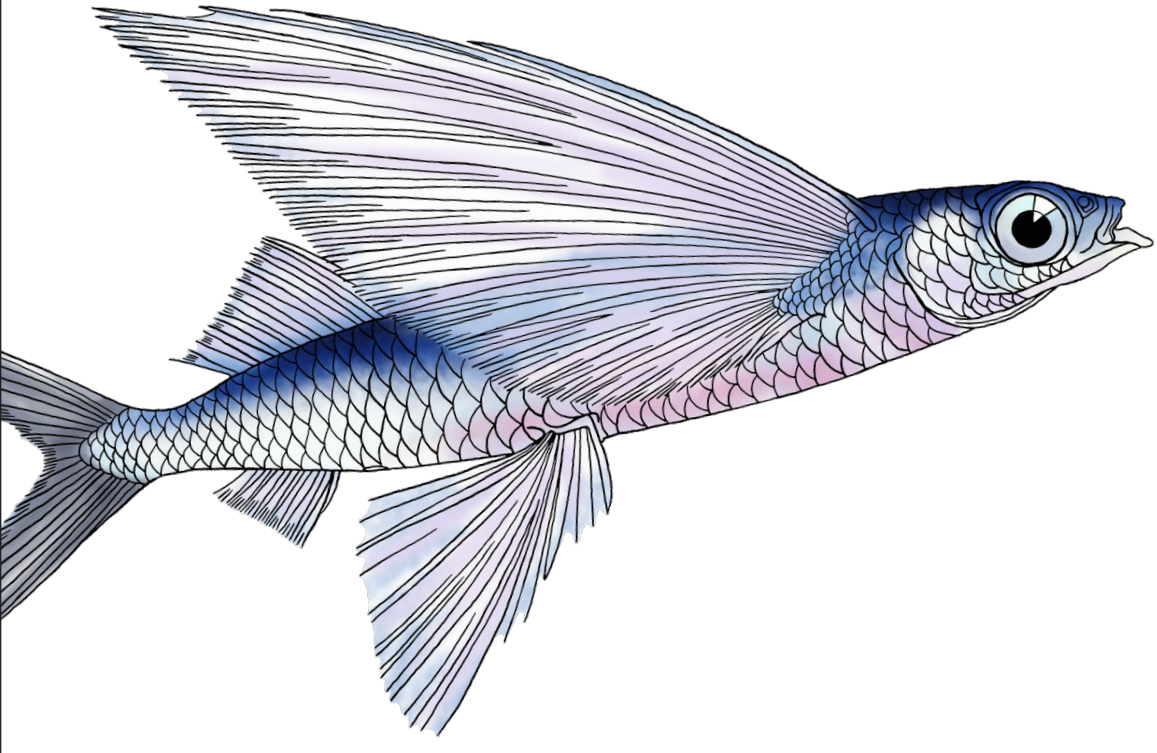
O'REILLY®

Kubernetes and Cloud Native Associate KCNA Study Guide

In-Depth Exam Prep and Practice



Adrián González Sánchez
& Jorge Valenzuela Jiménez
Foreword by Katie Gamanji

Kubernetes and Cloud Native Associate (KCNA) Study Guide

In-Depth Exam Prep and Practice

Adrián González Sánchez and Jorge Valenzuela Jiménez

Foreword by Katie Gamanji

Beijing • Boston • Farnham • Sebastopol • Tokyo

O'REILLY®

Kubernetes and Cloud Native Associate (KCNA) Study Guide
by Adrián González Sánchez and Jorge Valenzuela Jiménez

Copyright © 2024 13329054 Canada Inc. All rights reserved.

Printed in the United States of America.

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
Copyeditor: Charles Roumeliotis
Proofreader: Sharon Wilkey
Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Kate Dullea

May 2024: First Edition

Revision History for the First Edition
2024-05-29: First Release

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

The O'Reilly logo is a registered trademark of O'Reilly Media, Inc. *Kubernetes and Cloud Native Associate (KCNA) Study Guide*, the cover image, and related trade dress are trademarks of O'Reilly Media, Inc.

The views expressed in this work are those of the authors and do not represent the publisher's views. While the publisher and the authors have used good faith efforts to ensure that the information and instructions contained in this work are accurate, the publisher and the authors disclaim all responsibility for errors or omissions, including without limitation responsibility for damages resulting from the use of or reliance on this work. Use of the information and instructions contained in this work is at your own risk. If any code samples or other technology this work contains or describes is subject to open source licenses or the intellectual property rights of others, it is your responsibility to ensure that your use thereof complies with such licenses and/or rights.

978-1-098-16950-3

Table of Contents

Foreword: An Overview of the Cloud Native Ecosystem ix

Preface xiii

1. Introduction to the KCNA Certification and Study Guide 1
Defining Cloud Native 1
Inspirational Success Stories 3
Self-Questionnaire for New Cloud Native Practitioners 6
Part 1: The CNCF Ecosystem 7
Part 2: General Cloud Native Concepts 8
Part 3: Kubernetes Topics 10
Part 4: Kubernetes Commands 11
Part 5: Linux Fundamentals 13
Part 6: Other Related Projects 14
Expert Insights: Walid Shaari 18
Summary 24

2. The KCNA Exam as the Starting Point 25
The KCNA Certification by the Linux Foundation and CNCF 25
Why the KCNA Matters and Why You Should Get Certified 27
Exam Description and Curriculum 29
About the Format 30
Part 1: Kubernetes Fundamentals 31
Part 2: Container Orchestration 32
Part 3: Cloud Native Architecture 32
Part 4: Cloud Native Observability 33
Part 5: Cloud Native Application Delivery 33
Additional Areas of Knowledge 35

Kubernetes 101 Hands-on 35
Linux Fundamentals 35
API-Enabled Development 36
Telematics 36
Web Development 36
Applied Usage 37
Exam Study Plan 38
Step 1: Determine your Current Level of Knowledge 38
Step 2: Prepare and Adopt a Study Schedule 38
Step 3: Focus on the Right Material 39
Exam Logistics and What to Expect 39
Step 1: Schedule Your Exam 39
Step 2: Prepare for Exam Day 42
Step 3: Wait and Check Results 44
Potential Certification Paths 44
Case Study: Spotify 48
End-of-Chapter Knowledge Check 49
Summary 50

3. CNCF and the New Era of Cloud Native. 51
The Origins of CNCF 51
Timeline of the Cloud Native Industry 52
Containerization Origins 54
The Rise of Kubernetes and CNCF 55
Community Dynamics and Governance 56
Key CNCF Resources 58
Community Forums 59
In-Person and Online Events 60
Educational Resources 61
Mentorship and Ambassadors 65
CNCF Projects and Maturity Levels 65
Main CNCF Projects for the KCNA Exam 67
Related Initiatives and Work Groups 75
Expert Insights: Chris Aniszczyk 76
End-of-Chapter Knowledge Check 82
Summary 83

4. Essential Concepts for Cloud Native Practitioners. 85
General Cloud Computing 85
Commercialization of Cloud Computing 88
Infrastructure as a Service 89
Platform as a Service 90

Software as a Service 90
Container as a Service 91
Function as a Service 91
Cloud Computing Infrastructure 92
Public Cloud 92
Private Cloud 92
Hybrid Cloud 92
Multicloud 93
The Virtualization, Containerization, and Orchestration Trifecta 93
Virtualization 94
Containerization 94
Orchestration 96
Relationship Between Cloud Computing and Cloud Native 97
Building Cloud Native Applications 98
Other Relevant Cloud Native Topics 100
DevOps and CI/CD 100
GitOps and Infrastructure as Code 102
Declarative Versus Imperative 102
Stateful Versus Stateless 103
Egress and Ingress 103
Serverless 104
Expert Insights: Benjamin Muschko 104
Summary 113

5. The Key Role of Kubernetes. 115
An Introduction to Containers and Orchestration Technologies 116
Containers 117
Container Orchestration 121
Container Ecosystem and Standards 122
Linux Container 122
Docker 123
What Do the Container Standards Provide? 124
Container Solutions 125
What Is Kubernetes, and Why Is It So Important? 126
Current Industry Trends 128
Artificial Intelligence 129
Vector Databases 130
Edge Computing 131
Platform Engineering 132
Orchestration Alternatives to Kubernetes 133
Apache Mesos and Marathon 133
Red Hat OpenShift 134

Containerization Benefits for Businesses 135
Expert Insights: Brendan Burns 136
Summary 141

6. Technical Kubernetes Topics. 143
Kubernetes Basic Concepts: From Containers to Clusters 145
Kubernetes Cluster 145
Components of the Control Plane 149
Components of the Data Plane 153
Inside Kubernetes Architecture 155
Single-Container and Multi-Container Pods 156
Container Runtimes in Kubernetes 160
Container Network Interface 163
Container Storage Interface 163
Navigating Kubernetes States 163
Stateless Applications 163
Stateful Applications 164
Deployments: Orchestrating Stateless Scalability 164
StatefulSets: The Custodian of Stateful Applications 164
ReplicaSets: Safeguarding Replication and Reliability 164
DaemonSet Controller 165
Job and CronJob Controllers 165
Workload Autoscaling, Kubernetes Events, and Pod Lifecycles 165
Horizontal Pod Autoscaler 166
Vertical Pod Autoscaler 166
Cluster Proportional Autoscaler 167
Autoscaling Based on Events 167
Cluster Autoscaler 167
Scheduler 168
Kubernetes Observability and Performance 171
Logs 173
Metrics 173
Traces 174
Resource Metrics Pipeline 175
Full Metrics Pipeline 175
Kubernetes Storage 178
Persistent Volumes and Persistent Volume Claims 178
Storage Classes 178
Volumes and Volume Types 179
StatefulSets 179
Container Storage Interface 179
Dynamic Provisioning 180

Networking and Service Mesh 180
Cluster Networking 180
Ingress Controllers 181
Network Plug-ins 182
Service Discovery 183
Service Mesh 184
Kubernetes Security 187
Container Image Security Strategies 188
Kubernetes Cluster Protection Strategies 189
Other Kubernetes-Related Components 190
Helm 190
Dapr 191
ConfigMap 192
minikube 193
Main Commands 193
Expert Insights: James Spurin 198
Summary 203

7. Final Recommendations. 205
Additional Information for Exam Preparation 205
Kubernetes Community Experts to Follow 206
KCNA-Related Communities 216
Useful GitHub Repositories 218
Potential Career Options with KCNA Certification 222
Other Kubernetes Certifications 224
Relevant Books 226
Summary 231

A. End-of-Book Knowledge Check. 233

B. Solutions to Self-Questionnaire and Knowledge Checks. 235

Foreword: An Overview of the Cloud Native Ecosystem

The year 2024 marks the 10th anniversary of Kubernetes. It is undeniable that Kubernetes changed how entire sectors and industries approach software delivery in production environments. As the second largest open source project in the world after Linux, Kubernetes is the primary container orchestration framework for 71% of Fortune 100 companies. On a yearly basis, the Cloud Native Computing Foundation (CNCF) and other organizations conduct several annual surveys and audits to showcase overall user satisfaction, while the myriad of adopter case studies highlight the benefits and numerous solved technical challenges.

Looking back, Kubernetes served as the gravitational point for the cloud native ecosystem. Over the years, there was considerable growth in the number of vendor organizations that developed new tooling or functionalities that would benefit the container-focused infrastructure. At the same time, platform engineering teams around the world would leverage a pluggable system, integrating their desired tooling for networking, storage, service mesh, observability, and many more with minimal technical compromises. The constant adoption rate from the end user's side and the creation of new/enhanced offerings from third-party companies created a closed feedback loop that skyrocketed the growth of the cloud native landscape. Currently, there are more than 175 fully open source projects (as of November 2023) of varying maturity levels, including sandbox, incubation, and graduation.

The main factors in the success of Kubernetes were its open source nature and the community around it. Open governance and transparency are the fundamentals of any open source project, which unlocks unbiased feature development while at the same time welcoming community feedback and new ideas. Through the openness to contributions from multiple individuals and organizations, a defining momentum was built around Kubernetes and the overall cloud native ecosystem, engraving a long-lasting impact in the technology sector.

Considering the ongoing success of Kubernetes, we might ask why, at this very moment, we require an entry-level examination and how this will impact the larger cloud native ecosystem. To answer these questions, we must first explore the challenges we faced with existing resources and certifications. This will help us highlight why setting ambitious goals for our community's growth and focusing on paving the path for emerging talent is imperative.

A Bit of History

Cloud native's mission is to foster a sustainable and resilient ecosystem. Our community should not only focus on solving the problems of the present but also cultivate and adopt a culture of education and upskilling for the future generation of cloud native practitioners.

The Open Source Jobs Report released in 2022 found that knowledge of cloud and container technologies is the most in-demand skill set among hiring managers, but 93% of those hiring managers report difficulty finding sufficient talent with open source skills. Hence, our first challenge as a community is to bring, upskill, and retain new talent in the industry. Kubernetes certifications, like Certified Kubernetes Application Developer (CKAD), Certified Kubernetes Administrator (CKA), and Certified Kubernetes Security Specialist (CKS), are incredibly popular in demonstrating cloud native credentials. With hundreds of thousands of registrations to date, these professional-level certifications require candidates to have hands-on experience on a daily basis to be able to pass the exam. This highlights our next challenge and the need to lower the bar to entry for new members, which also correlates with community feedback and demand for a more beginner-friendly and inclusive evaluation to obtain CNCF credentials.

The aim of solving these challenges led to the creation of the Kubernetes and Cloud Native Associate (KCNA) certification in the summer of 2021 and the official announcement a few months later. I am one of the KCNA co-leads, who steered this effort alongside other community members, including submitting, reviewing, and examining hundreds of questions to ensure a smooth onboarding experience for cloud native novices.

As a community, we need to make our resources and tooling accessible to grow the next generation of practitioners. This is the raison d'être for creating the KCNA exam. Nowadays, there are a plethora of resources to upskill new members; however, it is critical to maintain these so they stay relevant and appropriate—especially with the fast-changing pace of technology and emergence of new tooling in the AI, sustainability, WebAssembly, and security domains, among others.

As a result, other associate-level examinations have similarly been created. If you are looking to grow your skills in a niche area, I would recommend exploring some of the new certifications too, including these:

- Kubernetes and Cloud Security Associate (KCSA)
- Prometheus Certified Associate (PCA)
- Istio Certified Associate (ICA)
- Cilium Certified Associate (CCA)
- Certified Argo Project Associate (CAPA)

KCNA Overview

KCNA is a multiple-choice, entry-level exam and offers everyone from students to those looking to make a career change a way to demonstrate the basic knowledge and skills of cloud native. KCNA will demonstrate a candidate's basic knowledge of Kubernetes and cloud native technologies, including usage of API and `kubectl` commands, the high-level Kubernetes architecture, and understanding the ramifications of the cloud native landscape in domains such as security, storage, observability, GitOps, and service mesh. Overall, the exam is divided into five main parts covering Kubernetes and the wider cloud native landscape.

Kubernetes Fundamentals covers 46% of the questions and tests whether students understand the correlations among resources, such as containers, Pods, ReplicaSets, and Deployments. Additionally, this part of the exam explores how the scheduler works to create new replicas of an application, alongside top-level components of the Kubernetes architecture.

The second exam subdomain, Container Orchestration, checks the candidate's understanding of runtime and how containerized workloads are created on the cluster nodes, alongside the connectivity and reachability to these workloads. This section covers the high-level functionalities of service mesh and storage, including the usage of security features within a cluster, such as network policies, role-based access control (RBAC), authentication, and authorization.

The Cloud Native Architecture part examines the basics of scaling techniques for containerized workloads, such as the Horizontal Pod Autoscaler (HPA), Vertical Pod Autoscaler (VPA), cluster autoscaler, and usage of serverless tooling. Open standards and interoperability are also covered, with an overview of the Container Runtime Interface (CRI), Container Network Interface (CNI), and Container Storage Interface (CSI). Lastly, this section also covers CNCF governance and community guidelines.

Finally, the last two exam parts explore the candidate's knowledge of observability and application delivery tools. These sections highlight the usage of telemetry tooling

such as Prometheus and OpenTelemetry, continuous integration and continuous delivery (CI/CD) fundamentals, GitOps functionalities with tools like ArgoCD, and cost management techniques.

The *Kubernetes and Cloud Native Associate (KCNA) Study Guide* is an excellent resource that will help you study for the exam and set you up for success. While it's essential to understand Kubernetes, its architecture, and its rich feature catalog, it is similarly crucial to lay down the nascent challenges the industry faced that led to the mass adoption of container-centric solutions. As such, this study guide will walk you through the cloud computing concepts that influenced the rise of DevOps practices and the establishment of the cloud native vision and CNCF. You will also learn about the Kubernetes predecessor, Borg, an orchestrator framework that is capable of managing and running hundreds of thousands of jobs on several distributed machines (also known as a *cluster*). Finally, you will learn key Kubernetes technical domains, tooling within the ecosystem, and what to expect during the exam.

KCNA Is Only the First Step!

Broadly successful and widely adopted tools require different perspectives and skill sets that can only be nourished in a welcoming and respectful environment. Openness to implementations, feedback, and new ideas is key to unlocking a high-velocity rate for feature development while lowering the entry bar for new contributors and adopters.

From the start, the Kubernetes community has operated on the policy that "Inclusive is better than exclusive." This is one of CNCF's core values. Our goal is to establish strategies to build, scale, and retain contributor communities, through open governance, transparency, and a diligent code of conduct. This is what pushes innovation forward and invites everyone to contribute!

As such, I would like to encourage everyone who studies for their KCNA exam to take the next step and become an active member of the community, as an adopter, contributor, maintainer, or any other role that suits your skill palette. We need to create a welcoming and respectful environment where feedback and new ideas are valued, and also drive diversity of the executive leadership, governing and advisory boards, and technical committees. We should continually strive toward creating a nurturing space for the next generation of cloud native practitioners to operate in.

Good luck studying and taking the KCNA exam, and I am looking forward to seeing how you can shape the cloud native ecosystem!

— Katie Gamanji
London, December 2023

Preface

KCNA. It's an interesting choice for a new technology certification acronym. When you search for this term online, many of the results refer to a national news agency in Asia. But we won't be talking about that here. The Linux Foundation released the Kubernetes and Cloud Native Associate exam in 2021, and even if the acronym can initially be a bit confusing, the name is not at all random. The KCNA exam and certification name is a direct reference to Kubernetes technology and its role as a key cloud native ecosystem enabler.

It makes sense. Containerization is quickly becoming the new normal for big and small technology companies and their developers around the world, and technologies such as Docker and Kubernetes are "nonofficial" standards for most of the adopters out there. This powerful movement has been brewing since the Cloud Native Computing Foundation (CNCF) was created in 2015, and companies like Google have leveraged cloud native technologies since the early 2000s. Given that this is a complex area of highly technical knowledge requiring a lot of practice, effort, and upskilling, the community needs new learning and training resources, documentation, and simple ways for professionals to learn and adopt cloud native tools. This exam is one of the avenues to make this happen.

But this is not really a hurdle. I (Adrián) still remember the first time I heard about cloud native. It was during a job interview, and I naively thought my interviewer meant something like general cloud computing. We will discuss the difference (and relationship) between these two terms later, but it is clear that I was missing something big. At the time, I was well versed in 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 my main reference for virtualization techniques were the traditional VMWare-ish virtual machines. New communities such as CNCF 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. 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 a discussion with a friend in the video-game 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 no professor explained this to us because it was just too early. Also, I had lost some of my command-line ability and abandoned my arduous exploration of Linux-related topics. In years past, I stayed awake at night trying to find a way to replicate some cool functionalities from Windows in Linux Ubuntu, participate in forums, try new open source software, debug new tools…but most of these skills 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 had been "devouring"), 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 collaborators, the open source governance model, cool projects beyond the omnipresent Kubernetes, and the levels of maturity and incubation. What a discovery! A community full of talent, willingness to teach and collaborate, existing resources, and more.

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? (The official Kubernetes documentary, including Part 2, is a must, and a great option to get some initial answers.) Then, I organically started to explore the technical topics and connect more dots. It was challenging but feasible to learn with a good study plan and learning material. This was the origin of the content 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 way 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. This was a fundamentals-based, 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—namely, the Certified Kubernetes Administrator (CKA), Certified Kubernetes Security (CKS), Certified Kubernetes Application Development (CKAD) paths. This is a bit too long a walk for people who are still trying to crawl. But the CNCF and Linux Foundation Certification teams nailed it; they released a great exam that, even if it is not the easiest one, helps to bring in 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. I even prepared some 101-level KCNA exam prep sessions with O'Reilly, and the Open Source Summit Latin America. I consider myself a pretty good lecturer, but this 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. This team had managed previous book publications for other Kubernetes certification topics, including the wonderful study guides that so many professionals had used before to prepare for their CKA, CKAD, and CKS certifications. And I was really willing to put some thought and time into designing something that would help KCNA candidates to (1) learn about Kubernetes and the cloud native ecosystem and landscape and (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 with up- and reskilling.

The rest is history. The wonderful O'Reilly team gave me the opportunity to sit and write this book, a relatively light guide for candidates to understand what to learn, in a pragmatic and simple way, that also serves as a starting point for their new cloud native journey. This was a pretty difficult task to manage in addition to my professional life (i.e., working at Microsoft, teaching, international speaking engagements, etc.). But it 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. If this book helps just a few of them, I will feel very satisfied, the same

as when I teach a small group of students. Contributing to create a new generation of cloud-enabled professionals is very cool. Please enjoy this study guide.

— Adrián González Sánchez

Technology evolves fast and most of the time even faster than the market can figure out how to take advantage of it. One of the biggest catalysts setting the pace of innovation at all levels is the "Open" movement. At its early stages, its roots were set under the software dogma that for the good of the world, all code should be open.

Nowadays, the "Open" philosophy is not only related to software but plenty of other computer science domains (for example, data and hardware, where there are projects and communities working hard to establish the same principles and lessons learned from the software side). For someone like me (Jorge) whose career started in the dot-com era, it was unthinkable to anticipate the wave of innovation being forged under the power of the open source communities.

When I worked for Sun Microsystems in 2006, I started to understand the real impact that people with a sense of changing the world for good can make in society. I had the opportunity to meet people whose professional lives were 100% dedicated to writing code for a bigger purpose than I could imagine at that time. It was a turning point in my professional maturity and changed the path of my career completely. I understood that a software project where people with different interests, views, and skills have the freedom to create, propose, and evolve it just to contribute to solving a problem or improve methods in specific areas of the industry creates a "sense of belonging" that cannot be seen anywhere else.

If you have this book (or ebook) in your hands, it's because one way or another something has awakened your interest in the cloud native space. This area is without a doubt one of the engines where the open source community works hard evolving how we develop applications and how we run them in a time where end users' expectations are higher than ever, artificial intelligence is expected to be the revolution of our time, and software has to be deployed everywhere (from very small devices like bulbs, thermostats, or even refrigerators to critical devices such as vehicles, planes, trains, and spaceships), often multiple times per week (even per day). Gone are the days of anxiously waiting for the new release of an operating system or an application whose performance created a bad user experience.

If the cloud native space looks to you like an area where innovation is happening, let me tell you something: it is! And whether getting the KCNA certification is your first dive into the cloud native ecosystem or another step in your career, it will be worth it. This is my first experience writing a book, and as an open source software "believer" and spending my whole professional life somehow linked to it, I feel a

great responsibility at the opportunity to spread the passion about cloud native and, specifically, the Kubernetes project through this book.

If you're reading this book, I have one piece of advice: learn something new every day. Connect with the community (you will soon see how rewarding it is) and don't be shy to share your passion for projects with which you have a personal connection. And to the amazing O'Reilly team and Adrián (my partner in this adventure): thank you! I couldn't imagine a better team to turn a dream into a reality.

— Jorge Valenzuela Jiménez

How This Book Is Organized

This book has seven chapters that contain the end-to-end knowledge you are expected to have when you decide to take your KCNA exam. They are applied and descriptive chapters that cover relevant exam topics, in addition to contextual information that will help you understand the full picture of the cloud native and Kubernetes space. Let's review each of them:

Chapter 1, "Introduction to the KCNA Certification and Study Guide"
: The first chapter is the entry point 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 weight of each domain, the types of questions to expect, and the significance of this certification in your 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), including 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"
: The 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, and declarative APIs. The idea is to equip you with additional information for your KCNA exam, but also to help 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. This 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 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 equips you with a final set of tools and some final pieces of advice. Benefit from a curated list of resources and real-life tips from professionals who have successfully trod this path.

Appendix A, "End-of-Book Knowledge Check"
: Appendix A contains a quick knowledge check for you to confirm your understanding of some basic topics related to the KCNA exam.

Appendix B, "Solutions to Self-Questionnaire and Knowledge Checks"
: Appendix B contains solutions to the self-assessment presented at the end of Chapter 1, as well as the knowledge checks in Chapters 2 and 3 and Appendix A.

This book covers the step-by-step learning blocks of your KCNA journey. Let's now talk about who we wrote this book for.

Who This Book Is For

If you are reading this book, you likely have some preliminary knowledge or interest in cloud computing and cloud native topics. For example, before starting preparation for the KCNA exam, you may have benefitted from cloud computing offerings on

topics like hyperscalers, Linux fundamentals, command-line experience, and telematics and networking. The perfect scenario would even include 6 to 12 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 may be missing some or all these knowledge blocks. Regardless of your profile, here are some recommendations 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 public and private cloud fundamentals. For that, you can rely on existing O'Reilly resources such as books and video lessons. You could even explore some cloud entry-level certifications from the main hyperscalers such as AWS Certified Cloud Practitioner, Microsoft Certified: Azure Fundamentals, or Google Cloud Digital Leader. All of them include terms that will help you understand cloud computing resources and capabilities, which you can easily apply by leveraging the amazing O'Reilly cloud lab 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 in the following chapters), and some basic software lifecycle terms. As before, you can definitively leverage other O'Reilly learning resources to explore the fundamentals of software development.

Industry knowledge
: We cover industry knowledge in this book, but it will be good for you to have a clear overview of the technology companies related to the cloud computing and cloud native industry. Names such as Google, Red Hat, Microsoft, and AWS should already sound familiar to you, but if you know their solution offering, this process will be a bit easier for you. You will also learn about managed Kubernetes services from leading companies in Chapter 6.

Containers and Kubernetes
: In the best-case scenario, you have already used Kubernetes, Docker, and other related technologies before taking the exam. That said, this book covers the fundamentals required to pass the exam, additional resources for you to continue learning, and technical aspects such as commands and architecture elements that may help you with 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 you will explore in Chapter 6.

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

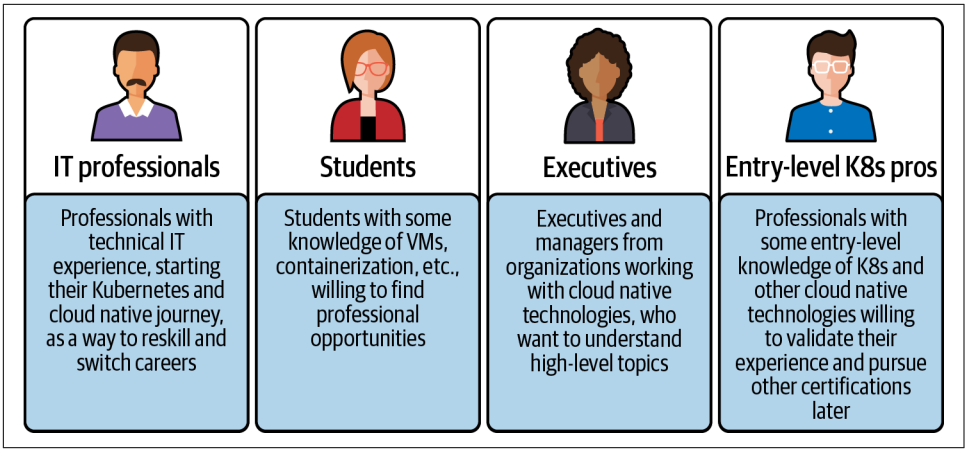


Figure P-1. Typical roles for KCNA exam candidates

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 looking to join the cloud native industry, and a powerful portfolio asset for new career opportunities. In Chapter 7, we 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
: Software developers are responsible for designing, coding, and testing the microservices that make up 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 cloud native applications on the chosen platform. They use container technologies such as Docker or Kubernetes to orchestrate microservices and ensure their availability and performance.

Solutions architects
: 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
: The 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 cloud native applications.

Project managers
: Plan, coordinate, and oversee the entire lifecycle of developing and implementing 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 engineers
: Responsible for designing, developing, and deploying applications based on microservices, containers, and cloud platforms.

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

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

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

Business and sales professionals
: Nontechnical folks working for cloud-enabled companies and/or selling software-as-a-service (SaaS) solutions.

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 do you need to make it happen, and what percentage of those are already covered in the KCNA 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 all 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 a living support for your cloud native upskilling journey, on top of helping you prepare for 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 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 with some of the key Kubernetes and KCNA actors, links, and references to rich and highly evolving relevant community repositories, and of course, a lot of applied knowledge. For this last point, we will be sharing examples of how companies are adopting cloud native technologies and generating value for their products and clients.

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic
: Indicates new terms, URLs, email addresses, filenames, and file extensions.

`Constant width`
: Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, databases, data types, environment variables, statements, and keywords.

`Constant width italic`
: Shows text that should be replaced with user-supplied values or by values determined by context.



This element signifies a general note.

O'Reilly Online Learning



For more than 40 years, *O'Reilly Media* has provided technology and business training, knowledge, and insight to help companies succeed.

Our unique network of experts and innovators share their knowledge and expertise through books, articles, and our online learning platform. O'Reilly's online learning platform gives you on-demand access to live training courses, in-depth learning paths, interactive coding environments, and a vast collection of text and video from O'Reilly and 200+ other publishers. For more information, visit *https://oreilly.com*.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-889-8969 (in the United States or Canada)
707-827-7019 (international or local)
707-829-0104 (fax)
support@oreilly.com
https://www.oreilly.com/about/contact.html

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at *https://oreil.ly/kcna-study-guide*.

For news and information about our books and courses, visit *https://oreilly.com*.

Find us on LinkedIn: *https://linkedin.com/company/oreilly-media*

Watch us on YouTube: *https://youtube.com/oreillymedia*

CHAPTER 1

Introduction to the KCNA Certification and Study 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 and contributions of the cloud native community in recent 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 look at the CNCF Landscape, you'll see more than 1,200 tiles that represent the large and small companies involved in cloud native activities, about 820 company members, and a rich list of projects with an amazing level of community engagement.

Defining Cloud Native

But before we dive in, what exactly is "cloud native," and what does it mean for an adopter company? We will discuss the difference between general cloud computing and cloud native later, but even if "cloud native" has no single definition, let's take a look at CNCF's definition as a guiding force:

> 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.

In short, this means having access to technical benefits typically related to cloud-enabled systems, regardless of the work environment. Let's look at other definitions

from big technology companies, which are obviously related but bring different perspectives, depending on the organization and its experience:

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

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.

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.

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

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.

Amazon Web Services (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.

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. Over the last two decades, we have seen the beginning of what we define as the new era of cloud native. It is a new era because of the appearance of CNCF, which has been growing since its founding in 2015, at both the community and technology level (we will explore this further in Chapter 3).

This generated not only a wide level of 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 has provided a multitude of events, training options, resources, and new projects in recent 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 searching for ways 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. All this makes for the perfect combination and timing for cloud native technologies.

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 over the last two decades. Initially, these powerful technologies were available to only a small group of technology-enabled companies including Google and AWS, but other organizations have leveraged them and showcased how to adopt them and transform their businesses. Here are a few examples from the public CNCF and Kubernetes.io case studies:

Boxed (acquired by MSG Distributors in 2023)
: This online wholesale retailer offered direct delivery of bulk-sized packages. As its customer base grew, it began seeking more agile and scalable infrastructure solutions to handle its ecommerce operations. As the company began to experience increased traffic, it was clear that the 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 its team, managing stateful services, and monitoring in a distributed environment. Boxed's journey with Kubernetes and cloud native emphasizes that even if a company is not a global giant, the principles of scalability, resilience, and agility provided by Kubernetes are great proof of the democratizing power of open source technologies, allowing companies of all sizes to harness top-tier tech solutions.

Airbnb

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

As traffic increased, deployment with its existing monolithic architecture became riskier, slower, and more error prone. To break away from the limitations of a monolithic structure, Airbnb looked toward a microservices architecture, where each function of its platform could operate as a distinct service.

Airbnb began by containerizing individual services, progressively transitioning them onto Kubernetes. By starting with less critical services, it could ensure that any potential issues didn't critically impact its platform, allowing it to learn and adjust 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. Its journey serves as an inspiration to companies worldwide, emphasizing the importance of adaptable infrastructure in today's digital age.

Zalando

Zalando is one of Europe's leading online fashion platforms. With millions of customers, a vast inventory, and numerous daily transactions, it needed a robust and scalable technical infrastructure to support its 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 its containerized applications but 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 its Kubernetes environment. Zalando could maintain and manage Kubernetes resources using declarative configurations, streamlining its deployment process. It also used Prometheus to monitor its services, set up alerts, and gain insights into its 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

The New York Times, one of the world's most renowned 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 that its infrastructure could support the demand. The company 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, the Times wanted to break its monolithic application into microservices. Kubernetes was chosen as the orchestration platform because of its technical features, community support, and 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 noncritical applications to understand Kubernetes better and to mitigate potential risks. Over time, as it gained confidence and expertise, more critical parts of its infrastructure were migrated. Thanks to this, the Times handled reader traffic spikes more efficiently and started to run its infrastructure more cost-effectively.

A.P. Moller - Maersk

Maersk is one of the world's largest shipping companies. As 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. Its legacy systems were disparate, slow, and often siloed, and the company wanted to consolidate its IT operations and introduce more agility and scalability into its 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 its Kubernetes-deployed applications, Maersk could derive actionable insights from its 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 are just a few examples 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 a positive consequence: huge talent demand, which means a lot of professional opportunities for learners like you.

The next section includes a self-assessment that will help you honestly analyze 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 a lot 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.

There are *45 multiple-choice questions* for the following topics:

- Seven questions related to the *CNCF ecosystem*, mostly focused on high-level details of the Foundation and its mission and mechanisms.
- Eight questions on *cloud native concepts*, as an initial test of your understanding of cloud computing and cloud native.
- Eight questions on *Kubernetes and orchestration topics*, to assess the knowledge required for the KCNA certification.
- Five questions for *Kubernetes commands*, which are the `kubectl` instructions required to manage technical aspects of Kubernetes. If you have never worked with Kubernetes on a technical level, don't worry; we will explore these commands in Chapter 6.
- Eight questions related to *Linux fundamentals*, which are technically *not* part of the official exam curriculum as we explain in Chapter 2, but will help you not only understand some Kubernetes-related areas of knowledge but also perform better at work.
- Last but not least, nine questions related to *other cloud native projects*, to explore the ecosystem of tools and projects (besides Kubernetes). We describe them in Chapter 3, but it is good for you to know whether you have some initial or high-level knowledge of those tools.

Again, don't try to guess or find the right answers just to "pass" this test. Focus on choosing an answer based on your current level of knowledge, and take notes while you review the correct answers, especially for those areas you know less about. Solutions can be found in Appendix B.

Part 1: The CNCF Ecosystem

1. What's the primary role of 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 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

You may have missed some questions, but that's easy to remedy as you will soon know the role and details of CNCF. Explore Chapter 3 to learn more about this and other related topics.

Part 2: General Cloud Native Concepts

1. What is the difference between monolithic and microservices architectures?
   a. I am not familiar with these terms.
   b. They are both different software design and architecture approaches, based on one versus several development "blocks."
   c. The connection between data services from different pieces of the software solution.
   d. They are similar, only frontend/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. Backend as a service (BaaS)
   c. Deployment as a service (DaaS)
   d. Frontend 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 offer only 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. If you struggled with these questions, pay particular attention to Chapter 4, which discusses these and other cloud computing and cloud native terms.

Part 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 nonexistent 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 the core 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.

Part 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 -c CONTAINER NAME -- /bin/bash`
   c. `kubectl shell POD_NAME -c CONTAINER_NAME`
   d. `kubectl run POD_NAME -c CONTAINER_NAME`

Kubernetes `kubectl` commands are a core area of knowledge, not only for the exam but for your professional activities. We include relevant information and additional resources for you to make sense of these technical commands in Chapter 6.

Part 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`

Your background and experience will determine how you performed on these questions. In Chapter 3, we will expand on a variety of Linux-related topics, as well as some existing O'Reilly and external resources that will help you catch up and gain some base knowledge before taking the KCNA exam.

Part 6: Other Related Projects

1. Identify which of the following is *not* a CNCF project:
   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 among 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 plug-ins 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 exam certainly includes questions related to them, and this study guide will help you develop this knowledge. As illustrated in Figure 1-1, you will need to pay special attention to some parts of this book and other resources we mention within the chapters, depending on your weakest areas of knowledge from the self-questionnaire as well as your preliminary level of experience and knowledge.

To summarize, these are relatively basic questions that do not fully cover the scope of the KCNA exam. However, if you found them difficult and hesitated while answering, 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 "ABCs" of it. The key is to gain, at the very least, basic knowledge of all these relevant areas. Figure 1-1 maps out the topography of exam content and the chapters in this book. We recommend that you use it as a guide to help focus the preparation for the exam.

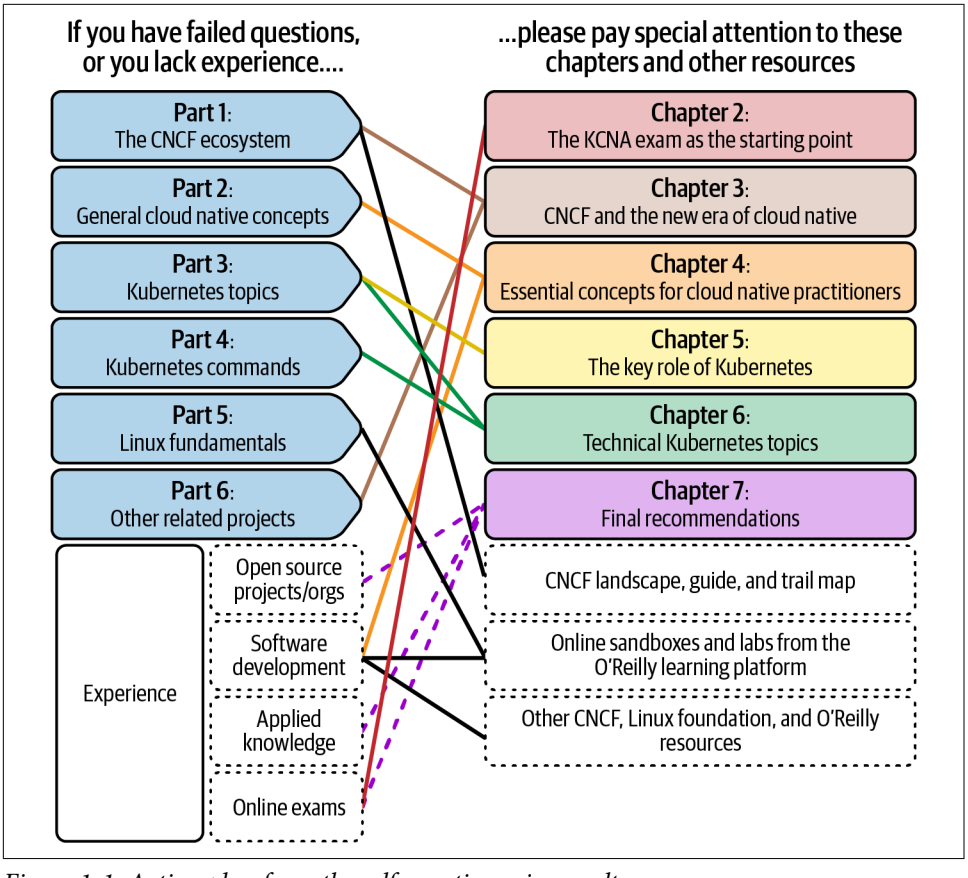


Figure 1-1. Action plan from the self-questionnaire results

The next section outlines a transcript of an interview that we did with an industry expert who has intimate knowledge of the KCNA exam and its ecosystem. The goal of this section is to provide context and relevant information that will not only help guide your path toward obtaining the certification but also lead to success in your career.

For this first chapter, our expert insider is Walid Shaari. Walid is an open source aficionado and one of the earliest participants in the creation of the KCNA. His hands-on expertise extends beyond his work at Amazon Web Services, where he is a container ambassador. He was a beta tester of the KCNA during its inception and is certainly one of the top KCNA experts out there. Walid was part of the original team who created the exam questions and is a major proponent of the open source culture and the CNCF community. Feel free to explore his valuable KCNA resources, such as his KCNA repository and webinar with Brad McCoy on preparing for the KCNA exam.

Expert Insights: Walid Shaari

Adrián: Hi. Welcome to this series of expert insights for the KCNA exam. Today we have the pleasure to welcome Walid Shari, who is an expert in the cloud native community.

Walid: Thank you for inviting me to participate.

Adrián: 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 the cloud native community, it's the best community in terms of inclusiveness and knowledge sharing. This sort of culture is exhibited during KubeCon sessions and other events.

Adrián: 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. The CNCF, the Cloud Native Computing Foundation, was established with the first project, Kubernetes, donated by Google. The CNCF saw the gap between the enterprises, the organizations, and the people, one of them being the skill gaps, which it solved by providing curriculum, training, 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 measure 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.

Adrián: You have been leading various activities of CNCF, 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 didn't have any content that you could rely on except whatever is publicly available or on Slack channels. I was also a beta tester for the Prometheus observability exam. It's good to do the beta testing because 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.

Adrián: Right now, there is a lot of content out there. We are not lacking in content and documentation. In the book, we mention 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. 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 is a good resource for many certifications in 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 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 supposed to be the 10,000-foot level, but it's not really, because it was written by engineers. So you'll still find command lines; you'll still find technical details. And these are the questions that might confuse you.

When I took 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 it is multiple-choice, it's not hands-on. If it was hands-on and you 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 saw a question that was a paragraph or a couple of lines long at that time.

Adrián: 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 provided 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 testers need to go deeper in their preparation, even if it's an intro-level certification. Because that's the kind of question they 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 the Linux command line, you usually use `ps` (list processes), `aux`, etc. 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 ps` in Kubernetes. 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.

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

Walid: Yes. The other confusing thing is when you get into details. 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 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 14-hour course, with some hands-on follow-up. Basically, he walks you through some exercises. Cloud Guru also has 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 101 content. For example, what is a 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 lifecycle of a project in CNCF.

Adrián: You mentioned the ecosystem including activities and projects. We know that besides Kubernetes, other projects are relevant for monitoring or for application delivery, etc. What are the projects that a new learner should be aware of?

Walid: The objective of the KCNA is to target newcomers and nontech people who are in the Kubernetes world. If 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 they 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 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 who 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 are my bottlenecks? How's the latency? How's the performance? How can I troubleshoot in 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 on problem-solving. So why did Kubernetes come into place? Why did containers take the technology world and the business world by storm? The portability, the resiliency, the business value. FinOps is part of the exam, but it's not very obvious. The total cost of ownership, and the cloud adoption value. 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. It's a win-win situation. 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.

Adrián: This is a good example, given that this topic related to FinOps is a piece of knowledge that is 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 their own laptop and learn about Kubernetes, there is Kind, 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 videos. There are books, this book, for example. There are meetups. The meetups are good because they are interactive. And there are conferences like KubeCon and Kubernetes Community Days.

CNCF saw that it cannot scale when it came to conferences and that these conferences are getting really huge and starting to be expensive. So one way to scale them up is to do 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. The operator in Kubernetes is how to combine human knowledge, operator knowledge, and technology, especially for stateful applications, into a packaging. And with this packaging, it becomes easy to install a software stack like a Postgres cluster and see 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 they have 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.

Adrián: 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 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. In a CNCF-related project, there are tags for 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 X (formerly Twitter) or Slack. With 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 haven't seen this in other projects.

Adrián: That's gold, because it offers beginners like many of our exam takers here 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: Giving feedback and helping with documentation, helping with localization, these are the easiest ways in.

Adrián: Perfect. I think we have covered more or less all the important topics for 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 for their associate-level exam?

Walid: For the associate-level exam, I would focus on the KubeCon business tracks. There haven't been that many, but they have been recent. You can learn about the ecosystem 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, X (formerly Twitter) is good. Follow James Barron, Rawkode, and David Flanagan. He has Rawkode Academy, and every couple of weeks he has a new topic. He's discovering a new technology or a new software stack or playing with it. There is also Thank God It's Kubernetes (TGIK) run by Heptio earlier and then later VMware.

You can also follow people in the community through social networks and LinkedIn. For me, I get my feeds from X (formerly Twitter) and LinkedIn. I follow specific people on LinkedIn and I get the updates and news from them of what the new projects are. For me, 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 has short videos. VMware has the same short videos. I like these small nuggets, actually. Maximum 15 minutes.

Adrián: Yes, like a mini-Netflix show. You mentioned Brendan Burns from Microsoft, Joe Beda's videos from VMware. There is lots of material. We recommend the videos as well. You mentioned it once when we were reviewing the book; it's very illustrative.

This is wonderful. If we had to use one word to prepare people for their journey, it would be awareness. Be 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. And as 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 has a GitHub repo for its exam curriculum. They keep updating it. They actually mention some resources there.

Adrián: Yes. Very good selection. This book also includes champions in the community, including yourself, your personal repositories, which you are recommending already, as 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.

Adrián: Well, I'm very happy we got some time to discuss these topics. I cannot wait to share everything, the material, and all the resources you have mentioned to help people pass the exam. 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.

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

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

Summary

We have now concluded the first of 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 gained insight on the KCNA exam creation process and gotten some recommendations on how to start studying for it. If you feel like you are still missing some details, at both the logistics and knowledge levels, don't worry. We will continue to walk you through everything you need.

CHAPTER 2

The KCNA Exam as the Starting Point

Since you are reading this book, you likely want to get the KCNA certification from the Linux Foundation, but you may know less about the Foundation itself (don't worry, we will cover it in this chapter). You should know that passing this exam demonstrates a strong understanding of Kubernetes and cloud native fundamentals, but it will also show that you have a grasp of the principles of cloud native development and security. Thus, having this certification under your belt prepares you for further advanced and more specialized credentials such as Certified Kubernetes Application Developer (CKAD) or Certified Kubernetes Security Specialist (CKS) 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're taking it, is not easy. In this chapter, we will explain exam details and logistics in order to set you 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 origins of the exam.

The KCNA Certification by the Linux Foundation and CNCF

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: first, you will want to leverage the Linux Foundation's online exam platform. Second, the content of the exam and other resources relies heavily on specific knowledge from experts who often act as CNCF ambassadors or contributors, as mentioned earlier.

As you begin learning about cloud native technologies, you will notice that both the Linux Foundation and CNCF names come up frequently. See Figure 2-1 for their foundation logos.



Figure 2-1. The Linux Foundation and CNCF logos

The Linux Foundation was created in 2007 from the merger of two existing organizations, the Free Standards Group (FSG) and Open Source Development Labs (OSDL). It is a nonprofit 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 is the umbrella organization for other projects and subcommunities, one of which is the Cloud Native Computing Foundation. Actually, CNCF is one of the Linux Foundation's largest subfoundations, 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. Figure 2-2 presents a sample of the list of Linux Foundation projects. Feel free to bookmark the preceding link so you can explore it to find other topics (e.g., databases, development, financial operations for the cloud) that will complement your cloud native upskilling journey.

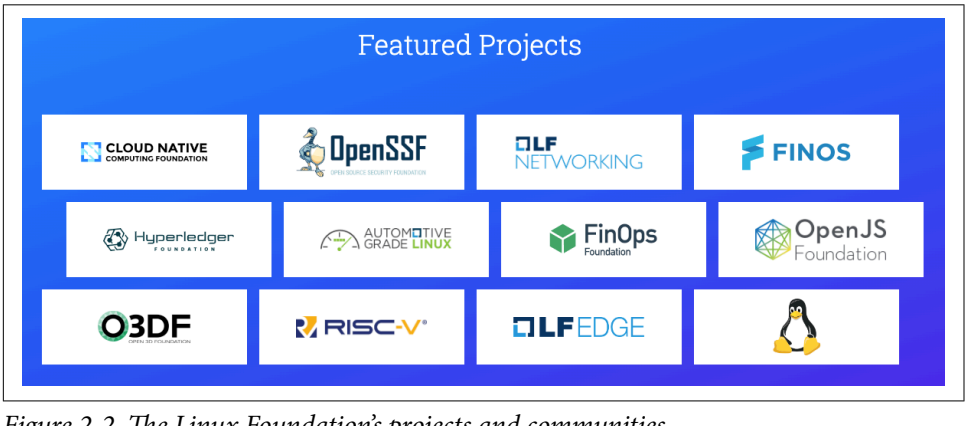


Figure 2-2. The Linux Foundation's projects and communities

Seeing that CNCF is just one of the many organizations under the Linux Foundation's 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 many open source communities 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 by 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 a 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 areas of improvement you detected (i.e., those where you feel you need to improve a bit more). For example, if you answered most of the Linux-related questions incorrectly, 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 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 Introduction to Kubernetes free courses, as they can bring a complementary perspective to the existing catalog of Kubernetes resources on the O'Reilly learning platform. Once again, depending on your level of experience and professional background, a successful formula will combine this KCNA Study Guide as your main reference to prepare for the exam along with other sources of knowledge.

Apart from the aforementioned role as industry knowledge hub, the Linux Foundation's international events (such as the Open Source Summits in North America, Europe, Latin America, and China) conduct open source research and provide practical guidelines and support for open source collaboration. We recommend these, 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 return to the KCNA exam and review 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 You Should 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 you 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. Some of the report insights that are relevant include the following:

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 work 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 cybersecurity 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 hiring priorities for most companies, along with AI and cybersecurity.

In parallel, CNCF's 2022 Annual Report indicated that the KCNA exam had 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 are the main roadblock 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 industry trends and the figures, 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 no official statistics exist on hireability of candidates passing the KCNA, the benefit of this certification really lies in preparing you to get some initial experience in the job market, as well as creating a path to complete further advanced certifications. Let's now take a deep dive into the details of the KCNA exam and its official curriculum.

Exam Description and Curriculum

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. Let's start by reviewing key information about the exam:

Target audience
: The KCNA serves as an introductory certification tailored to 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 KCNA examination 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's 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
: As a beginner-level certification, the exam focuses on testing conceptual as opposed to technical knowledge. This is meant to guide 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.

Scope of cloud native ecosystem
: How big is the *entire* cloud native ecosystem? The answer is large and growing. However, CNCF outlines the exact scope covered in this certification in its curriculum. We'll talk more about this in this chapter and Chapter 3.

About the Format

Even if the general description of the exam presents KCNA as an entry-level certification for practitioners who are starting out, the reality is that it can still be 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 studying, very targeted topics. In general, you will find multiple-choice questions 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 its 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 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 upcoming chapters. Meanwhile, here is an overview of the official KCNA curriculum, regularly updated and available via CNCF's GitHub, with a high-level introduction of each exam topic. Let's dive into the five official parts of the exam and additional areas of knowledge.



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 *primary documentation references*:

- Kubernetes.io documentation
- CNCF Glossary

You can consider these two URLs the primary sources of terms and descriptions, which is very useful for your cloud native upskilling, including KCNA exam preparation (some questions focus on core terms that are described within these two links).

Part 1: Kubernetes Fundamentals

Part 1 counts for 46% of the total exam content, and we refer to this block of the content as a Kubernetes 101 section. However, 201 may be more appropriate at times, because some concepts such as technical commands or architectural elements are not trivial. Here's what to know:

Kubernetes resources
: One of the fundamental aspects of Kubernetes is its variety of K8s objects. Here are some relevant terms and resources: Pods, Deployments, 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. You will need to know how each component interacts with the others to carry out the functions 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 create settings and configurations.

Containers
: This portion deals directly with the notion of containerization, and its role in microservices implementation. The concept of containers is at the heart of Kubernetes. The container orchestration part of the exam (discussed next) 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

Part 2 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 the following:

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
: Topics 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

Part 3 counts 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 CNCF and some industry standards:

Cloud native architecture fundamentals
: This content dives into the general cloud native architecture and relevant concepts beyond Kubernetes. It touches upon but is not limited to microservices, DevOps, and continuous integration/continuous deployment (CI/CD) pipelines. It also includes topics related to general cloud native knowledge areas and autoscaling and serverless concepts.

Community, governance, and personas
: These are specific topics related to CNCF and its setup. Given that K8s is the first project under its umbrella, reinforcing open source culture and governance is an important factor for the exam.

Open standards
: CNCF and the Linux Foundation support implementation and adoption of international open source standards for cloud native, containerization, and orchestration technology development.

Part 4: Cloud Native Observability

Parts 4 and 5 count for 8% each of the total exam content and grade. Part 4 contains administration, maintenance, and support building blocks that will help you manage your cloud native systems:

Telemetry and observability
: These topics (which 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 the cloud native level. It is a widely adopted tool for monitoring and alerts.

Cost management
: This content aims at having practitioners demonstrate their understanding of the cost implications 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 is oriented to application development topics. 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 the 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 Argo CD and Flux.

CI/CD
: CI/CD 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 here will be covered in this study guide. See Figure 2-3 for a high-level mapping of the study guide chapters with the KCNA curriculum.

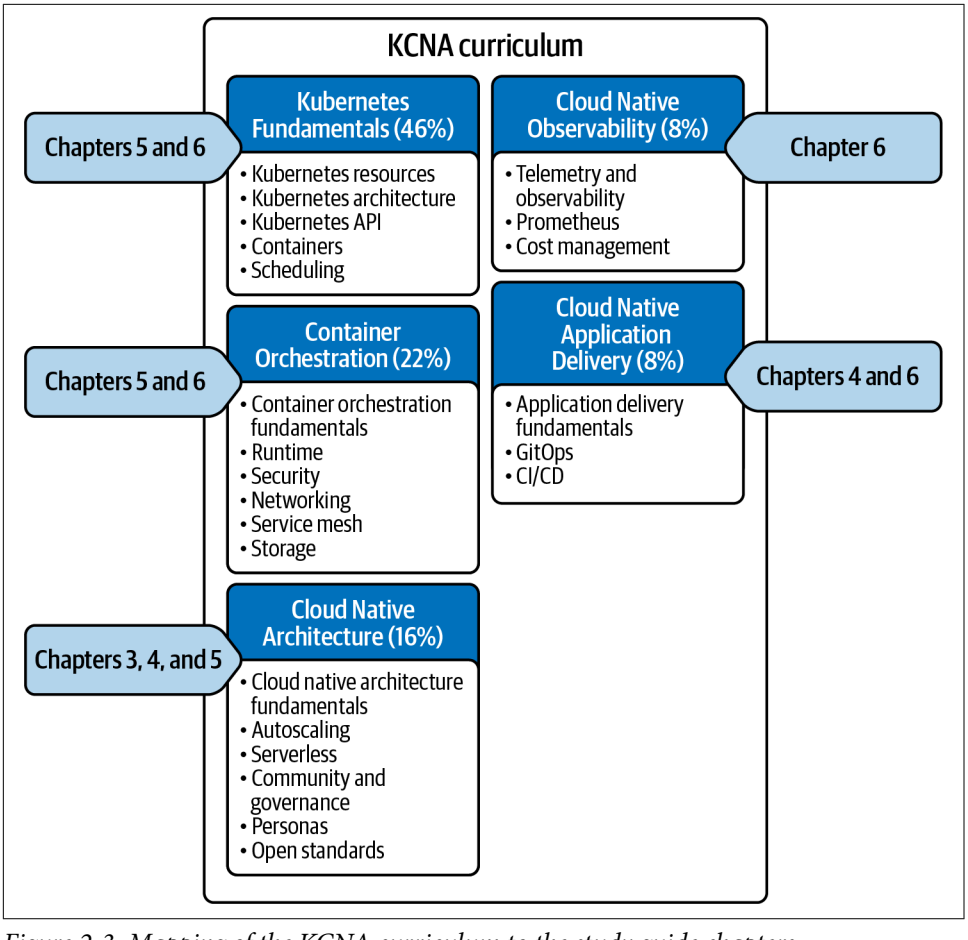


Figure 2-3. Mapping of the KCNA curriculum to the study guide chapters

Now, even if the official KCNA curriculum is the main point of 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 isn'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 have added 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

Once again, this section includes nonofficial topics, which means that they are not directly evaluated in the KCNA exam. However, we believe this content will help you identify additional knowledge gaps, depending on your background, and make passing this exam a bit easier for you.

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 Kubernetes 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 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. Based on the amazing Katacoda platform, they offer a variety of step-by-step scenarios (e.g., defining and deploying resources, launching nodes). Many of them were developed by one of our industry experts, Benjamin Muschko, who will join us in Chapter 4.

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 `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 from renowned industry organizations.
- You can leverage the O'Reilly sandbox platform. Concretely, you have 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.

API-Enabled Development

As a cloud native and Kubernetes professional, you will leverage application programming interfaces (APIs) as a way to connect to remote services, which 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 that you spend some time getting familiar with API-enabled environments by doing the following:

- 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) that have some 101 intro videos and documentation.

Telematics

Telematics is the area of study 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. However, if you work with cloud native tools (including Kubernetes), the notion of networking is essential. We will cover this topic in Chapter 6, but you can also review the official K8s networking documentation.

Web Development

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 models 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.

Applied Usage

Last but not least, one of the best ways to integrate the information you've learned from the official and nonofficial 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 that you can continue exploring in parallel with your regular exam study and preparation process:

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

Website creation
: Imagine an ecommerce website with microservices like product catalog, user authentication, and payment gateway, in which you containerize each microservice and deploy them in a Kubernetes cluster, so each service can work and scale independently, adapting to peak demand moments.

Cloud computing
: Imagine a company working with two or three public clouds such as AWS, Azure, or Google Cloud Platform (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
: You can, of course, containerize your developed applications and leverage tools you use as part of the development lifecycle. For example, if you want to implement a CI/CD pipeline, tools like Jenkins or GitLab CI will help you get the pushed code from the repository, then build, test, and deploy a new container.

Edge computing
: Imagine that you want to process data close to an Internet of Things (IoT) device (i.e., sensors) across different geographies. With 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

Now that you have an idea of what the exam will cover and the weight for each section, how do you go about tackling it? We've included a recommended study plan and checklist here.

We recommend this plan 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 as a starting point to help you customize a learning plan that works best for you.

Step 1: Determine your Current Level of Knowledge

Step 1 is all about being aware of your 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 your level of knowledge, we recommend you take a diagnostic to determine your strengths and areas of focus. The self-diagnostic questionnaire in 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 the 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 go back and take it, then return here with your results.

Step 2: Prepare and Adopt a Study Schedule

Now that you know what 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 one to three 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. 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 the topics that are new to you in this book, 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 toward 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 and 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 Linux Foundation's online examination platform and rules. Especially for first-time test takers, you will need to pay attention to these steps and get ready for 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 review the main steps.

Step 1: Schedule Your Exam

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

1. Go to the Training and Certification platform.
2. Select My Training Portal: this is the key to entering your own personal space (see Figure 2-4).

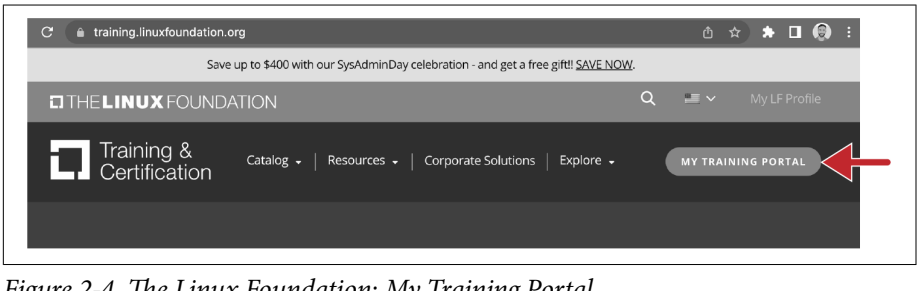


Figure 2-4. The Linux Foundation: My Training Portal

3. On the authentication page, shown 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, or LinkedIn account) if you already have one. If you're creating an account for the first time, a request to verify your email will be sent to the email address provided.

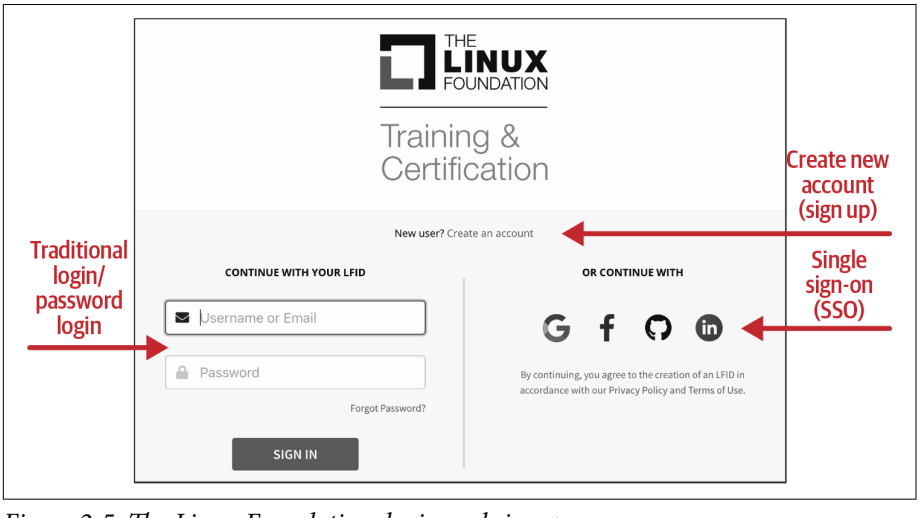


Figure 2-5. The Linux Foundation: login and signup

4. From your new learning dashboard, scroll down and find the browse section. Once there, click "search for content," and type **KCNA** to find the KCNA exam from the list of courses and exams available. As seen 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.

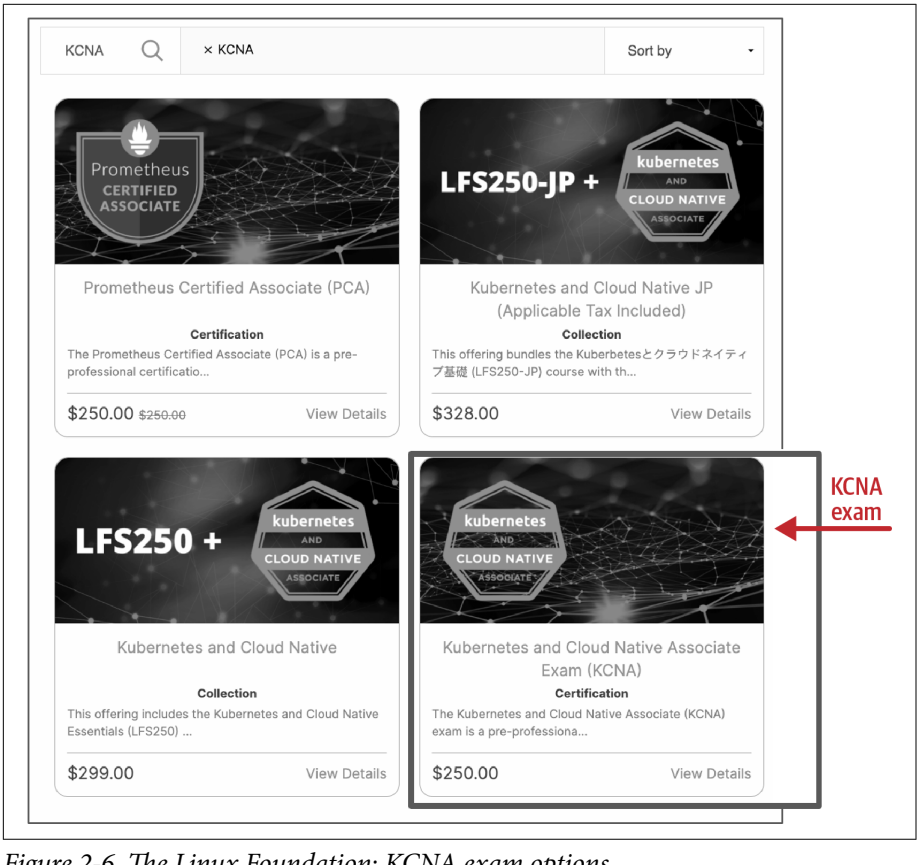


Figure 2-6. The Linux Foundation: KCNA exam options

5. After selecting the KCNA exam, click Enroll Now and complete the checkout and payment process (see Figure 2-7). 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 (which may be available for CNCF and the Linux Foundation members).

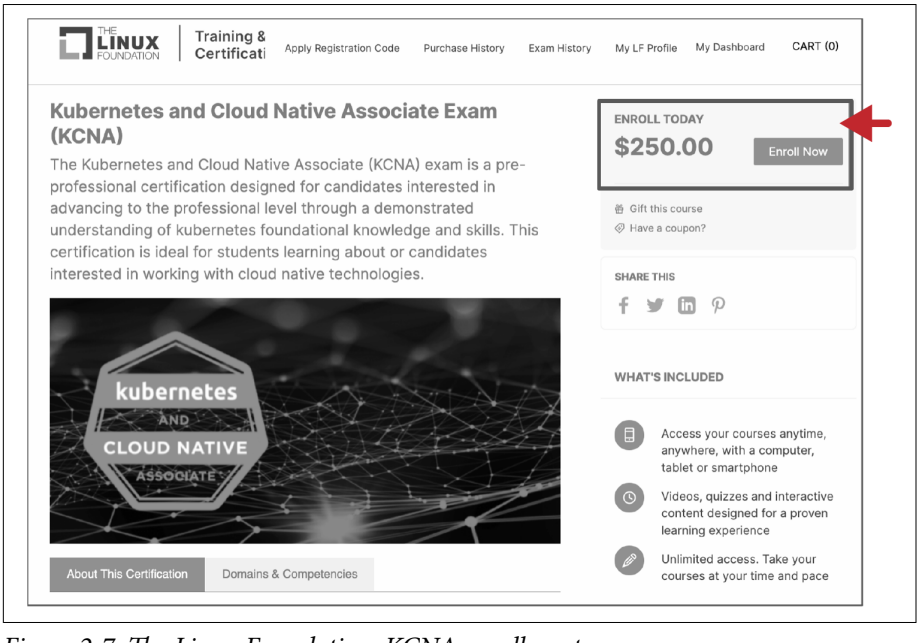


Figure 2-7. The Linux Foundation: KCNA enrollment

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

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

Step 2: Prepare for Exam Day

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 (including your internet connection) works 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 computer 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 online.

Given that your live audio and video will be captured (to proctor and review your KCNA exam), candidates must consent to 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 licenses, 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.

The Linux Foundation also mandates that test takers comply with its examination rules. Some may be intuitive, whereas others are good to know beforehand. For example, your testing location must be clutter free with clear walls. No paper or printouts can hang 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 take the exam. Ensure you show up and are ready to launch the exam within 30 minutes from the appointment time; otherwise you may be considered a no-show and may not be 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 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, you will see a prompt to click Review Exam. The exam will end when the timer completes or when you click Finish Exam.

Step 3: Wait and Check Results

The results do not follow immediately after the exam is completed. Once they're available, you will receive a notification via the email you used to register for the exam. Only at that moment will you 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 be able 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 it's 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 to access the Linux Foundation (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, don't sweat it. The exam comes with one retake opportunity (like any other Linux Foundation exams). If this happens, you will leverage your exam experience and actual understanding of the kind of exam questions to continue studying, focus on your weakest areas of knowledge, and take the exam again.

Let's now take a look at the potential learning and certification paths, beyond the official certification roadmaps, from the authors' applied industry-side point of view.

Potential Certification Paths

Although the KCNA exam is currently not a prerequisite for the 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 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 a KCNA candidate, before and after passing this exam, so you can contextualize where the KCNA stands and how you can both prepare for it and make the most of it for your future learning journeys.

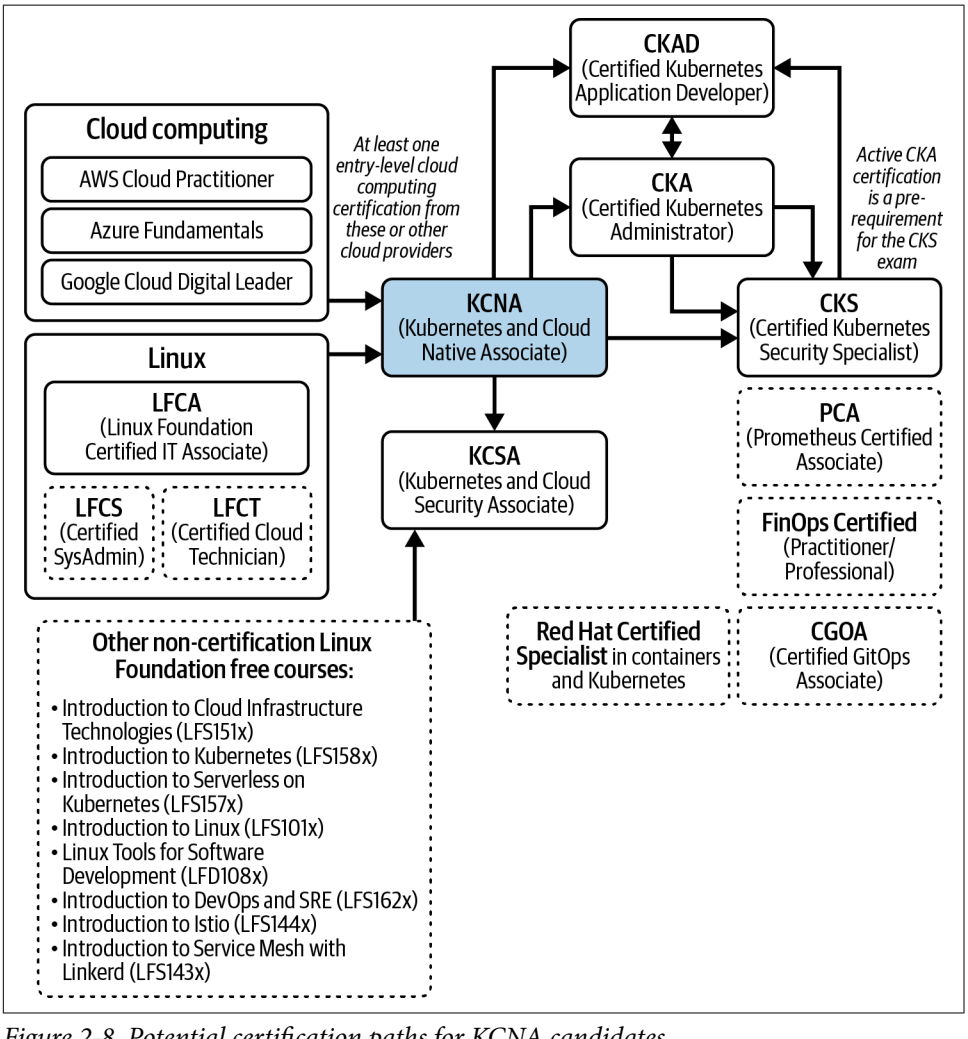


Figure 2-8. Potential certification paths for KCNA candidates

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.

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 exams 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 the 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 (which we will explain later in this list).
  - Based on the industry trends from the Linux Foundation and 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 consider whether 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 relating to hands-on troubleshooting of 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 CKA-certified employees to be eligible. That along with the scarcity of available CKA professionals (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, this certification is next on your list of training. This two-hour exam tests practical knowledge on areas including application design, build, deployment, all the way to maintaining 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.
- As with KCNA and CKA, the market demand is clearly justified. 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 whether it fits your case. Based on statistics, the CKAD is one of the popular options.
- The exam information and requirements are, as usual, available at the Linux Foundation exam page.

Certified Kubernetes Security Specialist (CKS)

While the CKA, CKAD, and KCNA do not have a prerequisite, the CKS requires the candidate to have an active (nonexpired) CKA certification. Does that 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 down the cloud native security road. The job market demand is clearly favorable for this type of role, but it will take you a few certifications to get here.
- Same as before, all information is available at the Linux Foundation's CKS page.

Looking at these four certifications gives you 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.

Case Study: Spotify

Spotify, a leading music streaming platform, is often highlighted as one of the prominent adopters of Kubernetes.[1] The transition to Kubernetes is a part of Spotify's broader journey into the world of microservices and cloud native technologies. Here's a brief overview of the Spotify case with Kubernetes and cloud native.

In its early days, Spotify's infrastructure was primarily based on in-house solutions and bare-metal hardware. As the user base grew, so did the challenges associated with scalability and deployment frequency. Spotify recognized the need to evolve its infrastructure and started migrating to the cloud—specifically, GCP.

Spotify aimed to create an environment where developers could easily build and deploy services without the overhead of managing infrastructure. Kubernetes offered a solution: a platform to reliably deploy, scale, and manage containerized applications. It aligned well with Spotify's move toward a microservices architecture, where services could be developed, deployed, and scaled independently.

1 This section draws from CNCF's Spotify case study.

Spotify's migration to Kubernetes was not just a technical transformation; it was also cultural. The company had to consider the developer experience, ensuring that developers could smoothly transition to the new platform. Spotify built "Golden Paths," which are well-documented best practices and tools, to guide developers in building and deploying applications on Kubernetes. Here are some of the ways the company benefited from this process:

Scalability
: Kubernetes allowed Spotify to scale its services easily based on demand. This dynamic scalability ensured that it could efficiently serve its growing global user base.

Developer productivity
: With Kubernetes handling much of the infrastructure management, developers could focus on writing code and creating features. CI/CD became more streamlined.

Resource efficiency
: Kubernetes optimizes hardware usage, ensuring that resources (like CPU and memory) are used efficiently, which can lead to cost savings.

Resilience and reliability
: The self-healing nature of Kubernetes meant that if a service went down, Kubernetes would automatically try to restart it, enhancing overall system reliability.

End-of-Chapter Knowledge Check

These true-or-false end-of-chapter questions are just a quick knowledge check for you to confirm the understanding of some basic topics related to the KCNA exam. Solutions can be found in Appendix B.

1. True or false: The CNCF is a part of the Linux Foundation.
2. True or false: The KCNA exam is a three-hour long exam.
3. True or false: The KCNA is available in French, English, and Spanish.
4. True or false: The KCNA is a pre-professional certification.
5. True or false: *CNCF* stands for Cloud Native Certification Foundation.

Summary

The goal of this chapter is to provide you with all required information to not only pass the exam, but also see the possibilities beyond the KCNA and how they will leverage the skills you develop in this KCNA journey.

We 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 options in front of you, as well as the specifics of the KCNA certification, its official curriculum, and its relationship with all the preliminary topics we discussed in Chapter 1.

The next chapter will focus on CNCF, its organizational details, some CNCF projects besides Kubernetes, and of course exclusive expert insights from a very relevant CNCF actor.

CHAPTER 3

CNCF and the New Era of Cloud Native

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, and individual contributions. We will cover all that in this chapter. As a KCNA candidate, this information is important to you because it is part of the scope of the exam questions, but it is also very relevant for your cloud native journey.

The Origins of CNCF

To set the stage for the new era of cloud native, let's first examine the origins of CNCF. This nonprofit was created in 2015 to push for the use of cloud native computing and give cloud native technologies a place to call home. It was set up by 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. CNCF is governed by a board of directors and is overseen by the Linux Foundation, the nonprofit organization that promotes the use of open source software. The CNCF website itself is a great source of information and resources.

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, no single organization was dedicated to advancing the cloud native landscape, and many technologies and approaches were being developed in isolation.

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. It 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 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

This section covers a series of events that describe the evolution of the cloud native industry before and after the creation of CNCF (see Figure 3-1).

Several key events led to what we understand as cloud native and containerization today. Let's get started.

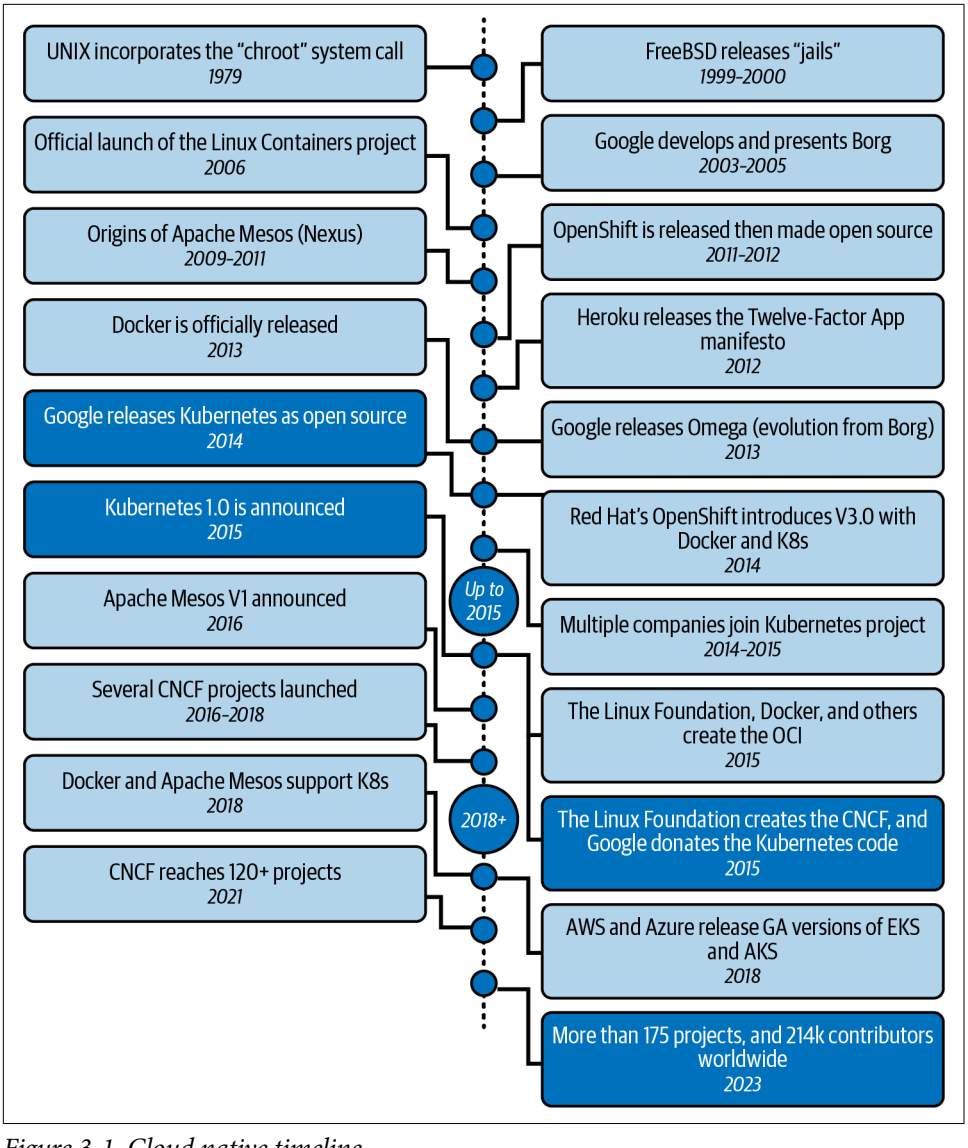


Figure 3-1. Cloud native timeline

Containerization Origins

This initial phase spans from the first Unix/Linux-based systems to the creation of the key baseline elements of modern cloud native:

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 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 a distribution- and vendor-neutral environment for the development of Linux container technologies.

2012

Heroku introduces the Twelve-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 to its success was the almost immediate release of its code as open source, weeks after the announcement. You can read more about its origins in *Docker: Up & Running*, 3rd edition, by Sean P. Kane and Karl Matthias (O'Reilly).

The same year, Google presents Omega, a research project within the company, focused on the next generation of cluster management, which is an evolution of Borg.

2014

Google releases Kubernetes as an open source version of Borg, which opens the code to all contributors and other companies. Joe Beda submitted the first GitHub commit, and companies like IBM, Red Hat, Microsoft, and Docker joined the Kubernetes community. It was the beginning of the Kubernetes era.

The Rise of Kubernetes and CNCF

This second phase extends from the release of Kubernetes and the CNCF to the current scale of the cloud native community:

2014

As an example of the initial traction Kubernetes had with the main industry actors, after a series of collaborations with the Google team, Red Hat decided to support container orchestration with Kubernetes in OpenShift V3.0. This illustrates the early reach of Kubernetes, as previous versions were based on Linux Containers and other in-house developments from Red Hat, but the community support as well as the specialized knowledge of the K8s Google team (thanks to its extensive experience with Borg) were key factors in convincing 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 CNCF was for it to govern the future open source development of K8s, including interoperability and performance for public and private clouds and on premises.

Kubernetes 1.0 is also released in 2015, with a total of 14,000 commits and more than 400 contributors. This version included improvements to App Services, networking and storage, cluster management, performance, and stability.

That same year, the Linux Foundation, with Docker and other organizations, launches the Open Container Initiative (OCI) to create open industry standards for container formats and runtimes. As part of the initiative, Docker donated its container format and the runC runtime (more information on the Docker blog).

2016
: The Pokemon GO video game uses Kubernetes via Google Cloud and becomes a very prominent success story because it was to that point the largest K8s deployment on Google Container Engine.

2016–2018
: During this period, several cloud native projects were launched, including Helm (package manager), kOps (Kubernetes operations), and minikube (for local implementations). Other projects like Istio came from a collaboration of 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.

  At this point, all the main hyperscalers, such as Google, Amazon, Microsoft, and IBM, had released cloud-enabled managed services to support Kubernetes.

2021–2023
: CNCF goes 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 its contributors, has evolved in just a few years, reaching alignment around container-related formats and standards, with high levels of collaboration thanks to open source dynamics. Let's now review the internal governance of the CNCF.

Community Dynamics and Governance

As an open source organization, 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 its central role include the following:

Multi-stakeholder structure
: CNCF relies on a series of groups with different roles, and a distributed structure that enables accountability and collaboration. You will see the committees and groups outlined later in this chapter.

Vendor neutrality
: One of the primary values of CNCF is vendor neutrality. No single company has undue influence over a project, regardless of its 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 lifecycle that includes sandbox, incubating, and graduated stages. We will explore this later in the chapter, but the idea behind this model is to adapt 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. It also supports 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 that 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.

CNCF is governed by a board of directors (officially called the *governing board*, or GB) and is overseen by the Linux Foundation. The 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 you can see in Figure 3-2, several committees and working groups are responsible for specific areas of the organization, made up of volunteers from CNCF member companies and individual members. They are responsible for things like project management, marketing, and events.

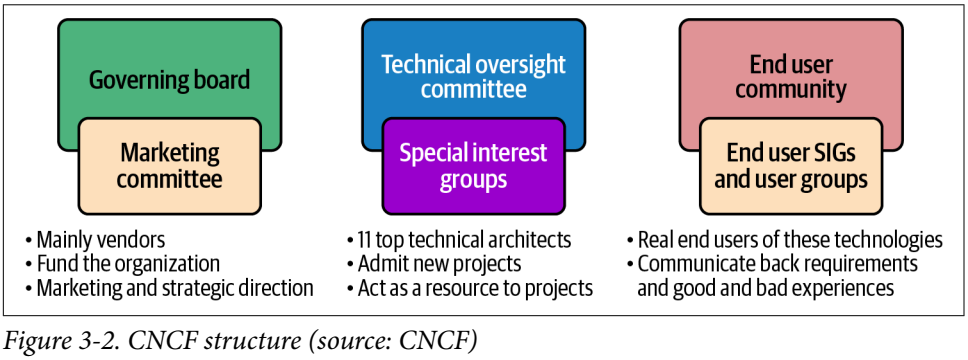


Figure 3-2. CNCF structure (source: CNCF)

Besides the governing board, other key elements of the CNCF structure help make the community dynamics work:

Technical oversight committee (TOC)
: A group of technical experts 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 technical leadership and decision making in different areas such as project oversight and connection with the rest of the CNCF stakeholders.

End-user community
: A group of cloud native adopters 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 (SIGs)
: 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. Intersecting topics such as contributor strategy, storage and security, networking, and others are part of the existing group of SIGs.

Other technical groups
: CNCF hosts a number of technical advisory groups (TAGs) and working groups (WGs). 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 its mission is to create a series of resources that will be very useful for your KCNA journey, so let's explore them.

Key CNCF Resources

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 join the cloud native practitioner community. We will highlight their usefulness for KCNA upskilling and general learning in the following sections.

Community Forums

CNCF boasts an active community that collaborates across various platforms and channels, by maintaining online platforms where members can ask questions, share information, and collaborate on projects. These include the following:

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. We recommend the following:

  - Join the CNCF workspace and explore the 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. We recommend the following:

  - 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 mail lists specific to CNCF projects, working groups, etc. These lists are not specific to the KCNA but will keep you updated with relevant community and project-related information.
  - The CNCF mailing list system is similar to the one utilized by the Linux Foundation. It basically relies on the Groups.io platform to provide scalable email distribution and subscription. 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 with any other repo, it contains code, issues, pull requests, and discussions related to that project. As part of the official repo, a feature called GitHub Discussions includes forum-type 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 later in the chapter.

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, 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 see when the next meeting takes place. As a KCNA candidate, it may be good for you to join some of these meetings to understand the work dynamics of these groups.

Social channels
: CNCF has a YouTube channel with 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 CNCF works. Also, CNCF has a presence on platforms like X/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

CNCF hosts events throughout the year, which bring together experts and practitioners from across the cloud native ecosystem to share ideas and knowledge. CNCF also supports regional events and meetups around the world (feel free to explore the upcoming ones by location). Here are some highlights:

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

  As you learn about cloud native and the KCNA, you should pay attention to any scholarship or sponsorship opportunities 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 one another.

CloudNative Days
: These are smaller, localized events that might be thought of as regional or more focused versions of KubeCon. CloudNative Days events are typically organized in collaboration with local cloud native communities.

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

Cloud Native Security Day and Cloud Native Storage Day
: 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

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 for your exam:

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 on just a single page (or even a website), you can see in Figure 3-3 what the landscape looks like.

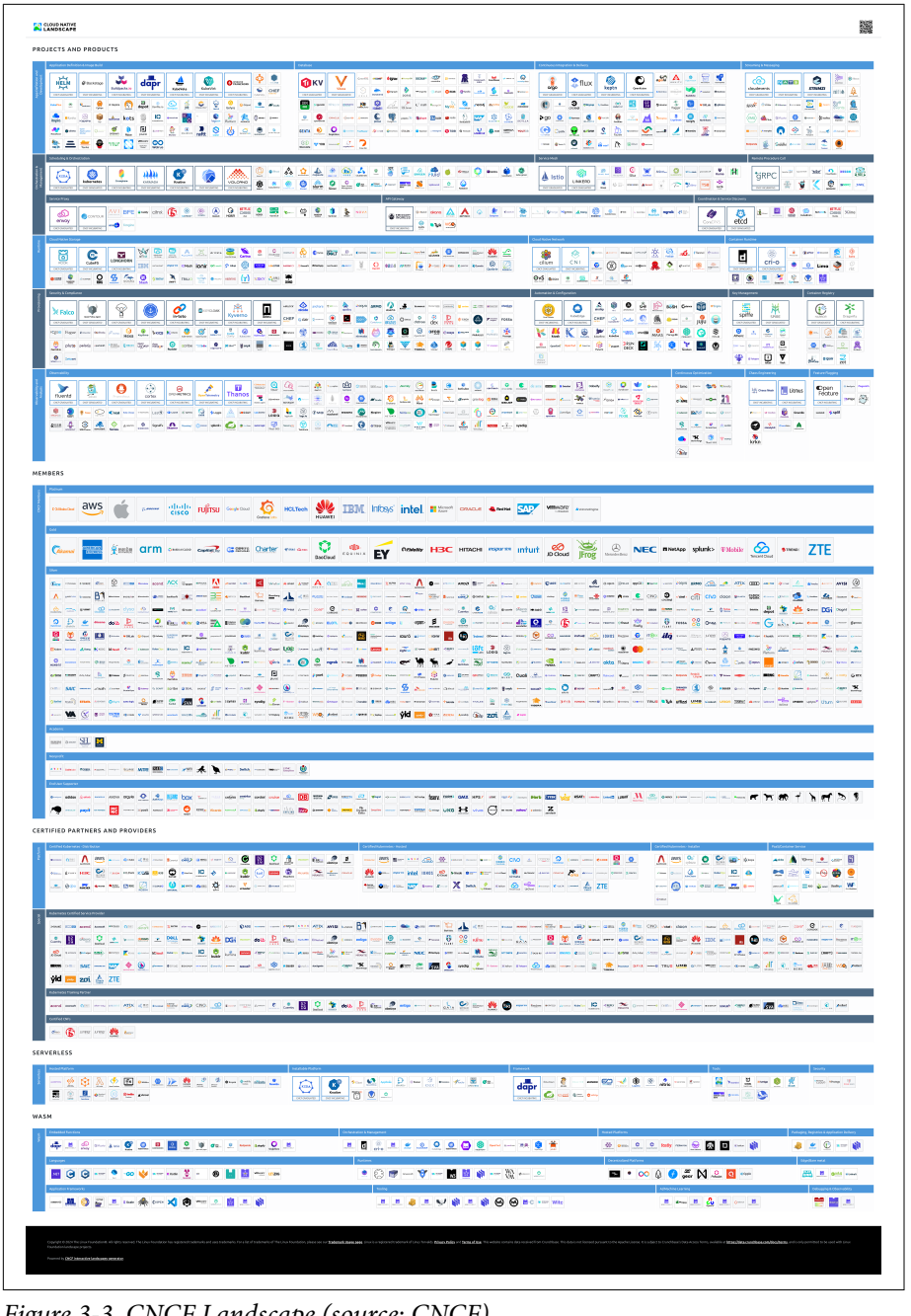


Figure 3-3. CNCF Landscape (source: CNCF)

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 is *one of the key sources of information to prepare for the KCNA exam* (we encourage you to print it out and keep it close to this KCNA study guide).

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 in recent years, including multi-language options (check the top-right menu of the glossary) to see the terms in other languages such as Spanish, Portuguese, Italian, Hindi, Bengali, and Dutch. 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.

CNCF End User Technology 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 to get some recommendations. 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 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 candidates like you to understand current adoption trends and the important projects and tools to learn about.

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 (see Figure 3-4). This is relevant for you as a KCNA candidate not only because of the explanation of the different cloud native steps, but also

the examples of CNCF solutions available for each of the steps (e.g., Argo for CI/CD). We will explain all these projects later in the chapter.

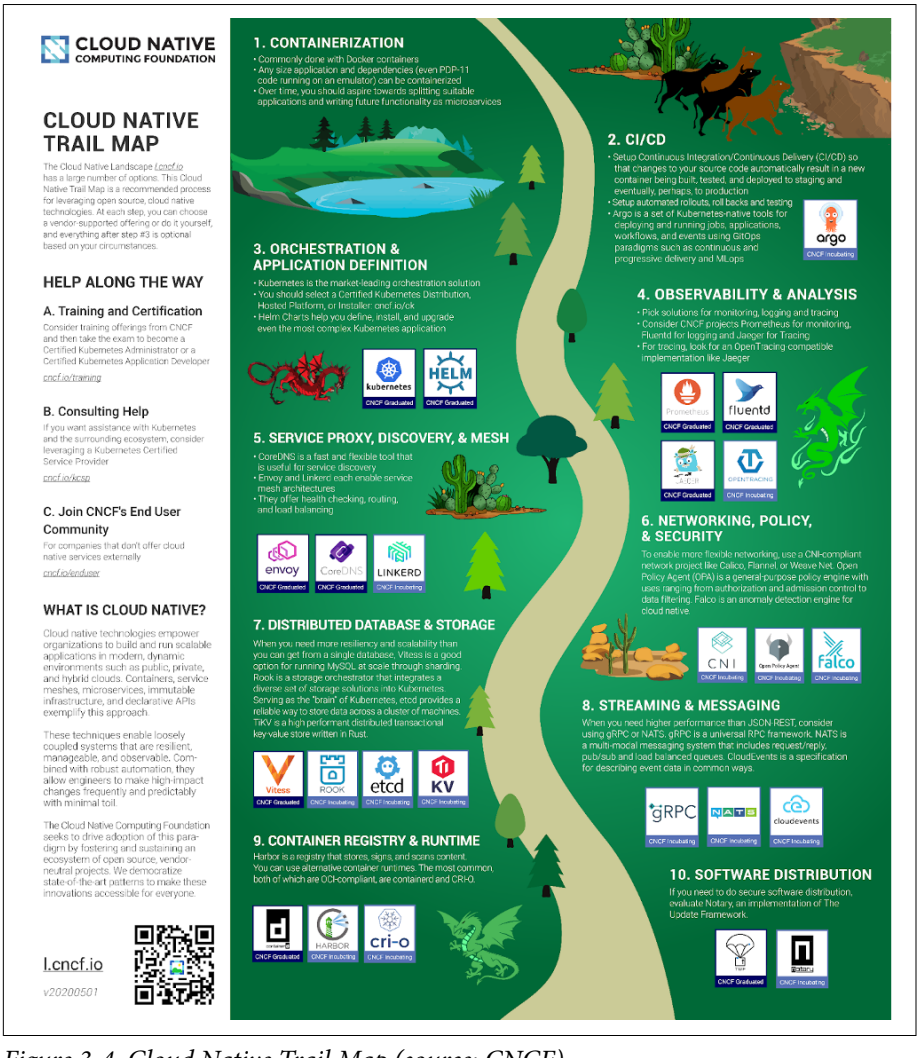


Figure 3-4. Cloud Native Trail Map (source: CNCF)

In parallel with this cloud native journey, you can explore the Cloud Native Maturity Model, a CNCF framework for adopter companies to understand how to progress along five maturity levels: build, operate, scale, improve, and adapt. You can combine this and the Cloud Native Trail Map stages to understand how companies move ahead with their cloud native adoption.

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

Mentorship and Ambassadors

Besides the educational resources, there are additional opportunities to support your cloud native journey, after or during your KCNA exam preparation. CNCF facilitates mentorship opportunities for people to develop new skills by helping with 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 their cloud native careers.

Also, consider taking 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 and content creation.

CNCF Projects and Maturity Levels

CNCF hosts a variety of open source projects for cloud native technologies. Hosting a project at 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 achieve credibility and adoption and garner wide-scale industry support, while 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 the technical, marketing, legal, training, and business development levels.

These projects follow the official levels of maturity of 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 and developer participation. As you can see in Figure 3-5, CNCF has four categories for maturity: incubating, graduated, sandbox, and archived:

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 a 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 CNCF anymore, including the interruption of marketing and other privileges that are available only for active projects. Some examples of archived projects include Brigade, Open Service Mesh, OpenTracing, and rkt.

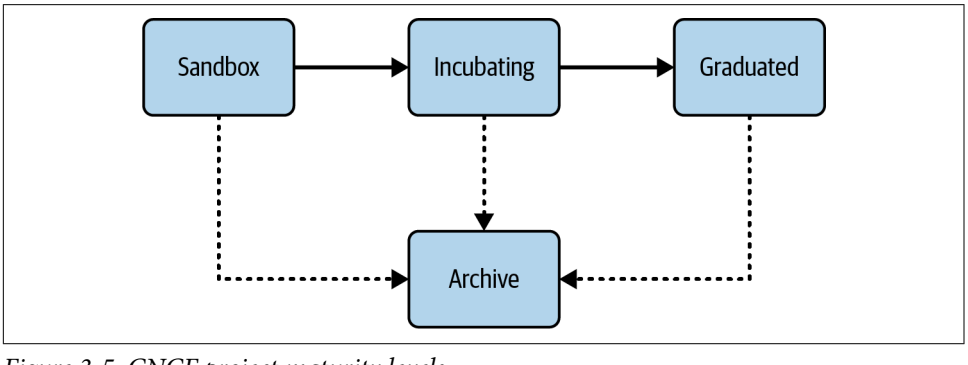


Figure 3-5. CNCF project maturity levels

CNCF projects have two levels of involvement:

Contributors
: Individuals who contribute to the project in any form, including sporadic contributions. This can range from code to 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. Official resources 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 CNCF requirements for each stage of maturity, and the project proposal process, which will extend your knowledge of how CNCF processes work.

Main CNCF Projects for the KCNA Exam

For your KCNA exam, you mostly need to know about graduated projects. You need to know only their main scope of activity, as the KCNA focuses mostly on project awareness. We include some official descriptions from CNCF's website, as they use 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 its 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 parameters of the Argo project, which is one of the most important CNCF projects out there:

- Subprojects of Argo include the following:
  - *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 that 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*
- Cloud Native Landscape card

Cilium

Cilium employs a new Linux kernel technology called *eBPF* that enables the dynamic insertion of security, observability, and networking control logic into the Linux kernel. Benefits to the business include reducing operational complexity and the cost of running highly scalable and dynamic cloud native environments while providing deep security observability and runtime enforcement.

- CNCF category: Cloud Native Network
- Website: *https://cilium.io*
- CNCF page: *https://www.cncf.io/projects/cilium*
- Cloud Native Landscape card

containerd

Defined as an open and reliable container runtime, this provides the basic functionalities required for running containerized applications, with a focus on simplicity, robustness, and portability. It was originally developed as a part of Docker, but it was announced in 2016 as a standalone project, then moved to CNCF one year later. You can get more insights from its 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*
- Cloud Native Landscape card

CoreDNS

CoreDNS is 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*
- Cloud Native Landscape card

CRI-O

CRI-O is a lightweight container runtime that acts as a bridge between Kubernetes and container images, allowing Kubernetes to use any OCI-compliant runtime (i.e., a runtime compliant with the Open Container Initiative, which we will explain, along with other initiatives, by the end of this chapter) 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*
- Cloud Native Landscape card

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 CNCF. It was built to address the specific challenges of microservices architectures, especially in high-scale, production environments. Its 2019 project report is a good way to understand its relevance within the community. Check the *#envoy* Slack channel for some interesting live discussions.

- CNCF category: Service Proxy
- Website: *https://www.envoyproxy.io*
- CNCF page: *https://www.cncf.io/projects/envoy*
- Cloud Native Landscape card

etcd

etcd is a persistence layer of Kubernetes data. It is a very relevant distributed key-value store (i.e., a database with a specific kind of format) for the most critical data of a distributed system. As with other CNCF projects, you can visit its 2021 project report to analyze its community insights and level of adoption, as well as the *#etcd* Slack channel.

- CNCF category: Coordination & Service Discovery
- Website: *https://etcd.io*
- CNCF page: *https://www.cncf.io/projects/etcd*
- Cloud Native Landscape card

Fluentd

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

- CNCF Category: Observability
- Website: *https://www.fluentd.org*
- CNCF page: *https://www.cncf.io/projects/fluentd*
- Cloud Native Landscape card

Flux

Like Argo, Flux is part of the CI/CD ecosystem of tools. It automates the deployment, scaling, and management of containerized applications by 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*
- Cloud Native Landscape card

Harbor

Harbor is a container registry that secures artifacts with policies and role-based access control (RBAC), ensures that 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*
- Cloud Native Landscape card

Helm

Helm is a package manager for Kubernetes that makes it easier to define, install, and manage Kubernetes applications with the use of preconfigured Kubernetes resources (in this context, those resources are called *charts*). Take a look at its 2020 project report and at 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*
- Cloud Native Landscape card

Istio

Istio is 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*
- Cloud Native Landscape cards: Istio and 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. Its latest 2020 project report will complement your knowledge of this relevant project.

- CNCF category: Observability
- Website: *https://www.jaegertracing.io*
- CNCF page: *https://www.cncf.io/projects/jaeger*
- Cloud Native Landscape card

KEDA

From its official description, KEDA is a Kubernetes-based event-driven autoscaling 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: Scheduling & Orchestration
- Website: *https://keda.sh*
- CNCF Page: *https://www.cncf.io/projects/keda*
- Cloud Native Landscape card

Kubernetes

The prior and following 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. As with other projects, you can review its 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*
- Cloud Native Landscape card

Linkerd

Linkerd is 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*
- Cloud Native Landscape card

Open Policy Agent (OPA)

OPA 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, and CI/CD pipelines.

- CNCF category: Security & Compliance
- Website: *https://www.openpolicyagent.org*
- CNCF page: *https://www.cncf.io/projects/open-policy-agent-opa*
- Cloud Native Landscape card

Prometheus
: Prometheus is a very relevant CNCF project with its own certification options (see the PCA exam). It provides a monitoring and alerting toolkit and has achieved great success because of 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: Observability
- Website: *https://prometheus.io*
- CNCF page: *https://www.cncf.io/projects/prometheus*
- Cloud Native Landscape card

Rook
: Rook provides cloud native storage for Kubernetes, for production-level management of file, block, and object storage.

- CNCF category: Cloud Native Storage
- Website: *https://rook.io*
- CNCF page: *https://www.cncf.io/projects/rook*
- Cloud Native Landscape card

SPIFFE
: Also known as *Secure Production Identity Framework For Everyone*, this project 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*
- Cloud Native Landscape card

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*
- Cloud Native Landscape card

The Update Framework (TUF)
: TUF is 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*
  - Cloud Native Landscape card

TiKV
: TiKV is 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 filesystem.

  - CNCF category: Database
  - Website: https://tikv.org
  - CNCF page: *https://www.cncf.io/projects/tikv*
  - Cloud Native Landscape card

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*
  - Cloud Native Landscape card

Obviously, maturity level is not the only sign of relevance for CNCF projects. At the time of writing this book, there are other incubating projects that you may want to explore, such as Thanos, CNI, gRPC, OpenTelemetry, or Kyverno.

This section contains the critical information that you need to know about CNCF projects, 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 improve your understanding of general cloud native culture.

Related Initiatives and Work Groups

Besides CNCF and its projects, other international initiatives and projects are very important to the cloud native ecosystem. They are part of the scope of your upcoming KCNA exam, and they serve as references for any existing or new CNCF project. Let's dive into those:

Open Container Initiative
: OCI is 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 platforms and environments.

Container Network Interface (CNI)
: CNI is 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)
: CRI is a plug-in 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 runtime.

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

Service Mesh Interface (SMI)
: SMI 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 service mesh technologies, such as Istio and Linkerd.

Now let's get some expert insights about the CNCF Landscape from one of the key executives of CNCF, Chris Aniszczyk, who is also a great contributor to the landscape project.

Expert Insights: Chris Aniszczyk

Adrián: Hello and welcome to our series of interviews with experts for the KCNA exam. Today, we're very happy to have Chris from CNCF. Welcome, Chris. Thank you for being here.

Chris: Good to be here and good to catch up.

Adrián: I think that this interview will be very interesting for all learners out there to understand what's going on, who you are, what's your role, what's the CNCF, and also the story behind this exam that I think is so interesting. So let's start with you. Who is Chris?

Chris: I'm Chris Aniszczyk; I have the kind of fun job of being CTO of the Cloud Native Computing Foundation. I helped start the organization as the original executive director almost eight years ago. So in December 2015 we officially held our first board meeting in CNCF to kickstart what was then a very small organization, but with a very important mission of wanting to democratize cloud software and making it freely available for everyone so that it's not just locked into like a Google or Facebook or Amazon of the world. The whole idea was to give everyone the pieces they needed to build modern and resilient cloud native applications.

In terms of my background, I've been involved in open source for 20 plus years. In the early days, I contributed to Linux and was involved in the Gen2 distribution. I worked at Red Hat a little bit. I also had my own open source startup around the Eclipse development environment and developer tooling. So I've been involved in open source from the low-level Linux world to more of the developer tools touching developers on a daily basis through the Eclipse platform. It's been amazing to see not only open source grow in many, many different industries, but also the whole aspect of building an open cloud stack, which is what CNCF has become. We're now 170 people, with three projects covering more than just Kubernetes.

I'd like to answer your earlier question of where did the KCNA come from? Traditionally we've had two certifications for Kubernetes, both CKA and CKD. And those were meant to be very hands-on. The testers are experts already, and they're just validating their skills in a proctored environment. What we ended up seeing is there were a lot of new people who would try it and suffer because it's a pretty difficult exam if you don't have the right background.

Also, we wanted something that covered more than just Kubernetes itself, because there are other projects out there. If you're going to roll out Kubernetes, you're probably going to roll out something like Prometheus or OpenTelemetry to do observability. You're probably going to roll out some adjacent technologies to help with your own custom ingress. You're going to use maybe Envoy or Kuma. There's a ton of projects out there in that space. So the idea was to give a more introductory-level

overview of the cloud native ecosystem, not only Kubernetes, to ensure that beginners and folks who are especially new to the space or maybe new to this area have a great set of fundamentals, knowing the terminology, the basic architecture, and how microservices are generally designed.

That was the original idea as there was no launchpad. If you were a brand-new student, it was pretty difficult to get involved with the traditional exams. Or if you were, let's say a sysadmin your whole life and you're learning, you want to try something new or a bit more modern. It was just very difficult to throw people in the Kubernetes traditional certification mix, because it required too much of a base knowledge of Linux, Kubernetes, and networking fundamentals without teaching people this stuff.

The KCNA serves a good area for students and newcomers to cloud native. So that's where the original motivation came from. Our failure rate, by the way, for CKA, CKD is about 50%, a little over, so there's a lot of people who fail because it's a difficult exam. And so we needed something to help people get there, have a stepping stone, essentially.

Adrián: That's amazing. I remember the initial release and even the beta exam that you released in the beginning, what are the trends, then? Are you seeing more people getting certified and passing the exams?

Chris: So what's interesting is that we're seeing great growth across all of our certifications, but we're seeing a lot of focus on security. We have a Kubernetes security certification, which I think is probably the hardest of them all. It's called CKS. And we're seeing a lot of people starting to take that.

I think the trend that you're looking at is that a lot of companies out there have deployed Kubernetes for a while. They're a little bit more advanced, and they're looking for some targeted training or upskilling around security, or observability, or some kind of cloud financial management. Something to up the level from what they already have from a baseline, since they already have that experience. So that's what we're seeing. There's still a healthy beginners and traditional CKA, CKD, but a lot of focus on security-specific content.

We're also seeing a lot of demand for training and certification in other areas. So for example, Argo, Backstage, and Prometheus get asked about a lot like, hey, what can we do to get up to speed there? Historically, we've been very quiet about providing more training outside of core Kubernetes, which I think has been a bit of a disservice. But we're a small foundation.

A lot of people don't realize we're only about 30 people, and half of those people are mostly dedicated to running events, things like KubeCons. Our conferences are 10,000 plus person events, and having a couple of those, it's a lot of work to put on. So we have a pretty small team that handles training and certification. So we've been a little bit behind.

We do something called Open Source Velocity Measurement every six months for all CNCF projects, and other larger open source projects as well. But obviously, if you look at Kubernetes, it is the most dominant and the largest project in the ecosystem. However, OpenTelemetry is not that far behind, along with Argo, Backstage, Prometheus, Envoy. We've been a little bit slow to have new training. And something you'll see in the future is, we're going to have a lot more introductory and associate-level content for these new projects outside of Kubernetes. Because if you have never used Argo before, getting up to speed with a GitOps workflow management system could be a little bit difficult. Or Backstage, which is a fantastic tool, but very difficult to get started with if you don't know where to begin. So we're looking at providing a lot more content in new projects at the associate level. Another interesting trend is we're seeing a lot of demand from China, Japan, and APAC regions, so we're doing our best to offer more translations for our exams and certifications too. It's been an interesting uptick the last 18 months from APAC. It's outrivaled everyone else in terms of growth.

Adrián: It's interesting to see that the KCNA is already creating trends for other projects beyond Kubernetes, where people can stay on the associate level. That's great.

Chris: That's for sure what we want to provide people. The ecosystem is so large that there's so much more to specialize in beyond Kubernetes, right? There are jobs out there for people if you know Backstage, OpenTelemetry, Argo. That's in the job descriptions now. So I think it's just as useful to have.

Adrián: We get this question very often, people are saying, OK, after the KCNA, what do I do? Is the CKA maybe the obvious option because of administration, or should people go directly to security or application development?

Chris: Yeah, our most common path is that people take the KCNA first. And then most likely after that, they take the CKAD, which is more geared toward the typical person deploying in a Kubernetes cluster, more application developer than admin of said cluster. So we see the intro to normal app developers is the most common pathway. I think 75% of people go along that path. And then the others are all over the place. You know, some do security, some are doing the Prometheus associate exam. But generally the path is to begin with KCNA and then jump to CKAD for the more advanced track, where you're proctored in front of like a command-line prompt trying to solve problems with your YAML, is the best way I can describe the CKAD.

Adrián: Yeah, it's not an easy one.

Chris: No, and people don't need to be discouraged because the pass rate is about 50% for the first time around. People take it multiple times and they do better. So don't be discouraged. It's almost like college preparatory exams. In the US we have something called the SAT, and I didn't do so well the first two times, but the third time was great. So don't be discouraged. People get a free retake too, if I recall, for

that. So, don't be discouraged if you don't pass the first time. It's all about learning and improving.

Adrián: Yeah, totally. And regarding the KCNA, because I imagine you have all sorts of analytics at this point about the profiles and the easiest topics, and those topics that are a bit more difficult for people. What are the classic knowledge gaps for the newcomers?

Chris: So the biggest knowledge gap I've seen is Linux fundamentals. We see a lot of folks who just don't really understand it. They'll do their best to pass the KCNA. And they'll pass it, but with very little Linux fundamentals, they struggle with where to go next.

And we do have another separate exam called the LFCA, which is hosted by the Linux community that gets people up to speed with the basics of the command prompt, bash, how networking works, sockets, and all this stuff. I think if people truly want a career in this space, they should get up to speed with all the different cloud native technology, the terms, architectures, but still get that really strong set of Linux skills, because anytime you're going to debug an issue or run into something weird, having Linux debugging skills, and just knowing your way around how the Linux kernel works is going to make your career super valuable. Because companies want this, and you're going to be better at your job.

We don't always need to know how everything works. It's almost like a car, right? Not everyone knows how to replace an engine, but maybe we know enough to do an oil change; we can learn about it. You want to learn enough to know how to do your own oil change or debug Linux networking, but you also don't have to know how everything works. The basics are going to serve you for life. And I think that's the most important bit of advice I could give folks in terms of gaps; do the work for KCNA. But please, please, the Linux fundamentals are going to serve you your whole life in many ways.

Adrián: We're seeing some patterns in terms of technical management, management people who are taking the exam and trying to learn because they will be working with different admins and application developers. Is that a representative group of learners?

Chris: Yeah, definitely. A lot of folks fit that profile. I think it's split between beginners or students and the people who are in technical career paths and want to move to the more Kubernetes side of the house, whether they're sysadmins, VMware admins, Windows machines admins. They're doing a lot of this kind of learning. And the other third is just a mix of everyone; it's hard to point out, but mostly people in companies. We have some companies in our ecosystem that actually mandate that their developers take the KCNA and then CKA as a requirement of employment, essentially, which is fascinating to see. It's good for people who are looking for jobs,

because this is something that helps. And I always tell people, if you have a CKA or CKAD, finding a Kubernetes-related job is pretty easy. There's so many companies looking, it definitely helps.

Adrián: It has a positive impact in either situation, because you can be working for a company where you can utilize your skills or it can help you find new professional opportunities.

Chris: Definitely. And the other thing I tell people, if cost is an issue, we have lots of scholarships for training. At KubeCon, we give away hundreds of scholarships per event where we fly people in. To me, if you really want to get involved in the community, start to learn about it, take the KCNA, and then try to attend a KubeCon because you'll meet great folks, you'll have great career opportunities. It's truly a very unique and friendly, welcoming, inclusive global community.

Adrián: It is. And they are all very aware that these are complex topics. It's a work in progress. We're learning all the time. People are well aware.

Chris: Yeah, definitely. You know, what's fascinating is early on we had a North American and European focus in our community. And now in the last couple of years, especially, we're seeing a lot of interest from South America, from APAC, and we're getting a more global community happening here. And if you look at cloud adoption, it almost maps cloud adoption where North America, Europe were fast to adopt public cloud, other geographies were a little bit slower, but they're now all finally catching up. So we're getting almost like a second wave of adoption from my perspective.

Adrián: The glossary is a good example of that. We have the glossary in different languages as people are getting involved to do it.

Chris: I think we have 10 or 11 languages now. The glossary is the first place I always send people; if you want to contribute or learn, learn the basic terminology, because any technology is going to be hard to get into if you don't understand the words and the lexicon. And I think the KCNA took a lot of inspiration and terms and layout from a lot of the stuff that was done in the glossary, too. We try to always link back to it.

Adrián: Certainly. The glossary and the landscape. These are great places to go to.

Chris: The lovely landscape, yeah. As the primary maintainer of the landscape, I love the love and hate that people have for the landscape. Some people are like, oh, I love it. Others say, oh, it's too complicated. But I'm like, look, distributed systems are complicated. This is our industry. There's a lot of companies; there's a lot of options. And I think it's a more real view of the world. There's not only one way to do stuff; there's always multiple ways. That's the beauty of open source and cloud native, in my opinion.

Adrián: Totally. And you have mentioned the events, like KubeCon, etc., but what are the ways for people, especially newcomers, to get involved and to get some real experience by getting involved in different projects? How can you do that?

Chris: So obviously we talked about the glossary; that's a great introduction. Go to *glossary.cncf.io*; you can contribute translations there. But one thing we do in CNCF is every year, we run three mentorship seasons called CNCF mentoring. Well, if you go to *github.com/CNCF/mentoring*, these are project ideas that you get paid a stipend for almost like a summer internship, but we do it a few times a year. It might be something like, how to make an improvement to a project, maybe write a performance testing framework, implement a feature, help debug an issue. And these are paid projects over a certain two- to three-month horizon where you get paired with a project maintainer, a mentor to help you. If you have any inkling of wanting to get better at programming and contributing to open source projects, this is a fantastic program to do. We've had something like 600–700 people that have gone through the mentoring process at CNCF. They've been hired as interns and even full-time. It's truly a wonderful program that I encourage. If you're new and truly interested in programming and development, the mentoring is great and you get paid a nice little stipend to work on things, which is lovely.

Adrián: That's amazing. So what would be your final recommendation? You have shared so much information that will be valuable for the learners.

Chris: Walid, one of our great community members out in the Middle East, has a great set of guides on GitHub, which I generally point to. James Spurin has some great content on YouTube also that I generally recommend people get started with. We don't have a prototype simulator for KCNA, because it's a different type of exam. But if you're starting to learn these concepts and you're like, hey, I want to go maybe experiment with how to mess with the Kubernetes cluster or fix something, we have something called Killer.sh. Anyone who signs up for both certified Kubernetes exams gets to take practice sessions. This is the equivalent of running a simulator, and you could practice there. And they do have a simulator for our LF, our Linux fundamental courses. So if you're looking to get stronger on Linux stuff, Killer.sh is fantastic. If you do this, I think you're going to be in a good place.

And the other thing you can do is attend our community meetings. There's a lot of projects that have public meetings. There's a great contributor experience group in CNCF that will always help. You could just go to *contribute.cncf.io*, find some resources, jump on Slack, and people will help you with your questions, whether it's around KCNA help or it could be hey, how do I contribute to this project? How do I download this? It's just a great friendly community.

Adrián: Yeah, indeed. It's a safe space to ask questions.

Chris: Sometimes people are shy. Sometimes it's cultural, but there should be no shame in asking questions, right? This is the open source spirit. I got involved in Linux 25 years ago and I was asking questions on newsgroups or Usenet back in the day and people would just answer it with no expectation in return. They just wanted to help a fellow Linux traveler, a fellow developer, and the CNCF community is the same way. People are very, very helpful, but instead of using that, we're using Slack and GitHub.

Adrián: Yeah, it's different tools for the same purpose, but the same spirit indeed. Well, this has been amazing, Chris. This is all very useful. Thank you very much for your time.

Chris: Thank you. And thanks for being involved in the community and helping people out in their cloud native journey. I think the more we can do to help beginners and newcomers feel welcomed and be able to learn to get the necessary skills to be cloud native experts, the better we're going to be off as a society and world, because we're upskilling folks, giving people great opportunities in life. So thank you very much.

Adrián: Thank you. Have a nice day.

End-of-Chapter Knowledge Check

Here is our second set of true-or-false knowledge-check questions, this time related to the CNCF information we covered in this chapter. Use them to validate your understanding of the cloud native community and its dynamics. Solutions can be found in Appendix B.

1. True or false: Both CNCF and Google's Borg were created at the same time.
2. True or false: Kubernetes is a CNCF graduated project focused on container orchestration.
3. True or false: Prometheus is primarily a service mesh solution.
4. True or false: Linkerd was one of the earliest service mesh projects to join the CNCF.
5. True or false: Helm, a CNCF project, is a package manager for Kubernetes applications.

Summary

This concludes our overview of the CNCF and cloud native ecosystem. This chapter, along with the first two, gives you the end-to-end context of your KCNA exam, including a rich variety of explanations and external resources that will accelerate not only your exam preparation but also your general cloud native upskilling. The next chapter will focus on cloud native terms, so you are fully equipped to learn about Kubernetes in Chapters 5 and 6.

CHAPTER 4

Essential Concepts for Cloud Native Practitioners

In this chapter, we will explore essential concepts related to cloud computing. Having a solid understanding of the basics allows you to build upon them for advanced implementation. Every exam candidate will have different levels of preexisting knowledge; to give you a solid base, we will cover cloud-related topics end to end, from the basics to other advanced concepts that you will leverage later in Chapters 5 and 6 for Kubernetes.

Concretely, we will cover an 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 evolved 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 use the word "cloud"?

The *cloud* refers to servers that are remotely accessible over the internet. Software and databases 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, CNCF defines cloud computing as follows:

> 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 (CSPs) 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*. This approach 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 single piece of traditional 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 with specific memory, networking, storage, and computational capacity and 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 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 organizations to respond quickly to 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, *microservices* are like a colony of bees. Bees come together to carry out many well-organized functions. The colony's shape and size is easy to adapt. Bees 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 one another 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 another, regardless of them being in the same application.

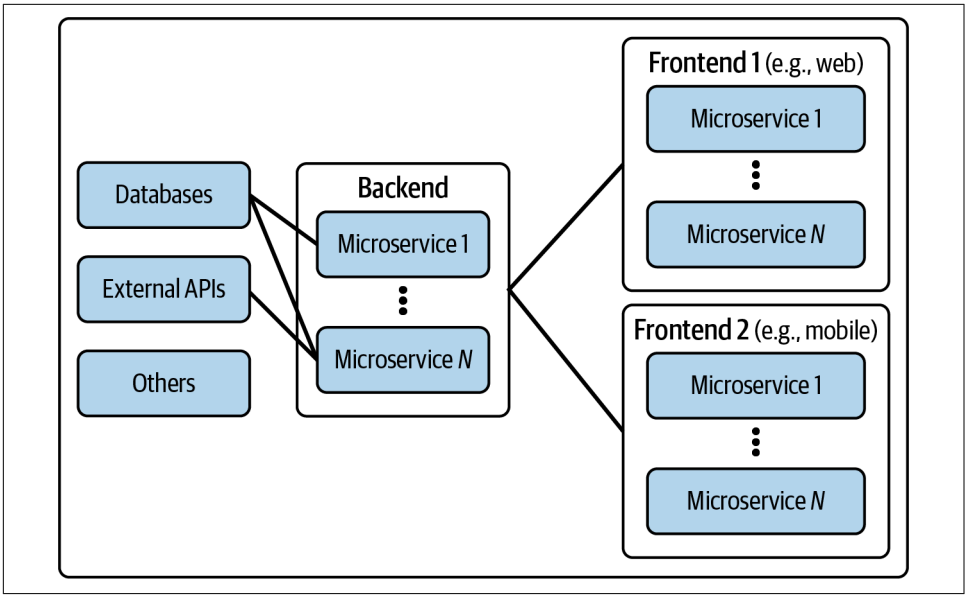


Figure 4-1. Microservices-enabled development

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 piece of hardware. The hardware can similarly be smaller and separated. Each microservice can find the hardware that best suits its 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 AWS 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, cloud computing enables 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 why microservices and cloud computing are a match made in heaven. Together, with containers (see details in Chapters 5 and 6), they are a trifecta of modern cloud application architecture. *Containers* fill in the missing piece in making microservices run easily anywhere. They can be considered an add-on package that isolates the microservice from dependencies and the operating system. Containers make microservices self-sufficient so they can 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, you 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 users to manage the containers across the clouds.

This application architecture is crucial in the popularization of cloud computing in modern times. The commercialization of the cloud depends on its connectivity and users' ability to access it from anywhere. In the next section, we will explore the different ways that 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 up-front 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).

The commercialization of cloud computing technologies offers *different levels of service*, as you can see in Figure 4-2. Basically, there are trade-offs between financial cost and technical control, giving adopters the option to choose the model that suits them better. The "as a service" model has different levels of implementation as described in the following subsections.

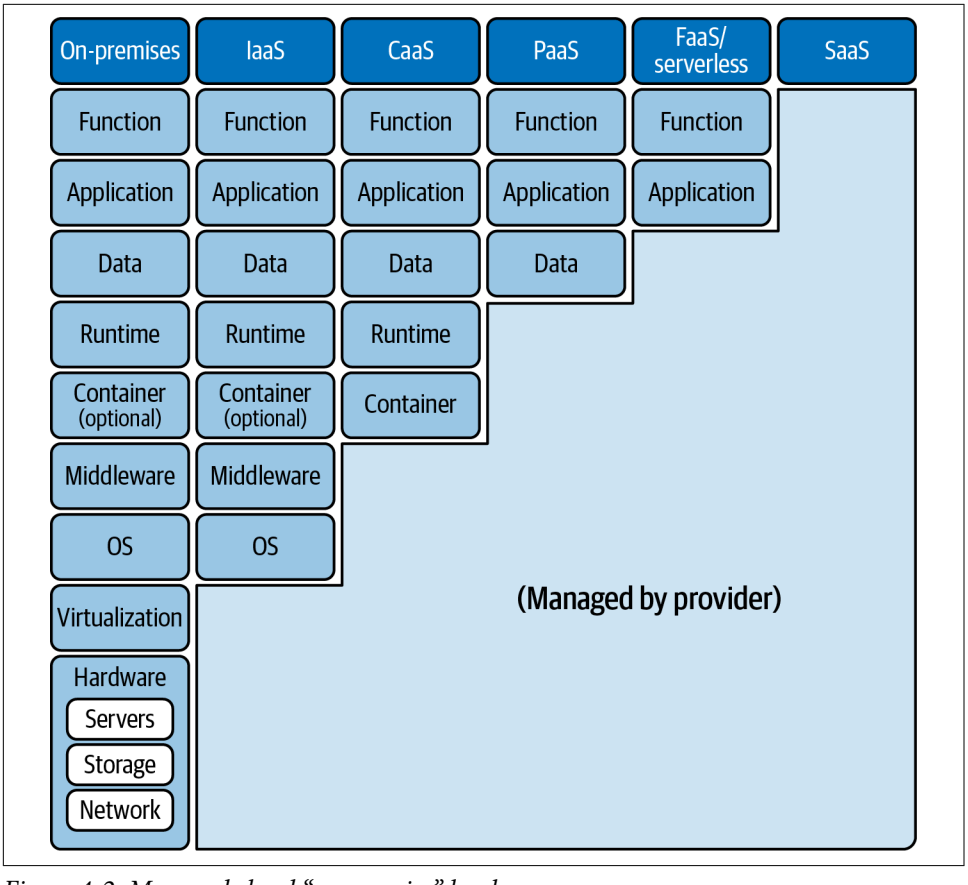


Figure 4-2. Managed cloud "as a service" levels

Infrastructure as a Service

In the *infrastructure-as-a-service* (IaaS) model, 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 requirements. In this case, the users rent bare-bones 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 IaaS is appealing to users is to reduce capital expenditures and transform them into operational expenses. Instead of procuring the hardware and building their own cooling server room, users 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. For businesses with changing and unpredictable needs, IaaS allows users to respond to needs by quickly increasing or decreasing resources as needed. 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 users at all times, even when there is a problem with equipment. This allows the organization to concentrate on strategic business activities.

Examples of IaaS offerings are Amazon Elastic Cloud Compute (EC2), Microsoft Azure Virtual Machines, and Google Compute Engine.

Platform as a Service

In the case of *platform as a service* (PaaS), cloud providers equip their users with a platform to run applications, 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 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 this option attractive in scenarios where they want to quickly build, test, and deploy applications.

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

Software as a Service

With *software as a service* (SaaS), CSPs offer applications plus the required platform and IT infrastructure over the internet. Cloud providers build, maintain, and offer applications for 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 the building as if they own it most of the time. However, they know that while the landlords will fix things when they are broken, not all functions can be customized according to their needs.

SaaS makes it easy for organizations to implement new technology with 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 by users. Popular SaaS offerings that you may be familiar with include Salesforce, Google Workspace, Slack, and Dropbox.

Container as a Service

While you may be more familiar with IaaS, PaaS, and SaaS, *container as a service* (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 OCI and 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.

Function as a Service

In the *function-as-a-service* (FaaS) model, 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.

The 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 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 and 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 by different clients that are logically isolated within their own work environments. However, let's dive into a comprehensive list of all the options.

Public Cloud

For the *public cloud*, 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, GCP, and Azure from Microsoft.

Private Cloud

In a *private cloud*, the whole "cloud" is for one single organization. It is commonly used in environments requiring a very high level of control. Private clouds allow users to leverage all the advantages that come with cloud computing but with the access control, security, and resource customization of an on-premises infrastructure. Some organizations may opt for a private cloud if they deal with sensitive and confidential information or strict regulations that mandate 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 a hybrid cloud solution.

Hybrid Cloud

Hybrid cloud refers to utilizing both public and private clouds for an organization's cloud computing, storage, or service needs. Sharing the load between private and public cloud options allows the organization to create a versatile setup based on its needs. This approach is one of the most common setups today because of its ability to optimize usage and costs, including risk sharing.

Multicloud

Multicloud refers to a type of hybrid cloud setup (see Figure 4-3 for an example). 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 offerings of services (which may be different depending on the combination of providers).

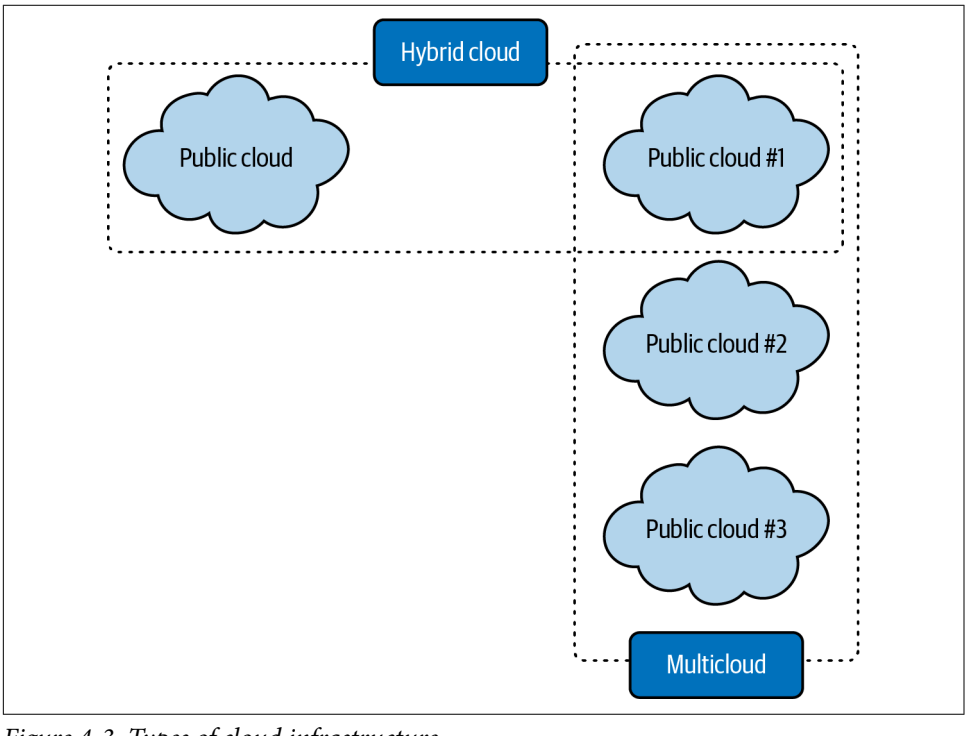


Figure 4-3. Types of cloud infrastructure

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 in.

The Virtualization, Containerization, and Orchestration Trifecta

At this point in the process, organizations 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. They will likely 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 encapsulates 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 as follows:

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 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 the following:

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 you can see in Figure 4-4, lighter weight is one of the key advantages of containers.

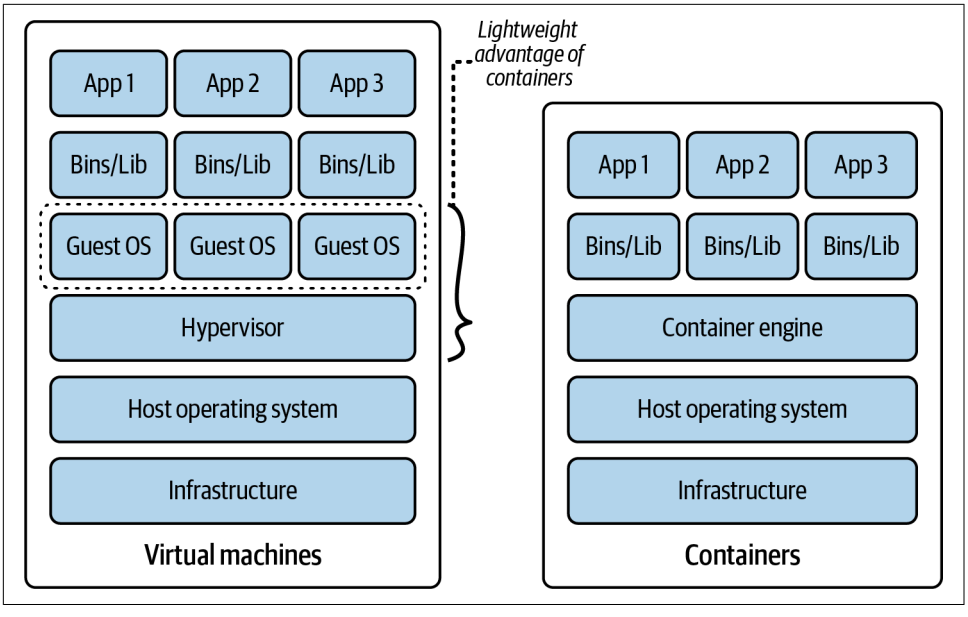


Figure 4-4. Virtualization versus containerization

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 that each handle 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 discussed, 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 CSPs to store and process data, run applications, and perform computing tasks. Examples of cloud computing service providers include AWS, Microsoft Azure, and GCP.

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 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 take full advantage of cloud computing capabilities and leverage 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. Developers 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 explain later in this chapter) and CI/CD, you have the key building blocks of cloud native, as illustrated in Figure 4-5.

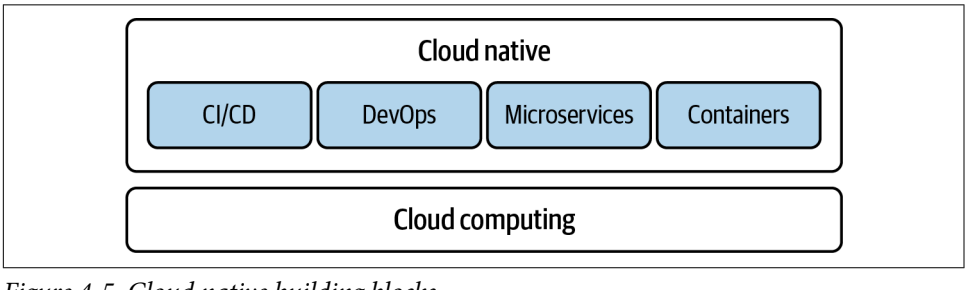


Figure 4-5. Cloud native building blocks

Let's discuss a couple of cloud native examples. Netflix is a prime case of a cloud native application. It utilizes microservices architecture and runs on the 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 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

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

Heroku created this methodology in 2012. It outlines building SaaS apps that can be written in any programming language, using any combination of backing services (database, queue, memory cache). The 12 factors are outlined here:

1. Codebase
: A Twelve-Factor App 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.

2. 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.

3. 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.

4. 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 Relational Database Service (RDS) instance, for example.

5. *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.

6. *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.

7. *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.

8. *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.

9. *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.

10. *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.

11. *Logs*
    Treat logs as event streams. Instead of managing logfiles, apps should produce logs as a continuous stream of events and let the environment handle storage and archiving.

12. *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 App 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 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 its official glossary of terms. Besides everything we have reviewed up to now, and the next two Kubernetes-first chapters, here are some fundamental terms you need to familiarize yourself 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 who 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 issues 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 a "left shift," as in shifting these steps to the left closer to or even before code is written.

See Figure 4-6 for an illustration of an iterative DevOps cycle, a series of steps for the software development and release process.

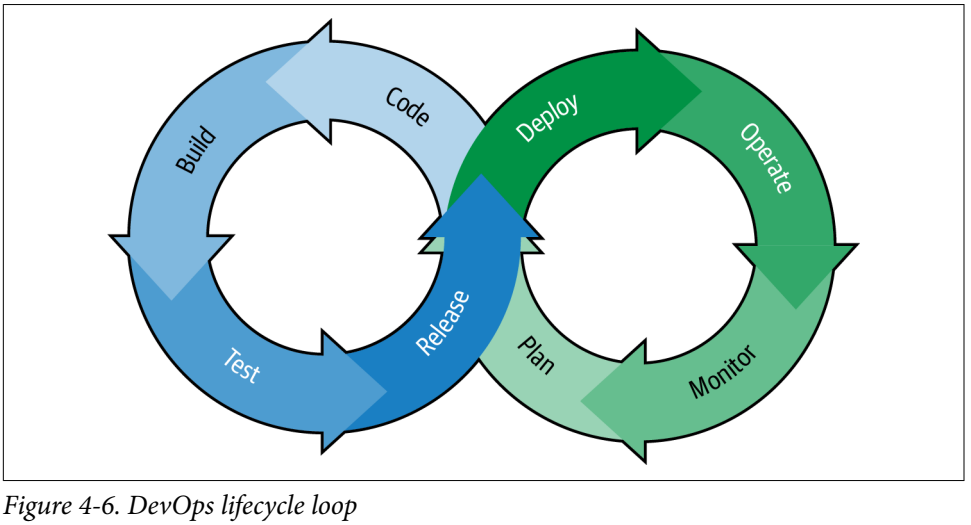


Figure 4-6. DevOps lifecycle loop

Let's discuss each of the eight steps:

1. *Plan*
   In this initial phase, 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 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 CI/CD pipelines.

6. *Deploy*
   The software is deployed to production servers. This step is sometimes combined with 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 Twelve-Factor App methodology lead to the concept of *CI/CD*. As noted earlier, 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 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 that 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 Infrastructure as Code

GitOps is a DevOps method 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.

GitOps 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 approaches from the public cloud providers.

Declarative Versus 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
: The declarative approach 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 it 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 Versus 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. For example, consider online banking transactions or a shopping cart at an online grocery store.

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. For example, think of a REST API in which each transaction is managed independently and there is no data retention from previous transactions.

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. These terms are applicable to Kubernetes with regard to the direction of the traffic coming to and leaving from the cluster. More specifically:

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 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 because of 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, which will exclusively focus on Kubernetes and its technical details. But before that, let's turn to the 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 renowned cloud native consultant and the author of the O'Reilly CKA, CKAD, and CKS study guides.

Expert Insights: Benjamin Muschko

Adrián: How are you, Ben? Welcome.

Benjamin: Hi, thank you. Doing pretty good. How about yourself?

Adrián: Doing well. Enjoying the day and having a blast sharing some time with you. We really appreciate that you are joining this series of interviews with different Kubernetes and cloud native experts.

Benjamin: You're welcome.

Adrián: So just for those who don't know you, I don't think there are a lot, but in case they don't know you, who is Ben and what's your relationship with the Kubernetes ecosystem?

Benjamin: Yeah, so I work as an independent software engineer and technical trainer, oftentimes actually with O'Reilly. It's one of my main content providers. And when I do consulting on cloud native migrations, I would say the move toward Kubernetes is oftentimes a big topic. Not only I would say Kubernetes, but also the migration toward microservices, containers, and so on. So I think that's my background. So ultimately, when I work with clients, Kubernetes comes up in many cases, I would say, because organizations simply want to migrate to the cloud. And that usually means Kubernetes these days.

Adrián: And I assume that you are also within the CNCF and Kubernetes communities.

Benjamin: Yeah, that's right. I've been trying to get into contributing to some of the open source projects. I just got started with the OPA project, Open Policy Agent. I am trying to get into that a bit more. I was actually really just starting to look into the list of issues that they have, and the contributions they're looking for. That's what I'm going for right now. I've not been contributing too much to the actual Kubernetes project itself. They have this big ecosystem with a lot of contributors there. But it's great to see what's around.

Adrián: Yeah. Since you have been working on this area for so many years, do you remember how hard or easy it was to start working with Kubernetes and, in general, with microservices and containers as an initial learner? Because these interviews are oriented to people who are starting to learn about the Kubernetes and cloud native ecosystem.

Benjamin: Yeah, moving toward microservices is somewhat independent of a specific technology. So it's more a methodology where you have to think about how to cut up your monolithic application into chunks, how you want to expose it, how you want to run it in the cloud, potentially. So I would say that is more a discipline than anything technological.

Oftentimes, people bring in Kubernetes or containers in that context, but it doesn't necessarily have to mean that you have to run it on the cloud or in a container. But generally speaking, when we talk about containers and Kubernetes, containers are relatively easy to pick up. I think their hype days are somewhat over nowadays since it's not super new anymore. So I would say that it wasn't too hard for me to pick it up.

Though the general usage of containers is relatively easy, when it comes down to best practices, it looks slightly different. Especially when it comes down to security aspects, how to build your container image properly, how to integrate it into your CI/CD pipelines, for example. Those are usually concerns that come with the notion of wanting to move toward containers.

In terms of Kubernetes, when I first looked at it, it wasn't necessarily apparent where to start. I mean, you obviously would start on the level of Pods, at least if you look at

it from the perspective of an application developer. So getting something to run in a Pod is relatively easy. But then when it comes down to actually operating applications at scale and exposing it to end users and picking the right product that you want to use as your Kubernetes cluster, I think those aspects are more complicated. And if you look at documentation, there's simply a lot of it. So it's hard to find a good starting point where you would say, OK, look at this first, this will give you a rough understanding about Kubernetes, and then you're good to go. I don't think it's like that.

How I usually look at Kubernetes is from the perspective of a specific role. What capacity are you working in? Are you looking at it from the perspective of, say, an administrator who wants to manage a Kubernetes cluster and everything that comes with it? Or are you basically a developer, where you work with an administrator who sets everything up for you and manages nodes and security aspects and things like that and you just want to operate your application? I think those are different perspectives on how to look at Kubernetes, and also how to learn Kubernetes, if you're somewhat new to it.

My clients are often application developers. They usually have a hard time grasping certain concepts and where to start. Often, the issue is time. So application developers come in and say, OK, I need to deploy something to the cloud. I know how to write my application independent of the language that you're using. But now I have to learn everything around Kubernetes. Where do I even start? What do I need to understand in terms of fundamentals and the Kubernetes primitives that help me run the actual application inside of a Pod? And not just a Pod, but anything that comes with it. Security, scaling, and so on. From that perspective, it's hard to know where to start.

Adrián: I think part of the goal of this book and the certification itself is just to give some structure to all these areas of knowledge. The architecture part, the architecture mindset, how to connect the pieces regardless of the technology or the provider we have. It doesn't matter if it's a private cloud, a public cloud, but all the security aspects, all the development aspects. How do we gain actual experience in Kubernetes and related technologies?

Benjamin: The certification curriculums are a good start. I think they provide a very good structure for anyone who's new to Kubernetes and they just want to learn what's really important to them, to their specific role. They can really go into it, learn it from scratch, independent of a cloud provider or a specific cloud environment. You simply just go ahead and set up minikube if that's the thing that you want to use. Obviously, it's not production ready, but once you gain experience, you can easily apply that knowledge to your cloud provider.

But generally speaking, I would say a good starting point is the Kubernetes documentation. I think they put in a lot of improvements over the past couple of years. When

I first started to look at it, it was somewhat overwhelming. I think the structure has changed and improved quite a bit. And they try to bring in best practices as well.

I wish there were a bigger focus on tutorial-style documentation as well, or at least as a complement to the reference documentation. If you look at other ecosystems, I think one company that does a pretty good job with that is HashiCorp. They usually have their reference documentation and then they have their tutorial-style documentation as well.

But generally speaking, there's a lot of information on the web. So if you Google for it, you can spend weeks going through tutorials and best practices. I think some of the prominent companies in the Kubernetes space have their own blog pages as well. And they are pretty useful when it comes down to picking up best practices, I would say.

Adrián: Do you have any favorite resources out there, like from companies, from experts?

Benjamin: There are many for different aspects. The Cilium project sometimes has pretty good blog posts. But there are so many companies now in the space so you can find very specialized aspects as well. For example, using Kubernetes at scale or multi-tenancy, things like that. Those are topics that are usually backed by the companies that have a special interest in that space too.

Adrián: Yes, I totally agree. The cookbooks and the how-tos and the manuals and the architectural views are all very intuitive for people to start learning. At the end of the book, we have a chapter dedicated to additional resources, community resources, because this is a living area of knowledge. It's easy for people to continue learning and exploring because there is so much.

Benjamin: I think it's always good to build that bridge from, I wouldn't say necessarily theoretical, but from the general content as a reference toward how do you actually apply this in a real project too? And what are the challenges you would potentially face? Anything that you know up front, even if you're a beginner to Kubernetes, will be helpful. Just a simple example of security aspects or resource management in general, that you can assign to Pods or applications that run inside of a container. If you already know about that, you can help yourself quite a bit when it comes down to production-level issues that you may run into in general.

Adrián: Totally. And that applies to Kubernetes, to cloud native, to general cloud computing. One thing is the theory of how you deploy things and then when we have a specific situation, experience is an additional value for these situations.

What do you think about the different Kubernetes-related certifications, exams in general, without going into detail of this one, but what do you think that the most challenging parts are for a Kubernetes exam? Where do you think that people will struggle the most when they are trying to pass these certifications?

Benjamin: I believe the certifications are a good concept, a good credential to have for anyone who tries to either get into the space or wants to show that they have the knowledge in that space already. I know that some folks like to acquire these certifications if they want to potentially transition into a different role. Maybe they already have worked in some sort of DevOps role, but they want to get more deeply into Kubernetes. From that perspective, the certifications clearly lay out a curriculum. They give you the building blocks that you can follow to have the most basic understanding about what this role involves when it comes down to Kubernetes. So I think in that way, they're super useful.

Independent of finding a new job, I think you can use these certifications to improve your knowledge as a developer, as an administrator, or whatever your role may look like. Don't view these certifications as something you do quickly just to gain the badge or credential to put on your resume. See it as the starting point for your career to become more proficient with Kubernetes in general.

But I would say generally speaking, if you've taken one of these certifications before, one of the challenges is you don't have any visual dashboard, which you may be already used to. If you're working with Kubernetes, maybe you already work with AWS or Azure, and you may not be that comfortable with `kubectl`. So `kubectl`, learning how to use the tool, knowing what's important about it is the most important skill for the practical side of certifications. I think that is something people struggle with.

I know in the new versions of these certifications, you get a lot of support. You don't have to memorize every command anymore, they actually provide you with auto-completion up front, and you have an alias for kube-control that's already set up for you. You don't have to struggle with that stuff. But generally speaking, being thrown into certain scenarios where you have to solve a problem is extremely useful because that's something you would face in your job as well. It's not an exam where you have to answer multiple-choice questions and memorize things. You actually have to work through a problem on a real Kubernetes cluster. I think from that perspective, I think it's awesome.

If you're a beginner to Kubernetes in general, the KCNA would be a good entry point. I can definitely talk about that a bit more. Even though it may just be on a level of multiple choice, you can see it as an entry point for you as a learner. What topics are even important in the cloud environment if you want to transition over? Just to give you a certain perspective. You don't have to understand everything in detail, but if you want to break into that field, where do you want to go? And you might actually find that you like one of those areas specifically, and you may want to specialize, go toward that goal as well.

But generally speaking, learning how to solve problems hands-on, when it comes down to the practical certifications, is the biggest thing people struggle with. And I

think they put you under time pressure for a reason. Even though you might think two hours is a long time for a certification for solving *X* amount of problems, time goes by fairly quickly. So you're pretty much under pressure and you have to know what you're doing up front. And I think that could be another challenging aspect to the certification. That doesn't mean that at work, you have to work the same way. Obviously, they try to make it harder than it potentially needs to be. But at least that puts you into a mindset of how do I solve that problem as quickly, as efficiently as I can.

Adrián: Totally. I wanted to chime in on the commands you mentioned because it's the same on the KCNA exam. Obviously, we are not using the visual interface. Just answering a question with multiple options. But there is a lot of focus on the exam for the same reason, because it's a core capability that we would use for any other certification later. Let's talk more about your O'Reilly books. I think that there was something new this year, but you have been working with O'Reilly for so long.

Benjamin: Yeah, I've been working with O'Reilly for probably four or five years now. It's one of the big topics for O'Reilly as well. I would say, generally speaking, there are a lot of learners looking for that content. And given that at that point of time, I got more into Kubernetes myself. I proposed the idea, why don't we run some training sessions on Kubernetes? And I know there were already a couple of training sessions out there on Kubernetes. They ran well, so they knew the content is something people are looking for.

But certifications were somewhat still new. I think they had been around for maybe two years before I started. And at that time, there was relatively little content on the Kubernetes certifications. So we did start with the live training on Kubernetes, specifically for the application developer certification. So we started with that. And then my editor asked me whether I wanted to write a book about it too. And it just so happened that I wrote the book about it then. I think if you compare the two mediums, it's interesting how they can complement each other. The live training helps a lot of people, if they work through a problem, hands-on live, to ask a lot of questions. But then they still need to go back and get more context about a certain topic, because live trainings are constrained to a certain timeframe. And at some point, it's simply over.

But then you let the information settle itself a little bit. And you notice you still have more questions, you need more context around certain topics. And I think at that point of time, at least for some learners who like to read up on things, those books could be super helpful. So they provide additional context. They come with additional questions and exercises. And they try to answer the why as well. Why do I even care about this specific Kubernetes primitive? Why is it important? How does it play together with other Kubernetes primitives? So at least from that perspective, it gives you more context, more opportunity to learn. And for some learners, it's easier

to read up on something than to sit in a training. And just now we're planning to also launch a recorded training. So a video course on the CKAD. So that's probably going to come up in like two months or so, at least I hope. The recording is done. We'll see how long it takes to process all the videos. I think we have something for anyone, depending on what kind of learner you are. Obviously, there are many other learning opportunities out there. So O'Reilly is not the only publisher that got into Kubernetes. There are plenty more that even started before O'Reilly that provide help with these certifications. So I think at least from a learning perspective, see what's out there. See what fits your needs the best. Whether that's a book, whether it's a video, live training, or anything practical. And then just pick and choose depending on what provider you like the best.

Adrián: Yeah. I think that's the best approach. And what I find very interesting is the way the CNCF structures different Kubernetes certifications is role-based. I personally love role-based certifications because they treat the same topics from different perspectives. You have covered the CKA for administration, the CKAD for application development, and the security one now.

Benjamin: Yeah. That's the latest launch we did with O'Reilly. So that one was an interesting one because the security aspects were less known to me before. And it's good to get some extra exposure in that field. As you may know, the whole industry is moving toward wanting to put in more security. The more security incidents we have as an industry, the more important that aspect becomes.

Depending on the organization that you work with, there's still a separation between administration, application development, and then security aspects. But I think they're moving somewhat all together with DevSecOps. So I think in that context, they blend together. It's important to have some exposure to each of those aspects, whether you work on this on a day-to-day basis or not. It's good to have an understanding of what potentially may happen in production.

Especially when it comes down to security, it can become very costly for a company not to consider those aspects up front. If you can educate yourself or your team members or anyone that you're working with on those aspects as well, I think it will save you a lot of money as a company. Whether that's going to be your main focus or not is a different question. But it's important to know about security. And the cool thing about the certification on security is that CNCF tries to bring in open source projects as well. So it's not constrained to just plain Kubernetes. It's also about open source projects in that space.

I think that could even make the scope much bigger if they wanted to. But at some point, you just have to make a cut and say, like, we can't cover more than two hours. The exam will probably evolve in terms of the content over the years. So as tools change, as practices change, I think they'll probably try to bring it in. And usually we see an update to those certifications, say, every three, four years. That might be the

case for the security certification as well. So I think from my perspective, I wanted to get more into security myself. And I think that was a great opportunity to get some exposure specifically in the Kubernetes space and ecosystem.

Adrián: Totally, totally. With the KCNA exam, we see that pattern because it is about Kubernetes, obviously, but also about the cloud native ecosystem and the rest of the projects. Some of the questions on the exam are related to other projects that are not Kubernetes. I think that that's interesting because, as you mentioned, it's not only about Kubernetes. It's highly related to Kubernetes, of course, but it's important for learners out there to know the whole ecosystem.

You mentioned the fact that the people don't need to be total experts. And that reminds me of the exam personas for the KCNA. We're talking about maybe like fresh graduates or people who are working on Kubernetes-enabled environments, but they don't need to be the experts or the people who will be the experts or will become the experts. Is it important that all these people get at the very least a basic level of knowledge on these technologies to improve security and other topics that you have mentioned before?

Benjamin: Yeah, absolutely. I think that is definitely the case, especially if you look at some position descriptions nowadays. They usually blend multiple technologies or responsibilities together. I'm thinking about the full-stack application developer. It's not just about you being the frontend or backend developer anymore. You have to know the full stack. You have to interact with the database. And then when it comes down to more cross-cutting functionality or features or technologies like security, I think it becomes more and more important. If you look at job descriptions nowadays, I would say security is oftentimes mentioned also just for application developers or administrators, obviously for good reasons. So I think from that perspective, it is important to get at least some high-level exposure, have some understanding. This doesn't mean you have to be an expert, because oftentimes organizations have specialized roles for security. But you want to speak the same language with different team members or across different departments within an organization. So I think it's good to have some additional exposure, but that doesn't necessarily mean you have to be an expert.

Adrián: Totally. Now that's amazing. You have shared a few recommendations already of resources and how to upscale or where to focus, but do you have any other general recommendations for the exam candidates and the learners out there, something that they should do in order to, I wouldn't say only pass the exam, but also to be successful with their Kubernetes-related careers?

Benjamin: Yeah, I would say, generally speaking, if you look at, for example, KubeCon and the wider Kubernetes community, I think there's a lot of opportunity to learn and to get involved. If you compare this to some other communities, I feel the Kubernetes community is extremely inclusive and welcoming to new members. I

think one way to start, if you want to get deeper into the community, is to hop on to the Slack channel. I think there's a lot of discussion going on there. And maybe based on those discussions, you may recognize certain areas, tools from the ecosystem that you may want to get involved into more deeply.

So apart from your main job, where you may focus on a specific role, let's say you're an administrator of a Kubernetes cluster, you may still be interested in something else, like helping people picking up Kubernetes easier. There are some learning opportunities and also roles within the CNCF organization that you can get involved with. I think the scope is really broad. There's a book club now and there's a new documentation approach now for beginners of Kubernetes. If you want to improve documentation, that's a good way to involve yourself more deeply. And with that, learn about the contribution process for the CNCF as well.

And then on the sidelines, you may recognize that you're interested in learning more about a specific tool in the ecosystem. Of those tools, there are almost hundreds by now. This is a good chance to make good connections and improve your knowledge about a specific tool or part of the Kubernetes ecosystem. And obviously, that comes with a big time involvement, I understand. Not everybody has the time, apart from family, hobbies, and a regular job to contribute. But if you want to, you can. So that is a great opportunity to learn more.

If you want to start your own blog, that is oftentimes a good way to start, not necessarily on the level of an expert about *XYZ*. You can just talk about your personal experience. That is always a perspective that people value a lot, independent of kind of, hey, I'm trying to teach you something. I'm just going to talk about what I'm struggling with and how I solve that specific problem. I guess, if you want to even take that to KubeCon or some other conference, that would be a great chance to do it. Because that's what attendees are usually looking for. It can just give a personal perspective on those sides of things. And maybe you evolve or develop a specific best practice even better. But I think from that perspective, open source contributions, blogging about things, getting exposure to those tools yourself, obviously, and maybe trying out new tools, comparing them.

For example, if you look at the different tools in the security space for Kubernetes, there are a lot of them by now. You could compare them, try them out, see what kind of features they bring, which one do you like the best, and then maybe you can make it a blog post and go from there. Generally speaking, there are so many learning opportunities, specifically because the community is so welcoming. Just go to KubeCon or one of the conferences or a local meetup and see what other people are talking about that may spark new ideas and show you the direction you may want to go.

Adrián: Yeah, totally. I love the recommendation because that's what we see in all the open source communities. Of course, this level of involvement requires time, and everyone is aware of the difficulty of Kubernetes topics. So, we don't need to be hyper experts. We can adapt the content to our level of knowledge, and people are looking for that, like applied experiences. So, that's a great recommendation for learners and for people who are trying to get into the KCNA exam, because that's their level usually.

I really appreciate the time you have spent here with us. I'll be paying attention to the new online training that you're releasing and will certainly read your latest book. I think that you are helping a lot of the community with your content and the way you explain things. Thank you very much.

Benjamin: Yeah, thank you. It was a pleasure.

Summary

We have now concluded your full immersion into cloud native and some of the key terms that you will leverage for the KCNA exam and in Chapters 5 and 6. Chapter 4 is especially important if you were missing some general cloud computing knowledge, and if you haven't been exposed to DevOps and other cloud native topics. The next chapter, along with Chapter 6, will equip you with the Kubernetes knowledge you will need to pass the exam. These chapters can be especially challenging if you haven't worked with Kubernetes before.

CHAPTER 5

The Key Role of Kubernetes

In this chapter, we will explore software architecture paradigm changes and the role that Kubernetes plays in streamlining both the development and operation of applications in the cloud native realm. We will also look at the industry adoption of Kubernetes from its inception to the present.

Understanding the ecosystem around container technology and in which scenarios Kubernetes excels over other technologies is worth the journey. Passion and excitement are behind the rise of container orchestration technologies; the intersection of industry momentum, the relentless open source community, and the industry shifting toward a more standardized and interoperable ecosystem has brought us to where we stand today.

As you can observe in Figure 5-1, the landscape of projects in orchestration and management is huge, with Kubernetes the project of reference as of today. This chapter will help you establish a foundational knowledge and solid roots on the basics of Kubernetes. Let's uncover the key role of Kubernetes through the following sections!

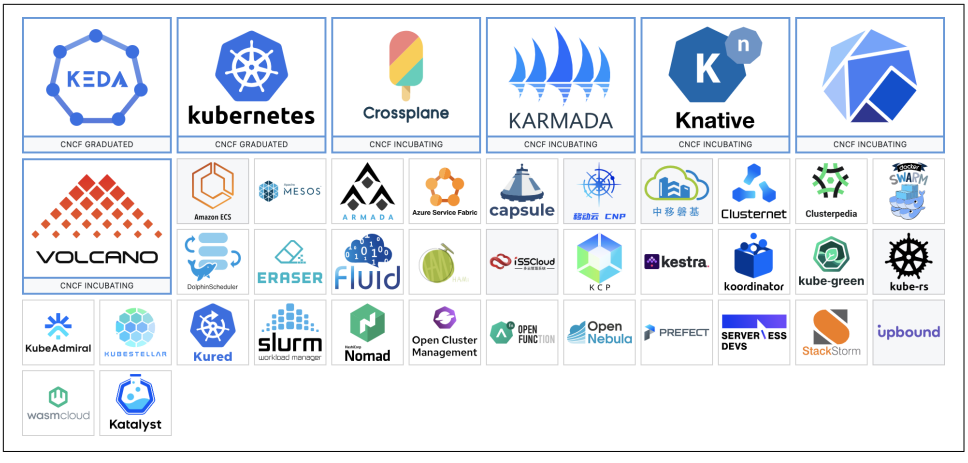


Figure 5-1. CNCF orchestration and management technology landscape

An Introduction to Containers and Orchestration Technologies

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

A term that resonates in many forums and discussions is "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 "cloud native" ecosystem from architecture and business angles can be overwhelming. Additionally, both angles are related to 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 through. Both sides need to be aligned because one cannot survive without the other.

No solution has only benefits without drawbacks, and making 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 is a reference guide for understanding the overall spectrum and getting good general knowledge about the community behind the main projects and the industry around container technology. Let's delve into container technology through the milestones that shaped the industry as of today.

Containers

There are many different ways to explain containers. It's more important, however, to understand the problem they try 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 by deploying fixes, updates, and new features in production (see Figure 5-2). It is as simple and as complex as that.

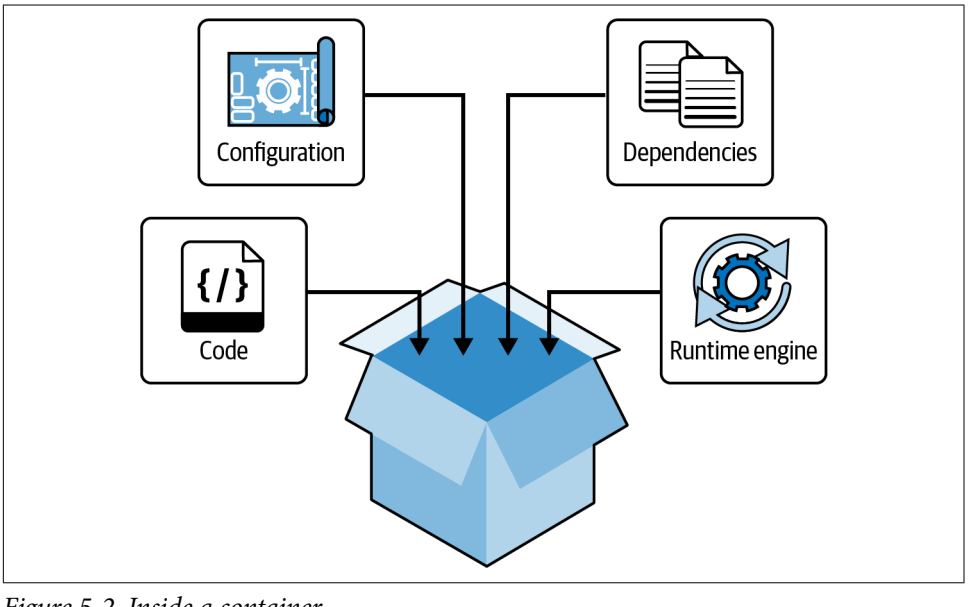


Figure 5-2. Inside a container

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 has potential impacts beyond the development and operational aspects. There are other impacts, like an organization's cultural aspects, which need to be taken into account if you intend to adopt a container-first strategy. 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 (Jorge) hope helps you 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 at the world's leading provider for networking and mobile network equipment. It was 2013, and at that time almost every telco operator used our technology to provide 2G, 3G, and 4G mobile networks. 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 application (in charge of network operation, including monitoring and fault management among other responsibilities).

This application was used in almost every telco operator in the world for operating their mobile network. As you can imagine (remember, it is 2013), it was a three-tier monolithic application with the middleware tier containing different functions but tightly coupled between all tiers (see Figure 5-3).

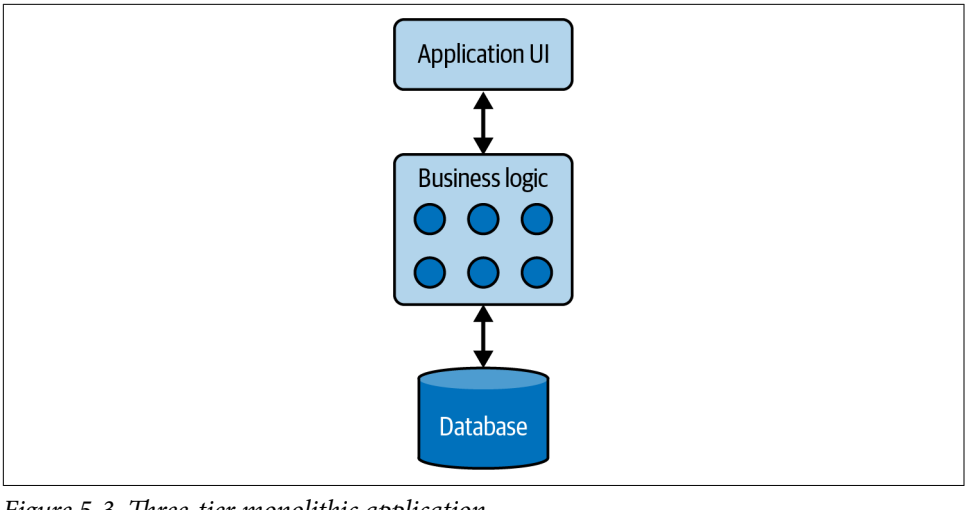


Figure 5-3. Three-tier monolithic application

Managing the radio network is a mission-critical task for any operator, so the application had to be reliable and deployed in a high-availability configuration. However, as radio access network equipment evolved to offer new cells to address network capacity and coverage issues (picocells, microcells, etc.) the number of nodes connected to the application grew exponentially. And because the components were highly coupled, 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 was run).

But as you can anticipate, this is a valid approach only until reaching the limits of the hardware or the underlying software itself (not to mention the cost in terms of downtime required as well as software fine-tuning and the 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 great success that user adoption grows exponentially and therefore business is expected to grow.)

However, a scaling issue for a mission-critical application that is responsible for keeping 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 they need to be in compliance with very strict requirements. We struggled scaling every tier, but it was especially hard to scale the persistence layer (the database engine) because the underlying technology had its own vertical scaling limitations.

And if you are thinking that scale was an issue only in relation to accommodating new nodes in the network, you are seeing only one aspect 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 updates to the OSS application to be able to configure and handle them properly. So if the application couldn't manage this requirement, it would put the radio network roadmap in danger (and thus endanger the signing of 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 may think that this could have been anticipated (and in fact it was anticipated a few years earlier), but the growth rate was larger than we could even imagine. Nowadays applications scale to almost unlimited levels, but in the timeframe of 2011–2013 there were no examples in the real world that solved such hyperscale issues.

We took every software piece to its practical limit, and even that wouldn't meet our requirements. This was the trigger to shift to a microservices architecture that could potentially scale limitlessly and accommodate any configuration and size for the radio network.

The preceding story is slightly misleading; the truth is that we had anticipated that our application would reach its limits and started working in parallel on a new generation OSS application. But it does provide a real-life example of issues around scaling. The reality is that it was not until 2014 that we were able to start testing the new app in the real world, and we had to struggle with the issues I mentioned earlier in the former application.

Scaling is a major barrier for growth and expansion even if your software architecture is highly decoupled. Scale needs to be managed. The scale an application can grow to today in an interconnected world is much larger, and the pattern and paradigm have completely changed. 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 antipatterns). It's an especially good fit for those whose domains of functionality are split into 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, and different user views).

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 try to imagine the number of microservices belonging to the 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… voilà…a perfect operational storm again for developer and operations teams!

Figure 5-4 helps us 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, taking into account the size of the container, the content inside, etc. Otherwise, it can make your server requirements unbalanced and it will struggle to serve the containers' needs, degrading 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.



Figure 5-4. An image used in my first container presentations at Red Hat explaining why container management is required to avoid a disaster

Container Orchestration

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

Multiple solutions today aim to solve the challenges of containerization management and operation at scale. These solutions differ in terms of architecture, deployment, management, and underlying technologies.

We will dig deeper in the following sections to understand the ecosystem and the solutions and implementations available. The minimum capabilities that a container orchestrator should provide are as follows:

- Provisioning and deployment of containers
- Scaling based on workload requirements
- Load balancing that allows you to distribute incoming traffic across container instances to maintain 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 capabilities enforcing security policies defined by admins to prevent or reduce security breaches and vulnerabilities exposure

Container Ecosystem and Standards

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

Let's trace the history in this section. In 2013, 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 the *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 one another. They provide process isolation inside the operating system. There are six 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. The Linux manual page explains this namespace as follows:

> 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
: At its simplest, a Unix Timesharing System (UTS) namespace provides a way to isolate two specific system identifiers: the hostname and the Network Information Service (NIS) domain name.

User namespace
: The user namespace allows you to create isolated environments for processes. Each user namespace has its own set of user and group IDs (UIDs and GIDs) that are separate from the global system.

MNT namespace
: The MNT namespace allows you to isolate mount points such that processes in different namespaces cannot view one another's files.

Interprocess communication namespace
: This namespace is probably one of the most complex. For simplicity's sake, let's think about it as the functionality that allows isolating communication resources between processes. This ensures that processes within the same namespace can interact with shared memory, semaphores, and message queues while remaining separate from processes in a different namespace.

Cgroups

In simple terms, *cgroups* (control groups) 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 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?

Docker

In 2013, Docker, Inc., changed the software shipment paradigm and gave the user, using a tool called *Docker*, the capability to package containers into images (the concept of images is crucial) to easily move containers between machines. Docker was the first to attempt to make containers a standard software unit, as it states in its 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 development of virtualization technology and the absence of standards for the technology, it was clear for the actors participating in container development that a set of standards should be promoted. This would guarantee interoperability between platforms, avoid vendor lock-in, and contribute to a healthy ecosystem of tools. 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 (which we will talk about in the next chapter).

What Do the Container Standards Provide?

Container standards provide a common set of specifications and guidelines for container platforms, making it easier to use containers across platforms and providing compatibility as long as the standards are followed by a solution's 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, this is the very first time we'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 in 2015 by Docker, CoreOS, and other leaders in the container industry.

Many people mistakenly believe that the OCI is relevant only to Linux container technologies because of 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 OCI members includes a variety of actors in the industry, as shown in Figure 5-5.



Figure 5-5. List of OCI members

Container Solutions

Now that we have reviewed the key events in the containerization ecosystem and the main standard, let's look at the commonly used container solutions in the realm of container orchestration.

Docker Swarm

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

Starting with 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 no longer required to install the Docker Swarm software as a standalone deployment. 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 then archived on February 1, 2021 (see the swarmkit GitHub repository).

Nomad

Nomad is a container orchestrator and workload scheduler developed by HashiCorp; the first version was released in 2015. You can think of Nomad as a lightweight orchestrator. It is architecturally simpler than other solutions in part because it is focused on cluster management and scheduling and designed with the Unix philosophy of having a small scope. It relies on external tools like Consul for service discovery/service mesh and Vault for secret management.

Kubernetes

Though we are going to dig deeper into what Kubernetes is in the next section and 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*, 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.

Other tools

The current ecosystem around container technology is so huge that there are also tools that you can consider meta orchestrators, as they allow the management of multiple container orchestration solutions from a single and centralized tool. This is the case with 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* onward, Rancher has been focused on being a management platform for deploying and running Kubernetes clusters anywhere and on any provider.

What Is Kubernetes, and Why Is It So Important?

Let's explore Kubernetes by going back to its roots in 2003/2004 and exploring the stages that made it the project it is today. In 2003/2004, Google introduced the Borg system. *Borg* is essentially a cluster manager that orchestrates the ballet of thousands of applications across a sprawling web of machines at Google. You 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. Figure 5-6 shows Borg's architecture, which was included in a paper written by Abhishek Verma et al.

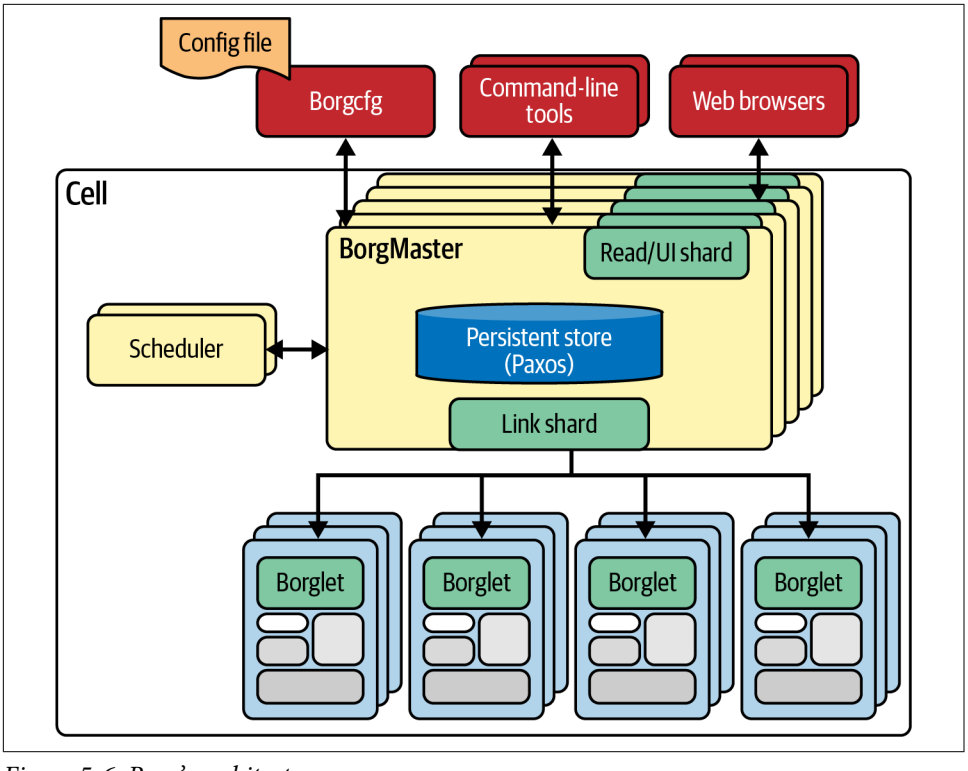


Figure 5-6. Borg's architecture

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's design and architecture has its roots in 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 in July 2015 Kubernetes version 1.0 was released as Google considered it production ready. Together with that release, Google also donated the Kubernetes project to a newly formed foundation—the Cloud Native Computing Foundation (CNCF), which is run by the Linux Foundation—making Kubernetes the very first project under the governance of CNCF.

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, ran 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 industry support and contribution to the Kubernetes project supports the market message around Kubernetes being the de facto container orchestration standard. Many factors 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's success is that it is an open source platform maintained by CNCF.

As presented in previous chapters of this book, CNCF is a vendor-neutral organization that promotes the adoption of cloud native technologies, and with that mission 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 create a thriving ecosystem around Kubernetes, with a large and active community of developers and contributors leading to innovation and adoption across the industry.

But this adoption wave has brought its own challenges, especially ones related to scaling, operating, and managing the applications and the underlying technology supporting them. These challenges have positively contributed to driving the adoption of DevOps 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, 36 projects 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 have a high-level view to understand the momentum that the open source community is experiencing in the Kubernetes ecosystem, specifically under CNCF (see Figure 5-7).



Figure 5-7. A portion of the cloud native landscape (source: CNCF)

Under this umbrella of constant evolution in the industry, it is risky to make predictions about the evolution of the ecosystem from containers and container orchestrators, 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

The year 2023 can be marked as 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 large language models (LLMs), presented new ways of interacting with different kinds of data. These systems also provided capabilities to what's called *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 continue to be key in the coming years.

As the use cases of generative AI increase and AI is infused into new or existing applications, it will be the norm to have a modern infrastructure that supports software architectures capable of behaving elastically to adapt to varying usage patterns. Being highly scalable will be the norm. 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 users to emulate some human cognitive functions through pre-trained models are a good match to be deployed in containers. Applications ranging from computer vision models to services capable of translating speech and text are ready to be deployed on containers, allowing for quicker and easier access to the technology.

There is also a third stream of evolution underway related to 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 expertise and tooling needed to operate Kubernetes the right way.

This creates a barrier for enterprises that want to adopt Kubernetes or scale its use where it has already been adopted. In addition to the learning curve, there is also a shortage of talent with solid skills to successfully tackle these projects. The open source community is working hard to simplify the use of Kubernetes with the help of AI. Projects such as K8sGPT are a good example of the efforts to simplify Kubernetes operations, specifically for site reliability engineers (SREs).

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 take a look at the reasons 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 creating 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 because of 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 is used as inputs for classification, clustering, regression algorithms, and more, facilitating the application of predictive models and advanced analytics.

Because of 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 new scenarios where it may be necessary to run a vector database on a Kubernetes cluster, and some open source projects already cover this possibility. An example from the Linux Foundation's data projects pillar is Milvus. Another interesting project that also runs on Kubernetes is Weaviate. We will continue to see advances in this field and new innovative projects emerging from the open source community.

Edge Computing

Some AI-driven apps are built to cover scenarios where near-real-time responses are required but requests are made in environments or devices with limited access to reliable telecommunications networks. Also on the rise are use cases where the execution of analytical models or even generative AI models takes place in embedded devices (in vehicles, home appliances, etc.). This is where the concept of edge computing takes on more relevance. 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.

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 utilize the IaC concept. This means that infrastructure resources, such as servers, networks, and databases, are defined and managed by code, which facilitates automation, reproducibility, lifecycle 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 platform engineering ecosystem, taking advantage of the following characteristics:

Automated deployments
: Kubernetes allows you 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 reroute traffic to a healthy service and ensure that the application continues to run.

Application updates without downtime
: Organizations can deploy upgrades and security fixes without downtime, thanks to Kubernetes's ability to manage multiple versions of an application at the same time.

A Practical Example: The Platform Engineering Team

To illustrate how platform engineering works with Kubernetes, let's consider a team of platform engineers at a fictitious ecommerce 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 so the rest of the pieces are correctly configured and secured. They use only the catalog of services available in the platform to build 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 handles redirecting traffic to healthy servers. The team also constantly monitors the performance and health of the infrastructure to ensure uninterrupted service.

Orchestration Alternatives to Kubernetes

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 determine the choice.

Knowing the alternatives allows you to make an informed decision about the container orchestration and distributed application platform that best suits your boundary conditions. Let's take a look at the main alternatives to Kubernetes.

Apache Mesos and Marathon

It could be argued that Apache Mesos doesn't fit in the container orchestration area. Comparing it with Kubernetes side by side wouldn't be fair as it was not originally built to cover the same functionality, but with Marathon as a container orchestrator, you can use the pair in the same scenarios. 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.

Without digging into the architecture aspects, the main differences between Mesos with Marathon and Kubernetes are as follows:

Focused design on resource management
: 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 such scenarios, and it can compete with Mesos and Marathon in this regard.

Red Hat OpenShift

Red Hat OpenShift (its upstream project is called *OKD*) is an orchestration platform that uses Kubernetes as its core technology. It is therefore an alternative platform but not an alternative technology since OpenShift's engine is Kubernetes. Red Hat OpenShift differs from Kubernetes in its approach, as it is designed as a platform to provide a ready-to-use solution for enterprise environments. This includes all the necessary ecosystem pieces required for 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.

The main differences between OpenShift and Kubernetes include the following:

Advanced security
: OpenShift is intended for use in highly regulated environments and therefore offers advanced security capabilities such as container isolation and centralized identity and access management. Kubernetes is also utilized in highly regulated environments, but the hardening needs to be done ad hoc.

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 vanilla Kubernetes clusters (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.

Containerization Benefits 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 to instill a cultural mindset that enables organizations to thrive in an increasingly competitive and digital world.

The benefits for an organization embracing container technology are not always easy to measure. In addition to the operational benefits (greater agility, scalability, efficiency, and security), let's take a closer look at some of the other benefits here:

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 to 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.

Expert Insights: Brendan Burns

Jorge: We have the pleasure to introduce you to Brendan Burns. Thank you, Brendan, for being here with us.

Brendan: Thanks for having me.

Jorge: Brendan, you are probably one of the most famous faces in the Kubernetes and cloud native area. But for those new to KCNA, would you please share about who you are and what your relationship is with Kubernetes and also with the cloud native movement?

Brendan: Sure, sure. I think that the most significant thing is that I was one of the three folks with Joe and Craig who started the Kubernetes open source project nearly a decade ago at this point, actually. So it's getting a little old, I guess. But I'd been doing cloud native development before that for a while, building large-scale distributed systems, and it sort of influenced a lot of the development. And since then, I've been working at Azure, delivering cloud native computing to everyone who wants to be a part of the Microsoft Cloud, and honestly, a lot of open source projects that are available to anyone who's running Kubernetes or cloud native. And so now I'm the corporate vice president for cloud native open source on Microsoft Azure, and do a lot of work with our customers, helping them adopt cloud native technologies and open source technologies.

Jorge: Wow, that's impressive, Brendan, and we are really lucky to have the opportunity to interview you. You are one of the most supportive people in the open source community. I'm really curious about your professional journey. How was your journey from Google to Microsoft?

Brendan: I started my career in the early .com, the first .com in the late 90s, and that was sort of when I first cut my teeth developing software that other people use. I majored in computer science and learned a lot, but it was a little bit boring. I went back to graduate school and got a PhD in robotics, actually, and spent a long time doing robotics. And then I was a professor, actually, for a couple of years. And I think

that was really instructive. You don't necessarily think of being a professor helping set you up for building a successful open source project, but it taught me a lot about how you create stuff that people can learn. Now people are motivated to learn Kubernetes, right? But when Kubernetes was created, I mean, nobody was motivated to learn it because nobody used it. It wasn't a thing.

In order to transform it from being an idea and a project to something that people use, they have to learn how to use it. And I think a lot of that experience teaching CS101 and other classes helped us, helped me structure things so that in the early days, people could get a sense for what it is and what they're doing. I've done a lot of videos since, and teaching was really a valuable part of that. Then we spent a long time working on that open source project and getting it out the door.

I live in the Seattle area. I've been in this area for a long, long time. And I was really interested in how people develop cloud native applications, not just operate them. I think Kubernetes has a lot of stuff about operating applications, but less about developing applications. Microsoft has this incredible history. It's unmatched really in the industry in terms of developer and developer productivity, whether it's VS Code or TypeScript or, you know, .NET or just about anything and then obviously it has a really successful public cloud. And that combination of things together, as well as Satya's cultural changes that he was making really made this place a really attractive place to land and to step in and have an opportunity in 2016 to really help the transition. The transition from Microsoft being a Windows-oriented company to a Linux-oriented company was already in the works, but it was definitely accelerating across that timeframe, and I had the opportunity to help Linux on Azure go from 20% to 30% to 40% to greater than 50%. So that whole journey has been really amazing and really fun to have a part of.

Jorge: Yeah, definitely, Brendan. I think having you at Microsoft was one of the things that made a lot of people from the open source community move to Microsoft. So yeah, you really did a great job there. It's amazing to see where you are taking the open source community in Microsoft. So, you worked on Borg at Google; how was the path from Borg to Kubernetes and also to create a cloud native computing foundation?

Brendan: I was a Borg user, not a Borg developer. I worked on web search, and Borg was built to support web search and ads. I always like to say that I had a better sense for it than the developers did because it's one thing to build something; it's another thing to use it and have it be the thing you're relying on. You learn its quirks very quickly when it pages you in the middle of the night or whatever else. I don't know if everybody necessarily feels this way, but I've always wanted to understand how the tools that I use work. And I've always thought, well, what would I do differently? What do I like about this? What do I not like about this? How could it be better? Some people really get aligned with a particular programming language. I think I've

always been the person who's like, I can tell you the long list of things that I don't like about every programming language. I've never had one where I'm like, that's my programming language. I've always been like, here's all the places you could improve. Here's all the mistakes that they made.

Similarly, I think with Borg, there was a lot of learning on the fly and opportunities for improvement. I think a lot of that influenced the development of Kubernetes. I was in Google Cloud at the time. We were thinking a lot about APIs and RESTful APIs and stuff like that. And Joe Beda, who is one of the co-founders, was just really, really good at APIs. I'm terrible at APIs, by the way. I can implement them, but I'm a bad designer. I'm not a detail person. And you need to be with a craftsperson to do that sort of stuff. Joe is great at that. And that's why it's really valuable. I think it's really, really valuable to have a group of people when you build a project like this, because they can complement your weaknesses, you can complement their strengths. We were lucky to have folks like Tim Hawken and others who had actually been on the board and Don Chen, who had a lot of experience there.

For the CNCF, one of the things that was very, very obvious to us from the beginning was that vendor neutrality was hugely important. I don't think anybody thought that the CNCF was going to be what the CNCF is today. We were really motivated around this idea that the only way to success for Kubernetes was for it to be present everywhere. You can't have this kind of system that everybody's relying on and have it only be available in certain places, right? It has to be something like the Linux kernel; it's accessible everywhere. Mostly we were focused on how we ensure that this open source project is successful.

And to be honest, Craig did all of the heavy lifting there. I think we knew that CNCF was valuable and important, but Craig was a product manager and the product leader for Kubernetes. I think he knew that this was critical and just drove toward it while we were busy managing the community and writing a lot of code. He was busy figuring out how to go to foundation and, you know, got that done. And we had the big announcement at, I think it was OSCON in 2015 down in Portland with O'Reilly. Then it just sort of ballooned from there. It's been interesting to watch it go from like 10 projects to 100 projects. I don't even know how many, I think we're getting to the point where we can't fit them on a screen anymore. So that's cool. It's fun to see it become the preeminent place that people come to hang out too. I think the community has always been a really important piece of what we do. And it's just fun to come together, you know?

Jorge: Yeah, absolutely. Brendan, you are seeing a lot of customers every day and you are in the most critical projects. What are the current trends that you are seeing in the cloud native area also related to Kubernetes? What's the current adoption piece? What are you seeing from your position?

Brendan: We're well past all the early adoption, we're well past all of the medium adoptions, and we're into the people who are like, oh, I guess we should do this because it's really gonna be a thing. And I think the consequence is that the percentage of people who actually care about Kubernetes is going down. And that's good, right? It's a necessary thing. By the time you're in the breadth market, people are there because they have a job to do, right? They have an application that they want to build, they have a service they want to run, they don't really care about how, it's a means to an end. It needs to do what it's supposed to do, but they're not going to be excited about upgrading, for example. They're also gonna be in regulated industries and otherwise where upgrading is very challenging.

That's one of the reasons why you see us pushing really hard on LTS and long-term support for Kubernetes. It's been out there in the community for a while, but it's never really turned into a thing. Last year at KubeCon EU, we announced that Azure was gonna do this long-term support thing. We basically just signed on the dotted line and said, hey, look, we're signing up for this. But what's been great is that it's crystallized a lot of energy in the community also. And we're seeing LTS show up, not just for Azure customers, but for all Kubernetes customers. That's an important step forward. I think that means that, in some sense, people can set it up and forget about it for a little bit. That leads to the other thing that we're seeing, and it's been there for a while but I think it's starting to get formalized, which is the growth of platform engineering as a thing.

For a long time, we focused on container orchestration. And now I think people are starting to say, OK, the next layer up is this platform engineering layer where maybe it's GitHub Actions, maybe it's a bunch of CRDs. I think we haven't quite crystallized it. I think we need to get to the LAMP stack. It's not literally gonna be LAMP, but you need that acronym where people are like, you need the GitOps thing and you need the actions there. I don't know what it's gonna be, but there's gonna be a set of components and people are gonna be like, oh, that's the standard platform on top of Kubernetes. And I think it's starting to crystallize a little bit. I think it's gonna be the most interesting thing for the next four or five years. And hopefully there's some stabilization in that space. And there's a bunch of different stuff that has to happen.

I think that for a while, people were trying to compete in that space. And different companies would be like, oh, we're gonna compete with our service mesh and we're not gonna donate our service mesh to the CNCF because we want it to be ours. And we're starting to see that fade a little bit. And that's good because it needs to not be competitive because it needs to be one platform. You can't have 20 different platforms. We need one. And so I think we're getting to a level of maturity there where there will be, roughly speaking, one platform that shows up on top where developers can say, hey, push a JAR file, push a TypeScript file, push a whatever it happens to be, and it deploys. And monitoring happens and the database is there and the credentials are handled securely and rotated and all this stuff. Because right

now, I think what we're seeing is it's really, really complicated. And even people who know what they're doing are spending way too much time on the setting up stuff as opposed to the development stuff. So that's the trend.

The last thing that's interesting is a little bit more future looking. We've done all this work on cloud native computing and these cloud native APIs. But if you look at what's underneath the Linux container, if you look at what your code is running against, it's a bunch of kernel APIs from the 1970s, right? And I think we need to do more work. WebAssembly is an interesting area where we're looking to do some work to say, actually, maybe your application shouldn't care about files. Maybe your application shouldn't care about sockets, right? And it shouldn't even be in the system. And instead, you get something that looks a little bit more like what you'd expect in a functions-as-a-service kind of environment, but without the runtime, where it's very clear, it's still your main, you're still in control, it's not request-driven. But the things you can call are things like set a key-value store or receive an event.

I think projects like the Dapr project that we've worked on and seen a lot of adoption in the CNCF is an example of how you deliver that kind of higher-level runtime into the container. So we're excited about where that's going to go. That's definitely a little bit more future oriented. It needs some exploration, but I'm hopeful that that's going to make a difference too.

Jorge: It sounds like a great opportunity for the people learning about the cloud native ecosystem right now. Probably it's a field where there'll be a lot of work.

Brendan: Yeah, there's a lot of opportunities out there, I think, for sure.

Jorge: OK, sounds great. To finish our interview, Brendan, do you have any recommendations for learners about cloud native? Are there any projects that they should have a look at?

Brendan: If you go to *dapr.io*, that's a pretty cool project. Helps make a lot of this stuff easier, helps eliminate a bunch of complexity around talking to key-value stores or receiving events from various different event sources and stuff like that. I definitely encourage people to also take a look at other open source projects they might want to get involved in. I think the community tends to be pretty welcoming. Maybe don't jump straight into the main Kubernetes repo, but there's a bunch of other repositories around that tend to have help wanted and that sort of thing.

I'd also say there's a lot of value as you're learning in actually building real stuff. Sometimes when you're learning, you do really toy stuff and it's small and you're like, oh, hello world. But I also have found personally that I like finding some hobby project to work on. Like set up something with Kubernetes for Halloween or whatever it is. Making it a little bit more real means you'll really learn the skills because it kind of has to work. And then you'll be motivated too because you can only get so excited about hello world, I guess. That little step above hello world tends to

make it a little stickier because otherwise you skim the surface and you think that you got it, but then you have to do it for real. And you're like, oh wait, I didn't actually do it, you know? The only other thing I would say is that the online community for Kubernetes is great. So don't be afraid to ask questions.

Jorge: Yeah, that's super. Thank you for being here and for sharing your thoughts because it's really a pleasure to learn from someone like you. And likely the people learning today will be our colleagues in the future. So that will be super, super great, I think.

Brendan: For sure. My team's always growing. So definitely feel free to ping us. We've got some great opportunities.

Jorge: Thank you so much, Brendan. Thank you for your time.

Brendan: Yeah, absolutely. Take care.

Summary

In this chapter, we explored how Kubernetes empowers businesses as a cornerstone of cloud native ecosystems and its pivotal role in containerization technologies. From agility and innovation to collaboration and resiliency, Kubernetes offers a robust platform for modern application deployment. The next chapter will continue your preparation for the KCNA exam with practical code examples to equip you with the knowledge to feel comfortable in your first steps with this technology.

CHAPTER 6
Technical Kubernetes Topics

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 the 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.

Kubernetes Releases

The pace of innovation in the open source community is fast, and every new Kubernetes release contains tons of new pieces of code that bring new features to the product. To keep up with the rhythm of news, we encourage you to check the release notes that are delivered with every new release.

In general, Kubernetes versions are expressed as *x.y.z*, following semantic versioning terminology:

- *x* is the major version.
- *y* is the minor version.
- *z* is the patch version.

The Kubernetes 1.19 cycle took far longer than usual. The Release Team extended it to lessen the burden on both Kubernetes contributors and end users because of the COVID-19 pandemic. Following this extended release, the Kubernetes 1.20 release became the third, and final, release for 2020. Sarting from the 1.20 release, there have been three release cycles per year (until 1.19, there was a cadence of four releases per year). End users now experience a slower release frequency and a more gradual enhancement graduation rate (see Figure 6-1).

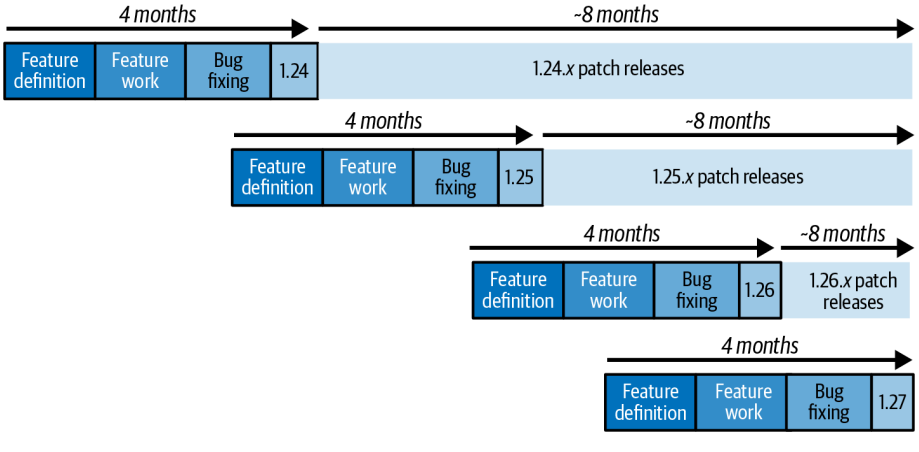


Figure 6-1. Example of release cadence of minor versions

We strongly recommend checking out the Kubernetes Patch Releases page on a monthly basis to review the delivery of new patch versions.

Kubernetes Basic Concepts: From Containers to Clusters

In the previous chapter, 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 to manage containerized applications. Its basic architecture is built from two planes: the control plane and data plane (see Figure 6-2). We will dive into these concepts later in the chapter but for now let's keep it simple with the following descriptions:

Control plane
: You can think of this as the *brain* of Kubernetes. The control plane orchestrates containers, manages clusters, and ensures that 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.

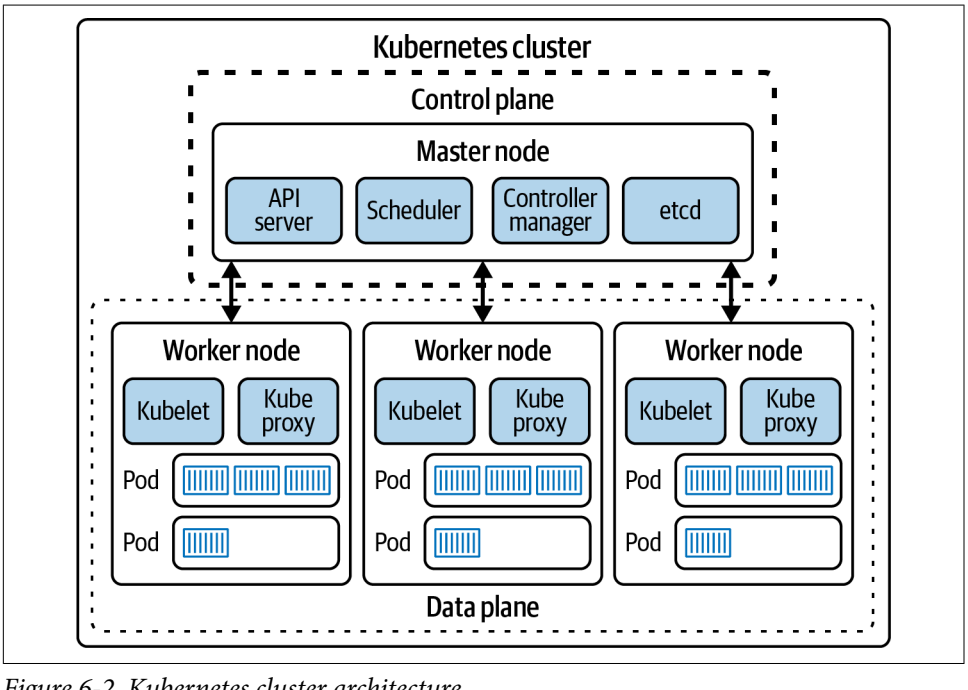


Figure 6-2. Kubernetes cluster architecture

Before going deeper into the main components of each Kubernetes plane, let's talk about 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 the dependencies and configurations necessary for it to function properly. You can think of it as a logical instantiation of a Dockerfile.

What Is a Dockerfile?

A *Dockerfile* (Figure 6-3) 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. We recommend learning the basics of creating images from Dockerfiles. The Dockerfile Reference is a good resource for learning about this topic.



Figure 6-3. Dockerfile to create an image that runs a Redis server and exposes port 6379

In the Docker ecosystem, the flow to run a container is shown in Figure 6-4.

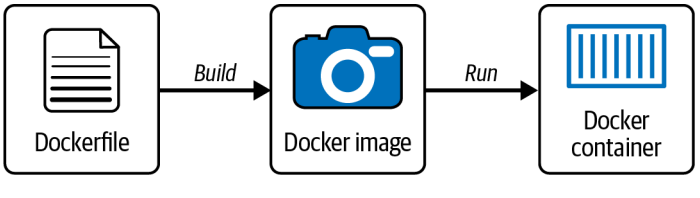


Figure 6-4. Steps for running a Docker container

A container can be a database, a web application, an API, etc. The concept of a container in Kubernetes is exactly the same as you learned in Chapter 5, 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; each node can host multiple containers, each running different applications or services (see Figure 6-5). Containers make it easy to package, deploy, and scale applications, and they are the runtime environment for workloads in a Kubernetes cluster.

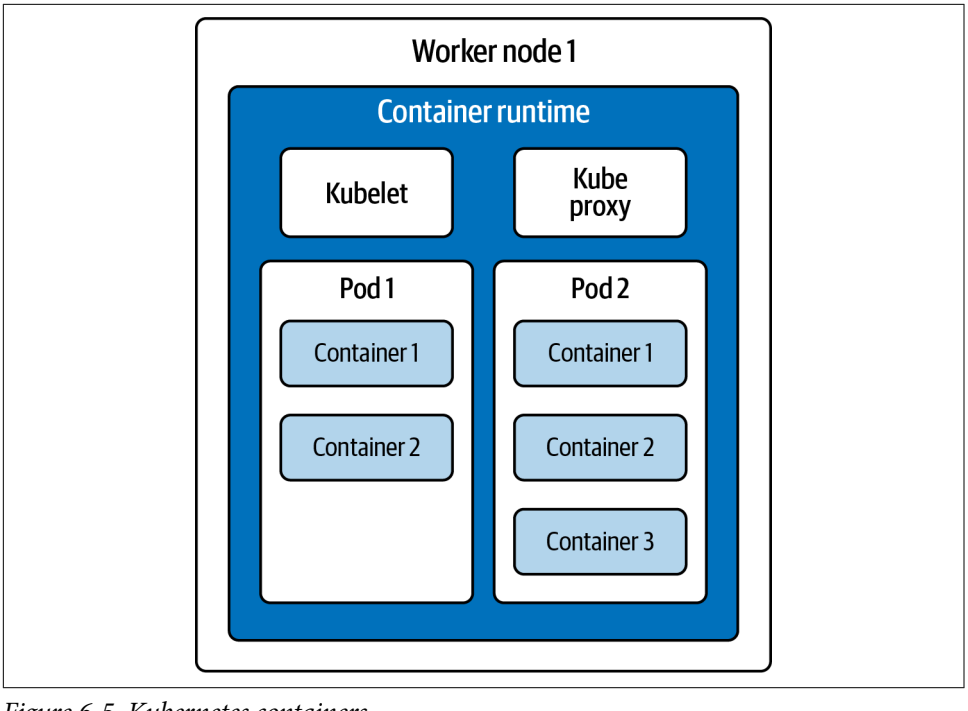


Figure 6-5. Kubernetes containers

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 a structure that abstracts one or more containers that are deployed together in the same host (see Figure 6-6). Typically, an application runs in one or more Pods (depending on its architecture).

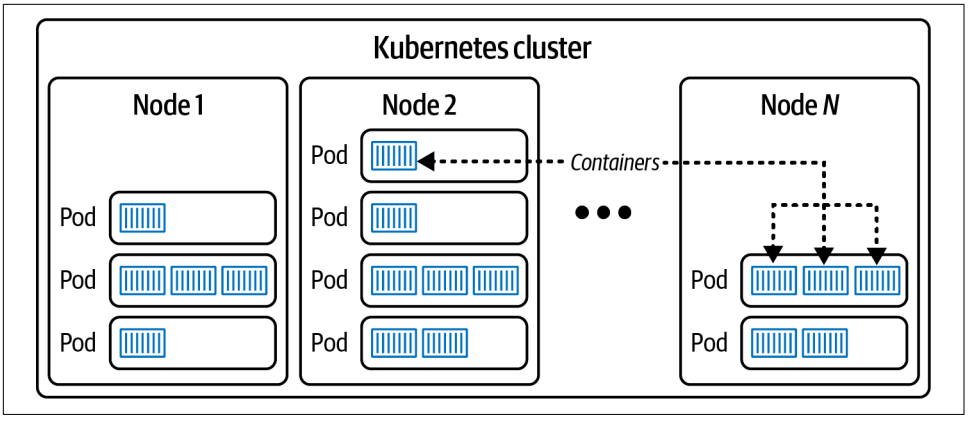


Figure 6-6. Pods running in a Kubernetes cluster

Nodes

Now that you know how containers and Pods fit into the greater Kubernetes architecture, let's go back to the Kubernetes cluster and go deeper into its components. The cluster itself provides the necessary infrastructure and resources for applications to run, including networking and storage, ensuring high availability and fault tolerance by distributing the applications across multiple *nodes*. If one node fails, an application can continue running on another.

A Kubernetes cluster has 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 (refer back to Figure 6-2).

These nodes are part of the two differentiated planes that build the cluster: the control plane and the data plane. Let's delve into the nodes 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 (with 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 is a distributed key-value store that 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. The function of an etcd node will be the same as if it runs on the master node: it is responsible for storing configuration data and the desired state of the cluster.

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 take 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 a user perspective, interaction with a Kubernetes cluster can be through the UI provided by the Kubernetes Dashboard (and/or via the command-line interface).

kubectl and kubeadm

For interacting with Kubernetes clusters, Kubernetes provides `kubectl`. This command-line tool facilitates communication with the cluster through the Kubernetes API.

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

The syntax of `kubectl` commands is `kubectl` *`command type name flags`*. The command specifies the operation (e.g., create, get, describe, delete). The type specifies the resource type (case insensitive). The 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 (see Figure 6-7).

```
~ 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 6-7. Example of a `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). It performs 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 out the official documentation.

etcd

etcd serves as a reliable, consistent, and highly available key-value store that underpins Kubernetes's 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 include PostgreSQL, mySQL, and YugabyteDB. Although switching away from etcd was not a very popular scenario for the first releases of Kubernetes, adoption of these alternatives is growing, mainly driven by increased

reliance on distributed SQL DB engines, which provide 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 when replacing etcd with any other Relational Database Management System (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.

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. In Kubernetes, a controller is a control loop that watches the state of the cluster and then makes or requests changes where needed.

The main controllers released in Kubernetes are as follows:

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

Cloud controller manager

In certain situations, it's beneficial to enable Kubernetes to interact with specific cloud providers or infrastructure platforms by using the provider's APIs. For instance, it can be optimal to use the provider's API to handle nodes, configure the load balancer, or execute 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 into an optional component decoupled it and allowed 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.

Control Loops, Namespaces, and ServiceAccounts

In applications of robotics and automation, a *control loop* is an endless loop that supervises and readjusts the state of the system.

Namespaces in Kubernetes are similar 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, etc.) within a namespace share a common scope for names. Namespaces allow you to isolate different projects, teams, or customers within the same cluster and provide a mechanism to attach authorization and policies to specific parts of the cluster.

A *ServiceAccount* in Kubernetes is a special type of account that represents nonhuman 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:

- Pods communicating with the API server
- Pods communicating with external services (for example, authenticating to private container image registries)
- External services communicating with the API server (for example, in a CI/CD pipeline to release a new app version)
- Third-party security software: some tools rely on ServiceAccount identities to group Pods into different security contexts

Components of the Data Plane

The data plane (sometimes called the *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 is the worker nodes, which are physical or virtualized machines where Pods are hosted. Let's discuss the main components of the data plane here.

Kubelet

The *kubelet* is an essential piece of the data plane that runs in each worker node and acts as a bridge between the control plane and the execution of the workloads. A kubelet has an extensive list of critical tasks inside the cluster, but the main ones are described here:

Pod lifecycle management
: A kubelet monitors the health status of Pods and starts, stops, and restarts containers as needed. It also receives commands sent from the control plane regarding Pod creation, updates, and deletions. The kubelet is responsible for ensuring that the Pod and container configuration and status match the desired state specified by the control plane.

Reporting back the node's status to the control plane
: A 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. The 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 its main tasks are the following:

Service load balancing
: When a Pod communicates with a service, `kube-proxy` ensures that requests are load-balanced across the relevant Pods.

`iptables` *or* `ipvs` *rules management*

In Linux nodes, two modes are available in `kube-proxy`: `iptables` mode and `ipvs` mode. In general, you can think of `iptables` rules as simpler to configure, but it becomes inefficient and operationally complex at scale when you work with a large number of services. `ipvs` is an alternative mode that 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.

About iptables, IPVS, and eBPF

Kubernetes relies on networking components to manage communication between Services and Pods. Multiple options are available: iptables, IPVS, and the emerging star, eBPF. *iptables* is a technology that has been around for a while but was not created for extensive cases such as Kubernetes networking. iptables struggles with large rule sets, and this leads to performance issues causing latency and CPU overhead.

IP Virtual Server (IPVS) offers an alternative to iptables for load balancing and service proxying; it is more efficient than iptables, but it also becomes complex when dealing with a large number of services and endpoints and struggles to dynamically update rules.

In that context, *extended Berkeley Packet Filter* (eBPF) is a rising technology that allows programmable processing within the Linux kernel. Think of it as a tool that supercharges networking components as eBPF outperforms IPVS significantly through ease of network management, reduced overhead in terms of resource consumption, improved observability, and dynamic updates in real time.

The industry standard for Kubernetes networking using eBPF technology is Cilium, which is currently a graduated project in the CNCF landscape (see Figure 6-8). Cilium can replace `kube-proxy` entirely as it implements a Container Network Interface (CNI) that we will present in the next section.

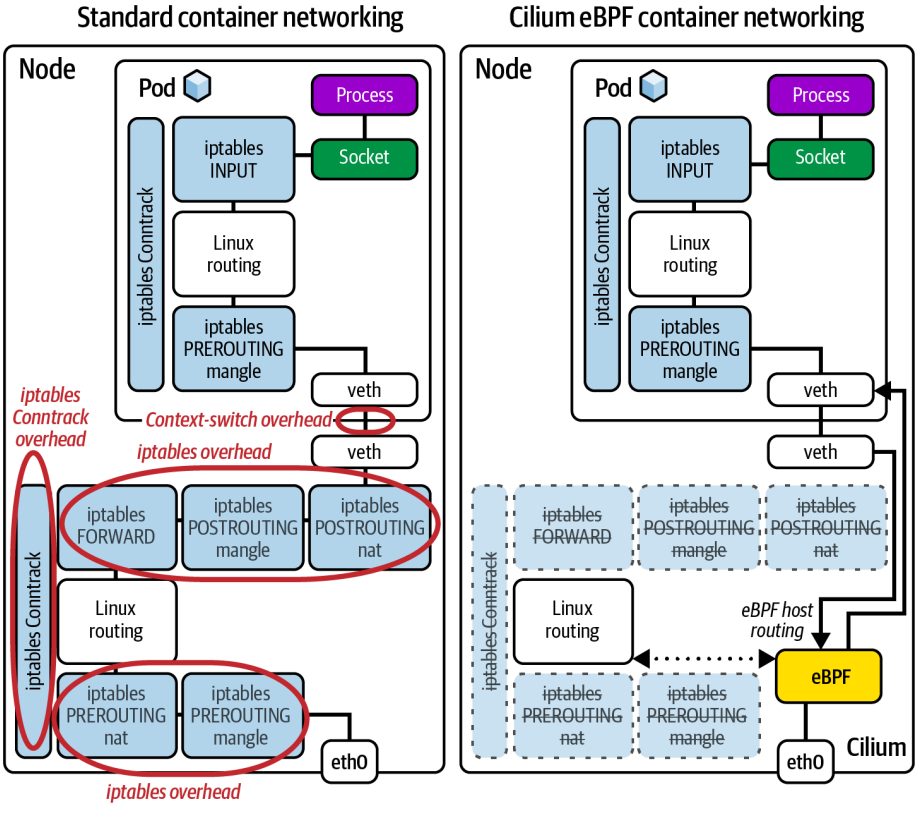


Figure 6-8. Diagram of traffic path with `iptables` mode versus Cilium eBPF (source: CNI Benchmark: Understanding Cilium Network Performance)

Inside Kubernetes Architecture

As described in previous chapters, Kubernetes is built as a distributed system: many pieces need to work together to make the system as reliable and trusted as required in today's applications. For that reason, you need to understand the architectural building blocks and concepts in order to deploy a cluster that fits the requirements of your applications.

In this section, we will focus on four fundamental architectural concepts:

- Container communication basics and single-container and multi-container Pod architecture patterns
- Container Runtime Interface (CRI)

- Container Network Interface (CNI)
- Container Storage Interface (CSI)

Single-Container and Multi-Container Pods

As described earlier in this 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 with a *multi-container Pod*.

You may be wondering why a multi-container Pod might 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 that is expected to receive big traffic (imagine an ecommerce website during Black Friday, for instance). The web server will handle user requests, serve product pages, and process transactions. The cache service (e.g., Redis or Memcached) will store frequently accessed data such as product listings, images, descriptions, and user session information reducing the load of the web server and ensuring a great user experience.

In this case, speed is key, and you 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 benefits 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 of whether to use it or not.

Multi-container Pod communication

There are three multi-container intra-Pod communication mechanisms:

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: 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).

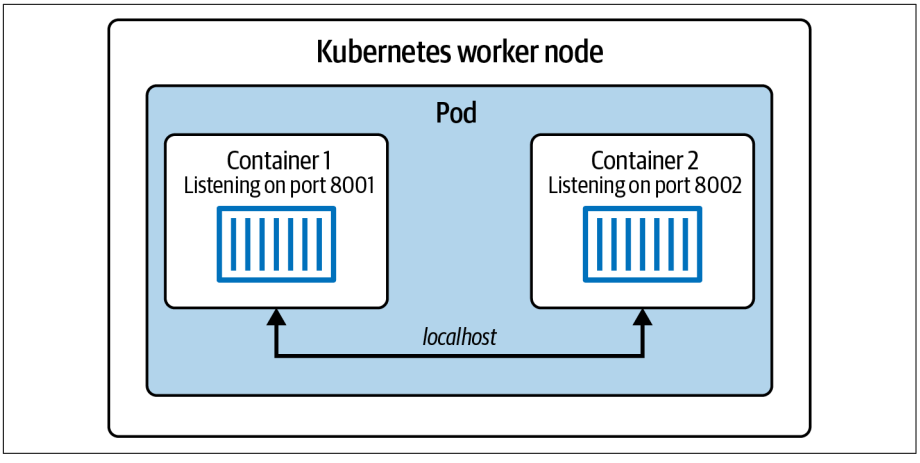


Figure 6-9. Multi-container Pod networking (for detailed information, see the Kubernetes documentation)

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 6-10).

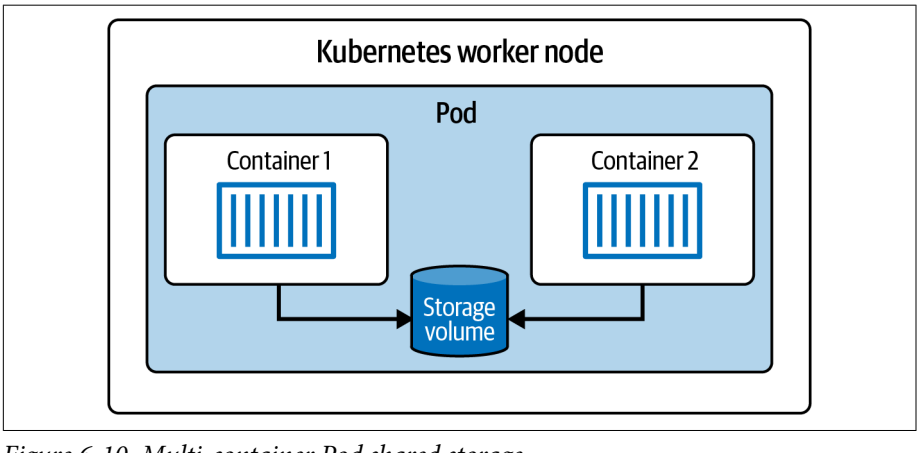


Figure 6-10. Multi-container Pod shared storage

Shared process namespace

An alternative method for containers to establish communication when belonging to the same Pod is through the shared process namespace (see Figure 6-11). When activated, processes in a container are visible to all other containers in the same Pod, and containers within the same Pod can 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`.

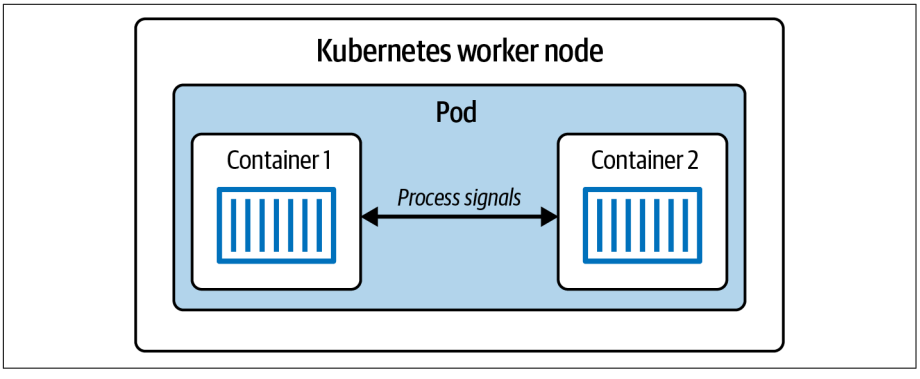


Figure 6-11. Multi-container Pod shared process namespace

Multi-container architectural patterns

Three specific architectural patterns 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.

These three primary patterns, with examples illustrating scenarios in which their application proves advantageous, are as follows:

Sidecar pattern

In this pattern, a *main container* is accompanied by one or more *sidecar containers* (see Figure 6-12). 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.

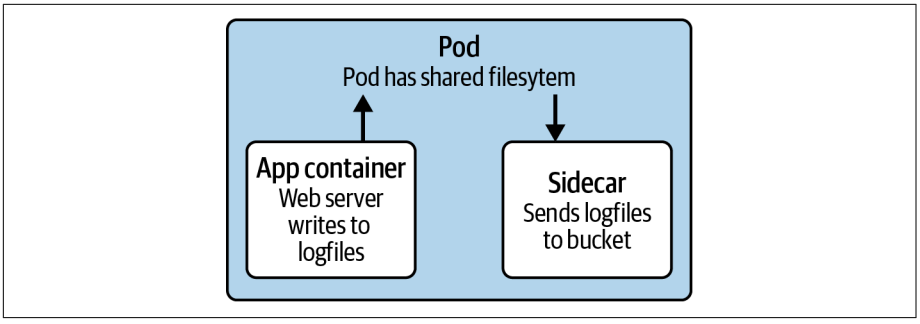


Figure 6-12. Sidecar container pattern example

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 (see Figure 6-13).

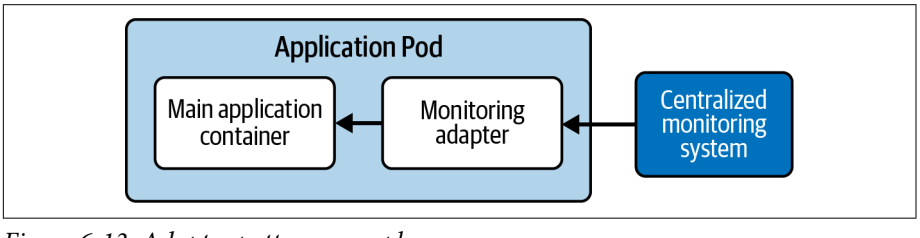


Figure 6-13. Adapter pattern example

Ambassador pattern

In this pattern, an ambassador container acts as a proxy or intermediary for the main container, as shown in Figure 6-14. It facilitates communication between the main container and external services.

One of the most common use cases for this ambassador pattern is including a container responsible for handling authentication, load balancing, or Secure Sockets Layer (SSL) termination for the main application container.

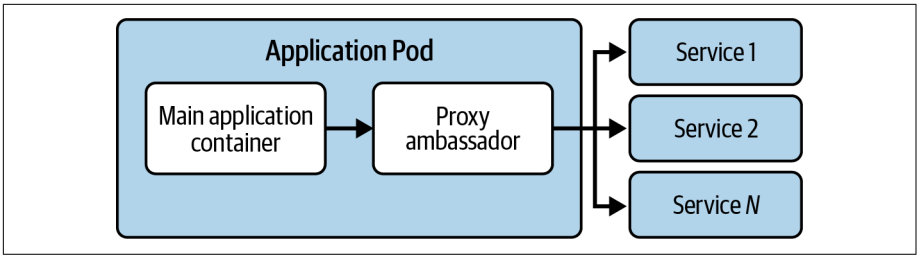


Figure 6-14. Ambassador pattern example

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.

About Multi-Container Pods

We first recommend reading this blog post written by Brendan Burns in 2015 on composite container patterns.

Kubernetes 1.28 introduced a new feature to support sidecar containers natively and ease their management. At the time of writing this book, it's still in the alpha stage (see Figure 6-15).

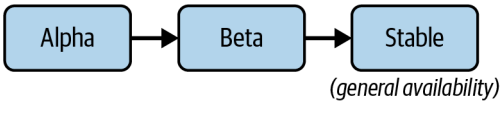


Figure 6-15. Kubernetes features lifecycle

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 that allows Kubernetes to seamlessly support a diverse array of container runtimes, all without the cumbersome need for recompilation. In short, the CRI is a plug-in interface that enables kubelets to use a variety of container runtimes.

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 these:

- containerd
- CRI-O
- Docker Engine
- Mirantis Container Runtime

Each container runtime offers a unique set of advantages, and within the Kubernetes community, there have been calls to expand support for a variety of runtimes. In Kubernetes 1.15, the CRI was initially introduced as an alpha release. Since then, it has matured and made substantial contributions, significantly enhancing Kubernetes's 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*.

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.

At this moment, you may be trying to figure out how each of these pieces (container runtime and low-level runtime) fits into the Kubernetes architecture. Figure 6-16 provides a simplified view to understand where each piece fits in the Docker and Kubernetes stack.

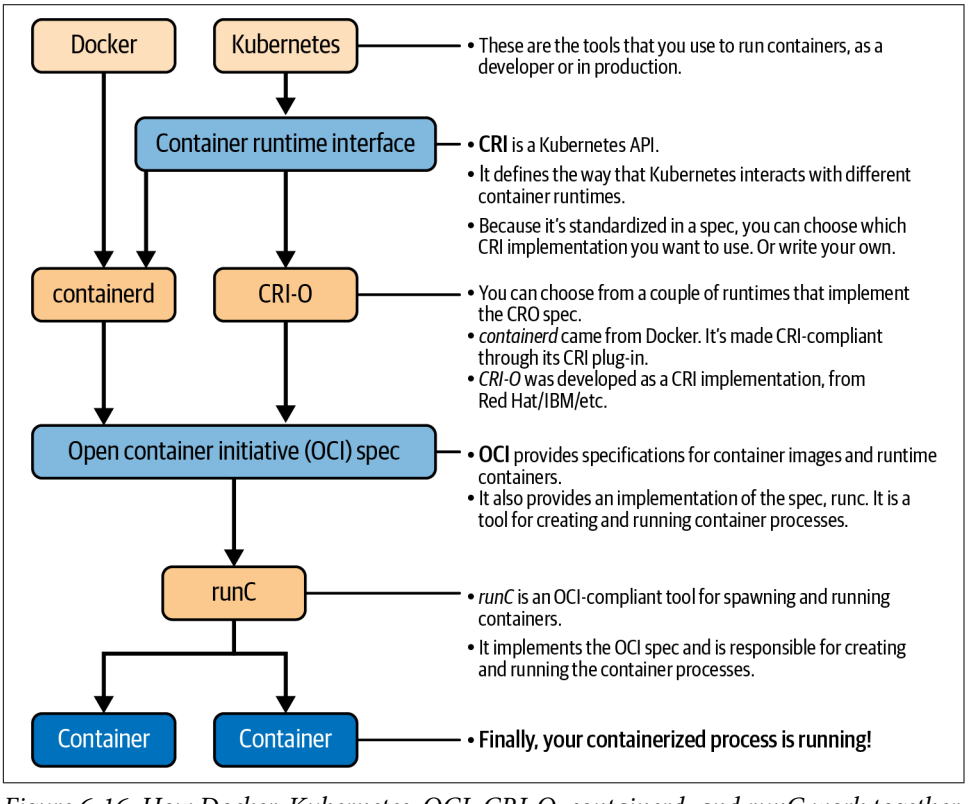


Figure 6-16. How Docker, Kubernetes, OCI, CRI-O, containerd, and runC work together

CRI-O and containerd 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 CNCF, 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 constraint reasons.

Container Network Interface

The Container Network Interface (CNI) is a specification that defines how container runtimes interact with network plug-ins. You can think of it as the bridge that connects containers and the network. For more detailed information about network plug-ins, we recommend reading the official documentation.

Container Storage Interface

The Container Storage Interface (CSI) is an initiative to unify the storage interface across container orchestrators (Kubernetes, in our case) and storage vendors (Ceph, PortWorx, NetApp, etc.). The interface defines the operations to allow dynamic provisioning and attaching and detaching storage volumes from containers.

Navigating Kubernetes States

In the previous section, we went through the main architecture pieces of Kubernetes. 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 Pods).

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*. 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, StatefulSets, and 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 that 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 that 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 persistent storage, enabling data to survive the lifecycle 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

A *DaemonSet controller* ensures that a specified Pod runs on every node within the Kubernetes cluster, unlike Deployments or ReplicaSets, which focus on keeping a given number of replicas across the cluster. For instance, imagine a cluster with five worker nodes and a Pod that is declared to run with `ReplicaSet = 3`. The ReplicaSet controller will ensure that three replicas are running across the cluster, but it does not mean at all that every node will run a replica.

DaemonSet's 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 by DaemonSet will run.

Job and CronJob Controllers

First let's define what a Job means in the Kubernetes realm. A *Job* represents a single, self-contained task or batch job that doesn't need to run all the time. It's a good fit for one-time tasks (for instance, a data import into a database) or running a batch (making a backup, generating reports, etc.) The Job controller will ensure that the task completes successfully, creating the Pods required for running the job. In the case of a failure, it will retry the task. A *CronJob* is a scheduled task in Kubernetes that needs to run periodically (hourly cleanup, daily backups, etc.) and uses a cron-like syntax to define the schedules.



We recommend reviewing the Kubernetes reference material on workload management.

Workload Autoscaling, Kubernetes Events, and Pod Lifecycles

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 from Figure 6-2) takes center stage, dynamically distributing computing and memory resources to cater to an application's evolving needs. Kubernetes provides a variety of mechanisms that allow scaling operations. Let's dive into those here.

Horizontal Pod Autoscaler

The *Horizontal Pod Autoscaler* (HPA) automatically updates a workload resource (Deployment or StatefulSet) for automatically scaling the workload to match horizontal scaling demands (see Figure 6-17). This means that in response to 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.

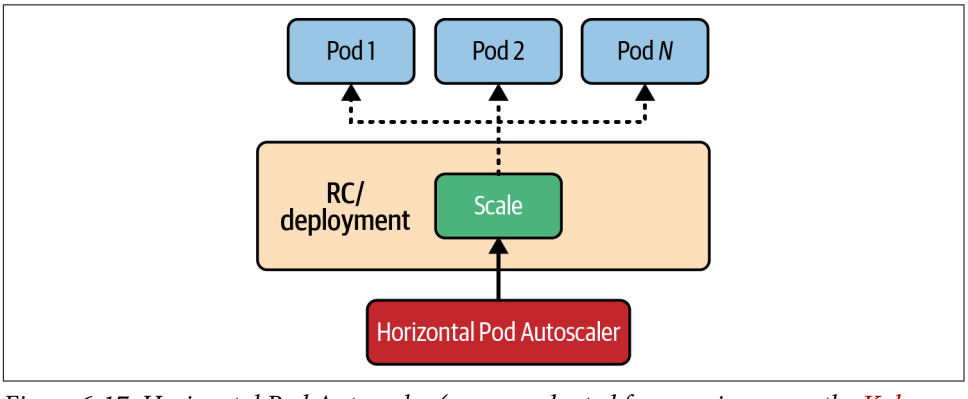


Figure 6-17. Horizontal Pod Autoscaler (source: adapted from an image on the Kubernetes website)

Vertical Pod Autoscaler

The *Vertical Pod Autoscaler* (VPA) provides the mechanism to adjust Pod resource requirements (mainly CPU and memory reservations) based on their historical resource usage (see Figure 6-18). It can scale vertically up (if more resources are needed) or down (if Pod requirements are over-reserved). Unlike the HPA, the VPA doesn't come with Kubernetes by default. It's a separate project that needs to be installed.

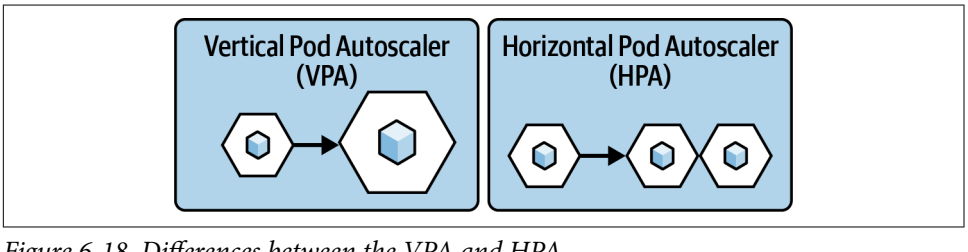


Figure 6-18. Differences between the VPA and HPA

The VPA project is found in the Kubernetes GitHub repository.

Cluster Proportional Autoscaler

Some workloads require scaling based on the cluster size. This includes core system components (CoreDNS is a typical workload for the Cluster Proportional Autoscaler [CPA]); based on the number of nodes or Pods in the cluster, the CPA dynamically adjusts the number of replicas required for the workload. Similar to the VPA, the CPA is not included in Kubernetes by default but it is available in its GitHub repository.

Autoscaling Based on Events

Another autoscaling mechanism often needed for your workloads is used when decisions for scaling up/down need to be triggered by an event. For example, imagine an application that processes images when a new image is stored in a queue. If a high number of images are in the queue waiting to be processed, you might want to scale out the number of Pod replicas to process the images faster. Once the number of images in the queue is low, the system can scale in to free up resources (or reduce costs). This is a simple example, but you can use this mechanism for any event-driven workload. The project that allows a Kubernetes cluster to implement autoscaling based on events is Kubernetes Event-Driven Autoscaler (KEDA).

CoreDNS and KEDA

CoreDNS is the default DNS service for Kubernetes. It's a graduated project within CNCF, and it's built as a chain of plug-ins, so you can use the plug-ins for only the functionality you really need. Among other functions, it can provide service discovery. The configuration is done through the *Corefile*, where you define the server configuration and which plug-ins to load. For detailed information about CoreDNS, you can check out the official site.

KEDA is another graduated project within CNCF. 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 check out the official site.

Cluster Autoscaler

We have seen how to autoscale Pods depending on a variety of situations. However, you may encounter another scenario 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 into your service. The HPA will evaluate the target metrics, and if it matches the configuration you specified in your

workload deployment, it will scale out your application by increasing the number of replicas (Pod replicas). Now you can rest because the Kubernetes HPA will manage it for you.

But what happens if the HPA updates 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? Now you will need to horizontally scale (scale out) your cluster so you have resources available to accommodate the new Pods. That's exactly what the Cluster Autoscaler is for; it allows you to dynamically adjust the cluster size (out and in) based on workload requirements.

For more information about Cluster Autoscaler, we recommend reading the official information in the corresponding GitHub repository.

Node Affinity, Taints, and Tolerations

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 a specific availability zone if using a hyperscaler infrastructure.

Taints and tolerations work together to ensure that Pods are not scheduled onto inappropriate nodes. *Taints* are applied to nodes and allow you to mark nodes that should not accept any Pods that do not tolerate the taints. *Tolerations* are applied to Pods and allow the scheduler to place Pods with matching 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. For more information, read the Kubernetes documentation on taints and tolerations as well as node affinity.

Now that you know the scenarios for scaling distributed applications and the autoscalers available, it's time to unfold the functionality of the scheduler.

Scheduler

The 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 queue, and the scheduler continuously selects 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. This 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 6-19). 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.

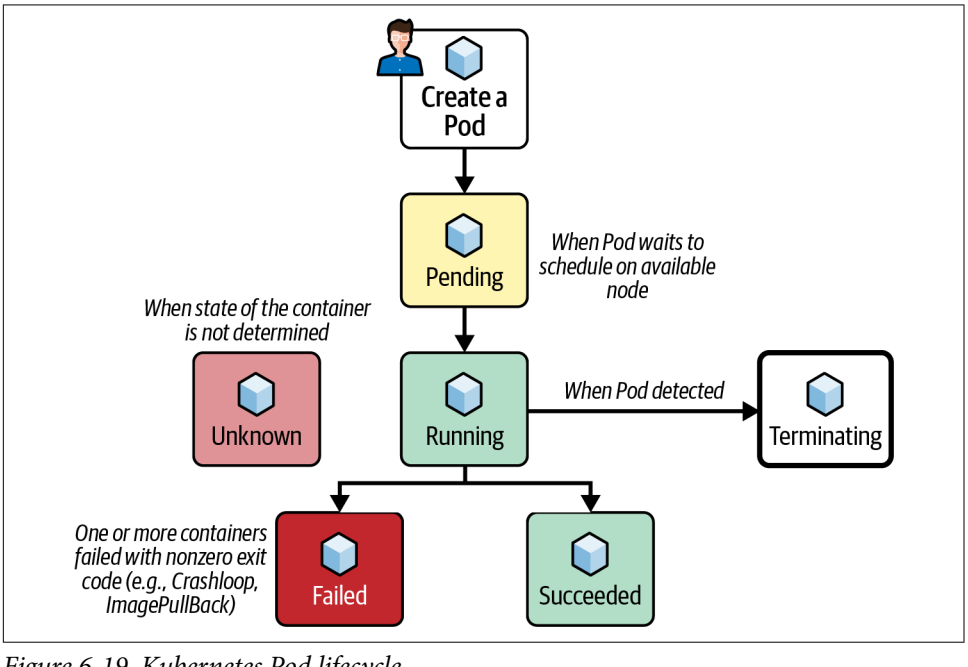


Figure 6-19. Kubernetes Pod lifecycle

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. This is the start of several phases for a Pod:

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
: 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 this 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 shut down, it is forcefully terminated, but this can lead to data loss (if the application 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 that 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.

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 lifecycle, as there's no mechanism for retention. They're short lived, available for only one hour after the event is generated.

The simplest way to view event objects is using `kubectl events`. See Figure 6-20 for an example.



Figure 6-20. `kubectl events` example

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 involves collecting system telemetry data such as logs, metrics, and traces:

Logs
: Logs are textual records that can be generated from applications, services, and Kubernetes system components. In Kubernetes, collecting logs from containers and Pods will help you understand the sequence of actions, debug issues, and trace the flow of data and requests.

Metrics
: Metrics provide quantitative data about the system's performance, offering numeric 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 multilayered and dynamic (Pods and their containers can be added or removed dynamically), and they do not expose logs, metrics, and traces in the same way as conventional systems. Achieving observability in Kubernetes involves stitching together different data sources (see Figure 6-21).

Data type	Data source or location
Cluster-level resource metrics	Metrics API
Cluster-level logs (kube-apiserver, kube-scheduler, etc.)	Master node(s) filesystem
Application logs	Containers (export logs to persistent storage to retain them)
Operating system logs (from master and worker nodes)	Node filesystem

Figure 6-21. Observability sources in Kubernetes

Let's examine how each of these work in the following sections.

Logs

Containers within Pods generate logs that are stored within the Pods themselves. When you need to access these logs, you have to interact directly with the Pod by using `kubectl` (i.e., `kubectl logs`), and unlike traditional systems where logs are generally centralized, Kubernetes distributes them across Pods (see Figure 6-22).

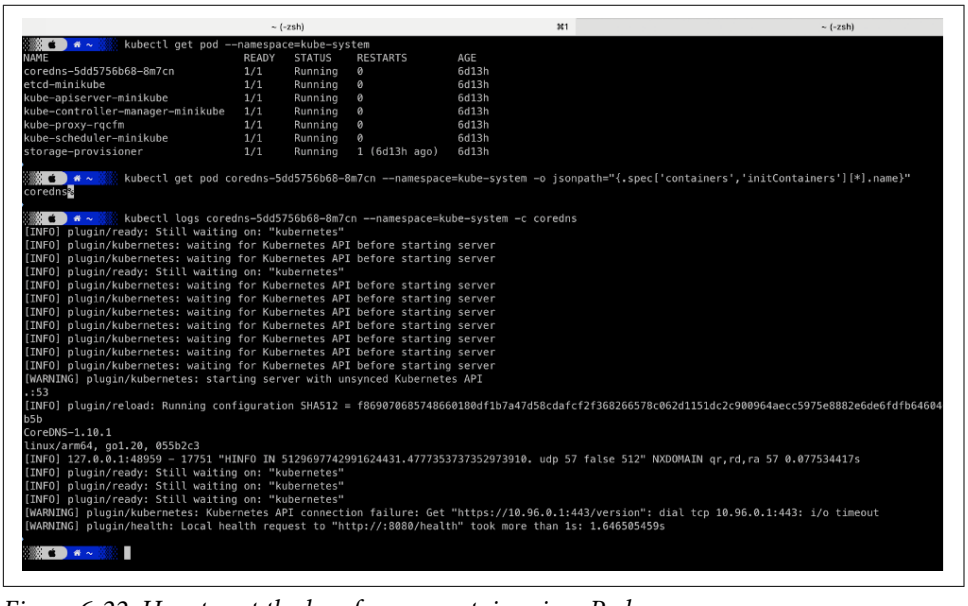


Figure 6-22. How to get the logs from a container in a Pod



In the example from Figure 6-22, here is the procedure:

1. List all the Pods running in `kube-system` namespace. This namespace is reserved for objects created by Kubernetes itself. In this specific namespace, you will find critical Pods that are responsible for the cluster's operation.
2. List the containers associated with the Pod `coredns-5dd5756b68-8m7cn`. This Pod contains only one container, whose name is `coredns`.
3. List the logs for the container `coredns` in the Pod `coredns-5dd5756b68-8m7cn`.

Metrics

Kubernetes exposes metrics through a 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 (see Figure 6-23). Read the documentation for more detailed information.

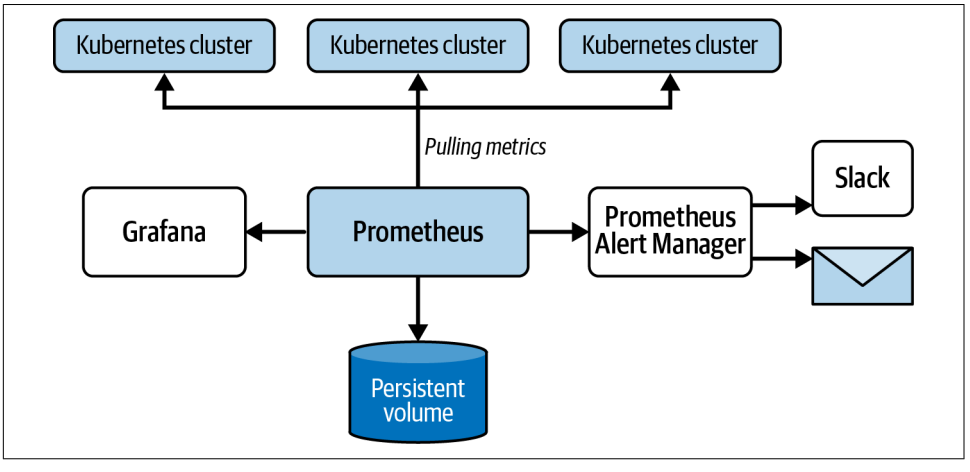


Figure 6-23. Prometheus and Grafana setup (source: adapted from an image on the Continuous Integration and DevOps Tools Setup and Tips blog)

The HPA and VPA, based on CPU/memory metrics from metrics-server, collect resource usage data from kubelets and expose this information through the Kubernetes API server via the Metrics API. We will cover this further in the following section as part of the resource metrics pipeline, but you can also read the documentation and review the metrics-server GitHub repository for more information.

Traces

Kubernetes doesn't provide native tracing data, so you will need to integrate additional tools (such as Jaeger and OpenTelemetry) to capture traces. OpenTelemetry and Jaeger will let you implement a distributed tracing system. OpenTelemetry is focused on instrumenting your applications so you can capture trace data from them, and Jaeger complements OpenTelemetry by providing powerful trace analysis capabilities with a UI for analyzing the traces.

Both are open source projects under the CNCF umbrella; Jaeger is in the graduated stage, and OpenTelemetry is in the incubation stage. Check out the Jaeger and OpenTelemetry websites for more information.

The Kubernetes monitoring stack is not constrained by a single monitoring solution. Instead, it offers flexibility by accommodating different approaches to gather monitoring statistics, including the resource metrics pipeline and full metrics pipeline.

Resource Metrics Pipeline

The resource metrics pipeline offers a set of metrics related to cluster components, including the HPA controller. It 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 an API (see Figure 6-24).

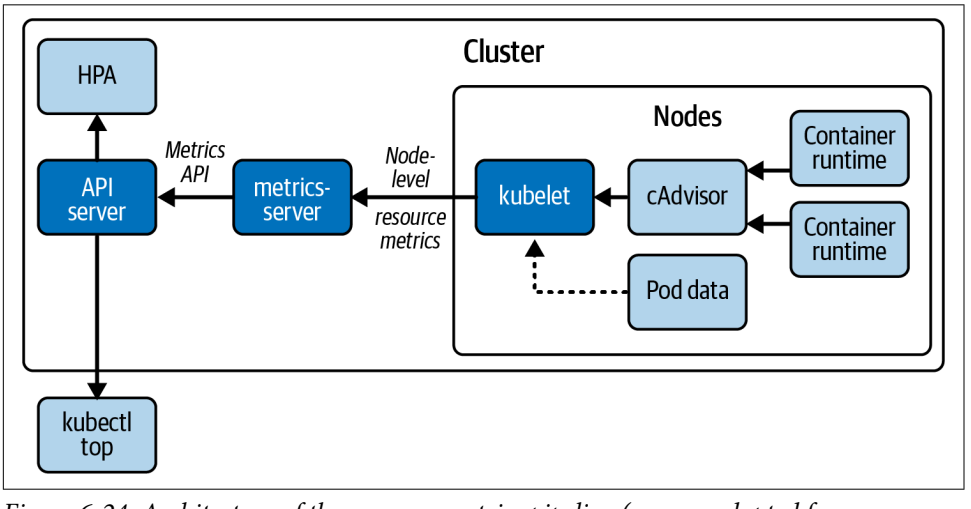


Figure 6-24. Architecture of the resource metrics pipeline (source: adapted from an image on the Kubernetes website)

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 cAdvisor component is a daemon for collecting, aggregating, and exposing container metrics included in the 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, 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 HPA. 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.



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 HPA purposes. The `external.metrics.k8s.io` API provides customer metrics, but they are external to the cluster, usually coming from third-party systems (database connections, API response time, etc.); these are used for autoscaling or other requirements within your application.

To understand the difference, imagine you have an ecommerce website that interacts with an external payment gateway. Your website needs to scale some microservices based on the response time of the payment gateway. How could you do it? In this case, you could expose this metric via the `external.metrics.k8s.io` API and then configure HPA to use it for scaling. See the documentation for more detail.

It is important to highlight that Kubernetes has been designed to seamlessly integrate with OpenMetrics, a project within the CNCF Observability and Analysis—Observability Projects category. The aim is to extend the Prometheus exposition format, ensuring compatibility while offering a rich set of metrics transmission capabilities.

When exploring the CNCF landscape, you will encounter a multitude of monitoring projects that can effectively work with Kubernetes, collecting metrics data and assisting in cluster observation. The choice of monitoring tools is vast, ranging from open source solutions to paid SaaS 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 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 follows:

Resource type
: Name used for the URL (Pod, namespace, service).

Kind
: Every resource type has its own schema that defines its properties. It's commonly called *kind*.

Resource
: A single instance of a resource type.

Collection
: A list of instances of a resource type. For some resource types, the API includes one or more subresources, 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. Kubernetes objects are persistent entities within the Kubernetes system and are represented in the Kubernetes API. Examples include Pods, Services, ConfigMaps, and ReplicaSets.

You express Kubernetes objects by using YAML format. This format defines the desired state of an object. The Kubernetes API ensures that the actual state matches the desired state. An example of creating a Kubernetes object is creating a Pod (see Figure 6-25).

```
~ 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 6-25. Creating a Pod object in Kubernetes

Kubernetes Storage

Kubernetes not only excels in managing containers, as we have presented in previous chapters, but also offers robust solutions for handling storage in a platform where applications scale in and out constantly. In this section, we'll explore the world of Kubernetes storage, from the fundamental concepts to advanced storage strategies, ensuring that your applications have access to reliable and scalable storage resources. Let's dive into the main concepts related to Kubernetes storage.

Persistent Volumes and Persistent Volume Claims

Persistent volumes (PVs) and persistent volume claims (PVCs) form the foundation of Kubernetes storage. Volumes are the solution when you need to ensure that your application's data persists even if the underlying Pod is rescheduled or deleted. They exist as separate entities at the cluster level (not tied to any specific Pod, unlike regular volumes, which exist at the Pod level). A 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 storage class (local storage, premium storage, cloud-based storage, etc.).



Three access modes are 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 (supported only 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

In Kubernetes, you can define 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 noncritical data. By defining different storage classes, you can ensure that critical data gets high-performance storage, while noncritical 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. The most commonly used volume types are as follows:

EmptyDir
: This volume type is ephemeral and exists only for the Pod's lifetime. It's useful for temporary storage.

HostPath
: This 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 are a specialized controller within Kubernetes which, in the context of storage, enables data to survive the lifecycle of a Pod and persist even in the face of node disruptions.

If 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

As explained earlier, the Container Storage Interface (CSI) is an initiative supported by CNCF that 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.

You can utilize CSI-compliant storage providers in Kubernetes, and today there are roughly a hundred providers compliant with the standard. We recommend always to check the latest list for the most up-to-date information.

Dynamic Provisioning

Dynamic provisioning is a game-changer in Kubernetes storage. It automates the allocation of storage resources as needed, eliminating manual intervention.

Imagine that 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. For more information, read the documentation.

Networking and Service Mesh

In the preceding sections, we 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 aim 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 in order to manage containerized applications effectively:

Container-to-container communication
: Within a Pod, containers often need to communicate with each other. As we discussed 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
: Applications often consist of multiple Pods, and Kubernetes orchestrates the deployment of Pods across nodes within a cluster. To enable Pod-to-Pod communication, Kubernetes employs a combination of network plug-ins, routing, and IP addressing. In one configuration, each Pod is 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 (see Figure 6-26). They allow you to define routing rules for incoming traffic. These rules include hostnames (which domain or subdomain the ingress controller should handle), paths (URL paths and how to map them to specific services) and TLS configuration for secure communication. See Figure 6-27 for an example.

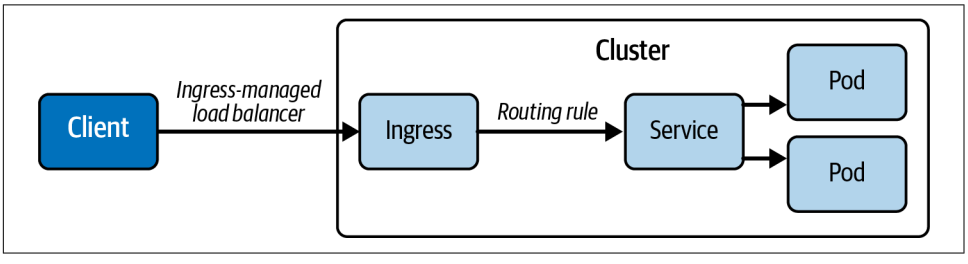


Figure 6-26. Ingress controller

```
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-servic
                port:
                  number: 80
          - path: /api
            pathType: Prefix
            backend:
              service:
                name: api-service
                port:
                  number: 8080
```

Figure 6-27. Example of an ingress controller definition

Some of the most popular ingress controllers are NGINX, Traefik, HAProxy, and Contour. See the documentation for more information.

Network Plug-ins

In Kubernetes, *network plug-ins* are essential components that provide the underlying infrastructure for networking within a cluster. They define how Pod-to-Pod communication is managed and orchestrated both within the same node and across nodes. Network plug-ins 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 plug-in are as follows:

Routing and networking rules
: Network plug-ins 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 plug-ins 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 plug-ins 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 nodes (see Figure 6-28).

Network policies
: Network plug-ins 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.

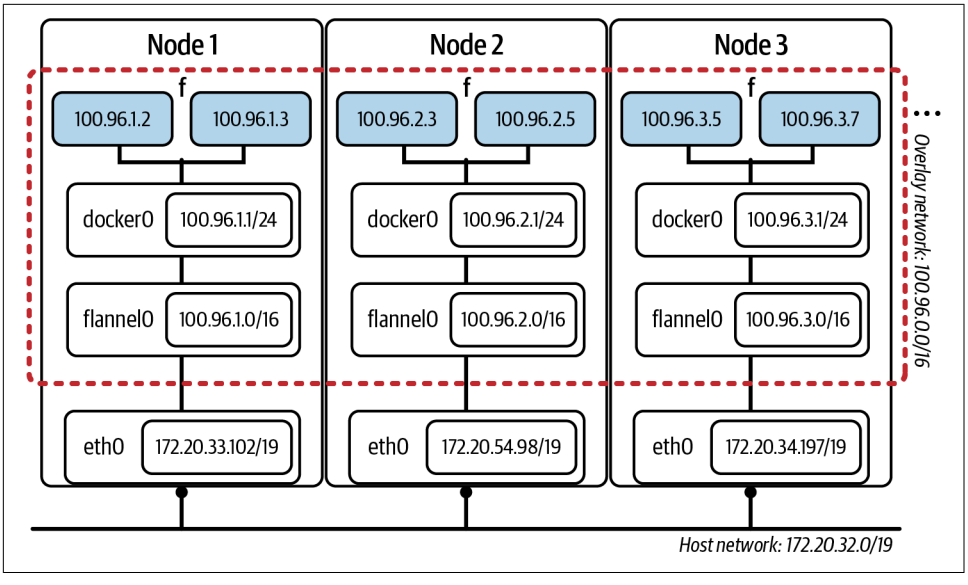


Figure 6-28. Example of an overlay network using Flannel (source: adapted from an image on the DevOpsSchool website)

Network plug-ins, such as Calico, Flannel, and Weave, manage routing and networking rules to ensure seamless communication across nodes. Here are some additional resources if you are interested in reading more:

- Calico web page
- Flannel GitHub repository
- Weave GitHub repository

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 next.

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 namespaces. This means that applications in one namespace can easily access services in other namespaces by 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. You can read more about external DNS integration in the external-dns GitHub repo.

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.

To put it simply, 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 the following:

- Dynamic routing and traffic control (can route a request based on a variety of criteria like service version, latency, or even custom rules)
- Mutual TLS for securing communication
- Distributed tracing and telemetry providing a comprehensive view of the system's performance and behavior; this can be integrated with the external tools discussed earlier in the observability section 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 (see Figure 6-29).

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.

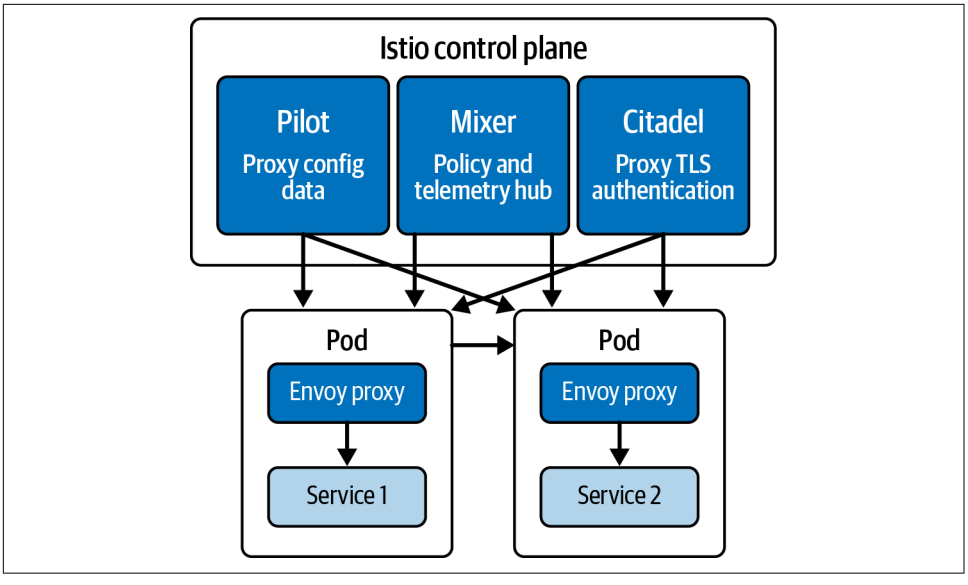


Figure 6-29. Istio service mesh example

Advantages of using service mesh

Let's discuss the 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. This includes routing, rate limiting, and fault injection and provides the following features:

  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 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 a weighted load balancer is that traffic shifting focuses on transitioning between different versions of a service and the focus of a weighted load balancer is on 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 there:

  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 the following:

  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 CNCF, there are two graduated projects in the Service Mesh domain: Istio and Linkerd. Explore the preceding links if you want to read more.

Kubernetes Security

As we have presented in the previous sections, Kubernetes offers advantages in terms of scalability, fault tolerance, and resource utilization in scenarios where distributed system 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, but it would require a whole book to cover all the aspects related to this. See *Learn Kubernetes Security* (Packt) by Kaizhe Huang and Pranjal Jumde 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 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 contain vulnerabilities, they can become the entry point for attackers.

Various strategies can be adopted to minimize the risk of image vulnerabilities. We discuss the principal ones here:

Image scanning
: There are a good number of tools (for example, Harbor, Anchore, Docker Scout, and 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 well-known project for image scanning is Harbor. Harbor provides static analysis of vulnerabilities in images through 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. Include only 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 throughout the lifecycle. Having the same policies applied from dev to prod will minimize the attack surface 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 (see "Kubernetes RBAC" on page 190) policies, and use network policies to restrict access to the API server.

etcd security
: The etcd database stores critical cluster information. Encrypt etcd data at rest and in transit. Maintain regular backups and restrict access to etcd endpoints to authorized users and components. See the documentation for more information.

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 alert about unusual behavior.

  Read more about node security on Kubernetes's Pod Security Admission web page and the Falco website.

Implement network policies: Controlling communication
: Kubernetes network policies define how Pods can communicate with each other and with 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 by using Kubernetes Secrets. Encrypt secrets at rest and restrict access to them through RBAC and Pod service accounts. Read more about secrets management on Kubernetes's Secrets web page.

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 containerized applications.

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 scenarios.

The main components of K8s RBAC are as follows:

Roles
: Define permissions within a specific namespace

ClusterRoles
: Non-namespaced resources allowing you to define permissions cluster-wide

RoleBindings and ClusterRoleBindings
: Associate roles/ClusterRoles with users, groups, or service accounts

Service accounts are nonhuman accounts 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 service account at creation time.

Other Kubernetes-Related Components

Let's discuss a few additional Kubernetes-related components, including Helm, Dapr, ConfigMap, and minikube.

Helm

Helm is the de facto package manager for Kubernetes and a graduated project in CNCF. It simplifies the management of Kubernetes applications, allowing you to define, install, and upgrade any type of application through Helm charts. *Helm charts* are packages of preconfigured Kubernetes resources (Deployments, Services, ConfigMaps) that describe the application. See Figure 6-30 for an example.

Artifact Hub is the central repository not only for Helm charts but also for other types of artifacts such as plug-ins and Kubernetes modules programmed in KCL among others.

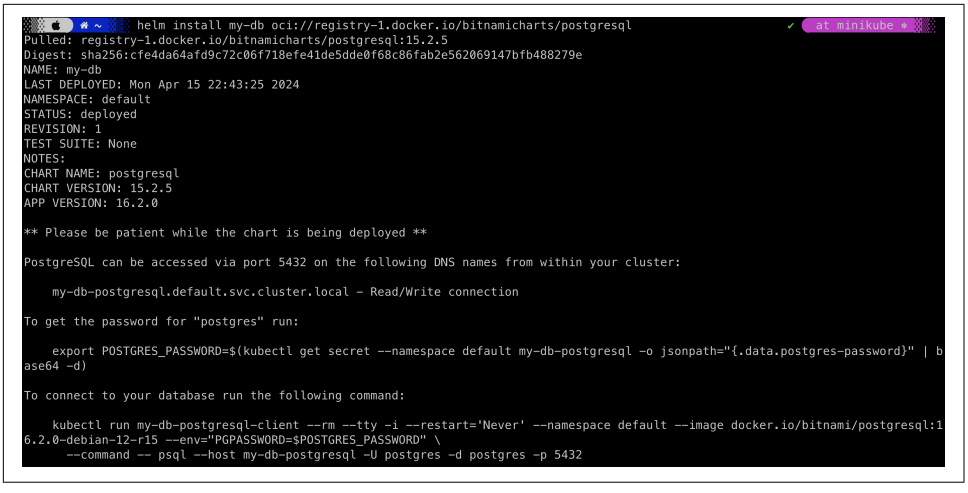


Figure 6-30. Installing a PostgreSQL database by using Helm

Dapr

Dapr (Distributed Application Runtime) is a project within CNCF at the incubation stage. It is used to design and deploy microservices applications (see Figure 6-31). An application architected and built for microservices presents a set of well-known challenges: encryption, message brokering, observability, service discovery, etc. You now have a good understanding of these concepts. Now imagine that not all the microservices were developed in the same programming language. Should you rewrite the "logic" or "patterns" for any of those languages? Each microservice likely has its own lifecycle, and in larger applications, many teams will be working on different microservices. How would you handle and test the cross-microservice calls?

That's exactly the space that Dapr wants to cover. It abstracts the application logic and programming to the underlying services through a well-defined set of RESTful APIs to call services through HTTP. Behind the scenes, 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.

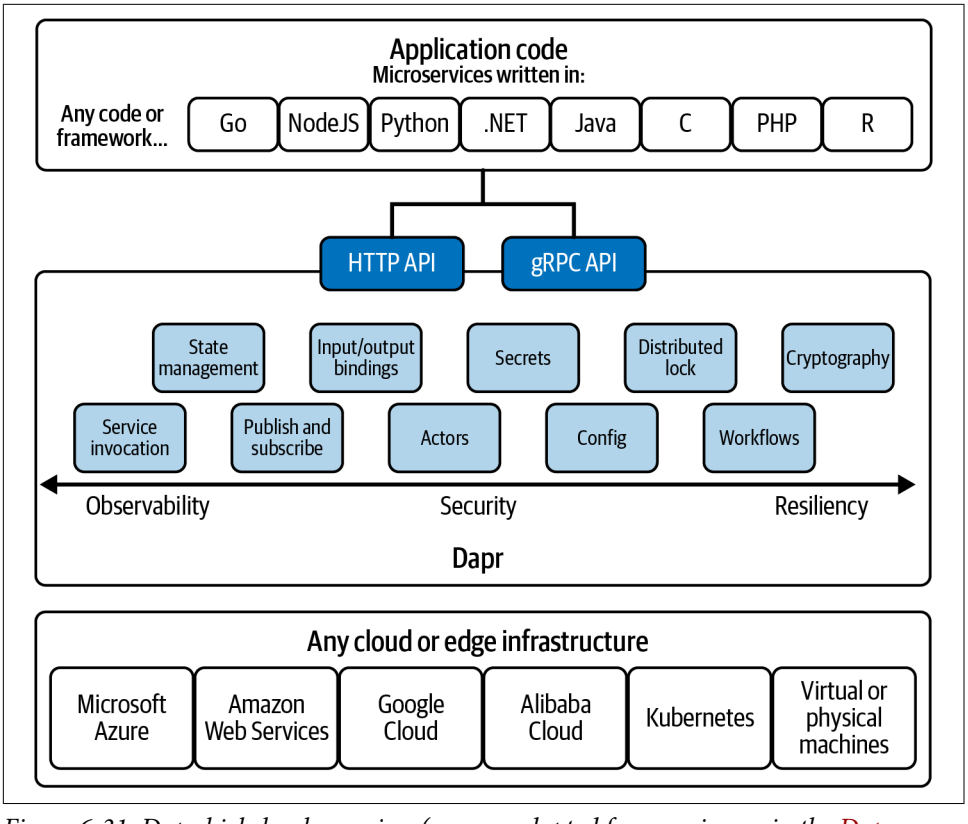


Figure 6-31. Dapr high-level overview (source: adapted from an image in the Dapr documentation)

ConfigMap

ConfigMap is a Kubernetes resource that allows you to store configuration data separately from your application code, providing a better way to configure settings, environment variables, and other nonsensitive data. See Figure 6-32 for an example.

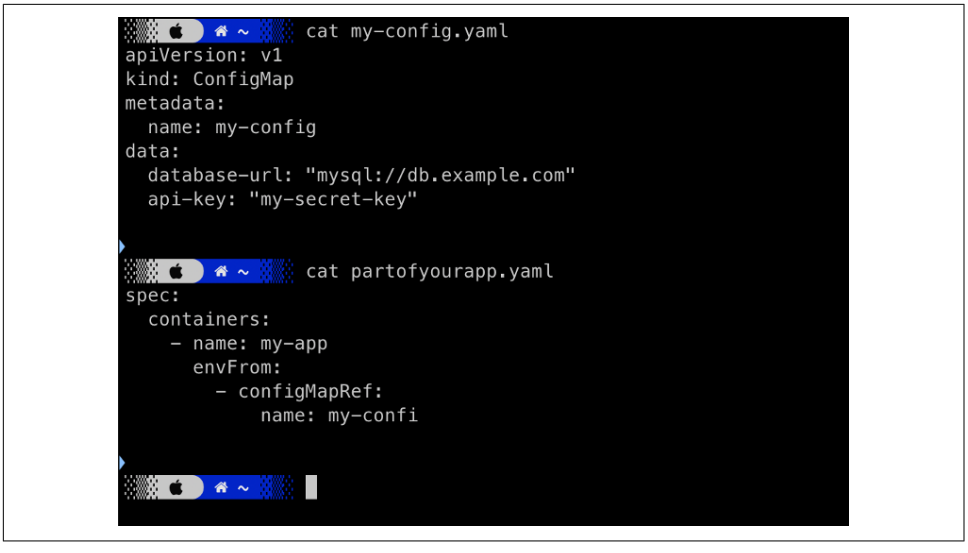


Figure 6-32. Creating a ConfigMap and using it when creating a Pod

minikube

minikube is a tool that helps you 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.

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`. Short for "Kubernetes Control," `kubectl` is the official command-line interface (CLI) for Kubernetes.

Following are some of the essential `kubectl` commands that allow you to interact with and manage resources within a Kubernetes cluster. Many more commands and options are available, so we recommend that you 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 (Figure 6-33).

```
kubectl version
Client Version: v1.29.2
Kustomize Version: v5.0.4-0.20230601165947-6ce0bf390ce3
Server Version: v1.28.3
```

Figure 6-33. `kubectl version` sample output

`kubectl cluster-info`
: Display information about the Kubernetes cluster, including the API server URL (Figure 6-34).

```
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 6-34. `kubectl cluster-info`

`kubectl get`
: Retrieve information about various Kubernetes resources (Figure 6-35). 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 6-35. `kubectl get` samples

`kubectl describe`
: Get detailed information about a specific resource (Figure 6-36). Here are a couple of 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=6
    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 6-36. Pod description sample

`kubectl create`
: Create resources from a YAML or JSON file. Here are a couple of 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. This command can create or update resources (Figure 6-37). Here's an example:

  `kubectl apply -f deployment.yaml`
  : Apply changes defined in *deployment.yaml*.

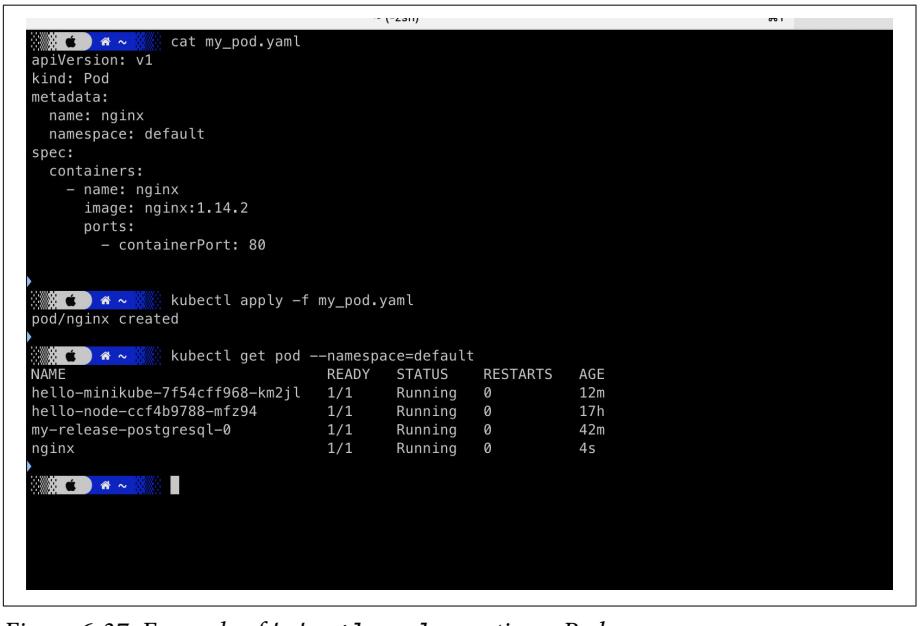


Figure 6-37. Example of `kubectl apply` creating a Pod

`kubectl delete`
: Delete resources. Here are some examples:

  `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 (Figure 6-38). Here's an example:

  `kubectl exec -it` *`pod-name`* `-- /bin/bash`
  : Open a shell in a specific Pod.

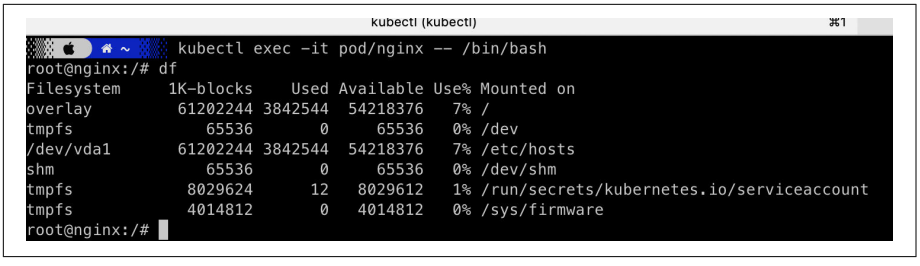


Figure 6-38. `kubectl exec` sample

`kubectl logs`
: Display the logs of a specific container in a Pod. Here are a couple of examples:

  `kubectl logs` *`pod-name`*
  : Display logs for the primary container in a Pod.

  `kubectl logs` *`pod-name`* `-c` *`container-name`*
  : Display logs for a specific container in a Pod.

`kubectl events`
: Display the events of a specific container in a Pod (Figure 6-39). Here are a couple of examples:

  `kubectl events` *`pod-name`*
  : Display events for the primary container in a Pod.

  `kubectl events` *`pod-name`* `-c` *`container-name`*
  : Display events for a specific container in a Pod.



Figure 6-39. `kubectl 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 a 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 are a couple of examples:

  `kubectl rollout status deployment` *`deployment-name`*
  : Check the status of a Deployment rollout.

  `kubectl rollout undo deployment` *`deployment-name`*
  : Roll back a Deployment to a previous version.

Expert Insights: James Spurin

Adrián: Welcome, James Spurin. We are very happy to have you here. How are you doing?

James: Yeah, very well, thank you. How's it going yourself?

Adrián: Doing well. Enjoying this week's work and working on the podcast, the book, etc. Busy times. Same for you, I know.

James: Yeah, absolutely.

Adrián: Even if I think a lot of people in the community know you already, who are you, and what's your relation with the cloud native community?

James: My name is James Spurin. I'm a CNCF ambassador and a Docker captain as well. I do a lot in the community in various ways. So professionally, I worked in IT for around 25 years. I worked in engineering roles where I covered a variety of areas. So I've worked as an engineer and manager. I worked in DevOps-like roles, enterprise storage, software development. Previously, I did my master's in software development. So there was a lot of crossover with this. I decided to go into technical course creation full time two years ago. The KCNA is currently my third course.

Adrián: And what are the main insights you're getting from all this KCNA experience about the learners and their professional background, the difficulties, the challenges?

James: I would say it's the community, the learners. There's a perception which is often out there for the KCNA examination. And I think that perception by many is typically wrong. I remember when this first came out and it was kind of seen as a qualification that suits a professional, like a manager or somebody who is actually less hands-on with Kubernetes. And it's completely wrong. I'm seeing a whole variety of people who are taking and targeting the KCNA, from people who want to actually start their journey with containers and Kubernetes, right through to people who are managers. So yes, I'm seeing a whole host of people. While there's a huge focus on the Kubernetes side, Kubernetes being 44% of the examination, this course covers so much in so many areas. You've got the cloud native side, you've got the Kubernetes side, you've got the telemetry and observability, you've got the GitOps.

And the world is changing, especially in the cloud native side with respect to Kubernetes. While there's a huge focus and a huge demand to learn Kubernetes, the cloud native ecosystem is growing tremendously. And a great example of this is KubeCon. Many years ago, you would go to KubeCon and that focus, in particular, was Kubernetes. This has transitioned from KubeCon to KubeCon and cloud native con. And each year when I go to KubeCon, I see more and more emphasis moving away from it being a Kubernetes-focused event. And this is the thing to think about. Kubernetes is now just the first graduated project. And all that terminology is, you know, understanding that whole lifecycle and crossing the chasm, etc. All of that is important to understand from the KCNA perspective. But Kubernetes is just one project now. It's the first graduated one. But there are so many projects in that graduated space, in that incubating space, in that sandbox area.

And going back to the demographic, I'm seeing more people now realize the value of the KCNA qualification, especially as a foundation course, because previously where the CKA or the CKAD was the introduction for most people, what you're actually coming away with from that, you're coming away with brilliant skills. It's excellent. You get a real deep dive into Kubernetes, but it's particularly that area only. With the KCNA, I'm seeing more people, more demographics who are looking to understand that, looking to understand cloud native, what it is, how it works, what the benefits are, how that structure works within the CNCF, how, for example, you would run a project like Kubernetes at huge scale and keep it vendor neutral and the structures in place. And not only that, the various other projects then, which you come away from this qualification with.

So you end up with the cloud native knowledge. You end up with Kubernetes knowledge. You end up with a very good understanding of different products in different spaces: Argo, Prometheus, Grafana, understanding things like Linkerd, for example. It's very diverse. And you can come away from this with a strong foundation to then go any which way you like, CKA, CKAD, CKS, Istio, GitOps, etc. So, yes, a long answer to your question. But I think that I'm certainly seeing positive changes in the audience who are taking the KCNA.

Adrián: I totally agree. I feel like it's becoming more tangible, the kind of audience that was supposed to be the audience for the KCNA. And anyone who takes this certification after that can go on different paths, not always Kubernetes. That obviously has a very important role in this community. How would you prepare if you were joining the committee now and trying to learn all these topics, knowing that there is so much to learn, so much documentation, so many resources? How would you approach it from a new learner point of view? What would be your recommended strategy for learning and passing this exam, but also to join these kinds of roles later?

James: Yeah, that's a great question. Previously, my journey with cloud native was the typical route that many people have taken with this. So I went down the route of Kubernetes. I studied for the CKA. I studied for the CKAD. And I passed those. And I was trying to get more and more involved in the cloud native community. And it always felt quite overwhelming in the bigger community in terms of understanding this and understanding how you can learn and be effective. And not only that, have a good grasp and good understanding of all of the various projects. It can be quite hard sometimes to relay how beneficial the cloud native educational aspect of this is and how this might help somebody more so in their career than this Kubernetes knowledge.

I was talking to someone who works in a financial, very senior position doing Kubernetes at the moment. And his focus in particular is on achieving the CKAD and the CKA. I was trying to encourage him to do the KCNA. One of the areas we got into a discussion on was, OK, so you're in your work environment and your manager says to you, we need a good solution for GitOps. How do you go about this? He gave a typical kind of response: OK, I would Google this, I would look into this. And I was explaining that with this core curriculum, certainly for me, when I studied this, it covered many areas which I was unsure of or they were gray areas or I didn't quite understand. I was explaining that if you have this knowledge, you would understand the lifecycle that a project will go through from sandbox to incubated to graduated. You'll know that the sandbox level is a very low bar for entry. And moving on, you'll know why incubation is special, and especially the challenges in moving to that.

If you are working in a profession, and you have that kind of knowledge, and your manager said to you, we need an effective product for this, you're able to go to look at the CNCF landscape, and you might be in a position where you can dismiss those options which are sandbox. You would have that knowledge of structures, such as Special Interest Groups, etc. You would be able to see how those communities are doing. And learning that area of the cloud native side, I think it can complement careers and it can make you more effective in your roles in ways that you would not even understand.

Here's a very simple example: I always found some of the terminology, like the Special Interest Groups and the TAG, quite confusing. I remember being at one KubeCon,

and I was a little bit lost on this and it was the transition point where previously Special Interest Group was used both by Kubernetes projects and the CNCF. Then the CNCF switched over to TAG to avoid the ambiguity. And when you're trying to get into this, and you've got that information overload with no filter, it's hard to be as effective as you can. I think the KCNA, especially with that cloud native side, provides you that filter, with many different areas, and then you can be highly effective with the knowledge you've got.

For me, the main contribution of this exam and certification—not that the Kubernetes part is not important, obviously it is—is the cloud native ecosystem overview and organizing the content. Before, the content was on the CNCF website, but asking someone to take a certification helps them to understand. Then they have the fundamentals to do what you mentioned, choosing a tool, understanding the mechanism of the community, knowing when is the tool mature enough to adopt it, or if it's not, what's the trade-off? I love that example, actually.

For me, when I was looking into this as a course and a potential option, I looked at the curriculum and I went through and I did my initial draft of what I was going to do to cover this. And then I did the exam and I was very, very surprised. And again, this is the thing, it is not an easy intermediate basic exam. There's a lot to it, a lot of detail and depth. I did the exam, then I went back through and compared the areas of my notes and I was like, I really need to expand this because there is so much more than I anticipated in the examination. And when I then started diving into those areas and really looking into them in detail, it was kind of like that moment where you're enlightened.

With 16% of knowledge on the cloud native side, you can think of it as a power up. It will make you more effective, whatever your job role is. Another great example is cloud native personas. If you're focused on the tech side, this might not be something you're interested in or you will think you know it already, the various different roles, the DevOps, the SRE, the FinOps, the security operations, etc. When you focus on this from a cloud native viewpoint and you take this in a direction of how companies like Google, Meta, etc. are using these different roles and you start asking how do they combine them? And what you realize is the role they're working in will deviate quite a lot from this. When you understand those pillars and you see how these bigger organizations are using these pillars, it can be extremely effective. If you were a manager and you've got that knowledge, you can decide, OK, I'm going to run my teams the same way Google is running the team.

It can be extremely powerful, especially where you combine that role where you've got these teams. I love trying to embrace that cloud native culture. It does not matter where you are. It does not matter if you're in that team and are one of the core members of that team or if you're a manager. Cloud native does not only apply to architecture. If a company is adopting a cloud native culture, you can be influential

in that culture. You can make changes and push positive changes, whether you're actually there, whether you're a manager, whether you're in another area. It's brilliant. Having this kind of knowledge is a power up for yourself.

Adrián: Totally, totally. I want you to tell more about your training and go into detail about what people will find in your KCNA class. And then if you want, you can add from your knowledge of the community, something that you would recommend for learners to get this kind of 360 experience.

James: OK, yes. So with my courses in general, I take a kind of a unique approach with everything that I go through on this. And one of my fundamental rules is that I will always create content as I would like to have that content and what I would like to get out of it. And for this reason, sometimes particular areas I will go into a lot more detail than is required. But again, I have this rule as a standard that I make in all content as I would actually like it.

A nice example of this is in my particular efforts with RBAC, role-based access control. Now, from a KCNA viewpoint, you don't need to have that much knowledge on the internals of RBAC. You really need to understand what the role of RBAC is and some basics on how it works, but you don't need to go into the low levels. From my side on the course, this is probably one I have spent the most time on. I've split this into three videos. The first video covers what you need to know from a theory perspective of RBAC. And this is how it works. This is how it integrates, etc. I put a disclaimer on this, you know, at the start. If your main focus is the KCNA, you can just watch this one video. If, however, you want to really understand RBAC and you really want to have a knowledge of this and be effective with it, you can watch the other two.

My aim on this is if you just wanted to pass the KCNA qualification, there's heaps of resources out there. With mine, I'm trying to push beyond that because I want people not only to get the KCNA qualification, but to actually have this knowledge in a thorough way. If they were in an interview, they would be able to tell me how Pod disruption budgets work, etc. I love this multilevel approach and offering that possibility to the learners to do it and to do it in a fun way. I engage well with video content, which is why I went down that route.

While I'm creating my own content, I run a lot of Kubernetes study groups. Study groups with thousands of members. And one of the key points that we get across with this group is that while I may be running these groups, I promote a lot of the great resources out there. For example, if somebody is doing something on the CKA side, I will often say, I've got a friend, Chad Crowell, who's done a really good book. Another friend of mine, Nigel Poulton, has a KCNA book. Kubernetes is that kind of topic where you're always learning new things. I created a course where we completely tear everything down then put it back together. But almost every day, I learn something new about Kubernetes or some kind of technique or little trick. As content creators,

we're not saying, hey, you know, just do ours; do everyone's and learn as much as you can, get that knowledge and that insight, because there might be one nugget that actually comes from this, which could be something defining you say in an interview, or something that you do at your work, which, you know, your managers appreciate. The more you engage yourself in it, the more opportunities there are.

Adrián: Yes, yeah, 100%. Well, James, this is amazing. Thank you very much for your time. I know that the readers will appreciate what you have shared here.

James: OK, brilliant. Thank you very much. Cheers.

Summary

This chapter has been a long journey exploring the fundamental concepts of Kubernetes, demystifying its architecture and components. From the basic building blocks to more sophisticated components like service mesh, we hope you now have a good understanding of what cloud native is, the relationship with the Cloud Native Computing Foundation and how Kubernetes plays an essential role moving the industry toward an interoperable system and a foundation of open standards that will accelerate technology adoption and bring new waves of innovation.

CHAPTER 7

Final Recommendations

You are approaching the final step in getting your KCNA certification: passing the exam. Whatever the reason that brought you here, the cloud native ecosystem is an endless source of learning. Having a good set of reference material to help you learn key concepts and guide you through practical examples is not an option but a requirement.

This chapter serves as a reference guide in the journey toward becoming a Kubernetes and Cloud Native Associate. It includes important material to enable you to not only excel in the certification exam but also 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 to understand 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. Here's a breakdown of the exam content:

  Kubernetes fundamentals
  : 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 the pool of resources and insights available during your preparation for the 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, planting the seeds to master the technology and provide solutions to real and complex challenges. The community experts are everywhere—blogs, X (formerly Twitter), YouTube, Twitch, etc.—making it easy to grab some serious knowledge on the fly.

The community experts we present in this section bring 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.

We recommend following these experts and community influencers:

Kelsey Hightower
: An influential advocate for Kubernetes, Kelsey is renowned for making complex Kubernetes concepts easy to grasp. He shares practical advice and step-by-step demonstrations in his X (formerly Twitter) and GitHub repositories. His ability to simplify and demystify Kubernetes makes him a go-to resource for both beginners and experienced practitioners. Figure 7-1 shows his GitHub repository.

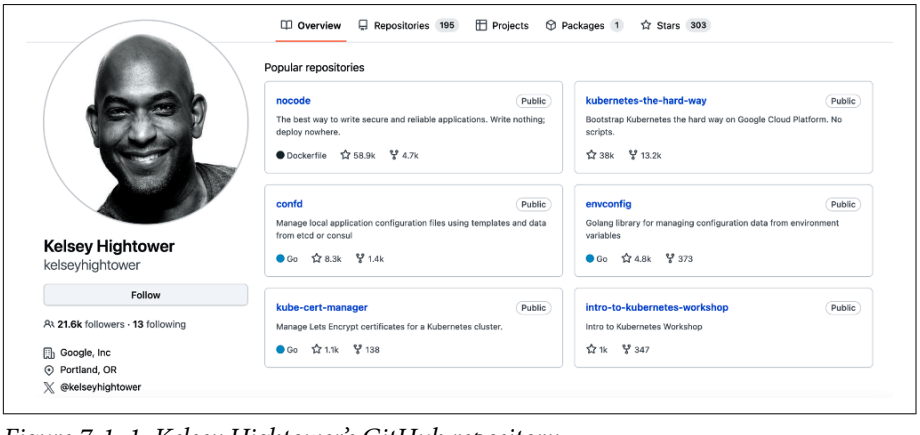
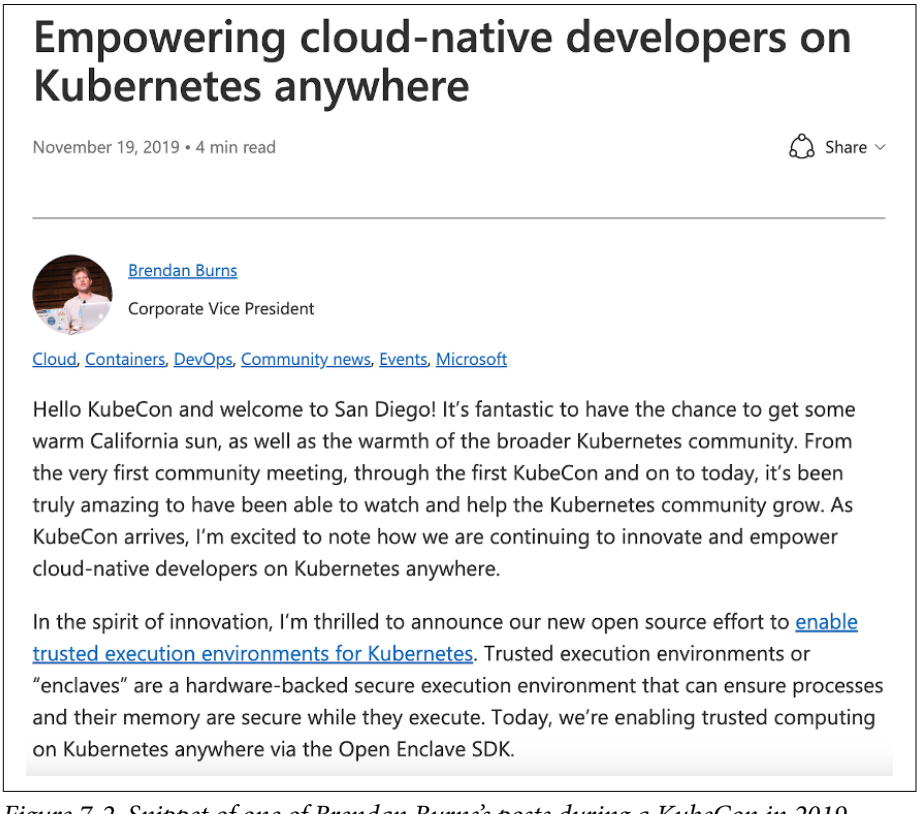


Figure 7-1. 1. Kelsey Hightower's GitHub repository

We strongly recommend forking the *kubernetes-the-hard-way* repository and building a Kubernetes cluster from scratch with the step-by-step guide inside the repository. This is without a doubt one of the best possible paths to practically understanding Kubernetes and its components.

Brendan Burns

As one of the brains behind Kubernetes, Brendan has a unique depth of knowledge about the system. He offers rich insights into the inner workings and future trajectory of Kubernetes. His blog posts (such as the one shown in Figure 7-2) and conference presentations are a goldmine for anyone seeking to understand Kubernetes at a deep structural level and get a glimpse into its evolving landscape.



Empowering cloud-native developers on Kubernetes anywhere

November 19, 2019 • 4 min read

Share

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 Brendan Burns's posts during a KubeCon in 2019

We highly recommend listening to this podcast, where Brendan Burns talks about the new long-term support strategy for Kubernetes.

Brendan's GitHub repository (Figure 7-3) is an invaluable source of practical guidance through labs that will help you see the complex technology behind the paradigm of distributed systems working in the real world.

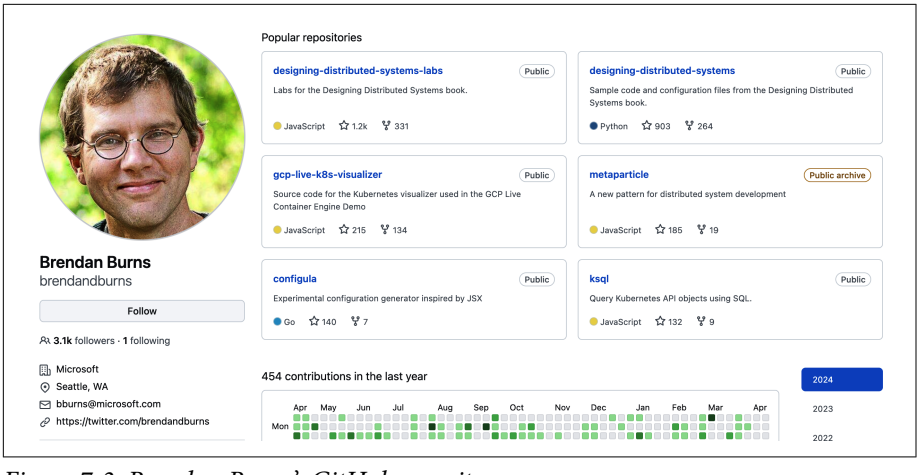


Figure 7-3. Brendan Burns's GitHub repository

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.

We recommend this interview with Joe Beda (Figure 7-4) talking about the community and how it was key for Kubernetes adoption.

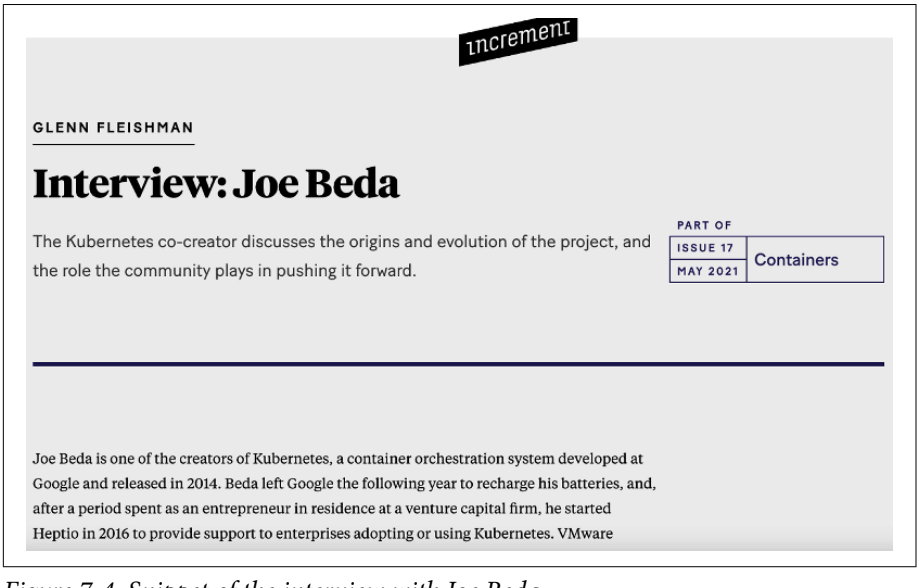


Figure 7-4. Snippet of the interview with Joe Beda

Saad Ali

Saad 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.

We recommend this CSI: Storage podcast (Figure 7-5) to learn more about Kubernetes storage.

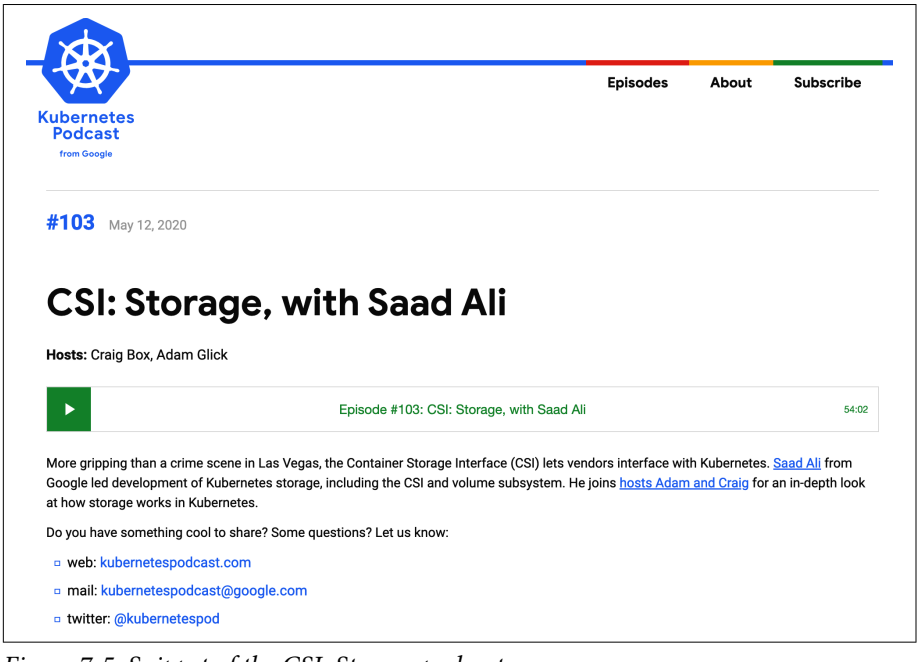


Figure 7-5. Snippet of the CSI: Storage podcast

Liz Rice

A leading voice in container and Kubernetes security, Liz's expertise is crucial for understanding how to effectively secure cloud native environments. Through her talks and writings, she provides essential insights into security best practices, making complex security concepts more accessible and understandable.

Her web page (Figure 7-6) contains links to all the content created by her, with special focus on eBPF and Kubernetes security content.

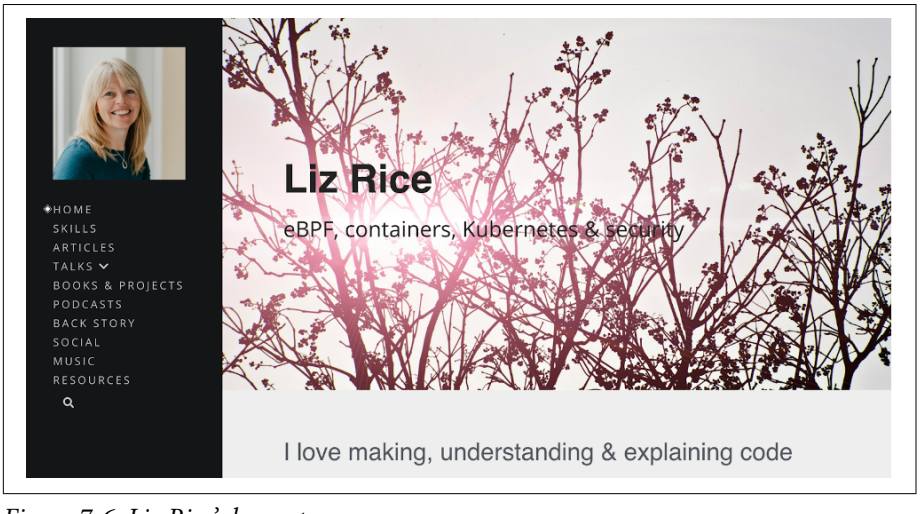


Figure 7-6. Liz Rice's home page

Tania Allard

As a senior developer advocate, Tania 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. See Tania's website for more info (Figure 7-7).

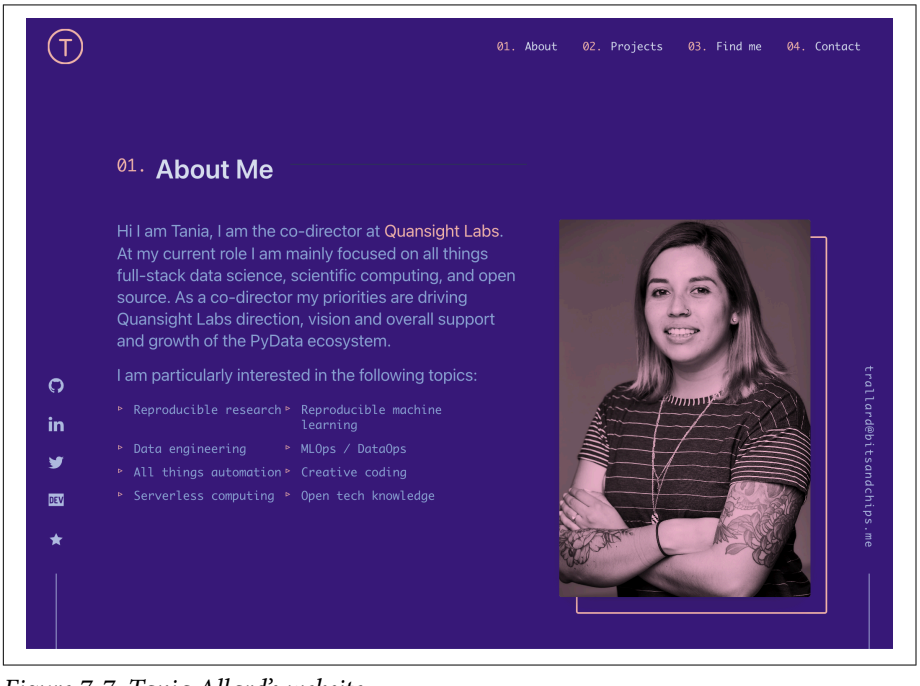


Figure 7-7. Tania Allard's website

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. Take a look at his GitHub repository (Figure 7-8) for further detail.

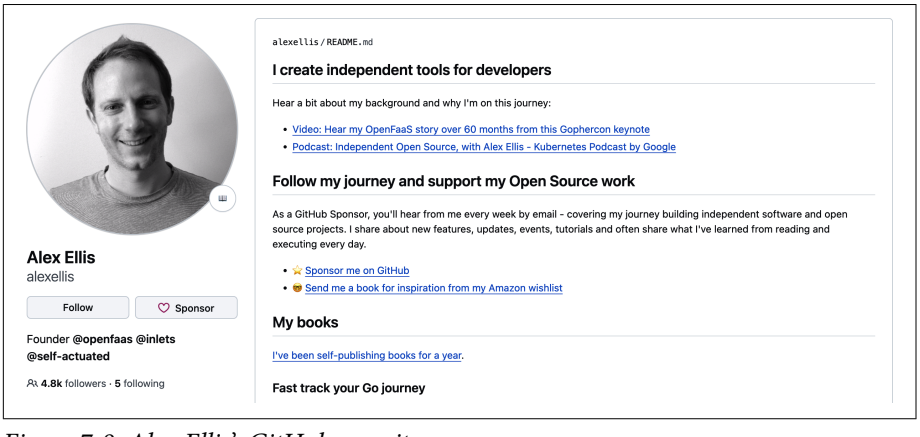


Figure 7-8. Alex Ellis's GitHub repository

James Spurin

James 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. See more info on his courses on DiveInto (Figure 7-9).

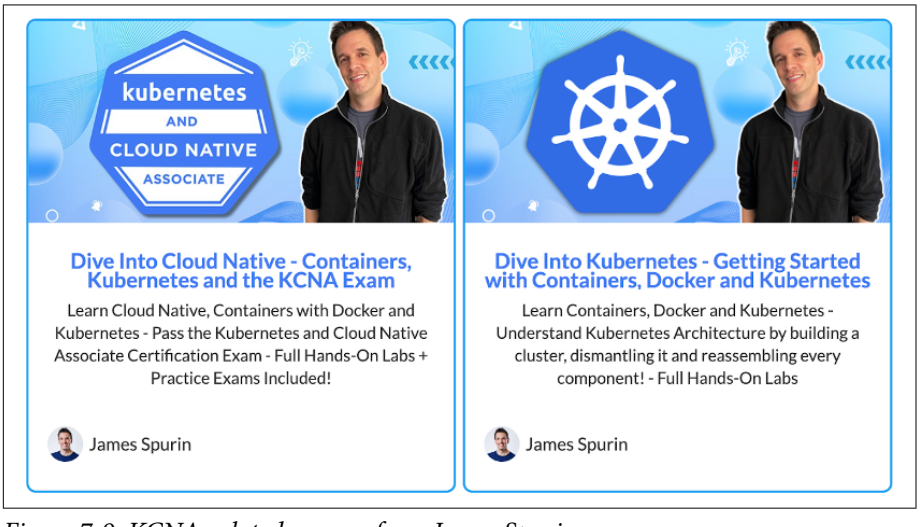


Figure 7-9. KCNA-related courses from James Spurin

KCNA-Related Communities

When preparing for the 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
: 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.

Kubernetes Slack channels
: These channels are great for real-time communication with Kubernetes professionals and enthusiasts. They're a platform for asking questions, sharing experiences, and getting tips and advice from community members.

GitHub Kubernetes community
: 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 (Figure 7-10).

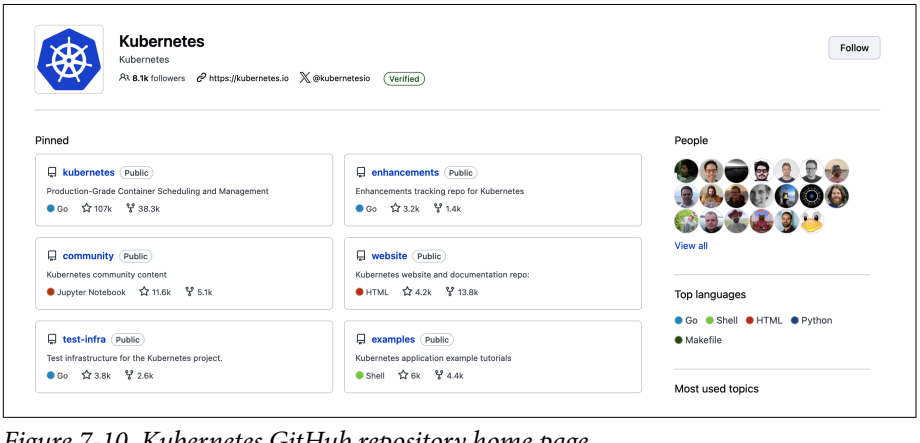


Figure 7-10. Kubernetes GitHub repository home page

Kubernetes forums

The Kubernetes forums (Figure 7-11) are an official place for discussions about Kubernetes. The community here is welcoming to questions from beginners, and it is a good place for detailed discussions.

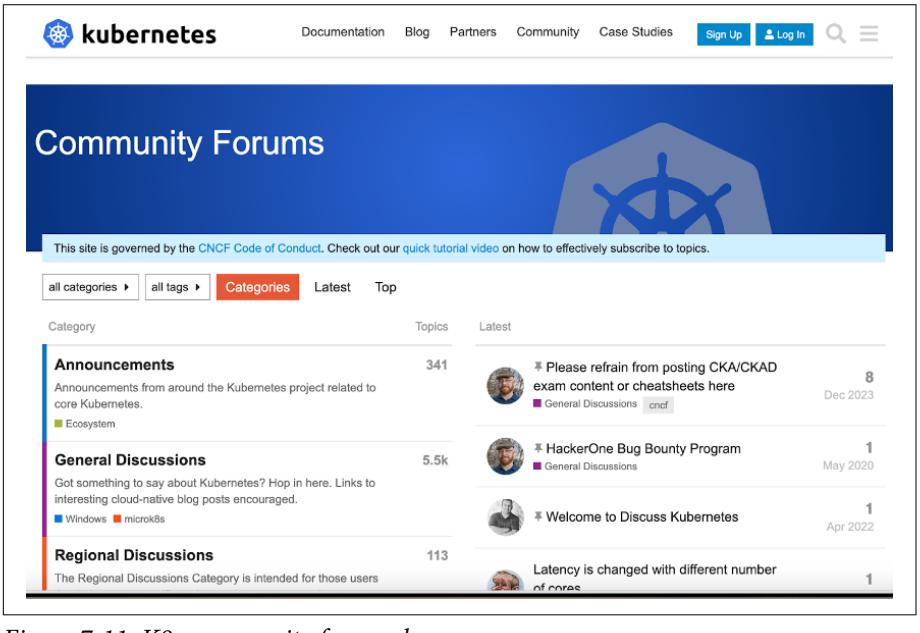


Figure 7-11. K8s community forums home page

Reddit Kubernetes community

The Kubernetes subreddit (Figure 7-12) is a place for news, articles, and discussions related to Kubernetes. It's a more informal setting where members share news and tips and ask for advice.

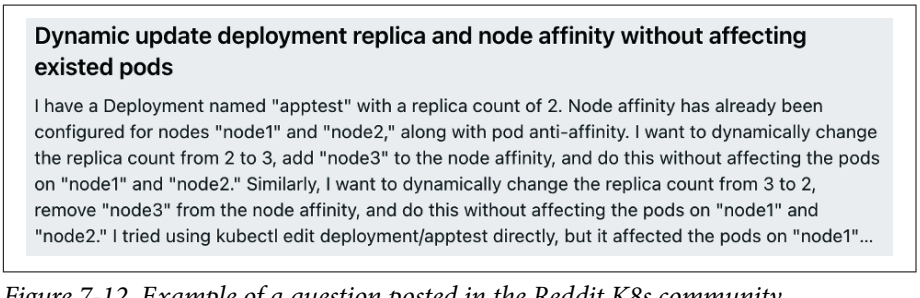


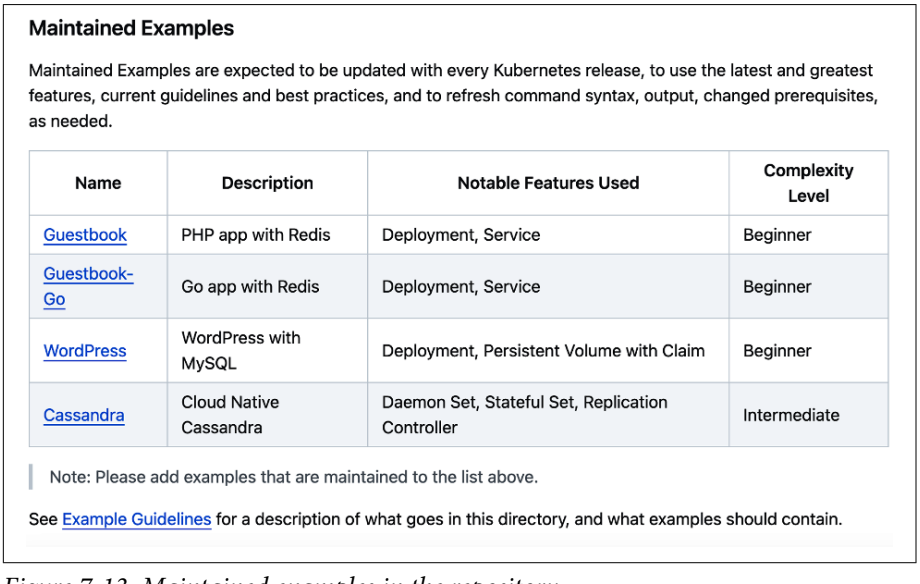
Figure 7-12. Example of a question posted in the Reddit K8s community

Useful GitHub Repositories

For students preparing for the 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
: 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
: This repository (Figure 7-13) 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 7-13. Maintained examples in the repository

kubernetes-sigs

The Kubernetes SIGs have their repositories in this spot on GitHub (Figure 7-14). These repositories contain projects and tools related to specific areas within Kubernetes.

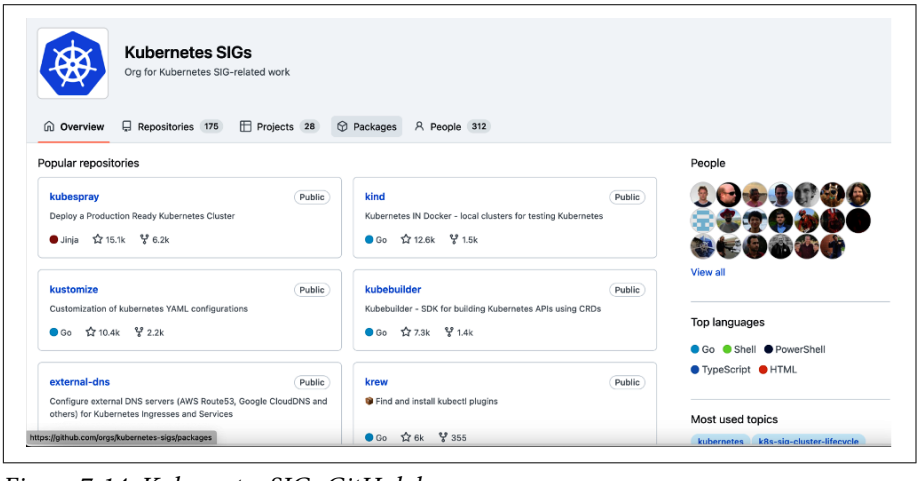


Figure 7-14. Kubernetes SIGs GitHub home page

Artifact Hub

A web-based platform for discovering, installing, and sharing packages and configurations related to CNCF projects (Figure 7-15). These offerings encompass a wide range of items, including Helm charts, Falco configurations, Open Policy Agent (OPA) and Gatekeeper policies, Operator Lifecycle Manager (OLM) operators, Tinkerbell actions, `kubectl` plug-ins, Tekton tasks and pipelines, KEDA scalers, CoreDNS plug-ins, Keptn integrations, container images, Kubewarden policies, Kyverno policies, the Knative client, Backstage plug-ins, Argo templates, KubeArmor policies, Kubernetes Client Library (KCL) modules, and the Headlamp plug-in.

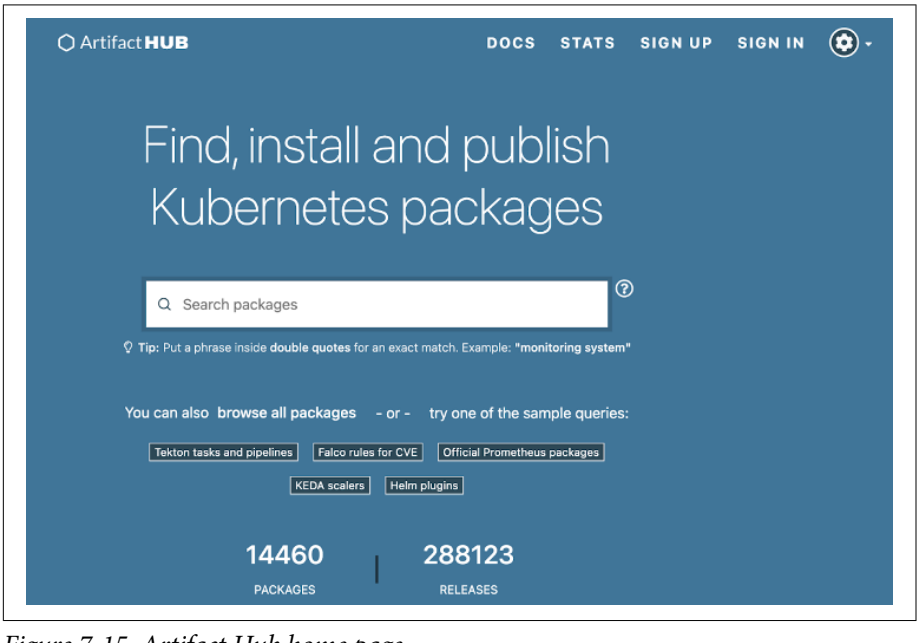


Figure 7-15. Artifact Hub home page

kubernetes/minikube

minikube is an essential tool for local Kubernetes development and testing (Figure 7-16). This repository contains the source code and documentation for setting up a local Kubernetes cluster.

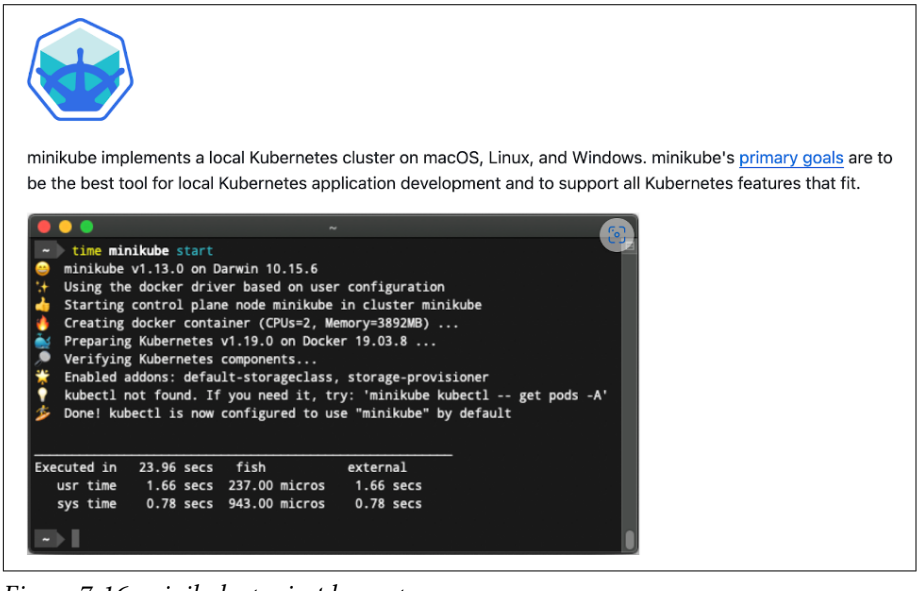


Figure 7-16. minikube project home page

kelseyhightower/kubernetes-the-hard-way

Created by Kelsey Hightower (discussed earlier), 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 how to deploy every building block and stitch them together.

cncf/landscape
: The CNCF's landscape (Figure 7-17) offers a comprehensive overview of the cloud native landscape, including tools, services, and technologies associated with Kubernetes.

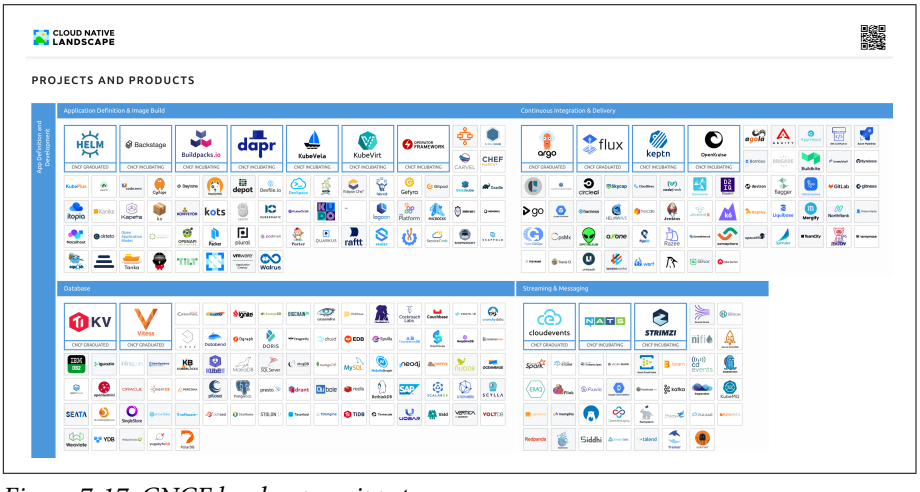


Figure 7-17. CNCF landscape snippet

Potential Career Options with KCNA Certification

Earning a 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
: This role ensures high availability and reliability of services. Site reliability engineers (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 focuses 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 KCNA certification, several other Kubernetes certifications are offered by 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 to 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 Kubernetes 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 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 Brendan Burns, Joe Beda, Kelsey Hightower, and Lachlan Evenson (Figure 7-18)

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.

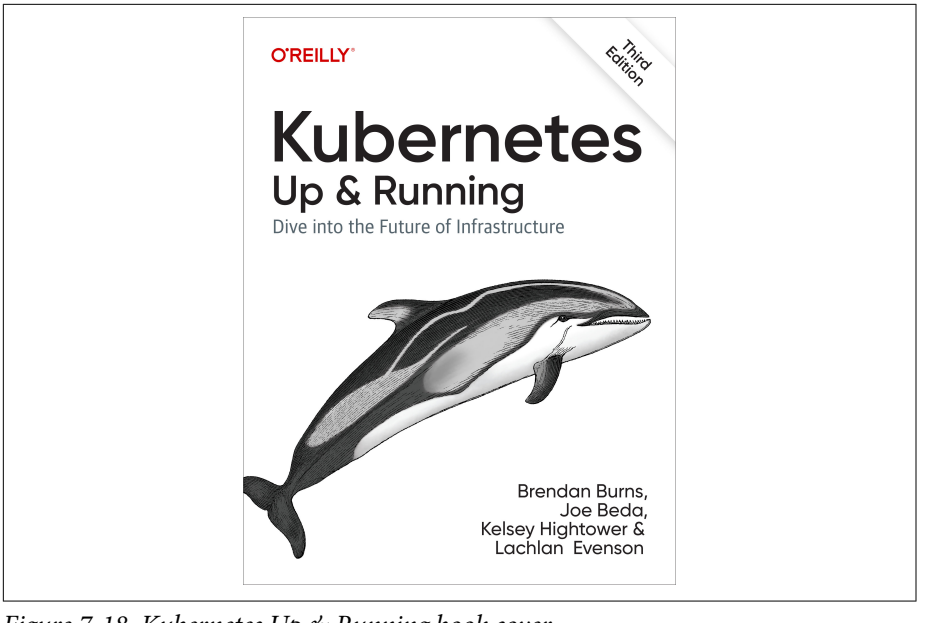


Figure 7-18. Kubernetes Up & Running book cover

Designing Distributed Systems: Patterns and Paradigms for Scalable, Reliable Services by Brendan Burns (Figure 7-19)

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.

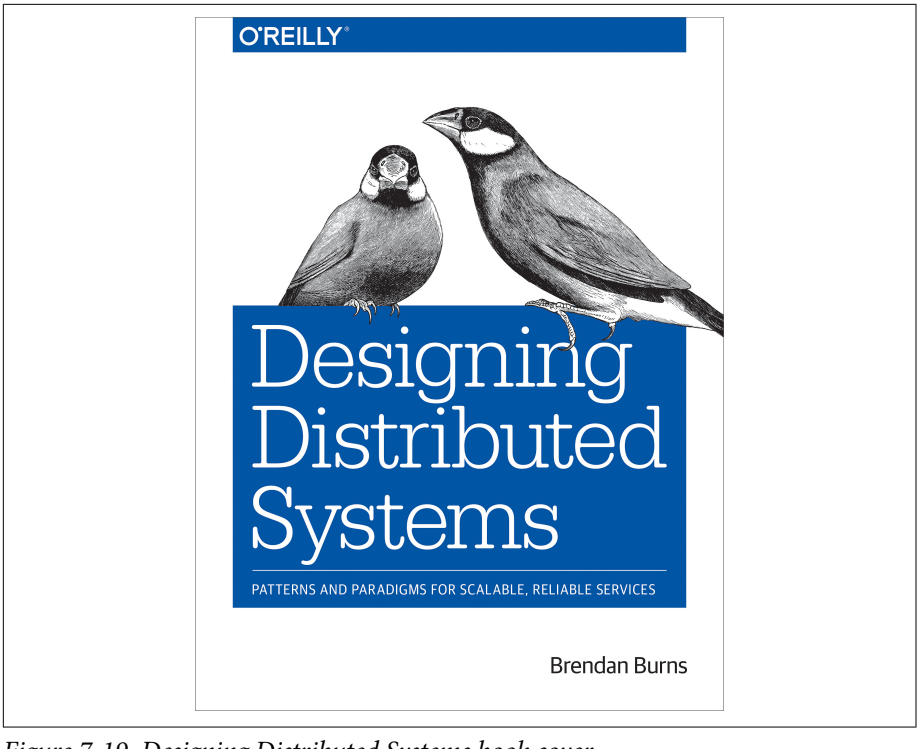


Figure 7-19. Designing Distributed Systems book cover

Kubernetes Patterns: Reusable Elements for Designing Cloud-Native Applications *by Bilgin Ibryam and Roland Huß (Figure 7-20)*

This book delves into Kubernetes-specific patterns for application development, offering a detailed guide for architecting cloud-native applications effectively.

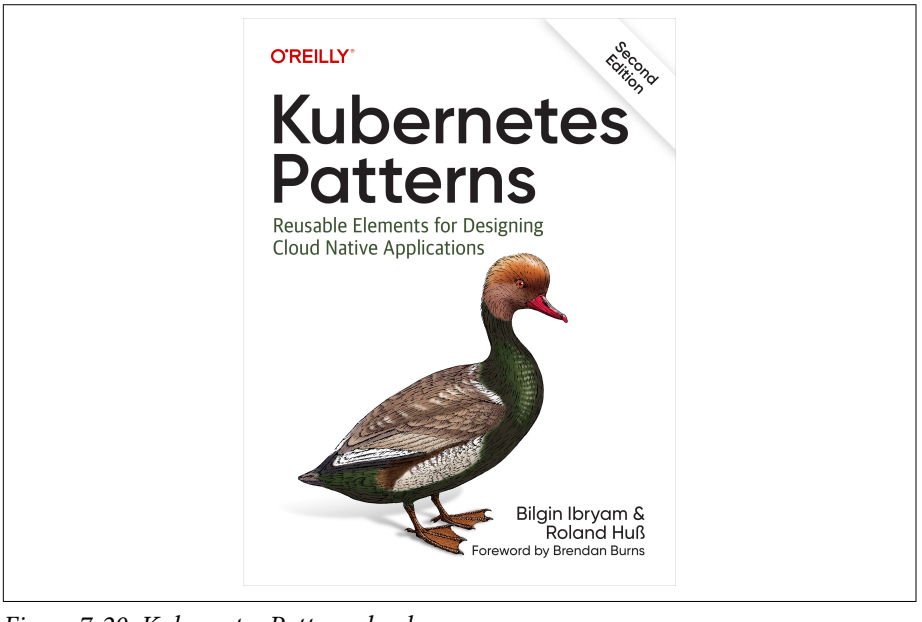


Figure 7-20. Kubernetes Patterns book cover

Cloud Native DevOps with Kubernetes: Building, Deploying, and Scaling Modern Applications in the Cloud *by Justin Domingus and John Arundel (Figure 7-21)*

Covering the DevOps side of Kubernetes, this book addresses continuous integration, continuous delivery, and infrastructure as code in Kubernetes environments.

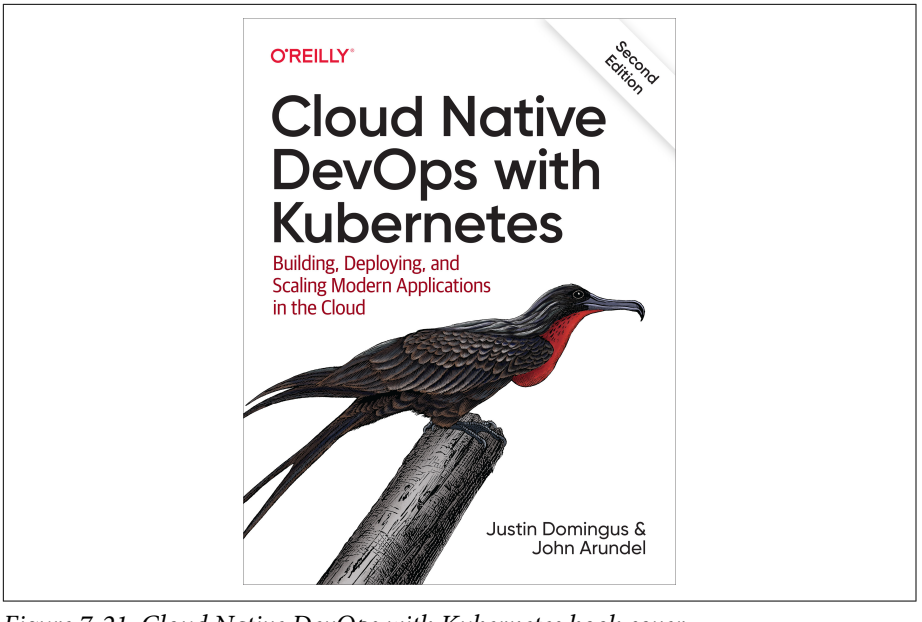


Figure 7-21. Cloud Native DevOps with Kubernetes book cover

Managing Kubernetes: Operating Kubernetes Clusters in the Real World by Brendan Burns and Craig Tracey (Figure 7-22)

Targeted at Kubernetes administrators, this book discusses the challenges and solutions of managing Kubernetes in production settings.

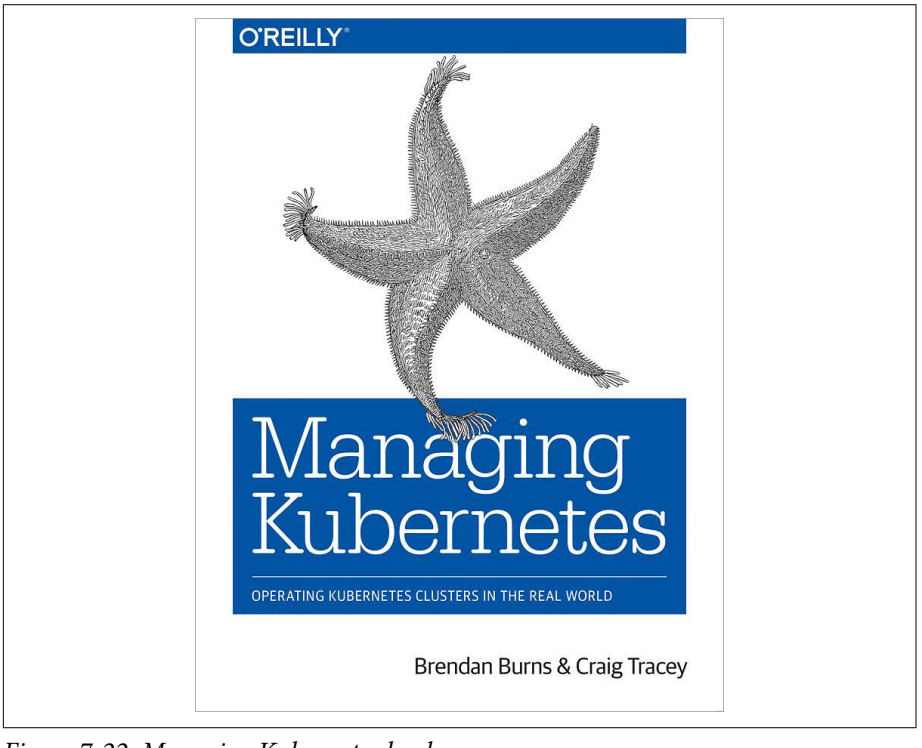


Figure 7-22. Managing Kubernetes book cover

Container Security by Liz Rice (*Figure 7-23*)
: This book is essential for those focusing on Kubernetes security. It discusses best practices and strategies for securing containers and clusters.

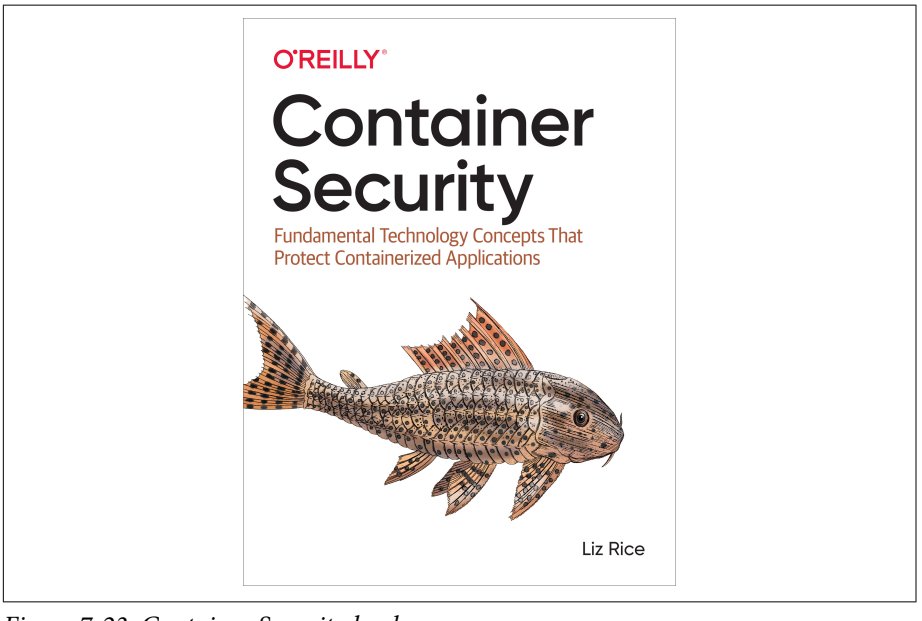


Figure 7-23. Container Security book cover

Summary

Congratulations! You have completed the Study Guide for the KCNA certification. Remember, this isn't just about passing an exam; it's about mastering a transformative technology. As 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!

APPENDIX A
End-of-Book Knowledge Check

These true-or-false questions are just a quick knowledge check for you to confirm your understanding of some basic topics related to the KCNA exam. Solutions can be found in Appendix B.

1. True or false: People who are Kubernetes professionals should not be taking the KCNA exam.
2. True or false: It is free to take the KCNA exam.
3. True or false: There are existing certification pre-requirements such as the KCSA.
4. True or false: You can take the exam only once.
5. True or false: The exam can be taken in both physical exam centers and online.
6. True or false: CNCF is responsible for defining and promoting hardware standards.
7. True or false: Envoy is a high-performance proxy and a CNCF-hosted project that can be used in service mesh architectures.
8. True or false: CNCF is a subsidiary, or subproject, of the Linux Foundation.
9. True or false: Fluentd is a data collector used for logging purposes in cloud native applications.
10. True or false: SPIFFE and SPIRE are projects under CNCF focused on container runtime interfaces.

APPENDIX B

Solutions to Self-Questionnaire and Knowledge Checks

This appendix lists the solutions to the self-questionnaire in Chapter 1 and the knowledge checks in Chapters 2 and 3 and Appendix A.

Self-Questionnaire Part 1: The CNCF Ecosystem

1. b
2. c
3. c
4. b
5. c
6. c
7. b

Self-Questionnaire Part 2: General Cloud Native Concepts

1. b
2. b
3. c
4. a
5. b
6. b

7. b
8. c

Self-Questionnaire Part 3: Kubernetes Topics

1. a
2. d
3. c
4. b
5. b
6. c
7. c
8. c

Self-Questionnaire Part 4: Kubernetes Commands

1. c
2. a
3. b
4. a
5. b

Self-Questionnaire Part 5: Linux Fundamentals

1. d
2. c
3. b
4. c
5. c
6. d
7. c
8. c

Self-Questionnaire Part 6: Other Related Projects

1. b
2. c
3. a
4. b
5. d
6. b
7. a
8. c
9. c

Chapter 2 Knowledge Check

1. True
2. False
3. False
4. True
5. False

Chapter 3 Knowledge Check

1. False
2. True
3. False
4. True
5. True

End-of-Book Knowledge Check

1. False
2. False
3. False
4. False
5. False
6. False
7. True
8. True
9. True
10. False

About the Authors

Adrián González Sánchez is a cloud, data, and AI specialist at Microsoft as well as the industrial AI lead for the Spanish Observatory of Ethical AI (OdiseIA). He has previously worked as a senior consultant and product owner in AI and data science with CGI Canada and IVADO Labs, as head of AI customer success for Peritus.ai, and at other data-driven companies in Europe and Latin America.

Jorge Valenzuela Jiménez is director of the Cloud Solution Architects team at Microsoft Spain, driving innovation and enabling organizations to harness the full potential of cloud technologies. With a telecommunications engineering degree from UC3M and a master's in artificial intelligence from UNIR, he brings a wealth of knowledge and expertise to his roles, as well as a deep passion for the open source community and its transformative influence on software development. Since 2013, Jorge has been at the forefront of cloud native technologies, actively working with cutting-edge solutions.

Colophon

The animal on the cover of *Kubernetes and Cloud Native Associate (KCNA) Study Guide* is a Mediterranean flying fish (*Cheilopogon heterurus*). This animal is a pelagic coastal species that was first described in 1810 by Constantine Samuel Rafinesque, who was an American naturalist, zoologist, and botanist.

The Mediterranean flying fish can be found in oceans, specifically in tropical and warm subtropical waters. They usually reside anywhere from the top layer of the ocean to a depth of about 200 meters, or 656 feet. The fish typically enters the Adriatic Sea from the Mediterranean Sea during the late summer to early autumn and rarely swims upstream to the northern part.

The Mediterranean flying fish can fly long distances, which is believed to be a behavior learned as a way to escape predators and other danger; however, it takes significant effort for the fish to actually fly. It thrusts from its long tail to help become more airborne. This motion, along with strong wind, can aid the fish in taking to the skies, gliding for hundreds of meters. The fish will then hit the water's surface with its tail to leap and keep elevated.

The Mediterranean flying fish's diet mainly consists of zooplankton, but they sometimes eat small crustaceans as well. The animal is typically 15 centimeters in length, but some male Mediterranean flying fish can grow to be 40 centimeters. What is most striking about this creature is its iridescent blue and silver color.

It can be hard for Mediterranean flying fish to protect their young in the oceans, so the female fish usually lay their eggs on a palm frond leaf. Afterward, the male fertilizes the egg. After a fish spawns, it attracts hundreds of other Mediterranean

flying fish to lay their eggs there, too, and the palm frond will sink to the ocean floor because of the weight. After a few days, the young hatch.

Flying fish populations are stable. The Mediterranean flying fish is not commercially exploited, and its conservation status is considered "Least Concern." Many of the animals on O'Reilly covers are endangered; all of them are important to the world.

The cover illustration is by Karen Montgomery. The series design is by Edie Freedman, Ellie Volckhausen, and Karen Montgomery. The cover fonts are Gilroy Semibold and Guardian Sans. The text font is Adobe Minion Pro; the heading font is Adobe Myriad Condensed; and the code font is Dalton Maag's Ubuntu Mono.

O'REILLY®

Learn from experts.
Become one yourself.

Books | Live online courses
Instant answers | Virtual events
Videos | Interactive learning

Get started at oreilly.com.

©2023 O'Reilly Media, Inc. O'Reilly is a registered trademark of O'Reilly Media, Inc. 175 7x9.1975