

CONTAINERS

The differences between Docker, containerd, CRI-O and runc

Containers run using a complex stack of libraries, runtimes, APIs and standards.

Written by [Tom Donohue](#) 

Updated: 02 January 2023

[26 Comments](#)

The explosion in containers was kicked off by Docker. But soon afterwards, the landscape seemed to explode with tools, standards and acronyms. So what is ‘docker’ really, and what do terms like “CRI” and “OCI” mean? Should you even care? Read on to find out.

Since Docker started the container frenzy, there’s been a Cambrian explosion of new container tools, standards and APIs.

But unless you’re right at the coal-face of technology, it can be very hard to keep up. And the power-games between the big tech companies can add to the confusion for the rest of us.

In this article, we’ll look at all the main names you’ve heard in container-land, try to descramble the jargon for you, and explain how the container ecosystem works.

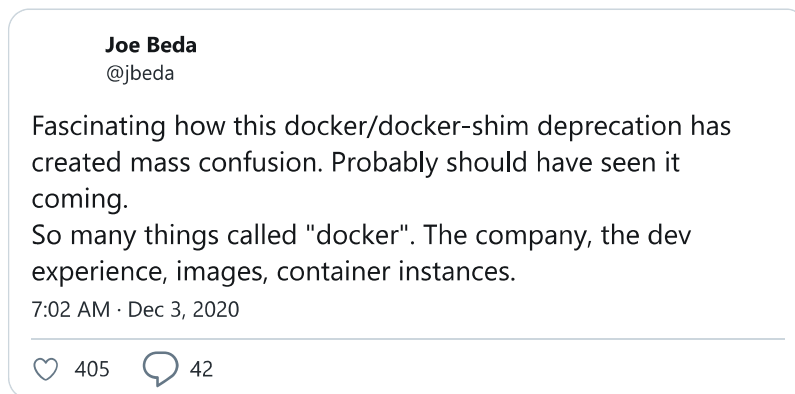
And if you think you’re the only one who doesn’t understand it all, don’t worry... you’re not. :)

Note:  You can also read this article in [German](#) (*translation courtesy of Anatoli Kreyman*)

How we got here

There's a difference between Docker Inc (the company), Docker containers, Docker images, and the Docker developer tooling that we're all familiar with.

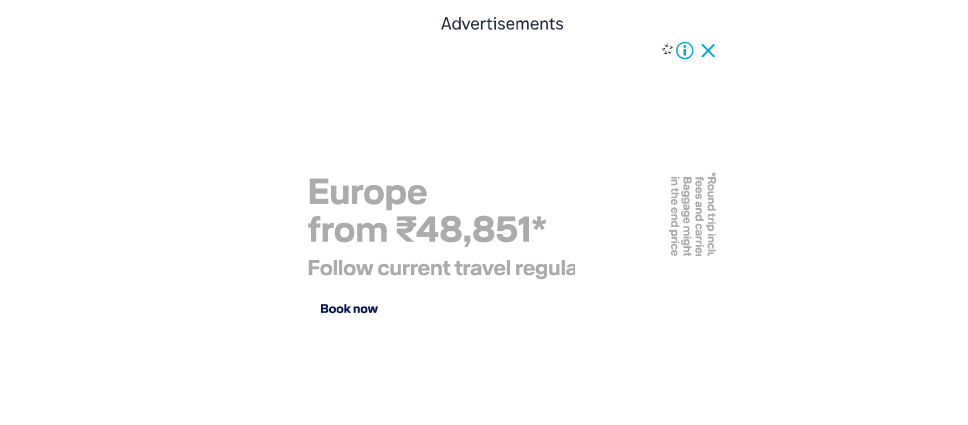
Joe Beda, co-founder of the Kubernetes project, noticed this:



(I told you that you're not the only one who's confused.)

This is a perfect opportunity to clear up some of the confusion and help you understand when it's Docker, and when it's not. Or perhaps when it's **containerd**, or **CRI-O**. This is especially useful if you're [learning Kubernetes](#).

The main thing to understand is this:



Containers aren't tightly coupled to the name Docker. You can use other tools to run containers.

Docker isn't the only container contender on the block.

You can be running containers with Docker, or a bunch of other tools which **aren't** Docker. `docker` is just one of the many options, and Docker (the company) creates some of the awesome [tools](#) in the ecosystem, but not all.

So if you were thinking that containers are just about Docker, then continue reading! We'll look at the ecosystem around containers and what each part does. This is especially useful if you're thinking of moving into [DevOps](#).

🔥 MORE DOCKER AND KUBERNETES GUIDES...

- [Containers 101](#): What is a container? What is an image?
- [Learn Kubernetes](#): How to begin your Kubernetes journey
- [Container networking](#): How containers talk to each other

- [Why use containers?](#): What are containers used for?

A bird's eye view

The container ecosystem is made up of lots of exciting tech, plenty of jargon, and big companies fighting each other.

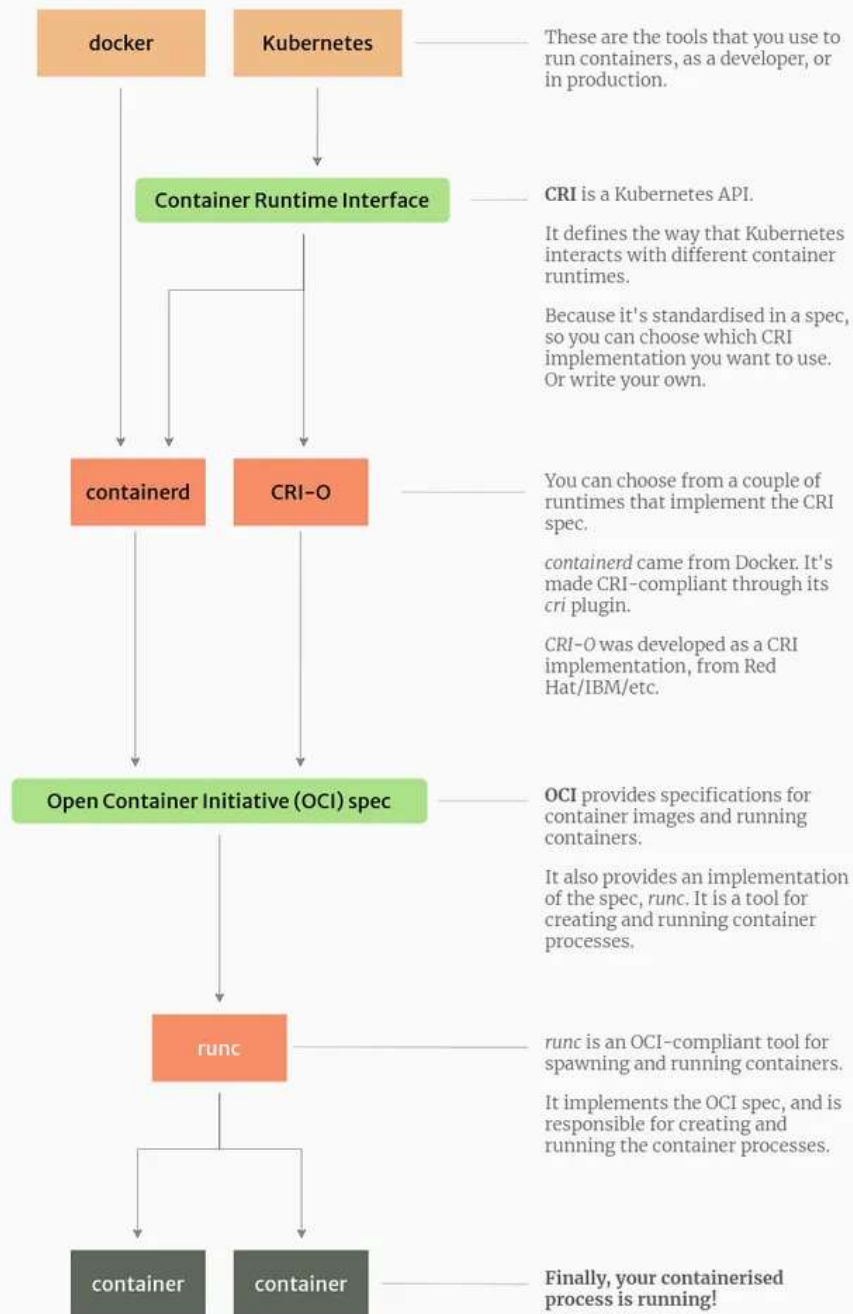
Fortunately, these companies occasionally come together in a fragile truce 🤝 to agree some **standards**. Standards help to make the ecosystem more interoperable. This means you can run software on different platforms and operating systems, and be less reliant on one single company or project.

There are two big standards around containers:

- **Open Container Initiative (OCI)**: a set of standards for containers, describing the image format, runtime, and distribution.
- **Container Runtime Interface (CRI) in [Kubernetes](#)**: An API that allows you to use different container runtimes in Kubernetes.

We'll take a look at these below. But first, this illustration gives an overview of how **Docker**, **Kubernetes**, **CRI**, **OCI**, **containerd** and **runc** fit together in this ecosystem:

Docker, Kubernetes, OCI, CRI-O, containerd & runc: How do they work together?



The relationship between Docker, CRI-O, containerd and runc – in a nutshell

Source: Tutorial Works

Let's look at each of these projects in turn.

Docker

In our rundown of container jargon, we've got to start with Docker. It's the most popular developer tool for working with containers. And, for a lot of people, the name "Docker" is synonymous with the word "container".

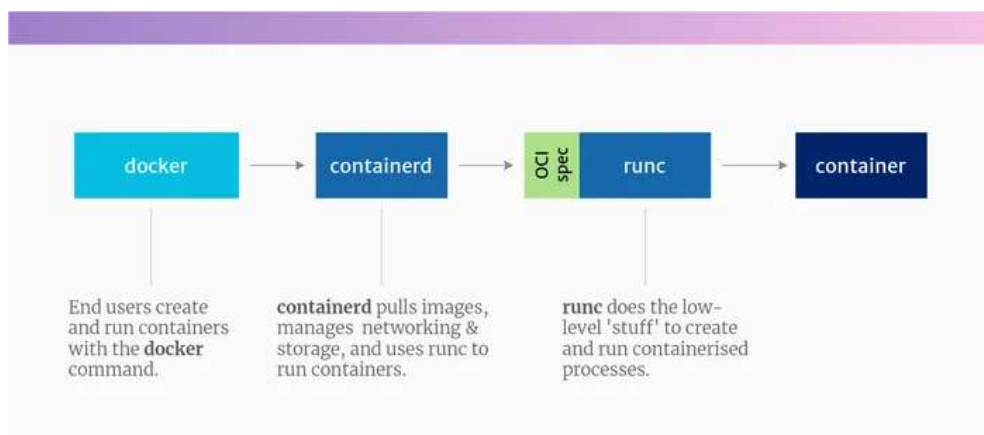
Docker kick-started the container space by creating a very *ergonomic* (nice-to-use) tool for creating and working with containers. The tool was called `docker`. It's now branded as [Docker Engine](#) and comes in two versions:

- Docker Desktop for your developer workstation, available for Windows, Mac and Linux
- Docker Engine for your server, available for Linux.

How the Docker stack works

Docker Engine comes with a bunch of tools to make it easy to build and run containers as a developer, or a systems administrator. It is basically a command-line interface (CLI) for working with containers.

But the `docker` command is just one piece of the puzzle. It actually calls down to some lower-level tools to do the heavy lifting:




The projects involved in running a container with Docker



Source: Tutorial Works

The `docker` command line tool can build container images, pull images from registries, create, start and manage containers. It does this by calling down to a bunch of lower-level tools.


What are the lower-level tools in the Docker stack?

From the bottom up, these are the tools that `docker` uses to run containers:

- **Lowest-level**  **The low-level container runtime.** [runc](#) is a low-level **container runtime**. It uses the native features of Linux to create and run containers. It follows the [OCI](#) standard, and it includes [libcontainer](#), a Go library for creating containers.

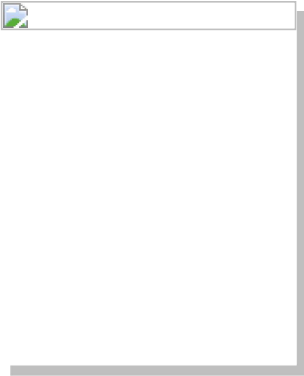
-  **The high-level container runtime.** [containerd](#) sits above the low-level runtime, and adds a bunch of features, like transferring images, storage, and networking. It also fully supports the OCI spec.
-  **The Docker daemon.** [dockerd](#) is a daemon process (a long-running process that stays running in the background) which provides a standard API, and talks to the container runtime ¹



- Highest level  **The Docker CLI tool.** Finally, [docker-cli](#) gives you the power to interact with the Docker daemon using `docker ...` commands. This lets you control containers without needing to understand the lower levels.

So, in reality, when you run a container with `docker`, you're actually running it through the Docker daemon, which calls `containerd`, which then uses `runc`.

If you want to learn more about the container ecosystem, you should check out Nigel Poulton's book, which explains it in more detail:



Docker Deep Dive
by Nigel Poulton

A really excellent technical book about Docker, for beginners and experts alike.

[Buy on Amazon →](#)

Does Kubernetes use Docker?

A really common question is “how do containers run in Kubernetes?”. Does Kubernetes use Docker? Well, it doesn’t anymore – but it used to.

Originally, Kubernetes used Docker (Docker Engine) to run containers.

But, over time, Kubernetes evolved into a container-agnostic platform. The [Container Runtime Interface \(CRI\)](#) API was created in Kubernetes, which allows different container runtimes to be plugged into it.

Docker Engine, being a project older than Kubernetes, doesn’t implement CRI. So to help with the transition, the Kubernetes project included a component called **dockershim**, which allowed Kubernetes to run containers with the Docker runtime.

It bridged the gap between the old world and the new.

☐ What is a shim?

Death of the shim

But, as of Kubernetes 1.24, the dockershim component [was removed completely](#) , and Kubernetes no longer supports Docker as a container runtime. Instead, you need to choose a container runtime that implements CRI.

The logical successor to Docker Engine in Kubernetes clusters is... [containerd](#) . *(10 points if you got that correct!)* Or you can use an alternative runtime, like [CRI-O](#) .

This doesn't mean that Kubernetes can't run so-called Docker-formatted containers. Both **containerd** and **CRI-O** can run Docker-formatted and OCI-formatted images in Kubernetes; they can do it without having to use the `docker` command or the Docker daemon.

Phew. Hope that cleared that up.

What about Docker Desktop?

Docker Desktop is a GUI application for Mac and Windows, which makes it easy to run Docker on your laptop. It includes a Linux virtual machine which runs the Docker daemon, and it also includes Kubernetes.

Docker Desktop is aimed mainly at developers. It provides a very nice UI for visualising your containers, and also includes an easy way to start a Kubernetes cluster.

You don't need Docker Desktop to work with containers, but it's a nice way to get started. If you're running Linux already, you can install Docker Engine instead, and you'll be good to go.

The standards and APIs: OCI and CRI

We saw earlier that some standards help to make it easier to run containers on different platforms. But what are these standards?

Open Container Initiative (OCI) specifications

The [OCI](#) was one of the first efforts at creating some standards for the container world. It was established in 2015 by Docker and others.

The OCI is backed by a bunch of tech companies and maintains a specification for the container image format, and how containers should be run.

The idea behind the OCI specifications, is to standardise what a container is, and what it should be able to do. That means you're free to choose between different runtimes which conform to the specification. And each of these runtimes might have a different lower-level implementation.

For example: you might use one OCI-compliant runtime for your Linux hosts, but a different runtime for your Windows hosts.

Kubernetes Container Runtime Interface

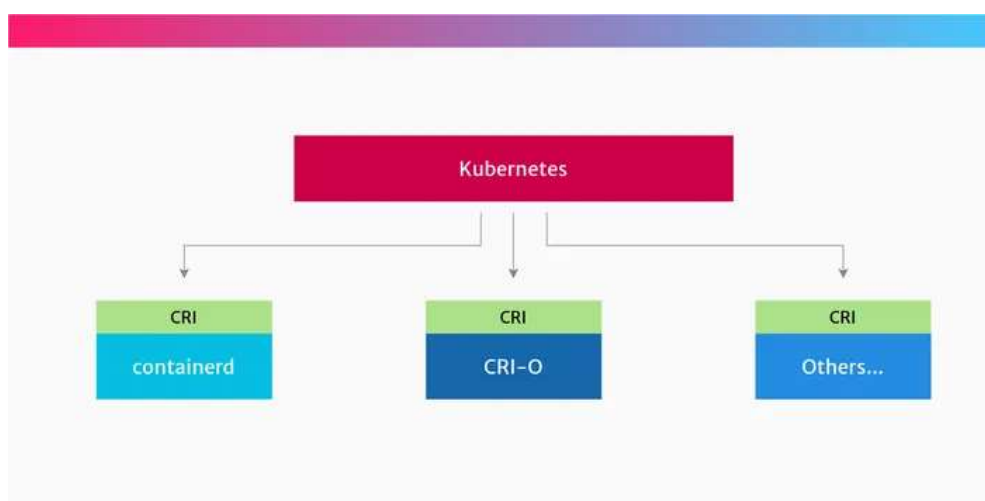
The other standard we need to talk about is the [Container Runtime Interface \(CRI\)](#) . This is an API that was created by the Kubernetes project.

CRI is an interface used by Kubernetes to control the different runtimes that create and manage containers.

Kubernetes tries not to care which container runtime you use. It just needs to be able to send instructions to it – to create containers for Pods, terminate them, and so on. And that's where CRI comes in. CRI is an abstraction for any kind of container runtime that might exist now, or in the future. So CRI makes it easier for Kubernetes to use different container runtimes.

Instead of the Kubernetes project needing to add support for each runtime individually, the CRI API describes how Kubernetes will interact with **any** runtime. As long as a given container runtime implements the CRI API, the runtime can create and start containers however it likes.

Here's a quick visualisation of how CRI fits into the container ecosystem:



You can choose your own container runtime for Kubernetes

Source: Tutorial Works

So if you prefer to use **containerd** to run your containers in Kubernetes, you can! Or, if you prefer to use **CRI-O**, then you can. This is because both of these runtimes implement the CRI specification.

If you're an end user (like a developer), the implementation mostly shouldn't matter. There are subtle differences between different CRI implementations, but they are intended to be pluggable and seamlessly changeable.

But, if you pay to get support (security, bug fixes etc) from a vendor, your choice of container runtime might be made for you. For example, Red Hat's [OpenShift uses CRI-O](#), and offers support for it. Docker provides support for their own **containerd**.

How to find out your container runtime in Kubernetes

containerd and CRI-O

We've seen that Docker Engine calls down to a bunch of lower-level tools. But what are these tools? And how do they fit together?

The first layer is the high-level runtimes: **containerd**, created by Docker, and **CRI-O**, created by Red Hat.

containerd

[containerd](#) is a high-level container runtime that came from Docker. It implements the CRI spec. It pulls images from registries, manages them and then hands over to a lower-level runtime, which uses the features of the Linux kernel to create processes we call 'containers'.

containerd was born from parts of the original Docker project. But why was it separated it out in this way? Well, it basically made Docker more modular. And this allowed Docker's particular 'flavour' of containers to run in more places (like on Kubernetes), without needing the Docker daemon and the Docker CLI tool.

So internally, Docker Engine uses **containerd**. When you install Docker, it will also install **containerd**.

containerd can be used as the container runtime for Kubernetes, because it implements the Kubernetes Container Runtime Interface (CRI), via its **cri** plugin.

CRI-O

CRI-O is another high-level container runtime which implements the Kubernetes Container Runtime Interface (CRI). It's an alternative to containerd. It pulls container images from registries, manages them on disk, and launches a lower-level runtime to run container processes.

Yes, CRI-O is another container runtime. It was born out of Red Hat, IBM, Intel, [SUSE](#) and others. [See here for the backstory on why CRI-O was created.](#)

CRI-O was created to be a container runtime for Kubernetes. It provides the ability to start, stop and restart containers, just like **containerd**.

Just like containerd, CRI-O implements the CRI API, so it can be used as a container runtime on Kubernetes.

runc and other low-level runtimes

runc is an OCI-compatible container runtime. It implements the OCI specification and runs the container processes.

runc is sometimes called the “reference implementation” of OCI.

What is a reference implementation?

runc provides all of the low-level functionality for containers, interacting with existing low-level Linux features, like namespaces and control groups. It uses these features to create and run container processes.

Other low-level runtimes

But, runc isn't the only low-level runtime. The OCI specification is allowing other tools to implement the same functionality in a different way:

- [crun](#) a container runtime written in **C** (by contrast, runc is written in Go.)
- [firecracker-containerd](#) from AWS, which implements the OCI specification as individual lightweight VMs (and it is also the same technology which powers AWS Lambda)



- [gVisor](#) from Google, which creates containers that have their own kernel. It implements OCI in its runtime called `runcsc`.

What's the equivalent of runc on Windows?

Summary

Well, we've come to the end of this journey. But what have we learned?

Docker is just one part in the ecosystem of containers.

There is a set of open standards which, theoretically, make it easier to swap out different implementations. Projects like **containerd**, **runc** and **CRI-O** implement parts of those standards.

In Kubernetes, you can choose which container runtime you want to use, as long as it supports the CRI API. You can use **containerd** or **CRI-O**.

With everyone *working hard* on all this new tech, expect this to change rapidly, as things progress. We'll try and keep this article updated where we can.

[i](#) [x](#)

29Rs Only Today

Try Oda Class and you will find math is not hard at all

Oda Class

Boo

So now you know everything there is to know about the fun (and slightly over-complicated) world of containers.

But the next time you strike up a conversation at a party 🍷, maybe it's best not to talk about containers, or you'll end up with a room full of people who are bored to tears. 😭

1. Poulton, Nigel. [Docker Deep Dive](#) . Nigel Poulton, 2020. [↩](#)



By **Tom Donohue, Editor** | [Twitter](#) | [LinkedIn](#)

Tom is the founder of *Tutorial Works*. He's an engineer and open source advocate. He uses the blog as a vehicle for sharing tutorials, writing about technology and talking about himself in the third person. His very first computer was an Acorn Electron.

Thanks for reading. Let's stay in touch.

It took us this long to find each other. So before you close your browser and forget all about this article, shall we stay in touch?

Join our free members' newsletter. We'll email you our latest tutorials and guides, so you can read at your leisure! 📧 (No spam, unsubscribe whenever you want.)

Subscribe

Join the discussion

Got some thoughts on what you've just read? Want to know what other people think? Or is there anything technically wrong with the article? (We'd love to know so that we can correct it!) **Join the conversation and leave a comment.**

Comments are moderated.

26 Comments

Type Comment Here (at least 3 chars)

Name

E-mail (optional)

Website (optional)

John Doe

johndoe@example.com

https://example.com

Preview

Submit



Suraj • 8 months ago

Very well explained.

[Reply](#)



Tom Donohue • 8 months ago

Thanks Suraj!

[Reply](#)



AnonymousWorm • 8 months ago

Amazing! Thank you so much, best explanation ever :)

[Reply](#)



Tom Donohue • 8 months ago

You're welcome, wormy.

[Reply](#)



Ivan • 8 months ago

Thanks!

[Reply](#)



Henk • 8 months ago

Thank you for this post! One thing that remains somewhat unclear is where does the docker daemon fit into the story? As the article states, docker uses containerd in the background, so what's dockerd then?

[Reply](#)



Tom Donohue • 8 months ago

Hi Henk, good question! *dockerd* is the Docker daemon. It opens a socket and listens for requests (to create/manage containers). The *docker* CLI utility actually talks to this socket. So when you type *docker run*, you're actually talking to the Docker daemon, not *containerd*.

Then, the Docker daemon calls the lower level container runtime (*containerd*) to actually start/run the container.

So the image above is a little simplified. You could add another box above, which shows that "docker CLI talks to -> docker daemon"

From [the docs](#):

"By default, the Docker daemon automatically starts *containerd* "

[Reply](#)



Henk • 8 months ago

Thank you for that :) This is what I assumed, but thought to ask anyway as *dockerd* was the only piece that was mentioned, but not really explained.

[Reply](#)



Srinivas • 8 months ago

Thank you so much for just a great explanation, Cheers.

[Reply](#)



Tom Donohue • 8 months ago

Thanks for your feedback, Srinivas!

[Reply](#)



hitfxw • 8 months ago

awesome

[Reply](#)



Mohan • 7 months ago

Superb explanation :)

[Reply](#)



Anonymous • 7 months ago

THX!

[Reply](#)



automation • 7 months ago

yes this is what i needed o know so let me explain you what i have understand and what will i going to do to fix my problem

i have been trying to set up k8s cluster i was doing this with docker but having problems with CNI as i was using weave CNI but pods were not being created then i reset cluster and set it up again another time i was having issues with

cluster initialization as it was saying please specify cri runtime to get cluster initialized so this problem i got understand now i think it was not recognizing containerd since docker use containerd as runtime but i want to use cri-o runtime now so what i got to understand from this article that i dont need to install docker if i am using cri-o runtime as this implements cri interface and come up with runc please correct me if i got it right and if i m right then please explain what is cli for cir-o as we have docker-cli and use docker commands however we have k8s but still sometime we need to troubleshoot it on docker-cli level also so please tell me if there is any cli for cri-o thanks

[Reply](#)

Tom Donohue • 7 months ago

You're correct! And I think the command you're looking for is 'crictl'. It will show you containers running on a node, e.g.

```
crictl ps
```

[Reply](#)

automation • 7 months ago

thanks i have been struggling in kubernetes networking when i deploy cni not able to fix this i want to understand its networkng how k8s networking work and i want a article or video as same way as you did for this(docker) could you explain me k8s networking how the hack its working

[Reply](#)

Tom Donohue • 7 months ago

I recommend you check out [Kubernetes in Action](#) - it's a great book and has a chapter which explains all about networking!

This is also a good blog post too: <https://itnext.io/kubernetes-networking-behind-the-scenes-39a1ab1792bb>

[Reply](#)

Hengki • 7 months ago

Thank you so much... Amazing! Great explanation

[Reply](#)

Sujit G • 6 months ago

What an excellent article, Tom. Really appreciate the clarity you provided to this topic.

[Reply](#)

Chris • 6 months ago

Great article!

[Reply](#)

John Dondapati • 6 months ago

Tom, this is the best article I have found that explains the whole container ecosystem in detail. The visualizations help clarify where each spec sits. Thank you for taking time to publish this and make it available for free. Much appreciated. Keep up the good work.

[Reply](#)**Rajkumar** • 5 months ago

Thanks!

[Reply](#)**Benson** • 5 months ago

What a excellent explanation.. much much appreciated

[Reply](#)**Mahesh** • 4 months ago

Thank you very much for the valuable article! It really helped to clear up some doubts I had.

[Reply](#)**krishnamohan yerrabilli** • 4 months ago

Great work Appreciated 🙌

[Reply](#)**Nitin** • last month

Very well articulated.

[Reply](#)

Want more? Read these articles next...

You've found the end of another article! Keep on learning by checking out one of these articles:

- [Run a web server in a Linux VM with Vagrant \[Learning Project\]](#): Learn Linux and virtualisation basics by deploying a website in this tutorial.
- [The Best Places to Learn & Try Kubernetes Online](#): Learning Kubernetes can seem challenging. But fear not! Here's a boatload of resources that will help you get there.
- [Why use Containers?](#): What are the benefits of using Docker containers, and what are containers used for?
- [Kubernetes and Docker: What's the difference?](#): Still don't get the difference between Docker and Kubernetes? We've got you! Find out what these tools do.

- [DevOps Project Ideas](#): A career in DevOps is all about building a broad skill base and understanding. Use these project ideas to invest in yourself and get that...

Advertisements

[report this ad](#)

Tutorial Works is a website to help you navigate the world of IT, and grow your tech career, with tips, tutorials, guides, and *real opinions*.

Thanks for being here today! 🙌

[About this website](#)

[Privacy policy](#)

[Contact us](#)

TOPICS

[Linux](#)

[Containers](#)

[DevOps](#)

[Kubernetes](#)

[Java](#)

FREE STUFF

[DevOps Roadmap](#)

SOCIALS

[Twitter](#) 

[GitHub](#)

[YouTube](#)

Copyright © 2022 Tom Donohue. All rights reserved, except where stated. You can use our illustrations on your own blog, as long as you include a link back to us.

Tutorial Works is a participant in the Amazon.com Services LLC Associates Program. As an Amazon Associate we earn from qualifying purchases. Amazon and the Amazon logo are trademarks of Amazon.com, Inc. or its affiliates.