Containers

Containers - Introduction

- Virtualization helps to share resources among many customers in cloud computing.
- Container is a lightweight virtualization technique.
- Container packages the code and all its dependencies so the application runs quickly and reliably from one computing environment to another.
- *Docker* is an open platform for developing, shipping, and running applications.
- *Kubernetes* is an open-source system for automating deployment, scaling, and management of containerized applications.

Containers - Introduction

- Containers are packages of software that contain all of the necessary elements to run in any environment.
- Containers virtualize the operating system and run anywhere, from a private data center to the public cloud or even on a developer's personal laptop.
- *Containers are lightweight packages of the application code together with dependencies such as specific versions of programming language runtimes and libraries required to run the software services.*
- Containers make it easy to share CPU, memory, storage, and network resources at the operating systems level and offer a logical packaging mechanism in which applications can be abstracted from the environment in which they actually run.

Ref: https://cloud.google.com/learn/what-are-containers

Containers - Needs

- Containers offer a logical packaging mechanism in which applications can be abstracted from the environment in which they actually run.
- This decoupling allows container-based applications to be deployed easily and consistently, regardless of whether the target environment is a private data center, the public cloud, or even a developer's personal laptop.
- *Agile development*: Containers allow the developers to move much more quickly by avoiding concerns about dependencies and environments.
- *Efficient operations*: Containers are lightweight and allow to use just the computing resources one need – thus running the applications efficiently.
- *Run anywhere*: Containers are able to run virtually anywhere. .

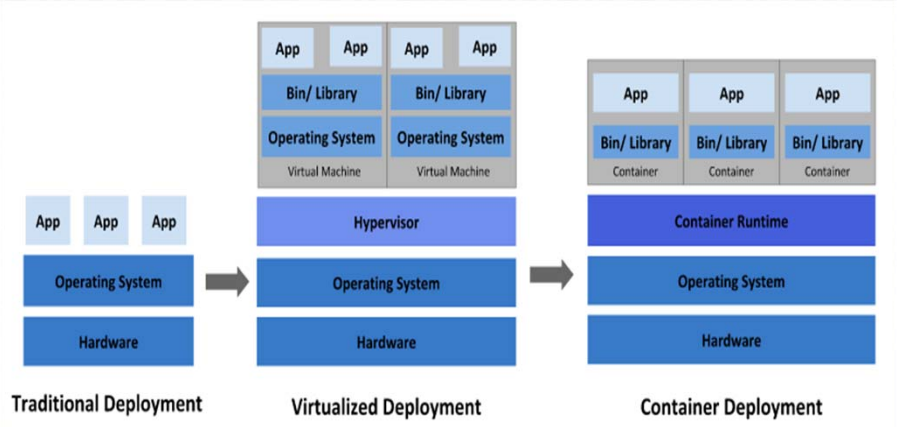
Ref: https://cloud.google.com/learn/what-are-containers

Containers – Major Benefits

- **Separation of responsibility**: Containerization provides a clear separation of responsibility, as developers focus on application logic and dependencies, while IT operations teams can focus on deployment and management instead of application details such as specific software versions and configurations.
- **Workload portability**: Containers can run virtually anywhere, greatly easing development and deployment: on Linux, Windows, and Mac operating systems; on virtual machines or on physical servers; on a developer's machine or in data centers on-premises; and of course, in the public cloud.
- **Application isolation**: Containers virtualize CPU, memory, storage, and network resources at the operating system level, providing developers with a view of the OS logically isolated from other applications.

Ref: https://cloud.google.com/learn/what-are-containers

Application Deployment



Ref: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

Traditional – Virtualized – Container Deployments

- **Traditional deployment :** Applications run on physical servers. There was no way to define resource boundaries for applications in a physical server, and this caused resource allocation issues.
- **Virtualized deployment :** Allows to run multiple Virtual Machines (VMs) on a single physical server's CPU. Virtualization allows applications to be isolated between VMs. It allows better utilization of resources in a physical server and allows better scalability. Each VM is a full machine running all the components, including its own operating system, on top of the virtualized hardware.
- **Container deployment:** Containers are similar to VMs, but they have relaxed isolation properties to share the Operating System (OS) among the applications. Therefore, containers are considered lightweight. A container has its own filesystem, share of CPU, memory, process space, and more. As containers are decoupled from the underlying infrastructure, they are portable across clouds and different OS distributions.

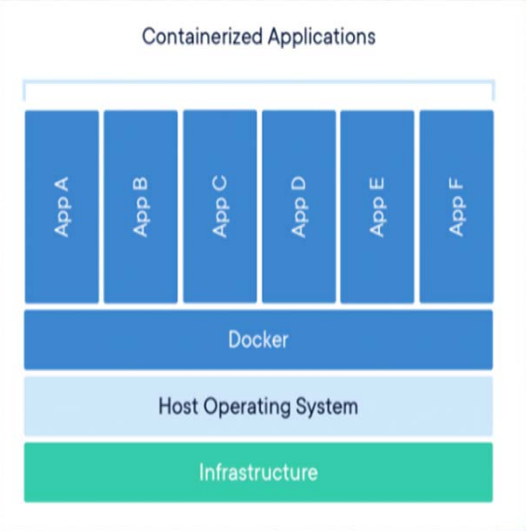

Containers and VMs

- VMs: a guest operating system such as Linux or Windows runs on top of a host operating system with access to the underlying hardware.
- Containers are often compared to virtual machines (VMs). Like virtual machines, containers allow one to package the application together with libraries and other dependencies, providing isolated environments for running your software services.
- However, the containers offer a far more lightweight unit for developers and IT Ops teams to work with, carrying a myriad of benefits.

- ➢ Containers are much more lightweight than VMs
- ➢ Containers virtualize at the OS level while VMs virtualize at the hardware level
- ➢ Containers share the OS kernel and use a fraction of the memory VMs require


Container

- A container is a sandboxed process that is isolated from all other processes on the host machine.
- A container is a runnable instance of an image.
- One can create, start, stop, move, or delete a container using the API (e.g. DockerAPI) or CLI.
- A container can be run on local machines, virtual machines or deployed to the cloud.



Ref: https://www.docker.com/resources/what-container

Kubernetes

- Kubernetes is a portable, extensible, open-source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation. It has a large, rapidly growing ecosystem. Kubernetes services, support, and tools are widely available.
- The name Kubernetes originates from Greek, meaning helmsman or pilot.
- Kubernetes operates at the container level rather than at the hardware level, it provides some generally applicable features common to PaaS offerings, such as deployment, scaling, load balancing, and lets users integrate their logging, monitoring, and alerting solutions.
- However, Kubernetes is not monolithic, and these default solutions are optional and pluggable.
- Kubernetes provides the building blocks for building developer platforms, but preserves user choice and flexibility where it is important.

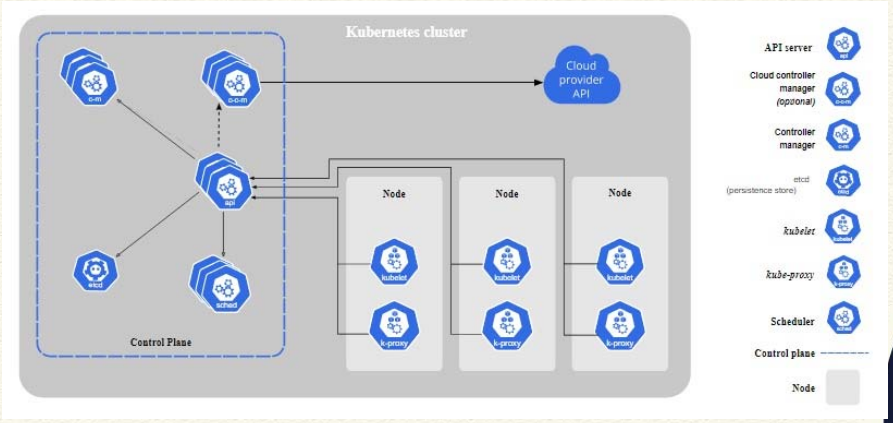
Ref: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

Kubernetes Components

- A Kubernetes cluster consists of a set of worker machines, called **nodes**, that run containerized applications. Every cluster has at least one worker node.
- The worker node(s) host the **Pods** that are the components of the application workload.
- The **control plane** manages the worker nodes and the Pods in the cluster.
- In production environments, the control plane usually runs across multiple computers and a cluster usually runs multiple nodes, providing fault-tolerance and high availability.

Ref: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

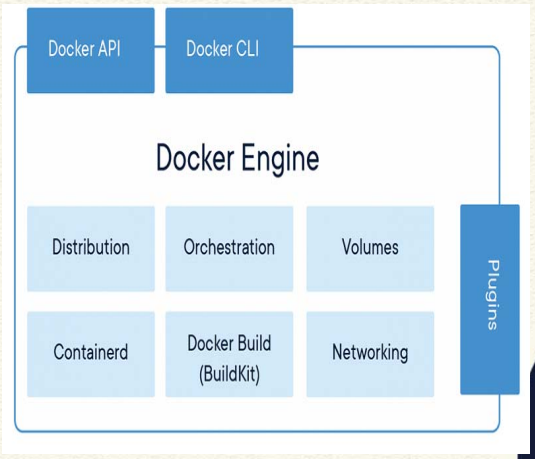
Kubernetes Cluster Components



Ref: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

Docker Engine

- Docker containers that run on Docker Engine:
- **Standard:** Docker created the industry standard for containers, so they could be portable anywhere
- **Lightweight:** Containers share the machine's OS system kernel and therefore do not require an OS per application, driving higher server efficiencies and reducing server and licensing costs
- **Secure:** Applications are safer in containers and Docker provides the strongest default isolation capabilities in the industry



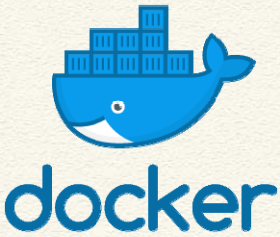
Ref: https://www.docker.com/resources/what-container

Dockers

- 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.
- Container images become containers at runtime and in the case of Docker containers - images become containers when they run on **Docker Engine**.
- Available for both Linux and Windows-based applications, containerized software will always run the same, regardless of the infrastructure. Containers isolate software from its environment and ensure that it works uniformly despite differences for instance between development and staging.

Ref: https://www.docker.com/resources/what-container

Docker



https://www.docker.com/

Docker - Overview

- Docker is a platform that allows you to “build, ship, and run any app, anywhere.”
- Considered to be a standard way of solving one of the challenging aspects of software: deployment.
- Traditionally, the development pipeline typically involved combinations of various technologies for managing the movement of software, such as virtual machines, configuration management tools, package management systems, and complex webs of library dependencies.
  - All these tools needed to be managed and maintained by specialist engineers, and most had their own unique ways of being configured.

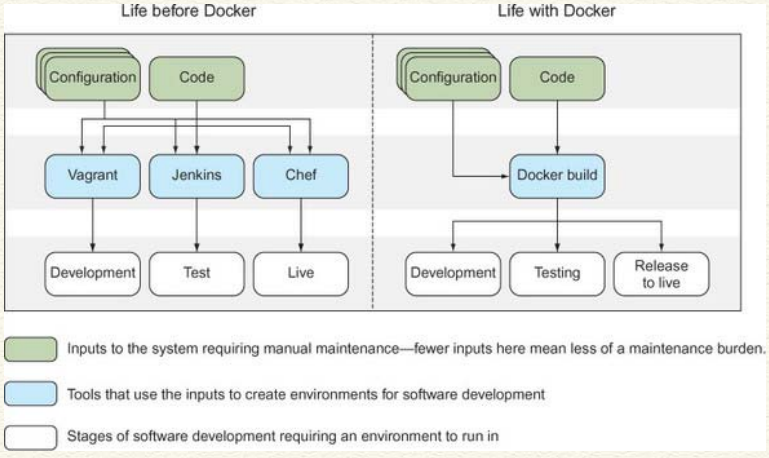
Ref: Docker in Practice, Second Edition, Ian Miell and Aidan Hobson Sayers, February 2019, ISBN 9781617294808

Docker - Overview

- Docker has changed the traditional approach - Everything goes through a common pipeline to a single output that can be used on any target—there's no need to continue maintaining a bewildering array of tool configurations
- At the same time, there's no need to throw away the existing software stack if it works for you—you can package it up in a Docker container as-is, for others to consume.
- Addionally, you can see how these containers were built, so if you need to dig into the details, you can.

Ref: Docker in Practice, Second Edition, Ian Miell and Aidan Hobson Sayers, February 2019, ISBN 9781617294808

Docker – Big Picture



Ref: Docker in Practice, Second Edition, Ian Miell and Aidan Hobson Sayers, February 2019, ISBN 9781617294808

Docker - Analogy

- *Analogy:* Tradionally, a docker was a laborer who moved commercial goods into and out of ships when they docked at ports. There were boxes and items of differing sizes and shapes, and experienced dockers were prized for their ability to fit goods into ships by hand in cost-effective ways. Hiring people to move stuff around wasn't cheap, but there was no alternative!
- This may sound familiar to anyone working in software. Much time and intellectual energy is spent getting metaphorically odd-shaped software into differently-sized metaphorical ships full of other odd-shaped software, so they can be sold to users or businesses elsewhere.

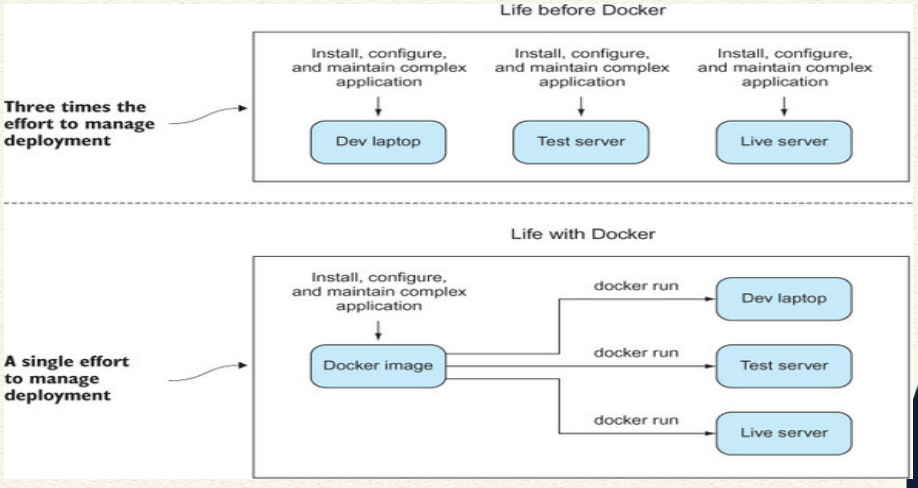
Ref: Docker in Practice, Second Edition, Ian Miell and Aidan Hobson Sayers, February 2019, ISBN 9781617294808

Docker - Benefit

- Before Docker, deploying software to different environments required significant effort. Even if you weren't hand-running scripts to provision software on different machines (and plenty of people do exactly that), you'd still have to handle the configuration management tools that manage state on what are increasingly fast-moving environments starved of resources.
- Even when these efforts were encapsulated in VMs, a lot of time was spent managing the deployment of these VMs, waiting for them to boot, and managing the overhead of resource use they created.

Ref: Docker in Practice, Second Edition, Ian Miell and Aidan Hobson Sayers, February 2019, ISBN 9781617294808

Docker - Benefit



Ref: Docker in Practice, Second Edition, Ian Miell and Aidan Hobson Sayers, February 2019, ISBN 9781617294808

Docker - Benefit

- With Docker, the configuration effort is separated from the resource management, and the deployment effort is trivial:
  - run docker, and the environment's image is pulled down and ready to run, consuming fewer resources and contained so that it doesn't interfere with other environments.
- You don't need to worry about whether your container is going to be shipped to a Red Hat machine, an Ubuntu machine, or a CentOS VM image; as long as it has Docker on it, it will run

Ref: Docker in Practice, Second Edition, Ian Miell and Aidan Hobson Sayers, February 2019, ISBN 9781617294808

Docker - Advantage

- **Replacing virtual machines (VMs)**: Docker can be used to replace VMs in many situations. If you only care about the application, not the operating system, Docker can replace the VM.
  - Not only is Docker quicker than a VM to spin up, it's more lightweight to move around, and due to its layered filesystem, you can more easily and quickly share changes with others. It's also rooted in the command line and is scriptable.

- **Prototyping software**: If you want to quickly experiment with software without either disrupting your existing setup or going through the hassle of provisioning a VM, Docker can give you a sandbox environment almost instantly. .

Ref: Docker in Practice, Second Edition, Ian Miell and Aidan Hobson Sayers, February 2019, ISBN 9781617294808

Docker - Advantage

- **Packaging software**: Because a Docker image has effectively no dependencies, it's a great way to package software. You can build your image and be sure that it can run on any modern Linux machine—think Java, without the need for a JVM.

- **Enabling a Microservices architecture**: Docker facilitates the decomposition of a complex system to a series of composable parts, which allows you to reason about your services in a more discrete way. This can allow you to restructure your software to make its parts more manageable and pluggable without affecting the whole.

Ref: Docker in Practice, Second Edition, Ian Miell and Aidan Hobson Sayers, February 2019, ISBN 9781617294808

Docker - Advantage

- **Modeling networks:** Several hundreds (even thousands) of isolated containers can be initiated on one machine, modeling a network can be done efficiently. .
- **Enabling full-stack productivity when offline** - All the parts of the system can be bundled into Docker containers, you can orchestrate these to run on your laptop and work on the move, even when offline.

Ref: Docker in Practice, Second Edition, Ian Miell and Aidan Hobson Sayers, February 2019, ISBN 9781617294808

Docker - Advantage

- **Reducing debugging overhead**: Complex negotiations between different teams about software delivered is a commonplace within the industry.
  - Docker allows you to state clearly (even in script form) the steps for debugging a problem on a system with known properties, making bug and environment reproduction a much simpler affair, and one normally separated from the host environment provided.
- **Documenting software dependencies**: By building your images in a structured way, ready to be moved to different environments, Docker forces you to document your software dependencies explicitly from a base starting point..

Ref: Docker in Practice, Second Edition, Ian Miell and Aidan Hobson Sayers, February 2019, ISBN 9781617294808

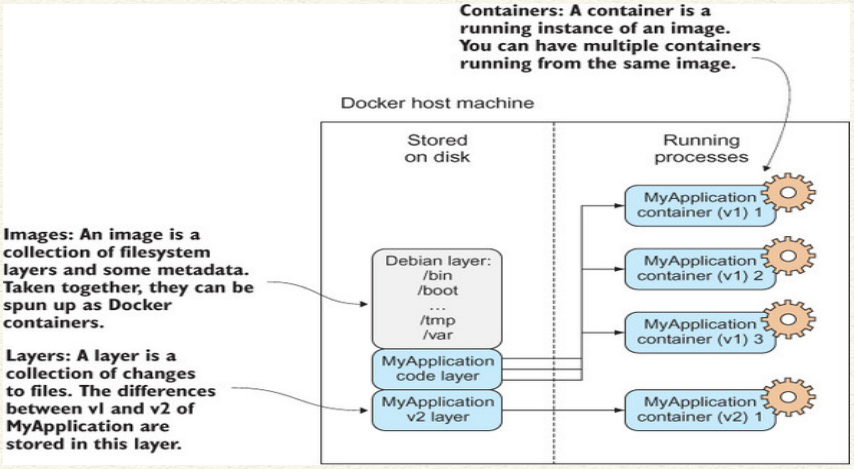
Docker - Advantage

- **Enabling continuous delivery**: Continuous delivery (CD) is a paradigm for software delivery based on a pipeline that rebuilds the system on every change and then delivers to production (or "live") through an automated (or partly automated) process.

➢ Docker builds are more reproducible and replicable than traditional software building methods. This makes implementing Continuous delivery (CD) much easier.

Ref: Docker in Practice, Second Edition, Ian Miell and Aidan Hobson Sayers, February 2019, ISBN 9781617294808

Docker – Key Concepts



Ref: Docker in Practice, Second Edition, Ian Miell and Aidan Hobson Sayers, February 2019, ISBN 9781617294808

Docker – Key Commands

Command	Purpose
`docker build`	Build a Docker image
`docker run`	Run a Docker image as a container
`docker commit`	Commit a Docker container as an image
`docker tag`	Tag a Docker image

Ref: Docker in Practice, Second Edition, Ian Miell and Aidan Hobson Sayers, February 2019, ISBN 9781617294808

Docker – Architecture

- Docker on your host machine is split into two parts—a daemon with a RESTful API and a client that talks to the daemon.
- The private Docker registry is a service that stores Docker images. These can be requested from any Docker daemon that has the relevant access. This registry is on an internal network and isn't publicly accessible, so it's considered private.

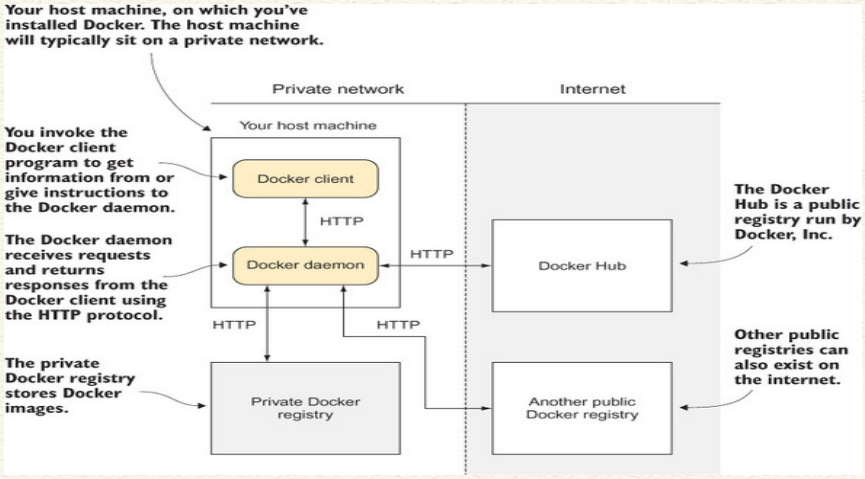
Ref: Docker in Practice, Second Edition, Ian Miell and Aidan Hobson Sayers, February 2019, ISBN 9781617294808

Docker – Architecture

- One invokes the Docker client to get information from or give instructions to the daemon; the daemon is a server that receives requests and returns responses from the client using the HTTP protocol.
- In turn, it will make requests to other services to send and receive images, also using the HTTP protocol.
- The server will accept requests from the command-line client or anyone else authorized to connect.
- The daemon is also responsible for taking care of your images and containers behind the scenes, whereas the client acts as the intermediary between you and the RESTful API.

Ref: Docker in Practice, Second Edition, Ian Miell and Aidan Hobson Sayers, February 2019, ISBN 9781617294808

Docker – Architecture



Ref: Docker in Practice, Second Edition, Ian Miell and Aidan Hobson Sayers, February 2019, ISBN 9781617294808

REFERENCES

- https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/
- https://cloud.google.com/kubernetes-engine/docs/concepts/verticalpodautoscaler
- https://www.docker.com/
- https://www.docker.com/resources/what-container
- Docker in Practice, Second Edition, Ian Miell and Aidan Hobson Sayers, February 2019, ISBN 9781617294808