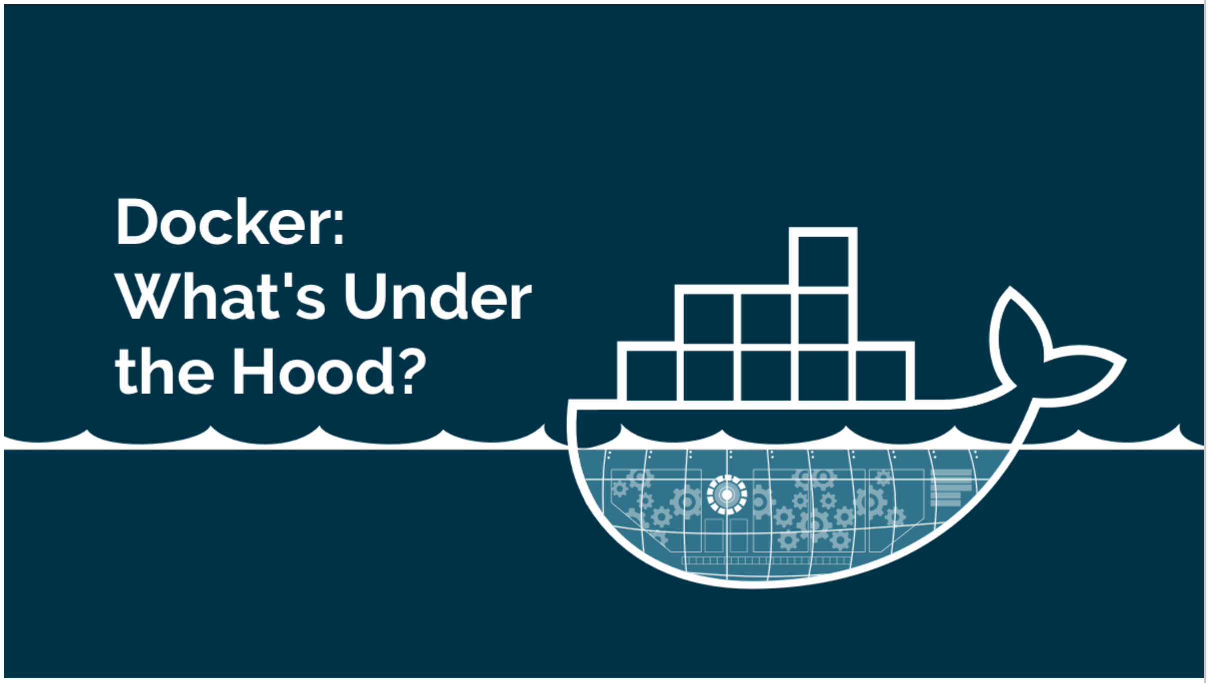
Codementor Blog

Last updated Nov 09, 2023

Docker: What's Under the Hood?

How does Docker work? Get a better understanding of the skeleton of Docker, Virtualization, and future development.



In recent years, Docker has taken a major place in developers' workflow. Many articles exist on what Docker is and how to start using it. I aim to provide an overview of how the technology works.

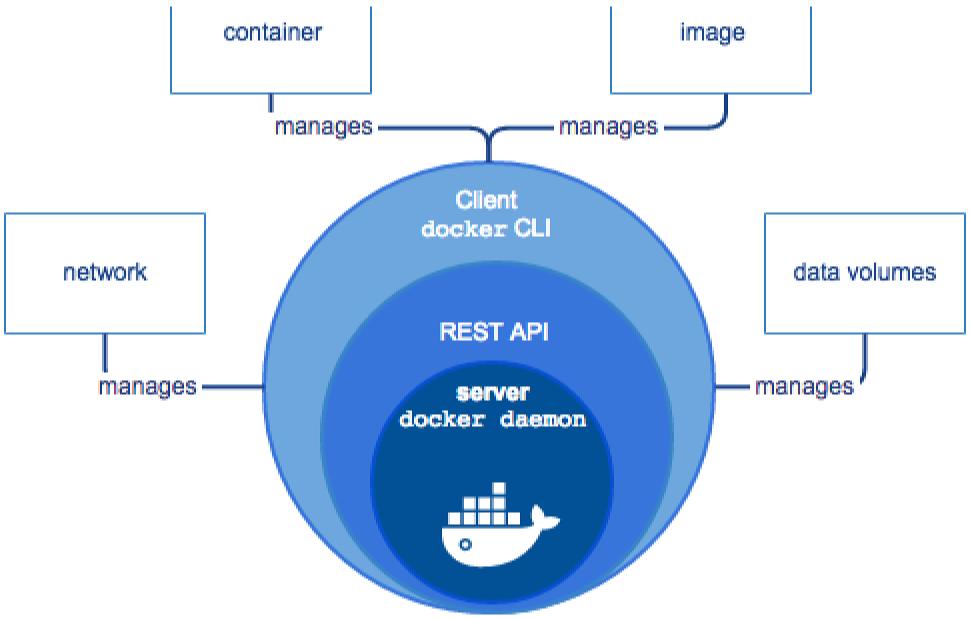
RELATED ARTICLES

3 Ways Software Engineers and Data Scientists Can Work Better Together

Swift Package Manager vs CocoaPods vs Carthage for All Platforms

How YC Alum Polymail Grew to Over 25,000 Active Users with Continuous Customer

By using Codementor, you agree to our Cookie Policy. ACCEPT codementor Become a mentor Log in



Need a boost in your Docker knowledge?
Work with expert Docker devs now
codementor

Docker's Features

One of the most important features Docker offers is it's instant startup time. A Docker container can be started within milliseconds, as opposed to waiting minutes for a virtual machine to boot. Docker piggybacks off of features in the Linux kernel to perform its magic. Because of this reliance on the Linux kernel, it's important to note that Docker *only* runs on Linux. For instance, if you develop on an Apple computer (which uses a Darwin/BSD Kernel), you'll need to install a lightweight Lir

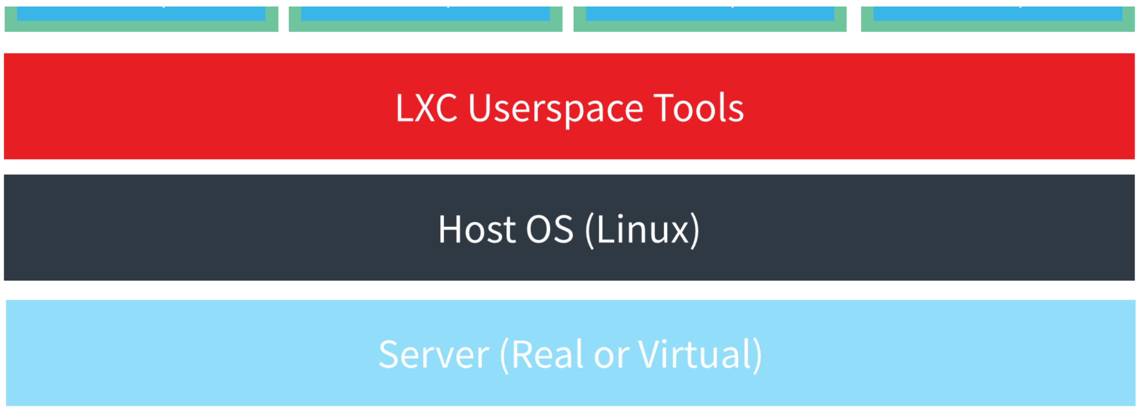
By using Codementor, you agree to our Cookie Policy.

Once you are familiar with the underlying technologies, Docker will seem less mystical. Docker is simulating various Linux distributions, environments or installs instead of running them. Did you know that, if you wanted to, you could take an Ubuntu distribution and install/remove software until it looks and feels like another distribution, such as Gentoo? After all, they both are just using the Linux kernel. Various Linux distributions are just different userland/package managers on top of the Linux kernel!

Let's take a step back. An operating system is software that runs on hardware. An operating system was originally called a "supervisor", which is a fantastic analogy of what the operating system is doing. Think of the operating system as a referee or administrator, watching over the other programs. The operating system lets other programs run while coordinating the scheduling and execution of those programs. Remember when computers could only do one thing at a time back in the day? Actually, that's still true, it's the operating system that is switching context back and forth very quickly, giving the appearance of multiple programs running at the same time.

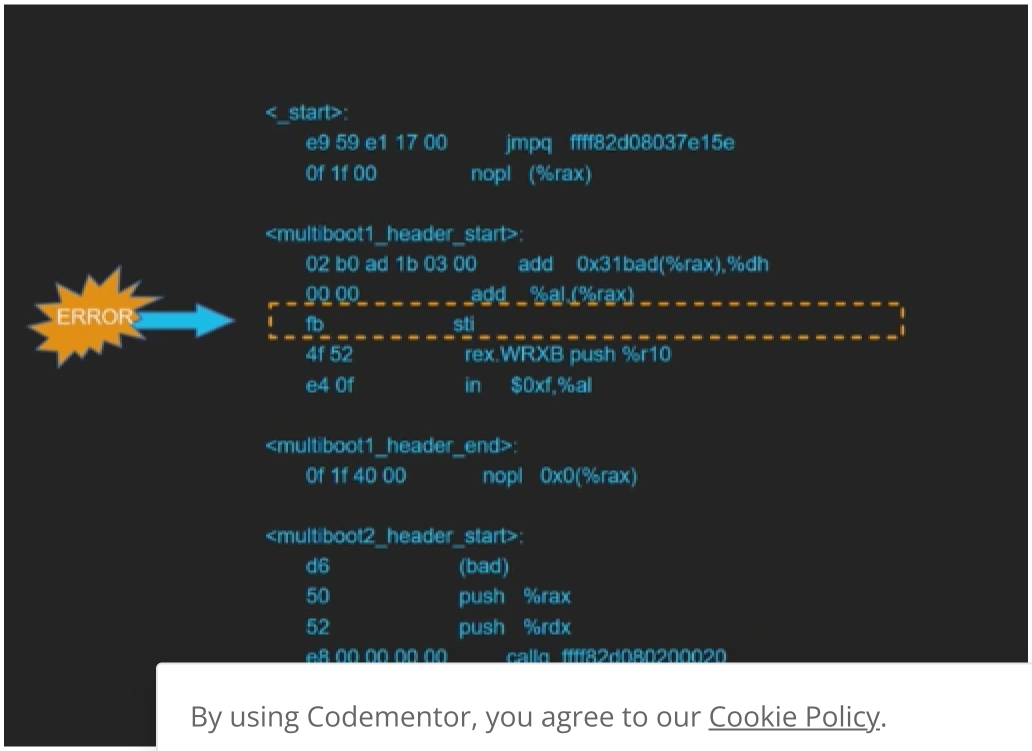
What is Virtualization?

By using Codementor, you agree to our Cookie Policy.



Let's talk about virtualizing a processor. A processor run machine languages (compiled assembly code) when running a program. Assembly and machine languages need to conform to the processor's instruction set, so an Intel processor will run x86 machine language while your iPhone runs ARM machine language.

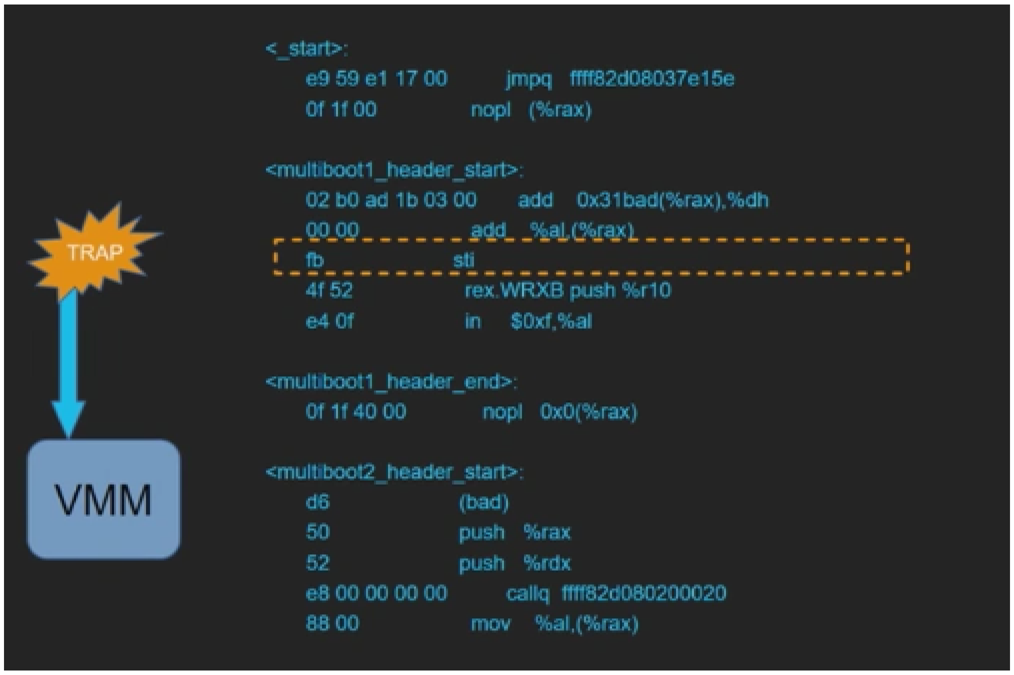
What happens if we were to run a x86 program on an ARM processor? Here is a simplified example:



```
<_start>:
     e9 59 e1 17 00        jmpq   ffff82d08037e15e
     0f 1f 00              nopl   (%rax)

<multiboot1_header_start>:
     02 b0 ad 1b 03 00     add    0x31bad(%rax),%dh
     00 00                 add    %al,(%rax)
     fb                    sti
     4f 52                 rex.WRXB push %r10
     e4 0f                 in     $0xf,%al

<multiboot1_header_end>:
     0f 1f 40 00           nopl   0x0(%rax)

<multiboot2_header_start>:
     d6                    (bad)
     50                    push   %rax
     52                    push   %rdx
     e8 00 00 00 00        callq  ffff82d080200020
```

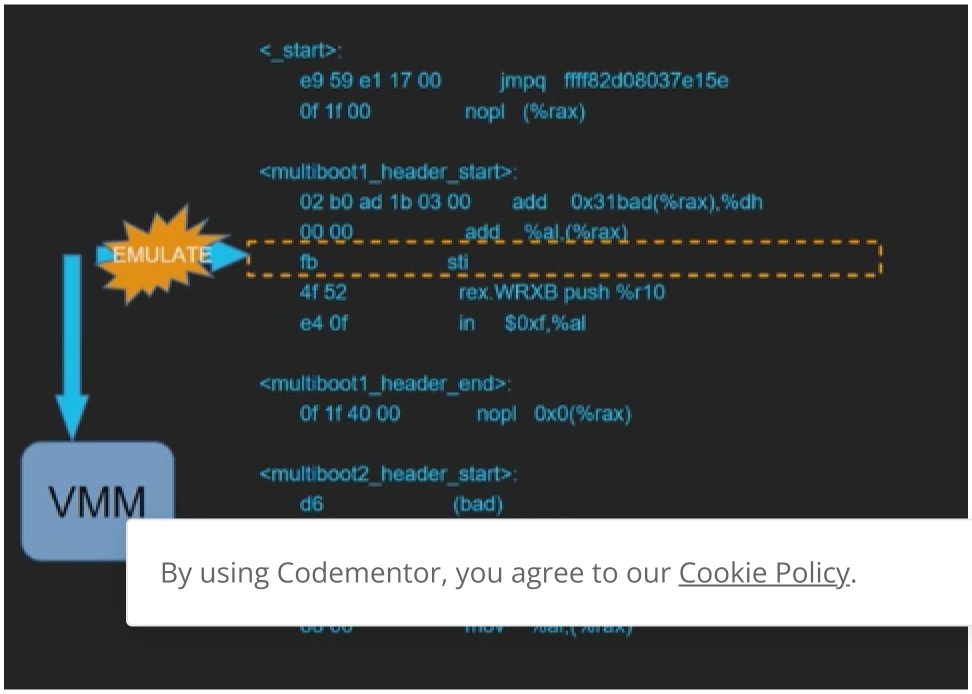
By using Codementor, you agree to our Cookie Policy.

crash.



```
<_start>:
    e9 59 e1 17 00        jmpq   ffff82d08037e15e
    0f 1f 00              nopl   (%rax)

<multiboot1_header_start>:
    02 b0 ad 1b 03 00     add    0x31bad(%rax),%dh
    00 00                 add    %al,(%rax)
    fb                    sti
    4f 52                 rex.WRXB push %r10
    e4 0f                 in     $0xf,%al

<multiboot1_header_end>:
    0f 1f 40 00           nopl   0x0(%rax)

<multiboot2_header_start>:
    d6                    (bad)
    50                    push   %rax
    52                    push   %rdx
    e8 00 00 00 00        callq  ffff82d080200020
    88 00                 mov    %al,(%rax)
```

However, the processor can trap particular errors, meaning if you see something like this, go shuffle it off somewhere. In this case, it would trap the error and send the instruction to the Virtual Machine Manager.



```
<_start>:
    e9 59 e1 17 00        jmpq   ffff82d08037e15e
    0f 1f 00              nopl   (%rax)

<multiboot1_header_start>:
    02 b0 ad 1b 03 00     add    0x31bad(%rax),%dh
    00 00                 add    %al,(%rax)
    fb                    sti
    4f 52                 rex.WRXB push %r10
    e4 0f                 in     $0xf,%al

<multiboot1_header_end>:
    0f 1f 40 00           nopl   0x0(%rax)

<multiboot2_header_start>:
    d6                    (bad)
```

By using Codementor, you agree to our Cookie Policy.

codementor Become a mentor Log in

Full Virtualization: QEMU

QEMU (quick emulator) is software that performs hardware virtualization. QEMU emulates CPUs through dynamic binary translation and provides device models, letting it run on unmodified host operating systems. The downsides is that it's very slow. Instructions need to be mapped from one set to another, translating the x86 `sti` [Set Interrupt Flag] would need to be mapped into many different ARM instructions.

Hardware-Assisted Virtualization (Intel Virtualization Technology)

Operating systems run in privileged mode to access drivers. Linux kernel has special instructions that are meant to run in that privileged mode. If those specialized instructions were to be run at an application privilege level, the special instruction will error. But this is very slow. The idea is to trap execution calls and send them to the virtualization system as the goal is to have most calls run natively and only trap/VMM a small subset of calls.

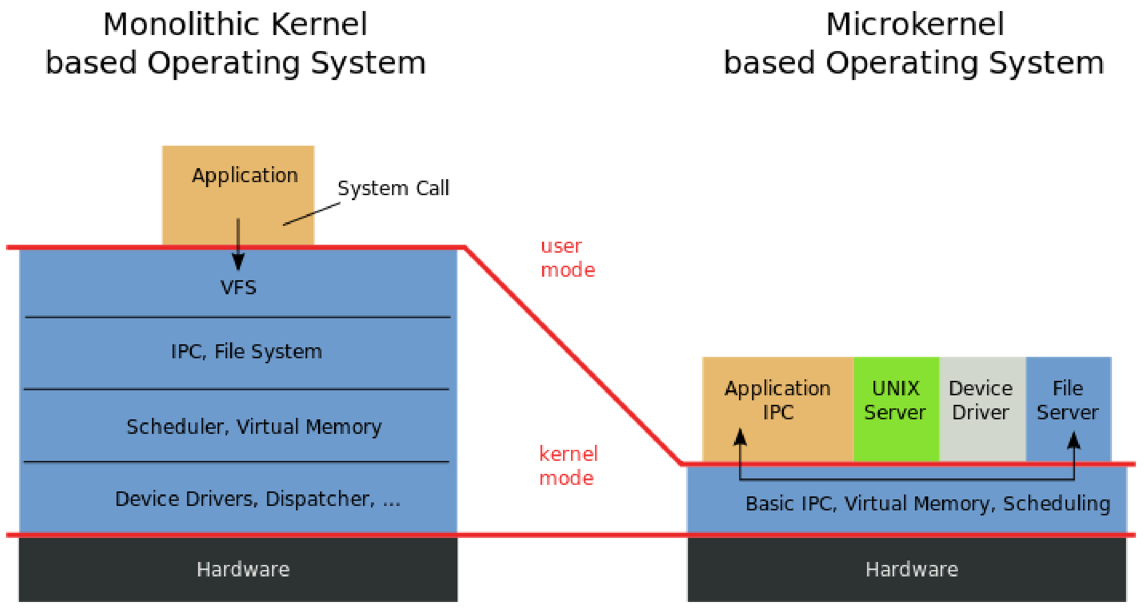
Hardware-assisted virtualization first appeared in an IBM System/370 in 1972. If you're on an Intel/AMD x86 CPU purchased in the last 10 years, you have it as Intel added this technology in 2005.

Paravirtualization: Xen Hypervisor

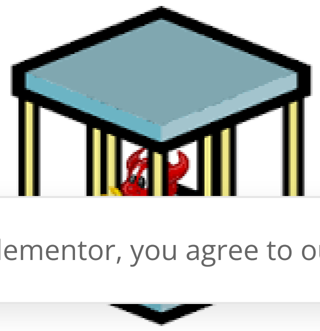
Applications will often make system calls, such as listening to a network socket or reading a file. These are privileged system calls only the kernel can

By using Codementor, you agree to our Cookie Policy.

The Xen hypervisor is like an operating system of operating systems. It uses a microkernel design and interacts with guests (such as Ubuntu or Debian Linux) via `dom0` (or the Xen hypervisor) over a "hypercall" (vs system call) using an ABI (application binary interface), as opposed to API. So an application running inside a virtualized Ubuntu operating system will make a system call for a file to be opened and read, which will then be sent to Xen as a hypercall. Xen will then use the driver to open and read the file from the hard drive. Here is a simplified example:



Other Operating System Level Virtualization: FreeBSD Jails



By using Codementor, you agree to our Cookie Policy.

Docker (2013) were released.

FreeBSD Jails are a BSD userland software that runs on top of the chroot(2) system call. The two are very similar, but FreeBSD Jails can only be used on FreeBSD, and thus, are not nearly as popular as Docker.

> *Since system administration is a difficult task, many tools have been developed to make life easier for the administrator. These tools often enhance the way systems are installed, configured, and maintained. One of the tools which can be used to enhance the security of a FreeBSD system is jails. Jails have been available since FreeBSD 4.X and continue to be enhanced in their usefulness, performance, reliability, and security.* ***- FreeBSD Handbook, Chapter 14. Jails***

Tools & Terminology

There are several tools for Docker.

The Docker client is `docker`, which most of your commands will use.

If you using an OS X host, there is also `docker-machine`. This is a wrapper on the OS X host to help set up the lightweight Linux virtual machine to execute to `docker` commands inside it

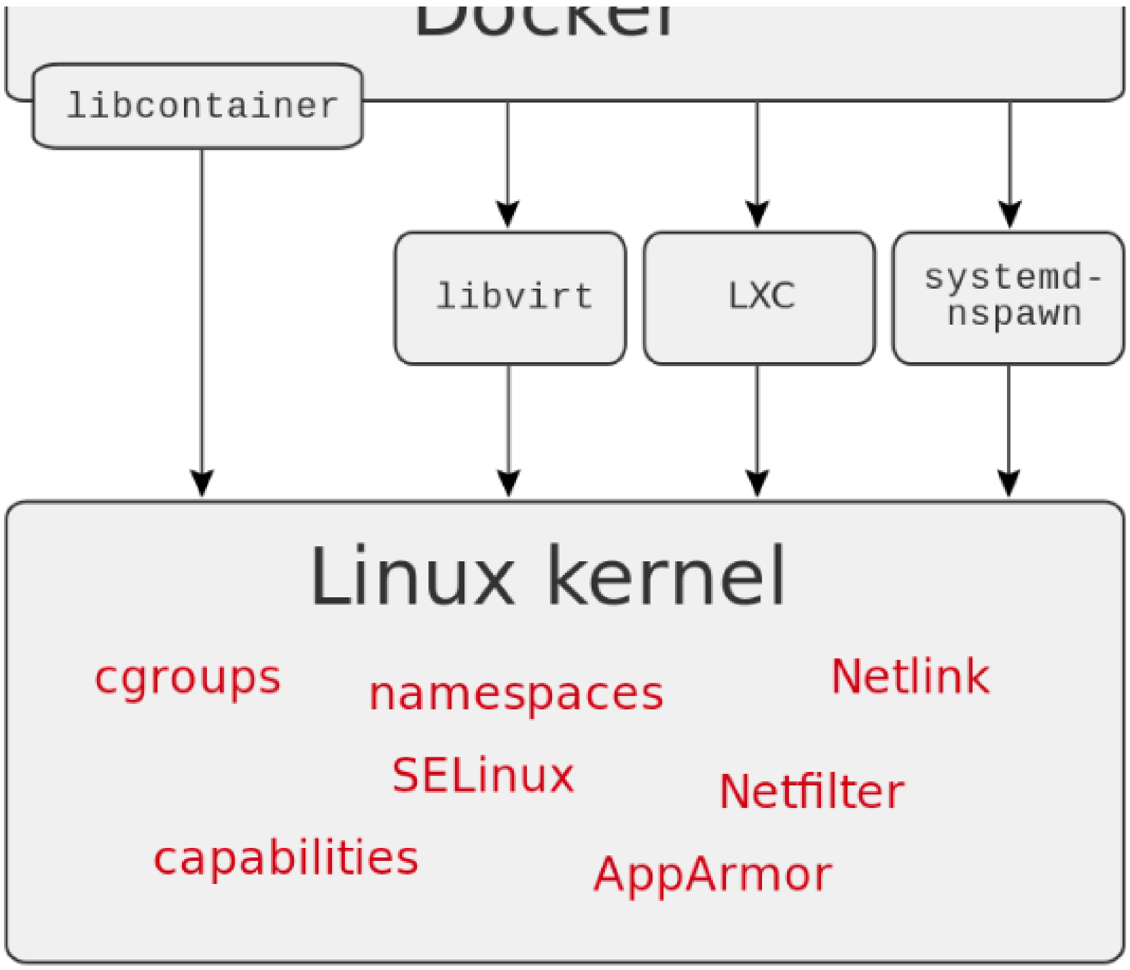
By using Codementor, you agree to our Cookie Policy.

docker image. Artifacts, or Docker images, can be stored on private or public repositories called registries. Some common registries are Docker Hub, quay.io, and AWS ECR.

A Docker container is the execution environment, or run-time instance of a Dockerfile. It is the
runnable version of an image and is written/run on your file system.
An analogy might be:
Docker image is to a `.jar` file as a Docker container is to a running process of that `.jar`.

Technology

By using Codementor, you agree to our Cookie Policy.



cgroups & namespaces

The backbone of the Docker technology are `cgroups` (short for control groups) and kernel namespaces, both of which are features already provided in the Linux kernel. With `cgroups`, the Linux operating system can easily manage and monitor resource allocation for a given process and set resource limits, like CPU, memory, and network limits. The kernel can now control the maximum amount of resources a process gets. This lets the Docker engine only give out 50% of the computer's memory, processors, or network, for example, to a running Docker container.

By using Codementor, you agree to our Cookie Policy.

and `uts`. Each container will have its own namespace and processes running inside that namespace, and will not have access to anything outside its namespace.

Docker containers also have network isolation (via `libnetwork`), allowing for separate virtual interfaces and IP addressing between containers.

Union File System

Docker uses the union file system to create and layer Docker images. This means all images are built on top of a base image, actions are then added to that base image. For example, `RUN apt install curl` creates a new image. Under the hood, it allows files and directories of separate file systems, known as branches, to be transparently overlaid, forming a single coherent file system. When branch contents have the same directory, the contents are merged and seen together the directory.

When building an image, Docker will save these branches in its cache. A benefit is, if Docker detects the command to create a branch or layer already existing in the cache, it will re-use the branch, not the command, which is a cache hit. This is known as docker layer caching.

libcontainer / runC

The C library that provides the underlying Docker functionality is called `libcontainer`, or now known as `runC`. For each execution, each container gets

By using Codementor, you agree to our Cookie Policy.

underlying host

`jail` is where the operating system prevents that started process from accessing a parent directory, such as `../`).

Faking it: udocker

One of the drawbacks of Docker is that the Docker engine requires root
privileges to run containers. Enter `udocker`. It's an open source project and provides the same basic functionality the Docker engine does but without root privileges.

It works by creating a `chroot`-like environment over the extracted container and uses various implementation strategies to mimic `chroot` execution with just user-level privileges. One of the execution environments you can use is `runC`, the same one used by Docker.

For instance, imagine the following script:

By using Codementor, you agree to our Cookie Policy.

```
set -eou pipefail

if [[ ! -d ~/bin ]]; mkdir ~/bin; fi
PATH="${PATH}:~/bin"

curl \
  https://raw.githubusercontent.com/skilbjo/lambdas/master/iris/resources/udock
  >./udocker \
  && chmod 744 udocker \
  && mv ./udocker ~/bin

udocker install
udocker pull quay.io/skilbjo/iris:x86-debian   # pull down image from docker re
udocker create --name=_ quay.io/skilbjo/iris:x86  # 'create' a container / unpa
udocker setup --execmode=F1 _  # set the execution engine
udocker run \
  --nosysdirs \
  _ \
  /bin/bash -c 'echo "hello from inside udocker!"'
```

It will return:

```

Warning: running as uid 0 is not supported by this engine
Warning: non-existing user will be created

##############################################################################
# #
# STARTING e7818ced-28b7-3bb5-9bdf-c8586cf6ae3c #
# #
##############################################################################
executing: bash
hello from inside udocker!
done!
```

By using Codementor, you agree to our Cookie Policy.

The future: Unikernels

Take a step back: much of the Internet runs inside AWS data centers, on virtual machines known as EC2 (an acronym for Elastic Computer Cloud). Many web services and web applications we use everyday are run on these EC2 machines inside Docker containers. The end result is many different middleware layers before the application can get access to the physical processor.

The idea behind `unikernels` is to strip out much of the kernel and unnecessary software (since an operating system is meant to be generic and do many different things) to be able to just run the application with the minimum number of kernel modules and drivers for the application to work. If the application doesn't need network, then no network drivers or kernel code is included in the kernel.

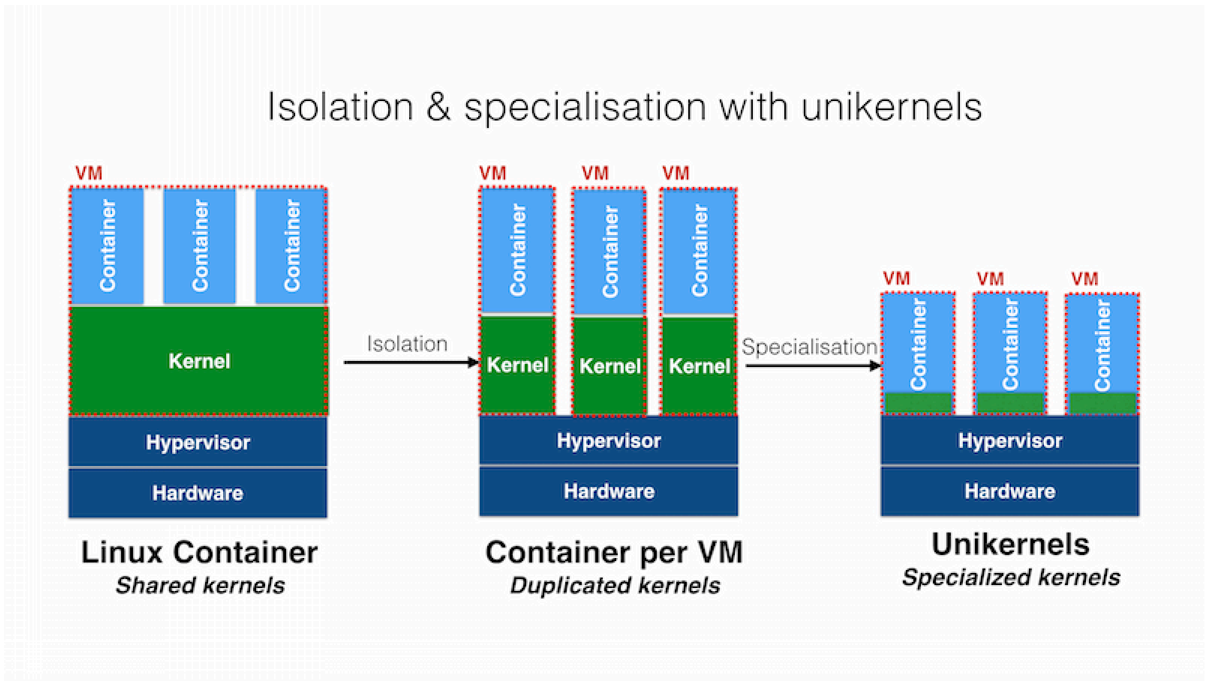
Three years ago, Docker acquired a company called Unikernel Systems to potentially build this concept, as one of Docker's goals is to provide high-quality tools and documentation for developing software on their platform.

One of Docker's incredible advantages is to provide an easy-to-use abstraction of

virtualization t

By using Codementor, you agree to our Cookie Policy.

native operating system.



Want to learn more for free?

Visit Codementor Events

Sources

- https://docs.docker.com/engine/docker-overview/#docker-engine
- https://github.com/skilbjo/articles/blob/master/talks/free_bsd_jails.md
- https://stackoverflow.com/questions/16047306/how-is-docker-different-from-a-virtual-machine
- https://devopscube.com/what-is-docker/
- https://elinu

By using Codementor, you agree to our Cookie Policy.

- https://github.com/moby/moby (Docker source code)
- https://github.com/docker/engine/blob/master/daemon/daemon.go
- https://books.google.com/books?id=4xQKBAAAQBAJ
- https://wxdublin.gitbooks.io/docker-code-walk/content/daemon.html
- https://www.itworld.com/article/2698646/virtualization/containers-bring-a-skinny-new-world-of-virtualization-to-linux.html
- https://www.youtube.com/watch?v=LabltEXk0VQ
- https://github.com/indigo-dc/udocker

Last updated on Nov 09, 2023

Engineering & Technical Insights

Like this article? Share it with your friends!

John Skilbeck

Clojure enthusiast, obsessed with computers

Code therapist here, talk to me about your code! I am here to help. I am a backend software engineer, and use my bash/Clojure ninja skills to write fault-tolerant data processing applications; wrap in docker, and deploy on a dist...

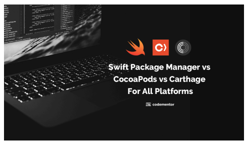
Related Articl

By using Codementor, you agree to our Cookie Policy.

Make 1+1 larger than 2. Three ways developers and data scientists can play to their strengths and compliment each other's weaknesses.

Read more

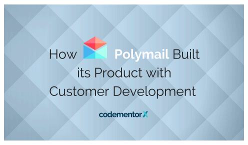
Swift Package Manager vs CocoaPods vs Carthage for All Platforms



What's the right package manager to manage your dependencies? This article compares the pros and cons of each package manager and how to use them.

Read more

How YC Alum Polymail Grew to Over 25,000 Active Users with Continuous Customer Development



How do you build the product that your users actually want? We sat down with Polymail CEO Brandon Foo to learn how they used a customer development strategy to do just that.

Read more

By using Codementor, you agree to our Cookie Policy.