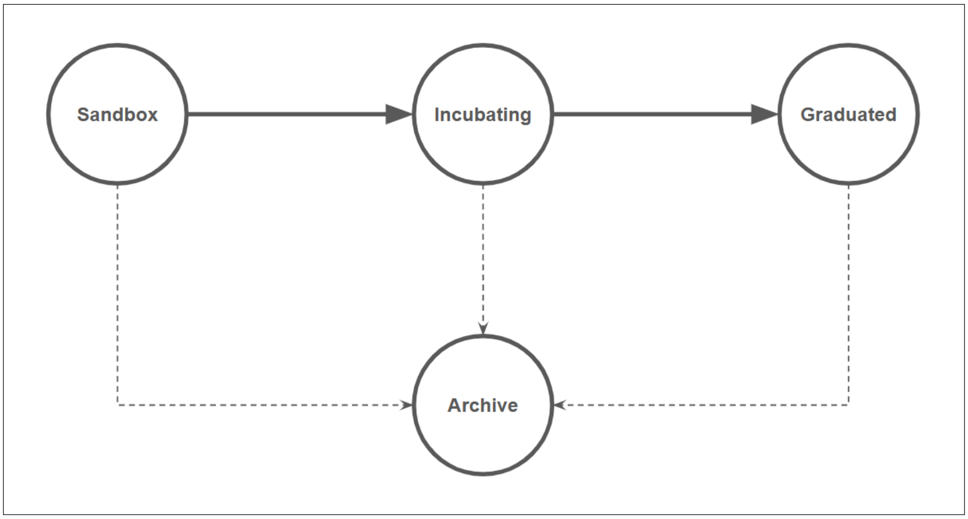


Figure 3-5. CNCF Project Maturity Levels

Sandbox projects
: Entry level for new or experimental projects, offering neutral hosting and access to the broader CNCF community for growth and exposure. The key criteria to enter the sandbox stage are adoption of the CNCF Code of Conduct, TOC sponsorship, and an open-source license. This stage is their way to start.

Incubating projects
: Promising projects, with substantial level of community contributions, an active user base, and integration within the CNCF ecosystem. They receive mentorship and increased visibility, but they require TOC sponsorship and passing a CNCF security audit.

Graduated projects
: Mature projects with wide adoption, clear governance, and regular releases. They have been fully tested and have shown that they are stable and widely used. These projects obtain premier visibility within the CNCF ecosystem, and prioritized access to CNCF resources and support.

Archived projects
: Archived projects aren't being maintained or supported by the CNCF anymore, including the interruption of marketing and other privileges that are only available for active projects. Some examples of archived projects? Brigade, Open Service Mesh, OpenTracing, and rkt.

Related to the CNCF projects, there are two levels of involvement:

Contributors

Individuals who contribute to the project in any form, including sporadic contributions. They can range from code, documentation, design, and more. You can explore contributing opportunities within the community.

Maintainers

People with a more elevated and sustained role in the project. They are typically trusted members of the community who have shown commitment and expertise. There are official resources dedicated to guide maintainers in their specific project goals.

For your KCNA preparation, take a look at these additional topics: The detailed graduation criteria, with specific details of the CNCF requirements for each stage of maturity, and the project proposal process, which will extend your knowledge of how the CNCF processes work.

Main CNCF Projects for the KCNA Exam

For your KCNA exam, you mostly need to know the graduated projects. They are not a lot, and you only need to know their main scope of activity, as the KCNA focuses mostly on project awareness. We will include some official descriptions from CNCF's website, as they are using the most accurate wording for your KCNA exam, and direct links to the annual reports from some of the most important projects:

Argo

Argo is a set of Kubernetes-native tools for deploying and running jobs and applications on Kubernetes. You can check some project insights from their 2022 project report, mostly related to its growth, community contribution and adoption, events, and training-related topics. It will give you an idea of the Argo project, which is one of the most important CNCF projects out there. Also, there are several Argo-related CNCF Slack channels for the different subprojects.

Subprojects:

Argo Workflows

A Kubernetes-native workflow engine for orchestrating parallel jobs on Kubernetes.

Argo CD

A GitOps continuous delivery tool for Kubernetes. It leverages Git repositories as a source of truth for Kubernetes resources and applications.

Argo Rollouts

An advanced deployment controller for Kubernetes which facilitates iterative deployments.

Argo Events
An event-driven solution for Kubernetes, allowing for triggering Argo Workflows or K8s native resources based on events.

- CNCF Category: Continuous Integration & Delivery
- Website: https://argoproj.github.io
- CNCF Page: https://www.cncf.io/projects/argo
- Landscape Card: https://landscape.cncf.io/selected=argo

Containerd
Defined as an open and reliable container runtime, it provides the basic functionalities required for running containerized applications, with focus on simplicity, robustness, and portability. It was originally developed as a part of Docker, but they announced it in 2016 as a standalone project, then moved the project to CNCF one year after. You can get more insights from their 2020 project report. You can also visit the #containerd channel in CNCF's Slack.

- CNCF Category: Container Runtime
- Website: https://containerd.io
- CNCF Page: https://www.cncf.io/projects/containerd
- Landscape Card: https://landscape.cncf.io/?selected=containerd

CoreDNS
A DNS server that provides everything required for service discovery (i.e. automatically detecting network locations) in most networked environments, including distributed systems like Kubernetes.

- CNCF Category: Coordination & Service Discovery
- Website: https://coredns.io
- CNCF Page: https://www.cncf.io/projects/coredns
- Landscape Card: https://landscape.cncf.io/?selected=core-dns

CRI-O
A lightweight container runtime that acts as a bridge between Kubernetes and container images, allowing Kubernetes to use any OCI (the Open Container Initiative, we will explain this initiative and others by the end of this chapter) compliant runtime as the container runtime for running its pods.

- CNCF Category: Container Runtime
- Website: https://cri-o.io
- CNCF Page: https://www.cncf.io/projects/cri-o
- Landscape Card: https://landscape.cncf.io/?selected=cri-o

Envoy

Its official description is "a cloud-native high-performance edge/middle/service proxy", which means that it is a high-performance proxy designed to manage and optimize network traffic in cloud-native applications. Envoy was originally developed at Lyft and then donated to the CNCF. It was built to address the specific challenges of microservices architectures, especially in high-scale, production environments. Their 2019 project report is a good way to understand its relevance within the community. Check the #enjoy Slack channel for some interesting live discussions.

- CNCF Category: Service Proxy
- Website: https://www.envoyproxy.io
- CNCF Page: https://www.cncf.io/projects/envoy
- Landscape Card: https://landscape.cncf.io/?selected=envoy

Etcd

A very relevant distributed key-value store (i.e. a database with some specific kind of format) for the most critical data of a distributed system. Same as other CNCF projects, you can visit their 2021 project report to analyze their community insights and level of adoption, and the #etcd Slack channel.

- CNCF Category: Coordination & Service Discovery
- Website: https://etcd.io
- CNCF Page: https://www.cncf.io/projects/etcd
- Landscape Card: https://landscape.cncf.io/?selected=etcd

Fluentd

Another important CNCF project. Fluentd provides a unified logging layer via an open source data collector between data sources and backend systems. You can visit their most recent project report from 2020.

- CNCF Category: Logging
- Website: https://www.fluentd.org
- CNCF Page: https://www.cncf.io/projects/fluentd
- Landscape Card: https://landscape.cncf.io/?selected=fluentd

Flux

Same as Argo, Flux is part of the CI/CD ecosystem of tools. It automates the deployment, scaling, and management of containerized applications using GitOps principles, where a Git repository serves as the single source of truth for declarative infrastructure (i.e. ensuring that the state of a K8s cluster matches the configuration stored in a Git repository).

- CNCF Category: Continuous Integration & Delivery
- Website: https://fluxcd.io
- CNCF Page: https://www.cncf.io/projects/flux
- Landscape Card: https://landscape.cncf.io/?selected=flux

Harbor

Harbor is a container registry that secures artifacts with policies and role-based access control, ensures images are scanned and free from vulnerabilities, and signs images as trusted.

- CNCF Category: Container Registry
- Website: https://goharbor.io
- CNCF Page: https://www.cncf.io/projects/harbor
- Landscape Card: https://landscape.cncf.io/?selected=harbor

Helm

Package manager for Kubernetes that makes it easier to define, install, and manage Kubernetes applications with the use of pre-configured Kubernetes resources (in this context, those resources are called charts). Take a look at their 2020 project report, and Slack channels such as #helm-dev and #helm.

- CNCF Category: Application Definition & Image Build
- Website: https://helm.sh
- CNCF Page: https://www.cncf.io/projects/helm
- Landscape Card: https://landscape.cncf.io/?selected=helm

Istio

A service mesh project that adds a layer of infrastructure between services and the network that allows them to communicate securely and reliably. Additionally, Istio Wasm uses WebAssembly modules (a web standard for binary instruction format) to extend the behavior of the Istio service mesh.

- CNCF Category: Service Mesh
- Website: https://istio.io
- CNCF Page
  - https://www.cncf.io/projects/istio
  - https://www.cncf.io/projects/istio-wasm
- Landscape Card
  - https://landscape.cncf.io/?selected=istio
  - https://landscape.cncf.io/wasm?selected=istio-wasm

Jaeger

Initially developed by Uber, Jaeger helps developers monitor and troubleshoot transactions in complex distributed systems by providing a visual representation of the flow of requests through services and latency bottlenecks. Their latest 2020 project report will complement your knowledge of this relevant project.

- CNCF Category: Tracing
- Website: https://www.jaegertracing.io
- CNCF Page: https://www.cncf.io/projects/jaeger
- Landscape Card: https://landscape.cncf.io/?selected=jaeger

Keda

From its official description, KEDA is a Kubernetes-based event driven auto scaling component. The goal of this project is to bridge the gap between K8s and event-driven architectures, ensuring that applications can scale based on external metrics or events.

- CNCF Category: Installable Platform
- Website: https://keda.sh
- CNCF Page: https://www.cncf.io/projects/keda
- Landscape Card: https://landscape.cncf.io/serverless?selected=keda

Kubernetes

Previous and next chapters of this KCNA Study Guide talk extensively about the K8s project, but we will keep the main URLs here for the sake of completeness of this list. Same as other projects, you can find their 2023 project report, including a second version in Japanese. Both CNCF and Kubernetes.io Slack workspaces contain plenty of K8s-related discussion channels.

- CNCF Category: Scheduling & Orchestration
- Website: https://kubernetes.io
- CNCF Page: https://www.cncf.io/projects/kubernetes
- Landscape Card: https://landscape.cncf.io/?selected=kubernetes

Linkerd

One of the pioneering service mesh projects designed to provide observability, reliability, and security for microservices-based applications, without requiring changes to the application code.

- CNCF Category: Service Mesh
- Website: https://linkerd.io
- CNCF Page: https://www.cncf.io/projects/linkerd

- Landscape Card: https://landscape.cncf.io/?selected=linkerd

Open Policy Agent (OPA)
It is a policy-based engine to control cloud native environments that allows you to enforce policies across a wide variety of software, including K8s, microservices APIs, CI/CD pipelines, etc.

- CNCF Category: Security & Compliance
- Website: https://www.openpolicyagent.org
- CNCF Page: https://www.cncf.io/projects/open-policy-agent-opa
- Landscape Card: https://landscape.cncf.io/?selected=open-policy-agent-opa

Prometheus
A very relevant CNCF project, with its own certification options (as we have seen with the PCA exam before). It provides a monitoring and alerting toolkit, with great success due to its robustness and community support. The 2019 project report is a good way to understand the adoption trends, making it one of the most important graduated projects.

- CNCF Category: Monitoring
- Website: https://prometheus.io
- CNCF Page: https://www.cncf.io/projects/prometheus
- Landscape Card: https://landscape.cncf.io/?selected=prometheus

Rook
Rook is a cloud native storage for Kubernetes, for production level management of file, block and object storages.

- CNCF Category: Cloud Native Storage
- Website: https://rook.io
- CNCF Page: https://www.cncf.io/projects/rook
- Landscape Card: https://landscape.cncf.io/?selected=rook

SPIFFE
Also known as Secure Production Identity Framework For Everyone, it provides a standardized way to assign and validate identity to software systems in a diverse environment.

- CNCF Category: Key Management
- Website: https://spiffe.io
- CNCF Page: https://www.cncf.io/projects/spiffe
- Landscape Card: https://landscape.cncf.io/?selected=spiffe

SPIRE

Known as the SPIFFE Runtime Environment, SPIRE is an implementation of the SPIFFE standards. It's a runtime environment that automatically validates, issues, and renews identity credentials for workloads in a given system.

- CNCF Category: Key Management
- Website: https://spiffe.io/spire
- CNCF Page: https://www.cncf.io/projects/spire
- Landscape Card: https://landscape.cncf.io/?selected=spire

The Update Framework (TUF)

A framework to secure software update systems. It provides protection even against attackers that compromise the repository or signing keys.

- CNCF Category: Security & Compliance
- Website: https://theupdateframework.github.io
- CNCF Page: https://www.cncf.io/projects/the-update-framework-tuf
- Landscape Card: https://landscape.cncf.io/?selected=the-update-framework-tuf

TiKV

Defined as a distributed transactional key-value database, based on the design of Google Spanner and HBase, but simpler to manage and without dependencies on any distributed file system.

- CNCF Category: Database
- Website: https://tikv.org
- CNCF Page: https://www.cncf.io/projects/tikv
- Landscape Card: https://landscape.cncf.io/?selected=ti-kv

Vitess

Last but certainly not least, Vitess is a MySQL compatible, horizontally scalable, cloud-native database solution. The 2020 project report contains more useful insights about its relevance within the cloud native ecosystem.

- CNCF Category: Database
- Website: https://vitess.io
- CNCF Page: https://www.cncf.io/projects/vitess
- Landscape Card: https://landscape.cncf.io/?selected=vitess

Obviously, the maturity level is not the only sign of relevance for the CNCF projects. There are other incubating projects (at the moment we write this Study Guide, but they may evolve and move to graduation) that you may want to explore, such as Thanos, Cilium, CNI, gRPC, OpenTelemetry, or Kyverno.

Summarizing, this section contains the main information that you need to know, but we encourage you to visit all the URLs, explore the project reports, and learn as much as you can about these projects. This will serve your KCNA preparation but also to general cloud native culture.

Related initiatives and work groups

Besides the CNCF and their projects, there are other international initiatives and projects that are very important for the cloud native ecosystem. They are part of the scope of your upcoming KCNA exam, and they serve as reference for any existing or new CNCF project. Concretely:

Open Container Initiative (OCI)
: A project under the Linux Foundation's umbrella. Its primary goal is to design open standards for container formats and runtimes in the cloud-native ecosystem, so containers are both consistent and interoperable, allowing for smooth portability between different platforms and environments.

Container Network Interface (CNI)
: It's a specification and a set of tools for configuring container networking, and it's especially relevant in the cloud-native ecosystem where containers play a pivotal role.

Container Runtime Interface (CRI)
: It is a plugin interface defined by Kubernetes that allows the kubelet (a component in the Kubernetes node we will see in Chapters 5 and 6) to use various container runtimes, without the need to recompile, and without being explicitly tied to any specific one of them.

Container Storage Interface (CSI)
: This interface is a standard one between container orchestrators like Kubernetes and the storage systems, that reduce the manual efforts to add diverse storage systems to the container orchestration engines.

Service Mesh Interface (SMI)
: It is a specification for service meshes on K8s. It provides a standard interface for service meshes on Kubernetes and defines a set of common, portable APIs. By doing so, SMI aims to provide interoperability between different service mesh technologies, such as Istio and Linkerd.

This section concludes the main content of Chapter 3, in which we have explored the CNCF, the origins of the cloud native ecosystem, the most relevant projects, and some important standards. All these knowledge blocks are part of the KCNA exam scope, so pay attention to these topics as they are relatively simple. Once again, leverage the education resources, including the CNCF Landscape Guide with additional 101 explanations, and references to existing projects.

And talking about the CNCF Landscape, let's get some expert insights from one of the key executives of the CNCF, who is also a great contributor to the landscape project. Concretely, our Chapter 3 expert insights will bring Mr. Chris Aniszczyk, CTO of the CNCF.

CHAPTER 4

Essential Concepts for Cloud Native Practitioners

A Note for Early Release Readers

With Early Release ebooks, you get books in their earliest form—the author's raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 4th chapter of the final book. Please note that the GitHub repo will be made active later on.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at *shunter@oreilly.com*.

In this chapter, we will explore the essential concepts related to Cloud Computing. Having a solid understanding of the basics allows us to build upon them for advanced implementation. For this purpose, and even if we know that every exam candidate may have different levels of preexisting knowledge, we will cover the end-to-end of cloud-related topics, from the basics to other advanced notions that you will leverage later in Chapters 5 and 6 for Kubernetes.

Concretely, we will cover the introduction to general cloud computing, its evolution, infrastructure, commercialization, and the road to cloud native. The goal is to provide a holistic picture of how cloud computing evolves into what we know today. The material in this chapter includes explanations of topics covered in the KCNA exam.

General Cloud Computing

Let’s start from the beginning. What is cloud computing? And why do we mean “clouds”?

The cloud refers to servers that are remotely accessible over the internet. Softwares and databases are run on these servers. Imagine buildings (often referred to as data centers) full of physical servers. A company that owns these data centers rents out the ability to use the computing powers of these servers to other users or companies. These users then do not have to manage physical servers themselves. Running their applications on these clouds, a computing model that allows remote allocation of computing resources owned by a third party for external users is what is generally referred to as ***Cloud Computing***.

More officially, the CNCF defines cloud computing as “a model that offers compute resources like CPU, network, and disk capabilities on-demand over the internet. Cloud computing gives users the ability to access and use computing power in a remote physical location. Cloud Service Providers (CSP) like AWS, GCP, Azure, DigitalOcean, and others all offer third parties the ability to rent access to compute resources in multiple geographic locations”.

Before the advent of cloud computing, organizations typically had to rely on building and managing their own physical infrastructure to support their computing needs. This is typically called ***on-premises or bare metal infrastructure***. It involved procuring servers, storage devices, networking equipment, and other hardware components, as well as establishing data centers or server rooms to house and maintain these resources, which are costly and often cumbersome to manage.

Also, you may have heard the term ***monolithic application***. The term refers to a traditional software that is a single piece of complicated software that continues to increase in size and complexity as new features and abilities are added onto it. In order for the monolith to function, it must reside and run on a single system that supports it. This hardware system of specific capacity such as memory, networking capabilities, storage, and computational ability is complex to build, maintain and expand. Scaling up or down a part of this operation is difficult as it requires another server with a load balancer. To do this, organizations have to assess the needs, weigh the benefits against the cost, go through a procurement process and once obtained, set it up. This process can take months or years, making it difficult for the organizations to respond quickly to the demand. This is not to mention the work that the engineers have to do in order to keep all server instances up to date with upgrades and security or operational patches. The disruptions in service are hard to avoid.

If a monolith is a hard to move mammoth, then ***microservices*** is like a colony of bees. Bees come together to carry out many well-organized functions. Their shapes and sizes are easy to adapt. They can adjust and move with agility. That is why organizations move away from monolithic applications and start building microservices instead.

As you can see in Figure 4-1, microservices refer to small independent processes coming together to form a complex application. These microservices communicate with each other through **Application Programming Interfaces (APIs)** over a network. You may have heard of external APIs. API specifications dictate the way that each component interacts with each other regardless of them being in the same application.

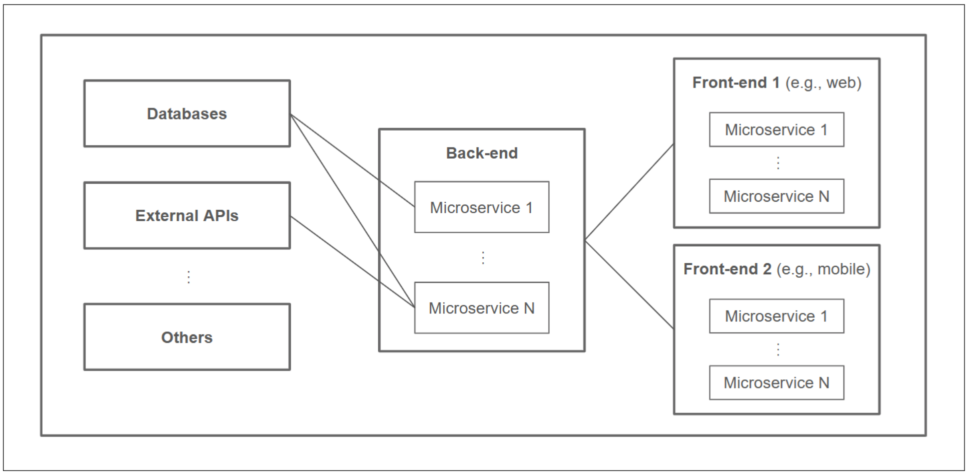


Figure 4-1. Microservices-enabled Development

Now, think of the fact that microservices are small and independent. They can communicate with each other easily. That means there is no need for them to reside together on a single hardware. The hardware can similarly be smaller and separated. Each microservice can find the hardware that best suits their characteristics and needs. This is where cloud computing comes in.

As connectivity, storage, and hardware technologies advanced, cloud computing became an answer to the challenges associated with traditional computing and monoliths. With Amazon's launch of Amazon Web Services in 2006, which enabled organizations to rent server space remotely, cloud computing gained mainstream adoption and revolutionized the way organizations approach their computing needs. With cloud computing, organizations can leverage the resources and services of cloud service providers, accessing computing power, storage, and other resources over the internet. They can scale resources up or down on-demand, pay for what they use, and benefit from the provider's expertise in managing and maintaining the underlying infrastructure. More importantly, it enabled organizations to focus more on their core business activities and software development rather than worrying about hardware maintenance, infrastructure planning, and scalability constraints.

This is what microservices and cloud computing are a match made in heaven. Together, with containers (see details in chapter 5 and 6), they are a trifecta of modern cloud application architecture. ***Containers*** fill in the missing piece in making microservices run easily anywhere. It can be considered as an add-on package that isolates the microservice from dependencies and operating system. Containers make microservices self-sufficient to be run in any environment, any cloud, and any desktop. Technologies like Docker or Podman are used to create and run containers.

To manage, deploy and scale containerized code in the cloud and take full advantage of it, we need a system. ***Container orchestration*** refers to the ability to automatically provision, deploy and scale containerized applications on the cloud. Systems like Kubernetes and Docker Swarm do just that. They allow the users to manage the containers across the clouds.

This application architecture is crucial in the popularization of cloud computing in the modern days. The commercialization of the cloud depends on its connectivity and user's ability to access it from anywhere. In the next section, you will explore the different ways that the cloud services are structured and made available in the market.

Commercialization of Cloud Computing

Cloud Service Providers (CSPs) offer cloud technology with differing degrees of administration, maintenance, and self-service functionalities, so their clients do not need to buy physical machines or develop their own services. Not dissimilar to a rental agreement, CSPs allow their clients to pay to use their infrastructure and services without having to build, own and maintain them. A consumption-based pricing strategy is typically employed where customers are charged based on the resources they utilize and the duration of usage. This flexible pricing structure eliminates the need for upfront capital investments and allows organizations to align their costs with actual resource consumption, optimizing their IT budgets. Public and private organizations use this type of service as a financial choice of ***operational expenditure (OPEX)***.

Now, the commercialization of cloud computing technologies offers ***different levels of service***, as we can see in Figure 4-2. Basically, tradeoffs between financial cost and technical control, giving the adopters the option to choose the model that suits them better. The "as a Service / aaS" model has different levels of implementation as listed below.

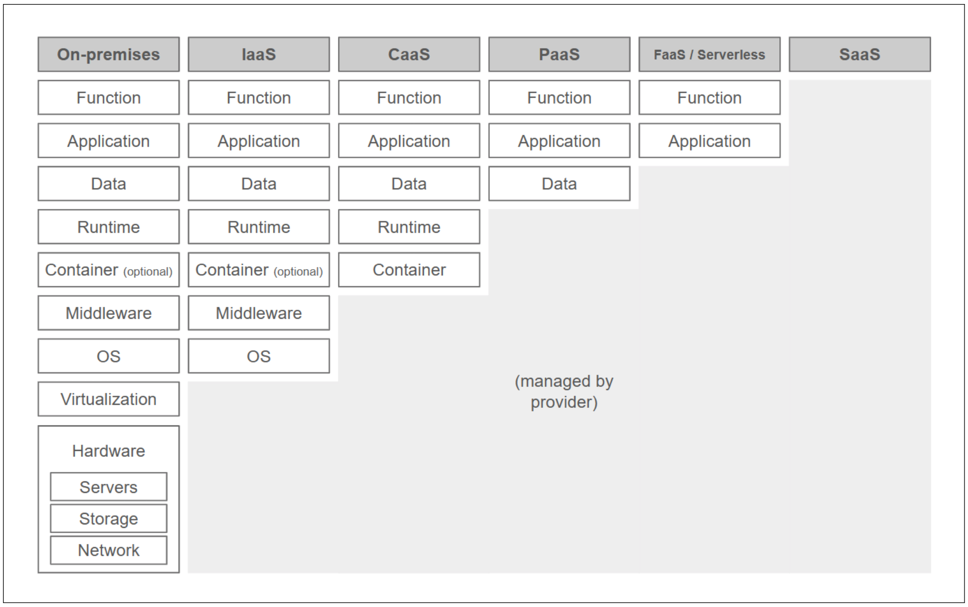


Figure 4-2. Managed Cloud As-a-Service Levels

IaaS, Infrastructure-as-a-Service

CSPs provide users with computing, networking, and storage resources to create and run their applications. Users have a high level of control but also higher maintenance required. In this case, the users rent barebone server space and storage from the providers. A popular analogy for IaaS is that it is like renting a plot of land. The possibilities are endless. You can build anything you want on it. However, know that you will bear the cost of building material and equipment.

One of the reasons why IaaS is appealing to the users is to reduce capital expenditure and transform them into operational expenses. Instead of procuring the hardware and building their own cooling server room, they simply engage the CSP to provision and manage the infrastructure which can be accessed over the internet.

Other benefits of IaaS include scalability efficiency and boosting productivity. In the scenario where the business need is unpredictable, IaaS allows the users to respond to the need by quickly increasing or decreasing the resources as needed. The IT professionals in the organization do not have to worry about maintaining a data center with high availability. The CSPs are in charge of ensuring that the service is available to the users at all times, even when there is a problem with an equipment. This allows the organization to concentrate on the strategic business activities.

Examples of IaaS are Amazon Web Services (AWS) EC2, Microsoft Azure Virtual Machines, and Google Compute Engine.

PaaS, Platform-as-a-Service

In the case of Platform-as-a-Service, cloud providers equip their users with a platform for applications to run, as well as all the IT infrastructure required. In other words, the users rent the equipment and tools to build their own building on the infrastructure. PaaS provides a platform for the developers to develop, run and manage their own application without having to worry about the infrastructure. Infrastructure management, security patches, updates and other administrative tasks are taken care of by the provider. When using PaaS, organizations can forget about setting up and maintaining application servers and the various environments. They pay only for the resources that they use which makes it attractive in the scenario where they want to quickly build, test and deploy applications.

PaaS requires less maintenance when compared to IaaS but the users also have less level of control. Some examples of PaaS providers include Google App Engine, Microsoft Azure App Service, and Heroku.

SaaS, Software-as-a-Service

The CSPs offer applications plus the required platform and IT infrastructure over the internet. The cloud providers build, maintain and offer the applications for the users on demand. Generally, these ready-to-use products are available for a subscription fee. In this model, the users do not have to build the building themselves. The building is available for them to rent, all they have to do is move in and pay rent. The renter can use it as if they own the building most of the time. However, knowing that while the landlords will fix things when they are broken, not all functions can be customized according to their needs.

SaaS makes it very easy for the organizations to implement new technology with very little effort to train their staff. There is no need for an IT staff member to even manage the software as it is all taken care of by the provider.

This model is simple to use with limited maintenance required for the users. Popular SaaS that you may be familiar with include Salesforce, Google Workspace, Slack and Dropbox.

CaaS, Container-as-a-Service

While you may be more familiar with IaaS, PaaS and SaaS, CaaS is gaining popularity among IT professionals, particularly microservices developers. CaaS refers to the automated hosting and deployment of containerized software packages. It is usually available for a subscription fee, hence, the as-a-service concept. Designed based on industry standards such as Open Container Initiative (OCI) and Container Runtime Interface (CRI), CaaS makes it easy to move from one to another cloud provider.

Without CaaS, software development and operations teams need to deploy, manage, and monitor the underlying infrastructure that containers run on. This infrastructure is a collection of cloud machines and network routing systems that requires dedicated DevOps resources to oversee and manage. There is no need to install, operate, or maintain the container orchestration control plane. In other words, CaaS solutions remove the complication of actually orchestrating the container orchestration tool.

Some examples of CaaS include Amazon Elastic Container Service (ECS) and Amazon Elastic Kubernetes Service (EKS). ECS is a container orchestration and management service provided by AWS. It offers a choice of orchestrators including its own native orchestration tool. Similar to ECS, EKS is a fully managed Kubernetes Service. Other popular CaaS tools are Azure Container Instances (ACI) and Google Cloud Run.

FaaS, Function-as-a-Service

CSPs enable users to create packages of code as functions without maintaining any infrastructure. This is a very granular cloud usage option which makes it especially interesting for development activities. To use the same analogy, FaaS allows the renter to pay only for the room in the building that's being used at the time.

FaaS model allows developers to write and deploy code in the form of functions that are executed in response to specific events or triggers. Since the code is deployed without provisioning or managing servers or backend infrastructure, it is often referred to as serverless cloud computing. FaaS is suitable for users who wish to run specific functions of the application without managing the servers. The users only have to provide the code and pay per duration or number of executions.

Some CaaS tools mentioned in the previous section are integrated within an environment that is serverless, so some CaaS solutions have the characteristics of FaaS. The key here for FaaS is that the underlying infrastructure and platform is all taken care of by the providers. Some examples of FaaS are AWS Lambda, Google Cloud Functions.

Cloud Computing Infrastructure

There are different types of cloud computing. General cloud computing often refers to the "public" cloud, which is a global infrastructure from big providers, physically shared for and by different clients, but logically isolated with their own work environments. However, here is a comprehensive list of all the options:

Public Cloud

The physical cloud resources (servers) are organized and distributed by the cloud provider. This means that two or more companies can use the same physical infrastructure. As opposed to public parks or libraries, public clouds actually have a private owner. The owner provides access and usage of the cloud to their clients who pay them a fee. Examples of public clouds include AWS, Google Cloud Platform (GCP) or Azure from Microsoft.

Private Cloud

The whole "cloud" is for one single organization. It is commonly used in environments where a very high level of control is required. Private clouds allow the users to leverage all the advantages that come with cloud computing but with the access control, security, and resource customization of an on premise infrastructure. Some organizations may opt for private cloud if they deal with sensitive and confidential information or strict regulations that mandates greater visibility and control on access, software/hardware choices and governance. What's interesting here is that because private clouds (often dubbed "on-prem" solutions) are built with the same fundamentals as public clouds, it allows the organization to easily migrate to or share its workload with public clouds, leading to something called "Hybrid Cloud" solution.

Hybrid Cloud

Hybrid cloud refers to when an organization decides to utilize both public and private clouds for their cloud computing, storage, or services needs. Sharing the load between private and public cloud options allow the organization to create a versatile setup based on your needs. This approach is one of the most common setups today because of its flexible options to optimize the usage and costs including risk sharing.

Multi Cloud

A type of Hybrid cloud setup. Some organizations may choose to use cloud technologies from multiple providers. This can make sense in terms of cost optimization (when provider A is cheaper than provider B for some specific services), or to leverage the combined offer of services (which may be different depending on the combination of providers)

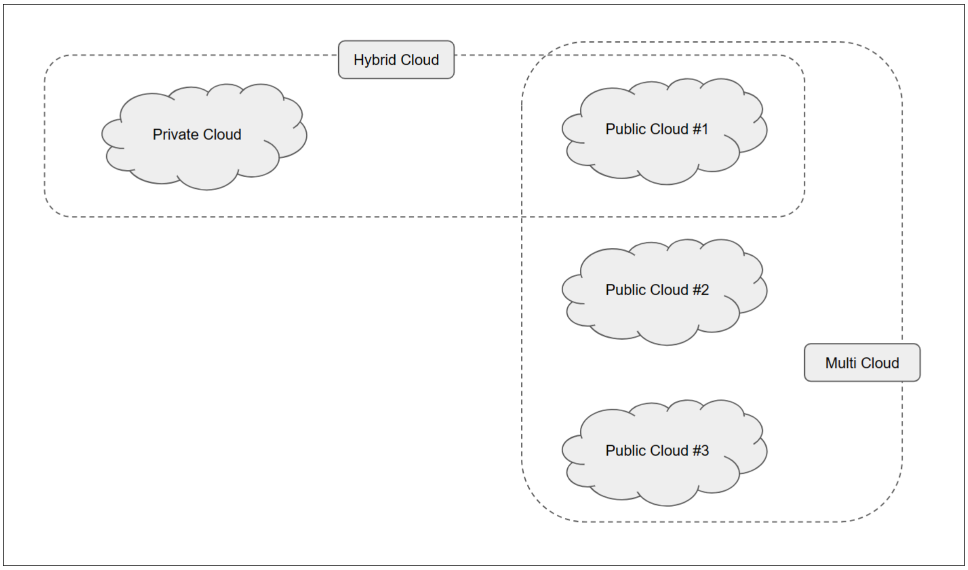


Figure 4-3. Types of Cloud Infrastructure

Organizations choose their "-aaS" models and type of clouds based on their own business needs and preferences. After they choose their cloud, they have to decide how to deploy their systems, services, and applications. That's where the concepts of virtualization, containerization, and orchestration come.

The Virtualization, Containerization, and Orchestration Trifecta

Organizations then have to ask themselves how to deploy systems, services or applications to make them available to everyone within the organization, embracing the benefits of cloud-enabled systems, such as availability or scalability. Likely, they will utilize already deployed services and deploy others based on techniques such as virtualization, containerization, and others.

Virtualization

Virtualization is a technology that enables the creation of virtual versions or representations of physical resources, such as servers, operating systems, storage devices, or networks. It allows multiple virtual instances to run simultaneously on a single physical machine, each isolated and behaving as if it were running on its dedicated hardware.

In traditional computing, each physical server or computer typically runs a single operating system and hosts specific applications. However, virtualization enables the abstraction and sharing of physical resources among multiple virtual machines (VMs) or virtual environments.

Containerization

Containerization is a technique in software development and deployment that involves encapsulating an application and its dependencies into a self-contained unit called a container. A container provides a lightweight and isolated runtime environment that allows applications to run consistently across different computing environments, such as development machines, testing environments, and production servers.

Containers are created from container images, which are pre-built packages that include the application code, libraries, runtime environment, and any other dependencies needed to run the application. These images are typically created using containerization platforms like Docker.

The key components and concepts in containerization are:

Container Image –
: A container image is a static, read-only file that serves as a blueprint for creating containers. It contains the application code, along with the necessary runtime dependencies, libraries, and configuration files. Container images are portable and can be shared and distributed across different systems.

Container Runtime –
: The container runtime is responsible for running and managing containers. It provides an isolated and secure execution environment for containers, ensuring that they have their own filesystem, processes, and network stack while sharing the host machine's operating system kernel.

Containerization Platforms –
: Containerization platforms, such as Docker and container orchestration systems like Kubernetes, provide tools and services for building, running, and managing containers at scale. They offer features like container orchestration, networking, storage, and monitoring to simplify the deployment and management of containers.

Container Orchestrators –
: Container orchestrators, such as Kubernetes, manage the deployment, scaling, and monitoring of containers across multiple hosts or nodes. They automate the scheduling of containers, ensure high availability, and provide advanced features like load balancing and service discovery.

Benefits of containerization include:

Portability –
: Containers can run consistently across different environments, from development to production, enabling easier deployment and migration of applications.

Efficiency –
: Containers are lightweight and have fast startup times, allowing for efficient resource utilization and rapid scaling.

Isolation –
: Containers provide isolation between applications and the host system, enhancing security and preventing conflicts between dependencies.

Reproducibility –
: Containers encapsulate all dependencies and configurations, ensuring consistent application behavior and simplifying collaboration between teams.

DevOps Enablement –
: Containerization aligns well with DevOps practices, enabling faster and more streamlined software development, testing, and deployment workflows.

The main difference between virtualization and containerization is that virtualization creates separate virtual machines with complete operating systems, while containerization encapsulates applications and their dependencies within lightweight containers that share the host OS. Virtualization provides stronger isolation, while containerization offers faster startup times, lower overhead, and more efficient resource utilization, making it well-suited for modern application deployment and microservices architectures. As we can see in Figure 4-3, lightweighting is one of the key advantages of containers:

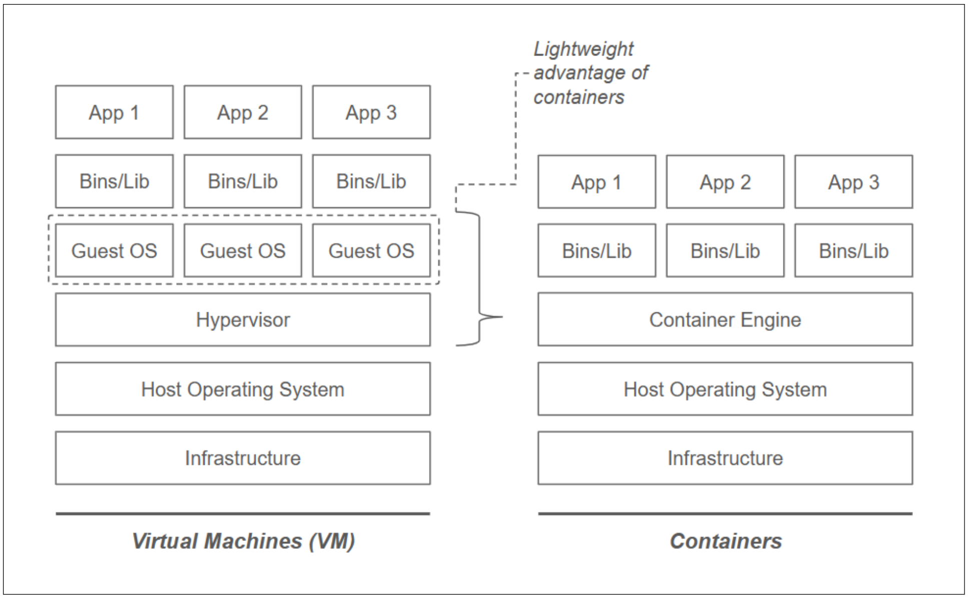


Figure 4-4. Virtualization vs Containerization

Orchestration

Orchestration refers to the automated management, coordination, and execution of complex tasks or processes, like a musical conductor in a symphony. It involves controlling and organizing various components, services, and resources to achieve a desired outcome efficiently and reliably.

Orchestration plays a crucial role in the deployment and operation of distributed applications, especially in cloud computing and microservices architectures. It helps automate and streamline processes that involve multiple interconnected components, such as provisioning and scaling resources, managing dependencies, handling deployments, and ensuring high availability.

Kubernetes is a type of container orchestrator. Basically, it manages service-oriented application components that are tightly scoped, strongly encapsulated, loosely coupled, independently deployable and independently scalable. Each service focuses on a specific business functionality and can be developed, deployed, and scaled independently. In a microservices architecture, an application is decomposed into a set of self-contained services, where each service handles a specific task or business capability.

Relationship between Cloud Computing and Cloud Native

Cloud computing and cloud-native are related concepts but refer to different aspects of modern computing. As we have seen, cloud computing refers to the delivery of computing services over the internet, providing on-demand access to a variety of resources, including servers, storage, databases, software, and networking. It involves leveraging remote servers and infrastructure provided by cloud service providers to store and process data, run applications, and perform computing tasks. Examples of cloud computing service providers include Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform.

Cloud computing offers benefits such as cost efficiency, scalability, flexibility, accessibility, and reliability. It eliminates the need for organizations to build and manage their own physical infrastructure, allowing them to focus on their core business activities while leveraging the resources and services offered by cloud providers.

Cloud native, on the other hand, is an approach to developing and deploying applications that takes full advantage of cloud computing capabilities and leverages cloud-native technologies and practices. It is about designing, building, and running applications specifically for the cloud environment, with a focus on scalability, resilience, and agility. They make use of microservices and containerization technologies, such as Docker, to package applications and their dependencies into lightweight, portable containers that are often managed and orchestrated using tools like Kubernetes. Along with DevOps (development operations, which we will explain by the end of this chapter, and CI/CD), we get the key building blocks of cloud native:

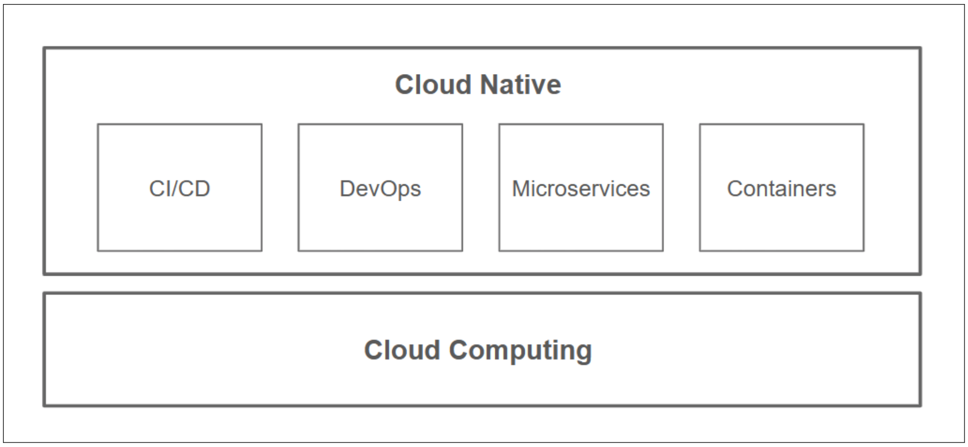


Figure 4-5. Cloud Native Building Blocks

Besides other success stories we have seen before, here are a couple of cloud native examples. Netflix is a prime case of a cloud-native application. It utilizes microservices architecture and runs on Amazon Web Services (AWS) cloud infrastructure.

Each component of Netflix's application, such as user profiles, recommendations, and video streaming, is built as a separate microservice, allowing for scalability, fault tolerance, and continuous deployment. Similarly, Lyft, a ride-sharing platform, is built as a cloud-native application. It uses microservices architecture to handle various functions such as ride matching, real-time tracking, and payments. Lyft employs cloud computing platforms like AWS and Google Cloud Platform (GCP) to ensure scalability, availability, and cost optimization.

In summary, cloud computing is about delivering computing services over the internet, while cloud-native refers to an approach to application development and deployment that leverages cloud computing capabilities and adopts cloud-native technologies and practices. Cloud-native applications are designed and built specifically for the cloud, taking advantage of microservices, containerization, and orchestration to achieve scalability, resilience, and agility.

Building Cloud Native Applications

So, we know cloud computing helps build cloud applications that are efficient to deploy and resilient to maintain. Like anything with new, best practices emerge out of experience. It is through this experience that the Twelve-Factor App manifesto came about.

The Twelve-Factor App Manifesto The manifesto was created by the company Heroku in 2012. It outlines a methodology for building software-as-a-service apps that can be applied to apps written in any programming language, and which use any combination of backing services (database, queue, memory cache). The twelve factors are outlined below:

Codebase –
: A twelve-factor application should have one codebase tracked in version control, but there can be multiple deployments of this codebase. For example, production, staging, and development environments should all come from the same codebase.

Dependencies –
: Declare and isolate dependencies explicitly. Do not rely on the existence of system-wide packages. This ensures consistent environments and avoids the "works on my machine" problem.

Config –
: Store configuration that varies between deployments (e.g., database credentials, API keys) in the environment. This separates the configuration from the application, ensuring that the application is environment agnostic.

Backing Services –
: Treat backing services like databases, caching, messaging systems, and so forth as attached resources. This means that a local MySQL database should be treated the same as a managed Amazon RDS instance, for instance.

Build, Release, Run –
: Strictly separate the build and run stages. The build stage converts code into an executable bundle, the release takes the build and combines it with the environment's config, and the run stage (runtime) runs the app in the execution environment.

Processes –
: Execute the application as one or more stateless processes. Any data that needs to persist should be stored in a stateful backing service, like a database.

Port Binding –
: Export services via port binding. Your application should be self-contained and not rely on a separate web server; it should be able to bind to a port and start accepting requests.

Concurrency –
: Scale out using the process model. This means that performance scaling should be achieved by adding more processes rather than trying to make individual processes bigger and more complex.

Disposability –
: Maximize robustness with fast startup and graceful shutdown. Processes should strive to start quickly and shut down gracefully by releasing their resources and shutting down in response to a SIGTERM signal.

Dev/Prod Parity –
: Keep development, staging, and production environments as similar as possible. This reduces discrepancies and "works on my machine" issues, leading to more predictable deployments.

Logs –
: Treat logs as event streams. Instead of managing log files, apps should produce logs as a continuous stream of events and let the environment handle the storage and archiving.

Admin Processes –
: Run admin/management tasks as one-off processes. This means that tasks like database migrations should be run as separate processes, ideally in an identical environment to the regular long-running processes of the app.

The Twelve-Factor methodology is a comprehensive guide, and while not all applications will strictly adhere to every factor, they provide a solid foundation for building scalable and maintainable SaaS applications. From a KCNA point of view, you can consider these factors as industry standards when it comes to cloud native development.

Other Relevant Cloud Native Topics

There are several terms related to cloud native that explain methodologies, best practices, technical paradigms, etc. Most of them are described by CNCF in their official glossary of terms. Besides everything you have seen up to now, and the next two Kubernetes-first chapters, here are some fundamental terms you need to familiarize with before taking the KCNA exam:

DevOps and CI/CD

According to the CNCF glossary, ***DevOps*** is a methodology in which teams own the entire process from application development to production operations, hence DevOps. It goes beyond implementing a set of technologies and requires a complete shift in culture and processes. DevOps calls for groups of engineers that work on small components (versus an entire feature), decreasing handoffs – a common source of errors.

The reality is that the definition of DevOps is one of the most debated terms in the field. DevOps implementation often entails a practice of moving testing, quality, and performance evaluation early in the development process. This move is sometimes referred to as "left shift", as in shifting these steps to the left closer to or even before code is written.

Even more, there is a visual representation (Figure 4-6) of what an iterative DevOps cycle means, as a series of steps for the software development and release process:

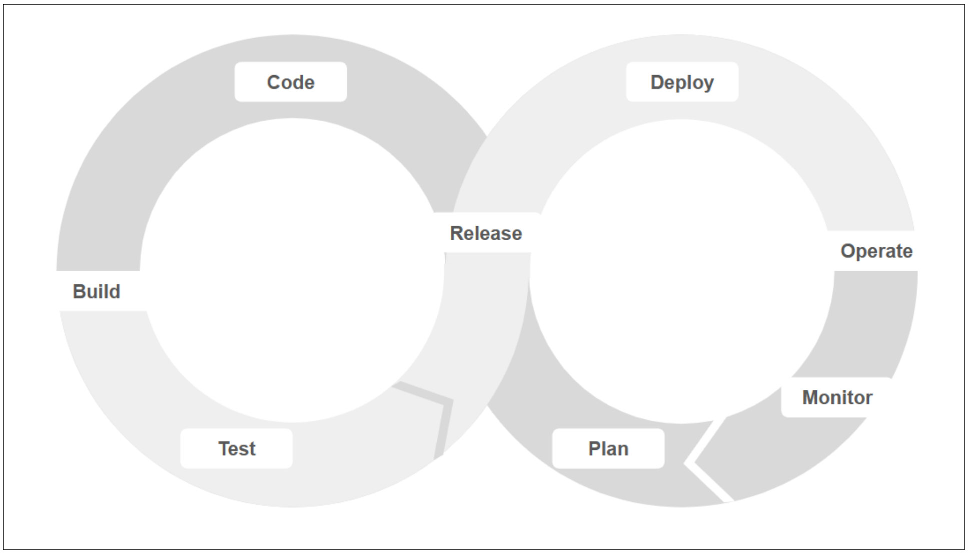


Figure 4-6. DevOps Lifecycle Loop

1. ***Plan*** - This is the initial phase where the team plans out the features, improvements, or fixes that they want to implement. Requirements are gathered, and tasks are prioritized.
2. ***Code*** - Developers write and commit the source code. This could involve feature development, bug fixing, or code refactoring. Tools like Git are commonly used for version control.
3. ***Build*** - The source code is compiled into executable artifacts. Build tools, such as Maven, Gradle, or Jenkins, automate this process, ensuring that the code can be reliably built into software.
4. ***Test*** - Automated tests are run to ensure that the software is reliable, secure, and meets the quality standards. This can encompass unit tests, integration tests, system tests, and more.
5. ***Release*** - Once the software passes tests, it's ready for release. Automated deployment tools can push the software into staging or production environments. The release process might involve Continuous Integration (CI) and Continuous Deployment (CD) pipelines.
6. ***Deploy*** - The software is deployed to production servers. This step is sometimes combined with the release, especially in CD environments where code is automatically deployed once it passes all tests.
7. ***Operate*** - After deployment, operations teams monitor and maintain the system. This includes ensuring uptime, scaling resources as necessary, and addressing any runtime issues.

8. ***Monitor*** - The software's performance and health are continuously monitored. Monitoring tools can detect and alert teams to anomalies or failures. Feedback from monitoring is used to identify areas of improvement, and the loop begins anew.

These steps plus the previously explored 12-factor manifesto lead us to the notion of ***CI/CD***. This acronym stands for ***Continuous Integration and Continuous Delivery/Deployment***. The "CD" part often refers to two possible options:

Continuous Delivery –
: An extension of CI. It ensures that the code is always in a deployable state after passing CI pipelines. In this case the release process is still manual.

Continuous Deployment –
: While Continuous Delivery ensures code is always deployable, Continuous Deployment goes a step further. Every change that passes the CI tests is automatically deployed to the production environment without manual intervention.

These are fundamental principles in the DevOps philosophy, emphasizing the automation of the software development and deployment processes. They are designed to increase the velocity, efficiency, and reliability of software releases, based on the eight iterative DevOps steps.

GitOps and Infra as Code (IaC)

GitOps is a way of doing DevOps using git repositories as the source of truth for all configuration and code and automating deployments. Using git principles provides a clear and versioned history of changes, enabling easy rollbacks, auditability, and peer review. Then, once changes are merged into the main branch of the repository, they are automatically applied to the infrastructure. This reduces manual steps and errors and ensures a consistent state of infrastructure.

Basically, it allows CI/CD tools to handle the ***infrastructure as code (IaC)***, which is the practice of storing the definition of infrastructure as one or more files. This replaces the traditional model where infrastructure as a service is provisioned manually, usually through shell scripts or other configuration tools. Examples of these tools include Terraform, Ansible, Puppet, and other native ones from the public cloud providers.

Declarative vs Imperative

One of the key concepts in cloud native is the notion of declarative or imperative, which focuses on two different programming paradigms, in this case to specify the expected configuration for cloud native systems:

Declarative –
: It requires that you specify what you want to achieve without detailing the step-by-step procedures to get there. For example, picture it like setting a destination in your GPS. You specify where you want to go, but you don't need to provide the GPS with turn-by-turn directions, because iit figures out the best route on its own. In software terms, this often involves setting configurations or states that the system should adhere to, and letting the underlying logic determine how to accomplish that.

Imperative –
: Now, imagine you're giving someone step-by-step directions to reach a particular destination. This embodies the imperative approach. In an imperative style, you give a series of commands or steps that need to be executed in sequence to achieve the desired result. It's like writing a detailed recipe where each step is explicitly stated. In programming, this translates to writing out each action and logic operation in detail.

Stateful vs Stateless

Stateful and Stateless are two fundamental concepts in computing and software design that describe how systems or processes retain information over time and across interactions. Concretely:

Stateful –
: A system or process is considered stateful when it retains or remembers information from one transaction to the next. This "state" information can be used to influence future transactions, meaning that past interactions can affect future ones.

Stateless –
: A system or process is considered stateless when it doesn't retain any record of previous interactions. Each transaction is processed without any memory of past transactions. Every request is treated as an entirely new operation.

Egress and Ingress

These two terms are commonly used in telematics / networking and cloud environments to describe the flow of traffic in relation to a particular point of reference, usually a network boundary or a specific system. This term is applicable to Kubernetes, and the direction of the traffic coming and leaving to/from the cluster. Concretely:

Ingress –
: Ingress refers to the incoming traffic that is being routed to a specific network, system, or application. It's essentially about managing and allowing external users

or systems to access your internal resources. In the K8s, an "Ingress" is also a specific object that manages external access to services within a cluster.

Egress –
: Egress pertains to the outgoing traffic that is being routed out of a specific network, system, or application. It's about managing and controlling the access of internal resources to external destinations.

In cloud computing, providers usually have controls and tools to manage egress traffic due to both cost and security concerns. For instance, an organization may want to restrict or monitor which external services a cloud-based application can access.

Serverless

Serverless is a cloud native development model that allows developers to build and run applications without having to manage servers. There are still servers in serverless, but they are abstracted away from the developer. A cloud provider handles the routine work of provisioning, maintaining, and scaling the server infrastructure. Developers can simply package their code in containers for deployment. Once deployed, serverless apps respond to demand and automatically scale up and down as needed. Serverless offerings from public cloud providers are usually metered on-demand through an event-driven execution model. As a result, when a serverless function is sitting idle, it doesn't cost anything.

These are just a few terms that you need to know before starting Chapters 5 and 6, that will exclusively focus on Kubernetes and its technical details. But before that, we will get our regular end-of-chapter expert insights, in this case from someone who knows the KCNA and other Kubernetes certifications very well: Benjamin Muschko, who is a renown cloud native consultant, and the author of the O'Reilly CKA, CKAD, and CKS Study Guides.

CHAPTER 5

The Key Role of Kubernetes

In this chapter we will explore the change in software architecture paradigm changes and the role that Kubernetes plays to streamline either the development and operation of applications in the Cloud Native realm and the industry adoption since its inception to now. Understanding the ecosystem around container technology and in which scenarios Kubernetes excels over other technologies is a journey, a worth it one. Passion and excitement; both are behind the rising of container orchestration technologies where the intersection of industry momentum , the relentless open source community and the industry shifting towards a more standardized and interoperable ecosystem has brought us where we stand today.

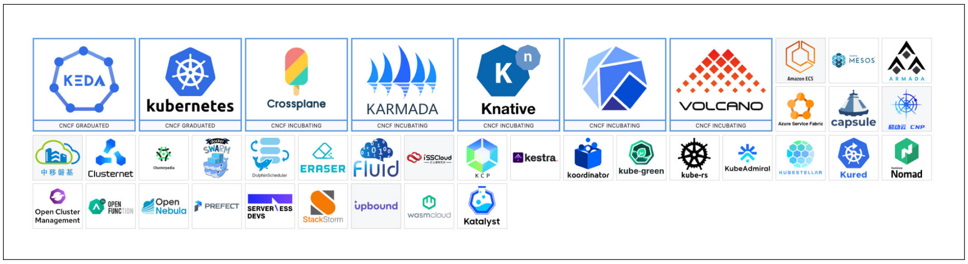


Figure 5-1. CNCF Orchestration and Management Technology landscape

As you can observe in the Figure 5-1, the landscape of projects in the Orchestration and Management are is huge with Kubernetes being the project of reference as of today.

In short this chapter will help you to establish a foundational knowledge and solid roots on the basics of Kubernetes which going deep as long as the you advances in the CNCF certification paths.

Let's uncover the Key role of Kubernetes through the following sections!

An introduction to containers and orchestration technologies

In the previous chapters you probably came to the conclusion that the way applications are built and deployed today has fundamentally changed with the rise of containerization adoption. And that change of the application has opened up a rich variety of tools available which are part of a healthy ecosystem where new projects are being incubated at the time you are reading this book.

A couple of words resonate in many different forums and discussions: "Cloud Native"...

But what is Cloud Native? as introduced in the previous chapter:

> "*Cloud native* is an approach to developing and deploying applications that takes full advantage of cloud computing capabilities and leverages cloud-native technologies and practices. It is about designing, building, and running applications specifically for the cloud environment, with a focus on scalability, resilience, and agility. They make use of microservices and containerization technologies, such as Docker, to package applications and their dependencies into lightweight, portable containers that are often managed and orchestrated using tools like Kubernetes"

However, looking at the commonly known "cloud native" ecosystem from an architecture and business angle can be overwhelming. And both angles are related one each other....

It does not make sense to define and deploy a “by the book” cloud native architecture if it does not fit into the business borders we navigate though. Both sides need to be aligned because one cannot survive without the other. As simple as that.

No solution has only benefits without drawbacks and taking a decision for choosing one tool over another without understanding the bigger picture can lead to poor decisions and add technical debt instead of the other way around.

This book is not an architecture design reference for Cloud Native architectures (specifically event-driven and/or microservices architecture) as there are specific books covering those areas in depth.

However it pretends to be the reference guide for understanding the overall spectrum and getting a good general knowledge about the community behind the main projects and the industry around container technology.

So let’s delve into the container technology through the different milestones that shaped the industry as of today.

Containers

There are many different ways to explain what this technology is. It’s more important, however, to understand the problem it tries to solve.

For our purposes, containers are a lightweight, portable and fast way to package and deploy software (along with all their dependencies) reducing the management burden needed to consistently deliver value deploying fixes, updates and new features in production. It is as simple and as complex as that.

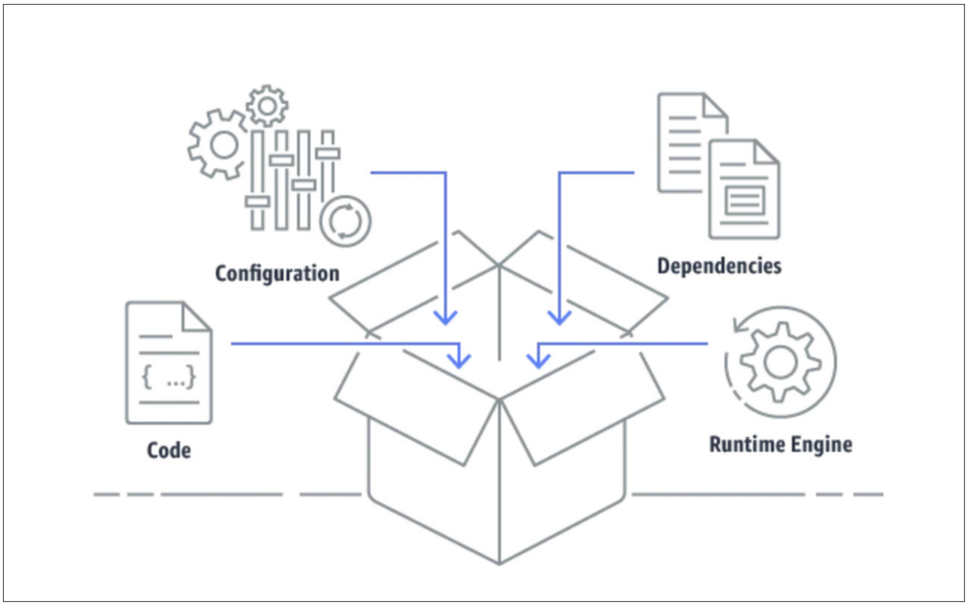


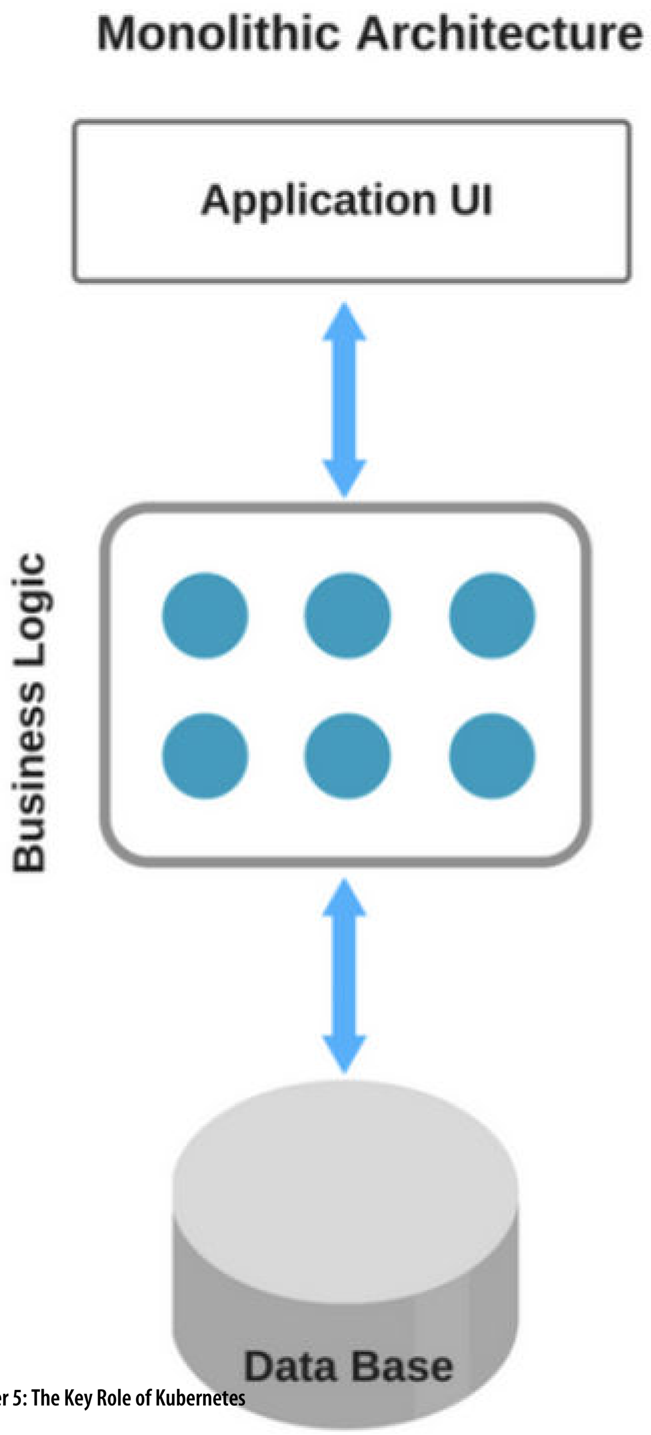
Figure 5-2. Inside a container

If you are familiar with software development techniques or IT operations, you may be thinking about all the different areas a technology like containers has the potential to impact. Perhaps you work as a developer in a well established organization where the software shipment processes have been established for many years. Or perhaps you work in the operations team keeping the lights on. Either way you will soon realize that this technology not only has a potential impact on the development and operational side. There are other sides like organization's teams cultural aspects which need to be taken into account if you pretend to adopt a container first strategy. Setting a DevSecOps culture needs to be in place if you want your team to succeed in the Cloud native era.

Let me share with you a personal story that I hope helps you to understand the need for doing things differently in the application development and software architecture area.

Many years ago I had the luck of being a Product Manager in the world's leading provider for networking and mobile networks equipment. It was 2013 and at that time almost every telco operator had our technology to provide 2G, 3G and 4G mobile networks. For the ones familiar with Telco business, there are two domains which are key for any telco operator: Oss (operational support systems) and Bss (Business support systems). I worked in the engineering team developing the Oss applications (application in charge of network operation, including monitoring and fault management among others).

This application was used in almost every telco operator in the world for operating their mobile's network. As you can imagine (remember it is 2013) it was a 3-tier monolithic application with the middleware tier containing different functions but tightly coupled between all of them.


Monolithic Architecture
Application UI
Business Logic
Data Base

Managing the radio network is a mission critical task for any operator so the application had to be reliable and deployed in high availability configuration. Nothing special for 2013 mission critical systems….

However, at the time that radio access network equipment evolved to offer new cells to address network capacity and coverage issues (pico cells, micro cells, etc..) the amount of Nodes connected to the application grew exponentially. And because of the components being highly coupled between them the only option to accommodate the new radio nodes was to scale the application vertically (i.e providing more CPU, RAM and Disk to the server where each tier run). But as you can anticipate this is a valid approach until reaching the limits of the hardware or the underlying software itself (not to mention the cost in terms of downtime required, software fine-tuning and a long list of drawbacks that it implies).

We were facing the "nice to have" scaling issue (I put "nice to have" in quotes because it's known that this issue appears when an application experiences such a great success that user adoption grows exponentially and therefore business is expected to grow).

However, the scaling issue for a mission critical application which is responsible to keep every mobile network working is more complex than other scenarios (telco networks are considered critical infrastructure in most of the countries around the world, hence needs to be in compliance with very hard requirements).

We struggled scaling every tier but specially hard was to scale the persistence layer (the database engine) because the underlying technology had it's own limits to scale vertically.

And if you are thinking that scale was only an issue to accommodate new nodes in the network, you are seeing only one side of the bigger picture. At that time we were stuck with an application that had a huge impact in several areas of the company:

- The backlog of new features for the radio nodes required the Oss application to be able to configure and handle it properly. So if the application couldn't manage this requirement, it would put in trouble the radio network roadmap (and therefore signing new contracts with the operators)
- The backlog of use-cases for the app itself started to grow and teams struggled to deliver new functionality and keep commitments about deliveries.
- The application that was awarded a few years ago for being state of the art in terms of reliability and security was suddenly a software stack unable to deliver the functionality required for the new radio scenarios.

You can think that this could have been anticipated (and in fact it was anticipated few years earlier) but the growth rate was bigger than we could even imagine. Nowadays we are tired to see examples of applications scaling to almost unlimited levels but in the timeframe 2011-2013 there were not such examples to learn or even anticipate the hyperscale issues.

We took every software piece to its practical limit and even with that it was far to our requirements.

This was the trigger to shift to a micro-services architecture which could potentially scale limitless and accommodate any radio network configuration and size operation.

Well, I changed the above story a little bit (only the date where we started to see that our architecture would reach the limits) to show you a real example about scaling. The truth is that we anticipated that our application would reach its limits soon and started working in parallel in a new generation Oss application. However it was not until 2014 where we were able to start testing it in the real world and we had to struggle with the issues I mentioned above in the former application.

Scaling is a major barrier for growth and expansion even having a software architecture highly decoupled. Scale needs to be managed and the scale an application can grow today in a very interconnected world is much faster and in a different pattern than the paradigm it was before.

In a situation like this, container technology is your ally for making scale a truly “nice to have” issue.

Working with container technology is usually a great fit for different application architecture patterns (and also a big “don’t” for anti patterns) but especially for those whose domains of functionality are split in smaller pieces of software , generally independent and communicated through well defined APIs (in the application mentioned earlier some examples of domains were: node performance collector, node fault management, node synchronization, different user views, etc…..)

That’s basically what microservices architectures are used for and the number of microservices running to shape an application can be large (or extremely large). Commonly a microservice will run in 1 to n containers and the complexity of management rises exponentially when more microservices are waiting to be handled. Let’s just try to imagine the amount of microservices belonging to our OSS application mentioned earlier (from hundreds to thousands of them) and now, let’s extrapolate that to the whole set of applications needed in an organization…….and…. Voilá….. A perfect operational storm again for developer and operations teams!

Figure 5-4. shows an image I included in my first container presentations when I worked for RedHat trying to explain why a container management is required to avoid a disaster.

This image helps us to understand how putting containers in production without any control can lead to issues. Think of the ship as the hardware where your containers will be running. You need to find the right balance when putting (deploying) containers in the host machine having into account the size of the container, the content inside, etc. Otherwise it can make your server requirements unbalanced and struggling to serve the containers needs degrading the performance (and the user experience). The whole ship is sinking.

Managing containers at scale is challenging, especially when dealing with large numbers of containers and complex application architectures. This is where container orchestration comes in.

Container orchestration

Container orchestration provides a useful set of functionalities, which are always evolving as the industry requirements and use cases bring new requirements. For the sake of simplicity we will define container orchestration as the cornerstone piece which provides us the required functionality to automate the deployment, scaling, and management of containerized applications.

There are different solutions today that aim to solve the containerization management and operation at scale challenges, each of them with some similarities but differences among architecture, deployment, management and underlying technologies.

We will dig deeper in the following sections to understand the ecosystem and the different solutions and implementations available but the bare minimum capabilities that a container orchestrator should provide are:

- Provisioning and Deployment of containers
- Scaling based on workload requirements
- Load balancing that allows us to distribute incoming traffic across different container instances keep an optimal performance.
- Resource Allocation utilizing compute resources (CPU, Memory and Disk) efficiently while maintaining application performance
- Health Monitoring and Recovery so if a container fails or becomes unhealthy the orchestrator can automatically restart it or instantiate a new one.
- Service Discovery capabilities allowing containers to find and communicate with each other.
- Security and Isolation enforcing security policies defined by admins preventing or reducing security breach and vulnerabilities exposure.

Containers ecosystem and standards

With the proliferation of different container platforms during the last few years it started to be difficult to use containers across different platforms. And many are still confused with some terms and even with the relationship between Docker Inc (the company) and docker (as technology).

Let's quickly look back to 2013 when Docker Inc created a tool named Docker built on top of the already existing LXC stack. Yes, you read it correctly: 2013 . However the technology behind Docker was present in the Linux Kernel several years before 2013.

Linux Container

LXC or "Linux Container" as it is known, emerged as a distinctive method of operating-system-level virtualization with the purpose of orchestrating the operation of numerous discrete Linux systems, known as containers.

In contrast to conventional virtual machines that establish complete autonomous virtual systems, LXC instead builds a simulated environment with its unique allocation of resources such as CPU processing power, memory, block I/O, and networking capacity, among others. The underlying mechanisms that enable this resource governance are "namespaces" and "cgroups" within the Linux kernel that governs the LXC host .

Namespaces:

Namespaces do a very special job in a Linux box – they keep processes isolated from each other. We can simplify its function saying that namespaces provide process isolation inside the operating system.

There are 6 common types of namespaces in wide use today:

- PID Namespace: In Unix-like operating systems, every process receives a unique numeric identifier called a Process ID (PID) when it's created. Even if multiple processes share the same human-readable name, their PIDs distinguish them.

  As described in the man page:

  "PID namespaces allow containers to provide functionality such as suspending/resuming the set of processes in the container and migrating the container to a new host while the processes inside the container maintain the same PIDs."
- Net Namespaces: Imagine a network environment where processes function independently, separate from the host and other namespaces. Each net namespace has its unique network interfaces, routing tables, firewall rules, and network settings.
- UTS Namespace (Unix Timesharing System): For simplicity let's assume it provides a way to isolate two specific system identifiers: the hostname and the NIS domain name.
- User Namespace: The user namespace allows to create isolated environments for processes.
- Each user namespace has its own set of user and group ID (uid and gid) that are separate from the global system.
- Mnt Namespace: It allows to isolate mount points such that processes in different namespaces cannot view each others' files.
- Inter-process communication namespace: This namespace is probably one of the most complex. For simplicity let's think about it as the functionality that allows isolating communication resources between processes ensuring that processes within the same namespace can interact with shared memory, semaphores and message queues while remaining separate from processes in a different namnespace.

Cgroups (or Control Groups)

In simple terms, cgroups are like traffic cops for important resources such as CPU, memory, network, and disk usage. They ensure that each process or group of processes doesn't hog too much of these resources. They're crucial for containers because containers often have many different processes running together, and cgroups help handle them as a unit.

These two pieces of software are the foundation of the technology we use for running containers. Amazing, isn't it?

At that time (2013) , what Docker released (and it changed the software shipment paradigm) was that the user, using a tool called "docker" , had the capabilities to package containers into images (the concept of images is crucial) to move them between machines in a very easy way. Docker were the first ones who tried to make containers a standard software unit, as they state in their "Standard Container Manifesto".

At this point, you are probably seeing some similarities with the virtualization technology sunrise. But this time, with the previous experience in the industry around the virtualization technology development and the absence of standards for the technology, it was clear for the actors participating in the container development that a set of standards should be promoted to guarantee interoperability between platforms , avoiding vendor lock in and contributing to a healthy ecosystem of tools.

Note: I always highlight that we have to give credit to Docker as a company for starting and actively supporting many initiatives around standardization of container technology and donating projects as important as runc (we will talk about it in the next chapter).

Then, What do the Container standards provide?

They provide a common set of specifications and guidelines for container platforms, making it easier to use containers across different platforms providing compatibility as long as the standards are followed by the solutions' ecosystem. Following the standards will ensure that containers are portable and interoperable, regardless of where they are deployed.

With more than 20 years working in technology I must admit this is the very first time I've seen a real commitment from each actor in the industry to make interoperability a reality and push for it at every layer.

The most important and widely known standard in the container ecosystem is the Open Container Initiative (OCI).

The OCI is a collaborative project, formed under the auspices of the Linux Foundation with the purpose of creating open industry standards around container formats and runtimes. The OCI was launched on June 22nd 2015 by Docker, CoreOS and other leaders in the container industry.

Many people mistakenly believe that the Open Container Initiative (OCI) is only relevant to Linux container technologies due to its affiliation with the Linux Foundation.

However, this perception is misguided and while Docker technology originated in the Linux ecosystem, Docker worked closely with Microsoft to expand the container technology, platform, and tooling to the world of Windows Server. In fact the technology that Docker has contributed to the OCI is not limited to any specific architecture or operating system. It could be applied to any multi-architecture environment.

At the time of writing this book, the list of Members of the OCI includes a variety of actors in the industry, as it can be seen in the following figure:

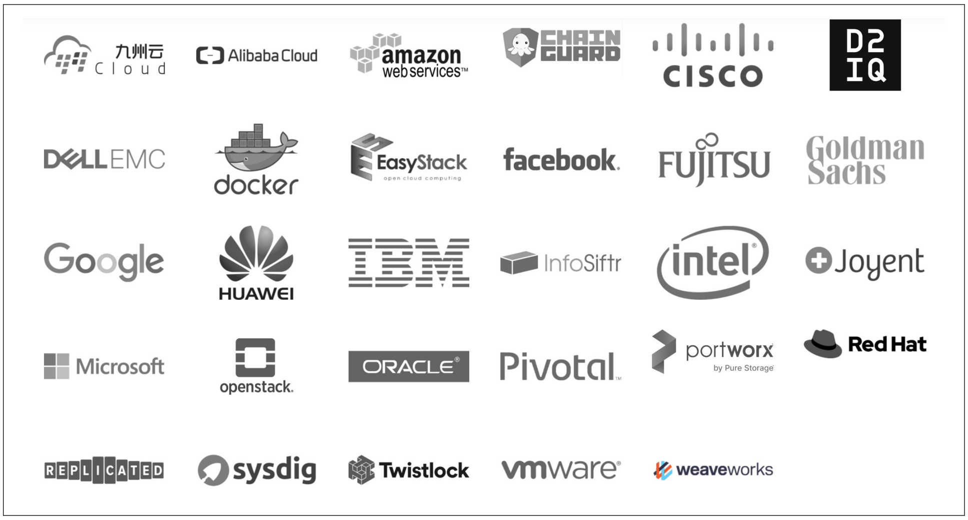


Figure 5-5. List of OCI Members in 2023

Container Solutions

Now that we have seen the dates for key events in the containerisation ecosystem and we have presented the main standard available, let's have a look at the container solutions available in the realm of container orchestration and are commonly used:

Docker Swarm

It is the first Docker's container orchestration project with the first version released in 2014. Prior to Docker version 1.12 it was the only native Docker option for clustering hosts. It was installed as standalone and used to create a cluster of Docker engines, enabling us to connect multiple Docker engines together and manage them as a single entity.

Starting from Docker 1.12 a set of features were added to the core Docker Engine making multi-host and multi-container orchestration easier. With this release it was not required to install the Docker Swarm software as a standalone deployment anymore. Instead it was included in the Docker Engine and starting a swarm was called "Enabling swarm mode".

Today, Swarm mode (based on the swarmkit project) is the only solution from Docker to manage and orchestrate Docker hosts.

In 2020 the Docker swarm project was renamed to classicswarm and was finally archived 1st February 2021.

Nomad

Nomad is a container orchestrator and workload scheduler developed by HashiCorp and first version was released in 2015.

You can think in Nomad as a lightweight orchestrator as it is architecturally simpler than other solutions in part because it is focused in cluster management and scheduling designed with the Unix philosophy of having a small scope. It relies in external tools like Consul for service discovery/service mesh and Vault for secret management for instance.

Kubernetes

Though we are going to dig deeper into what Kubernetes is in the next section and next chapter, it's important to present it here as it's part of the ecosystem of container solutions and one of the most important projects in the container area nowadays.

Kubernetes or K8s in its abbreviated name is an open-source platform built with the purpose of automating the orchestration, scaling, management, and deployment of applications encapsulated within containers. Its role is to streamline the management of applications that are constructed and launched via containerization, including technologies like Docker containers.

The current ecosystem around container technology is so huge that there are also tools which we can consider meta orchestrators as they allow the management of different container orchestration solutions from a single and centralized tool. This is the case of Rancher , developed by Rancher Labs and acquired by Suse in 2020. Rancher versions 1.x supported Swarm, Mesos and Kubernetes orchestrators and its own orchestrator called Cattle. From version 2.x onwards Rancher is focused on being a management platform for deploying and running Kubernetes clusters anywhere and on any provider.

What is Kubernetes?

We briefly presented Kubernetes in the previous section but for understanding it from its roots we have to look back to 2003/2004 and the different stages that made it possible to be the project it's today.

Going back to 2003/2004, it was the time when Google introduced "Borg" system (which is known as the Kubernetes' prelude).

Borg is essentially a cluster manager that orchestrates the ballet of thousands of applications across a sprawling web of machines at Google. We can think of Borg as the precursor to Kubernetes, and as an example of how innovation transforms the way we wield computing power.

Before Kubernetes took the open-source world by storm, Borg was already redefining how applications interact with machines.

The following figure shows Borg's architecture which was included in a paper written by Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, John Wilkes

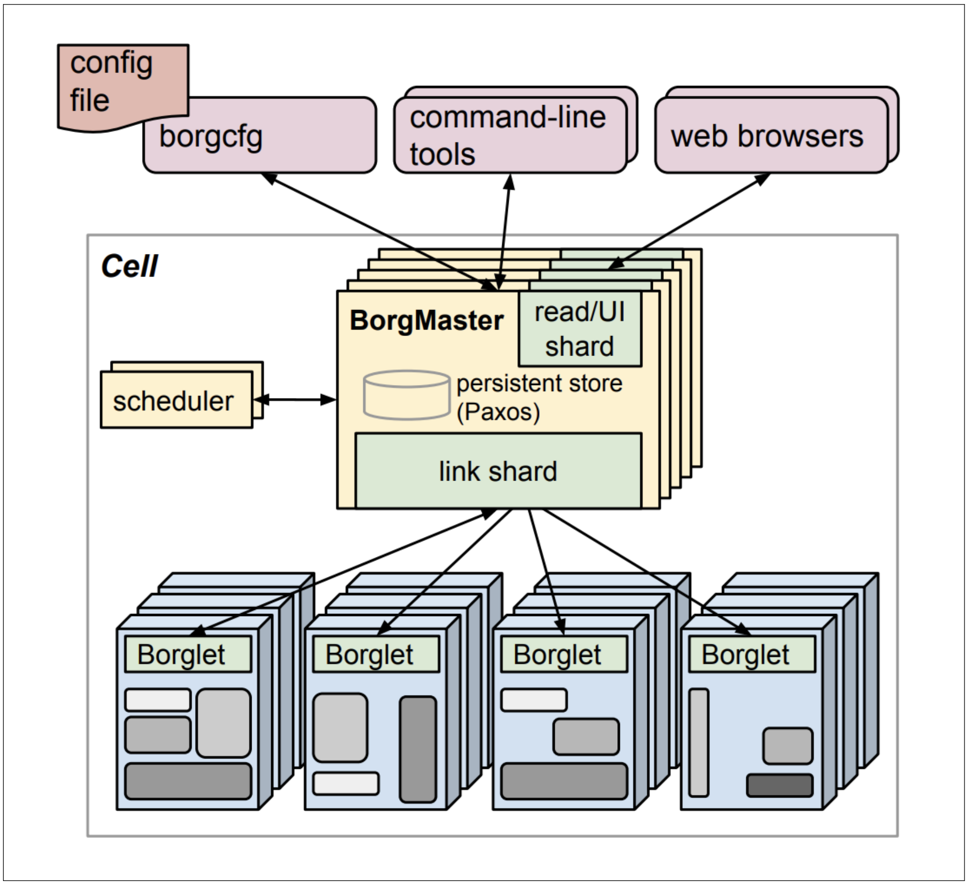


Figure 5-6. Borg's architecture

In 2014 Google announced kubernetes, a project created by Joe Beda, Brendan Burns, and Craig McLuckie who were soon joined by other Google engineers, including Brian Grant and Tim Hockin.

That's the reason Kubernetes design and architecture has its roots at project Borg as many of the developers at Google working on Kubernetes were formerly developers on the Borg project.

As soon as the Kubernetes community was created in mid 2014, several companies joined the project (Red Hat, Microsoft , IBM and Docker) and July 2015 is the month when the Kubernetes 1.0 version was released since Google considered it was production ready. Together with that release, Google also donated the Kubernetes project to a newly formed foundation — the Cloud Native Computing Foundation (CNCF) that will be run by the Linux Foundation- making Kubernetes the very first project under the governance of the CNCF.

And Why is Kubernetes so important and why should you learn it?

Kubernetes adoption has grown rapidly since the release of its first version and the pace of adoption is accelerating year over year.

Dynatrace (a global technology company known for providing a software observability platform) run a report that shows some interesting figures about Kubernetes adoption:

- In 2022 Kubernetes became the key platform for moving workloads to the public cloud
- With an annual growth rate of +127 percent, the number of Kubernetes clusters hosted in the cloud grew about five times as fast as clusters hosted on-premises.

The rhythm of adoption together with the industry support and contribution to Kubernetes project supports the market message around Kubernetes being the facto container orchestration standard.

There are many factors which have contributed to Kubernetes becoming the industry standard for container orchestration, but among its numerous advantages and benefits one of the primary reasons for Kubernetes' success is that it is an open-source platform maintained by the Cloud Native Computing Foundation (CNCF).

As presented in previous chapters of this book, the CNCF is a vendor-neutral organization that promotes the adoption of cloud-native technologies, and with that mission the CNCF ensures that Kubernetes is developed in an open and transparent manner, and is not tied to any specific vendor or platform.

This has helped to create a thriving ecosystem around Kubernetes, with a large and active community of developers and contributors leading innovation and adoption across the industry.

But this adoption wave has brought its own challenges, especially the ones related to scaling, operating and managing the applications and the underlying technology supporting them. As these challenges started it has positively contributed to drive Devops adoption practices in many organizations and teams, requiring a shift in the way Developers, Operations and Security teams work together, promoting a DevSecOps culture in many organizations.

Current Industry trends

We are living in an exciting time of technological evolution where numerous projects in the open source community are maturing from incubation stages to mature projects and of course, new projects are entering incubation.

The CNCF website itself provides this information through the CNCF landscape.

At the time of writing, there are 36 projects that have entered the incubation phase, the vast majority of which belong to the Kubernetes ecosystem.

Although it is not the purpose of this book to go through all the projects in the Kubernetes ecosystem (which would make this book endless) it is important that you keep a generic view to understand the momentum that the open source community is experiencing in the Kubernetes ecosystem, specifically under the CNCF.

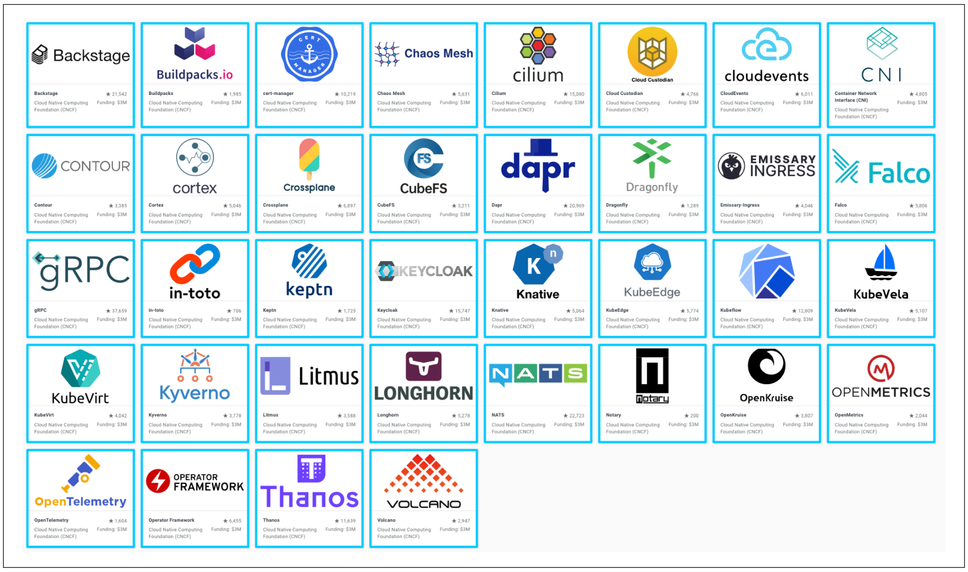


Figure 5-7. Source: Cloud Native Landscape (cncf.io)

Under this umbrella of constant evolution in the industry it is risky making predictions about the evolution of the containers and container orchestrators ecosystem, but there are determinant drivers that will mark its evolution and future development. Let's take a look at the most significant ones:

Artificial Intelligence

2023 can be marked as the year one of Generative AI. It was the year where artificial intelligence systems, specifically those oriented to the Natural Language Processing (NLP) subset and thanks to the emergence of LLMs presented new ways of interacting with different kinds of data and also providing capabilities to what it's call Augmented Intelligence through embedding AI in a variety of applications.

It is specifically in this framework, embedding AI in applications, where containers and orchestrators such as Kubernetes are and will be key in the coming years.

As the use cases of generative AI increase and its use is democratized and AI is infused into new or existing applications, having a modern infrastructure supporting software architectures capable of behaving elastically to adapt to the usage patterns of the same and being highly scalable will be the norm, and here containers and specifically Kubernetes are positioned as the preferred technology to host these applications.

But it is not only the use cases derived from generative AI that will further accelerate (if possible) the evolution and adoption of Kubernetes. Current deep learning models that allow to emulate in some way some of the human cognitive functions, through pre-trained models ready to be used, are a good match to be deployed in containers. From computer vision models to services capable of translating speech and text, there are today models ready to be deployed on containers, allowing a quicker and easier access to the technology.

Apart from the two shared scenarios where AI will have a major impact on the evolution and adoption of Kubernetes, there is a third stream of evolution underway that has to do with simplifying the operation of Kubernetes clusters.

Kubernetes is a complex system, as we will see in the next chapter, and its management and operation are not straightforward. Enterprises and teams often underestimate the complexity of Kubernetes and containers at scale, and underestimate the amount of both, expertise and tooling needed to operate Kubernetes the right way.

This is creating a barrier for enterprises that want to adopt Kubernetes or scale its use in those that have already begun adoption. In addition to this learning curve, there is also a shortage of talent with solid skills to successfully tackle these projects.

The open source community is working hard with focus on projects to simplify the use of Kubernetes with the help of AI. Projects such as k8sgpt-ai are a good example of the efforts to simplify Kubernetes operations specifically for Site Reliability Engineers (SRE).

Vector Databases

Vector databases provide a powerful and efficient way to store, index and work with high-dimensional vector data making them essential for applications like recommendation engines, object detection, audio recognition and any modern AI-driven application.

These databases store information in the form of vectors, which are mathematical representations of objects or data in a multidimensional space where each element of the vector encodes relevant and contextual information about the data it represents. This feature is essential for generative AI, as it allows generative models to understand and manipulate data more effectively.

Let's have a look at the reasons why vector DBs are a fundamental technology in the current paradigm:

Efficient search and recovery: Vector databases allow highly efficient search and retrieval of data using similarity-based algorithms. This is essential in applications such as searching for similar images, retrieving related music and videos, and personalized recommendations on streaming platforms.

Complex Data processing: Vector databases are ideal for handling complex data, such as time series, geospatial information and biological data, as these can be effectively represented as vectors in a multidimensional space.

Generative Artificial Intelligence: The popularity of vector databases has increased significantly due to their relevance in generative artificial intelligence. These databases are used to store representations of data, such as text or images, which can then be creatively manipulated and combined to generate new and realistic content, such as AI-generated text or images.

Machine Learning and Data Analytics: Vector representations are fundamental to a wide range of Machine Learning and data analysis techniques. Vector data are used as inputs for classification, clustering, regression algorithms and more, facilitating the application of predictive models and advanced analytics.

Due to the evolution of artificial intelligence in recent years, it is increasingly common to find applications running machine learning models that require reliability and scalability to respond to an unpredictable number of requests. It is here where once again Kubernetes is the ideal ally for its execution, allowing scaling based on different metrics. This in turn also enables or will enable new scenarios where it may be necessary to run a vector ddbb on a Kubernetes cluster and there are some open source projects that already cover this possibility.

An example from the Linux Foundation's Data projects pillar is the Milvus Vector DB.(Vector database - Milvus).

Another interesting project for the reader that also allows it to run on Kubernetes is Weaviate Welcome | Weaviate - vector database

We will continue to see advances in this fieldand new innovative projects emerging from the OpenSource community.

Edge Computing

Some AI-driven apps are built to cover scenarios where Near Real Time responses are required and that generally the requests are made in environments or devices with difficult access to reliable telecommunications networks.

Also on the rise are use cases where the execution of analytical models or even generative AI models need to be executed embedded (vehicles, home appliances....etc). This is where the concept of Edge Computing takes on more relevance as instead of relying exclusively on remote data centers or the cloud, Edge Computing brings compute capacity directly to the edge of the network, close to where the data is generated.

As edge computing infrastructure has evolved, efficient orchestration of containers and distributed services has become imperative and Kubernetes has become the cornerstone in managing applications and services in edge infrastructure scenarios.

Edge Computing and Kubernetes will continue to redefine the way we interact with technology in an increasingly connected world. As edge applications continue to expand into industries with relevant IoT capabilities such as Industry 4.0 and even telemedicine scenarios, the orchestration and management capabilities offered by Kubernetes will continue to be essential to ensure their execution.

Platform Engineering

Platform Engineering is a holistic approach focused on providing a unified technology platform that enables development teams to efficiently build, deploy and maintain applications without worrying about the underlying complexity of the infrastructure.

The basis of the Platform Engineering paradigm is to treat infrastructure as code (IaC). This means that infrastructure resources, such as servers, networks and databases, are defined and managed by code, which facilitates automation, reproducibility, life cycle management and scalability.

In this context, a technology like Kubernetes, which has revolutionized the way organizations manage and scale applications, is a fundamental piece of the ecosystem of technologies used under the Platform Engineering concept, taking advantage of its main characteristics:

- **Automated deployments:** Kubernetes allows to automate the deployment of containerized applications and services, reducing time to market and minimizing human error.
- **Dynamic Scalability:** Kubernetes provides the mechanism for applications to automatically scale on demand, ensuring optimal performance at all times, whether for a steady stream of users or unexpected traffic spikes.
- **Resource management:** Kubernetes provides advanced resource management, which means that applications can efficiently share underlying resources without conflict.
- **Resilience and fault tolerance:** Kubernetes is designed to handle failures transparently (as long as the application software architecture is designed for this paradigm). If a container or node fails, Kubernetes can automatically re-route traffic to a healthy service and ensure that the application continues to run.
- **Application updates without downtime:** Upgrades and security fixes can be deployed without downtime thanks to Kubernetes' ability to manage multiple versions of an application at the same time.

A Practical Example: The Platform Engineering Team in a fictitious company

To illustrate how Platform Engineering works with Kubernetes, let's consider a team of Platform Engineers at an e-commerce company. This team is responsible for providing a robust technology platform for the web application development team.

They use Kubernetes to create highly available and scalable container clusters. These clusters can manage web applications, databases, caching systems and more. Platform engineers define and maintain all infrastructure resources as code, allowing them to make changes quickly and securely so the developers don't need to worry about Kubernetes and the rest of pieces are correctly configured and secured. They only use the catalog of services available in the platform and use it for building the application.

When the development team releases a new version of the application, platform engineers use Kubernetes to deploy it seamlessly, scaling dynamically on demand. If a server fails, Kubernetes automatically takes care of redirecting traffic to healthy servers. They also constantly monitor the performance and health of the infrastructure to ensure uninterrupted service.

Alternatives for Kubernetes in the Orchestration

Although Kubernetes is widely popular, there are alternatives that allow the ecosystem to continue to innovate in container management and specifically in distributed service orchestration. The choice of a container orchestration platform will always depend on several factors and although Kubernetes is widely accepted as the de facto standard in the industry, factors such as the complexity of an application, the skills of the team (DevOps) and the investments made in infrastructure, can be determinant in the choice.

Knowing the alternative solutions allows us to make an informed decision about the container orchestration and distributed application platform that best suits our boundary conditions.

Let's take a look at the main alternatives to Kubernetes:

Apache Mesos and Marathon

If there is a solution we could argue whether they fit in the container orchestration area or not it's the case of Apache Mesos with Marathon framework. Comparing it with Kubernetes side by side wouldn't be fair as they are not originally built to cover the same functionality but with Marathon as the container orchestrator they can fit to solve the same scenario.

Apache Mesos is a distributed resource (not container) manager that focuses on providing an abstraction layer for resource management in server clusters. Marathon, on the other hand, is a container orchestrator that runs on top of Mesos and allows you to manage containerized applications.

Seeing that both could fit into an architecture of an application designed to be deployed and run in a distributed fashion you would want understand what are the main differences and features of Mesos and Marathon compared to Kubernetes. Without deeping in architecture aspects the main differences are:

- **Focused design on resource management:** The main difference between Mesos and Kubernetes is their focus. Mesos focuses on resource (infrastructure) management, which makes it highly scalable and suitable for managing resources in large-scale environments. Kubernetes, on the other hand, tries to abstract the management of the underlying compute resources and focuses on application orchestration and state management.
- **Configuration and Deployment:** Configuring and deploying a Mesos and Marathon cluster can be more complex than doing so with Kubernetes, especially for users less familiar with cluster-level resource management.

- **Top Scalability ranks:** Mesos and Marathon are designed to manage large-scale applications and handle extremely large and distributed workloads. However, numerous success stories of using Kubernetes in highly scalable and distributed applications have demonstrated the reliability of Kubernetes in this regard, and it can compete with Mesos and Marathon in this regard.

Openshift

Red Hat Openshift (whose upstream project is called OKD) is an orchestration platform whose core technology is Kubernetes. It is therefore an alternative platform but not an alternative technology since the engine of Openshift is Kubernetes.

Red Hat OpenShift differs from Kubernetes in its approach, since being designed as a platform it provides a ready-to-use solution for enterprise environments including all the necessary ecosystem pieces to have a tested and certified application development and deployment platform with interoperability between its different pieces guaranteed by Red Hat. As an enterprise solution, it includes additional security, administration and development features to reduce the learning and adoption curve of the teams.

The main differential features of Openshift are:

- **Advanced security enabled:** OpenShift is also intended for use in highly regulated environments and therefore offers advanced security capabilities such as container isolation and centralized identity and access management as a base. Kubernetes is also utilized in highly regulated environments but the hardening needs to be done adhoc.
- **Reactive Support:** Red Hat offers enterprise support and training services for OpenShift, which makes it a very attractive option for teams that want to abstract from the complexity of managing Kubernetes vanilla clusters (a vanilla kubernetes cluster refers to the most basic setup of Kubernetes) **.**
- **Integrated development:** OpenShift includes all the necessary tools for application development, such as CI/CD pipelines, Jenkins integration and code repositories.

Benefits of Containerization for Businesses

Containerization has evolved from a niche technology to a mainstream solution that is changing how businesses develop, deploy, and manage software. Embracing containerization is not just a technological shift; it's a strategic move that enables organizations to thrive in an increasingly competitive and digital world including a cultural mindset in the teams.

The benefits for an organization embracing container technology adoption are not always easy to measure.

Besides the operational benefits (greater agility, scalability, efficiency, and security) there are other benefits which are not easily measure but nevertheless , has the larger impact in an organization, such as:

Business Agility

Adopting containerization technologies can drive business agility. Organizations can respond quickly to changes in the marketplace, implement new features or services faster, and adapt nimbly to changing customer demands. This agility can make all the difference in a company's ability to stay competitive.

Innovation Acceleration

Containerization fosters an environment conducive to innovation. Development teams can more easily experiment with new ideas and technologies, as implementation and management of containers is faster and less error-prone. This can result in the creation of innovative products or services that set the company apart in the marketplace.

Collaboration

Containerization promotes collaboration between development and operations (DevOps) teams. Standardizing environments and automating deployments facilitates cooperation and communication between these teams, leading to faster and more reliable software delivery.

Resiliency and Business Continuity

The scalability and fault tolerance inherent in containerization can improve a company's resilience to disruptions and disasters. The ability to migrate applications and services quickly and efficiently in the event of failures can minimize downtime and reduce the impact on business continuity.

Talent Attraction and retention

Adopting cutting-edge technologies, such as containerization, can make a company more attractive to technology professionals. Developers and IT professionals are often looking to work in innovative and up-to-date environments, which can help in retaining talent and attracting new, qualified employees.

Customer Satisfaction

The benefits of containerization, such as faster delivery of features and greater application stability, can improve customer satisfaction. Customers appreciate seamless experiences and frequent updates that are responsive to their needs and this will finally results in business improvements.

Summary

We have ended our journey about Kubernetes and its key role in the cloud native realm. We have explored why Kubernetes is a cornerstone of the cloud native ecosystems and its pivotal role in containerization technologies and how it empowers businesses.

From agility and innovation to collaboration and resiliency, Kubernetes offers a robust platform for modern application deployment.

The next chapter will be more practical chapter with code examples equip you with the knowledge to feel comfortable in your first steps with this technology and allowing you pass the exam.

CHAPTER 6

Technical Kubernetes Topics

A Note for Early Release Readers

With Early Release ebooks, you get books in their earliest form—the author's raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 6th chapter of the final book. Please note that the GitHub repo will be made active later on.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at *shunter@oreilly.com*.

The way that applications are built and deployed today has fundamentally changed with the rise of containerization adoption and Kubernetes is rapidly becoming the preferred solution for deploying and managing software in cloud environments. However, it presents a significant learning curve, particularly for newcomers.

In this chapter, our goal is to simplify the Kubernetes landscape by providing a high-level overview of its essential components and their interactions, making it accessible to those who are just getting started.

We will explore the key components and features of Kubernetes, such as the master and worker nodes, the pod lifecycle, the service discovery and networking mechanisms, and the security and observability tools.

We will also present the concept of service mesh, a powerful technology that enhances the management and reliability of microservices architectures.

In the last section you will be provided with a list of most used Kubernetes commands and output samples that will help you to grasp the concepts.

By the end of this chapter, you should have a solid understanding of how Kubernetes works under the hood and how it enables you to build and deploy scalable, resilient, and secure applications.

The innovation pace in the open source community is fast and every new kubernetes release contains tons of new pieces of code that brings new features to the product.

For keeping up the learning with the rhythm of news we encourage you to check the release notes that is delivered with every new release.

LINK *https://kubernetes.io/releases/*

Note about Kubernetes releases:

In general, Kubernetes versions are expressed as `x.y.z`, where:

- **x** is the major version.
- **y** is the minor version.
- **z** is the patch version, following Semantic Versioning terminology.

The Kubernetes 1.19 cycle was far longer than usual. The Release Team extended it to lessen the burden on both Kubernetes contributors and end users due the COVID-19 pandemic. Following this extended release, the Kubernetes 1.20 release became the third, and final, release for 2020. It means that starting from 1.20 release there are 3 release cycles per year (until 1.19 it has a cadence of 4 releases per year).

End users will now experience a **slower release frequency** and a more gradual enhancement graduation rate.

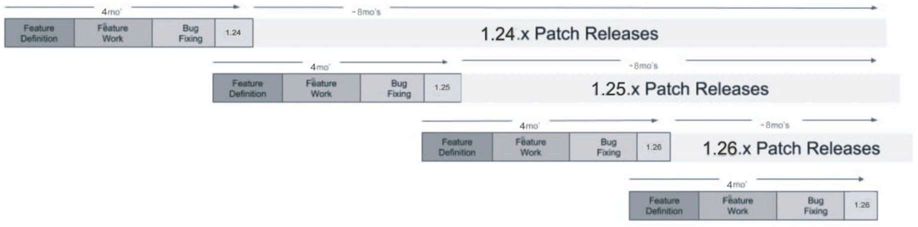


Figure 6-1. Example of release cadence of minor versions.

We strongly recommend you to check the following link in a monthly basis to review the delivery of new patch versions:

https://kubernetes.io/releases/patch-releases/

Kubernetes Basic Concepts: from Containers to Clusters

In the previous chapter of this KCNA study guide we discussed how Kubernetes abstracts away the complexities of infrastructure and empowers people and businesses to focus on their applications rather than operational burden. Now, let's venture beneath the surface and uncover the magic that powers Kubernetes.

Kubernetes Cluster

A Kubernetes cluster is the fundamental building block that allows you managing containerized applications.

At the more basic architecture concepts it is built up from two different planes: Control Plane and Data Plane. We will deep into these concepts later in the chapter but for now let's simplify with the following description:

- Control Plane: You can think of it like the **brain** of Kubernetes. The control plane orchestrates containers, manages clusters, and ensures everything runs smoothly. It's responsible for maintaining the desired state of the system.

- Data Plane: The **data plane** is like the **body** of Kubernetes. It carries out commands from the control plane and handles the actual work.

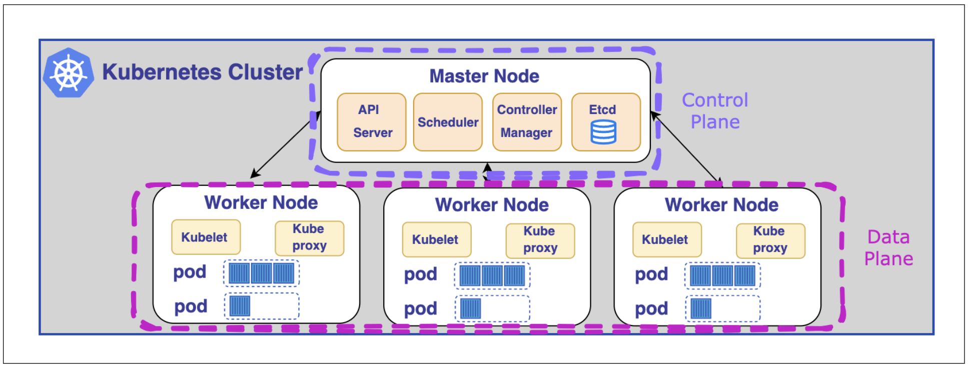


Figure 6-2. Kubernetes Cluster Architecture

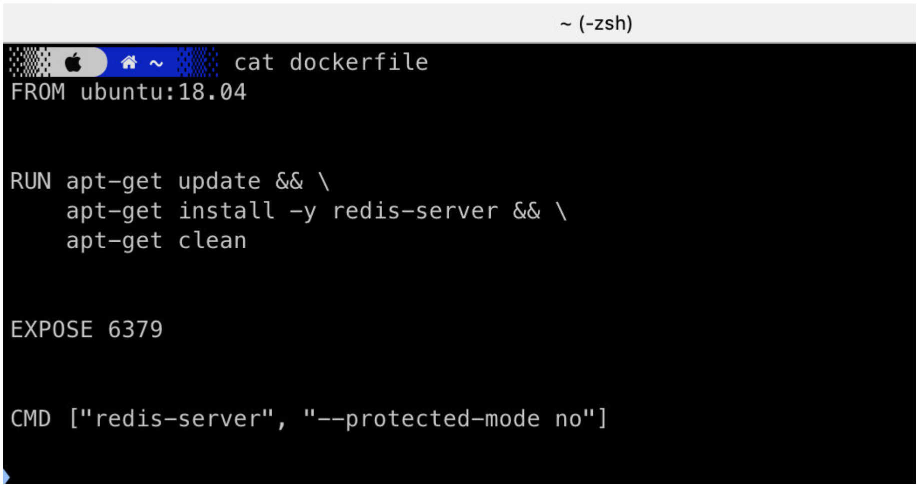
Before going deeper in the main components of each Kubernetes plane, let's present the container and pod concepts in a Kubernetes cluster.

Kubernetes Container:

A "container" in Kubernetes refers to a self-contained, lightweight instance that encapsulates a service or application along with all of its dependencies and configurations necessary for it to function properly. We can think of it as a logical instantiation of a docker file.

Note: What is a Docker file?

A **Dockerfile** is a **simple text document** containing a set of instructions. These instructions guide the creation of a **Docker image**. Think of it as a recipe for building a customized container image where you specify details like the base image, environment variables, file locations, network ports and other components required to run your application.



```
~ (-zsh)
 ~  cat dockerfile
FROM ubuntu:18.04


RUN apt-get update && \
    apt-get install -y redis-server && \
    apt-get clean


EXPOSE 6379


CMD ["redis-server", "--protected-mode no"]
```

Figure 6-3. - Dockerfile to create an image which run a redis-server and expose the port 6379

We recommend learning the basics of creating images from dockerfiles. The following link includes the reference for learning it:

https://docs.docker.com/reference/dockerfile/

In the docker ecosystem, the flow to run a container is showed in the next figure:

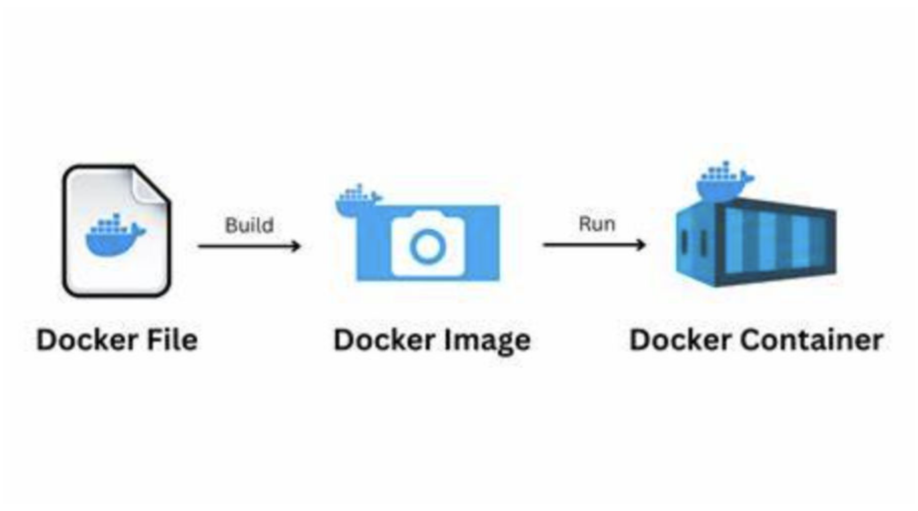


Figure 6-4. Steps for running a Docker container

A container can be a database, a web application, and API, etc....

The concept of a container in Kubernetes is exactly the same as we saw in the previous section of this study guide, highlighting its portability and scalability, meaning that once an application is running in a container, it can run in any Kubernetes environment, whether in development, test or production.

Containers are deployed and run on the worker nodes within the Kubernetes cluster and each node can host multiple containers, each running different applications or services. Besides containers make it easy to package, deploy, and scale applications, and they are the runtime environment for workloads in a Kubernetes cluster.

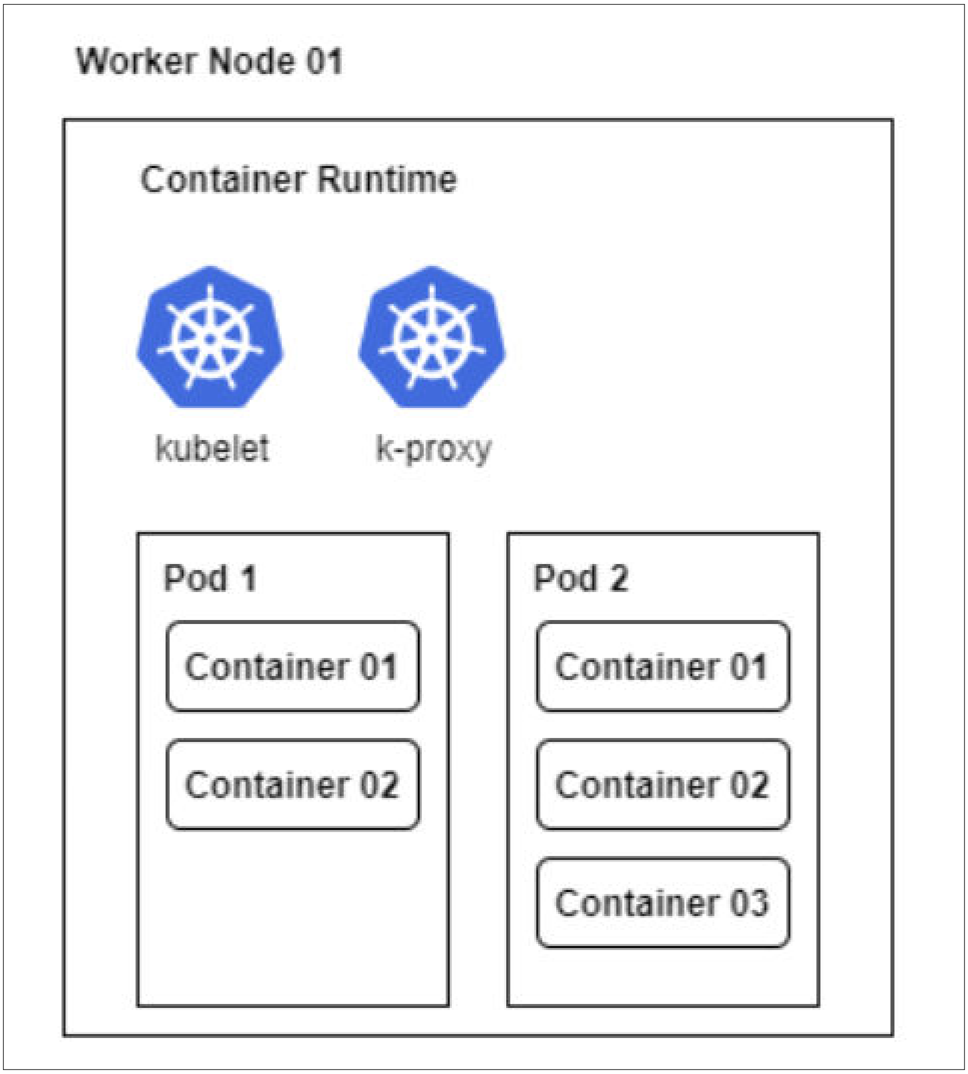


Figure 6-5. Kubernetes Containers

Kubernetes Pod

Pods are the smallest deployable units in Kubernetes and represent a single instance of a running process within the cluster. A pod is seen as a structure which abstracts one or more containers which are deployed together in the same host (see figure 6-6).

Typically an application runs in one or more pods (depending on their architecture).

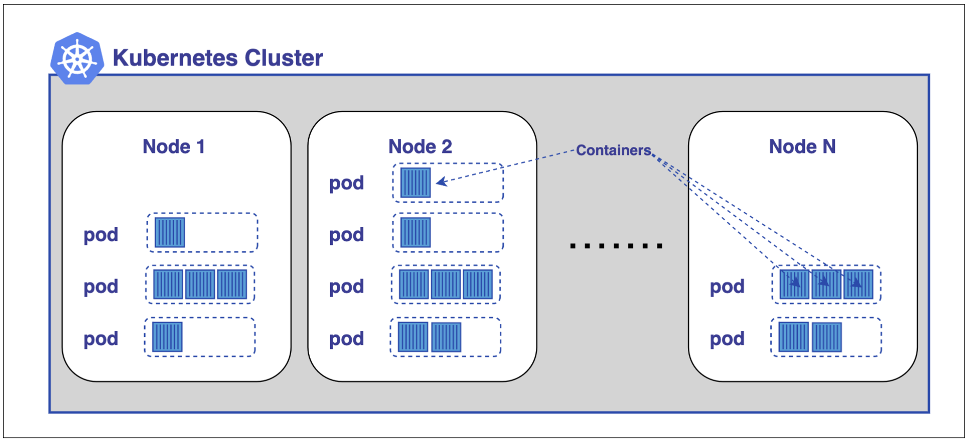


Figure 6-6. Pods running in a Kubernetes Cluster

Now that you know how container and pod fits into the greater Kubernetes architecture, let's go back to our Kubernetes Cluster and go deeper into its components.

The cluster itself provides the necessary infrastructure and resources for the applications to run, including networking and storage ensuring high availability and fault tolerance by distributing the applications across multiple nodes, so if one node fails, an application can continue running on another.

In a Kubernetes cluster, there are different types of nodes, each serving a specific purpose in the overall infrastructure and workload management. These nodes collectively work together to ensure the proper functioning and resilience of the cluster. There can be nodes serving different roles or just one, depending on the cluster configuration needed.

These nodes are part of the two differentiated planes that builds the cluster:

- Control Plane
- Data Plane

Let's delve into the nodes presented in a Kubernetes cluster.:

Master Node

The master node is the control plane of the Kubernetes cluster. It manages and controls all the activities in the cluster, such as scheduling, scaling, and monitoring of workloads. In production clusters, it's common to have multiple master nodes for high availability and redundancy (a minimum of three master nodes).

Key components running on the master node include the Kubernetes API server, which acts as the entry point for cluster management, the controller manager, the scheduler, and etcd, which is a distributed key-value store used for storing cluster configuration data.

Worker Node

Worker nodes are responsible for running the actual workloads, such as containers and pods. They belong to the data plane and execute the tasks assigned to them by the master node. Worker nodes can be scaled horizontally to accommodate more workloads.

Key components running on worker nodes include the Kubelet and the Container Runtime. The Kubelet communicates with the master node and manages containers on the node, and the Container Runtime is responsible for running containers (e.g., Docker or containerd).

Etcd Node (Optional)

Etcd (*https://etcd.io/*) is a distributed key-value store which often runs on the master node. However, in some configurations, especially in larger clusters or highly available setups, etcd might run on dedicated etcd nodes. Etcd nodes function will be the same as if it runs on the master node: responsible for storing the configuration data and the desired state of the cluster.

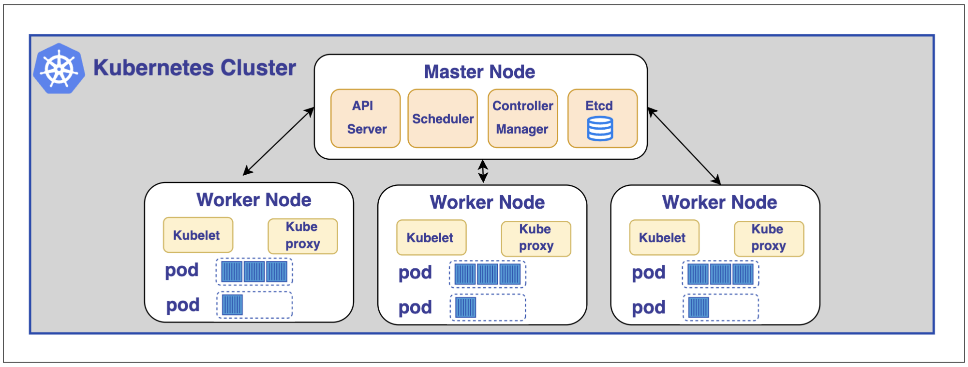


Figure 6-7. Kubernetes Cluster components

Components of the Control Plane

The Kubernetes' control plane manages the overall state of the system and ensures that the desired state is maintained.

While control plane components can be distributed across various machines within the cluster, setup scripts often follow a simplified approach by deploying all control plane components on a single machine (Master Node). Typically, user containers are not scheduled on this machine. Let's have a look at each of the control plane components

Api Server

The Kubernetes API server, a key component of the control plane, serves as the interface for accessing the Kubernetes API. The primary implementation of this server is known as 'kube-apiserver.' It's designed for horizontal scalability, meaning you can expand its capacity by deploying multiple instances. These instances can be utilized to load balance incoming traffic effectively.

From an user perspective the interaction with a Kubernetes cluster can be through the UI provided by Kubernetes Dashboard (*https://kubernetes.io/docs/tasks/access-application-cluster/web-ui-dashboard/*) and/or via Command Line Interface.

Note: Kubectl and Kubeadm

For interacting with Kubernetes Clusters Kubernetes provinces a command-line tool called kubectl. This tool facilitates the communication with the cluster through the Kubernetes API.

Once kubectl is installed in your laptop, the tool looks for a configuration file named kubeconfig in the $HOME/.kube directory (it can look into a different directory by setting the KUBECONFIG environment variable or using the -kubeconfig flag).

The syntax of kubctl commands is: kubectl [command] [TYPE] [NAME] [flags]

Command: Specifies the operation (e.g create, get, describe, delete)

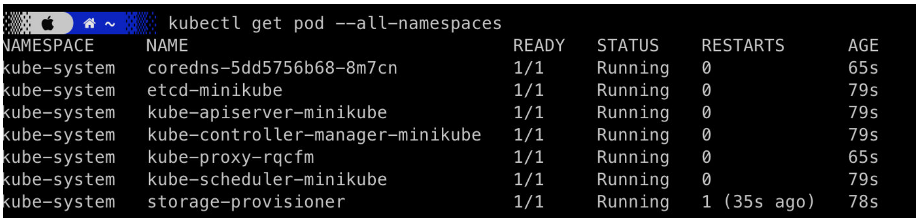
Type: Specifies the resource type (case-insensitive)

Name: Specifies the name of the resource (case-sensitive)

For example:

#kubectl get pods —all-namespaces

This kubectl command will list all the pods present in the cluster.



```
 ~  kubectl get pod --all-namespaces
NAMESPACE     NAME                               READY   STATUS    RESTARTS      AGE
kube-system   coredns-5dd5756b68-8m7cn           1/1     Running   0             65s
kube-system   etcd-minikube                      1/1     Running   0             79s
kube-system   kube-apiserver-minikube            1/1     Running   0             79s
kube-system   kube-controller-manager-minikube   1/1     Running   0             79s
kube-system   kube-proxy-rqcfm                   1/1     Running   0             65s
kube-system   kube-scheduler-minikube            1/1     Running   0             79s
kube-system   storage-provisioner                1/1     Running   1 (35s ago)   78s
```

Figure 6-8. Example of kubectl command in a minikube environment

Kubeadm on the other hand is a tool provided by Kubernetes to simplify the process of creating and bootstrapping a Kubernetes cluster (it's used if you plan to install your own cluster from scratch) performing the necessary actions to get a minimum viable cluster up and running. Kubeadm does not handle the underlying infrastructure provisioning. If you are interested in learning more check the official documentation:

https://kubernetes.io/docs/reference/setup-tools/kubeadm/

Etcd

Etcd serves as a reliable, consistent, and highly-available key-value store that underpins Kubernetes' storage for all cluster-related data. If your Kubernetes cluster relies on etcd as its storage backend, it's crucial to establish a data backup strategy to ensure data integrity and availability.

Alternatives to etcd includePostgreSQL , mySQL or yugabyteDB and it's seeing a great adoption mainly driven to rely on distributed SQL DB engines which provides better reliability and recovery options.

As of today Kubernetes is tightly coupled to etcd so control plane components (API Server) expect an etcd-like interface to which they can write and from which they can read. So for replacing etcd for any other RDBMS a piece of software in charge of translating the underlying RDBMS engine to an etcd-like one is needed.

The project that is widely used for translating etcd API is Kine (*https://github.com/k3s-io/kine*)

Scheduler

The scheduler is responsible for assigning newly created Pods to specific nodes in a Kubernetes cluster. It considers a range of factors when making scheduling decisions, including resource needs, constraints, affinity/anti-affinity rules, data locality, workload interactions, and deadlines.

Controller Manager

The controller manager is a daemon that embeds the core control loops released in Kubernetes.

Note: What are Control Loops?

In applications of robotics and automation a control loop is an endless loop that supervise and readjust the state of the system.

In Kubernetes a controller is a control loop that watch the state of the cluster and then make or request changes where needed.

The main controllers released in Kubernetes are:

- Replication Controller - Ensures the desired number of replicas for a pod.

- Endpoints Controller - Populates the endpoints resource (used for service discovery)
- Namespace Controller: Manages kubernetes namespaces
- ServiceAccount Controller: Creates default service accounts for pods

Note: What are Kubernetes namespaces?

Namespaces in Kubernetes are a similar approach to Linux namespaces. Basically they provide a way to organize and isolate resources within a cluster.

Each namespace serves as a virtual cluster within the physical Kubernetes cluster. Resources (pods, services, volumes,...) within a namespace share a common scope for names.

Namespaces allow you to isolate different projects, teams or customer within the same cluster and provide a mechanism to attach authorization and policies to specific parts of the cluster.

Note: What are ServiceAccount?

A **ServiceAccount** in Kubernetes is a special type of account that represents non-human entities within a cluster. It grants a unique identity, and both application pods and system components can utilize its credentials. This identity serves various purposes, such as authenticating with the API server and enforcing identity-based security policies.

ServiceAccounts play a crucial role in several scenarios:

1. Pods Communicating with the API Server
2. Pods Communicating with External Services (for example authenticating to private container image registries)
3. External Services communicating with the API Server (for example in a CI/CD pipeline to release a new app version)
4. Third party security software: Some tools rely on service account identities to group pods into different security contexts

Cloud Controller Manager

In certain situations, it's beneficial to enable Kubernetes to interact with specific cloud providers or infrastructure platforms using the provider's APIs.

For instance it can be optimal to use the provider's API to handle nodes, configure load balancer or any other infrastructure operation.

With the Kubernetes 1.16 release a new binary called cloud-controller-manager was shipped.

This controller is an optional component in the cluster and the main reason behind shipping it as an optional component is because in earlier releases it was part of the Kubernetes Controller manager. Splitting it in an optional component decoupled it and allows different providers to develop their controller managers.

Similar to the controller-manager, the cloud-controller-manager consolidates multiple independent control loops into a single binary, executed as a single process.

Components of the Data Plane

The Data Plane (sometimes called worker plane) is where workloads (applications) run within a Kubernetes cluster. You can think of it as the plane where containers execute, communicate and process requests.

The foundation of the data plane are the Worker Nodes which are physical or virtualized machines where pods are hosted.

Let's see what are the main components of the Data Plane:

Kubelet

Kubelet is an essential piece of the Data Plane running in each worker node and acting as a bridge between the control plane and the execution of the workloads. Kubelet has an extensive list of critical tasks inside the cluster but the main ones are:

- Pod lifecycle management: Kubelet monitors the health status of pods and performs Starting, stoping and restarting containers as needed. It also receives the commands sent from the control plane regarding pod creation, updates and deletions.
- Kubelet is responsible for ensuring that the pod and containers configuration and status match the desired state specified by the control plane.
- Reporting back the node's status to the control plane: Kubelet communicates with the Kubernetes Control Plane establishing a communication channel for exchanging information required for cluster health and functionality together with the API-Server running in the control plane. Kubelet updates the control plane about the node's health, availability and capacity metrics.

Kube-proxy

The other essential component of the data plane is Kube-proxy and its main area of responsibility is managing network connectivity within the cluster. Among their main tasks are:

Service Loce Balancing:
: When a pod communicate with a service, kube-proxy ensures that requests are load-balanced across the relevant pods.

Iptables or IPVS rules management:
: In Linux nodes there are two modes available in kube-proxy:

  - Iptables Mode:
  - IPVS Mode

  In general you can think of Ip tables rules as simpler to configure but it become inefficient and operationally complex at scale when we work with a large number of services. IPVS is an alternative mode which provides in general better performance.

Network Address Translation:
: NAT is used by kube-proxy for managing traffic between services and pods . It allows services to abstract away the underlying pod network space. When a new service or endpoint is created, it abstracts the underlying pod IPs, allowing other services to communicate with it using the service's virtual IP address.

Note: About Iptables, IPVS and eBPF

Kubernetes relies on networking components to manage communication between services and pods.

In this journey today there are different options available: iptables, IPVS, and the emerging star, eBPF.

Iptables is a technology being around for a while but was not created for an extensive case as Kubernetes networking. Iptables struggle with large rule sets and this leds to performance issues causing latency and CPU overhead.

While IPVS (IP Virtual Server) offers an alternative to iptables for load balancing and service proxying being more efficient than Ip tables it also become complex when dealing with a large number of services and endpoints and struggle to dynamically update rules being less flexible for real-time changes.

In that context eBPF (extender Berkeley Packect Filter) is a rising technology that allows programmable processib within the Linux Kernel. Think of it as a tool that supercharges networking components as eBPF outperforms IPVS significantly through an ease of network management, reduced overhead in terms of resource consumption and improving observability and dynamic updates in real-time.

The industry standard for Kubernetes networking using eBPF technology is Cilium (*https://cilium.io/get-started/*) which is currently a graduated project in the CNCF landscape.

Cilium can replace kube-proxy entirely as it implements a Container Netowrking Interface (CNI) that we will present in the next section.

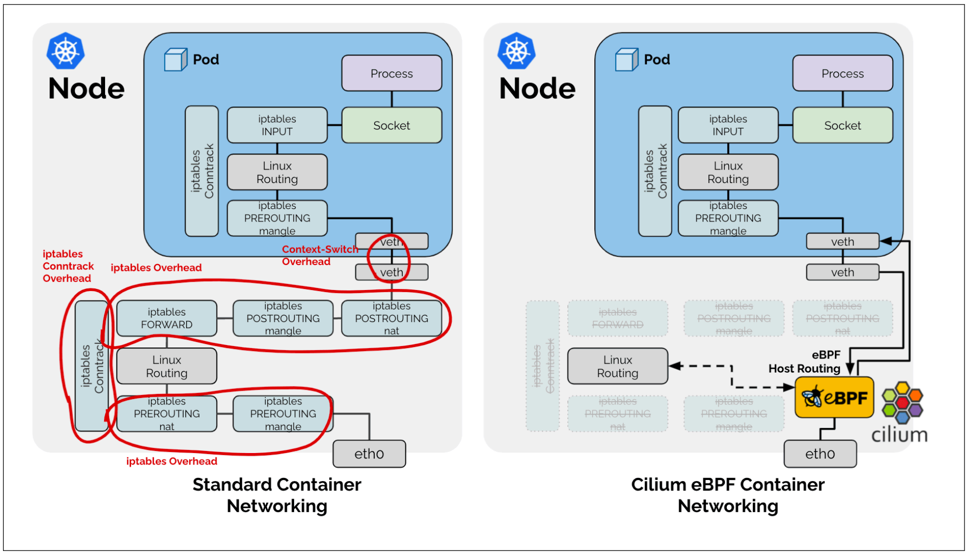


Figure 6-9. Diagram of traffic path with Iptables mode vs Cilium eBPF (source: CNI Benchmark: Understanding Cilium Network Performance)

Inside Kubernetes Architecture

As described in the earlier chapters of this book, Kubernetes is built as a distributed system where many different pieces need to work together to make the system as reliable and trusted as required in today's applications.

For that reason there are architectural building blocks and concepts that need to be well understood to deploy a cluster that fits the requirements or your applications.

In this section we will focus on four fundamental architectural concepts:

- Containers communication basics and single container and multi container pod architecture patterns
- Container Runtime Interface

- Container Network Interface
- Container storage Interface

Kubernetes Single Container Pod and Multi Container Pod

As described earlier in the chapter, Pods are the smallest deployable units in Kubernetes and a pod can be composed of one or various containers. Practically if you want to run a container in a kubernetes cluster you will need to create a pod to run it. This type of construction is called a single container pod. On the other hand, if you need to run several containers in the same pod, you will end up having a multi-container pod.

You may be wondering why a multi-container pod can be required. This is a common question when learning kubernetes and how software architectures can be "deployed" in kubernetes.

Well, imagine that you want to run a web server in Kubernetes which is expected to receive hig-traffic (imagine an e-commerce website during the Black Friday for instance) . The web server will handle user requests, server product pages and process transactions. The cache service (Redis, Memcached…) will store frequently accessed data such as product listings, images, descriptions and user session information reducing the load of the web server ensuring a great user experience.

In this case speed is key and we want both services to be in sync. This is a scenario of tightly coupled processes and a multi-container pod could be a good fit. While the multi-container pod pattern will bring some beneficials in terms of resource efficiency it also introduces some complexity in terms of operational management (dependency challenges, scalability constraints, spin up order,, etc).

The multi-container pod has its own characteristics in terms of intra pod communication and understanding them will facilitate your decision whether to use it or not.

Multicontainer pod communication

There are three multi-container intra pod communication mechanism:

Shared Network Namespace . Within a pod, all containers operate within a shared network namespace, allowing them to establish communication via the localhost interface.

For example, consider the scenario displayed in figure 6-9 where a pod houses two containers — one listening on port 8001 and the other on port 8002. In this setup, Container 1 has the capability to interact with Container 2 seamlessly using the localhost address, specifically on port 8002.

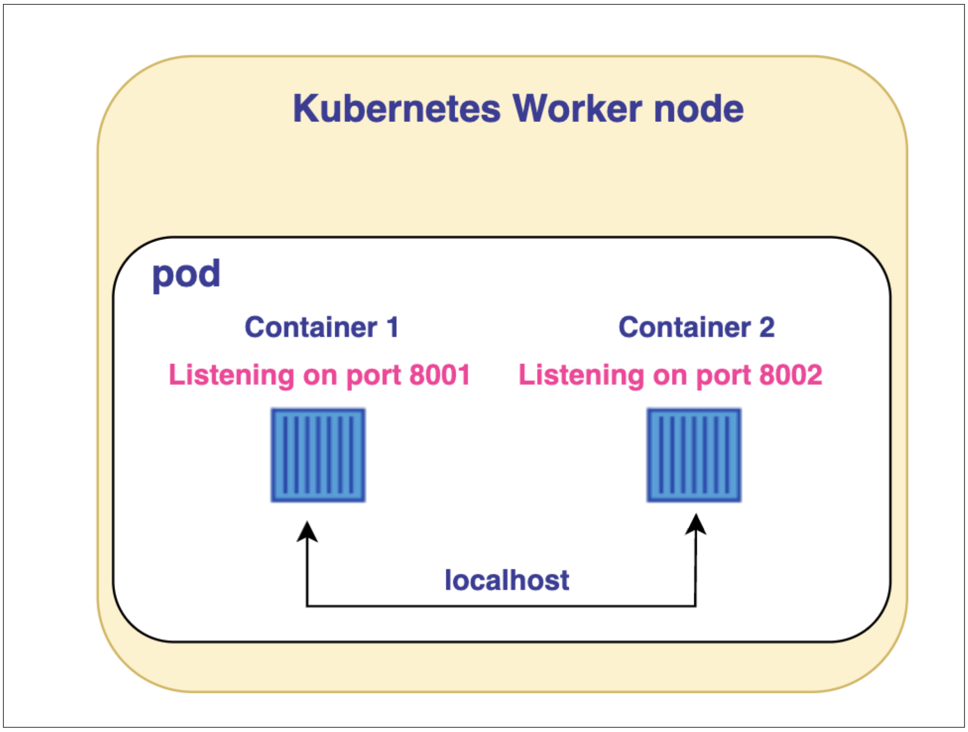


Figure 6-10. Multi-Container Pod Networking

Note:For detailed information see *https://kubernetes.io/docs/concepts/workloads/pods/#pod-networking*

Shared Storage Volume. Within a pod, containers have the ability to share a common storage volume, enabling them to exchange information by reading and making alterations to files within this shared storage space (see figure 7-5).

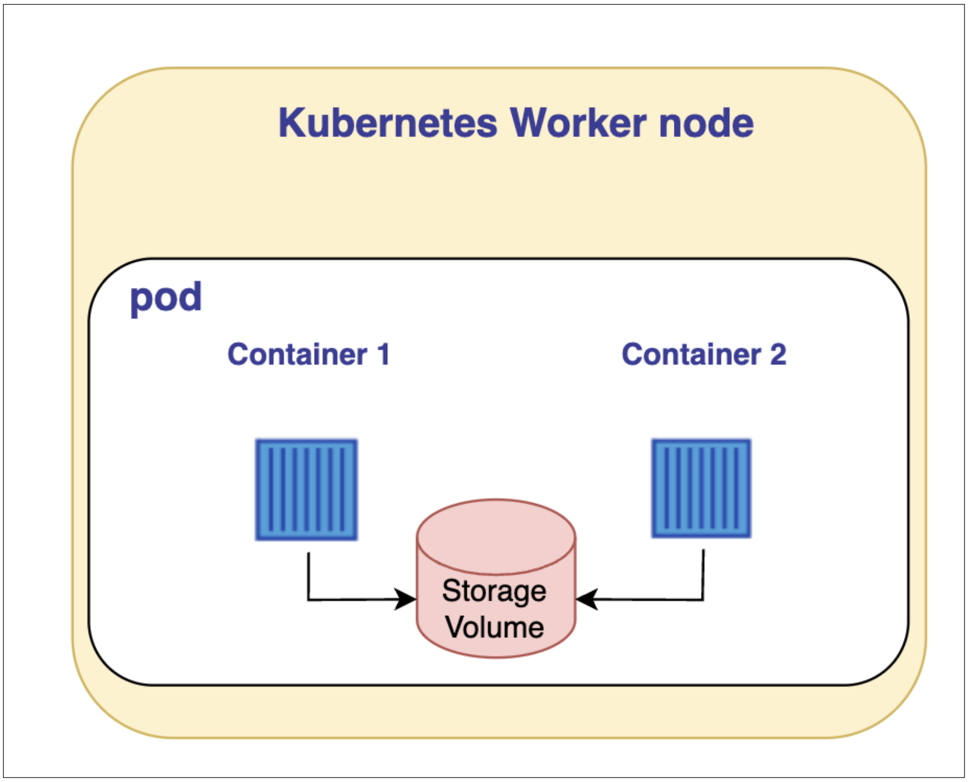


Figure 6-11. - 11: Multi-container Pod storage shared sample

Shared process Namespace. An alternative method for containers to establish communication when belonging to the same Pod is through the Shared Process Namespace. When activated, processes in a container are visible to all other containers in the same pod and enables containers within the same pod to signal one another (for example sending a SIGHUP signal which is one of the set of Termination Signals used to tell a process to terminate). To activate this functionality, the shareProcessNamespace setting in the pod specification must be configured as 'true'.

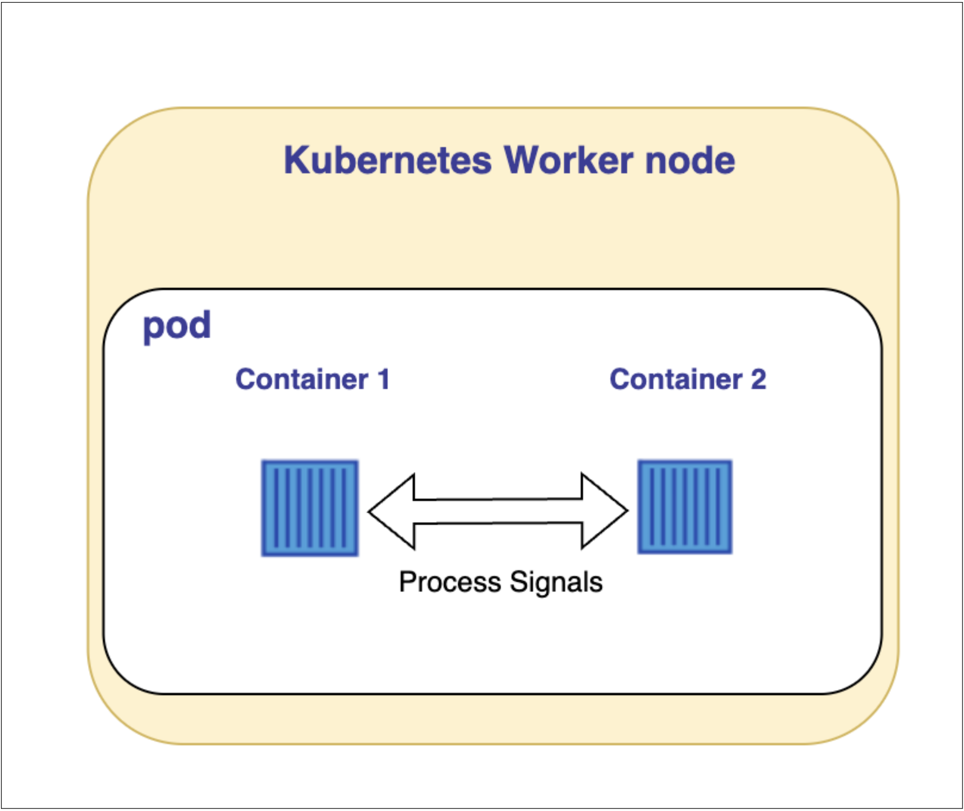


Figure 6-12. Multi-Container Pod Shared process namespace

Multi-container Architectural Patterns

The use of the multi-container pod is closely associated with three specific architectural patterns that offer structure and guidance when designing multi-container pods in Kubernetes. They promote modularity, maintainability, and flexibility in orchestrating complex applications and choosing the appropriate pattern depends on the specific application requirements.

We delve into these three primary patterns below and provide examples illustrating scenarios in which their application proves advantageous.

Sidecar Pattern. In this pattern, a *main container* **is accompanied by one or more** *sidecar containers* (see figure 6-13). The sidecar containers extend or enhance the functionality of the main container such as logging, monitoring or data synchronization.

For instance consider an NGINX web server container responsible for hosting a website. This web server container can be paired with a complementary "sidecar" container, which is designed to fetch website content from a GIT repository and serve it to users.

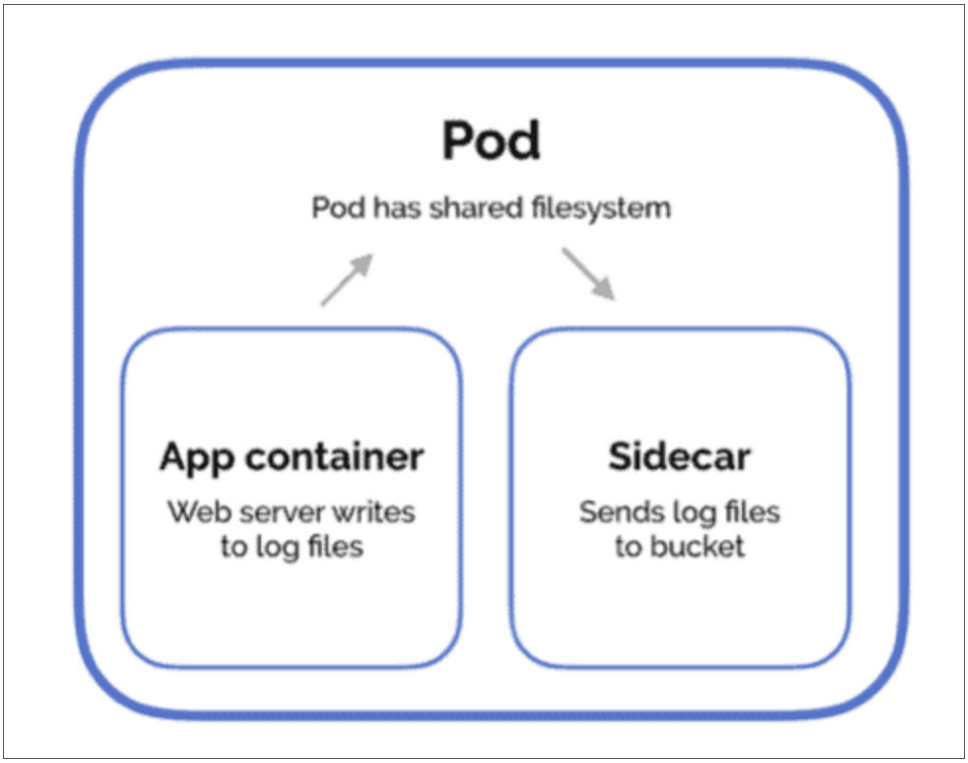

Pod
Pod has shared filesystem
App container
Web server writes to log files
Sidecar
Sends log files to bucket

Figure 6-13. Example of Sidecar container pattern

Adapter Pattern. In the adapter pattern the deployment involves placing a primary container alongside an adapter container. The adapter container assumes the responsibility of standardizing and normalizing the output generated by the primary container

Consider for example a main container that produces logs in different formats. The adapter container can convert these logs into a consistent format for centralized logging.

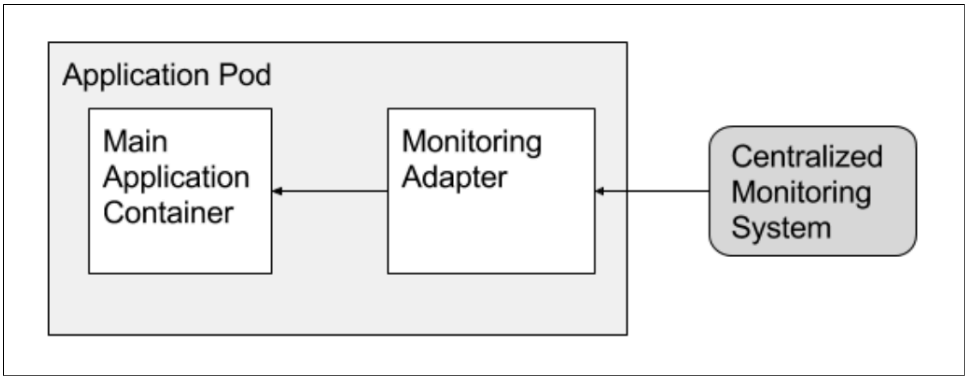


Figure 6-14. Example of Adapter Pattern

Ambassador Pattern. In this pattern, an **ambassador container** acts as a proxy or intermediary for the main container as shown in figure 6-15. It facilitates communication between the main container and external services.

One of the most common use cases for using this ambassador pattern is including a container responsible to handle authentication, load balancing, or SSL termination for the main application container.

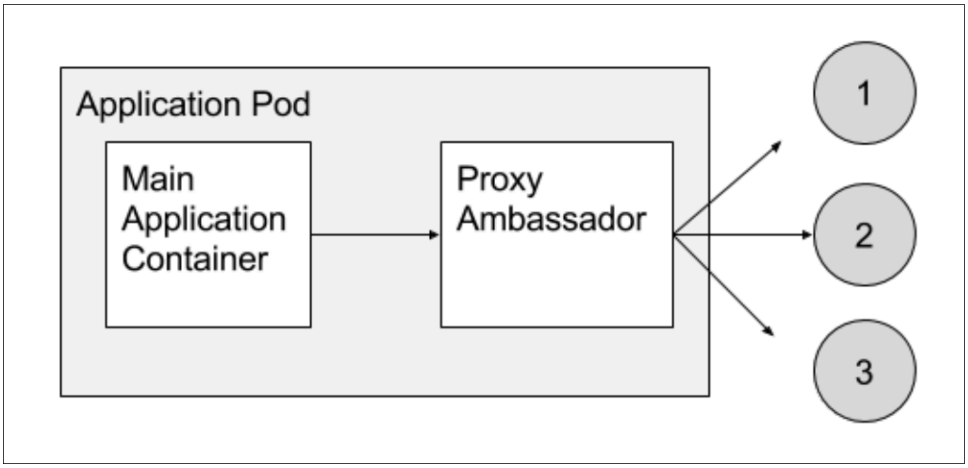


Figure 6-15. Example of Ambassador pattern

Keep in mind that although multi-container pods deviate from the 'one process per container' guideline, they present advantages such as the ability to reuse granular containers and facilitate seamless communication within the pod. Thus, whether dealing with closely integrated components or specialized functions, multi-container pods offer versatility and operational efficiency in Kubernetes deployments.

Note: About Multi-Container pods

Patterns for composite containers by Brendan Burns

We recommend reading the following post written by Brenda Burns in 2015:

https://kubernetes.io/blog/2015/06/the-distributed-system-toolkit-patterns/

In Kubernetes 1.28 a new feature to support sidecar containers natively in Kubernetes that ease the lifetime management of the sidecar container. At the time of writing this book it's still in alpha stage.

.

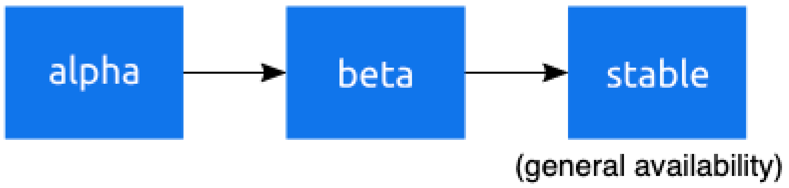


Figure 6-16. Kubernetes features lifecycle

https://kubernetes.io/blog/2023/08/25/native-sidecar-containers/

Container Runtimes in Kubernetes

Kubernetes accommodates diverse container runtimes, also commonly called container engines. As of the writing of this book, Kubernetes version 1.28 mandates that the runtime aligns with the Container Runtime Interface (CRI), a standardized interface which allows kubernetes to seamlessly support a diverse array of container runtimes, all without the cumbersome need for recompilation. In short, the CRI is a plugin interface which enable kubelet to use a variety of container runtimes.

https://kubernetes.io/docs/concepts/architecture/cri/

A container runtime is a software component responsible for executing and managing containers. It serves as the underlying engine that allows containers to run on a host system. In the context of Kubernetes, the choice of container runtime plays a pivotal role in the orchestration and execution of containerized workloads.

Kubernetes supports several container runtimes that comply with the CRI, including:

- Containerd
- CRI-O
- Docker Engine
- Mirantis Container Runtime

Each container runtime offers its unique set of advantages, and within the Kubernetes community, there have been calls to expand support for a variety of runtimes. In the Kubernetes 1.15 release, the Container Runtime Interface (CRI) was initially introduced as an Alpha release. Since then, it has matured and made substantial contributions, significantly enhancing Kubernetes' capabilities in terms of flexibility and extensibility regarding container runtime support.

As you may recall from the previous chapter, we discussed a pivotal standard in the container ecosystem: the Open Container Initiative (OCI). This standard outlines how containers should operate, and one of its most renowned implementations is 'runc,' often referred to as a Low-Level Runtime.

Low-Level Runtime (Runc)

To get a clear idea of what Runc is , let's look at Docker's own explanation from 2015:

"Over the last 5 years Linux has gradually gained a collection of features which make this kind of abstraction possible. Windows, with its upcoming version 10, is adding similar features as well. Those individual features have esoteric names like "control groups", "namespaces", "seccomp", "capabilities", "apparmor" and so on. But collectively, they are known as "OS containers" or sometimes "lightweight virtualization"... Docker makes heavy use of these features and has become famous for it. Because "containers" are actually an array of complicated, sometimes arcane system features, we have integrated them into a unified low-level component which we simply call runC. And today we are spinning out runC as a standalone tool, to be used as plumbing by infrastructure plumbers everywhere. runC is a lightweight, portable container runtime. It includes all of the plumbing code used by Docker to interact with system features related to containers"

Source: *https://www.docker.com/blog/runc/*

At this moment you may be trying to figure out how each of these pieces (Container Runtime and Low-Level Runtime) fit into the Kubernetes architecture. The Figure provides a simplified view to understand where each piece fits in the Docker and Kubernetes stack.

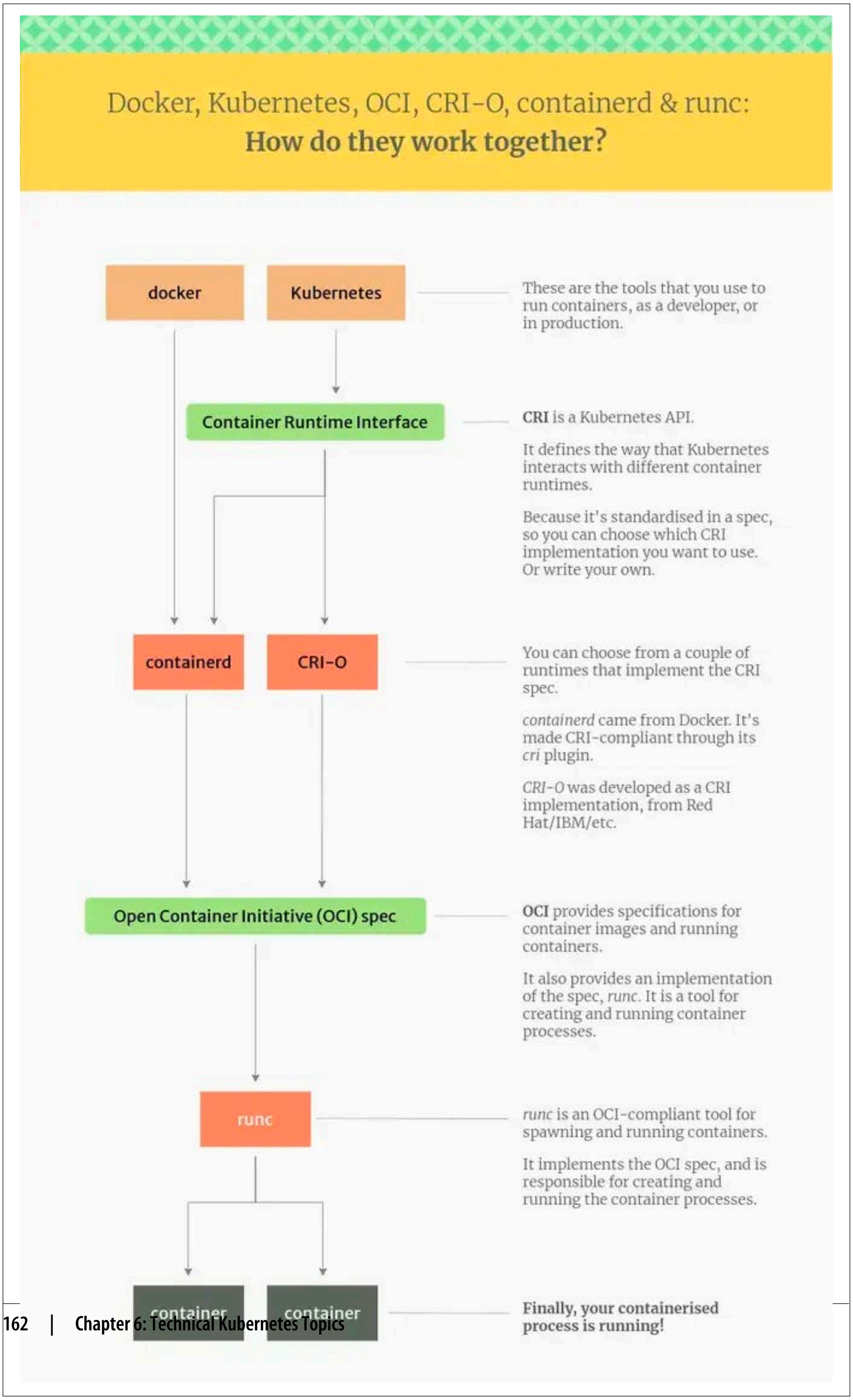

Docker, Kubernetes, OCI, CRI-O, containerd & runc:
How do they work together?
docker
Kubernetes
These are the tools that you use to run containers, as a developer, or in production.
Container Runtime Interface
CRI is a Kubernetes API.
It defines the way that Kubernetes interacts with different container runtimes.
Because it's standardised in a spec, so you can choose which CRI implementation you want to use. Or write your own.
containerd
CRI-O
You can choose from a couple of runtimes that implement the CRI spec.
containerd came from Docker. It's made CRI-compliant through its cri plugin.
CRI-O was developed as a CRI implementation, from Red Hat/IBM/etc.
Open Container Initiative (OCI) spec
OCI provides specifications for container images and running containers.
It also provides an implementation of the spec, runc. It is a tool for creating and running container processes.
runc
runc is an OCI-compliant tool for spawning and running containers.
It implements the OCI spec, and is responsible for creating and running the container processes.
container
container
Finally, your containerised process is running!

Figure 6-17.

Containerd and CRI-O stand as the most widely deployed container runtimes in Kubernetes. However, the choice between them falls beyond the scope of this book. If you're interested in delving deeper into these two runtimes, we recommend examining the differences in architecture and minimum hardware requirements for each. In general, for the vast majority of users, the selection between ContainerD and CRI-O may not be readily apparent, although it's worth noting that ContainerD was contributed by Docker to the CNCF (Cloud Native Computing Foundation), fostering ongoing collaboration and project evolution, while CRI-O is primarily supported by Red Hat.

Modifying the container runtime in Kubernetes is not a common occurrence, but thanks to the CRI specification, it is a straightforward operation.

Teams may consider changing the container runtime based on performance , security, compatibility or resource constraints reasons.

Container Network Interface

Container Network Interface is a specification that defines how container runtimes interact with network plugins. You can think of it as the bridge that connects containers and the network.

For a detailed information about Network plugins we recommend to read the official documentation referred in:

https://kubernetes.io/docs/concepts/extend-kubernetes/compute-storage-net/network-plugins/

Container storage Interface

Container Storage Interface is an initiative to unify the storage interface across container orchestrators (Kubernetes in our case) and storage vendors (Ceph, PortWorx, NetApp, etc).

The interface define the operations to allow dynamic provisioning and attaching and detaching storage volumes to containers .

More detailed information in the CSI project repository: *https://kubernetes-csi.github.io/docs/*

Navigating Kubernetes states

In the previous section we went through the main architecture pieces over kubernetes is build up. Now you are equipped to navigate to the functional aspects of Kubernetes and specifically in this section, the states and lifecycle management of applications (which ultimately run as Pod).

In the Kubernetes' domain, managing the lifecycle of containerized applications across a distributed cluster of nodes involves intricate coordination. At the heart of this orchestration lies the pivotal concept of application states. These states, which can be broadly categorized into two distinct types—stateless and stateful—play a foundational role in how Kubernetes efficiently deploys, scales, and maintains applications through controllers.

Furthermore, Kubernetes introduces a critical resource known as ReplicaSets, and building upon this foundation, it offers even more sophisticated controllers like Deployments and StatefulSets.

Let's explore these states and the role played by ReplicaSets, Deployments, and StatefulSets, DaemonSet and Job/CronJob Controllers

Stateless Applications

Stateless applications are the essence of agility in the Kubernetes landscape. They operate seamlessly without reliance on maintaining persistent data or state information on the local node. Stateless applications are designed to excel in a dynamic and ever-changing environment, where horizontal scaling and resilience are top priorities.

Consider typical stateless applications like web servers, microservices, and load balancers. These software components excel at rapidly generating new instances, efficiently handling traffic, and accommodating a surge in user activity. Stateless applications are usually easier to deploy, and resilient in the face of node failures.

Stateful Applications

In contrast to their stateless counterparts, stateful applications bear the weight of responsibility for managing data and state information. These applications are tied to the data they oversee, and their deployment requires measures to ensure data persistence, consistency, and availability during scaling or failure scenarios.

Kubernetes addresses the unique needs of stateful applications through the introduction of StatefulSets, a specialized controller that orchestrates the deployment and scaling of stateful workloads.

Stateful applications span a broad spectrum including databases, message queues, and applications reliant on unique network identifiers.

Deployments: Orchestrating Stateless Scalability

A Deployment, a higher-level controller, provides a declarative approach to managing the rollout and scaling of stateless applications.

Deployments offer a powerful and user-friendly way to ensure the desired number of replicas is maintained for stateless applications. They not only facilitate the horizontal scaling of containers but also allow for easy updates and rollbacks without downtime. By defining the desired state of the application, Deployments handle the process of creating and managing ReplicaSets, which in turn ensure the specified number of replicas is maintained.

StatefulSets: The custodian of stateful applications

StatefulSets, as a specialized controller within Kubernetes, provide stable network identifiers, ensuring that each replica of a stateful application maintains a consistent identity. Moreover, StatefulSets offer the gift of persistent storage, enabling data to survive the life cycle of a Pod and persist even in the face of node disruptions.

This combination of identity and persistence makes StatefulSets the go-to choice for managing databases, message queues, and other data-intensive workloads.

ReplicaSets: Safeguarding Replication and Reliability

ReplicaSets serve as vigilant guardians for both stateless and stateful applications. These resource controllers are indispensable for maintaining the optimal replication and reliability of applications. Whether it's a stateless application relying on Deployments, or a stateful application managed by StatefulSets, ReplicaSets play a pivotal role in ensuring that the desired number of replicas is maintained, thus facilitating scalability and fault tolerance.

DaemonSet Controller

DaemonSet Controller ensures that a specified Pod runs on every node within the Kubernetes cluster. Unlike Deployments or ReplicaSets that focus on keeping a given amount of replicas across the cluster (for instance imagine a cluster with 5 worker nodes and a pod which is declared to run with a ReplicaSet=3. The ReplicaSet controller will ensure that there are 3 replicas running across the cluster but it does not mean at all that every node will run a replica) DaeonSets focus is to ensure that the specified Pod runs in every node. You can rely on properties such as Node Affinity and Tolerations to define in which specific nodes the Pod specified as Daemonset will run on.

Job and CronJob Controllers

First let's define what a job means in Kiubernetes realm.

A Job represents a single, self-contained task or batch job that don't need to run all the time. It's a good fit for one-time tasks (for instance a data import into a DB) or running a batch (making a backup, generating reports, etc..)

The Job Controller will ensure that the task completes successfully, creates the Pods required for running the job and in case of a fail, it will retries the task.

A CronJob is an scheduled task in Kubernetes which need to run periodically (hourly cleanup, daily backups, etc) and uses a cron-like syntax to define the schedules

Note: Reference material recommended

https://kubernetes.io/docs/concepts/workloads/controllers/

Workload Auto-scaling, Kubernetes Events and Pod lifecycle

To comprehend Kubernetes events, it's essential to grasp the underlying dynamics of resource allocation and container management within an active application.

In the real world, applications are never used at a constant rate. User activity ebbs and flows, influenced by factors like time of day or specific events (Black Friday for instance). CPU, memory, and networking demands fluctuate throughout runtime. Here, the Kubernetes scheduler (remember the architecture Figure 6-7) takes center stage, dynamically distributing computing and memory resources to cater to application's evolving needs. Kubernetes provides a variety of mechanism which allow scaling operations:

Horizontal Pod Autoscaler (HPA):

It automatically updates a workload resource (Deployment or statefuSet) for automatically scaling the workload to match the demand Horizontal Scale means that as response for more load, more Pods will be deployed. The HPA can be actioned based on observing metrics such as average CPU utilization, average memory utilization, or any other custom metric you specify.

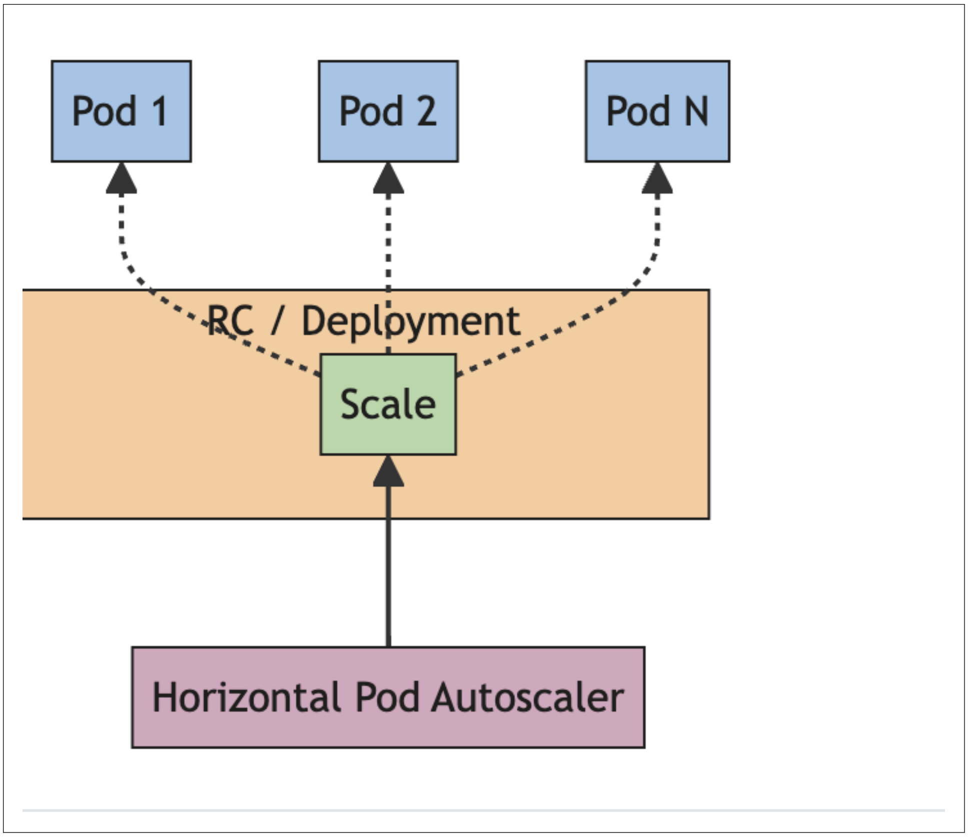


Figure 6-18. Horizontal Pod Autoscaler

(source: *https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/*)

Vertical Pod Autoscaler (VPA):

Vertical pod Autoscaler provides the mechanism to adjust the pod resources requirements (mainly CPU and memory reservations) based on their historical resource usage. It can scale vertically up (if more resources are needed) or down (if the pod requirements are over reserved).

Unlike HPA, VPA doesn't come with Kubernetes by default. It's a separate project that needs to be installed.

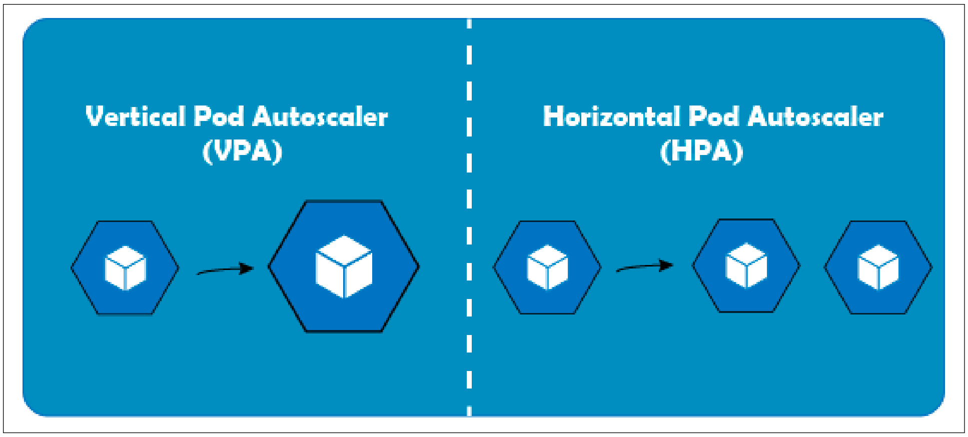

Vertical Pod Autoscaler (VPA)
Horizontal Pod Autoscaler (HPA)

Figure 6-19. Represent differences between VPA and HPA

The VPA project is found in the following repository: *https://github.com/kubernetes/autoscaler/tree/master/vertical-pod-autoscaler*

Cluster Proportional Autoscaler (CPA):

There can be workloads that require to scale based on the cluster size as for example core system components (CoreDNS is a typical workload that fits for using CPA) where based in the number of nods or pods in the cluster, CPA dynamically adjusts the number of replicas required for the workload.

Same as VPA, CPA is not included in Kuberentes by default but in their GitHub repository:

https://github.com/kubernetes-sigs/cluster-proportional-autoscaler

Autoscaling based on Events

Another Auto scaling mechanism that can be required for your workloads is when decisions for scaling up/down needs to be triggered by an event. For example imaging an application that processes images when a new image is stored in a queue. In the event of a high number of images in the queue waiting to be processed you might want to scale out the number of pod replicas to process the images faster and once the number of images in the queue is low, scale in for freeing up resources (or reducing costs). That's a simple example but it can be used for any event-driven workload.

The project that allows a Kubernetes cluster implement autoscaling based in events is KEDA (Kubernetes Event-Driven Autoscaler).

Cluster Autoscaler

We have seen how to auto scale pods depending on a variety of situations. However there is another scenario that you can find when working with distributed systems and applications. Imagine that you have configured your HPA for scaling dynamically (scale out and scale in) based on the requests coming to your service. HPA will evaluate the target metrics and if it matches the configuration you specified in your workload deployment, HPA will scale out your application by increasing the number of replicas (pod replicas). Now you can rest because Kubernetes HPA will manage it for you.

But, what happens if the HPA update the deployment and ReplicaSet number for new pods being deployed but there are no free resources in your worker nodes that can accommodate the new replicas?

Well that is a very different scenario….. Now you will need to horizontally scale (Scale Out) oyour cluster so you have resources available to accommodate the new Pods.

That's exactly what Cluster Autoscaler is used for and it allows you to dynamically adjust the cluster size (out and in) based on the workload requirements.

For more information about Cluster Autoscaler we recommend to read the official information in the corresponding GitHub repository:

https://github.com/kubernetes/autoscaler

Note: CoreDNS and KEDA

CoreDns is the default DNS service for Kubernetes. It's a graduated project under the CNCF and it's build as a chain of plugins so you can only use the plugins for the functionality you really need. Between other functions it can provide service discovery. The configuration is done through the "Corefile" where you define the server configuration and the plugins loaded.

For detailed information about CoreDNS you can read the official site:

https://coredns.io/

Keda (Kubernetes event-driven autoscaler) is another graduated project under the CNF. KEDA enhances Kubernetes by providing better scaling options for event-driven applications with a catalog of 50+ built-in scalers for various cloud platforms, databases, messaging systems, telemetry systems, CI/CD and more.

For detailed information you can read the official site:

https://keda.sh/

Note: Node Affinity , tolerations and taints.

Node affinity, taints, and tolerations are Kubernetes concepts that help to schedule pods onto appropriate nodes.

Node affinity: is a property of Pods that attracts them to specific nodes (either as a preference or a hard requirement). For example you can ensure that Pods are scheduled on nodes with GPU available or in an specific availability zone if using an hyperscaler infrastructure.

Tains and tolerations works together to ensure that Pods are not scheduled into inappropriate nodes.

Taings are applied to nodes and allow you marking nodes that should not accept any Pods that do not tolerate the tains.

Tolerations however are applied to Pods and allow the scheduler to place Pods with marching taints.

Taints and tolerations are useful for scenarios where critical workloads need to run in a reserved bunch of nodes.

If a Pod has a matching toleration, it can be scheduled on a tainted node.

More information:

https://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration/

Assign Pods to Nodes using Node Affinity | Kubernetes

Now that you know what are the scenarios for scaling distributed applications and the austoscalers available, it's time to unfold the functionality of the Scheduler.

Kubernetes scheduler is a control plane component responsible for assigning Pods to Nodes. Behind the scenes the Pods that need to be scheduled are added to a queu and the scheduler continuously select Pods from the queue and assigns them to suitable Nodes based on constraints and available resources.

Scheduler actions may involve terminating nodes or pausing pods linked to particular applications. The resources reclaimed are either reassigned to other applications or held in reserve until demand resurfaces.

Resource reallocation can therefore be intentional or unforeseen as for instance sudden performance shifts can trigger node failures or even the removal of nodes from the cluster. These events may also lead to disruptions, such as pod evictions, kernel panics, or the deletion of virtual machines.

Responding effectively to these events is paramount. Equally critical is gaining insight into the underlying causes behind specific application behaviors and in this scenario is where Kubernetes event objects play a pivotal role in providing context. Let's delve deeper into how these events fit into the overall picture.

In the Kubernetes ecosystem, an event serves as an automated record created in response to changes in resources, such as nodes, pods, or containers. At the core of these events are alterations in state and Kubernetes events serve as the watchful eyes that monitor and capture key milestones and transitions. For instance, transitions in a pod's lifecycle or reallocation processes and scheduling may also generate events (see figure 6-20).

Understanding the Pod lifecycle is fundamental for managing and troubleshooting applications in Kubernetes as it enables us to monitor the status of the workloads, handle errors gracefully, and ensure efficient resource utilization within the cluster.

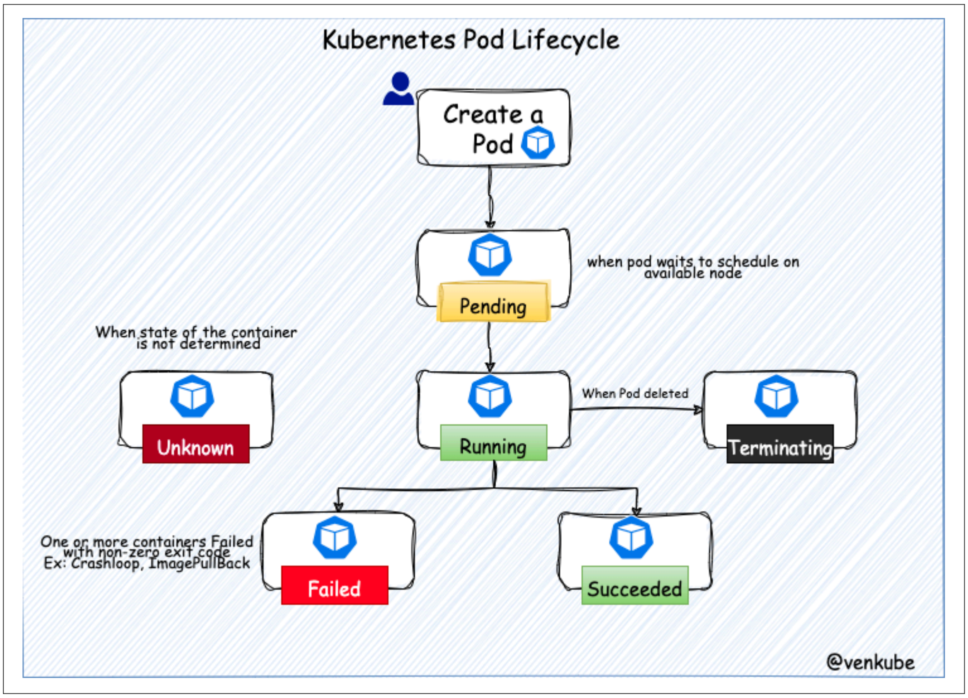

Kubernetes Pod Lifecycle
Create a Pod
Pending
when pod waits to schedule on available node
When state of the container is not determined
Unknown
Running
When Pod deleted
Terminating
One or more containers Failed with non-zero exit code Ex: Crashloop, ImagePullBack
Failed
Succeeded
@venkube

Figure 6-20.

The journey of a Pod begins with its creation. This typically occurs when an application or workload is deployed to the cluster. A Pod specification is defined, detailing the containers, their images, and resource requirements. When this specification is submitted to the Kubernetes API server, the Pod enters the "Pending" phase.

Pending Phase

During the Pending phase, Kubernetes orchestrates the allocation of resources required by the Pod. It seeks an appropriate node in the cluster to host the Pod. This node should satisfy the Pod's resource requests and affinity rules while respecting any anti-affinity constraints. Once a suitable node is found, the Pod transitions to the "Scheduled" phase.

Scheduled Phase

In the Scheduled phase, the Kubernetes scheduler successfully assigns the Pod to a node. However, the Pod is not yet running on the node. Kubernetes now instructs the Kubelet on the chosen node to initiate the container(s) defined within the Pod specification.

Running Phase

With the Kubelet's action, the Pod enters the Running phase. In this state, the containers within the Pod are actively executing, serving your application or workload. The Pod remains in the Running phase until it completes its task or encounters an issue.

Succeeded or Failed Phase

When a Pod's containers complete their intended tasks without errors, the Pod transitions to the "Succeeded" phase. Conversely, if any container within the Pod encounters an unrecoverable error, the Pod moves to the "Failed" phase. In both cases, the Pod's primary purpose is fulfilled.

Termination

The last phase of a Pod's lifecycle is Termination. During this phase, the Pod is gracefully shut down. Resources are released, and the Pod is removed from the cluster. The termination process may involve cleaning up any attached storage volumes and releasing network resources. Once completed, the Pod ceases to exist within the cluster.

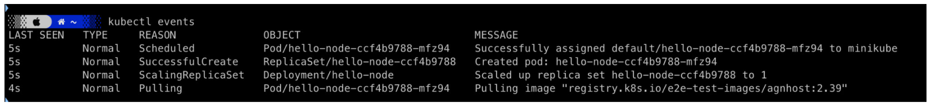
If the Pod cannot be gracefully shutdown it is forcefully terminated but it can lead to Data Loss (if the applicacion inside the Pod has open connections or unfinished transactions).

As outlined earlier in the chapter, a Pod comprises one or more containers, and Kubernetes not only monitors the overall phase of the Pod but also keeps a close watch on the state of each individual container within the Pod. These states include the following:

- Running: When a container is in the ‘Running’ state, it is actively executing its assigned tasks and is functioning as expected within the Pod. This is the desired state for containers, indicating their health.
- Terminated: The ‘Terminated’ state is reached when a container has completed its tasks and exited successfully or when it encounters an error or issue that leads to an unexpected termination. Containers in this state may require investigation to determine the cause of termination.
- Waiting: The ‘Waiting’ state indicates that a container is not yet in the ‘Running’ state and is currently waiting for specific conditions to be met, such as the availability of required resources or dependencies. Containers in this state are in a pending state, awaiting the transition to ‘Running.’
- ImagePullBackOff: When a container repeatedly fails to pull its required container image, it enters the ‘ImagePullBackOff’ state. This status suggests that there may be issues with image availability or authentication, which need to be resolved to enable the container to start successfully.
- CrashLoopBackOff: The ‘CrashLoopBackOff’ status occurs when a container continuously crashes shortly after starting. It indicates a recurring problem within the container, such as misconfigurations, unhandled errors, or insufficient resources, requiring troubleshooting to resolve the underlying issue.

Bsically we can safely say that whenever something happens inside the cluster, it produces an event object that provides visibility. However, Kubernetes events don’t persist throughout the cluster life cycle, as there’s no mechanism for retention. They’re short-lived, only available for one hour after the event is generated.

The simplest way to view event objects is using “kubectl events “



```
 ~ kubectl events
LAST SEEN   TYPE     REASON              OBJECT                             MESSAGE
5s          Normal   Scheduled           Pod/hello-node-ccf4b9788-mfz94     Successfully assigned default/hello-node-ccf4b9788-mfz94 to minikube
5s          Normal   SuccessfulCreate    ReplicaSet/hello-node-ccf4b9788    Created pod: hello-node-ccf4b9788-mfz94
5s          Normal   ScalingReplicaSet   Deployment/hello-node              Scaled up replica set hello-node-ccf4b9788 to 1
4s          Normal   Pulling             Pod/hello-node-ccf4b9788-mfz94     Pulling image "registry.k8s.io/e2e-test-images/agnhost:2.39"
```

Figure 6-21. Example of a kubectl events

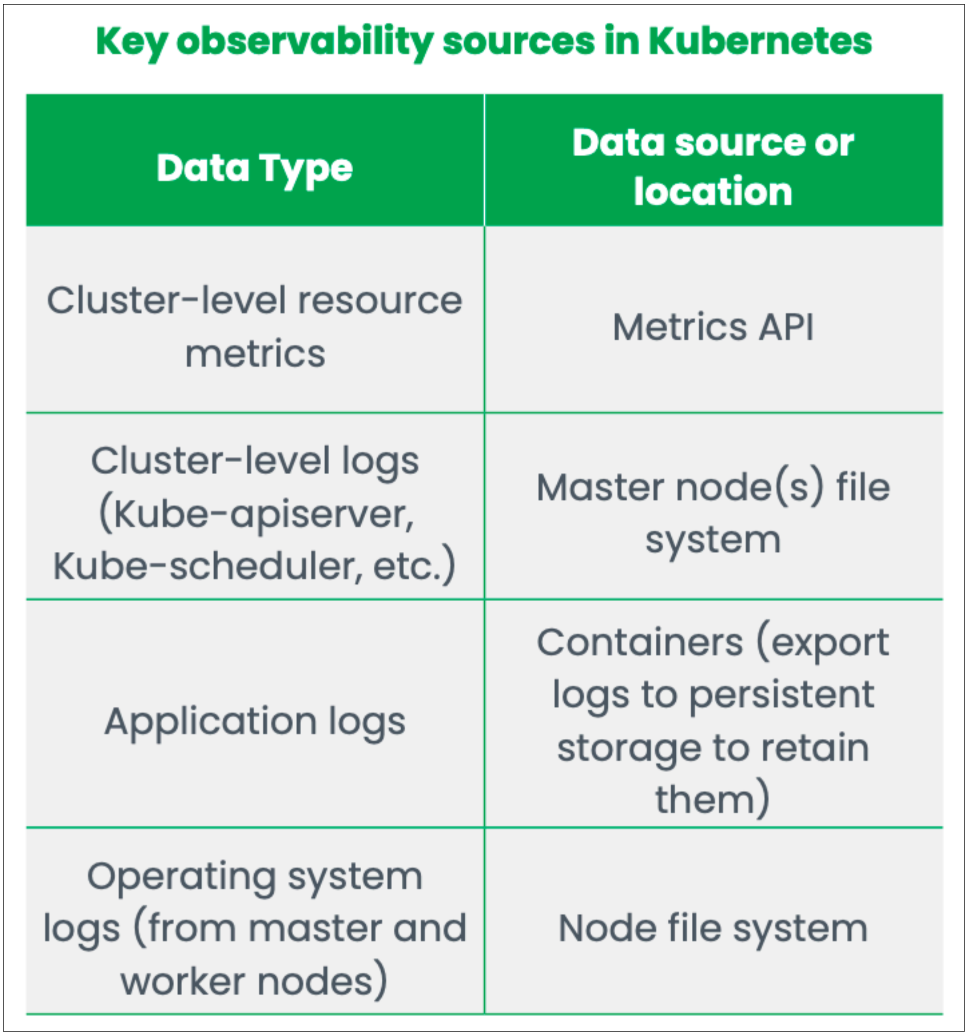
Kubernetes Observability and Performance

Kubernetes, with its dynamic nature and rapid scalability, presents unique challenges when it comes to understanding what’s happening under the hood. Observability should allow you to gain insights into your cluster’s behavior that help you detect and diagnose issues effectively.

Observability generally refers to the ability to gain insights into the behavior and performance of applications and systems. It involces collecting system telemetry data such as logs, metrics and traces. Let's explore this concept in Kubernetes further:

- Logs: Logs are textual records which can be generated from applications, services and Kubernetes system components. In Kubernetes, collecting logs from containers, and pods will help you for understanding the sequence of actions, debugging issues, and tracing the flow of data and requests.
- Metrics: Metrics provide a quantitative data of the system's performance, offering numerical data on aspects such as CPU usage, memory consumption, network activity, and the health of individual components. Metrics are essential for tracking trends, identifying anomalies, and optimizing resource allocation.
- Traces: Traces, particularly distributed traces, follow the flow of requests across different services allowing you to identify bottlenecks, service dependencies and performance issues from a specific request as they traverse through various microservices within the Kubernetes cluster.

Kubernetes clusters are multi-layered and dynamic (Pods and its containers can be added or removed dynamically) and it does not expose logs, metrics and traces the same way as conventional systems and achieving observability in Kubernetes involves stitching different data sources.


Key observability sources in Kubernetes

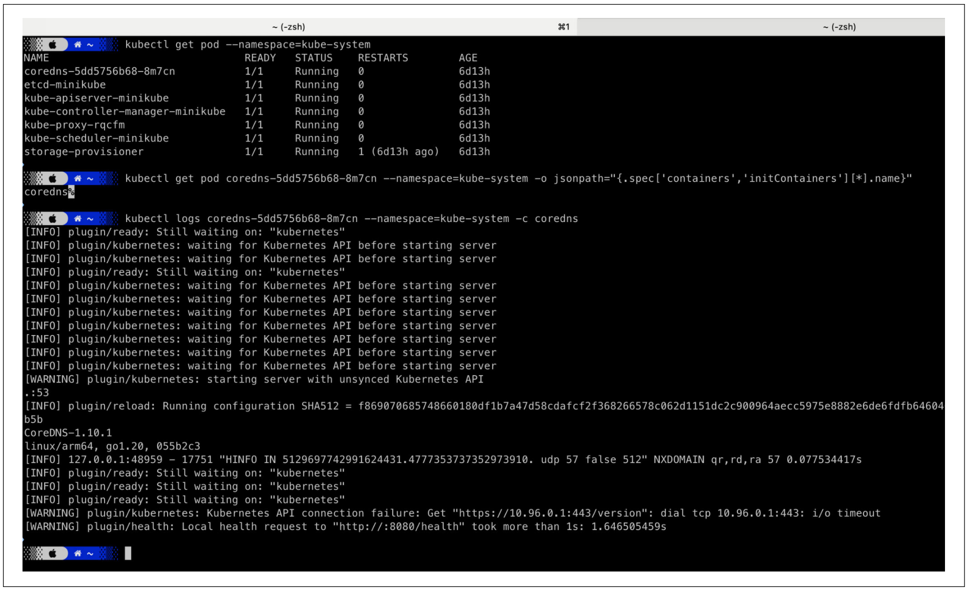
Data Type	Data source or location
Cluster-level resource metrics	Metrics API
Cluster-level logs (Kube-apiserver, Kube-scheduler, etc.)	Master node(s) file system
Application logs	Containers (export logs to persistent storage to retain them)
Operating system logs (from master and worker nodes)	Node file system

Figure 6-22. Observability sources in Kubernetes

Let's see how it works :

Logs:

Containers within pods generate logs which are stored within the pods themselves. When you need to access them, you have to interact directly with the pod using kubectl for instance (Kubectl logs) and unlike traditional systems where logs are generally centralized, Kubernetes distributes them across pods.



```
 kubectl get pod --namespace=kube-system
NAME                               READY   STATUS    RESTARTS        AGE
coredns-5dd5756b68-8m7cn           1/1     Running   0               6d13h
etcd-minikube                      1/1     Running   0               6d13h
kube-apiserver-minikube            1/1     Running   0               6d13h
kube-controller-manager-minikube   1/1     Running   0               6d13h
kube-proxy-rqcfm                   1/1     Running   0               6d13h
kube-scheduler-minikube            1/1     Running   0               6d13h
storage-provisioner                1/1     Running   1 (6d13h ago)   6d13h

 kubectl get pod coredns-5dd5756b68-8m7cn --namespace=kube-system -o jsonpath="{.spec['containers','initContainers'][*].name}"
coredns%

 kubectl logs coredns-5dd5756b68-8m7cn --namespace=kube-system -c coredns
[INFO] plugin/ready: Still waiting on: "kubernetes"
[INFO] plugin/kubernetes: waiting for Kubernetes API before starting server
[INFO] plugin/kubernetes: waiting for Kubernetes API before starting server
[INFO] plugin/ready: Still waiting on: "kubernetes"
[INFO] plugin/kubernetes: waiting for Kubernetes API before starting server
[INFO] plugin/kubernetes: waiting for Kubernetes API before starting server
[INFO] plugin/kubernetes: waiting for Kubernetes API before starting server
[INFO] plugin/kubernetes: waiting for Kubernetes API before starting server
[INFO] plugin/kubernetes: waiting for Kubernetes API before starting server
[INFO] plugin/kubernetes: waiting for Kubernetes API before starting server
[INFO] plugin/kubernetes: waiting for Kubernetes API before starting server
[WARNING] plugin/kubernetes: starting server with unsynced Kubernetes API
.:53
[INFO] plugin/reload: Running configuration SHA512 = f869070685748660180df1b7a47d58cdafcf2f368266578c062d1151dc2c900964aecc5975e8882e6de6fdfb64604
b5b
CoreDNS-1.10.1
linux/arm64, go1.20, 055b2c3
[INFO] 127.0.0.1:48959 - 17751 "HINFO IN 5129697742991624431.4777353737352973910. udp 57 false 512" NXDOMAIN qr,rd,ra 57 0.077534417s
[INFO] plugin/ready: Still waiting on: "kubernetes"
[INFO] plugin/ready: Still waiting on: "kubernetes"
[INFO] plugin/ready: Still waiting on: "kubernetes"
[WARNING] plugin/kubernetes: Kubernetes API connection failure: Get "https://10.96.0.1:443/version": dial tcp 10.96.0.1:443: i/o timeout
[WARNING] plugin/health: Local health request to "http://:8080/health" took more than 1s: 1.646505459s
```

Figure 6-23. Example showing how to get the logs from a container in a pod

In the example from Figure 6-23 the flow followed is:

1. We list all the pods running in the namespace called kube-system

Kube-system namespace is a namespace reserved for objects created by Kubernetes itself. In this specific namespace you will find critical pods which are responsible for the cluster's operation.

2.- We list the containers associated to the pod coredns-5dd5756b68-8m7cn. This pod only contains one container whose name is "coredns"

3.- We list the logs for the container coredns in the pod coredns-5dd5756b68-8m7cn

Metrics:

Kubernetes exposes metrics throught the Kubelet but collecting and aggregating these metrics across the entire cluster requires additional tools . Prometheus (collecting metrics) and Grafana (visualizing the metrics) are commonly used for monitoring Kubernetes metrics as Kubernetes natively exposes metrics in Prometheus format.

More detailed information:

https://kubernetes.io/docs/concepts/cluster-administration/system-metrics/

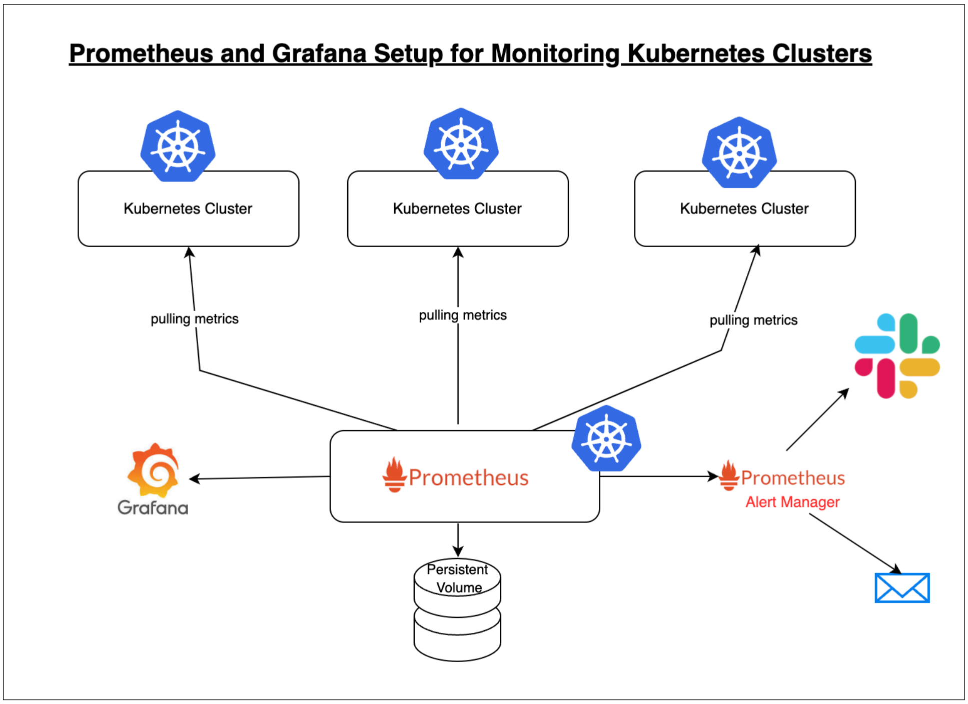


Figure 6-24. Prometheus and Grafana Setup (Source https://www.cidevops.com/2022/05/how-to-setup-monitoring-on-kubernetes.html#google_vignette

For the Horizontal and Vertical Pod Autoscaler based on CPU/memory metrics metrics-server collects resource usage data from Kubelets and exposes this information through the Kubernetes API server via the Metrics API (we will cover it in the following section as part of the Resource Metrics Pipeline)

More information:

https://kubernetes.io/docs/tasks/debug/debug-cluster/resource-metrics-pipeline/#metrics-server

https://github.com/kubernetes-sigs/metrics-server

Traces:

Kubernetes doesnt provide natively tracing data so you will need to integrate additional tools (such as Jaeger and Open Telemetry) to capture traces. OpenTelemetry and Jaeger will let you implement a distributing tracing system.

Open Telemetry is focused on instrumenting your applications so you can capture trace data from them and Jaeger complements Open Telemetry by providing powerful trace analysis capabilities with a UI for analyzing the traces.

Both are open-source projects under the CNCF umbrella where Jaeger is in graduated stage and Open Telemetry is in incubation stage.

More information:

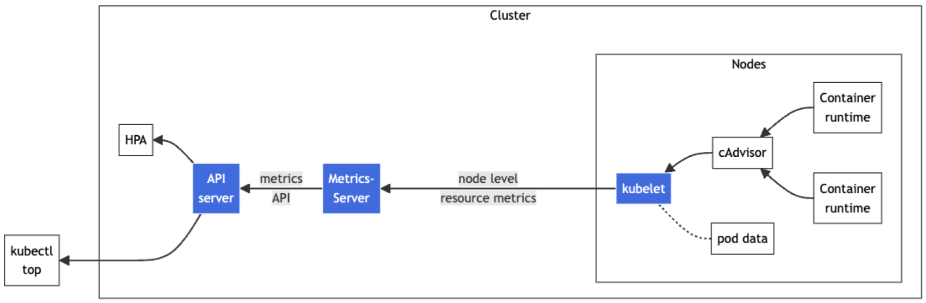
https://www.jaegertracing.io/

https://opentelemetry.io/

In summary , Kubernetes monitoring stack is not constrained by a single monitoring solution. Instead, it offers flexibility by accommodating different approaches to gather monitoring statistics, including resource metrics pipeline and full metrics pipeline, discussed below.

Resource Metrics Pipeline

The resource metrics pipeline (*https://kubernetes.io/docs/tasks/debug/debug-cluster/resource-metrics-pipeline/*) offers a set of metrics related to cluster components, including the Horizontal Pod Autoscaler controller, and facilitates monitoring through tools like the kubectl top utility. These metrics are collected by the lightweight, short-term, in-memory metrics-server and are accessible via API.



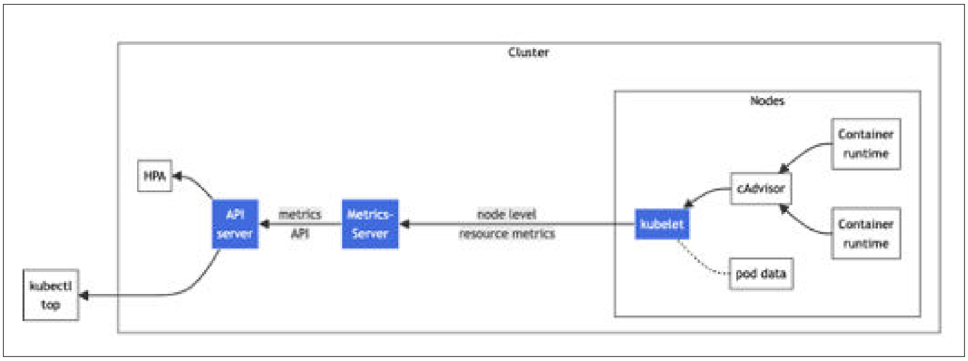


Figure 6-25. architecture of the resource metrics pipeline (source: https://kubernetes.io/docs/tasks/debug/debug-cluster/resource-metrics-pipeline/

The metrics-server dynamically discovers all nodes in the cluster and queries each node's kubelet for real-time CPU and memory usage data. The kubelet, acting as an intermediary between the Kubernetes master and the nodes, oversees the management of pods and containers on a given machine. It translates each pod into its constituent containers and retrieves individual container usage statistics from the container runtime through the container runtime interface. The component cAdvisor is a daemon for collecting, aggregating and exposing container metrics included in Kubelet.

The Kubelet subsequently exposes aggregated pod resource usage statistics through the metrics-server Resource Metrics API, typically served at /metrics/resource/v1beta1 .

Full Metrics Pipeline

For a more comprehensive monitoring approach, Kubernetes offers a full metrics pipeline (*https://kubernetes.io/docs/tasks/debug/debug-cluster/resource-usage-monitoring/#full-metrics-pipeline*) , which grants access to richer metrics data. These metrics enable Kubernetes to respond dynamically by automatically scaling or adapting the cluster based on its current state. This is achieved through mechanisms like the Horizontal Pod Autoscaler. The monitoring pipeline retrieves metrics from the kubelet and exposes them to Kubernetes through an adapter, often implementing either the custom.metrics.k8s.io or external.metrics.k8s.io API

Note: Custom and External metrics

The custom.metrics.k8s.io API is an extension of the Kubernetes API which allows you to define custom metrics for your workloads and is usually used for Horizontal Pod Autoscaler purposes.

Unlike custom.metrics.k8s.io the external.metrics.k8s.io API also provides customer metrics but providing metrics that are external to the cluster usually coming from third-party systems (database connections, API responde time, etc) and are used for autoscaling or other requirements within your application.

To understand the differenc imagine you have an ecommerce website which interacts with an external payment gateway. You ecommerce needs to scale some microservices based on the response time of the payment gateway.

How could you do it? In this cases, you can expose this metric via the external.metrics.k8s.io API and then configure HPA to use it for scaling.

https://kubernetes.io/docs/reference/external-api/custom-metrics.v1beta2/

Note: Kubernetes API

The Kubernetes API is a resource-based RESTful API that supports CRUD operations via HTTP verbs (GET, POST, PUT, PATCH and DELETE). Kubernetes API terminology can be summarized as:

REsource Type: Name used in the URL (pod, namespace, service)

Kind: Every resource Type has its own schema that defines their properties. It's commonly called "kind"

Resource: A single instance of a resource tyep

Collection: A list of instances of a resource type. For some resource types, the API includes one or more sub-resources, which are represented as URI paths below the resource.

Kubernetes objects are the building blocks of your cluster. They represent various aspects of the cluster's state.

Representation in the API:

Kubernetes objects are persistent entities within the Kubernetes system.

They are represented in the Kubernetes API.

Examples include Pods, Services, ConfigMaps, and ReplicaSets (*https://kubernetes.io/docs/concepts/overview/working-with-objects/*)

YAML Format:

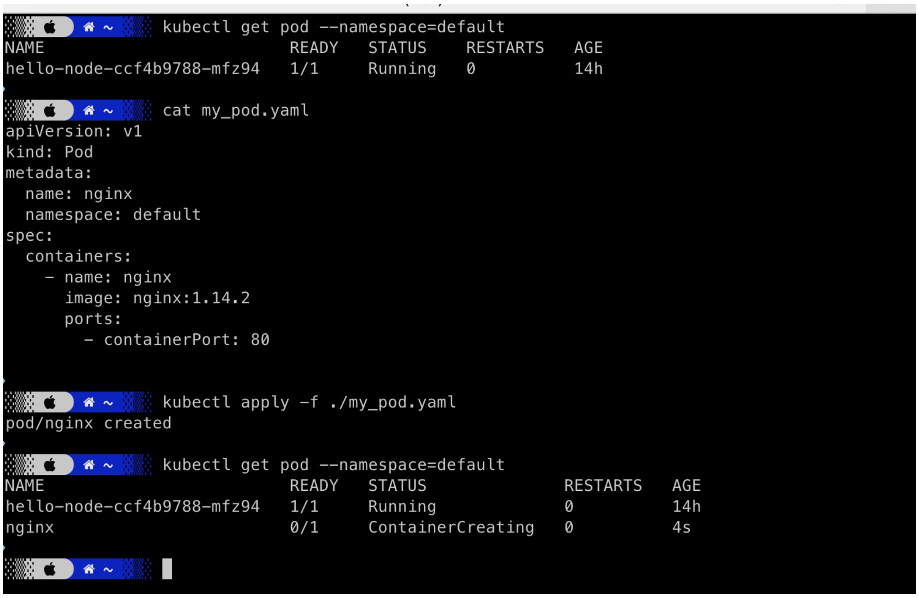
You express Kubernetes objects using YAML format.

This format defines the desired state of an object.

The Kubernetes API ensures the actual state matches the desired state (*https://kubernetes.io/docs/concepts/overview/working-with-objects/*)

An examples of creating a Kubernetes object is when you create a Pod.

Pods



```
 kubectl get pod --namespace=default
NAME                         READY   STATUS    RESTARTS   AGE
hello-node-ccf4b9788-mfz94   1/1     Running   0          14h

 cat my_pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  namespace: default
spec:
  containers:
    - name: nginx
      image: nginx:1.14.2
      ports:
        - containerPort: 80


 kubectl apply -f ./my_pod.yaml
pod/nginx created

 kubectl get pod --namespace=default
NAME                         READY   STATUS              RESTARTS   AGE
hello-node-ccf4b9788-mfz94   1/1     Running             0          14h
nginx                        0/1     ContainerCreating   0          4s
```

Figure 6-26. Example of creating a Pod object.

It is important to highlight that Kubernetes has been designed to seamlessly integrate with OpenMetrics (*https://openmetrics.io/*) , a project within the CNCF Observability and Analysis - Monitoring Projects category with the aim of extending the Prometheus exposition format, ensuring compatibility while offering a rich set of metrics transmission capabilities.

When exploring the CNCF Landscape (*https://landscape.cncf.io/*)you will encounter a multitude of monitoring projects that can effectively work with Kubernetes, collecting metric data and assisting in cluster observation. The choice of monitoring tools is vast, ranging from open-source solutions to paid software-as-a-service platforms and commercial products. Selecting the most suitable monitoring platform depends on a multitude of factors, including your specific requirements, budget constraints, and available technical resources. Kubernetes refrains from endorsing any particular metrics pipeline, recognizing the importance of tailoring the choice to align with your infrastructure platform's design and deployment.

Kubernetes storage

Kubernetes not only excels in managing containers as we have presented in the previous chapters but also offers robust solutions for handling storage in a platform where applications scale out and in constantly. In this section, we'll explore the world of Kubernetes storage, from the fundamental concepts to advanced storage strategies, ensuring your applications have access to reliable and scalable storage resources.

Let's dive into the main concepts around Kubernetes storage.

Persistent Volumes (PVs) and Persistent Volume Claims (PVC)

Persistent Volumes (PVs) and Persistent Volume Claims (PVCs) (*https://kubernetes.io/docs/concepts/storage/persistent-volumes/*) form the foundation of Kubernetes storage. In a scenario where you need to ensure your application's data persists even if the underlying pod is rescheduled or deleted persistent volumes are the solution. Basically they exist as separate entities at the cluster level (not tied to any specific pod unlike regular volumes which exist at the pod level). PVC on the other hand is another type of Kubernetes object that allows pods to request access to PVs i.e when a pod needs persistent storage it creates a PVC.

When creating a PVC you need to specify the size, access mode and the storage class (defines the underlying storage type: local storage, premium-storage, cloud-based storage, etc).

Note: Access Mode

There are three access mode available for a PVC:

ReadWriteOnce(RWO): Allows read and write access by a single pod.

ReadOnlyMany(ROX): Allows read access by multiple pods

ReadWriteMany(RWX): Allows read and write access by multiple pods (only supported in specific storage types such as CephFS and NFS)

A PVC cannot be created with more than one access mode property simultaneously because when creating a PVC you specify the desired access mode (either RWO, ROX or RWX)

Storage Classes

The concept of Storage Classes in Kubernetes refers to the capability to define different classes of storage with different performance and availability characteristics. You can think of it as categorizing storage resources based on their quality of service (QoS).

Imagine you are running a web application with both critical and non-critical data. By defining different Storage Classes, you can ensure that critical data gets high-performance storage, while non-critical data can reside on lower-cost, slower storage, optimizing cost-efficiency.

Volumes and Volume Types

Kubernetes offers a variety of volume types, each tailored to specific use cases. Volumes serve as abstractions that allow pods to access storage. Most commonly used volume types are:

- EmptyDir: This volume type is ephemeral and exists only for the pod's lifetime. It's useful for temporary storage.
- HostPath: It allows pods to use storage on the host node. This can be useful when dealing with node-specific data.
- NFS: Network File System (NFS) volumes enable sharing files across pods and nodes in a cluster. NFS volumes are commonly used for file-sharing scenarios.

StatefulSets

As presented in previous sections, StatefulSets is a specialized controller within Kubernetes which, in the context of storage, enables data to survive the life cycle of a Pod and persist even in the face of node disruptions.

In a scenario where you are running a distributed database where each node must have a unique identifier and stable storage, StatefulSets ensure that pods receive predictable names and persistent storage, allowing for seamless scaling and replacement.

Container Storage Interface (CSI)

As explained in previous section the Container Storage Interface (CSI) is an initiative supported by the CNCF which standardizes the integration between container orchestrators and storage equipment defining how a storage system should make data available to container orchestrators and how an orchestration platform should establish connections with storage equipment.

CSI compliant storage providers can be used in Kubernetes and today there are roughly a hundred providers compliant with the standard. We recommend always to check the latest list at the following site:

Drivers - Kubernetes CSI Developer Documentation (kubernetes-csi.github.io)

Dynamic provisioning

Dynamic Provisioning is a game-changer in Kubernetes storage. It automates the allocation of storage resources as needed, eliminating manual intervention.

Imagine a scenario where your application experiences varying storage demands. With dynamic provisioning, Kubernetes can automatically create and attach storage volumes to pods when the application requires more space. This ensures optimal resource utilization and minimizes administrative overhead.

More information in the following link:

https://kubernetes.io/docs/concepts/storage/dynamic-provisioning/

Networking and service mesh

In the preceding sections, we have introduced a variety of Kubernetes concepts and features that simplify application deployment and management. However, the complexities of networking and the rise of service mesh technologies remain a source of confusion for many. In this section we are aiming to simplify the complexities of networking in Kubernetes and provide insights into the world of service meshes.

Let's dive into the main concepts around networking.

Cluster Networking

Cluster networking forms the foundation of communication in Kubernetes. It's essential to understand how this communication happens to manage containerized applications effectively:

- Container-to-Container Communication: Within a pod, containers often need to communicate with each other. As we presented in previous sections, Kubernetes ensures that containers within a pod can communicate with each other over localhost. This local communication is critical for applications that require tightly coupled components, as it eliminates the need for external network calls and boosts performance.
- Pod-to-Pod Communication: Beyond a single pod, applications often consist of multiple pods and Kubernetes orchestrates the deployment of pods across different nodes within a cluster. To enable pod-to-pod communication, Kubernetes employs a combination of network plugins, routing, and IP addressing. One configuration is having each pod get assigned a unique IP address within the cluster, allowing it to communicate with other pods, regardless of their physical location.

Ingress Controllers

Managing external traffic and routing it to services within the Kubernetes cluster can be challenging. Ingress controllers play a pivotal role in this process:

- **Defining Routing Rules:** Ingress controllers allow us to define routing rules for incoming traffic. These rules include Hostnames (which domain or subdomaing the Ingress should handle), Paths (URL paths and how to map them to specific services) and TLS Configuration for secure communication.

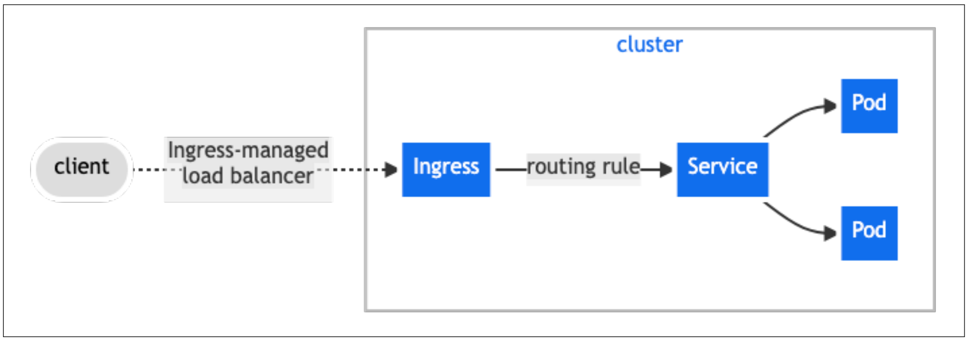
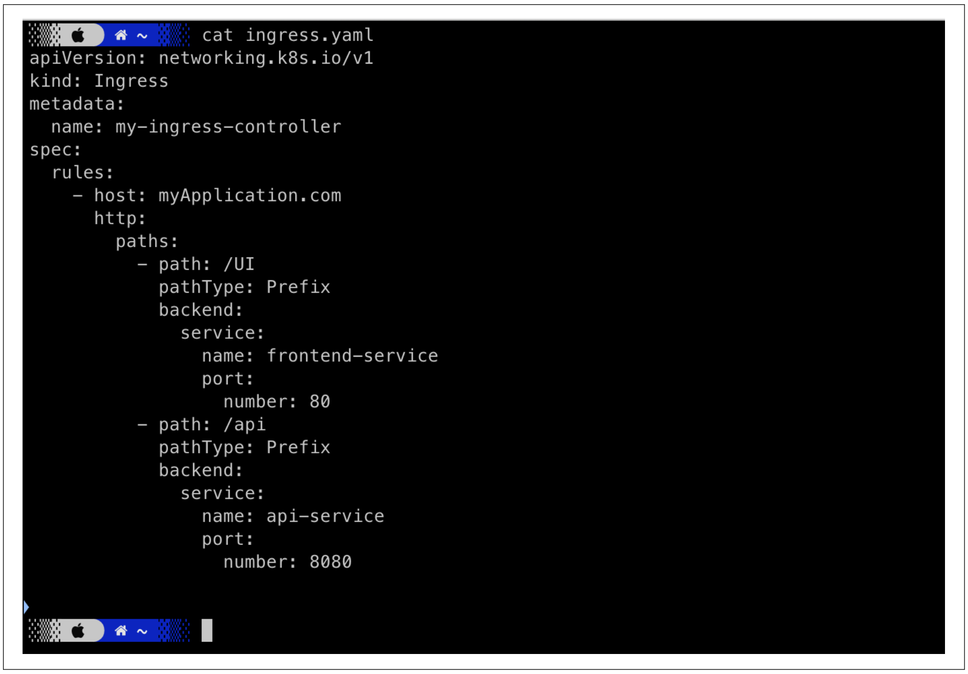


Figure 6-27. Ingress



```
cat ingress.yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-ingress-controller
spec:
  rules:
    - host: myApplication.com
      http:
        paths:
          - path: /UI
            pathType: Prefix
            backend:
              service:
                name: frontend-service
                port:
                  number: 80
          - path: /api
            pathType: Prefix
            backend:
              service:
                name: api-service
                port:
                  number: 8080
```

Figure 6-28. Example of an Ingress Definition

Some of the most popular Ingress Controllers are Nginx, Traefik, HAProxy and Contour.

More information: https://kubernetes.io/docs/concepts/services-networking/ingress/

Network Plugin

In Kubernetes, network plugins are essential components that provide the underlying infrastructure for networking within a cluster. They define how pod-to-pod communication, both within the same node and across different nodes, is managed and orchestrated. Network plugins are key to ensuring that containers and services can interact seamlessly within a dynamic and distributed environment like a Kubernetes cluster.

The main functions that are covered by a Network plugin are:

- **Routing and Networking Rules:** Network plugins are responsible for managing the routing and networking rules that govern how network traffic flows within the cluster. They ensure that packets originating from one pod reach their intended destination, whether it's another pod on the same node or a pod located on a different node within the cluster.
- **IP Address Assignment:** Network plugins are in charge of assigning unique IP addresses to each pod in the cluster. These IP addresses are used to identify and route traffic to specific pods. Proper IP address management is crucial for enabling pod-to-pod communication across nodes.
- **Overlay Networks**: Many network plugins employ overlay network technologies to create a virtual network layer on top of the physical infrastructure. This overlay network abstracts the complexities of the underlying network and allows pods to communicate as if they were on the same network segment, even if they are physically distributed across different nodes.

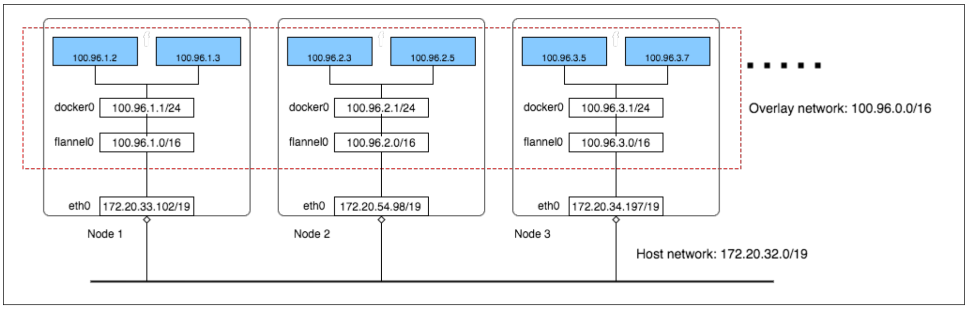


Figure 6-29. Example of an Overlay Network using flannel (Source: https://devopsschool.com/tutorial/kubernetes/kubernetes-cni-flannel-overlay-networking.html

- **Network Policies:** Network plugins often support Kubernetes Network Policies, which define rules for controlling pod-to-pod communication. These policies enable administrators to specify which pods can communicate with each other and under what conditions. Network policies enhance security by segmenting and isolating pods based on specific criteria.

Network plugins, such as Calico, Flannel, and Weave, manage the routing and networking rules to ensure seamless communication across nodes.

https://www.tigera.io/project-calico/

https://github.com/flannel-io/flannel?tab=readme-ov-file

https://github.com/weaveworks/weave

Service Discovery

Service discovery is a fundamental aspect of managing applications within a dynamic Kubernetes environment. It provides the ability to locate and interact with services seamlessly, regardless of their underlying network details. In Kubernetes, this is made possible primarily through the Kubernetes DNS service, discussed in detail below.

Kubernetes DNS Service

Kubernetes provides an internal DNS service that acts as a robust and automated way to discover and connect to services within the cluster. This service assigns DNS names to services, making them accessible to other components and applications running in the same cluster. Each service created within Kubernetes automatically receives a DNS entry in the cluster's DNS namespace.

Service Naming Conventions . Kubernetes follows a specific naming convention for services.

The DNS name of a service is typically formatted as [service-name].[namespace].svc.cluster.local, where:

- [service-name] is the name assigned to the Kubernetes service.
- [namespace] is the Kubernetes namespace where the service resides.
- svc.cluster.local is the DNS suffix representing services within the cluster.

Service Discovery by Name . Applications running within Kubernetes can discover and access services by their DNS names, abstracting away the need to know the underlying IP addresses or specific ports of those services. This decoupling simplifies service interactions and makes it easier to adapt to changes in the cluster's infrastructure.

Cross-Namespace Service Discovery. Kubernetes DNS also supports service discovery across different namespaces. This means that applications in one namespace can easily access services in other namespaces using the DNS name format mentioned earlier.

External DNS Integration

In addition to internal service discovery, Kubernetes can integrate with external DNS services. This enables seamless communication between services inside the Kubernetes cluster and those outside, such as services running in on-premises data centers or external cloud environments.

More information:

https://github.com/kubernetes-sigs/external-dns

Service Mesh

Service mesh has emerged as a transformative technology in the Kubernetes networking landscape and microservices architecture. It addresses several challenges related to traffic management, security, observability, and reliability in distributed applications.

In simple words a service mesh is a dedicated infrastructure layer that handles communication between microservices within a Kubernetes cluster. It provides a set of tools and services for managing, securing, and monitoring the interactions between these microservices. Service meshes are designed to abstract away the complexity of network communication, making it easier for developers to build resilient and efficient applications.

Service mesh builds on the service discovery mechanism by providing additional features such as:

- Dynamic Routing and traffic control (can route request based on a variety of criteria like service version, latency or even custom rules).
- Mutual TLS for securing the communication
- Distributed tracing and telemetry providing a comprehensive view of the system's performance and behavior and can be integrated with external tools seen in the observability section before like Prometheus, Grafana, or Jaeger

Main Service Mesh components

Here are the three main components of service mesh:

- **Proxy Sidecars:** Service mesh implementations typically rely on lightweight, sidecar proxy containers deployed alongside each microservice. These proxies intercept and manage all incoming and outgoing traffic, allowing for advanced traffic control and policy enforcement.
- **Control Plane:** The control plane is the brain of the service mesh. It consists of various components responsible for configuring and managing the proxy sidecars. These components include controllers, configuration servers, and policy engines.
- **Data Plane:** The data plane encompasses the actual microservices and the sidecar proxies that handle traffic between them. It includes the application code and the proxy sidecars that implement service mesh functionality.

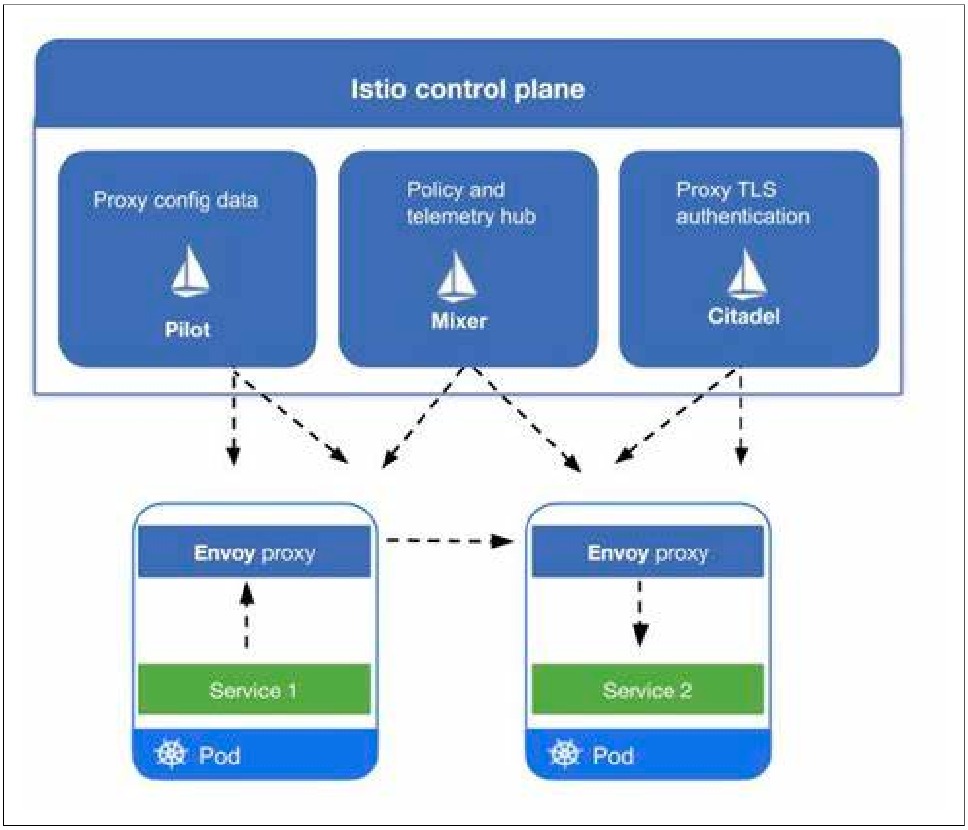


Figure 6-30. Example of Istio Service Mesh

Advantages of using Service Mesh

Traffic COntrol . One of the primary benefits of a service mesh is its ability to control and manage traffic between microservices. It includes routing, rate limiting and fault injection and provides features such as:

Load Balancing
: Service meshes can distribute incoming network traffic across multiple instances (pods) of a service preventing overload, optimizing resource utilization and improving availability.

  The traffic can be evenly distributed across healthy instances based on predefined rules (round-robin, least connections or even custom algorithms).

  In a service mesh load balancing is usually handled by the sidecar proxies (such as Envoy or Linkerd)

Traffic Shifting
: Advanced service meshes allow for gradually transitioning traffic from one version of a service to another (used often in Canary releases or gradual deployments). The main difference between Traffic Shifting and Weighted Load Balancer is that Traffic Shifting focus on transitioning between different versions of a service and Weighted Load Balancer focus is distributing traffic based on predefined weights.

Circuit Breaking
: Service meshes can detect unhealthy microservices and prevent further traffic from being sent to them, improving the overall stability of the application.

Security. Service meshes enhance security by implementing features like:

- **Encryption:** They can encrypt traffic between microservices, ensuring data confidentiality and integrity.
- **Authentication and Authorization:** Service meshes can enforce authentication and authorization policies, ensuring that only authorized services can communicate with each other.

Observability. Monitoring and observability are critical in microservices architectures. Service meshes provide tools for:

- **Metrics and Tracing:** They offer detailed metrics and tracing capabilities, allowing developers to gain insights into the behavior of their microservices.
- **Logging:** Service meshes can capture and centralize logs from all microservices, simplifying troubleshooting and debugging.

Reliability. By managing traffic, enforcing policies, and offering tools for monitoring, service meshes contribute to the reliability of microservices-based applications. They help prevent cascading failures and improve overall application uptime.

Kubernetes networking offers a full set of capabilities and standards that allows you to extend the cluster with specific functionality depending on the needs of your application and the environment where they are running.

Under the umbrella of the CNCF there are two graduated projects in the Service Mesh domain: Istio and Linkerd.

More information:

https://istio.io/latest/

https://linkerd.io/

Kubernetes security

As we have presented during the previous sections of this book, Kubernetes offers advantages in terms of scalability, fault tolerance, and resource utilization in scenarios where distributed systems architecture patterns are an advantage. That's the essence of Kubernetes: a distributed system managing containerized applications across multiple nodes or machines in a cluster.

While this design offers multiple advantages it also presents challenges when it comes to observability and security and it would require a whole book to deepen on all the aspects related to this. See Learn Kubernetes Security [Book] (oreilly.com) for more in depth coverage.

The goal for this book is to give you a high level picture about the different concepts and projects under the CNCF umbrella. With that in mind, we'll focus on broad knowledge surrounding Kubernetes security that anyone working with Kubernetes should know.

Container Image security strategies

The journey to increase the security in a Kubernetes environment begins with container images. Containers encapsulate application code and dependencies, making them portable and efficient. However, if the container images used contain vulnerabilities, they can become the entry point for attackers.

There are different strategies that can be adopted to minimize the risk of image vulnerabilities. We discuss the principal ones below.

Image Scanning

There are a good amount of tools (for example Harbor, Anchore, Docker Scout or OpenScap) to automatically inspect container images for known vulnerabilities and misconfigurations. These tools integrate with the container registry (whatever it is) and can prevent vulnerable images from being deployed.

Under the umbrella of CNCF the most known project for Image Scanning is Harbor *https://goharbor.io/*. Harbor provides static analysis of vulnerabilities in images through the open source projects Trivy and Clair and can be connected to additional scanners like Anchore for example.

Vulnerability patching

Adopt a proper process to regularly update the base images and application dependencies to patch known vulnerabilities. Implementing a patching strategy significantly reduces the probability of a security breach in the distributed environment.

Minimal Image build strategy

Follow the principle of least privilege when creating container images. Only include absolutely necessary components, libraries, and dependencies, minimizing the attack surface and reducing the potential for vulnerabilities.

Kubernetes cluster protection strategies

Whether a cluster is running in a dev environment or in production you should always follow the same security principles along the life cycle. Having the same policies applied from dev to prod will minimize the surface attack and reduce the possibilities of being impacted by a threat.

API Server Security

The API server is the central point of interaction with the Kubernetes cluster. Implement proper authentication and authorization mechanisms, enforce RBAC (Role-Based Access Control) policies, and use network policies (*https://kubernetes.io/docs/concepts/services-networking/network-policies/*) to restrict access to the API server

Note: Kubernetes RBAC

Kubernetes RBAC will allow you to ensure that cluster users and workloads have only the necessary access to resources preventing unauthorized actions and privilege escalation escenarios.

The main components of K8S RBAC are:

Roles: Which define permissions within a specific namespace

ClusterRoles: Non-namespaced resource allowing you to define permissions cluster-wide

RoleBinding and ClusterRoleBindings: It's the roles/ClusterRole association with users, groups or service accounts.

Service accounts are non-human account used by pods, system components and other entities (inside or outside the cluster) to authenticate and control their access. Every namespace automatically gets a default ServiceAccount at the creation time.

etcd Security

The etcd database stores critical cluster information. Encrypt etcd data at rest and in transit. Maintain regular backups, and restrict access to the etcd endpoints to authorized users and components.

More information:

https://kubernetes.io/docs/tasks/administer-cluster/encrypt-data/

Node Security (including runtime security)

Nodes are the worker machines in the cluster. Secure nodes by applying security updates promptly, implementing container runtime security features like SELinux or AppArmor, and using tools like PodSecurity Admission or OPA Gatekeeper to enforce security policies.

Implement runtime security measures to detect and respond to suspicious activities within your cluster. Tools like Falco or Sysdig can provide real-time monitoring and alerting for unusual behavior.

More information:

https://kubernetes.io/docs/concepts/security/pod-security-admission/

https://falco.org/

Implement Network Policies: Controlling Communication

Kubernetes network policies define how pods can communicate with each other and external resources. Employ network policies to control traffic flow and limit pod-to-pod communication to only what's necessary for your applications.

Secrets Management: Protecting Sensitive Data

Manage sensitive information like API tokens, passwords, and certificates using Kubernetes Secrets. Encrypt secrets at rest and restrict access to them through RBAC and pod service accounts.

More information:

https://kubernetes.io/docs/concepts/configuration/secret/

In summary, Kubernetes security is a multifaceted endeavor that requires a holistic approach. By addressing image vulnerabilities, securing the Kubernetes cluster, implementing network and pod security policies and managing secrets you can create a robust security posture for the containerized applications.

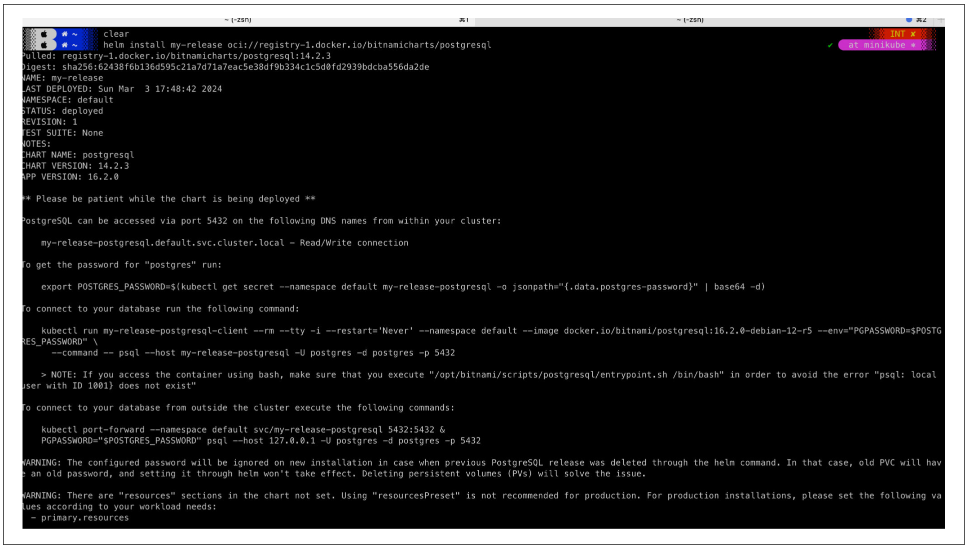
Other Kubernetes related components

Helm:

Is the de facto package manager for Kubernetes and a graduated project in the CNCF. . It simplifies the management of Kubernetes applications allowing you to define, install and upgrade any type of applicacion through the helm charts.

The Helm Charts are packages of pre-configured Kubernetes resources (Deployments, Services, ConfigMaps) that describe the applicaiont.

The ArtifactHub is the central repository not only for lookin for helm charts but also for other types of artifacts such as plugins, Kubernetes Modules programmed in KCL among others.



```
clear
helm install my-release oci://registry-1.docker.io/bitnamicharts/postgresql
ulled: registry-1.docker.io/bitnamicharts/postgresql:14.2.3
igest: sha256:62438f6b136d595c21a7d71a7eac5e38df9b334c1c5d0fd2939bdcba556da2de
AME: my-release
AST DEPLOYED: Sun Mar  3 17:48:42 2024
AMESPACE: default
TATUS: deployed
EVISION: 1
EST SUITE: None
OTES:
HART NAME: postgresql
HART VERSION: 14.2.3
PP VERSION: 16.2.0

* Please be patient while the chart is being deployed **

ostgreSQL can be accessed via port 5432 on the following DNS names from within your cluster:

   my-release-postgresql.default.svc.cluster.local - Read/Write connection

o get the password for "postgres" run:

   export POSTGRES_PASSWORD=$(kubectl get secret --namespace default my-release-postgresql -o jsonpath="{.data.postgres-password}" | base64 -d)

o connect to your database run the following command:

   kubectl run my-release-postgresql-client --rm --tty -i --restart='Never' --namespace default --image docker.io/bitnami/postgresql:16.2.0-debian-12-r5 --env="PGPASSWORD=$POSTG
ES_PASSWORD" \
     --command -- psql --host my-release-postgresql -U postgres -d postgres -p 5432

   > NOTE: If you access the container using bash, make sure that you execute "/opt/bitnami/scripts/postgresql/entrypoint.sh /bin/bash" in order to avoid the error "psql: local
ser with ID 1001} does not exist"

o connect to your database from outside the cluster execute the following commands:

   kubectl port-forward --namespace default svc/my-release-postgresql 5432:5432 &
   PGPASSWORD="$POSTGRES_PASSWORD" psql --host 127.0.0.1 -U postgres -d postgres -p 5432

ARNING: The configured password will be ignored on new installation in case when previous PostgreSQL release was deleted through the helm command. In that case, old PVC will hav
 an old password, and setting it through helm won't take effect. Deleting persistent volumes (PVs) will solve the issue.

ARNING: There are "resources" sections in the chart not set. Using "resourcesPreset" is not recommended for production. For production installations, please set the following va
ues according to your workload needs:
 - primary.resources
```

Figure 6-31. Installing a postgreSQL DB using helm

More information:

https://artifacthub.io/

https://helm.sh/

Dapr

Dapr is a project under the CNF at incubation stage. Dapr stands for Distributed Application Runtime which outperforms when designing and deploying a microservices Application.

An application architected and build for microservices uses to present a set of well know challenges: Encryption, Message brokering, observability, service discovery….. You now have a good understanding of these concepts.

And now imagine that not all the microservices which compose your application are not developed in the same programming language ? Should you re-write the "logic" or "patterns" for any of those languages? Each microservice is likely having its own lifecyle and in larger applications, there will be many different teams working each of them in a different microservice. How would you handle and test the cross microservice calls?

Well that's exactly the space that Dapr wants to cover abstracting the application logic and programming to the underlaying services through a well defined set of Restful API to call services through HTTP. Behind the scene Dapr runs as a sidecar process or container alongside the application code, exposing its APIs via HTTP or gRPC which enables easy integration with Dapr without having to modify the application code.

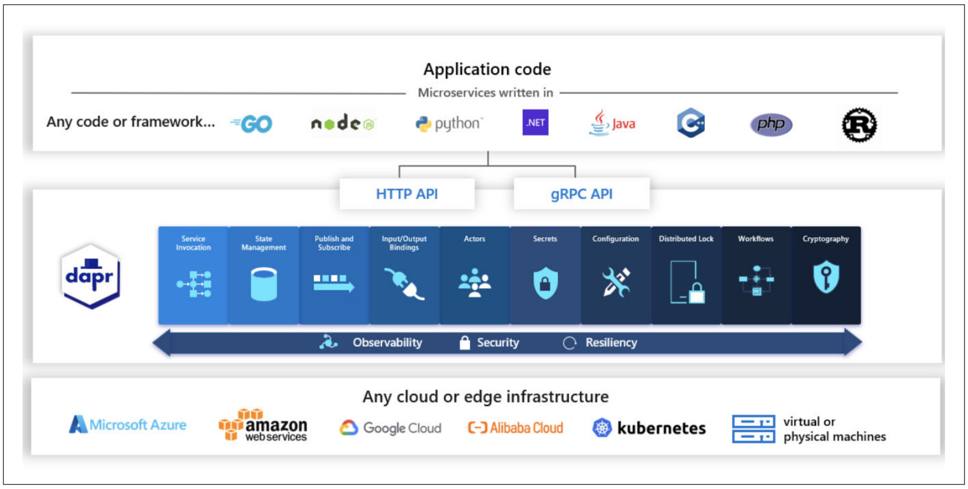
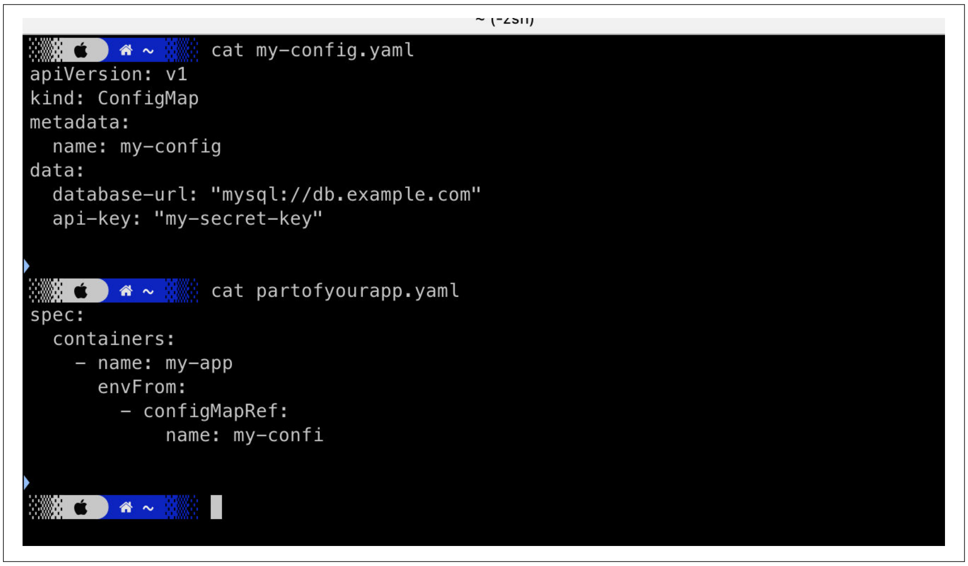


Figure 6-32. Dapr High level overview

ConfigMap

ConfigMap is a kubernetes resource that allows you to store configuration data separately from your application code, providing a better way for configuration settings, environment variable and other non-sensitive data.



```
~ (-zsh)
 ~  cat my-config.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-config
data:
  database-url: "mysql://db.example.com"
  api-key: "my-secret-key"

 ~  cat partofyourapp.yaml
spec:
  containers:
    - name: my-app
      envFrom:
        - configMapRef:
            name: my-confi

 ~  
```

Figure 6-33. Example for creating a ConfigMap and using it when creating a pod.

MiniKube

MiniKube is a tool that helps you to quickly set up a local Kubernetes cluster on macOS, Linux, and Windows. It is the recommended tool for practicing when learning and developing applications for Kubernetes.

For detailed instructions about minimum requirements and installation steps go to the official site:

https://minikube.sigs.k8s.io/docs/start/

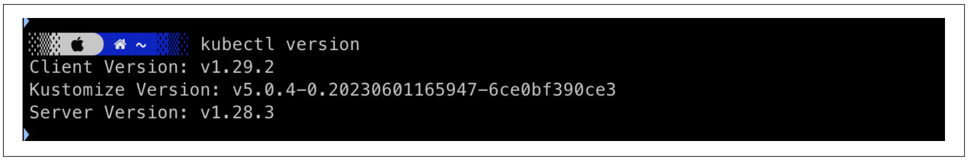
Main commands

The most common way to interact with a Kubernetes cluster, whether you are an administrator, developer, or operator, is through the command-line tool called kubectl.

kubectl is short for "Kubernetes Control," and it is the official command-line interface (CLI) for Kubernetes.

Below are some of the essential kubectl commands that allow you to interact with and manage resources within a Kubernetes cluster. There are many more commands and options available, so we recommend you to explore the Kubernetes documentation for in-depth information and usage examples.

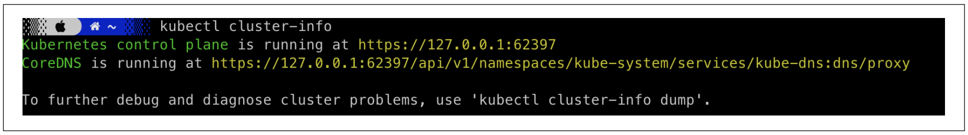
kubectl version: Check the client and server versions of kubectl and the Kubernetes cluster.



```
kubectl version
Client Version: v1.29.2
Kustomize Version: v5.0.4-0.20230601165947-6ce0bf390ce3
Server Version: v1.28.3
```

Figure 6-34. Version sample output

kubectl cluster-info: Display information about the Kubernetes cluster, including the API server URL.

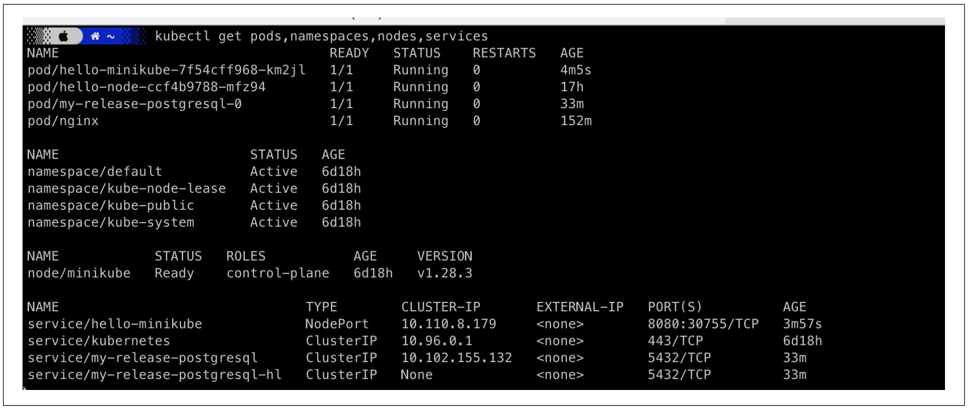


```
kubectl cluster-info
Kubernetes control plane is running at https://127.0.0.1:62397
CoreDNS is running at https://127.0.0.1:62397/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
```

Figure 6-35. cluster-info

kubectl get: Retrieve information about various Kubernetes resources. Here are some examples:

- kubectl get pods: List all pods in the current namespace.
- kubectl get nodes: List all nodes in the cluster.
- kubectl get services: List all services in the current namespace.

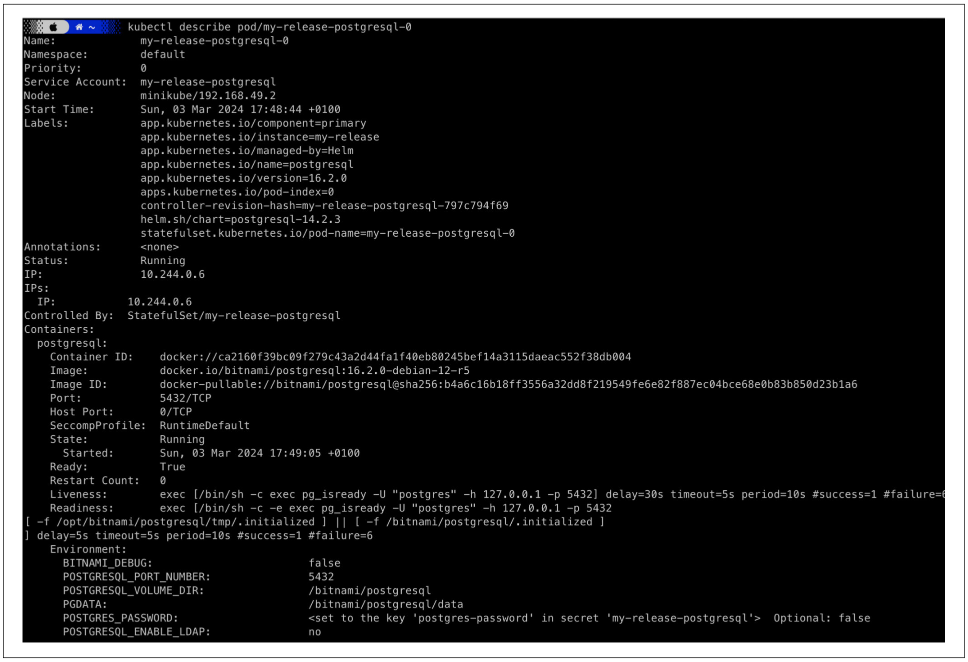


```
kubectl get pods,namespaces,nodes,services
NAME                                  READY   STATUS    RESTARTS   AGE
pod/hello-minikube-7f54cff968-km2jl   1/1     Running   0          4m5s
pod/hello-node-ccf4b9788-mfz94        1/1     Running   0          17h
pod/my-release-postgresql-0           1/1     Running   0          33m
pod/nginx                             1/1     Running   0          152m

NAME                         STATUS   AGE
namespace/default            Active   6d18h
namespace/kube-node-lease    Active   6d18h
namespace/kube-public        Active   6d18h
namespace/kube-system        Active   6d18h

NAME            STATUS   ROLES           AGE     VERSION
node/minikube   Ready    control-plane   6d18h   v1.28.3

NAME                               TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
service/hello-minikube             NodePort    10.110.8.179     <none>        8080:30755/TCP   3m57s
service/kubernetes                 ClusterIP   10.96.0.1        <none>        443/TCP          6d18h
service/my-release-postgresql      ClusterIP   10.102.155.132   <none>        5432/TCP         33m
service/my-release-postgresql-hl   ClusterIP   None             <none>        5432/TCP         33m
```

Figure 6-36. kubectl get samples

kubectl describe: Get detailed information about a specific resource. Here are a couple examples:

- kubectl describe pod <pod-name>: Display details about a specific pod.
- kubectl describe node <node-name>: Display details about a specific node.



```
kubectl describe pod/my-release-postgresql-0
Name:             my-release-postgresql-0
Namespace:        default
Priority:         0
Service Account:  my-release-postgresql
Node:             minikube/192.168.49.2
Start Time:       Sun, 03 Mar 2024 17:48:44 +0100
Labels:           app.kubernetes.io/component=primary
                  app.kubernetes.io/instance=my-release
                  app.kubernetes.io/managed-by=Helm
                  app.kubernetes.io/name=postgresql
                  app.kubernetes.io/version=16.2.0
                  apps.kubernetes.io/pod-index=0
                  controller-revision-hash=my-release-postgresql-797c794f69
                  helm.sh/chart=postgresql-14.2.3
                  statefulset.kubernetes.io/pod-name=my-release-postgresql-0
Annotations:      <none>
Status:           Running
IP:               10.244.0.6
IPs:
  IP:           10.244.0.6
Controlled By:  StatefulSet/my-release-postgresql
Containers:
  postgresql:
    Container ID:    docker://ca2160f39bc09f279c43a2d44fa1f40eb80245bef14a3115daeac552f38db004
    Image:           docker.io/bitnami/postgresql:16.2.0-debian-12-r5
    Image ID:        docker-pullable://bitnami/postgresql@sha256:b4a6c16b18ff3556a32dd8f219549fe6e82f887ec04bce68e0b83b850d23b1a6
    Port:            5432/TCP
    Host Port:       0/TCP
    SeccompProfile:  RuntimeDefault
    State:           Running
      Started:       Sun, 03 Mar 2024 17:49:05 +0100
    Ready:           True
    Restart Count:   0
    Liveness:        exec [/bin/sh -c exec pg_isready -U "postgres" -h 127.0.0.1 -p 5432] delay=30s timeout=5s period=10s #success=1 #failure=
    Readiness:       exec [/bin/sh -c -e exec pg_isready -U "postgres" -h 127.0.0.1 -p 5432
[ -f /opt/bitnami/postgresql/tmp/.initialized ] || [ -f /bitnami/postgresql/.initialized ]
] delay=5s timeout=5s period=10s #success=1 #failure=6
    Environment:
      BITNAMI_DEBUG:                          false
      POSTGRESQL_PORT_NUMBER:                 5432
      POSTGRESQL_VOLUME_DIR:                  /bitnami/postgresql
      PGDATA:                                 /bitnami/postgresql/data
      POSTGRES_PASSWORD:                      <set to the key 'postgres-password' in secret 'my-release-postgresql'>  Optional: false
      POSTGRESQL_ENABLE_LDAP:                 no
```

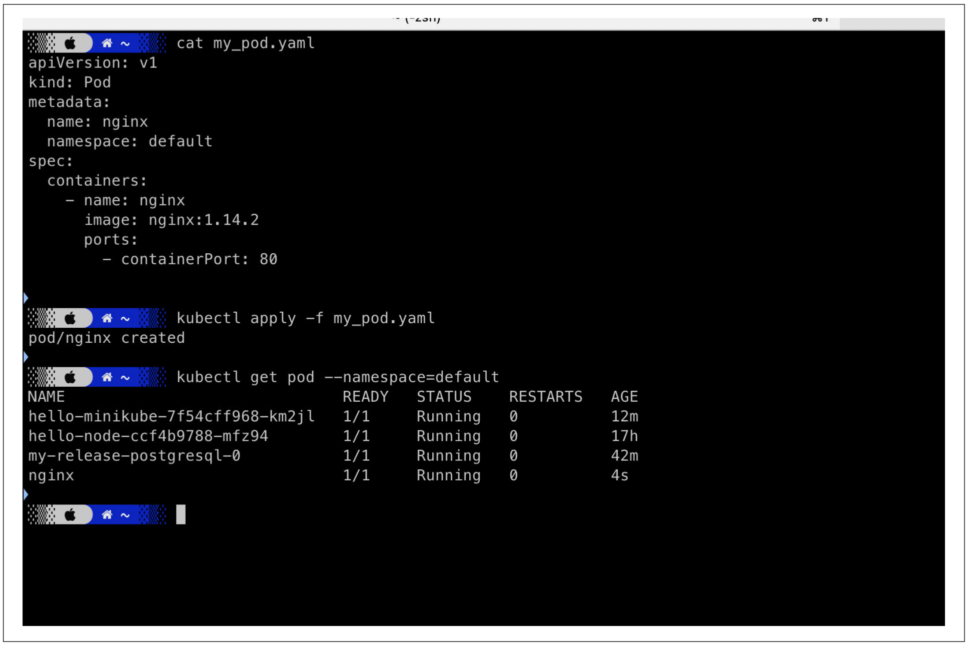
Figure 6-37. Sample of pod description

kubectl create: Create resources from a YAML or JSON file. Here are a couple examples:

- kubectl create -f pod.yaml: Create a pod defined in pod.yaml.
- kubectl create deployment <deployment-name> --image=<image-name>: Create a deployment using a specified container image.

kubectl apply: Apply changes to resources from a YAML or JSON file. It can create or update resources. Here's an example:

- kubectl apply -f deployment.yaml: Apply changes defined in deployment.yaml.



```
cat my_pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  namespace: default
spec:
  containers:
    - name: nginx
      image: nginx:1.14.2
      ports:
        - containerPort: 80

kubectl apply -f my_pod.yaml
pod/nginx created

kubectl get pod --namespace=default
NAME                              READY   STATUS    RESTARTS   AGE
hello-minikube-7f54cff968-km2jl   1/1     Running   0          12m
hello-node-ccf4b9788-mfz94        1/1     Running   0          17h
my-release-postgresql-0           1/1     Running   0          42m
nginx                             1/1     Running   0          4s
```

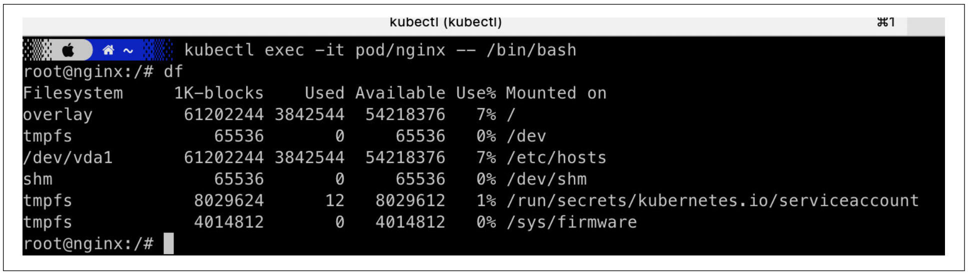
Figure 6-38. Sample of kubectl apply creating a pod

kubectl delete: Delete resources. Here's an example:

- kubectl delete pod <pod-name>: Delete a specific pod.
- kubectl delete deployment <deployment-name>: Delete a deployment.

kubectl exec: Execute a command in a running container. Here's an example:

- kubectl exec -it <pod-name> -- /bin/bash: Open a shell in a specific pod.



```
kubectl exec -it pod/nginx -- /bin/bash
root@nginx:/# df
Filesystem     1K-blocks    Used Available Use% Mounted on
overlay         61202244 3842544  54218376   7% /
tmpfs              65536       0     65536   0% /dev
/dev/vda1       61202244 3842544  54218376   7% /etc/hosts
shm                65536       0     65536   0% /dev/shm
tmpfs            8029624      12   8029612   1% /run/secrets/kubernetes.io/serviceaccount
tmpfs            4014812       0   4014812   0% /sys/firmware
root@nginx:/#
```

Figure 6-39. Kubectl exec sample

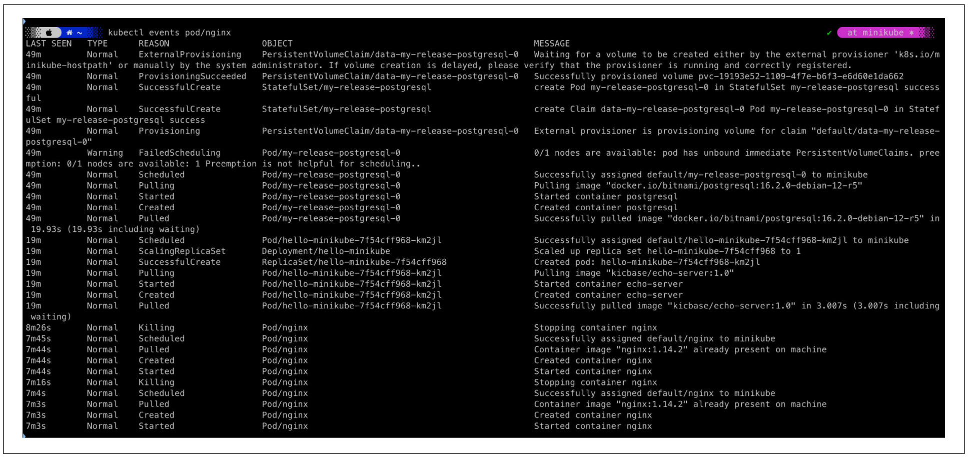
kubectl logs: View the logs of a specific container in a pod. Here's a couple examples:

- kubectl logs <pod-name>: View logs for the primary container in a pod.

- kubectl logs <pod-name> -c <container-name>: View logs for a specific container in a pod.

kubectl events : View the events of a specific container in a pod. Here's a couple examples:

- kubectl events <pod-name>: View logs for the primary container in a pod.
- kubectl logs <pod-name> -c <container-name>: View events for a specific container in a pod.



```
kubectl events pod/nginx                                                                                                            ✔  at minikube ⎈
LAST SEEN   TYPE      REASON                    OBJECT                                             MESSAGE
49m         Normal    ExternalProvisioning      PersistentVolumeClaim/data-my-release-postgresql-0   Waiting for a volume to be created either by the external provisioner 'k8s.io/m
inikube-hostpath' or manually by the system administrator. If volume creation is delayed, please verify that the provisioner is running and correctly registered.
49m         Normal    ProvisioningSucceeded     PersistentVolumeClaim/data-my-release-postgresql-0   Successfully provisioned volume pvc-19193e52-1109-4f7e-b6f3-e6d60e1da662
49m         Normal    SuccessfulCreate          StatefulSet/my-release-postgresql                    create Pod my-release-postgresql-0 in StatefulSet my-release-postgresql success
ful
49m         Normal    SuccessfulCreate          StatefulSet/my-release-postgresql                    create Claim data-my-release-postgresql-0 Pod my-release-postgresql-0 in Statef
ulSet my-release-postgresql success
49m         Normal    Provisioning              PersistentVolumeClaim/data-my-release-postgresql-0   External provisioner is provisioning volume for claim "default/data-my-release-
postgresql-0"
49m         Warning   FailedScheduling          Pod/my-release-postgresql-0                          0/1 nodes are available: pod has unbound immediate PersistentVolumeClaims. pree
mption: 0/1 nodes are available: 1 Preemption is not helpful for scheduling..
49m         Normal    Scheduled                 Pod/my-release-postgresql-0                          Successfully assigned default/my-release-postgresql-0 to minikube
49m         Normal    Pulling                   Pod/my-release-postgresql-0                          Pulling image "docker.io/bitnami/postgresql:16.2.0-debian-12-r5"
49m         Normal    Started                   Pod/my-release-postgresql-0                          Started container postgresql
49m         Normal    Created                   Pod/my-release-postgresql-0                          Created container postgresql
49m         Normal    Pulled                    Pod/my-release-postgresql-0                          Successfully pulled image "docker.io/bitnami/postgresql:16.2.0-debian-12-r5" in
 19.93s (19.93s including waiting)
19m         Normal    Scheduled                 Pod/hello-minikube-7f54cff968-km2jl                  Successfully assigned default/hello-minikube-7f54cff968-km2jl to minikube
19m         Normal    ScalingReplicaSet         Deployment/hello-minikube                            Scaled up replica set hello-minikube-7f54cff968 to 1
19m         Normal    SuccessfulCreate          ReplicaSet/hello-minikube-7f54cff968                 Created pod: hello-minikube-7f54cff968-km2jl
19m         Normal    Pulling                   Pod/hello-minikube-7f54cff968-km2jl                  Pulling image "kicbase/echo-server:1.0"
19m         Normal    Started                   Pod/hello-minikube-7f54cff968-km2jl                  Started container echo-server
19m         Normal    Created                   Pod/hello-minikube-7f54cff968-km2jl                  Created container echo-server
19m         Normal    Pulled                    Pod/hello-minikube-7f54cff968-km2jl                  Successfully pulled image "kicbase/echo-server:1.0" in 3.007s (3.007s including
 waiting)
8m26s       Normal    Killing                   Pod/nginx                                            Stopping container nginx
7m45s       Normal    Scheduled                 Pod/nginx                                            Successfully assigned default/nginx to minikube
7m44s       Normal    Pulled                    Pod/nginx                                            Container image "nginx:1.14.2" already present on machine
7m44s       Normal    Created                   Pod/nginx                                            Created container nginx
7m44s       Normal    Started                   Pod/nginx                                            Started container nginx
7m16s       Normal    Killing                   Pod/nginx                                            Stopping container nginx
7m4s        Normal    Scheduled                 Pod/nginx                                            Successfully assigned default/nginx to minikube
7m3s        Normal    Pulled                    Pod/nginx                                            Container image "nginx:1.14.2" already present on machine
7m3s        Normal    Created                   Pod/nginx                                            Created container nginx
7m3s        Normal    Started                   Pod/nginx                                            Started container nginx
```

Figure 6-40. Events sample

kubectl port-forward: Forward a local port to a port on a pod. Here's an example:

- kubectl port-forward <pod-name> <local-port>:<pod-port>: Forward local port to a pod.

kubectl scale: Scale the number of replicas in a deployment. Here's an example:

- kubectl scale deployment <deployment-name> --replicas=<desired-replicas>: Scale a deployment to the desired number of replicas.

kubectl rollout: Manage rollouts and updates to resources. Here's a couple examples:

- kubectl rollout status deployment <deployment-name>: Check the status of a deployment rollout.
- kubectl rollout undo deployment <deployment-name>: Rollback a deployment to a previous version.

Summary

This chapter has been a long journey exploring the fundamental concepts of Kubernetes, demystifying its architecture and component. From the Basic building blocks to a more sophisticated components like service mesh we hope you now have a good understanding on what Cloud Native is, the relationship with the Cloud Native Computing Foundation and how Kuberentes plays an essential role moving forward the industry towards an inter-operable system and a foundation of open standards that will for sure accelerate technology adoption and will bring new waves of innovation.

You are now just one chapter left for finishing this journey and without any doubt you are now at the beginning of a nurturing cloud native journey.

CHAPTER 7

Final recommendations

A Note for Early Release Readers

With Early Release ebooks, you get books in their earliest form—the author's raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 7th chapter of the final book. Please note that the GitHub repo will be made active later on.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at *shunter@oreilly.com*.

You are approaching the final step for getting your KCNA certification: Passing the exam. Whatever the reason that brought you to prepare for the KCNA, the Cloud Native ecosystem is an endless source of learning. Having a good set of reference material that let you land key concepts and guide you through practical examples is not and option but a requirement.

This chapter serves as a reference guide in the journey towards becoming a Kubernetes and Cloud Native Associate including important references to material to not only excel in the certification exam but also to thrive in your ongoing career as a Kubernetes expert.

Throughout this book, you've gained in-depth knowledge and practical insights into Kubernetes, its architecture, functionalities, and applications and the CNCF ecosystem in general. Remember, the journey of learning Kubernetes is continuous and ever-evolving. As you prepare to validate your skills through certification, know that this is just one milestone in a rewarding path of lifelong learning and professional growth in the dynamic world of Kubernetes and distributed systems.

Additional information for exam preparation

The Kubernetes and Cloud Native Associate (KCNA) exam is a foundational step for those embarking on their journey in Kubernetes and cloud-native technologies. This exam is structured to assess the fundamental understanding of Kubernetes and is the baseline for more advanced certifications and professional growth in the field.

It's crucial as a candidate understanding the basics of the KCNA exam:

- Exam Format and Length: The KCNA is predominantly a multiple-choice test conducted online under proctored conditions. Typically, the exam duration is around 90 minutes.
- Topics Covered: The exam's curriculum is comprehensive, covering a range of topics related to Kubernetes and cloud-native technologies. This includes basic principles of Kubernetes, understanding of cloud-native architecture, elementary deployment and management of applications, and an overview of relevant tools and projects in the ecosystem.
  - Kubernetes Fundamentals: Covers 46% of the exam, including core Kubernetes concepts.
  - Container Orchestration: 22% of the exam, focusing on managing containers.
  - Cloud Native Architecture: 16% of the exam, exploring cloud-native design patterns.
  - Cloud Native Observability: 8% of the exam, emphasizing monitoring and observability.
  - Cloud Native Application Delivery: 8% of the exam, addressing application deployment.
- Level of Difficulty: Designed with beginners in mind, the KCNA serves as an introductory certification setting the stage for further exploration and specialization in Kubernetes and cloud-native technologies.

Kubernetes community experts to follow

Engaging with community experts and influencers in the Kubernetes and cloud-native sphere enriches your pool of resources and insights available during your preparation for the Kubernetes and Cloud Native Associate (KCNA) exam.

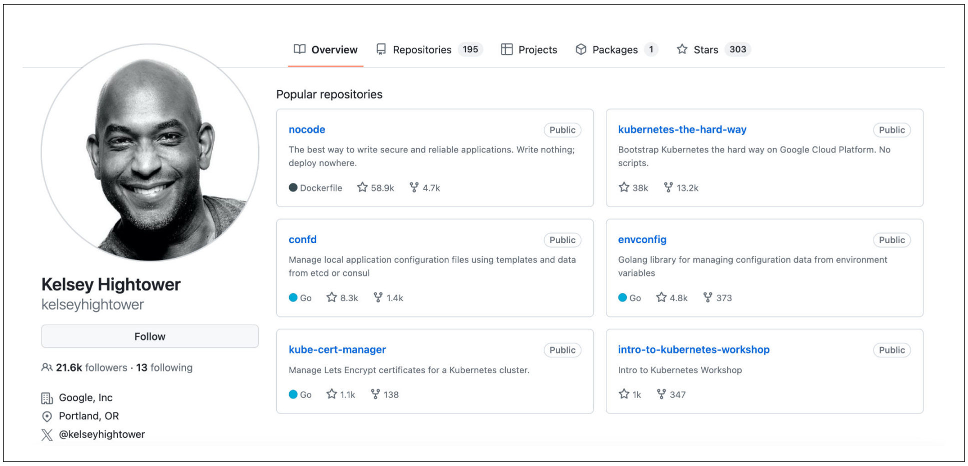
The learning path for projects in the cloud native realm generally requires more than just studying books; it's seeing how things play out in the real world that makes the difference and planting the seeds to master the technology and provide solutions to real and complex challenges. The community experts are everywhere – blogs, Twitter, YouTube,twitch,...– making it easy to grab some serious knowledge on the fly.

The community experts we discuss belowbring a wide variety of unique perspectives and expertise. Their contributions range from in-depth technical knowledge to practical tips and industry trends. This makes them some of the best sources for understanding both Kubernetes and the wider cloud-native landscape.

Here is the list of experts and community influencers which we recommend to follow:

Kelsey Hightower: An influential advocate for Kubernetes, Kelsy is renowned for making complex Kubernetes concepts easy to grasp. He shares practical advice and step-by-step demonstrations in his "Twitter" and Github repositories. His ability to simplify and demystify Kubernetes makes him a go-to resource for both beginners and experienced practitioners.

Link to the GitHub Repository: *https://github.com/kelseyhightower*



Kelsey Hightower
kelseyhightower

21.6k followers · 13 following

Google, Inc
Portland, OR
@kelseyhightower

Popular repositories

nocode — Public
The best way to write secure and reliable applications. Write nothing; deploy nowhere.
Dockerfile ☆ 58.9k 4.7k

kubernetes-the-hard-way — Public
Bootstrap Kubernetes the hard way on Google Cloud Platform. No scripts.
☆ 38k 13.2k

confd — Public
Manage local application configuration files using templates and data from etcd or consul
Go ☆ 8.3k 1.4k

envconfig — Public
Golang library for managing configuration data from environment variables
Go ☆ 4.8k 373

kube-cert-manager — Public
Manage Lets Encrypt certificates for a Kubernetes cluster.
Go ☆ 1.1k 138

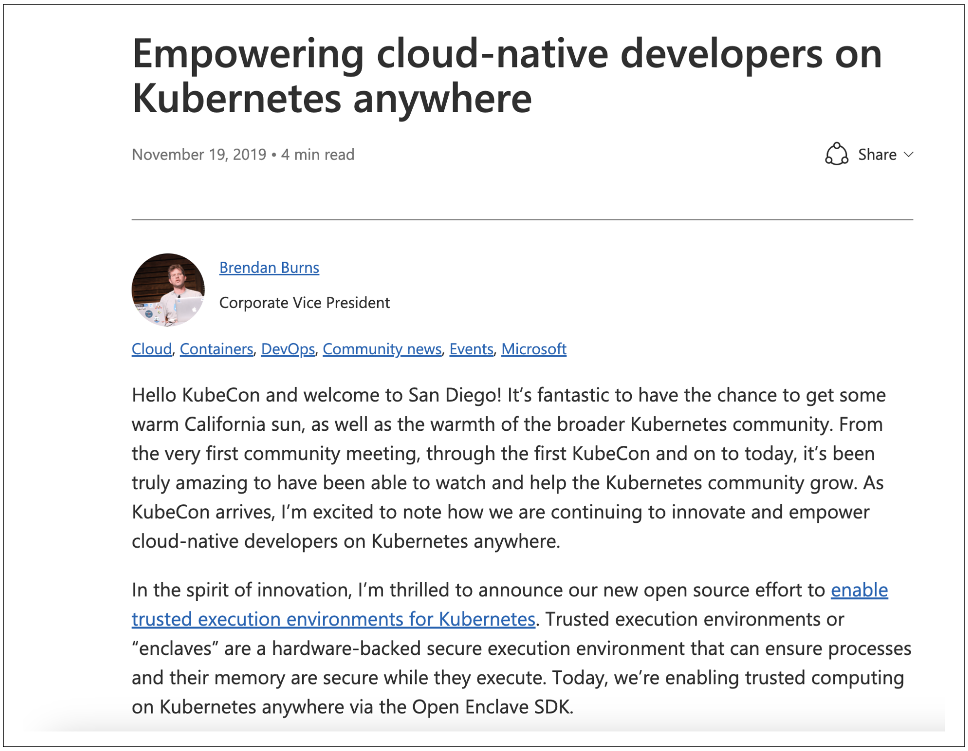
intro-to-kubernetes-workshop — Public
Intro to Kubernetes Workshop
☆ 1k 347

Figure 7-1. Kelsey Hightower Github Repository

We strongly recommend forking the kubernetes-the-hard-way and building a Kubernetes cluster from scratch with the step by step guide inside the repository. This is without any doubt one of the best possible path to understand Kubernetes practically and its components.

Brendan Burns: As one of the brains behind Kubernetes, Brendad has an unique depth of knowledge about the system. He offers rich insights into the inner workings and future trajectory of Kubernetes. His blog posts and conference presentations are a goldmine for anyone seeking to understand Kubernetes at a deep structural level to get a glimpse into its evolving landscape.

The link for his latest post is available here: *https://cloudblogs.microsoft.com/opensource/author/brendan-burns/*



Empowering cloud-native developers on Kubernetes anywhere

November 19, 2019 • 4 min read

Brendan Burns

Corporate Vice President

Cloud, Containers, DevOps, Community news, Events, Microsoft

Hello KubeCon and welcome to San Diego! It's fantastic to have the chance to get some warm California sun, as well as the warmth of the broader Kubernetes community. From the very first community meeting, through the first KubeCon and on to today, it's been truly amazing to have been able to watch and help the Kubernetes community grow. As KubeCon arrives, I'm excited to note how we are continuing to innovate and empower cloud-native developers on Kubernetes anywhere.

In the spirit of innovation, I'm thrilled to announce our new open source effort to enable trusted execution environments for Kubernetes. Trusted execution environments or "enclaves" are a hardware-backed secure execution environment that can ensure processes and their memory are secure while they execute. Today, we're enabling trusted computing on Kubernetes anywhere via the Open Enclave SDK.

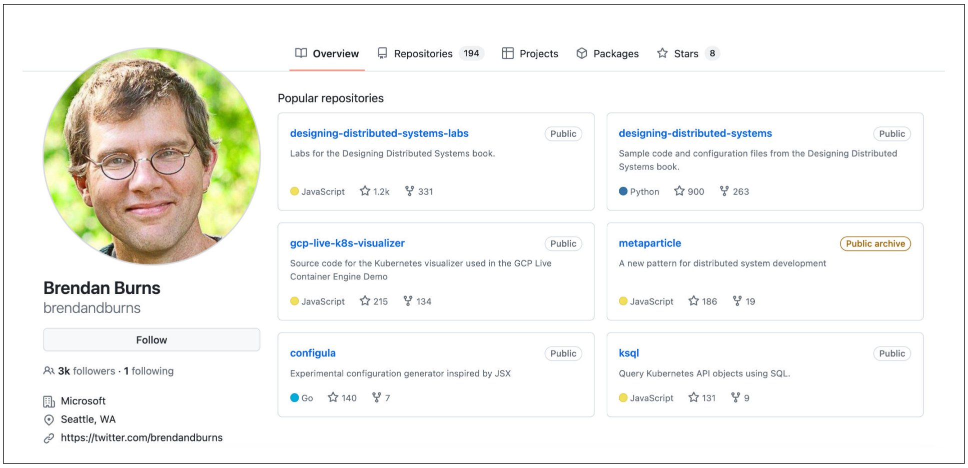
Figure 7-2. Snippet of one of the Brendan Burns post during a Kubecon in 2019

We highly recommend to listen the following podcast where Brendan Burns talks about the new Long term support strategy for Kubernetes:

https://www.youtube.com/watch?v=F5Wyj-pbG0M

Brendan Burn's GItHub repository is an invaluable source of practical guidance through labs that will help you to see complex technology behind the paradigm of distributed systems working in the real world.

Link: *https://github.com/brendandburns*



Popular repositories

designing-distributed-systems-labs (Public) — Labs for the Designing Distributed Systems book. JavaScript ☆ 1.2k, forks 331

designing-distributed-systems (Public) — Sample code and configuration files from the Designing Distributed Systems book. Python ☆ 900, forks 263

gcp-live-k8s-visualizer (Public) — Source code for the Kubernetes visualizer used in the GCP Live Container Engine Demo. JavaScript ☆ 215, forks 134

metaparticle (Public archive) — A new pattern for distributed system development. JavaScript ☆ 186, forks 19

configula (Public) — Experimental configuration generator inspired by JSX. Go ☆ 140, forks 7

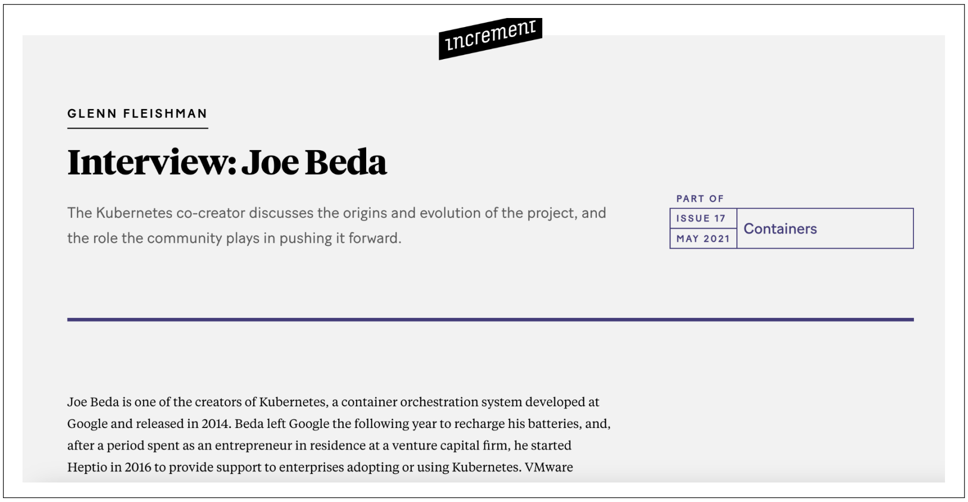
ksql (Public) — Query Kubernetes API objects using SQL. JavaScript ☆ 131, forks 9

Figure 7-3. Brendan Burns GitHub repository

Joe Beda: Joe's insights cover a wide range of topics from basic functionalities to advanced features. His contributions often reflect forward-thinking perspectives on Kubernetes development and emerging trends, making him a valuable source for those looking to stay ahead in the field.

The following interview to Joe Beda talking about the community and how it was key for Kubernetes adoption is a recommended read.

Link: *https://increment.com/containers/joe-beda-interview/*



GLENN FLEISHMAN

Interview: Joe Beda

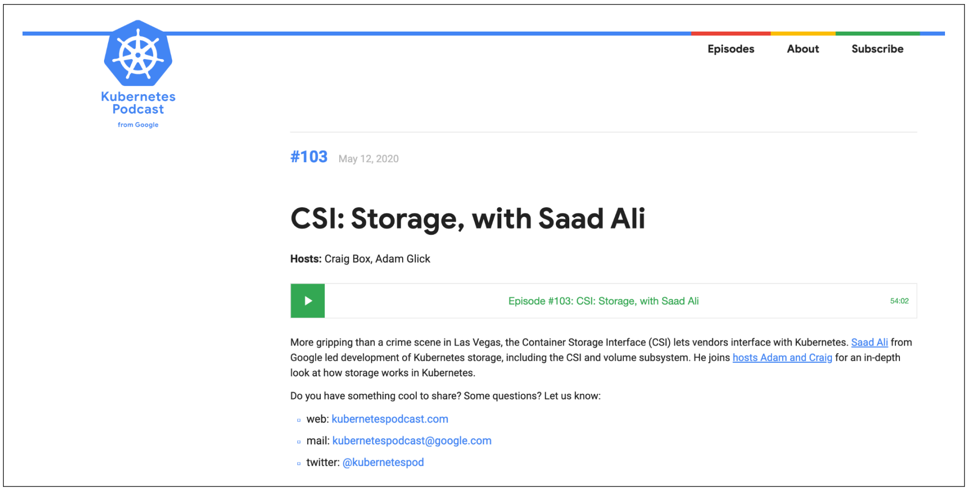
The Kubernetes co-creator discusses the origins and evolution of the project, and the role the community plays in pushing it forward.

Joe Beda is one of the creators of Kubernetes, a container orchestration system developed at Google and released in 2014. Beda left Google the following year to recharge his batteries, and, after a period spent as an entrepreneur in residence at a venture capital firm, he started Heptio in 2016 to provide support to enterprises adopting or using Kubernetes. VMware

Figure 7-4. Snippet of the Interview

Saad Ali: Ali stands out for his significant contributions in the area of Kubernetes storage solutions. His expertise is essential for navigating the complexities of storage in Kubernetes, offering clarity and solutions to some of the most challenging aspects of storage management within Kubernetes environments.

The following podcast is just one of the recommended listening to learn more about Kubernetes storage: *https://kubernetespodcast.com/episode/103-csi-kubernetes-storage/*



#103 May 12, 2020

CSI: Storage, with Saad Ali

Hosts: Craig Box, Adam Glick

Episode #103: CSI: Storage, with Saad Ali 54:02

More gripping than a crime scene in Las Vegas, the Container Storage Interface (CSI) lets vendors interface with Kubernetes. Saad Ali from Google led development of Kubernetes storage, including the CSI and volume subsystem. He joins hosts Adam and Craig for an in-depth look at how storage works in Kubernetes.

Do you have something cool to share? Some questions? Let us know:

- web: kubernetespodcast.com
- mail: kubernetespodcast@google.com
- twitter: @kubernetespod

Figure 7-5. Snippet of the CSI: Storage podcast

Liz Rice: A leading voice in container and Kubernetes security her expertise is crucial for understanding how to secure cloud-native environments effectively. Through her talks and writings, she provides essential insights into security best practices, making complex security concepts more accessible and understandable.

Her own web page contains links to all the content created by her, with special mention to the eBPF and Kubernetes security content.

Link: *https://www.lizrice.com/*

Liz Rice

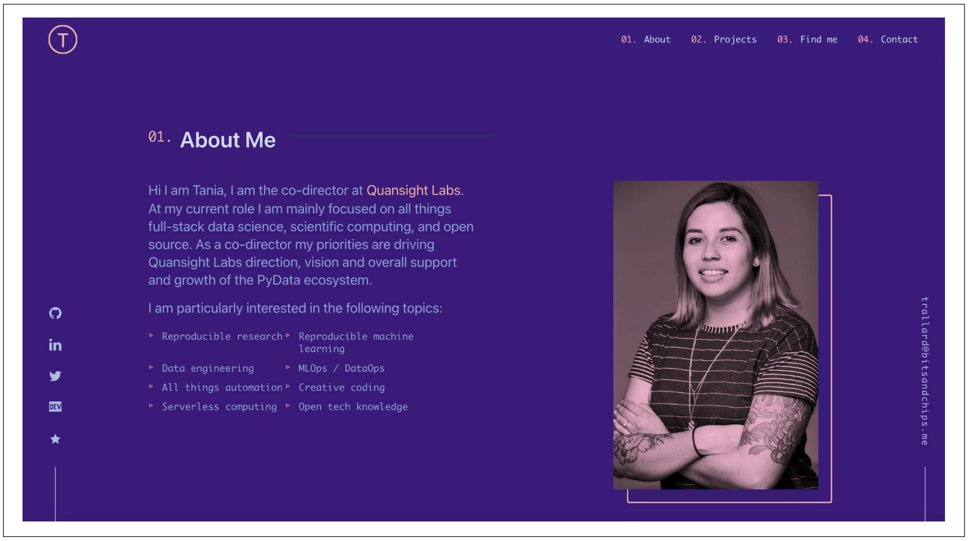
eBPF, containers, Kubernetes & security

I love making, understanding & explaining code

Figure 7-6. Liz's home page

Tania Allard: As a senior developer advocate, Allard is a key source of knowledge on developing and deploying applications in Kubernetes. She actively engages with the community through social media, GitHub, and her speaking engagements at tech conferences, offering practical advice and insights into the latest Kubernetes trends and best practices.

Link: *https://www.trallard.dev/*



01. About Me

Hi I am Tania, I am the co-director at Quansight Labs. At my current role I am mainly focused on all things full-stack data science, scientific computing, and open source. As a co-director my priorities are driving Quansight Labs direction, vision and overall support and growth of the PyData ecosystem.

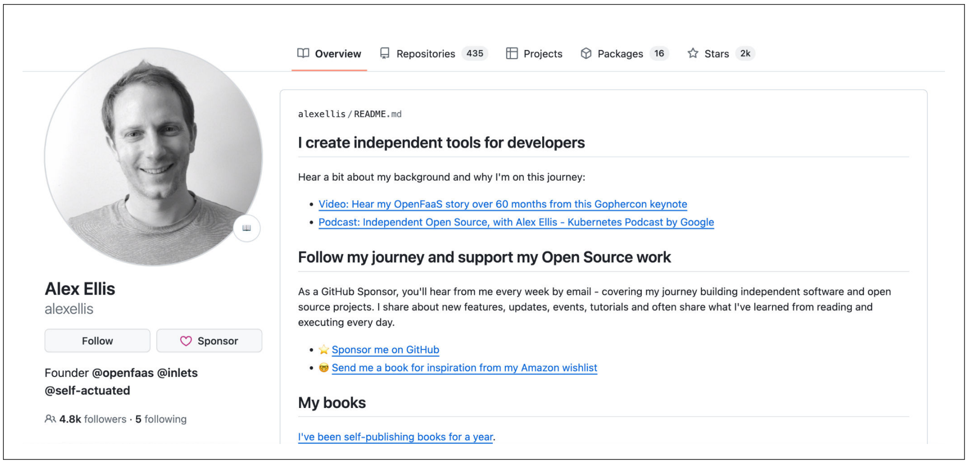
I am particularly interested in the following topics:

- Reproducible research
- Reproducible machine learning
- Data engineering
- MLOps / DataOps
- All things automation
- Creative coding
- Serverless computing
- Open tech knowledge

Figure 7-7. Tania's presentation on her web.

Alex Ellis: As the founder of OpenFaaS, Alex is a well-known figure in serverless architectures and Kubernetes. He regularly blogs and posts on social media about the intersections of Kubernetes with cloud-native applications and serverless technologies, offering valuable insights for those interested in modern, scalable application development.

Link: *https://github.com/alexellis*



Alex Ellis
alexellis

Founder @openfaas @inlets @self-actuated

4.8k followers · 5 following

alexellis / README.md

I create independent tools for developers

Hear a bit about my background and why I'm on this journey:

- Video: Hear my OpenFaaS story over 60 months from this Gophercon keynote
- Podcast: Independent Open Source, with Alex Ellis - Kubernetes Podcast by Google

Follow my journey and support my Open Source work

As a GitHub Sponsor, you'll hear from me every week by email - covering my journey building independent software and open source projects. I share about new features, updates, events, tutorials and often share what I've learned from reading and executing every day.

- ⭐ Sponsor me on GitHub
- 🤓 Send me a book for inspiration from my Amazon wishlist

My books

I've been self-publishing books for a year.

Figure 7-8. Alex's GitHub repository

James Spurin: James Spurin is a respected and influential figure in the Kubernetes and DevOps communities. His educational efforts, public speaking, and active participation in the community contribute significantly to the understanding and advancement of Kubernetes and related practices.

Link: *https://www.diveinto.com/p/home*

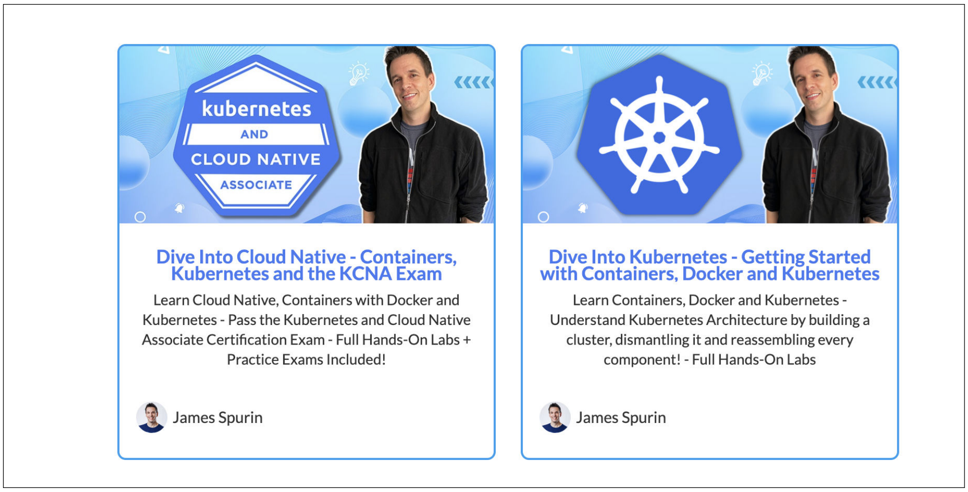

kubernetes
AND
CLOUD NATIVE
ASSOCIATE
Dive Into Cloud Native - Containers, Kubernetes and the KCNA Exam
Learn Cloud Native, Containers with Docker and Kubernetes - Pass the Kubernetes and Cloud Native Associate Certification Exam - Full Hands-On Labs + Practice Exams Included!
James Spurin
Dive Into Kubernetes - Getting Started with Containers, Docker and Kubernetes
Learn Containers, Docker and Kubernetes - Understand Kubernetes Architecture by building a cluster, dismantling it and reassembling every component! - Full Hands-On Labs
James Spurin

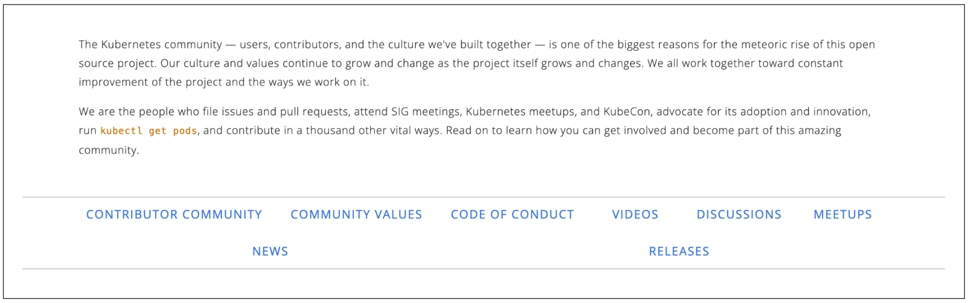
Figure 7-9. KCNA related courses

KCNA-related communities

When preparing for the Kubernetes and Cloud Native Associate (KCNA) exam, becoming involved in related communities can be incredibly beneficial. These communities provide resources, support, and networking opportunities that can enhance your learning experience. Here are some active KCNA-related communities:

Kubernetes Community (kubernetes.io/community):

This is the central hub for the Kubernetes project and the place for Special Interest Groups (SIGs), working groups, and committees. It's an excellent place for beginners to learn directly from experienced contributors, participate in discussions, and stay updated on Kubernetes developments.



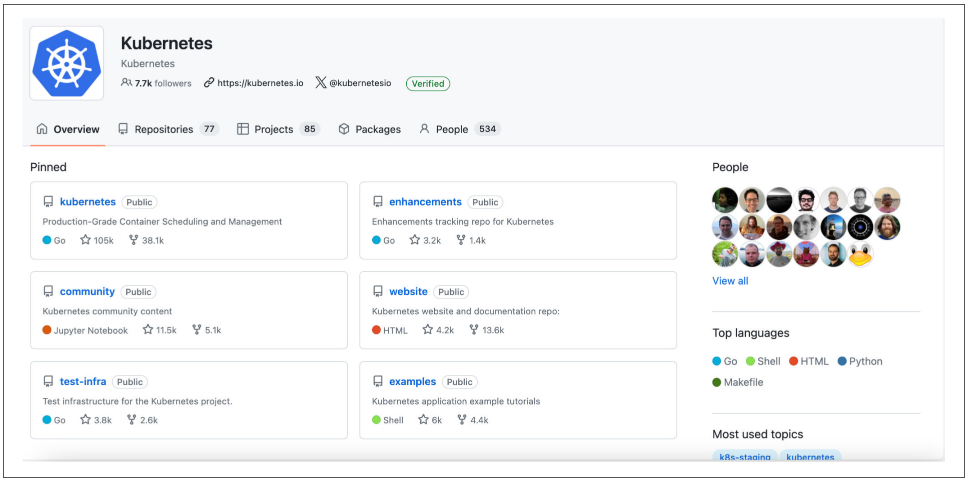
The Kubernetes community — users, contributors, and the culture we've built together — is one of the biggest reasons for the meteoric rise of this open source project. Our culture and values continue to grow and change as the project itself grows and changes. We all work together toward constant improvement of the project and the ways we work on it.

We are the people who file issues and pull requests, attend SIG meetings, Kubernetes meetups, and KubeCon, advocate for its adoption and innovation, run `kubectl get pods`, and contribute in a thousand other vital ways. Read on to learn how you can get involved and become part of this amazing community.

Figure 7-10. Kubernetes Community manifesto.

Kubernetes Slack Channels (slack.k8s.io): These channels are great for real-time communication with Kubernetes professionals and enthusiasts. It's a platform for asking questions, sharing experiences, and getting tips and advice from community members.

GitHub Kubernetes Community (github.com/kubernetes): GitHub hosts the source code and documentation for Kubernetes. Contributing to Kubernetes on GitHub or even just following the repositories can provide deep insights into how the project evolves.



Kubernetes
Kubernetes
7.7k followers https://kubernetes.io @kubernetesio Verified

Overview | Repositories 77 | Projects 85 | Packages | People 534

Pinned

kubernetes Public
Production-Grade Container Scheduling and Management
Go 105k 38.1k

enhancements Public
Enhancements tracking repo for Kubernetes
Go 3.2k 1.4k

community Public
Kubernetes community content
Jupyter Notebook 11.5k 5.1k

website Public
Kubernetes website and documentation repo:
HTML 4.2k 13.6k

test-infra Public
Test infrastructure for the Kubernetes project.
Go 3.8k 2.6k

examples Public
Kubernetes application example tutorials
Shell 6k 4.4k

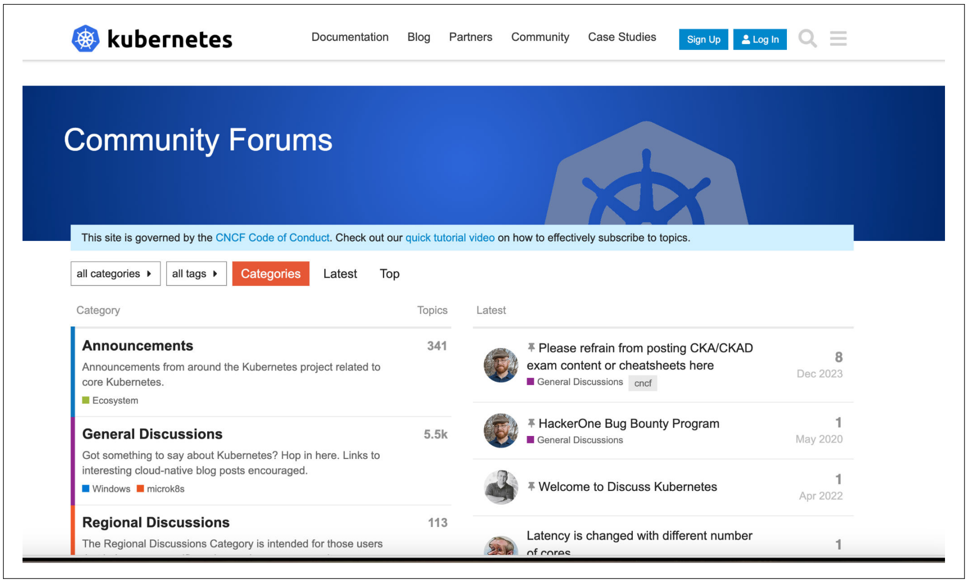
People
View all

Top languages
Go Shell HTML Python
Makefile

Most used topics
k8s-staging kubernetes

Figure 7-11. Kubernetes GitHub repository home page

Kubernetes Forums (discuss.kubernetes.io): The Kubernetes forums are an official place for discussions about Kubernetes. The community here is welcoming to questions from beginners and is a good place for detailed discussions.



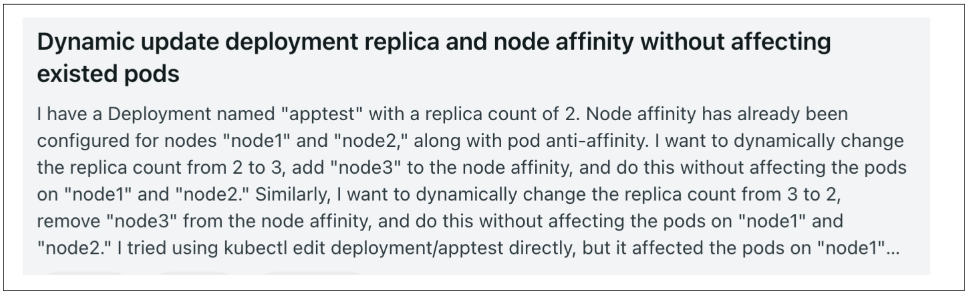
This site is governed by the CNCF Code of Conduct. Check out our quick tutorial video on how to effectively subscribe to topics.

Category	Topics
Announcements Announcements from around the Kubernetes project related to core Kubernetes. Ecosystem	341
General Discussions Got something to say about Kubernetes? Hop in here. Links to interesting cloud-native blog posts encouraged. Windows microk8s	5.5k
Regional Discussions The Regional Discussions Category is intended for those users	113

Latest		
Please refrain from posting CKA/CKAD exam content or cheatsheets here General Discussions cncf	8	Dec 2023
HackerOne Bug Bounty Program General Discussions	1	May 2020
Welcome to Discuss Kubernetes	1	Apr 2022
Latency is changed with different number of cores	1	

Figure 7-12. K8S Community Forums home page

Reddit Kubernetes Community (reddit.com/r/kubernetes): The Kubernetes subreddit is a place for news, articles, and discussions related to Kubernetes. It's a more informal setting where members share news, tips, and ask for advice.



Dynamic update deployment replica and node affinity without affecting existed pods

I have a Deployment named "apptest" with a replica count of 2. Node affinity has already been configured for nodes "node1" and "node2," along with pod anti-affinity. I want to dynamically change the replica count from 2 to 3, add "node3" to the node affinity, and do this without affecting the pods on "node1" and "node2." Similarly, I want to dynamically change the replica count from 3 to 2, remove "node3" from the node affinity, and do this without affecting the pods on "node1" and "node2." I tried using kubectl edit deployment/apptest directly, but it affected the pods on "node1"...

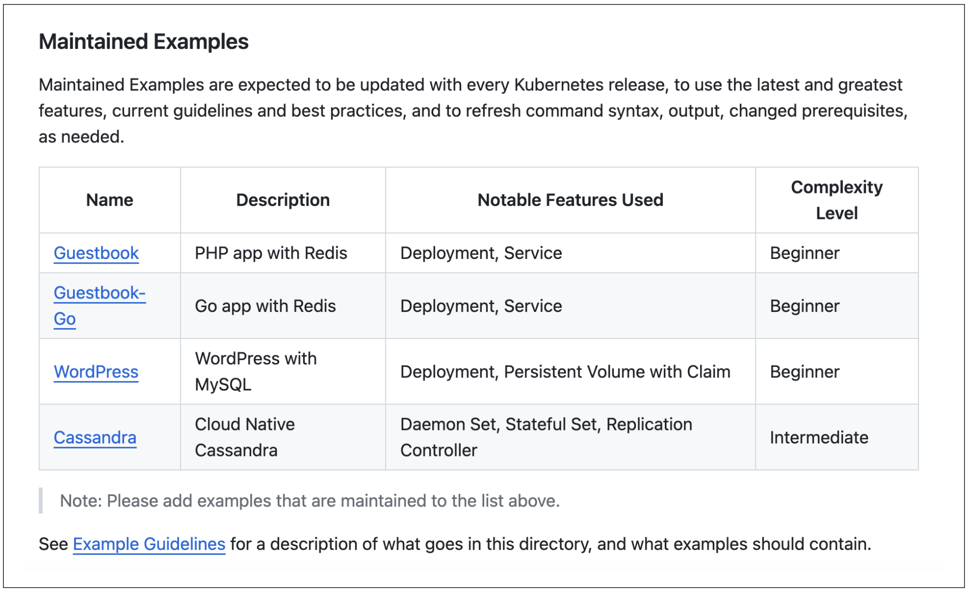
Figure 7-13. Example of a question posted in the Reddit K8S Community

Useful Github repositories

For students preparing for the Kubernetes and Cloud Native Associate (KCNA) exam, GitHub hosts a variety of repositories that are useful for study and practice. Here's a list of some repositories that could provide support during your learning journey:

kubernetes/kubernetes (github.com/kubernetes/kubernetes): This is the repository for the Kubernetes source code. Exploring this repository helps in understanding how Kubernetes is built and works under the hood.

kubernetes/examples (github.com/kubernetes/examples): This repository contains a set of examples and use cases for Kubernetes. It's great for seeing how to deploy and manage applications in a Kubernetes environment.



Maintained Examples

Maintained Examples are expected to be updated with every Kubernetes release, to use the latest and greatest features, current guidelines and best practices, and to refresh command syntax, output, changed prerequisites, as needed.

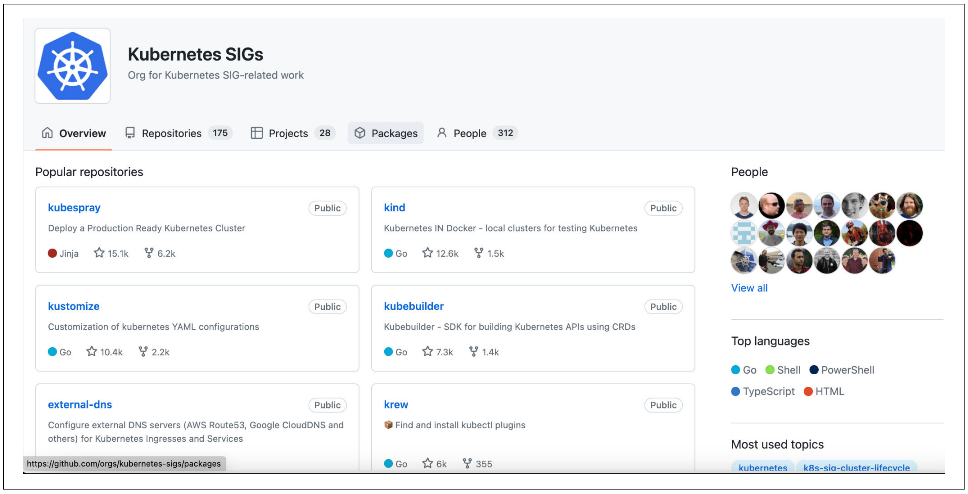
Name	Description	Notable Features Used	Complexity Level
Guestbook	PHP app with Redis	Deployment, Service	Beginner
Guestbook-Go	Go app with Redis	Deployment, Service	Beginner
WordPress	WordPress with MySQL	Deployment, Persistent Volume with Claim	Beginner
Cassandra	Cloud Native Cassandra	Daemon Set, Stateful Set, Replication Controller	Intermediate

> Note: Please add examples that are maintained to the list above.

See Example Guidelines for a description of what goes in this directory, and what examples should contain.

Figure 7-14. Maintend Examples in the repository

kubernetes-sigs (github.com/kubernetes-sigs): The Kubernetes SIGs (Special Interest Groups) have their repositories under this organization. These repositories contain projects and tools related to specific areas within Kubernetes.



Kubernetes SIGs
Org for Kubernetes SIG-related work

Overview | Repositories 175 | Projects 28 | Packages | People 312

Popular repositories

kubespray Public
Deploy a Production Ready Kubernetes Cluster
● Jinja ☆ 15.1k ⑂ 6.2k

kind Public
Kubernetes IN Docker - local clusters for testing Kubernetes
● Go ☆ 12.6k ⑂ 1.5k

kustomize Public
Customization of kubernetes YAML configurations
● Go ☆ 10.4k ⑂ 2.2k

kubebuilder Public
Kubebuilder - SDK for building Kubernetes APIs using CRDs
● Go ☆ 7.3k ⑂ 1.4k

external-dns Public
Configure external DNS servers (AWS Route53, Google CloudDNS and others) for Kubernetes Ingresses and Services

krew Public
Find and install kubectl plugins
● Go ☆ 6k ⑂ 355

https://github.com/orgs/kubernetes-sigs/packages

People
View all

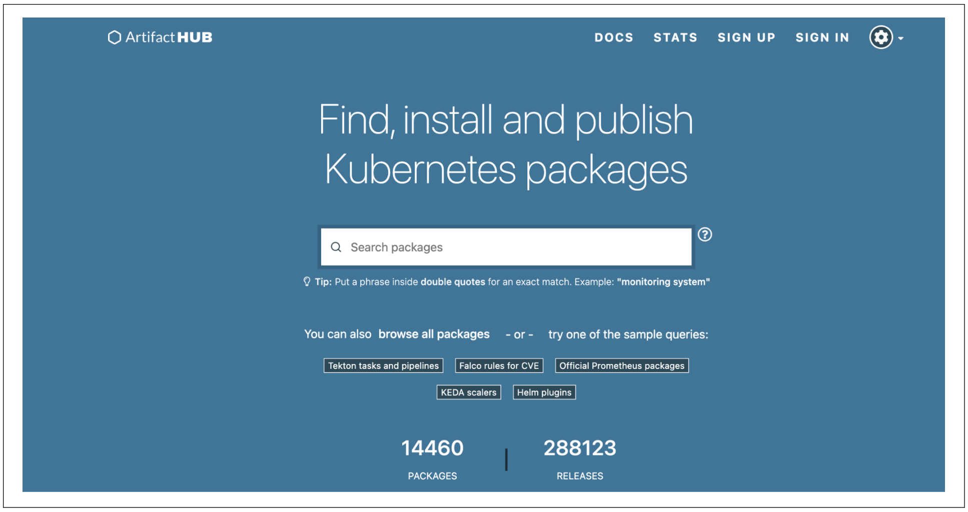
Top languages
● Go ● Shell ● PowerShell
● TypeScript ● HTML

Most used topics
kubernetes k8s-sig-cluster-lifecycle

Figure 7-15. Kubernetes SIG GItHub Home Page

Artifact Hub (*https://artifacthub.io/*): is a web-based platform for discovering, installing, and sharing packages and configurations related to CNCF (Cloud Native Computing Foundation) projects.

These offerings encompass a wide range of items, including Helm charts, Falco configurations, Open Policy Agent (OPA) and Gatekeeper policies, OLM (Operator Lifecycle Manager) operators, Tinkerbell actions, kubectl plugins, Tekton tasks and pipelines, KEDA scalers, CoreDNS plugins, Keptn integrations, container images, Kubewarden policies, Kyverno policies, the Knative client, Backstage plugins, Argo templates, KubeArmor policies, KCL (Kubernetes Client Library) modules, and the Headlamp plugin.



Find, install and publish Kubernetes packages

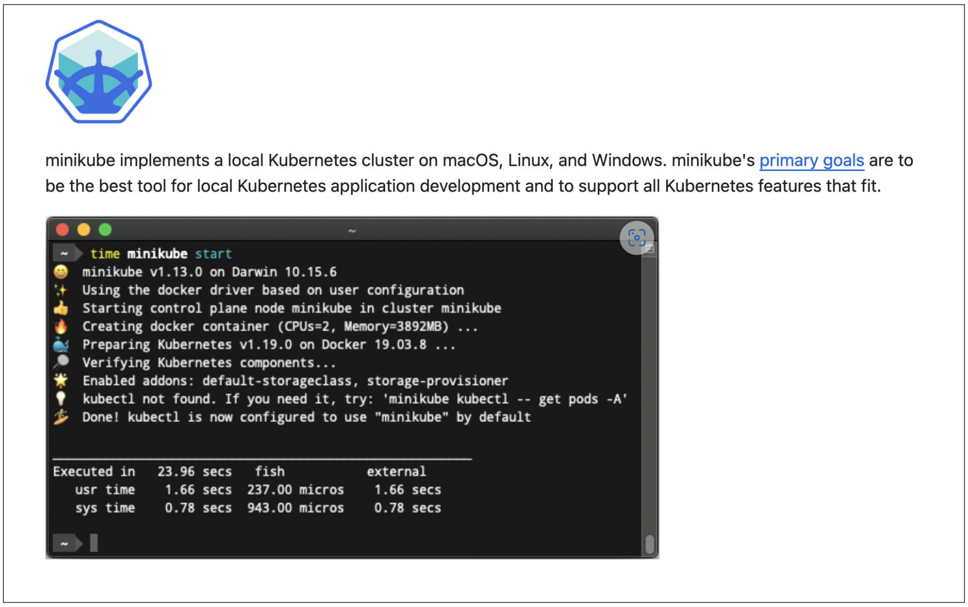
Tip: Put a phrase inside **double quotes** for an exact match. Example: **"monitoring system"**

You can also browse all packages - or - try one of the sample queries:

14460 PACKAGES | 288123 RELEASES

Figure 7-16. Artifact Hub Home page

kubernetes/minikube (github.com/kubernetes/minikube): Minikube is an essential tool for local Kubernetes development and testing. This repository contains the source code and documentation for setting up a local Kubernetes cluster.

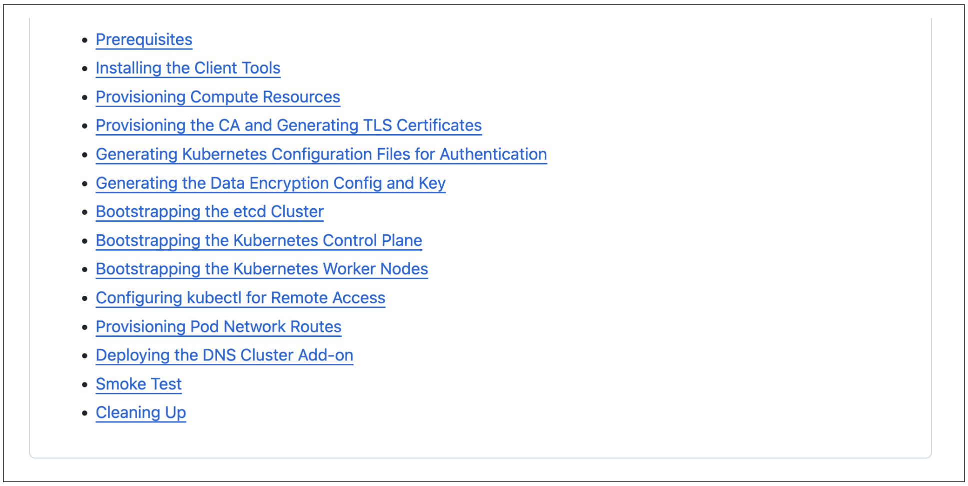


minikube implements a local Kubernetes cluster on macOS, Linux, and Windows. minikube's primary goals are to be the best tool for local Kubernetes application development and to support all Kubernetes features that fit.

```
~ time minikube start
😄 minikube v1.13.0 on Darwin 10.15.6
✨ Using the docker driver based on user configuration
👍 Starting control plane node minikube in cluster minikube
🔥 Creating docker container (CPUs=2, Memory=3892MB) ...
🐳 Preparing Kubernetes v1.19.0 on Docker 19.03.8 ...
🔎 Verifying Kubernetes components...
🌟 Enabled addons: default-storageclass, storage-provisioner
💡 kubectl not found. If you need it, try: 'minikube kubectl -- get pods -A'
🏄 Done! kubectl is now configured to use "minikube" by default

________________________________________________________
Executed in   23.96 secs   fish         external
   usr time    1.66 secs  237.00 micros    1.66 secs
   sys time    0.78 secs  943.00 micros    0.78 secs

~ ▌
```

Figure 7-17. Minikube project Home Page

kelseyhightower/kubernetes-the-hard-way (github.com/kelseyhightower/kubernetes-the-hard-way): Created by Kelsey Hightower (discussed above), this repository provides a set of exercises to set up Kubernetes from scratch. It's one of the best ways to deeply understand how the components of Kubernetes fit together and deploy every building block and stitch them together.



- Prerequisites
- Installing the Client Tools
- Provisioning Compute Resources
- Provisioning the CA and Generating TLS Certificates
- Generating Kubernetes Configuration Files for Authentication
- Generating the Data Encryption Config and Key
- Bootstrapping the etcd Cluster
- Bootstrapping the Kubernetes Control Plane
- Bootstrapping the Kubernetes Worker Nodes
- Configuring kubectl for Remote Access
- Provisioning Pod Network Routes
- Deploying the DNS Cluster Add-on
- Smoke Test
- Cleaning Up

Figure 7-18. - Summary of the topics to practice

cncf/landscape (*https://landscape.cncf.io/*): The Cloud Native Computing Foundation's landscape offers a comprehensive overview of the cloud-native landscape, including tools, services, and technologies associated with Kubernetes.

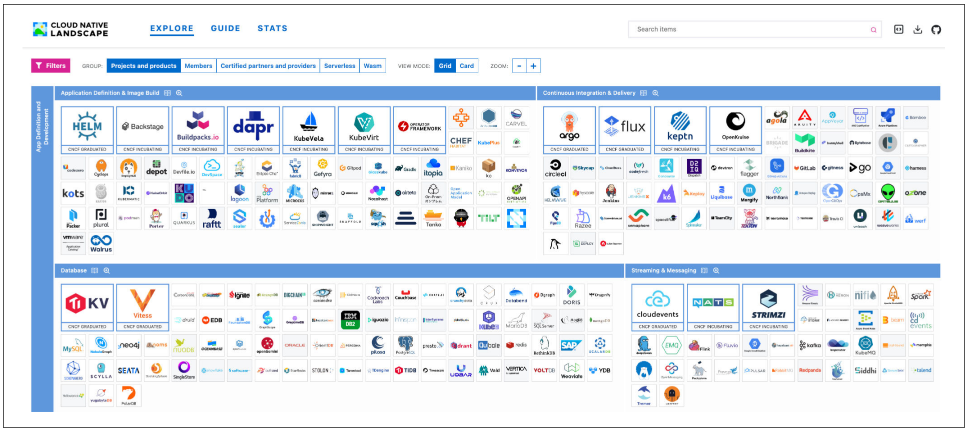


Figure 7-19. CNCF Landscape Snippet

Potential career options for KCNA certified

Earning a Kubernetes and Cloud Native Associate (KCNA) certification opens up a range of career opportunities in the rapidly growing field of cloud-native tech.

This certification is an attestation that you received a foundational understanding of Kubernetes and its ecosystem. It positions you for various roles in organizations adopting modern, scalable, and efficient software practices.

Here is an incomplete list of some of the applicable roles for those with the KCNA certification:

- DevOps Engineer: This role specializes in automating, improving, and optimizing development and deployment processes. DevOps Engineers with Kubernetes knowledge are in high demand to manage containerized applications and cloud-native solutions.
- Cloud Engineer: This role focuses on designing, implementing, and maintaining cloud-based applications and infrastructure. A KCNA certifies an understanding of how Kubernetes operates within cloud environments, making it a valuable credential for this role.
- Site Reliability Engineer (SRE): This role ensures high availability and reliability of services. SREs with Kubernetes expertise can efficiently manage containerized applications, implement scalable architectures, and maintain system health in cloud-native environments.
- Kubernetes Administrator: People in this role directly manage Kubernetes clusters and their lifecycle. This role involves configuring and maintaining Kubernetes environments, ensuring their optimal performance and security.
- Systems Administrator: This role focusess on managing and operating Kubernetes clusters in addition to traditional systems administration duties. The KCNA certification provides a solid foundation for integrating Kubernetes into broader system management practices.
- Technical Support Engineer: This role provides support for Kubernetes-based applications and infrastructure. The understanding of Kubernetes fundamentals from a KCNA certification is beneficial for diagnosing and resolving issues in cloud-native environments.
- Product Manager/Technical Sales in Cloud Technologies: People in this role assist in the design, promotion, and selling of cloud-native solutions. A KCNA certification equips individuals with the necessary Kubernetes knowledge to effectively communicate the benefits of cloud-native technologies to clients or internal teams.
- Training and Education: In the context of teaching and education roles, you could develop educational materials related to Kubernetes and cloud-native technologies. This role suits those who enjoy mentoring and guiding others in their learning journey.

- Consultant or Freelancer: This role provides expert advice and services to organizations adopting Kubernetes and cloud-native technologies. This career path offers flexibility and a variety of projects, appealing to those with entrepreneurial spirits.

Having a KCNA certification demonstrates foundational knowledge and a commitment to professional development in Kubernetes and cloud-native technologies, making these individuals valuable assets in various roles within tech-forward organizations. As the demand for cloud-native skills continues to grow, these career paths offer promising opportunities for professional growth and specialization.

Other Kubernetes certifications

In addition to the Kubernetes and Cloud Native Associate (KCNA) certification, there are several other Kubernetes certifications offered by the Cloud Native Computing Foundation (CNCF) in partnership with the Linux Foundation. These certifications are designed for different levels of expertise and career paths in the Kubernetes ecosystem. Here's an overview of these key Kubernetes certifications:

Certified Kubernetes Administrator (CKA)

- Target Audience: The CKA certification is aimed at professionals responsible for the administration of Kubernetes clusters.
- Focus: It covers the skills required to install, configure, and manage Kubernetes clusters, including networking, storage, security, maintenance, logging, and monitoring.
- Suitability: Ideal for system administrators, DevOps engineers, and IT professionals who work with Kubernetes on a regular basis.

Certified Kubernetes Application Developer (CKAD)

- Target Audience: This certification is designed for software developers who wish to demonstrate their skills in designing, building, and deploying applications on Kubernetes.
- Focus: The CKAD focuses on the practical aspects of defining application resources, using core primitives to build, monitor, and troubleshoot scalable applications and tools in Kubernetes.
- Suitability: Best suited for software developers who want to specialize in building Kubernetes-native applications.

Certified Kubernetes Security Specialist (CKS)

- Target Audience: The CKS is tailored for Kubernetes administrators, engineers, and security professionals.
- Focus: It emphasizes Kubernetes security, covering cluster setup, system hardening, minimizing vulnerabilities, and ongoing security maintenance.
- Prerequisites: Candidates must have passed the CKA exam before attempting the CKS.
- Suitability: Ideal for individuals focused on the security aspect of Kubernetes clusters.

Each of these certifications requires passing a performance-based exam, which tests hands-on skills in real-world scenarios. These exams are:

- Highly Recognized: These certifications are widely recognized in the industry and are often considered benchmarks of expertise in their respective areas.
- Practical and Hands-on: Unlike many traditional certifications, these exams are practical, requiring candidates to perform tasks on live systems.
- Career-Enhancing: Earning these certifications can significantly boost career prospects, as there is a growing demand for skilled Kubernetes professionals.

These certifications not only validate your skills and knowledge in Kubernetes but also help in specializing in specific areas, whether it's administration, application development, or security. As Kubernetes continues to be a leading platform in the cloud-native ecosystem, these certifications provide significant value to IT professionals looking to advance in their careers.

Relevant books

O'Reilly Media publishes a range of comprehensive and highly regarded books that are relevant for those interested in Kubernetes, cloud-native technologies, and related subjects. Here's a list of some notable O'Reilly titles that would be particularly beneficial for those studying for Kubernetes certifications like the KCNA, CKA, CKAD, or CKS:

- Kubernetes Up & Running: Dive into the Future of Infrastructure" by Kelsey Hightower, Brendan Burns, and Joe Beda:

  Link: *https://www.oreilly.com/library/view/kubernetes-up-and/9781492046523/*

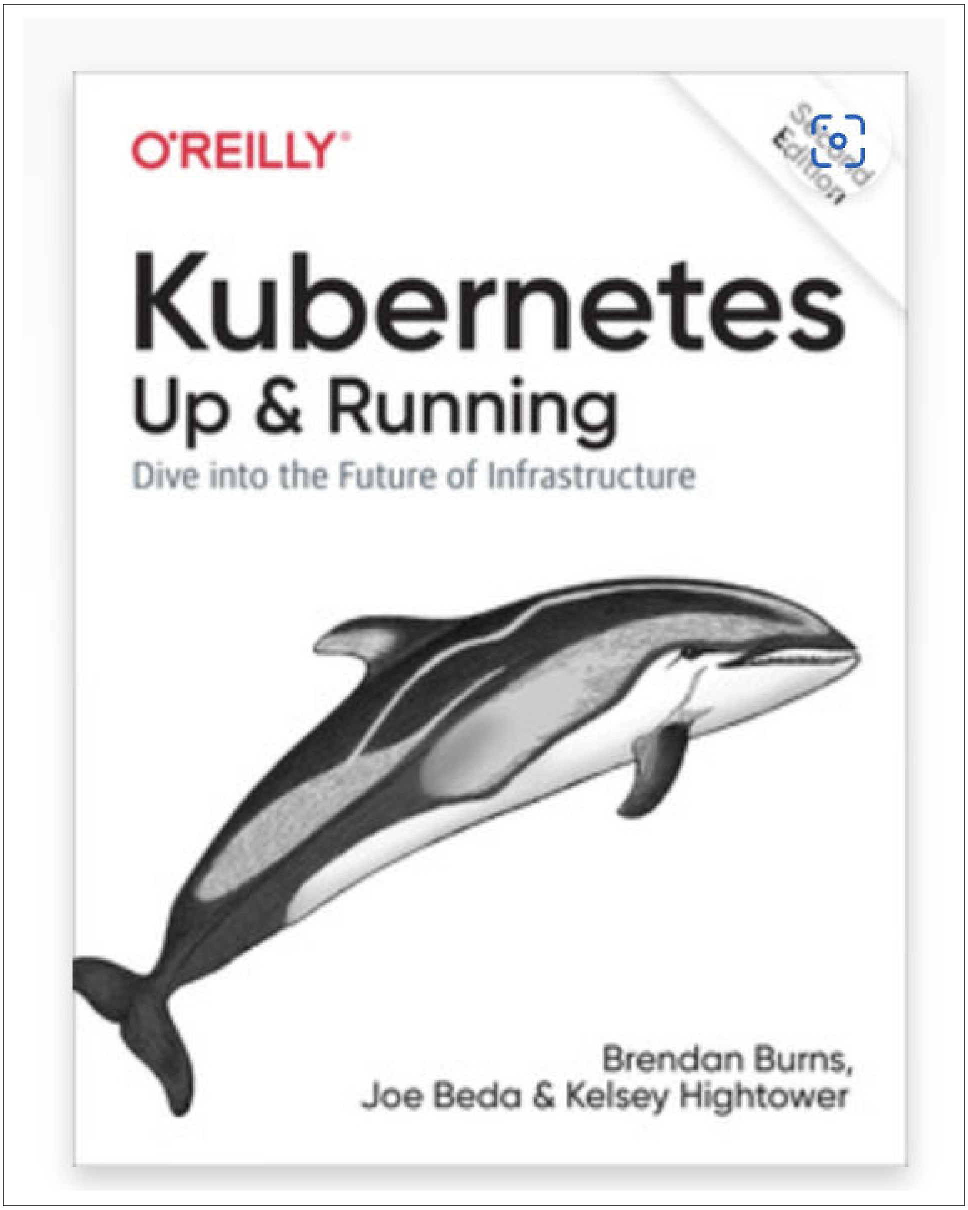

O'REILLY
Kubernetes
Up & Running
Dive into the Future of Infrastructure
Second Edition
Brendan Burns,
Joe Beda & Kelsey Hightower

Figure 7-20. Kubernetes Up & Running book cover

This foundational book provides a practical guide to Kubernetes, authored by key Kubernetes creators. It's great for understanding the architecture and operation of Kubernetes.

- “Designing Distributed Systems: Patterns and Paradigms for Scalable, Reliable Services” by Brendan Burns: Authored by one of the co-founders of Kubernetes, this book offers insights into designing and building scalable and reliable distributed systems, which is a key aspect of cloud-native applications.

  Link: *https://www.oreilly.com/library/view/designing-distributed-systems/9781491983638/*

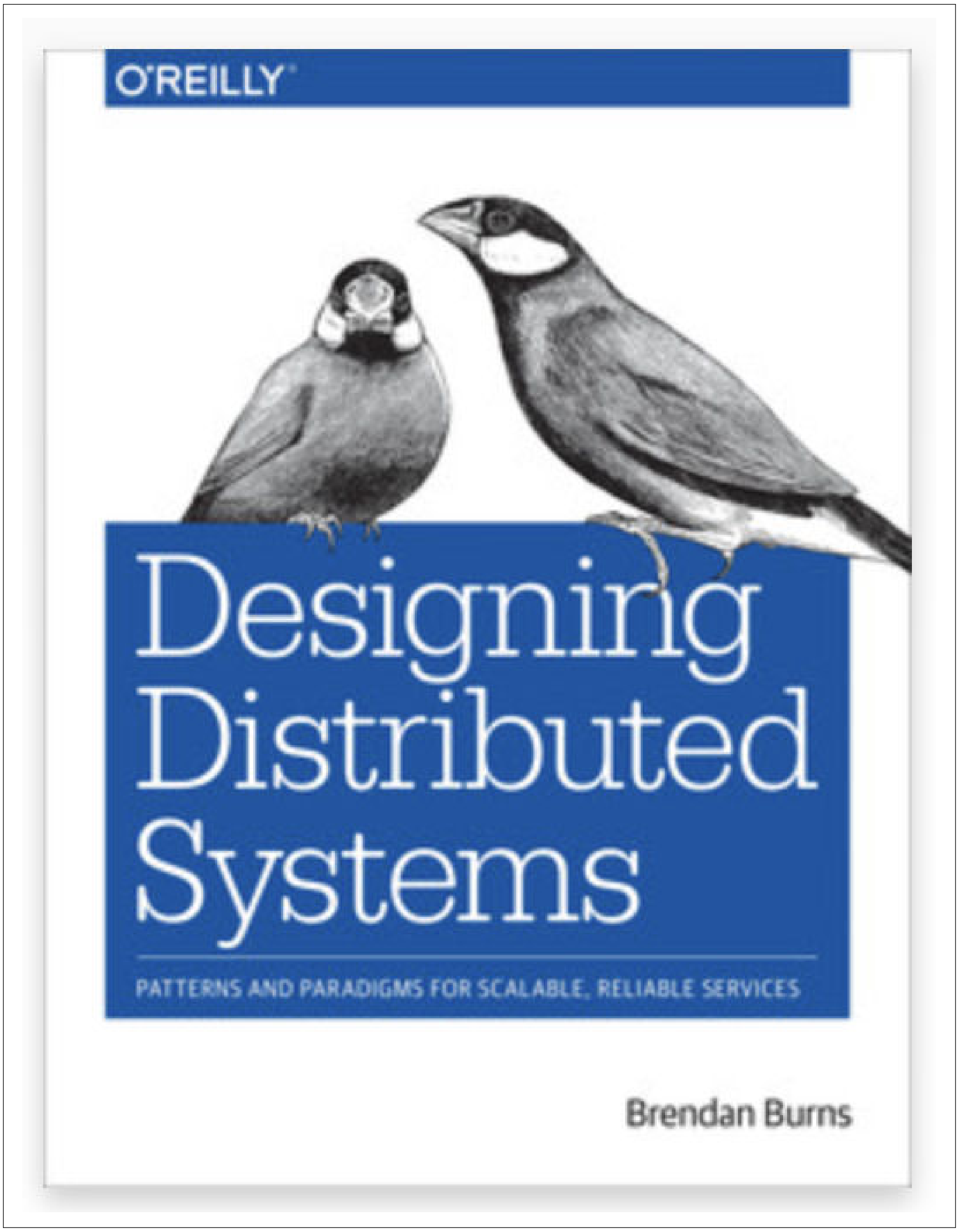

O'REILLY
Designing Distributed Systems
PATTERNS AND PARADIGMS FOR SCALABLE, RELIABLE SERVICES
Brendan Burns

Figure 7-21. Designing Distributed Systems Book Cover

- “Kubernetes Patterns: Reusable Elements for Designing Cloud-Native Applications” by Bilgin Ibryam and Roland Huß: This book delves into Kubernetes-specific patterns for application development, offering a detailed guide for architecting cloud-native applications effectively.

LInk: *https://www.oreilly.com/library/view/kubernetes-patterns/9781492050278/*

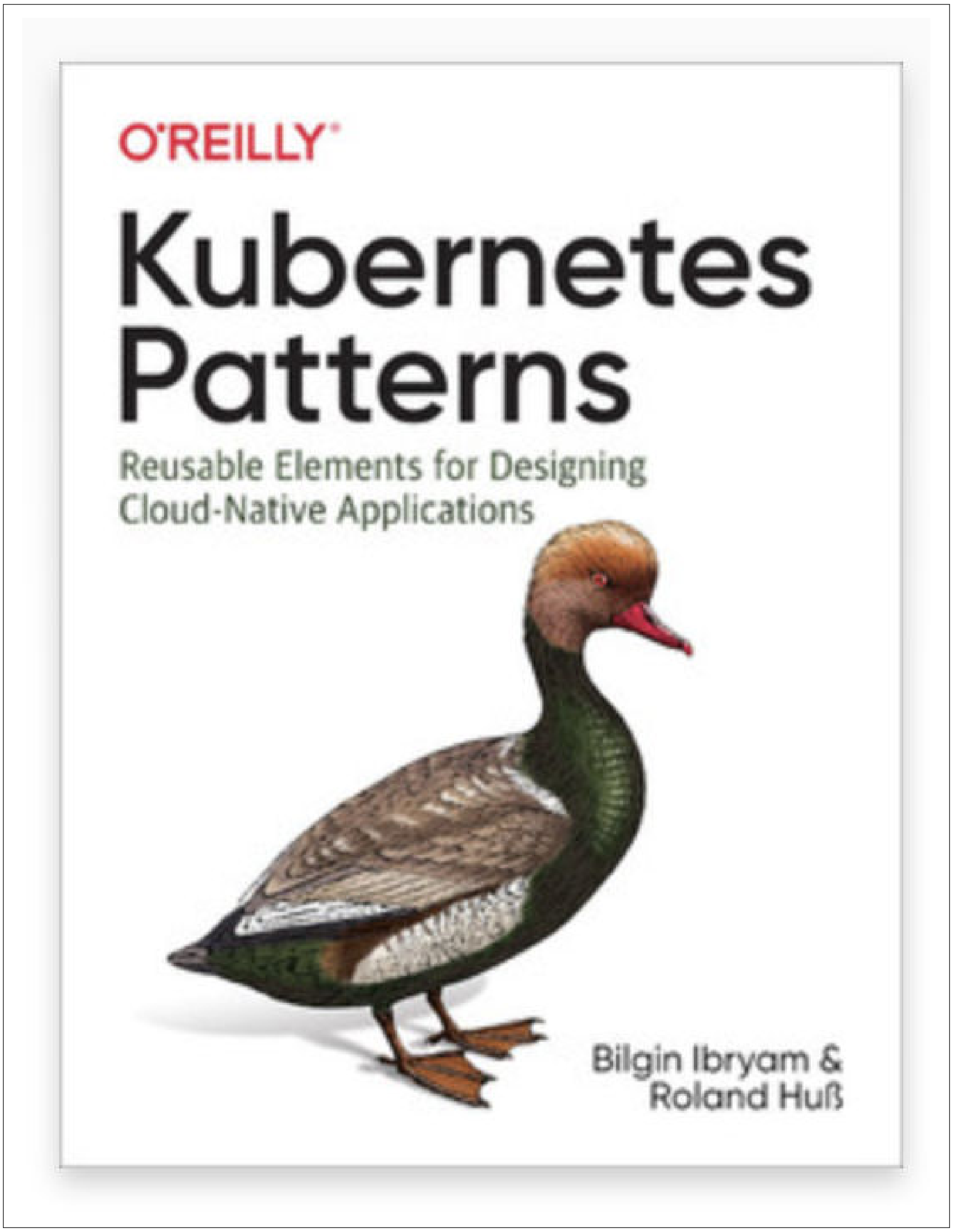

O'REILLY
Kubernetes Patterns
Reusable Elements for Designing Cloud-Native Applications
Bilgin Ibryam & Roland Huß

Figure 7-22. Book Cover

- "Cloud Native DevOps with Kubernetes: Building, Deploying, and Scaling Modern Applications in the Cloud" by John Arundel and Justin Domingus: Covering the DevOps side of Kubernetes, this book addresses continuous integration, continuous delivery, and infrastructure as code in Kubernetes environments.

  Link: *https://www.oreilly.com/library/view/cloud-native-devops/9781492040750/#:~:text=Cloud%20Native%20DevOps%20with%20Kubernetes%201%201.%20Revolution,Clusters%20Cluster%20Sizing%20and%20Scaling%20Capacity%20Planning%20*

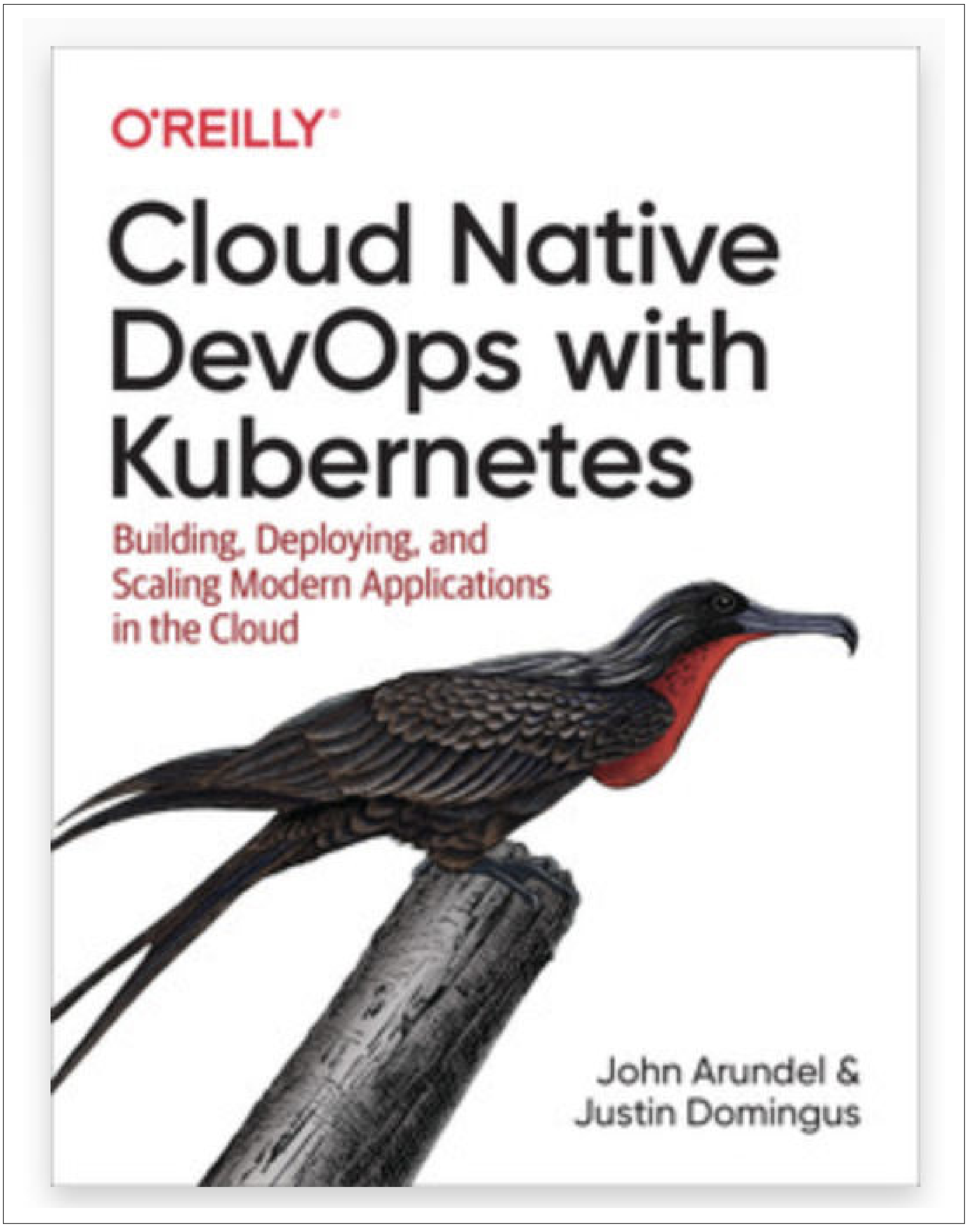

O'REILLY
Cloud Native DevOps with Kubernetes
Building, Deploying, and Scaling Modern Applications in the Cloud
John Arundel & Justin Domingus

Figure 7-23. Book Cover

- "Managing Kubernetes: Operating Kubernetes Clusters in the Real World" by Brendan Burns and Craig Tracey: Targeted at Kubernetes administrators, this book discusses the challenges and solutions for managing Kubernetes in production settings.

LInk: *https://www.oreilly.com/library/view/managing-kubernetes/9781492033905/*

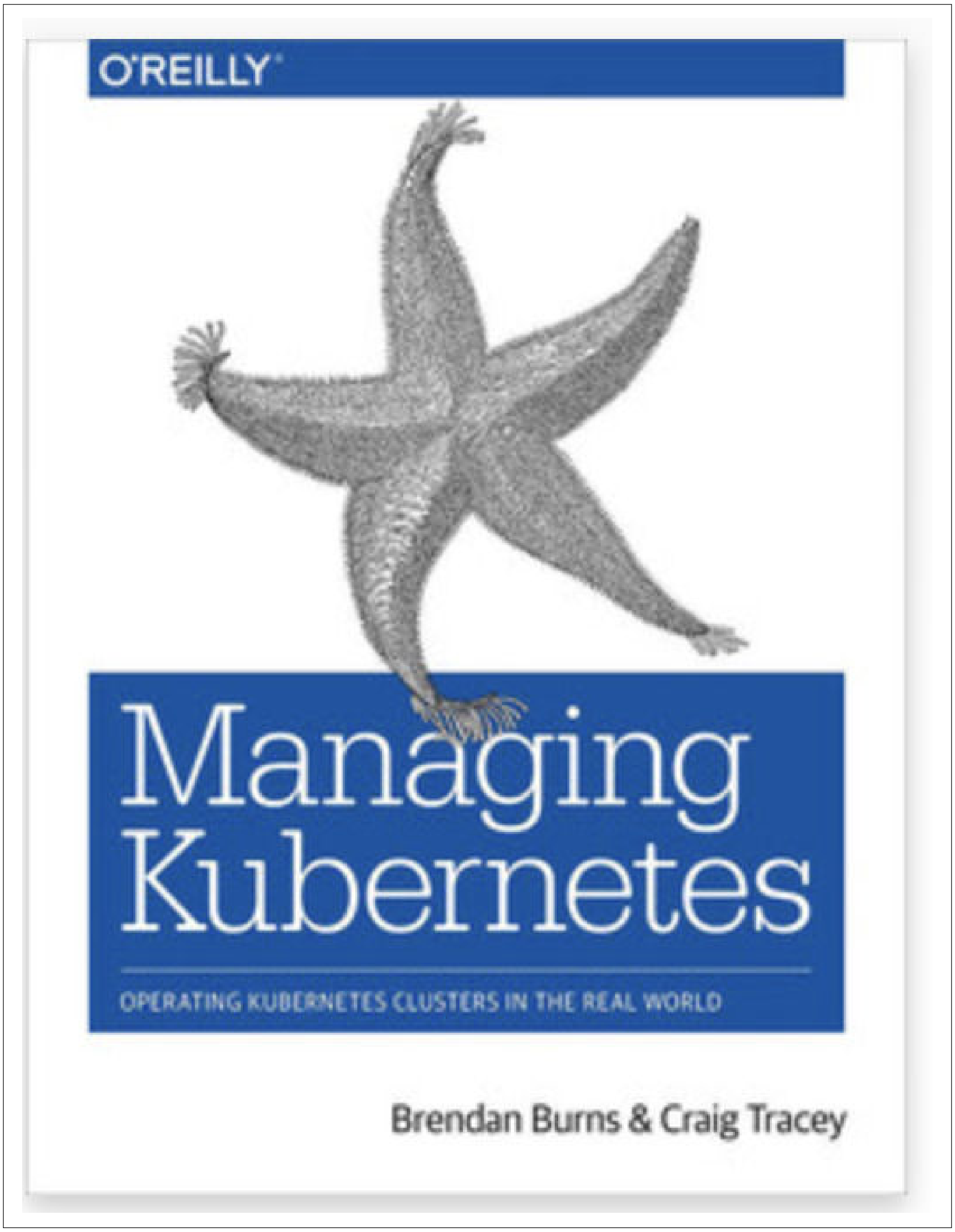

O'REILLY
Managing Kubernetes
OPERATING KUBERNETES CLUSTERS IN THE REAL WORLD
Brendan Burns & Craig Tracey

Figure 7-24. Book Cover

- "Container Security" by Liz Rice Essential for those focusing on Kubernetes security, it discusses best practices and strategies for securing containers and clusters.

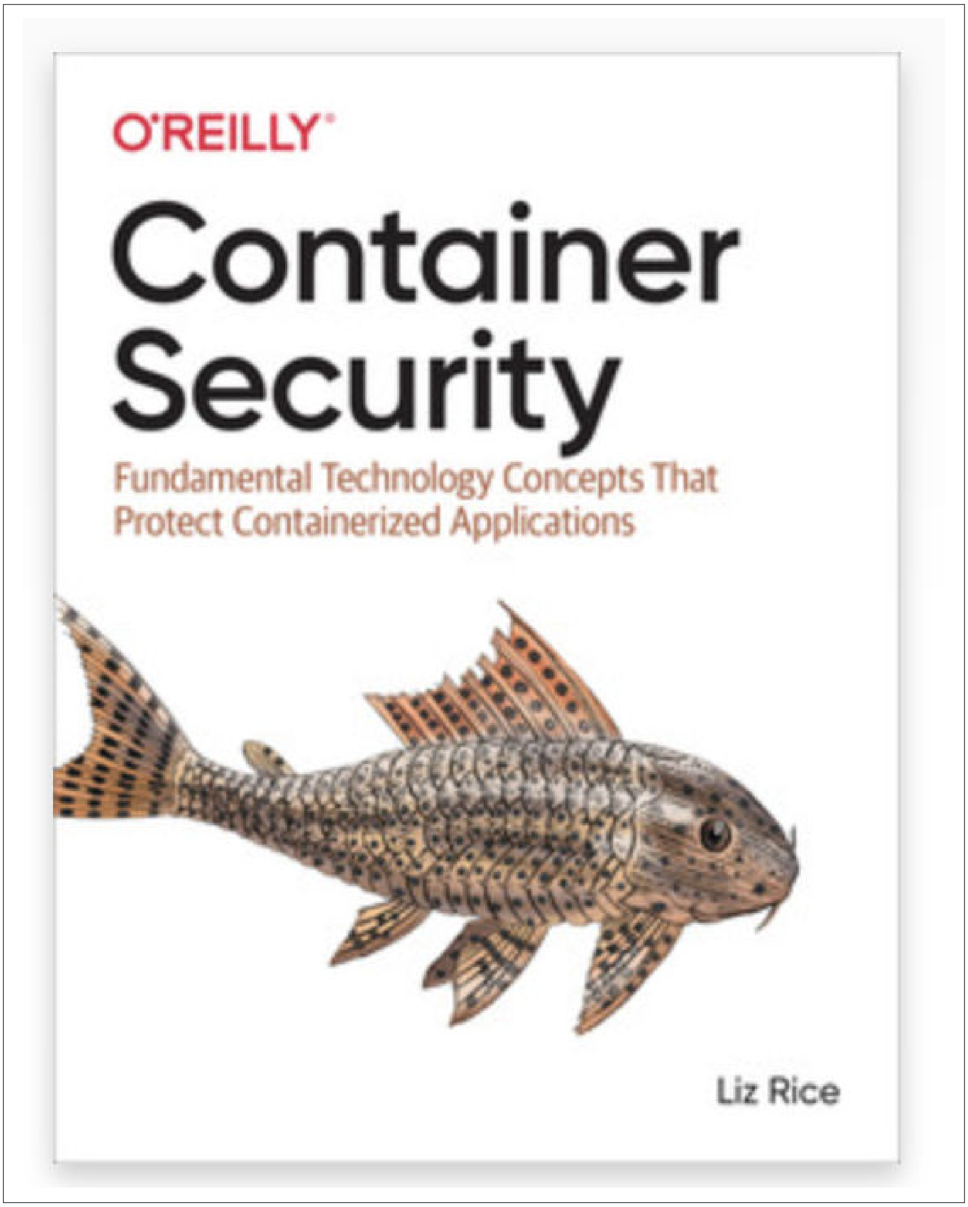

O'REILLY
Container Security
Fundamental Technology Concepts That Protect Containerized Applications
Liz Rice

Figure 7-25. Figure7-24 book cover

Summary

Congratulations! You have completed the Study Guide for KCNA.Remember, this isn't just about passing an exam; it's about mastering a transformative technology. A s you conquer KCNA, consider leveling up. The Certified Kubernetes Administrator (CKA), Certified Kubernetes Application Developer (CKAD), and Certified Kubernetes Security Specialist (CKS) certifications await. These advanced badges will propel your career skyward.

Remember this: The journey may be challenging, but the view from the summit is worth every step.

Keep pushing forward!