Full Stack AI Projects

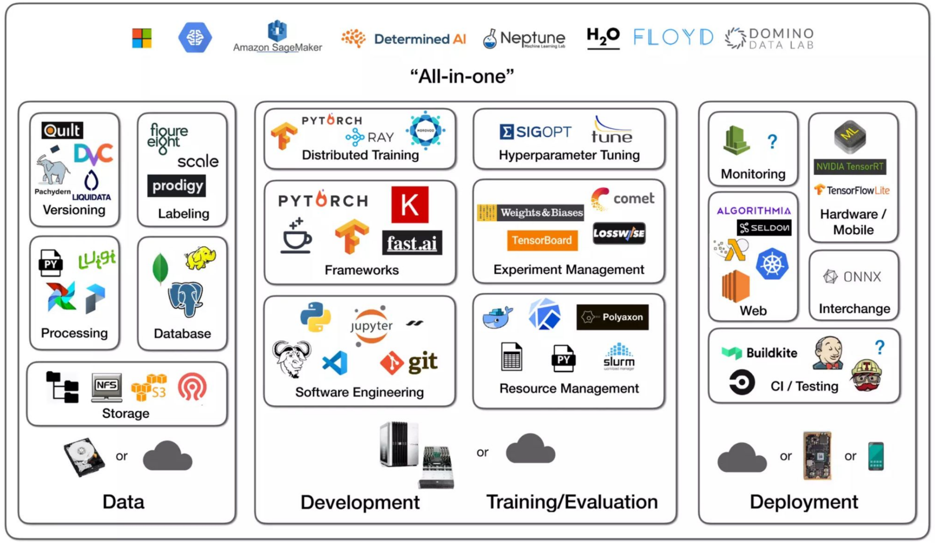
Infrastructure and tooling

Henry Ruiz
GDE ML
@devharuiz
https://haruiz.github.io/

Infrastructure & Tooling

The number of **available tools to work with ML seems endless.**

Selecting the appropriate tools depends on:
the kind of problem, **<u>type of solution, deployment scenario, capacity building</u>**, team experience, cost, hardware and software infrastructure, etc.



amazon web services™
Google Cloud Platform
Azure
IBM Watson™
FLOYD
scikit learn
TensorFlow
PyTorch
DASK
Numba
NumPy
Pandas
astropy
gprlib
OPTUNA
SciPy
OpenCV

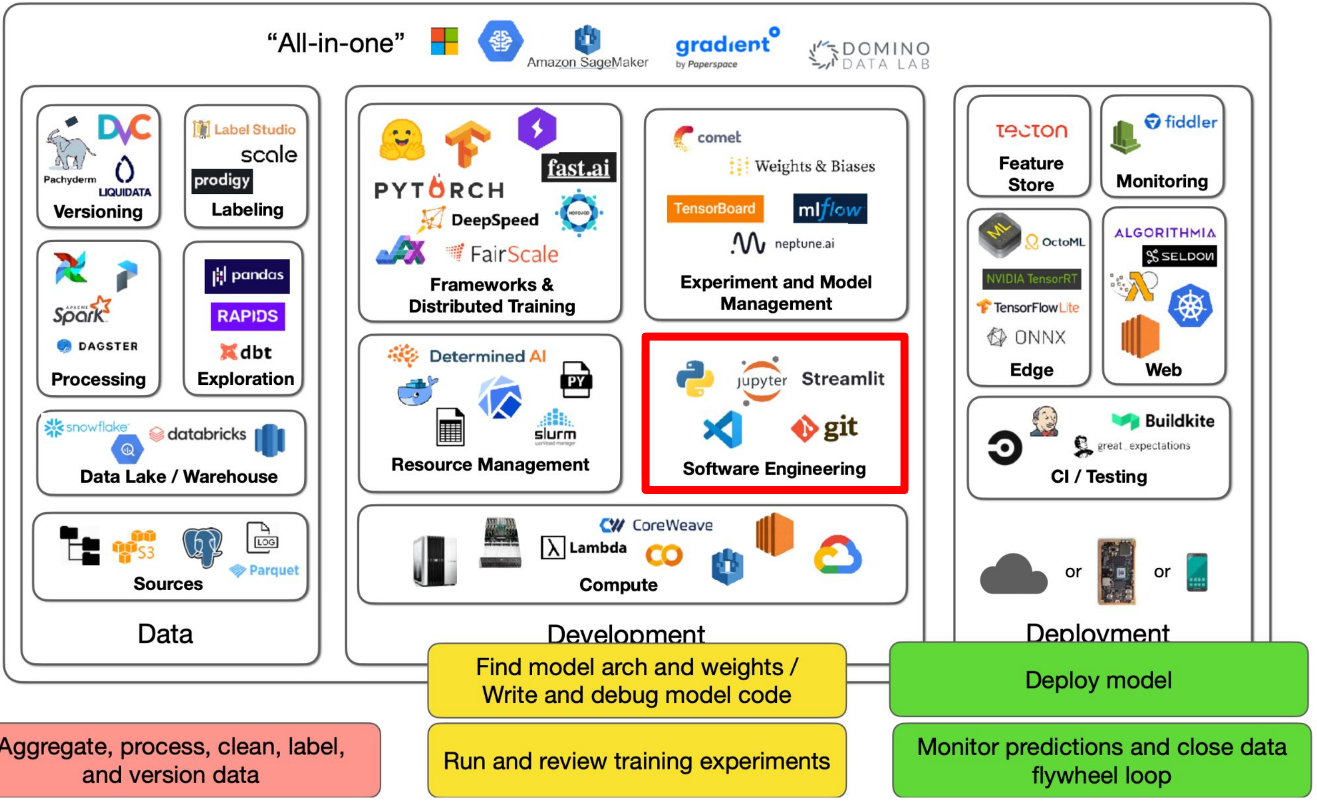
Introduction

The **dream** of ML development is that given a project spec and some sample data, you get a continually improving prediction system deployed at scale.

The **reality** is starkly different:

- You have to collect, aggregate, process, clean, label, and version the data.
- You have to find the model architecture and their pre-trained weights and then write and debug the model code.
- You run training experiments and review the results, which will be fed back into the process of trying out new architectures and debugging more code.
- You can now deploy the model.
- After model deployment, you have to monitor model predictions and close the data flywheel loop. Basically, your users generate fresh data for you, which needs to be added to the training set.

FullStack ML Engineering Tools



Isolating python dependencies using Pyenv and Poetry

Pyenv

- Pyenv is a simple **Python version management tool.**
- It allows users to easily switch between different versions of Python on the same system.
- Pyenv can be used to manage both system-wide and per-user installations of Python.
- It can be used to install different versions of Python, and to set the global and local versions of Python for a specific project.

Poetry

- Poetry is a **package and dependency manager for Python.**
- It provides a simple and convenient way to manage dependencies, packages, and virtual environments for a Python project.
- Poetry can be used to create new projects, to add and manage dependencies, and to build and publish packages.
- It provides a consistent and repeatable way to manage dependencies, making it easier to test and deploy Python projects.
- Poetry uses a pyproject.toml file to manage the dependencies and configuration of a Python project, and it integrates with Pyenv to provide an easy way to manage multiple Python versions and virtual environments.

Other tools exist….

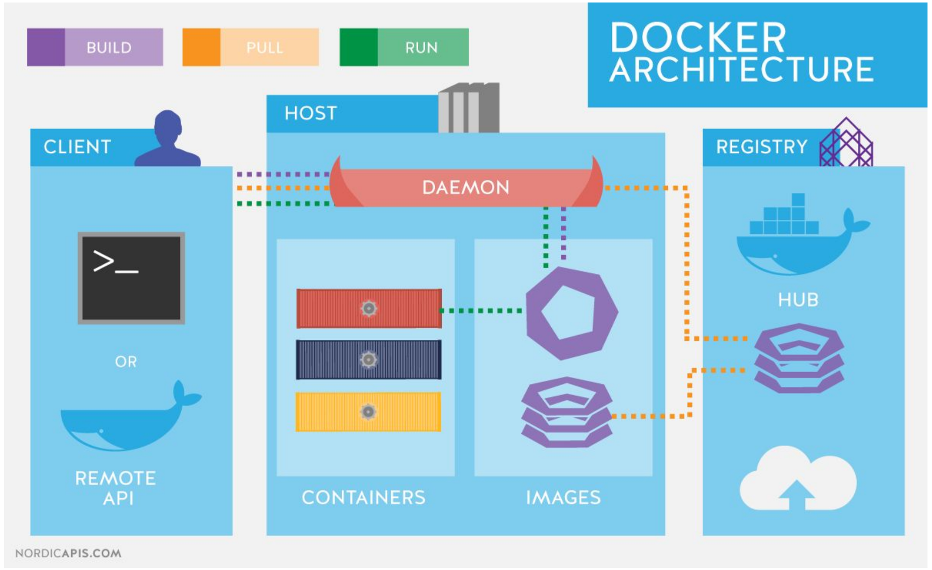
Isolating App dependencies using Docker

Docker was introduced to the world by Solomon Hykes, founder and CEO of a company called dotCloud.

It provides a platform for building, shipping, and running distributed applications.

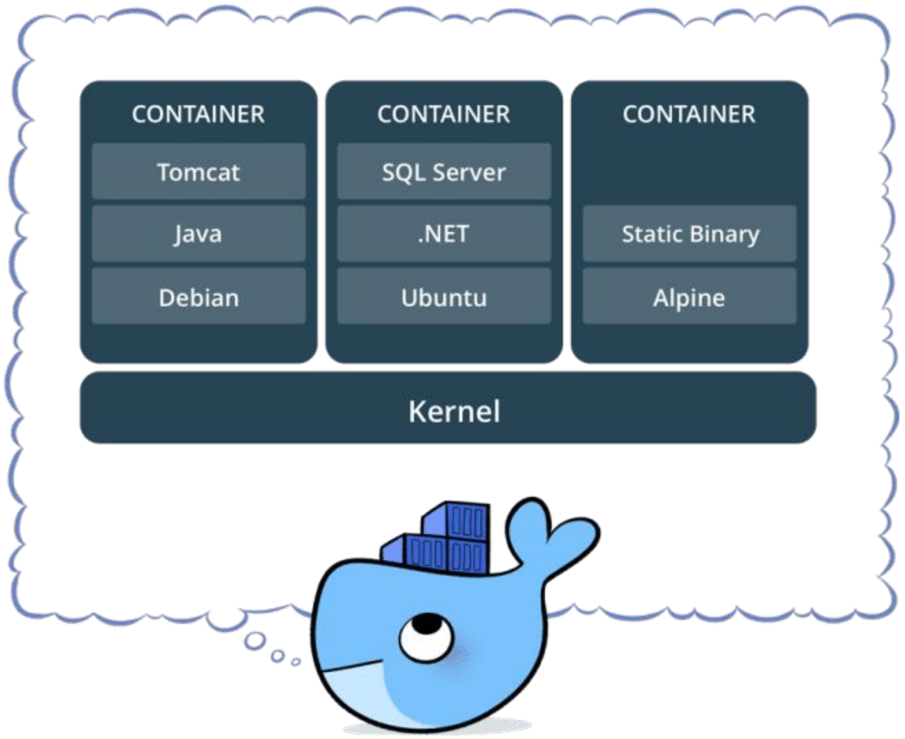
Docker introduce the concept of "containers" to package software into isolated environments that can run on any system with the Docker engine installed.

This makes it easy to run the same application in different environments, such as development, testing, and production, without worrying about dependencies, configurations, or compatibility issues.



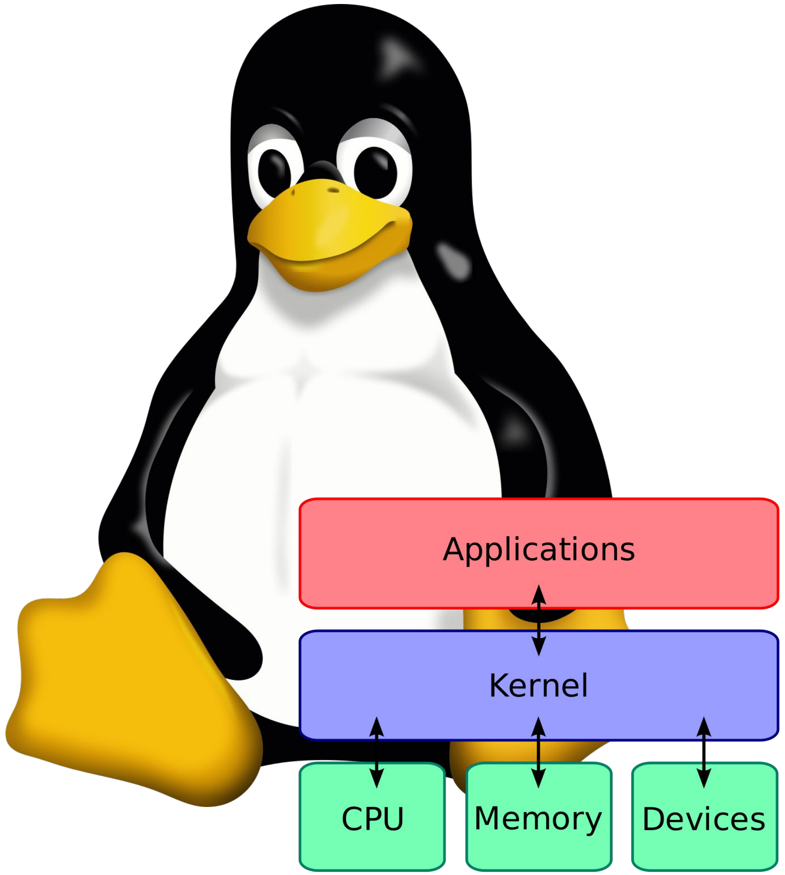
Does docker use linux kernel under the hood?

Yes, Docker uses **the Linux kernel under the hood**. **Docker containers run as isolated processes on the host system, and they use the host system's Linux kernel to interact with the underlying hardware.** This is because **Docker was originally designed to run on Linux**, and it leverages many features of the Linux kernel to provide the isolation and security that containers need.



Linux kernel

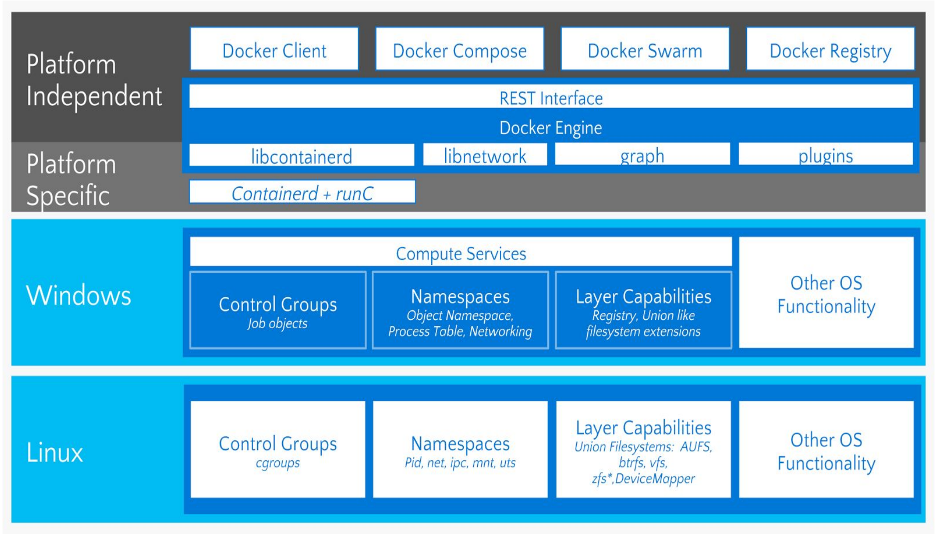
The Linux kernel is the main component of the Linux operating system (OS). It's a computer program that acts as the interface between a computer's hardware and its processes. The kernel manages resources as efficiently as possible and enables communication between applications and hardware.



Does docker use linux kernel under the hood?

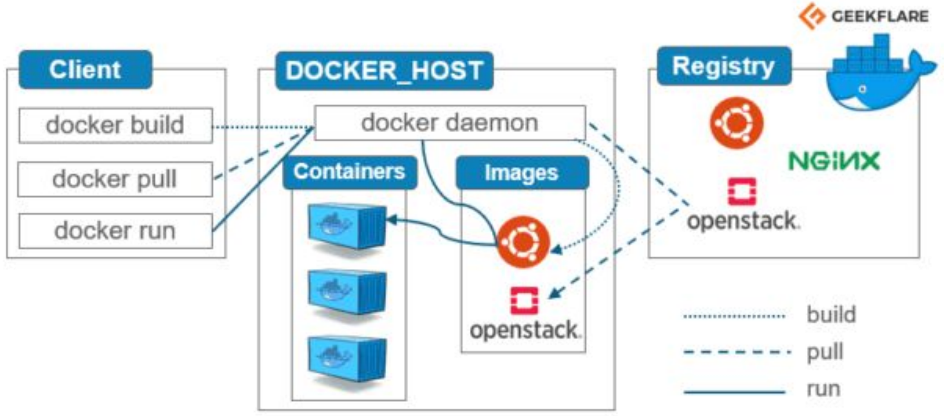
However, **Docker has since been made to run on other operating systems as well, including Windows and macOS**. In these cases, *Docker uses a lightweight Linux virtual machine to run the containers*, which provides the necessary compatibility with the Linux-based Docker containers.

So while Docker can run on multiple operating systems, it still "relies on the Linux kernel" to provide its core containerization features. This is why many people consider Docker to be a Linux-based technology, even when it's running on other operating systems.



Docker under the hood

Docker containers are lightweight and fast, as **they share the host system's kernel and only include the necessary files and libraries for the application to run**. This makes it possible to run many containers on the same system, making it an efficient solution for scaling and deploying applications.

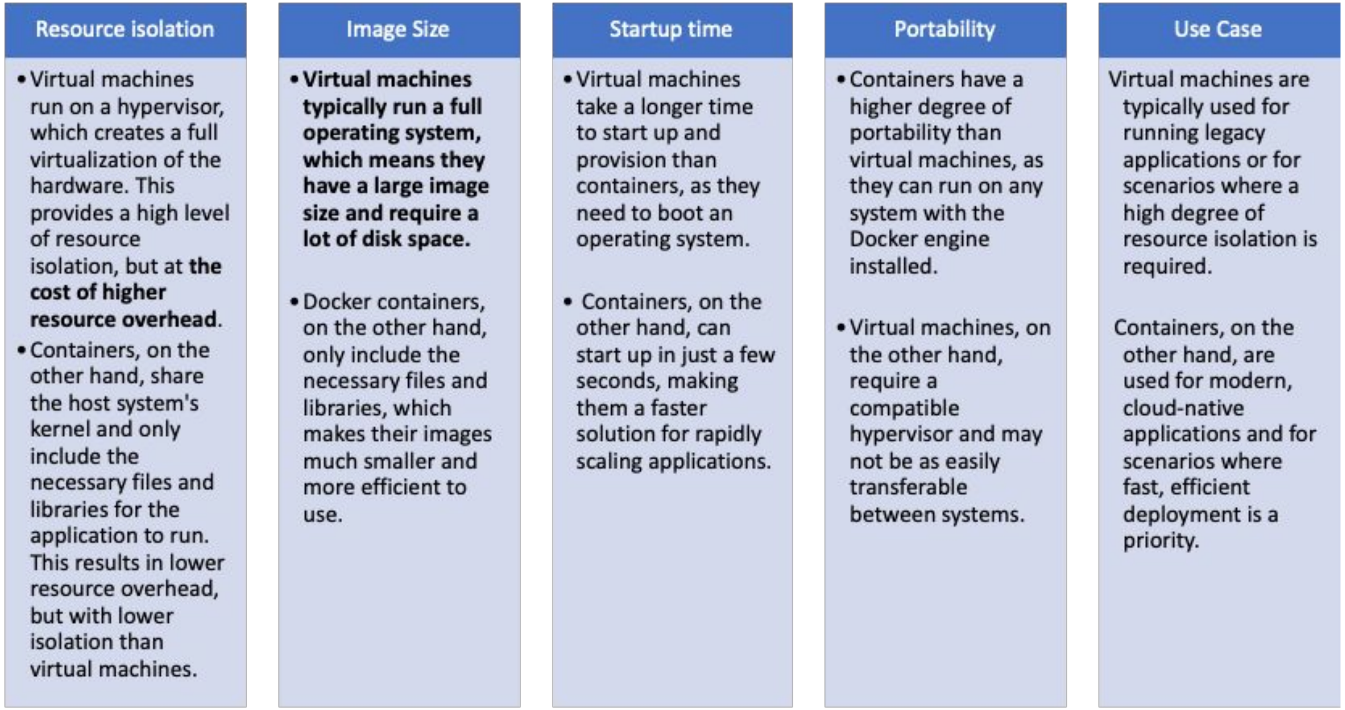


Docker components

- **Docker Engine:** This is the core component of Docker that allows you to **create and run Docker containers**. It is responsible for managing the Docker environment, including creating and starting containers, networking, and storage.
- **Docker Image:** A Docker Image is **a lightweight, stand-alone, executable package that includes everything needed to run a piece of software, including the code, a runtime, libraries, environment variables, and config files.**
- **Docker Container:** A Docker Container is a **running instance of a Docker Image**. It is an isolated environment that includes the software and all its dependencies. Containers can be started, stopped, and moved between hosts with ease.
- **Docker Hub:** This is a **cloud-based repository for storing and sharing Docker Images**. It provides a centralized location for developers to store and distribute their images, making it easy for others to find, download, and use them.
- **Docker Compose:** This is a **tool for defining and running multi-container Docker applications**. With Docker Compose, you can define your application's services, networks, and volumes in a single file, and then start and stop all services from this file.

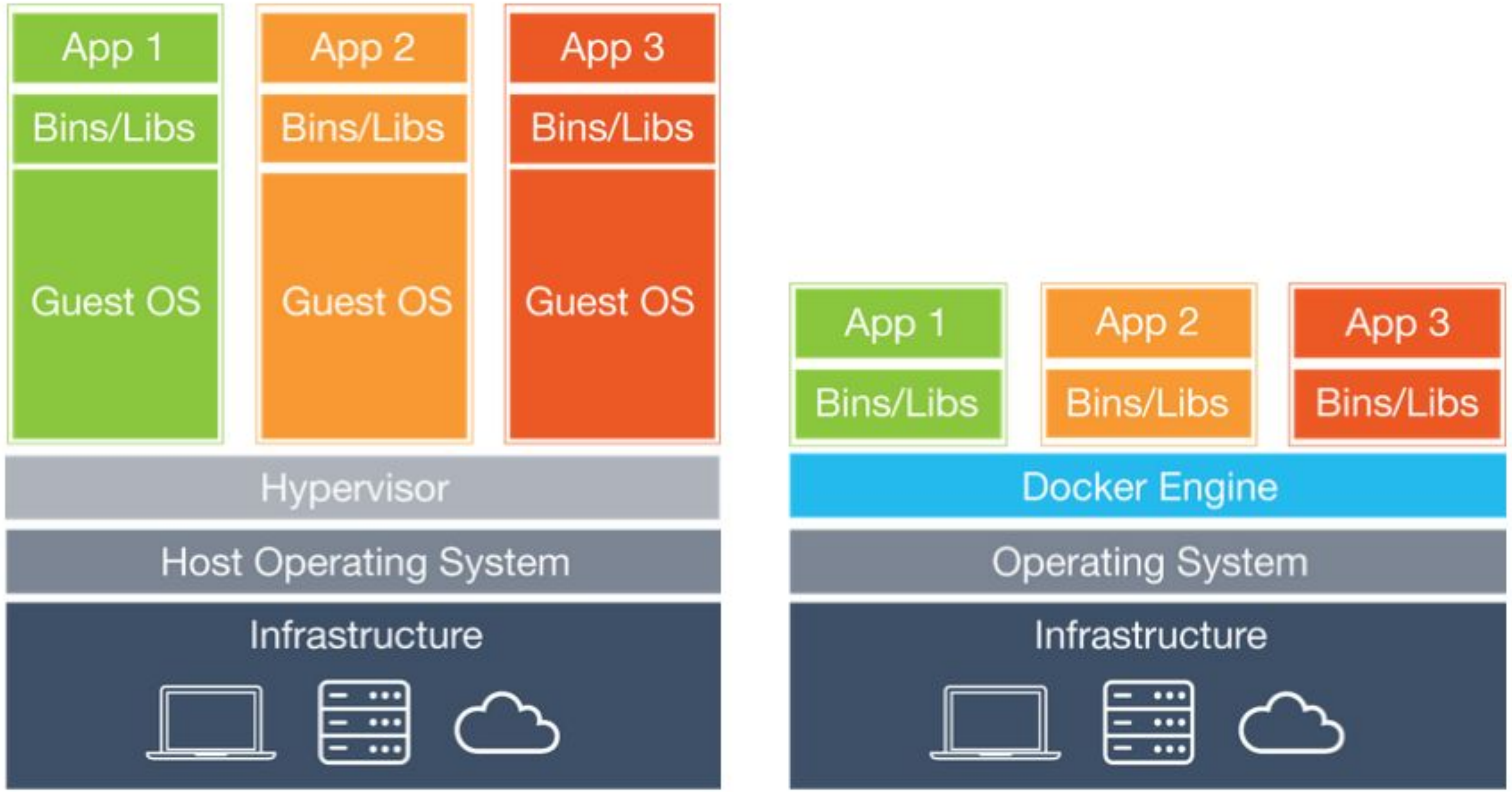
Docker vs Virtual Machines

Docker containers and virtual machines are both technologies for running multiple isolated applications on the same physical host. However, there are several key differences between the two:



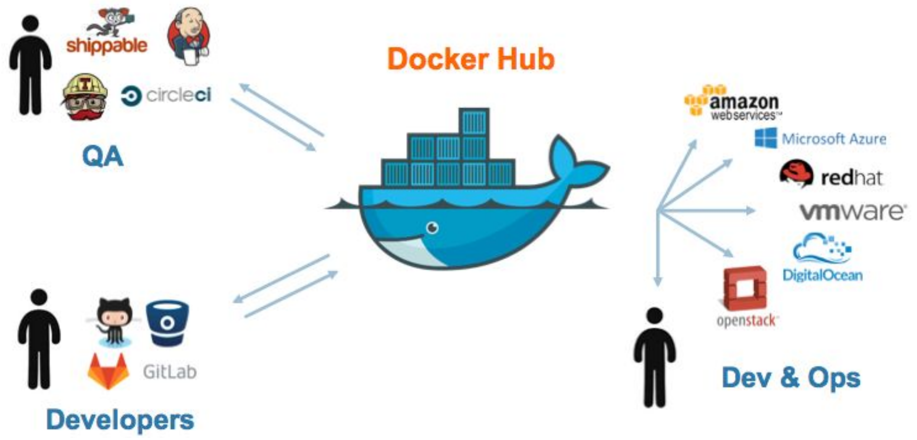
Resource isolation	Image Size	Startup time	Portability	Use Case
• Virtual machines run on a hypervisor, which creates a full virtualization of the hardware. This provides a high level of resource isolation, but at **the cost of higher resource overhead**. • Containers, on the other hand, share the host system's kernel and only include the necessary files and libraries for the application to run. This results in lower resource overhead, but with lower isolation than virtual machines.	• **Virtual machines typically run a full operating system, which means they have a large image size and require a lot of disk space.** • Docker containers, on the other hand, only include the necessary files and libraries, which makes their images much smaller and more efficient to use.	• Virtual machines take a longer time to start up and provision than containers, as they need to boot an operating system. • Containers, on the other hand, can start up in just a few seconds, making them a faster solution for rapidly scaling applications.	• Containers have a higher degree of portability than virtual machines, as they can run on any system with the Docker engine installed. • Virtual machines, on the other hand, require a compatible hypervisor and may not be as easily transferable between systems.	Virtual machines are typically used for running legacy applications or for scenarios where a high degree of resource isolation is required. Containers, on the other hand, are used for modern, cloud-native applications and for scenarios where fast, efficient deployment is a priority.

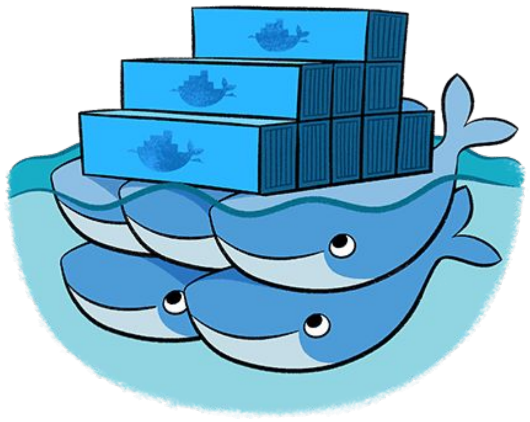
Docker vs Virtual Machines



Docker Hub

Docker also provides a centralized repository, called the Docker Hub, where developers can store and share their containers with others. This makes it easy to find and reuse existing containers, and to collaborate with other developers on projects.





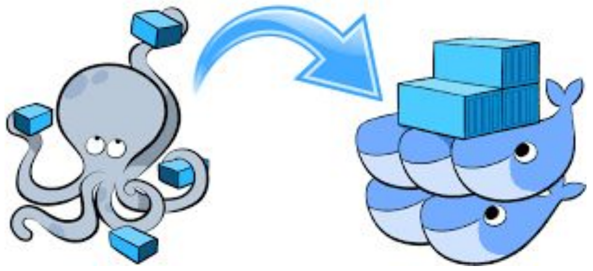
Docker compose

Docker Compose is **a tool for defining and running multi-container Docker applications**. It allows developers to define their application's services, networks, and volumes in a single file, called a docker-compose.yml file. This file can then be used to set up and run the entire application stack, with a single command.

Docker compose

Docker Compose makes it easy to manage complex **applications that are composed of multiple services**, as it provides a convenient way to define, configure, and run all the services together. This reduces the complexity of managing multiple containers and makes it easier to test, deploy, and scale applications.

With Docker Compose, **developers can specify the desired state of their application, such as the environment variables, network settings, and volumes, and Docker Compose will take care of creating and starting the containers for each service**. This makes it easy to automate the deployment process and to ensure that the application is always running the same way, regardless of the environment.

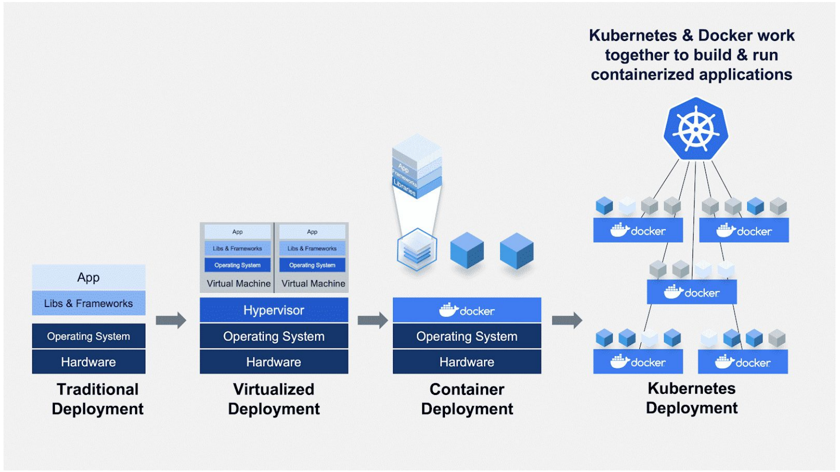


Docker compose vs Kubernetes

Docker Compose and Kubernetes are both popular tools for managing multi-container applications, **but they are designed for different use cases and have different architectures.**

Docker Compose is a tool for defining and running multi-container applications on a single host. It is easy to use and requires minimal setup, making it a popular choice for developers who want to quickly set up and test their applications locally. Docker Compose provides a simple and convenient way to define, configure, and run multi-container applications, but it is limited to a single host and does not provide the same level of scalability and resource management as Kubernetes.

Kubernetes, on the other hand, is a production-ready platform for deploying, scaling, and managing containerized applications. It provides a powerful and flexible architecture for managing multi-container applications at scale, and it includes features for automatic scaling, rolling updates, self-healing, and resource management. Kubernetes is designed for large, complex, and mission-critical applications, and it provides a high degree of availability and resilience.


Kubernetes & Docker work together to build & run containerized applications
App
Libs & Frameworks
Operating System
Hardware
Traditional Deployment
App
Libs & Frameworks
Operating System
Virtual Machine
Hypervisor
Operating System
Hardware
Virtualized Deployment
docker
Operating System
Hardware
Container Deployment
docker
Kubernetes Deployment