**1) Docker is not just a Container**

In 2013, Docker introduced what would become the industry standard for containers. For millions of developers today, Docker is the standard way to build apps. However, Docker is much more than that command that you use on the terminal. Docker is a set of platform as a service products that uses OS-level virtualization to deliver software in packages called containers.  
  
Today, apart from Docker (the containerization tool) Docker Inc (the company) offers:

* **Docker Engine**: an open-source containerization technology for building and containerizing your applications. Available for Linux and Windows.
* **Docker Compose**: a tool for running and orchestrating containers on a single host.
* **Docker Swarm**: A toolkit for orchestrating distributed systems at any scale. It includes primitives for node discovery, raft-based consensus, task scheduling and more
* **Docker Desktop**: tools to run Docker on Windows and Mac.
* **Docker Hub**: the public container registry
* **Docker Registry**: server-side application that stores and lets you distribute Docker images
* **Docker Desktop Enterprise**: offers enterprise tools for the desktop
* **Docker Enterprise**: a full set of tools for enterprise customers.
* **Docker Universal Control Plane (UCP)**: a cluster management solution
* **Docker Kubernetes Service**: a full Kubernetes orchestration feature set
* **Security Scans**: available on Docker Enterprise.

**2) The second most loved platform**

|  |
| --- |
|  |
| **Source**: [StackOverflow's 2019 survey](https://insights.stackoverflow.com/survey/2019#technology-_-most-loved-dreaded-and-wanted-platforms) |

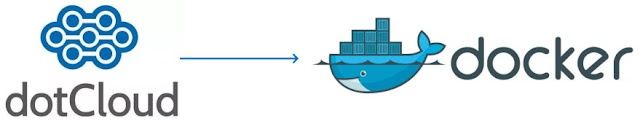
**3) Images != Containers**

A lot of people confuses this and interchangeability mix images and containers. The correct way to think about it is by using an Object-Oriented programming paradigm of Class/Instance. A Docker image is your class whereas the container is your instance. Continuing on the analogy with OO you can create multiple instances (containers) of your class (image)*.*  
  
As per [Docker themselves](https://www.docker.com/resources/what-container),

A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another. A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.

**4) Docker was once named dotCloud**

What you know today as Docker Inc., once was called [dotCloud Inc..](https://www.docker.com/blog/dotcloud-is-becoming-docker-inc/) dotCloud Inc. changed its name to Docker Inc. to grow the ecosystem by establishing Docker as a new standard for containerization, an alternative approach to virtualization which rapidly gained adoption. The project became one of the fastest-growing open source projects on [GitHub](https://github.com/docker). Surely it worked!

[](https://1.bp.blogspot.com/-2OFwJBbAdM8/Xn1_ct_YmtI/AAAAAAAACjI/dOr304poVXAqy3pb5BMFSdwF44ntNhTBQCLcBGAsYHQ/s1600/img-04.jpg)

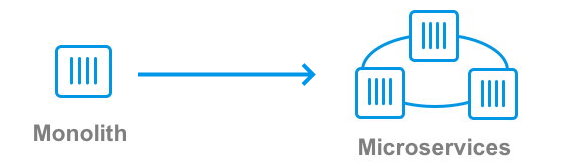
**5) Docker is neither the first nor the only tool to run containers**

Docker is neither the first nor the only tool to run containers. In fact, one of the key technologies Docker is based of, the chroot syscall was released in 1979 for Unix v7. Next, the first known container technology was [FreeBSD jails](https://www.freebsd.org/doc/handbook/jails.html) (2000), Solaris Zones (2004), [LXC](https://linuxcontainers.org/lxc/introduction/) (2008) and  [Google's mcty](https://github.com/google/lmctfy/). However, Docker made significant contributions to the segment since the establishment of [Open Container Initiative (OCI)](https://www.opencontainers.org/) in 2015. Today the open standards allow tools such as [Podman](https://podman.io/) to offer an equivalent [Docker CLI](https://docs.docker.com/engine/reference/commandline/cli/).

[](https://1.bp.blogspot.com/-loTdQyFrXAk/Xn14XKWaX-I/AAAAAAAACio/dunV4OqxC1kZYopsv9QVALIzKqc3iOyXQCLcBGAsYHQ/s1600/img-01.jpg)

**6) Containers are the new unit of deployment**

In the past, applications included tens, and on some cases, hundreds of distinct business units and a huge number of lines of code. Developing, maintaining, deploying and even scaling out those big monoliths required a huge effort. Developers were frequently frustrated that their code would fail on production but would work locally. Containers alleviate this pain as they are deployed exactly as intended, are easier to deploy and can be easily scalable.

[](https://1.bp.blogspot.com/-kTR3H1g8rQE/Xn2FxxCZ5qI/AAAAAAAACjc/Z7KY_CApmGAY0RNEvPBRqhB-cprqmEl7ACLcBGAsYHQ/s1600/img-16.jpg)

**7) Containers run everywhere**

Due to that fact that containers run on top of the container framework, they abstract the platform they're running in. That's a huge enhancement from the past where IT had to replicate the exact same setup on different environments. It also simplifies due to the fact that today you can deploy your images to your own datacenter, cloud service or even better, to a managed Kubernetes service with confidence that they'll run as they ran on your machine.

|  |
| --- |
|  |
| **Source**: [Docker - What's a Container?](https://www.docker.com/resources/what-container) |

**8) Docker Hub**

[Docker Hub](https://hub.docker.com/) is Docker's official container repository. Docker Hub is the most popular container registry in the world and one of the catalysts for the enourmous growth of Docker and containers themselves. Users and companies share their images online and everyone can download and run these images as simple as running the dock run command such as:

docker run -it alpine /bin/bash

Docker Hub is also the official repo for some of the world's most popular (and awesome!) technologies including:

[](https://1.bp.blogspot.com/-C5dGy2ZCFSY/Xn2H5sRHvyI/AAAAAAAACjo/2G8Kpnp-IbIk0_DqQ1Uf_J68_DrOd3L3QCLcBGAsYHQ/s1600/img-17.jpg)

**9) Docker Hub Alternatives**

But [Docker Hub](https://hub.docker.com/)'s not the only container registry out there. Multiple vendors including AWS, Google, Microsoft and Red Hat have their offerings. Currently, the most popular alternatives to Docker Hub are [Google Container Registry (GCR)](https://cloud.google.com/container-registry/), [Amazon Elastic Container Registry (ECR)](https://aws.amazon.com/ecr/), [Azure Container Registry (ACR)](https://azure.microsoft.com/en-us/services/container-registry/), [Quay](https://quay.io/) and [Red Hat Container Registry](https://catalog.redhat.com/software/containers/explore). All of them offer public and public repos.  
  
Speaking of private repos, Docker also has a similar offering called [Docker Trusted Registry](https://docs.docker.com/ee/dtr/). Available on [Docker Enterprise](https://docs.docker.com/ee/), you can install it on your intranet and securely store, serve and manage your company's images.

**10) Docker Images**

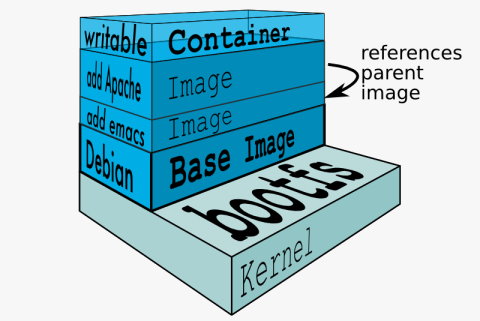
As [per Docker](https://www.docker.com/resources/what-container), an image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.   
  
There are important concepts about [images and containers](https://docs.docker.com/engine/reference/commandline/images/) that are worth repeating:

* Images are built on layers, utilizing a technology called [UnionFS (union filesystem)](https://en.wikipedia.org/wiki/UnionFS).
* Images are readonly. Modifications made by the user are stored on a separate [docker volume](https://docs.docker.com/storage/volumes/) managed by the [Docker daemon](https://docs.docker.com/config/daemon/). They are removed as soon as the container is removed.
* Images are managed using  docker image <operation> <imageid>
* An instance of an image is called a container.
* Containers are managed with the  docker container <operation> <containerid>
* You can inspect details about your image with docker image inspect <imageid>
* Images can be created with docker commit, docker build or [Dockerfiles](https://docs.docker.com/engine/reference/builder/)
* Every image has to have a base image. [scratch](https://hub.docker.com/_/scratch/) is the base empty image.
* [Dockerfiles](https://docs.docker.com/engine/reference/builder/) are templates to script images. Developed by Docker, they became the standard for the industry.
* The docker tool allows you to not only create and run images but also to create volumes, networks and much more.

For more information about how to build your images, check the [official documentation](https://docs.docker.com/engine/reference/commandline/images/).

**11) A Layered Architecture**

Docker images are a compilation of read-only layers. The below image shows an example of the multiple layers a Docker image can have. The upper layer is the writeable portion: modifications made by the user are stored on a separate [docker volume](https://docs.docker.com/storage/volumes/) managed by the Docker daemon. They are removed as soon as you remove the container.

[](https://1.bp.blogspot.com/-mQ_nmFYnMdA/Xn1O8AQuaII/AAAAAAAACic/IpzS0iIzOTQ0IPP4SUN4C-JP5mN6A63mACLcBGAsYHQ/s1600/img-13b.jpg)

**12) Volumes are your disks**

Because images and containers are readonly, and because the temporary volume created for your image is lost as soon as the image is removed the recommended way to persist the data for your container is volumes. [As per Docker](https://docs.docker.com/storage/volumes/):

Volumes are the preferred mechanism for persisting data generated by and used by Docker containers. While [bind mounts](https://docs.docker.com/storage/bind-mounts/) are dependent on the directory structure of the host machine, volumes are completely managed by Docker.

Some advantages of Volumes over bindmounts are:

* Volumes are easier to back up or migrate than bind mounts.
* You can manage volumes using Docker CLI commands or the Docker API.
* Volumes work on both Linux and Windows containers.
* Volumes can be more safely shared among multiple containers.

Creating volumes is as simple as:

docker volume create myvol

And using them with your container should be as simple as:

docker run -it -v myvol:/data alpine:latest /bin/sh

Other commands of interest for volumes are:

* docker volume inspect <vol>:  inspects the volume
* docker volume rm <vol>:  removes the volume

You can also have read-only volumes by appending :ro to your command

**13) Dockerfiles**

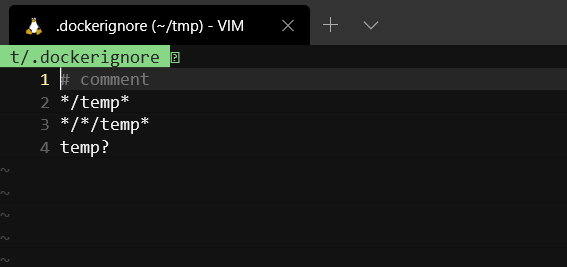
[Dockerfile](https://docs.docker.com/engine/reference/builder/) is a text file that contains all the commands a user could call on the command line to assemble an image. Using docker build users can create an automated build that executes several command-line instructions in succession.  
  
The most common commands in [Dockerfiles](https://docs.docker.com/engine/reference/builder/) are:

* FROM <image\_name>[:<tag>]: specifies the base the current image on <image\_name>
* LABEL <key>=<value> [<key>=value>...]: adds metadata to the image
* EXPOSE <port>: indicates which port should be mapped into the container
* WORKDIR <path>: sets the current directory for the following commands
* RUN <command> [ && <command>... ]: executes one or more shell commands
* ENV <name>=<value>: sets an environment variable to a specific value
* VOLUME <path>: indicates that the <path> should be externally mounted volume
* COPY <src> <dest>: copies a local file, a group of files, or a folder into the container
* ADD <src> <dest>: same as COPY but can handle URIs and local archives
* USER <user | uid>: sets the runtime context to <user> or <uid> for commands after this one
* CMD ["<path>", "<arg1>", ...]: defines the command to run when the container is started

**14 .dockerignore**

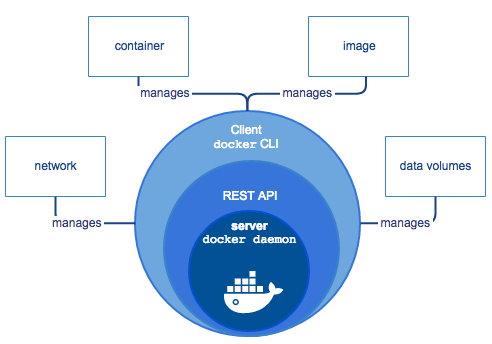
You may be familiar with [.gitignore](https://git-scm.com/docs/gitignore). Docker also accepts a [.dockerignore](https://docs.docker.com/engine/reference/builder/#dockerignore-file) file that can be used to ignore files and directories when building your image. Despite what you heard before, [Docker recommends using .dockerignore files](https://docs.docker.com/engine/reference/builder/#usage):

To increase the build’s performance, exclude files and directories by adding a .dockerignore file to the context directory.

[](https://1.bp.blogspot.com/-3FXqVRblnDc/Xn2UUd4NS2I/AAAAAAAACkI/XcihmGNVRq8w7t_QGtFJSaBm6o-jku64ACLcBGAsYHQ/s1600/img-19.jpg)

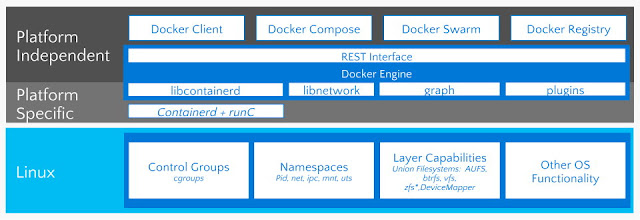
**15) Client-Server Architecture**

The Docker tool is split into two parts: a daemon with a RESTful API and a client that interacts to the daemon. The docker command you run on the command line is a frontend and essentially interacts with the daemon. The daemon is a server and a service that listens and responds to requests from the client or services authorized using the HTTP protocol. The daemon also manages your images, containers and all operations including image transfer, storage, execution, network and more.

[](https://1.bp.blogspot.com/-uR395o9Kw0g/Xn2UjoTOzdI/AAAAAAAACkM/TzTGCoysq5cBdkY85-mMBhH8GmnPkxPDgCLcBGAsYHQ/s1600/img-11.jpg)

**16) runc and containerd**

A technical overview of the internals of Docker can be seen on the below image. Since already discussed some of the technologies on the Platform layer, let's focus now on the technologies found on the platform layer: containerd and runc.

[](https://1.bp.blogspot.com/-qXxq11VzNbI/Xn5911vgXLI/AAAAAAAAClQ/rsJwCKJMjEMJLsCTN1_DhgAfkRPPhgLFwCLcBGAsYHQ/s1600/img-14.jpg)

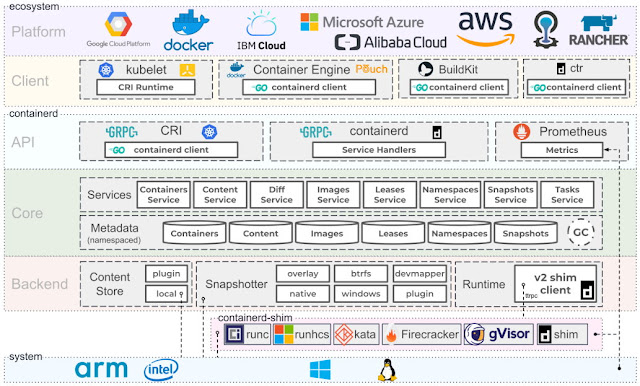
[runc](https://github.com/opencontainers/runc) is a CLI tool for spawning and running containers according to the OCI specification. runc was created from [libcontainer](https://github.com/docker/libcontainer), a library developed by Docker and [donated to the OCI](https://www.opencontainers.org/faq#faq9). libcontainer was open sourced by Docker in 2013 and donated by Docker to the [Open Container Initiative (OCI)](https://www.opencontainers.org/).  
  
[containerd](https://www.docker.com/blog/cncf-containerd-1-0-ga-announcement/) is an [open source project](https://github.com/containerd/containerd) project and the industry-standard container runtime. Developed by Docker and [donated to the CNCF](https://www.docker.com/blog/docker-donates-containerd-to-cncf/), containerd builds on top of runc, is available as a daemon for Linux and Windows and adds features, such as image transfer, storage, execution, network and more. containerd is by far the most popular container runtime and is the default runtime of Kubernetes 1.8 + and Docker.

**17) Linux kernel features**

In order to provide isolation, security and resource management, Docker relies on the following features from the [Linux Kernel](https://www.kernel.org/):

* **Union Filesystem (or UnionFS, UFS)**: UnionFS is a filesystem that allows files and directories of separate file systems to be transparently overlaid, forming a single coherent file system.
* **Namespaces**: Namespaces are a feature of the Linux kernel that partitions kernel resources so that one set of processes sees one set of resources while another set of processes sees a different set of resources. Specifically for Docker, PID, net, ipc, mnt and ufs are required.
* **Cgroups**: Cgroups allow you to allocate resources — such as CPU time, system memory, network bandwidth, or combinations of these resources — among groups of processes running on a system.
* **chroot**:  chroot changes the apparent root directory for the current running process and its children. A program that is run in such a modified environment cannot name files outside the designated directory tree.

**18) A huge Ecosystem**

The ecosystem around containers just keep growing. The image below lists *some* of the tools and services in the area.  
[](https://1.bp.blogspot.com/-djW68cJmaB8/XnpwIGDkwwI/AAAAAAAACfo/GpAsiARF-pcV2b7mSDt3VgZ49hCd1mq9QCLcBGAsYHQ/s1600/img-09.jpg)  
Today the ecosystem around containers encompasses:

* **Container Registries**: remote registries that allow you to push and share your own images.
* **Orchestration**: orchestration tools deploy, manage and monitor your microservices.
* **DNS and Service Discovery**: with containers and microservices, you'll probably need DNS and service discovery so that your services can see and talk to each onther.
* **Key-Value Stores:** provide a reliable way to store data that needs to be accessed by a distributed system or cluster.
* **Routing**: routes the communication between microservices.
* **Load Balancing**: load balancing in a distributed system is a complex problem. Consider specific tooling for your app.
* **Logging**: microservices and distributed applications will require you to rethink your logging strategy so they're available on a central location.
* **Communication Bus:** your applications will need to communicate and using a Bus is the preferred way.
* **Redundancy**: necessary to guarantee that your system can sustain load and keep operating on crashes.
* **Health Checking**: consistent health checking is necessary to guarantee all services are operating.
* **Self-healing**: microservices will fail. Self-healing is the process of redeploying them when they crash.
* **Deployments, CI, CD**: redeploying microservices is different than the traditional deployment. You'll probably have to rethink your deployments, CI and CD.
* **Monitoring**: monitoring should be centralized for distributed applications.
* **Alerting**: it's a good practice to have alerting systems on events triggered from your system.
* **Serverless**: servless technologies are also growing year over year. Today you can even find solid alternatives clouds such as [AWS](https://aws.amazon.com/), [Google Cloud](https://cloud.google.com/) and [Azure](https://azure.microsoft.com/).

**19) Containers are way more effective than VMs**

While each VM has to have their own kernel, applications, libraries and services, containers do not since they share the resources of the host. VMs are also slower to provision, deploy and restore. So since containers also provide a way to run isolated services, can be lightweight (some are only a few MBs), start quickly, are quicker to deploy and scale, containers are usually the preferred unit of scale these days.

|  |
| --- |
|  |
| **Source**: [ZDNnet](http://www.zdnet.com/article/what-is-docker-and-why-is-it-so-darn-popular/) |

**20) Containers are booming**

According to the latest [Cloud Native Computing Foundation](https://www.cncf.io/) survey, 84% of companies use containers in production with 78% using Kubernetes. The [Docker Index](https://www.docker.com/blog/introducing-the-docker-index/) also provides impressive numbers reporting more than 130 billion pulls just from Docker Hub.

|  |
| --- |
|  |
| Source: [Docker](https://docker.com) |

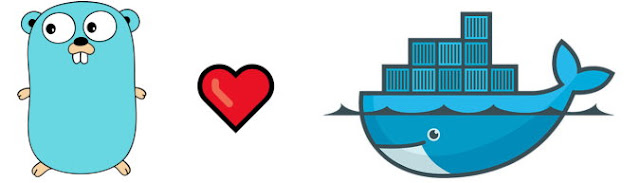
**21) Open standards**

Companies such as Amazon, IBM, Google, Microsoft and Red Hat collaborate under the [Open Container Initiative (OCI)](http://www.opencontainers.org/) which was created from [standards and technologies developed by Docker](https://www.docker.com/docker-news-and-press/industry-leaders-unite-create-project-open-container-standards) such [libcontainer.](https://github.com/docker/libcontainer) The standardization allows you to run Docker and other LXC-based containers on third-party tools such as [Podman](https://podman.io/) in any operating system.

[](https://1.bp.blogspot.com/-uvot5Cz6Z-8/Xn03uVJpMEI/AAAAAAAAChY/9EoG3iZCEY4pMPlsMAuat4OISld-EUzlQCLcBGAsYHQ/s1600/img-08.jpg)

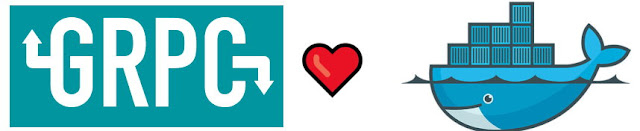
**22) Go is the container language**

You won't see this mentioned elsewhere but I'll make this bold statement: Go is the container language. Apart from kernel features (written in C) and lower level services (written in C++), most of the open-source projects in the container space use Go, including:  [runc](https://github.com/opencontainers/runc), [runtime-tools](https://github.com/opencontainers/runtime-tools), [Docker CE](https://github.com/docker/docker-ce), [containerd](https://github.com/containerd/containerd), [Kubernetes](https://github.com/kubernetes/kubernetes), [libcontainer](https://github.com/docker-archive/libcontainer), [Podman](https://github.com/containers/libpod), [Buildah](https://github.com/containers/buildah), [rkt](https://github.com/rkt/rkt), [CoreDNS](https://github.com/coredns/coredns), [LXD](https://github.com/lxc/lxd), [Prometheus](https://github.com/prometheus/prometheus), [CRI](https://github.com/containerd/cri), etc.

[](https://1.bp.blogspot.com/-ap25ViqpxDI/Xn2XOJ8VayI/AAAAAAAACkc/DxyoPvf1VagEcJ_LTu6raXJeJyb7aMBEQCLcBGAsYHQ/s1600/img-20.jpg)

**23) gRPC is the standard protocol for synchronous communication**

[gRPC](https://grpc.io/) is an open source remote procedure call system developed by Google. It uses HTTP/2 for transport, Protocol Buffers as the interface description language, and provides features such as authentication, bidirectional streaming and flow control, blocking or nonblocking bindings, and cancellation and timeouts. gRPC is also the preferred protocol when communicating between containers.

[](https://1.bp.blogspot.com/-oUtS6k1ex6I/Xn2XsD-HhsI/AAAAAAAACkk/rL7FoJ2kyvgumwugfMPelbDTvYPVdvGwQCLcBGAsYHQ/s1600/img-21.jpg)

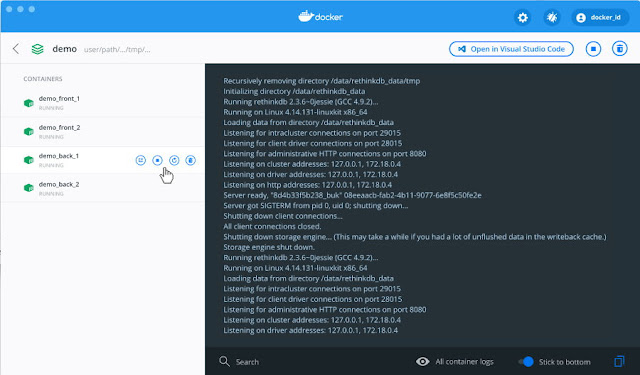
**24) Orchestration technologies emerge**

The [most deployed orchestration tool today is Kubernetes](https://www.cncf.io/blog/2020/03/04/2019-cncf-survey-results-are-here-deployments-are-growing-in-size-and-speed-as-cloud-native-adoption-becomes-mainstream/) with 78% of the market share. Kubernetes was developed at Google then donated to the the [Cloud Native Computing Foundation (CNCF)](https://www.cncf.io/). There are however other container orchestration products are [Apache Mesos](http://mesos.apache.org/), [Rancher](https://rancher.com/), [Open Shift](https://openshift.com/), [Docker Cluster on Docker Enterprise](https://docs.docker.com/ee/cluster/), and more.

**25) Kubernetes is the Container Operating System**

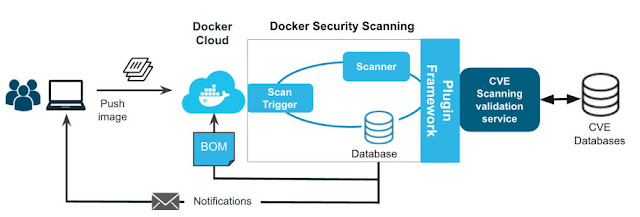
It's impossible talk Docker these days without mentioning [Kubernetes](https://kubernetes.io/). Kubernetes is an open source orchestration system for automating the management, placement, scaling and routing of containers that has become popular with developers and IT operations teams in recent years. It was first developed by Google and contributed to Open Source in 2014, and is now maintained by the Cloud Native Computing Foundation. Since version 1.9 Kubernetes uses [containerd](https://containerd.io/) sas its container runtime so it can use other container runtimes such [CRI-O](https://cri-o.io/). A container runtime is responsible for managing and running the individual containers of a pod.

Today [Kubernetes is embedded with Docker Desktop](https://www.docker.com/products/kubernetes) so developers can develop Docker and Kubernetes at the comfort of their desktops. Plus, because Docker containers implement the OCI specification, you can build your containers using a tools such as [Buildah](https://buildah.io/) / [LXD](https://linuxcontainers.org/lxd/) and run in on Kubernetes.

[](https://1.bp.blogspot.com/-VkDMVRz_w_k/Xn6MV_M-IoI/AAAAAAAAClo/6EAurtyIFss9-pq6EToTpO6b01se4PM8QCLcBGAsYHQ/s1600/img-22.jpg)

**26) Security Scans**

Docker also offers a automated scans via [Docker Enterprise Platform](https://docs.docker.com/ee/). Because images are readonly, automated scans are simple as it becomes simply checking the sha of your image. The image below details how this happens. More information is [available here](https://docs.docker.com/ee/dtr/user/manage-images/scan-images-for-vulnerabilities/).

[](https://1.bp.blogspot.com/--2YwCF1z-ww/Xn2L42kZqJI/AAAAAAAACj8/TdDYeFhv-uMUGkxZzxW2Z0UchXl2EFLXgCLcBGAsYHQ/s1600/img-18.jpg)

**27) Windows also has native container**

As a Windows user, you might already be aware that there exists so-called Windows containers that run natively on Windows. And you are right. Recently, Microsoft has ported the Docker engine to Windows and it is now possible to run Windows containers directly on a Windows Server 2016 without the need for a VM. So, now we have two flavors of containers, Linux containers and Windows containers. The former only run on Linux host and the latter only run on a Windows Server. In this book, we are exclusively discussing Linux containers, but most of the things we learn also apply to Windows containers.   
  
Today you can [run Windows-based or Linux-based containers on Windows 10](https://docs.microsoft.com/en-us/virtualization/windowscontainers/about/) for development and testing using [Docker Desktop](https://store.docker.com/editions/community/docker-ce-desktop-windows), which makes use of containers functionality built-in to Windows. You can also [run containers natively on Windows Server](https://docs.microsoft.com/en-us/virtualization/windowscontainers/quick-start/set-up-environment?tabs=Windows-Server).

**28) Service meshes**

**Separating logic of maincar and sidecar containers.**

More recently, the trend goes towards a service mesh. This is the new buzz word. As we containerize more and more applications, and as we refactor those applications into more microservice-oriented applications, we run into problems that simple orchestration software cannot solve anymore in a reliable and scalable way. Topics in this area are service discovery, monitoring, tracing, and log aggregation. Many new projects have emerged in this area, the most popular one at this time being Istio, which is also part of the CNCF.

**Conclusion**

On this post we reviewed 28 about Docker everyone should know. Hope this article was interesting and that you learned something new today. Docker is a fantastic tool and given it's popularity, it's ecosystem will only grow bigger thus, it's important to learn it well and understand its internals.