Here’s What’s Actually Going On with Kubernetes

The complete Kubernetes guide in plain language - Free sample (Chapter 1)



Luis Felipe Ariza Vesga

Here's What's Actually Going On with Kubernetes

Here's What's Actually Going On with Kubernetes

The complete Kubernetes guide in plain language - Free sample (Chapter 1)

Luis Felipe Ariza Vesga

Here's What's Actually Going On with Kubernetes
The complete Kubernetes guide in plain language - Free sample (*Chapter 1*)

Copyright © 2026 Luis Felipe Ariza Vesga.
All rights reserved.

The information in this book is distributed on an "as is" basis, without warranty. While every precaution has been taken in the preparation of this work, neither the author nor the publisher shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in this book.

ISBN: 9798173447777

A tribute to the Colombian grebe (Podiceps andinus), which lived in the wetlands of the Bogota savanna and Lake Tota, and became extinct in 1977 with nobody watching.

Contents

Conventions Used in This Book **1**
Typography and References 1
The Four Callout Boxes 2
About the Labs 3
Third-party credits and licenses 3

Preface **5**
What This Book Is About 5
What This Book Is Not 6
A Series That Pays Tribute 7
Navigating This Book 7
Using Code Examples 8
How to Contact Us 9
Acknowledgments 10

I Kubernetes Fundamentals **11**

1 Containers: Where They Came From and What They Solve **13**
1.1 It Works on My Machine 13
1.2 The Virtual Machine: Shipping the Whole House 14
1.3 What a Container Actually Is: Namespaces and cgroups 15
1.4 The Open Container Initiative (OCI): The Day Everybody Agreed . . 17
1.5 So Where Does Kubernetes Fit In? The Container Runtime Interface (CRI) 17
1.6 Summary 18
Bibliography 19

Conventions Used in This Book

How this book is written: the tone, the typography, the four callout boxes and what the labs are.

Before we start, half a page on how this book is written. It reads in two minutes and saves confusion later.

This book addresses you respectfully. If you are from somewhere else and it reads oddly, take it as a form of address that shakes your hand while it speaks.

Every chapter opens with an everyday scene. That scene is not decoration or padding: it **is** the technical concept told another way, and it will come back at the end of the chapter, in the summary, explained with the proper words. If at any point a technical explanation does not land, go back to the opening scene: the answer is almost always sitting there in simpler form.

You will also find humour. The humour serves a concrete purpose, which is to lower the book's technical difficulty. It never replaces rigour: after the joke always comes the official definition, verbatim, with its link to the source.

Typography and References

Monospaced type is used for anything you would type or read on a screen: object names such as `Pod` and `Deployment`, YAML fields such as `restartPolicy`, commands such as `kubectl describe pod`, and paths such as `/api/v1`.

Bold marks the first appearance of an important term, and also the words the exam tends to use verbatim. When you see a bold word inside a definition, it is because that exact word matters.

Angle brackets mark what you must replace: in `kubectl logs <pod-name>`, you type the real name and you do not type the brackets.

Quotations from the official documentation appear woven into the text rather than as separate blocks, and they always carry a link to the source the first time they are used in a chapter. That is intentional: you should be able to verify any claim in this book without searching for it, by clicking.

Every chapter closes with a **bibliography** of at least three sources, all official and all with links.

Figures are referenced in the text and carry an explanatory caption. If a figure sits near an explanation, it is because that explanation reads better alongside it.

The Four Callout Boxes

Throughout the book you will meet four kinds of box. Each means something different, and it pays to recognize them at a glance because they do not all carry the same urgency.

> **Note**
> Complementary information expanding on what you just read. It can be skipped without losing the thread, but it usually holds a technical detail worth knowing. A real example from this book: kube-proxy is optional if your network plugin implements Service forwarding on its own.

> **Tip**
> A shortcut, a mnemonic or a faster way of doing something. This is where the sentences worth memorizing live, such as "liveness kills, readiness disconnects".

> **Warning**
> A frequent mistake, a common confusion or a behaviour that surprises people. If you are only going to read one kind of box before sitting the exam, make it these. Example: CrashLoopBackOff is not a Pod phase.

> **Caution**
> Something that can cause real damage on a real cluster: data loss, an outage or a security hole. These are few and they are serious. Example: under dynamic provisioning the default reclaim policy is Delete, so deleting a PVC (PersistentVolumeClaim) deletes the disk.

In short: **note** expands, **tip** helps, **warning** corrects and **caution** protects.

About the Labs

Each of the four parts closes with a **lab**, and it is worth saying now what they are and are not.

They are not copy-and-paste exercises. Every lab has an objective, steps worth points, and checks that read the real state of the cluster. You do not pass a step because you ran a command: you pass because the cluster ended up as it should. If you reach the same result by another route, the lab accepts it just the same, because what is being assessed is the end state and not your command history.

Every command in the labs was executed on a real cluster before publication, and the versions are pinned. That means if something does not work for you, the likely cause is an environment difference worth understanding, rather than the book making things up.

An important note: KCNA is a 90 minute multiple choice exam. You will not be sat in front of a terminal. So the labs are **not** muscle-memory training for the exam: they are how you turn each claim in this book into something you saw with your own eyes and that can be useful for the Kubernetes practical exams. Believing that a node without a network plugin stays NotReady is different from having watched it happen on your machine.

And on where to run them: never on a production cluster.

Third-party credits and licenses

This book reproduces and adapts excerpts from the official Kubernetes documentation (https://kubernetes.io/docs/), published by The Kubernetes Authors under the

Creative Commons Attribution 4.0 International (CC BY 4.0) license. The license text is available at https://creativecommons.org/licenses/by/4.0/.

Definitions quoted verbatim appear in quotation marks and link to the page they come from. Any adaptation or paraphrase of that text is this book's own and has not been reviewed or endorsed by The Kubernetes Authors.

Kubernetes is a registered trademark of The Linux Foundation. The names of the certifications cited belong to the Linux Foundation and the Cloud Native Computing Foundation. This book is independent study material, not affiliated with, endorsed by, or sponsored by those organizations.

Preface

Why this book exists, what it covers, what it does not, and how to read it.

This book was born out of an irritation.

The irritation is that almost all Kubernetes learning material is written as though the reader already knew Kubernetes. It opens with a diagram of seven boxes joined by arrows, states that "the kubelet communicates with the apiserver via the CRI" (an assertion that, incidentally, is false: the kubelet talks to the apiserver over HTTPS/REST; the CRI, the Container Runtime Interface, is the kubelet interface to the container runtime, not to the apiserver), and moves right along without correcting it. Whoever already understood nods without noticing the error. Whoever did not feels stupid and closes the tab.

They are not stupid. Nobody told them the whole story.

This book tells the story. It starts in a borrowed kitchen and ends with a bird that went extinct because nobody was watching, and in between sit the four KCNA exam domains with their official weights. It speaks plainly, without a tie and without dumbing anything down.

Because that is the underlying bet: that clarity and rigour do not fight. That a well placed joke does not make an explanation less serious, it opens the door to it. And that once through that door you can put the official definition, verbatim, with its link to the source.

What This Book Is About

This volume covers the **KCNA** certification, Kubernetes and Cloud Native Associate, from the Cloud Native Computing Foundation and the Linux Foundation.

It is organized exactly on the four official exam domains, with their real weights, taken from the curriculum published by CNCF rather than from third-party summaries:

Part 1, Kubernetes Fundamentals, 44%. What a container actually is, how a cluster is put together, the objects you use every day, how you talk to the API (Application Programming Interface), and how the scheduler decides where each Pod goes.

Part 2, Container Orchestration, 28%. The network model and who implements it, storage that outlives the pod, orchestrator security, and troubleshooting when nothing starts.

Part 3, Cloud Native Application Delivery, 16%. GitOps with its four formal principles, Helm, the three CI and CD acronyms, and debugging applications that are already running.

Part 4, Cloud Native Architecture, 12%. The official cloud native definition, horizontal and vertical scaling, CNCF and its maturity levels, and observability.

One important warning about those percentages: the KCNA curriculum **changed as of 24 November 2025**. It went from five domains to four, and observability was folded into Cloud Native Architecture. Any material you find with five domains and a standalone observability domain is describing the **previous** exam.

What This Book Is Not

This book is **not** a command reference. You will not find every `kubectl` option here, because the official documentation exists for that and because memorizing commands is not what KCNA evaluates.

It is **not** a book on running clusters in production. Installing a cluster with kubeadm, upgrading versions and operating etcd are not covered. That is the CKA exam territory.

It is **not** a book on application development. Microservice patterns and how to write your application are not covered. That points at CKAD exam.

It is **not** an exam simulator. There are practice questions in Appendix C, but this book was written so that you **understand**, not so that you recognize answers. If what you

want is question banks, this is not the book, and I say so up front rather than spend your money.

And it is **not** the security volume. KCSA exam has its own volume, and everything touched on here regarding security (RBAC (Role-Based Access Control), ServiceAccounts, Pod Security Standards) sits at the level KCNA exam requires, not the level KCSA exam requires.

Nor is it a book about humour, even though it talks that way. The tone is a teaching tool, not the subject.

A Series That Pays Tribute

Every book in the **Here's What's Actually Going On** series carries on its cover an animal driven extinct by human activity, with a short dedication. It is not decoration: it is a tribute, and a memory that the animal once lived here with us. This volume belongs to the Colombian grebe, a bird of the wetlands of the Bogota savanna and Lake Tota.

This book is the first step on a longer road. According to the official Kubestronaut program page, the CNCF (Cloud Native Computing Foundation) awards the title of **Kubestronaut** to anyone who passes all of its Kubernetes certifications, currently five: KCNA, KCSA, CKA, CKAD and CKS. This volume covers the KCNA exam. Three more books in the series follow: **Here's What's Actually Going On with Kubernetes Administration** (CKA), **Here's What's Actually Going On with Kubernetes Application Development** (CKAD) and **Here's What's Actually Going On with Kubernetes Security** (CKS).

Along that road, the series' tributes pass through animals of the air, the water and the land. So to become a Kubestronaut you do not only learn to administer a cluster, to develop applications on it and to secure it: you also travel, book by book, through the memory of species of the air, the water and the land that are no longer with us.

Navigating This Book

The thirteen chapters are meant to be read in order, because each leans on the one before. The container from Chapter 1 returns as a Pod in Chapter 3, as a security

context in Chapter 8 and as an ephemeral container in Chapter 11. If you skip, you will hit references to things you did not read.

That said, the structure lets you attack in parts: each of the four parts corresponds to an exam domain, so if you already work with Kubernetes and want to reinforce one domain only, you can go straight to that part.

Every chapter has the same shape. It opens with a scene, almost always an everyday situation, and that scene **is** the technical concept, not an ornament. Then comes the explanation, grounded in official documentation with links to the source at first use. Along the way there are figures and callout boxes. And it closes with a summary that returns to the opening scene and explains it with the vocabulary you have now learned.

The summaries are written to be reread on their own, the day before the exam, without going back through the whole chapter.

At the end of the volume there are three appendices: a **glossary** of the terms the exam uses, a **cheatsheet** with what gets asked most, and a **FAQ** covering exam logistics and the most common conceptual traps.

And every part has a **lab**. The labs are neither optional nor decorative: they are where you confirm on a real cluster that what you read is true.

Using Code Examples

The code examples in this book are Kubernetes YAML fragments and commands for `kubectl`, `helm` and system utilities.

They appear in monospaced type, like this: `kubectl get pods`. When a command contains parts you must replace, they appear in angle brackets, as in `kubectl logs <pod-name>`. Do not type the brackets.

Every command in the labs was executed on a real cluster before publication. There are no commands written from memory and no "this should work": if a command is in a lab, somebody ran it and watched its output.

Versions matter in Kubernetes. The labs pin the versions they use, and the text states the version whenever a behaviour depends on it, as with the CRI v1 API requirement

since Kubernetes 1.26. If you run a different version something may change, which is why the official documentation is always linked for you.

All the lab code is published at https://github.com/lfarizav/lo-que-pasa-es-que-kubernetes. There you will find the complete YAML manifests, every lab's commands, and the exact versions they were tested against, ready to clone and run in a Linux terminal:

```
git clone https://github.com/lfarizav/lo-que-pasa-es-qu
 ↪  e-kubernetes.git
cd lo-que-pasa-es-que-kubernetes
```

If a command from the book does not work for you, check that repository first: when the official documentation changes, the correction lands there before it reaches the next printed edition.

You may use the code in this book in your programs and documentation without asking permission, unless you are reproducing a significant portion of it. Selling or distributing the examples does require permission.

And a practical warning: never run a command from a book, a blog or a video on a production cluster without understanding what it does. This book included.

How to Contact Us

This book is part of a catalogue of technical books written by Luis Felipe Ariza Vesga, and it is maintained continuously: when the exam curriculum changes, or when the official documentation changes a definition, the book is updated, we report to readers, and the new version of the book is given free.

If you find an error, have a question about an explanation, or think something was badly told, write. Error reports are welcome and are the fastest way to improve the material, especially the ones that say "I read this part three times and did not understand it", because that almost always means it is badly written rather than that you are slow.

Two things are of particular interest. First, whether any explanation has fallen out of date against the official documentation, with a link to the source that contradicts it.

Second, whether any lab command failed in your environment, with the Kubernetes and runtime versions you are using.

For the certification itself, the authoritative source is always the Linux Foundation and CNCF: they publish the curriculum, programme changes, exam logistics and retake policies. No book, this one included, replaces that source. Before sitting the exam, check the official exam page.

Acknowledgments

To the cloud native community, which wrote all the documentation this book leans on and left it open and free for anybody. Every fact in this volume came from there.

To the maintainers of Kubernetes, of the OCI (Open Container Initiative), of OpenTelemetry, of Helm and of OpenGitOps, who besides building the software took the trouble to version their definitions and publish their programme changes.

To the students who at some point raised a hand to say "I did not understand". That gesture, which costs something, is what makes material improve. A good share of the scenes in this book exist because somebody said the previous explanation had not helped them.

And finally to the Colombian grebe, which lived in the wetlands of the Bogota savanna and in Lake Tota, and whose last two or three individuals were seen in 1977. This book talks about that in its final chapter, and its cover remembers it.

PART I

Kubernetes Fundamentals

1

Containers: Where They Came From and What They Solve

Why the container exists, what it actually is on the inside, and why Kubernetes needs one in order to do its job.

1.1 It Works on My Machine

Picture this. You cook a spectacular stew at home. It comes out perfect, the kind people remember. You invite the family over, everyone is happy, right there in your grandmother's kitchen, where a framed picture of some odd bird has hung on the wall for years; she used to just say, short and serious, that you had to pay attention to the details. And then your aunt says: ”make me that same stew at my place on Sunday”.

You show up confident, same recipe, same hands, and it turns out her stove is a different brand, the pot is smaller, the burner runs hotter, the water tastes different, and she does not have half the herbs. Same cook. Same recipe. And it comes out terrible.

Now look: that is exactly what has been happening to software since software existed. You write a program, you test it on your computer, it runs perfectly. You ship it to the

company server and it falls over. And always, always, the same sentence shows up, delivered with the same innocent face: ”but it works on my machine”.

Here is what is actually going on. The problem was never the program. The problem was the kitchen. The library version, the environment variable you had set and forgot about, the file that lived on your disk and nowhere else, the operating system version. All of that is the kitchen. And you were handing over the recipe without handing over the kitchen.

A container is exactly the solution to that problem: pack the kitchen together with the recipe. That is all it is.

So when somebody tells you a container is ”a lightweight virtual machine”, you will already know that it is not. Not even close. It is a different thing, and the difference will show up on your exam, because containerization is one of the declared competencies in the official KCNA curriculum. In this chapter we are going to see what a container actually is, why it appeared, and why Kubernetes could not exist without it. Ready? Go for it, son, my grandma used to say.

1.2 The Virtual Machine: Shipping the Whole House

Before containers the industry had already tried to solve this, and the answer was called a virtual machine. The idea was reasonable: if the problem is that the kitchen over there is different from the kitchen over here, then let us ship the whole kitchen. A virtual machine packs a complete operating system, with its own kernel, its own drivers and its own boot sequence, and runs it on top of a hypervisor.

And it worked. It genuinely worked, and a good part of the cloud you use today was built on top of it. But it had a cost, and the cost was weight. Every virtual machine drags a complete operating system behind it: gigabytes of disk, hundreds of megabytes of memory just to exist, and a boot time measured in minutes, not seconds. If you want to run twenty small services, you end up with twenty complete operating systems fighting over the same physical machine. It is as if, to bring your aunt the stew, you brought the house.

Then somebody asked the right question. And the right question is almost always an uncomfortable one: do we actually need another kernel?

Because think about it. If the server over there and the computer over here both already run Linux, the kernel is already in place. What differs between the two machines is not the kernel: it is the libraries, the files, the versions, the environment variables. That is what needs packing. The kernel can be shared. It is already there.

And out of that uncomfortable question the container was born: share the host machine's kernel, but give each process the illusion that the system belongs to it alone. It is not a lightweight virtual machine. It is a completely different thing, built on a different idea.

1.3 What a Container Actually Is: Namespaces and cgroups

Now look, this is where a lot of people get lost, and where the exam can lay a trap for you. A container is not an object. There is no thing in Linux called a "container" that you can point a finger at. A container is an ordinary process, the same kind as any other process on your machine, that the kernel is lying to in two ways at once.

The first lie is called namespaces, and it is a lie about what the process can see. The Linux kernel can give a process its own isolated view of certain system resources: its own process list, so the process believes it is number one and cannot see anybody else; its own network interface, with its own addresses; its own filesystem tree; its own hostname; its own users. The process looks around and honestly believes it is alone on the machine. It is not alone. It simply cannot see the others.

The second lie is called cgroups, short for control groups, and it is a lie about what the process can consume. Namespaces isolate the view, but they do not divide the resources: an isolated process that goes haywire can eat all the memory and all the CPU on the machine and take down every neighbour. Cgroups set the limit: this process may use this much memory and this much CPU, and not one gram more.

Namespaces for what it sees, cgroups for what it spends. The full comparison against the virtual machine is in Figure 1.1. Those two things together, applied to an ordinary process, are a container.

And there is the underlying reason a container starts in milliseconds and a virtual machine in minutes: the container boots no operating system at all, because the operating system was already running. Makes sense, right?

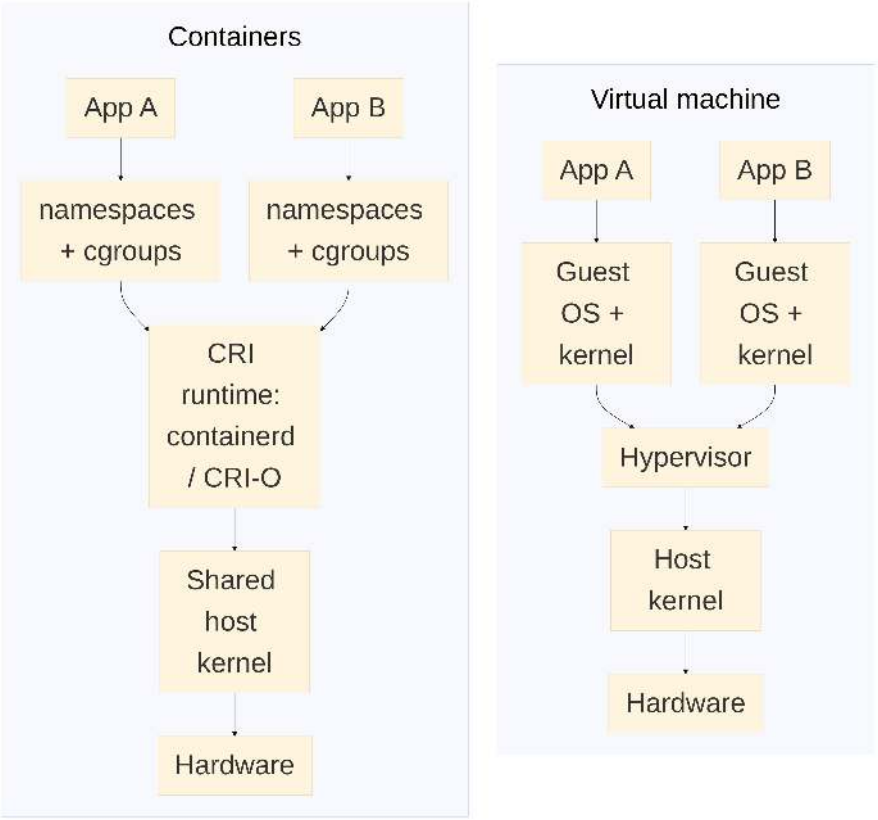


Figure 1.1. Each virtual machine boots its own guest kernel on a hypervisor; containers share the host kernel and isolate themselves with namespaces and cgroups.

> **Warning**
> A container is not a lightweight virtual machine. A virtual machine boots its own guest kernel on a hypervisor. A container boots no kernel at all: it is a host process, isolated with namespaces and limited with cgroups, on the kernel that was already running.

1.4 The Open Container Initiative (OCI): The Day Everybody Agreed

Now, having the technology is not enough. In the early years everybody packed containers their own way, and that is the perfect recipe for an ecosystem to shatter into pieces. If the image one tool produces can only be run by that same tool, you do not have a standard: you have a cage with a nice logo on it.

That is why on 22 June 2015 the Open Container Initiative, the OCI, was launched by Docker, CoreOS and other leaders in the container industry, under the umbrella of the Linux Foundation. It is a lightweight, open governance structure, and its stated purpose is to create open industry standards around container formats and runtimes. The goal was brutally simple: agree, so that an image built with any tool can be run by any other.

The OCI maintains three specifications, and it is worth knowing all three by name, because that is how they appear on the exam:

The **runtime-spec** describes how to execute a "filesystem bundle", meaning a package already unpacked on disk and ready to run.

The **image-spec** defines the OCI image format: the image manifest, the filesystem layer serialization, and the configuration needed to launch the application.

The **distribution-spec** standardizes the API (Application Programming Interface) for distributing images, and reached version 1.0 in May 2020. It is designed generically enough to distribute any kind of content, not only containers.

The three together produce the result you take for granted every day: you pull an image, it is unpacked into a bundle, and it executes with minimal extra arguments. Without having to ask anybody who built it, or with what.

1.5 So Where Does Kubernetes Fit In? The Container Runtime Interface (CRI)

Here we reach the point where this chapter connects to the rest of the book. Kubernetes does not run containers.

Read that again, because the exam asks it in a twisted way: Kubernetes does not run containers. Kubernetes asks another program to run them.

That other program is called a container runtime, and the conversation between the two happens through a protocol called the Container Runtime Interface, the CRI. On the Kubernetes side the speaker is the kubelet, the agent running on every node. On the other side is the runtime. The CRI is simply the common language between those two.

The runtimes Kubernetes officially supports are containerd, CRI-O, Docker Engine and Mirantis Container Runtime. And there is a version detail worth gold: since Kubernetes 1.26 the project only works with v1 of the CRI API. If a runtime does not support v1, the kubelet simply does not register the node. It is not that it half works or runs slowly: the node does not show up.

There is a second trap, quieter, and it has to do with the cgroups we already saw. Both the kubelet and the runtime need to interface with cgroups to enforce resource limits for pods and containers. It is critical that the two use the same cgroup driver and are configured the same way. The two available drivers are cgroupfs, which is the kubelet default, and systemd, which is recommended when the system uses systemd as its init and is also the recommended choice with cgroup v2. If you leave them mismatched, it will not blow up immediately: it will fail strangely, under resource pressure, which is the worst possible way to fail.

> **Caution**
> The kubelet and the runtime must use the SAME cgroup driver. If one uses cgroupfs and the other systemd, the node can start and look healthy, then fail later under resource pressure. With cgroup v2 and a systemd init system, the official recommendation is systemd on both sides.

1.6 Summary

Let us go back to your aunt's kitchen. It is Sunday, the stew already smells so good your mouth is watering, and yet, as we already know, it comes out terrible. Now we can finally explain why.

The underlying problem was never the program, it was the environment: the libraries, the versions, the variables, the files. Virtual machines solved that by shipping the whole kitchen, own kernel included, and they genuinely worked, but at the price of

dragging a complete operating system along for every service.

The container makes the opposite bet: it shares the host machine's kernel and isolates only what actually needs isolating. A container is not an object, nor a lightweight virtual machine: it is an ordinary process to which the kernel applies namespaces, which limit what the process can see, and cgroups, which limit what the process can consume. That is why it starts in milliseconds instead of minutes.

So that this model would not shatter into proprietary pieces, the industry created the Open Container Initiative in June 2015, under the Linux Foundation, with three specifications worth remembering by name: runtime-spec for execution, image-spec for the image format, and distribution-spec for distribution.

And Kubernetes, which is the subject of the entire rest of this book, does not run containers itself: the kubelet talks to a runtime through the CRI. containerd, CRI-O, Docker Engine and Mirantis Container Runtime are the officially supported ones, since version 1.26 only with the v1 API, and with the cgroup driver configured identically on both sides.

We have the kitchen packed, and now we can take it anywhere: same recipe, same result, at your aunt's house, on the company server, wherever. The stew finally tastes good everywhere. Except, mark this down, your aunt's still tastes better, because she puts love into the recipe, and no container packages that.

Now we can talk about who organizes the kitchens when there are hundreds of them. That is Kubernetes, and that is the next chapter.

Bibliography

- Kubernetes Documentation, Container Runtimes (production environment setup): supported runtimes, CRI v1 requirement, and cgroup driver configuration. https://kubernetes.io/docs/setup/production-environment/container-runtimes/
- Open Container Initiative, About the OCI: governance, founding, and the runtime, image and distribution specifications. https://opencontainers.org/about/overview/
- Kubernetes Blog, Upcoming changes in Kubernetes 1.26: removal of the CRI

v1alpha2 API and the resulting v1-only requirement. https://kubernetes.io/blog/2022/11/18/upcoming-changes-in-kubernetes-1-26/

- CNCF, Kubernetes and Cloud Native Associate (KCNA) Exam Curriculum: domain weights and competencies, Containerization under Kubernetes Fundamentals. https://github.com/cncf/curriculum