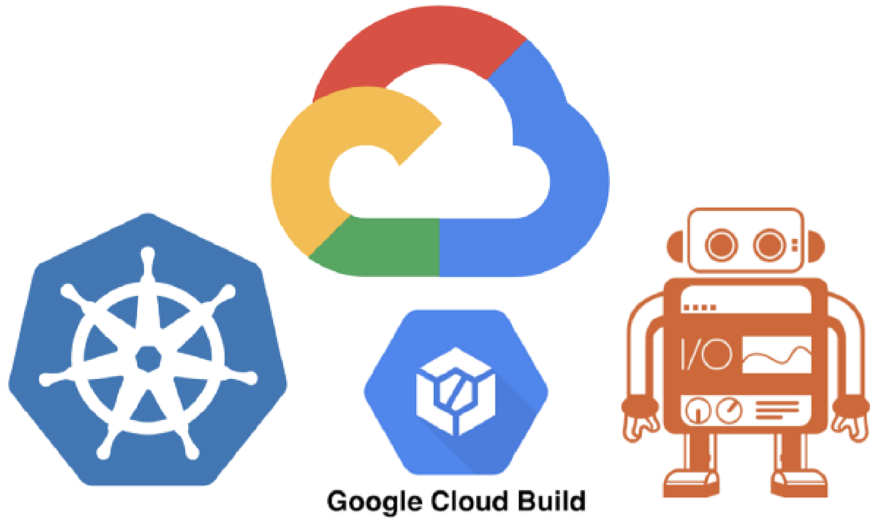


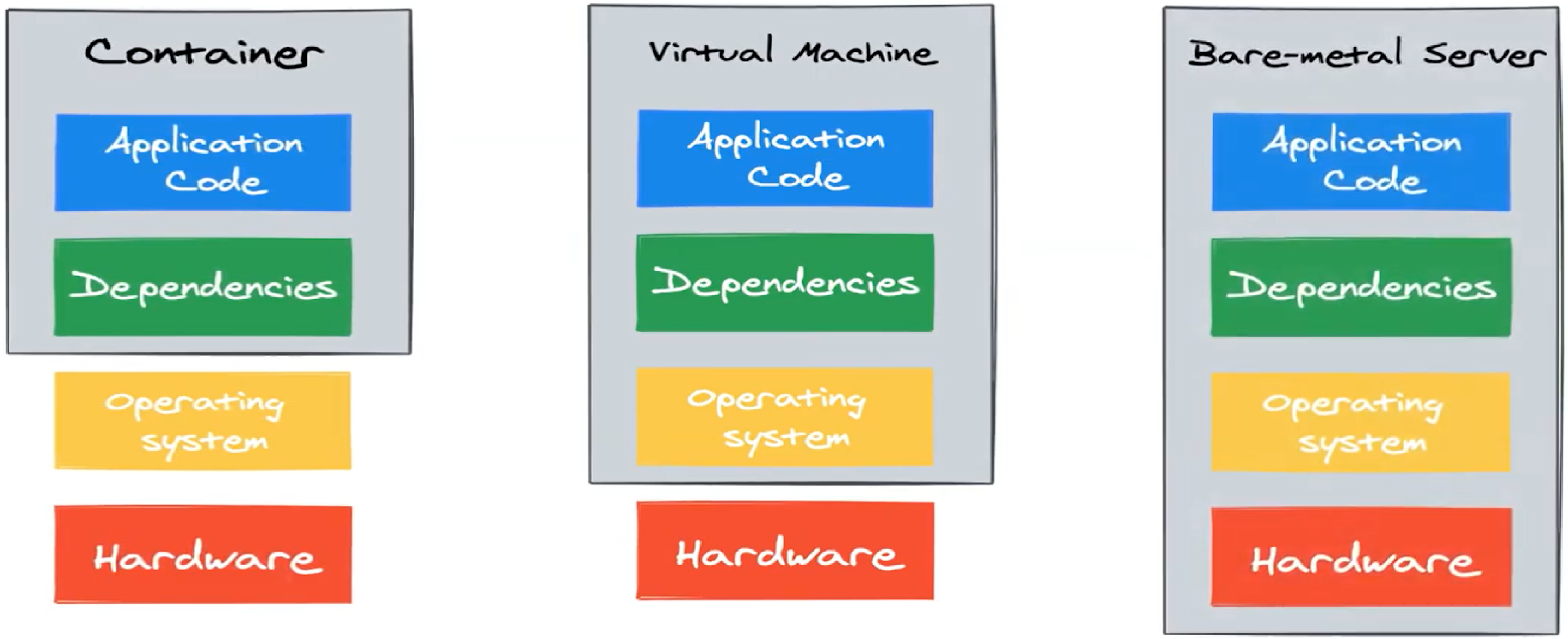
Scaling tests on Google Kubernetes Engine with Cloud Build

Agenda

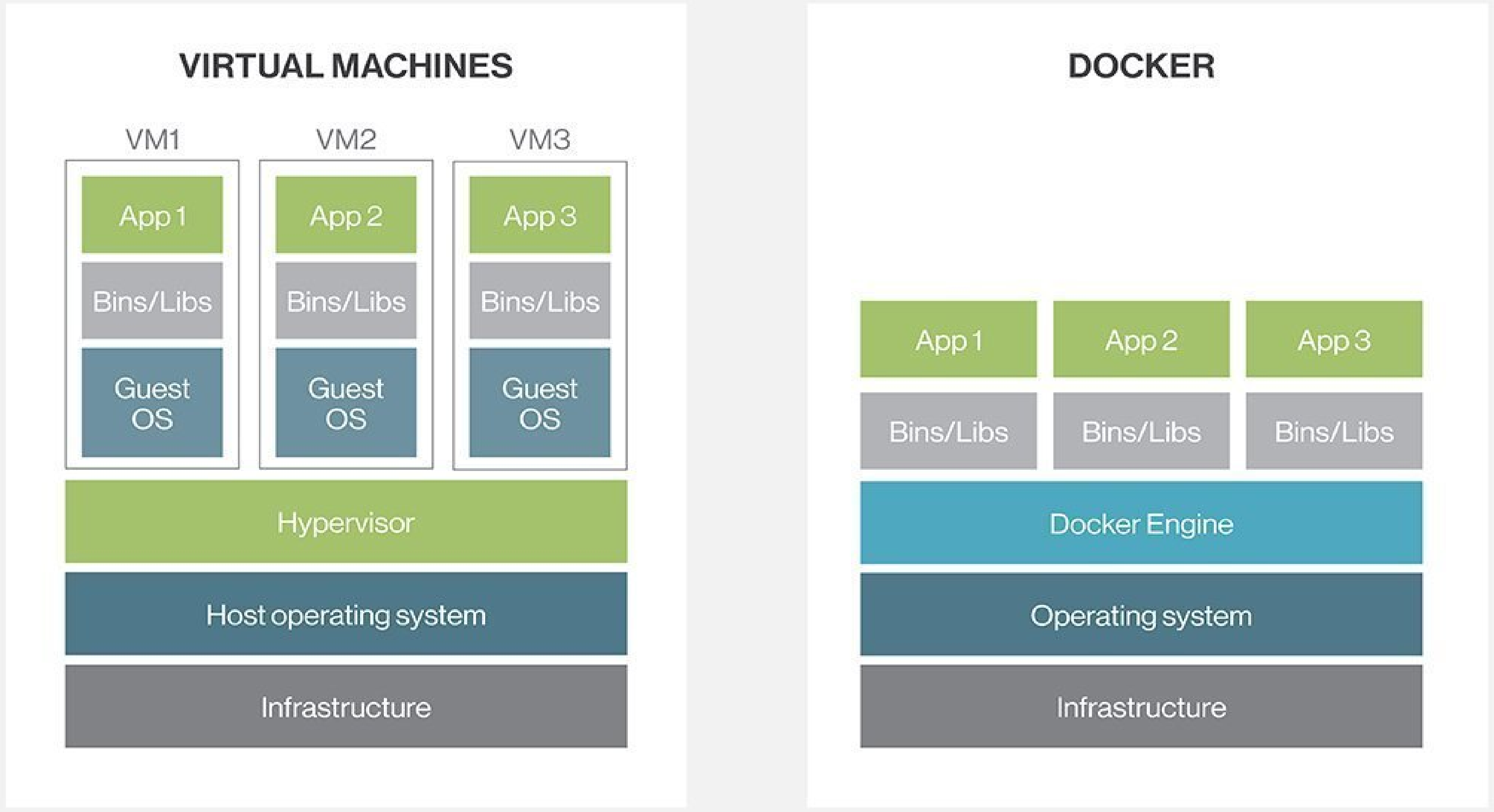
- Containerization
    - Docker Architecture
    - Main components of Docker
    - Demo time !!!
- Orchestration
    - Kubernetes Architecture
    - Kubernetes Components
    - Code walk through
    - Demo time !!!

Evolution

Containerization is the packaging together of software code with all it’s necessary components like libraries, frameworks, and other dependencies so that they are isolated in their own **container**.

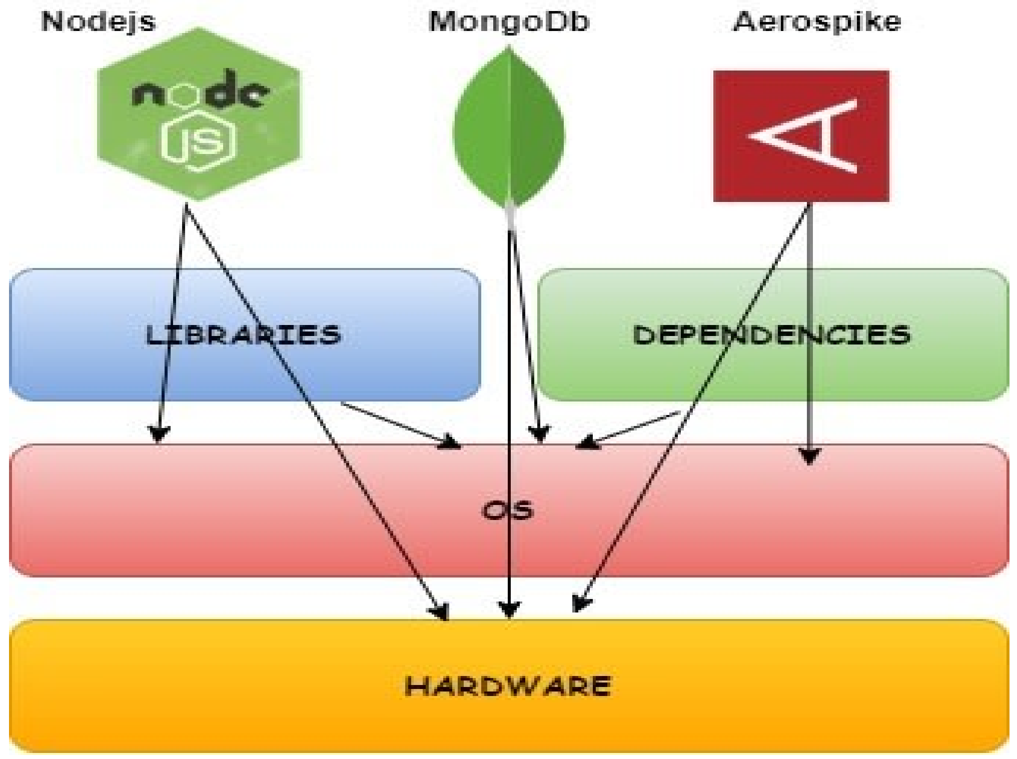


Virtual machines versus Docker



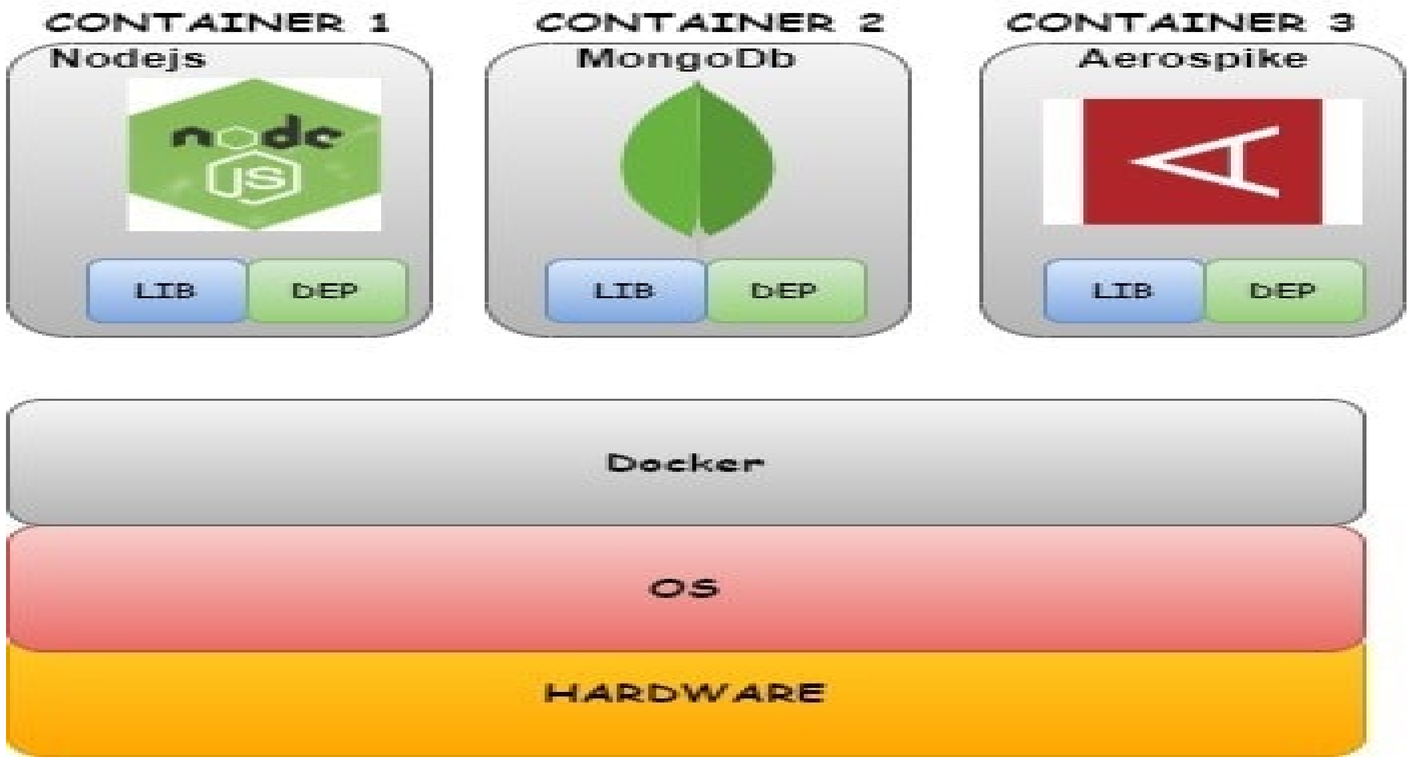
SOURCE: DOCKER

Why we need ?



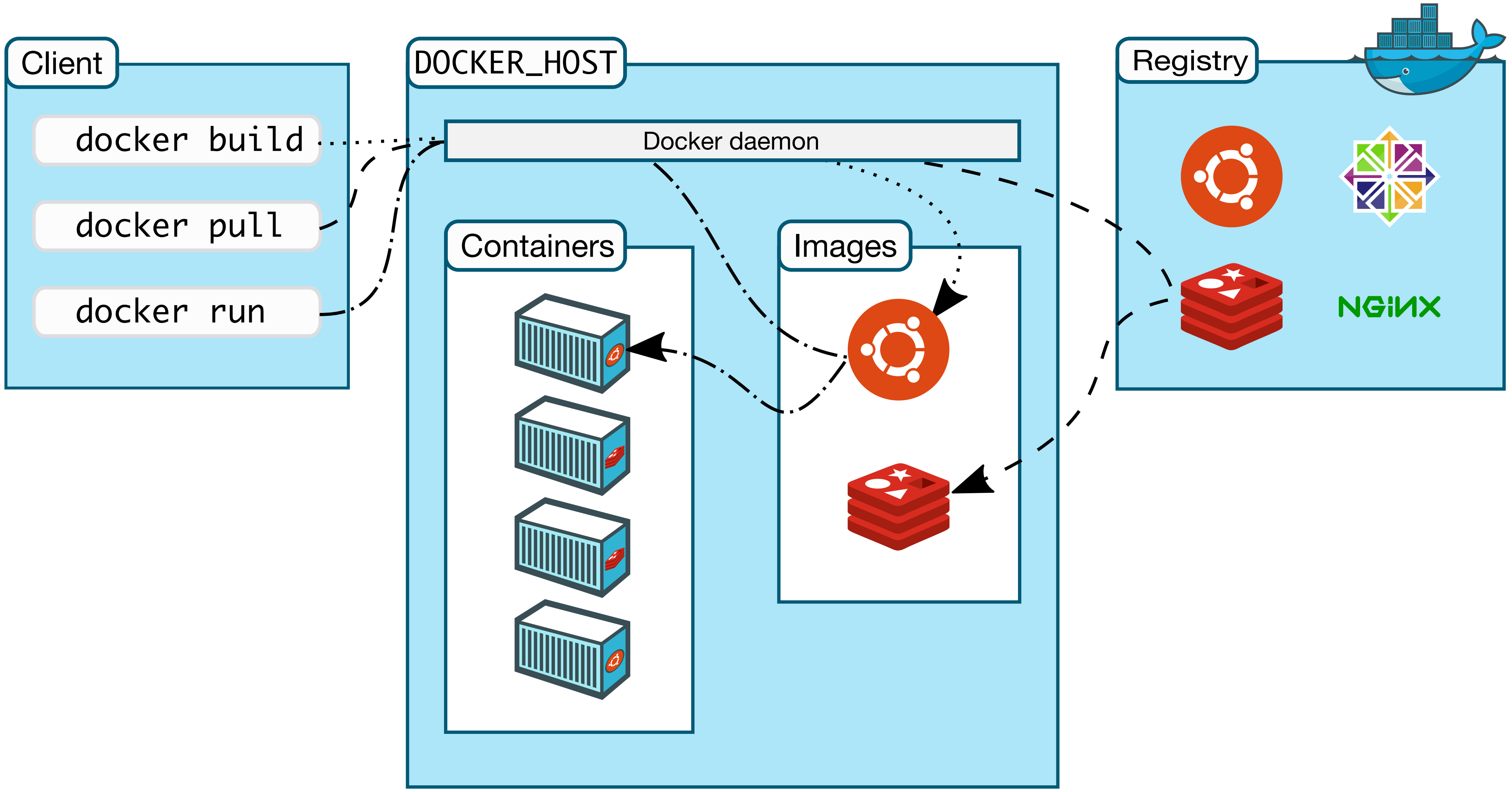
- To avoid compatibility issues with an underlying OS or between the services & libraries dependencies with the OS. (**So no more - It works on my machine!**)
- To reduce local development environment setup time.

What it does ?



- Containerize an applications
- Isolates apps from each other
- Run each service with its own dependencies in separate containers

Docker architecture



Docker Components

- **Client** - is the primary way to interact with Docker. When commands such as `docker run` executed the client sends these commands to dockerd (daemon), which carries them out.
- **Daemon** - receives the commands from the client through CLI or REST API. It listens to requests and processes Docker commands related to images, containers, networks, and volumes.
- **Dockerfile** - is a text document that contains all the commands to build an image.
- **Image** - are read-only templates built from a set of instructions written in Dockerfile. Images contains application and its dependencies.
- **Registries** - is a repository (storage) for Docker images.
- **Containers** - is the instance of an image. We can create, run, stop, or delete a container using the Docker CLI.

Some of the famous Containerization technology

- Docker - de facto standard
- cri-o
- podman - it is a daemon less container engine

Demo Time !!!

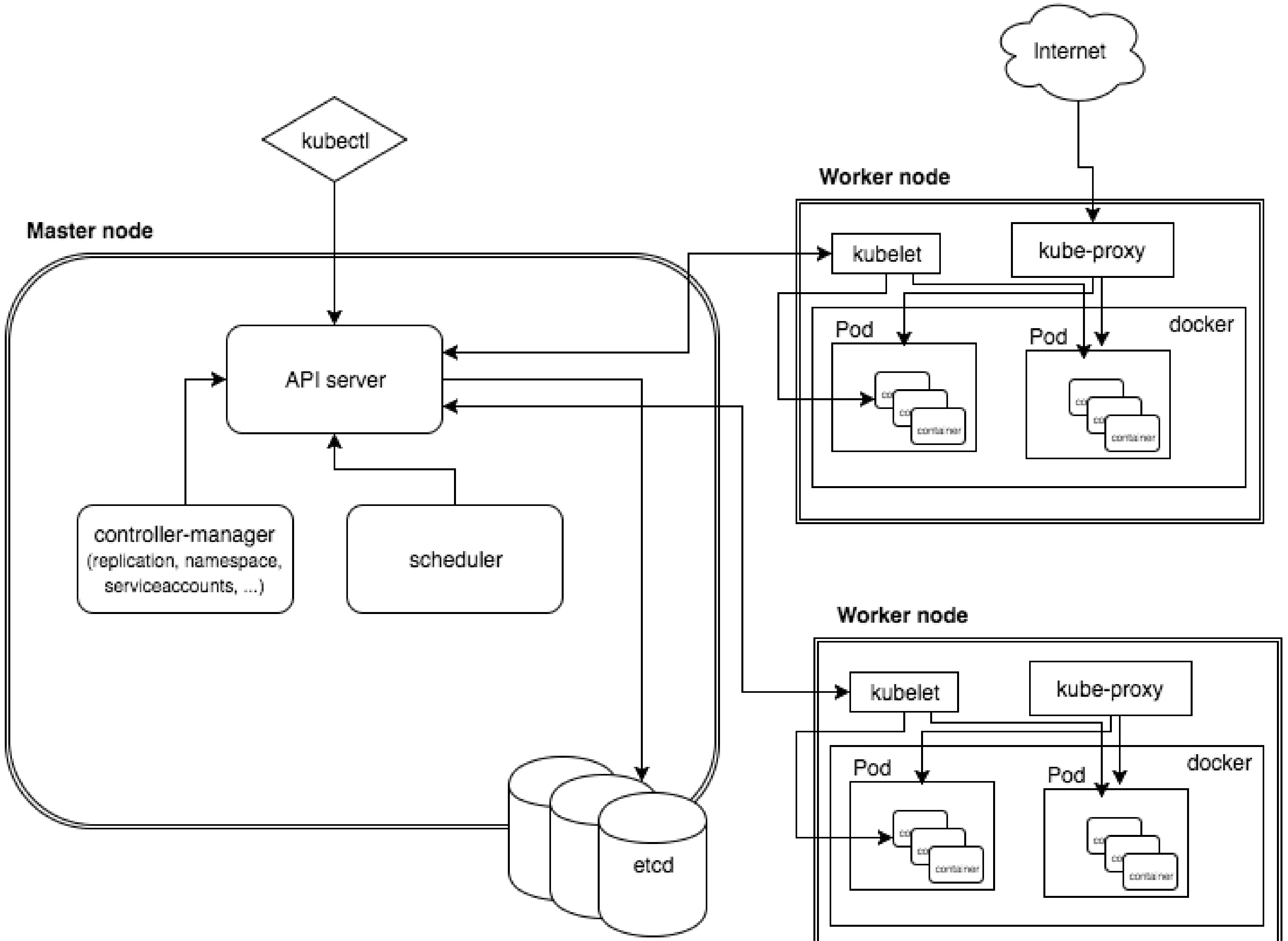
Orchestration

Container orchestration is the automation and management of the lifecycle of containers and services.

It solves the problem by automating the scheduling, deployment, scalability, load balancing, availability, and networking of containers.

K8s

Architecture



Kubernetes components

Control plane / Master node

- **kube-apiserver** - It's an entry point to interact with Kubernetes API (acts as a gateway). It determines if a request is valid and, if it is, processes it. It can be accessed through the kubectl cmd or kubeadm.
- **kube-scheduler** - It considers the resource needs of a pod, such as CPU or memory, along with the health of the cluster. Then it schedules the pod to an appropriate compute node. Then it passes on the requests to kubelet to execute it.
- **kube-controller-manager** - Detects state changes in the cluster (eg: if pod crashes, detects it and recovers the cluster state). Checks with scheduler and makes sure the correct number of pods is running.
- **etcd** - It is a key-value store database, stores the cluster state. All of the cluster information (eg: new pod created, pod crashed) is stored in the etcd.

Compute machine / Worker node

- **kubelet** - Kubernetes agent running on nodes, a tiny application that communicates with the control plane & containers vice versa and when the control plane needs something to happen in a node, the kubelet executes the action. It ensures the containers are running in the pod by providing the health information to the control plane and carry out actions based on control plane.
- **kube-proxy** - The kube-proxy handles network communications inside or outside of your cluster. It uses operating system’s packet filtering layer if there is one, otherwise , kube-proxy forwards the traffic itself.
- **container runtime** - It is the software responsible for running containers. It can be Docker, Containerd and CRI-O.

Kubernetes Objects

- **Namespace** - which help in organizing resources and partition a single Kubernetes cluster into multiple virtual clusters. It provides a degree of isolation from the other parts of the cluster.
- **Service** - is an abstract way to expose an application running on a set of Pods as a network service. It creates a permanent IP address, the lifecycle of pod and service are not connected. Even if the pods crash and are recreated, the service IP remains the same.
- **Deployment** - describes the desired state of a pod or a replica set, then gradually updates the environment (for example, creating or deleting replicas) until the current state matches the desired state specified in the deployment file. In general, we don’t work directly with pods, we will create deployments. It is mainly for stateless apps.

Some of the famous Orchestration tools

- Kubernetes - de facto standard
- OpenShift
- Nomad
- Docker swarm
- Apache Mesos

Cloud build

- it is a serverless CI/CD platform to build, test, and deploy.
- vendor lock in

Demo time !!!