COPY Instruction

- For Use Case, let’s build a container that compiles a basic "Hello world" program in C.
- hello.c

```
[root@TechLanders yogesh]# cat hello.c
#include<stdio.h>
int main () {
    puts("Hello, TechLanders!");
    return 0;
}
```

- Dockerfile

```
[root@TechLanders yogesh]# cat Dockerfile
FROM ubuntu
RUN apt-get update
RUN apt-get install -y build-essential
COPY hello.c /
RUN make hello
CMD /hello
```

Note: Using COPY keyword we can copy the files from Docker Host to a container.

COPY Instruction

- [root@TechLanders yogesh]# docker build -t fifthfile .

- [root@TechLanders yogesh]# docker run -it fifthfile /bin/bash

- root@519a9d815a29:/# ls -lrt /hello
- -rwxr-xr-x. 1 root root 8600 Sep 18 11:54 /hello

- root@519a9d815a29:/# /hello
- Hello, world!
- root@519a9d815a29:/#

Environment Setting

```
FROM debian:stable
RUN apt-get update && apt-get install -y apache2
ENV MYVALUE Sandeep Kumar
EXPOSE 80
CMD ["/usr/sbin/apache2ctl","-D","FOREGROUND"]
```

Real Example-2

- vi Dockerfile

- FROM debian:stable
- MAINTAINER Gagandeep <gagandeep.singh@TechLanders.com>
- RUN apt-get update && apt-get upgrade -y && apt-get install -y apache2 telnet elinks openssh-server

- ENV MYVALUE gagandeep

- EXPOSE 80

- CMD ["/usr/sbin/apache2ctl","-D","FOREGROUND"]

- docker build -t yogeshraheja/myapache .

- docker run -d –p 8989:80 <image>

- docker ps

Real Example-2

- docker inspect <container> | grep -i ipad
- elinks http://<IP> (if not given dedicated IP) OR Access via public IP
- docker exec -it <container> /bin/bash
- ps aux| grep -i apache
- echo $MYVALUE

Real Example-3

- vi Dockerfile

- FROM debian:stable
- MAINTAINER Gagandeep <gagandeep.singh@TechLanders.com>
- RUN apt-get update && apt-get upgrade -y && apt-get install -y apache2 telnet elinks openssh-server

- ENV MYVALUE yogesh-raheja

- EXPOSE 80

- CMD ["/usr/sbin/apache2ctl","-D","FOREGROUND"]

- docker build -t yogeshraheja/myapache .

- docker run -d –p 8989:80 <image>

- docker ps

Docker Restart Policy

$ docker run -dit --restart unless-stopped centos

Flag	Description
no	**Do not automatically restart the container. (the default)**
on-failure	**Restart the container if it exits due to an error, which manifests as a non-zero exit code.**
always	**Always restart the container if it stops. If it is manually stopped, it is restarted only when Docker daemon restarts or the container itself is manually restarted. (See the second bullet listed in restart policy details)**
unless-stopped	**Similar to always, except that when the container is stopped (manually or otherwise), it is not restarted even after Docker daemon restarts.**

Docker Registries

Local Registry (Local to Host)

Remote Registry (Private)

Global Registry (Public)

Docker Trusted Registry

- Enterprise-grade image storage solution from Docker
- Highly Secure
- Image and job management with CICD
- HA Availability
- Efficiency with Near to user storage and bandwidth sharing
- RBAC
- Security Scanning
- Similar tools in market Sonatype Nexus (open source), AWS ECR (PaaS), Azure Container Registry, GCP container Registry etc

Docker Trusted Registry

- Enterprise-grade image storage solution from Docker
- Highly Secure
- Image and job management with CICD
- HA Availability
- Efficiency with Near to user storage and bandwidth sharing
- RBAC
- Security Scanning
- Similar tools in market Sonatype Nexus (open source), AWS ECR (PaaS), Azure Container Registry, GCP container Registry etc

Docker Compose

- Compose is a tool for defining and running multi-container Docker applications
- Write syntax in yaml - docker-compose.yml to configure the services

```
version: '2.0'
services:
  web:
    build: .
    ports:
    - "5000:5000"
    volumes:
    - .:/code
    - logvolume01:/var/log
    links:
    - redis
  redis:
    image: redis
volumes:
  logvolume01: {}
```

Docker Swarm

- Clustering (basically Orchestration) of Docker Hosts
- Provide High availability for node failure and container failures
- Load balancing and service distribution
- Role based access control
- Scalability

Universal Control Plane

- Enterprise-grade cluster management solution from Docker
- Centralized cluster management
- Deploy, manage, and monitor via GUI
- Built-in security and access control
- Integrated with DTR
- Can utilize CLI

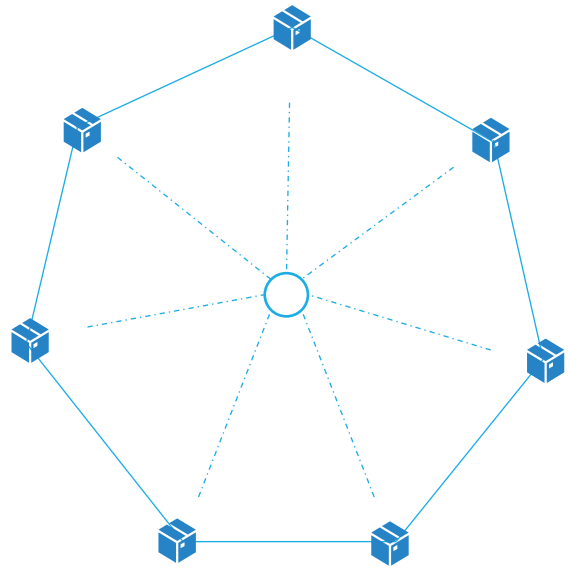
Note: Better tools available in market like Rancher, Openshift

Docker Limitations

- Hardware Issues? High Availability?
- How IP address will be managed for failover?
- Scaling?
- Auto Healing?
- Autoscaling?
- No Application Management - Only Containerization
- Updation of application/management

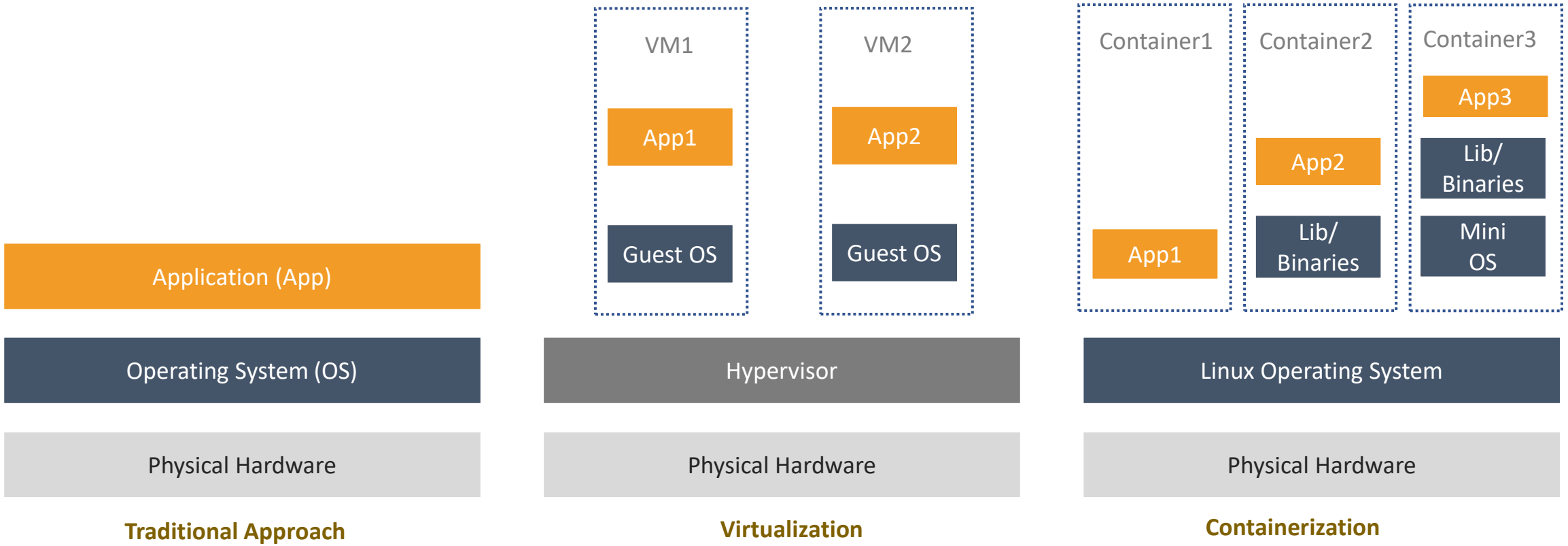
Why Kubernetes

- Kubernetes can schedule and run application containers on clusters of physical or virtual machines.
- **host-centric** infrastructure to a **container-centric** infrastructure.
- Orchestrator
- Load balancing
- Auto Scaling
- Application Health checks
- Rolling updates



kubernetes

Containers

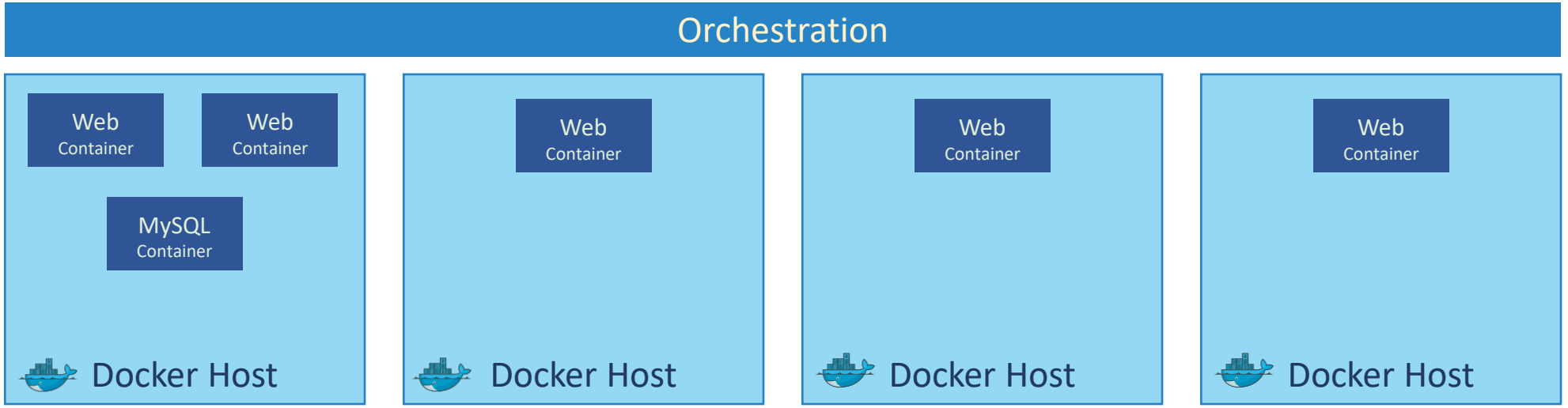

Application (App)
Operating System (OS)
Physical Hardware
Traditional Approach
VM1
App1
Guest OS
VM2
App2
Guest OS
Hypervisor
Physical Hardware
Virtualization
Container1
App1
Container2
App2
Lib/ Binaries
Container3
App3
Lib/ Binaries
Mini OS
Linux Operating System
Physical Hardware
Containerization

Container Orchestration

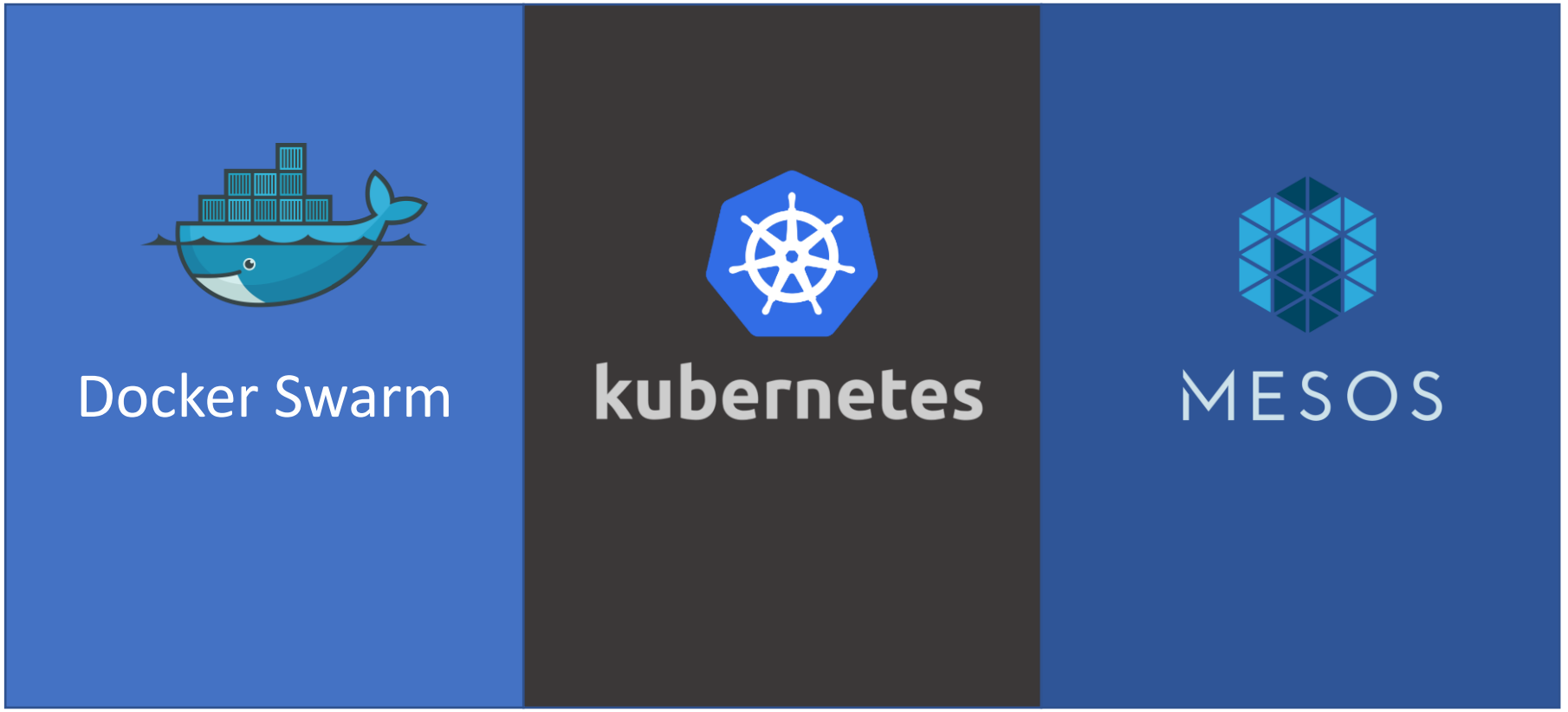
Containers Limitation?

High Availability?
Overlay Network?
Application Centric or Infra Centric?
Versioning of Application – Rollout, Rollback?
Scaling?
Autoscaling?
Monitoring?
Dependency between containers?

Container orchestration



Orchestration Technologies



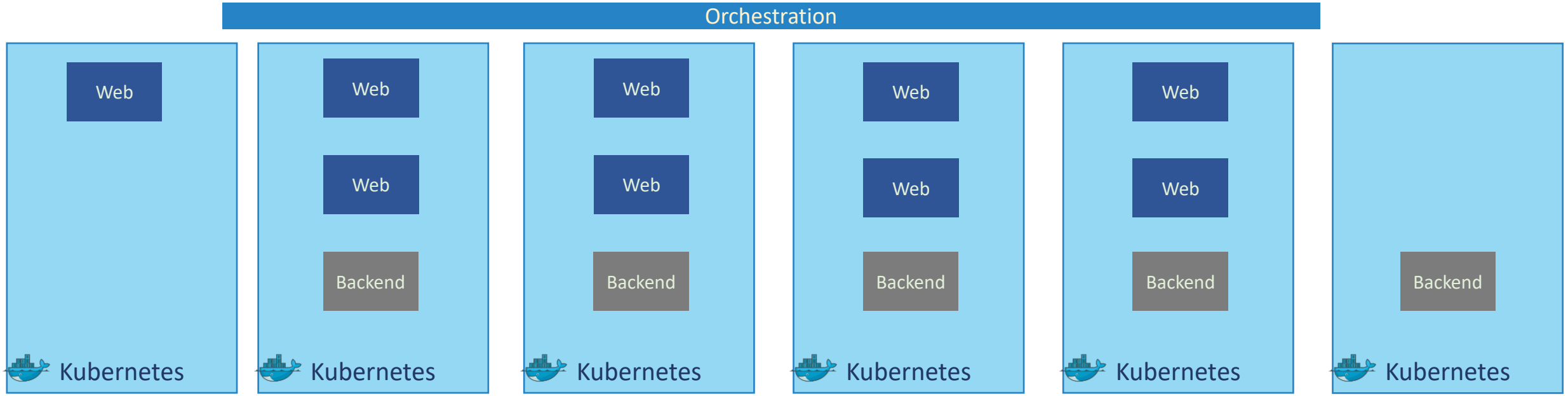
What is Kubernetes?

- The Kubernetes project was started by Google in 2014.
- Kubernetes builds upon a decade and a half of experience that Google has with running production workloads at scale.
- Kubernetes can run on a range of platforms, from your laptop, to VMs on a cloud provider, to rack of bare metal servers.
- Kubernetes is an open-source platform for automating deployment, scaling, and operations of application containers across clusters of hosts, providing container-centric infrastructure.
- **portable**: with all public, private, hybrid, community cloud
- **self-healing**: auto-placement, auto-restart, auto-replication, auto-scaling

Why Kubernetes

- Kubernetes can schedule and run application containers on clusters of physical or virtual machines.
- **host-centric** infrastructure to a **container-centric** infrastructure.
- Orchestrator
- Load balancing
- Auto Scaling
- Application Health checks
- Rolling updates

Kubernetes Advantage



And that is kubernetes..

Setup

Minikube

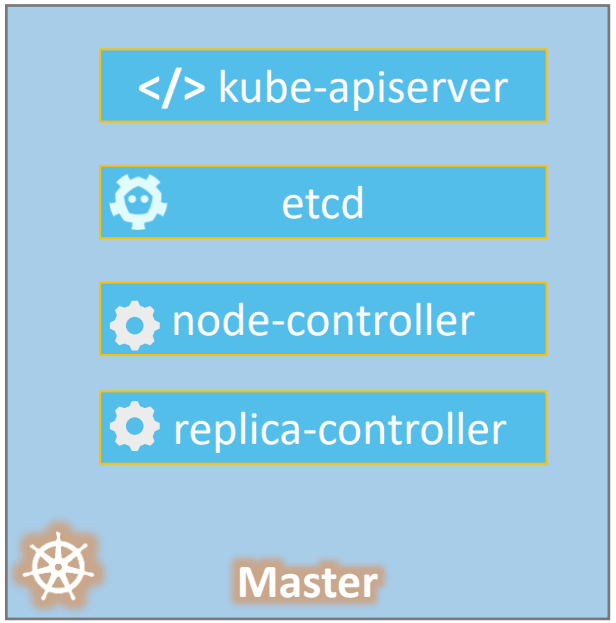
Kubeadm

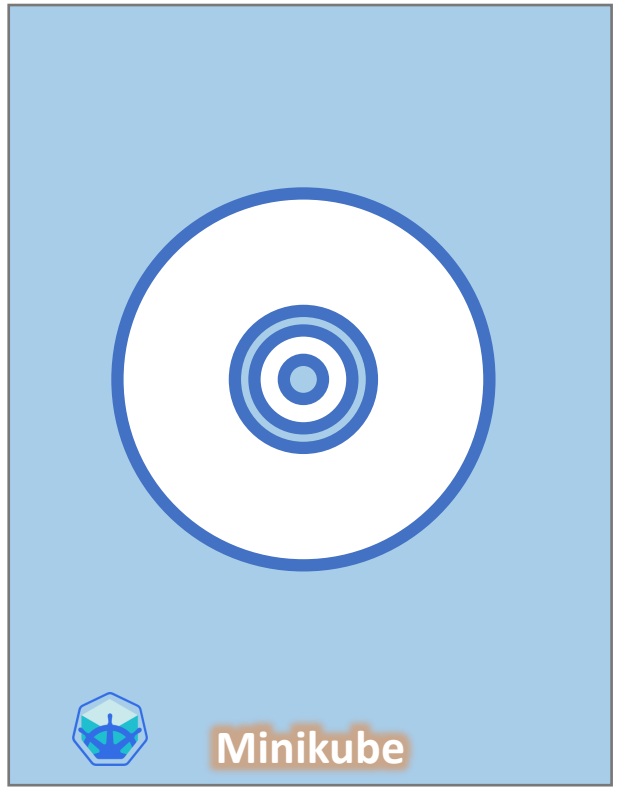
Google Cloud Platform

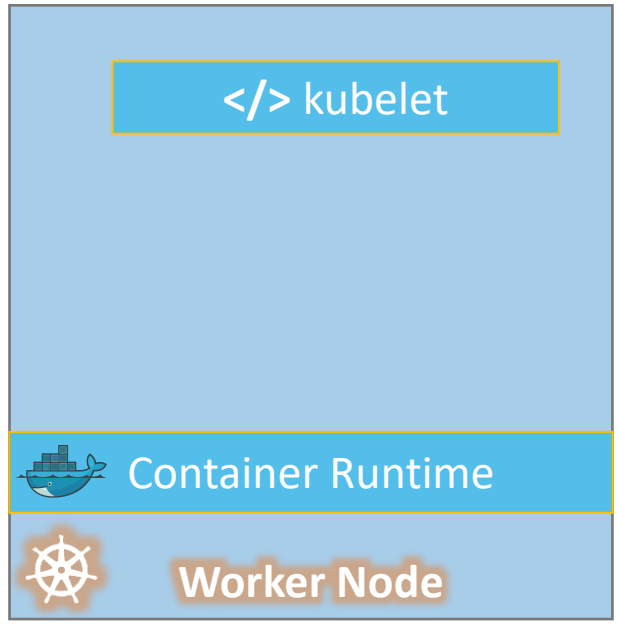
Amazon Web Services

play-with-k8s.com

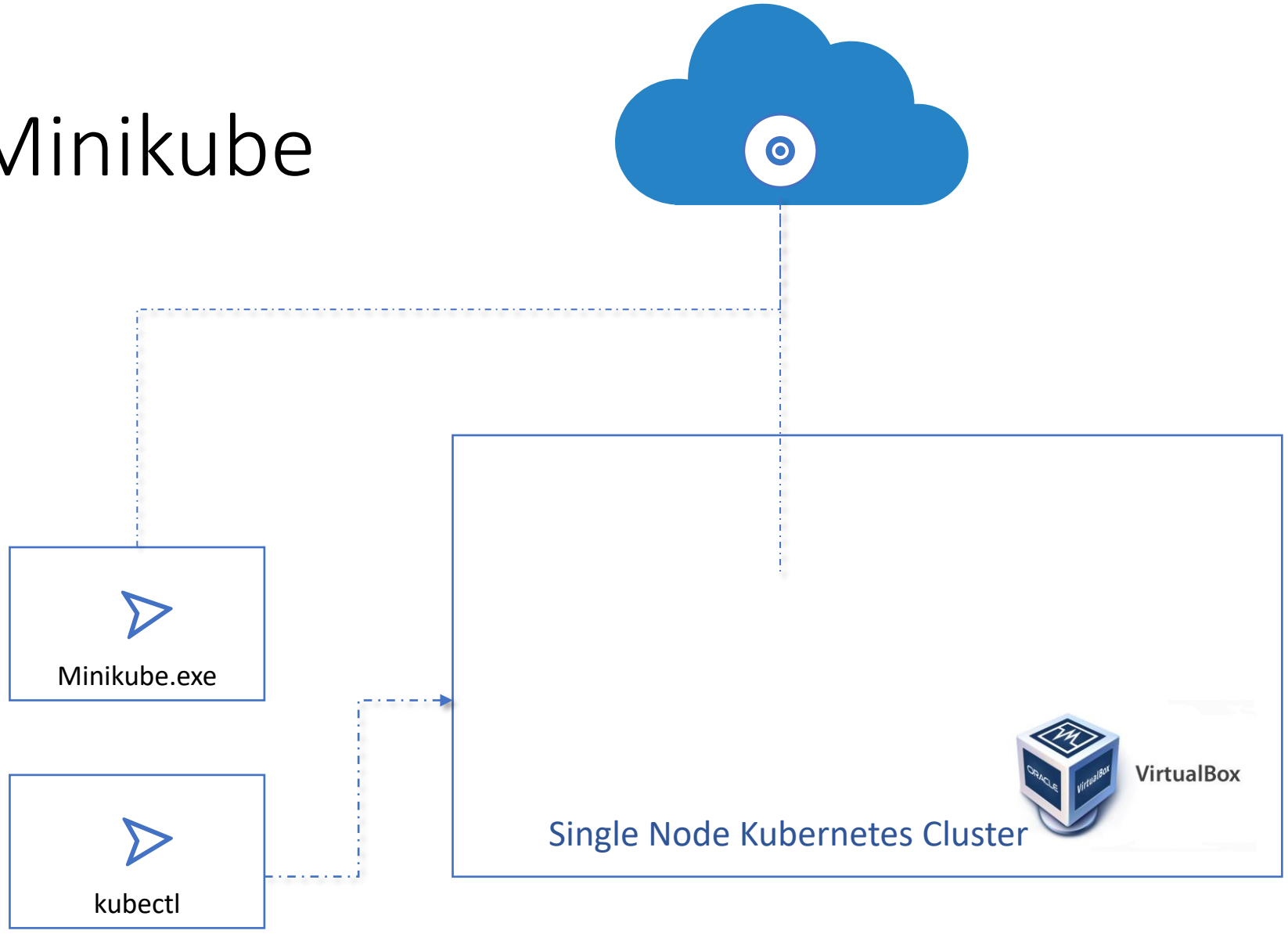
Minikube


kube-apiserver
etcd
node-controller
replica-controller
Master


Minikube


kubelet
Container Runtime
Worker Node

Minikube


Minikube.exe
kubectl
Single Node Kubernetes Cluster
VirtualBox

Demo

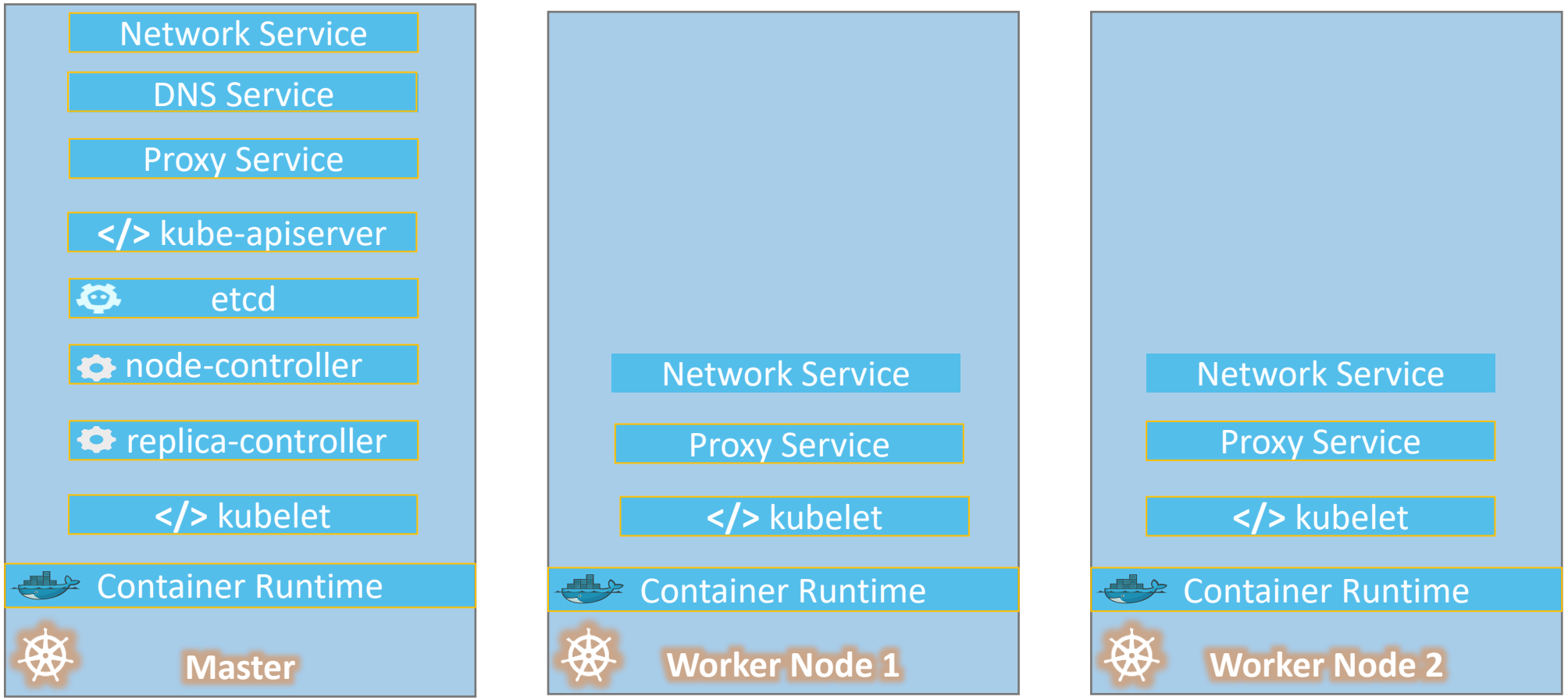
minikube

Setup - kubeadm

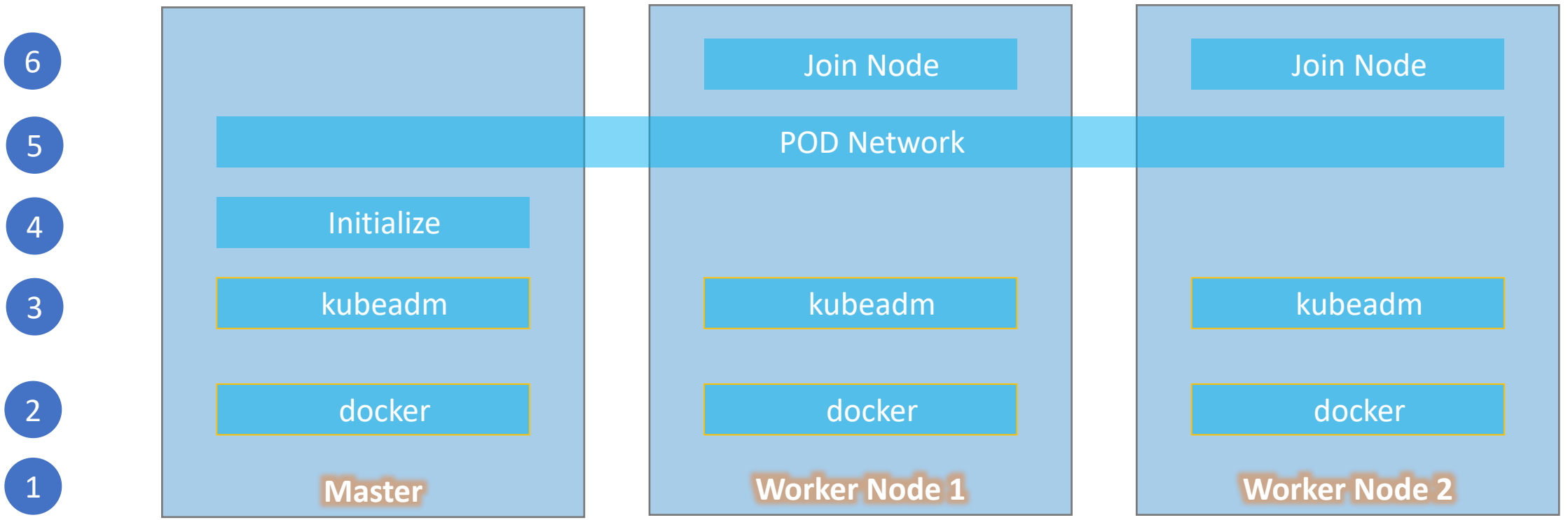
Kubernetes Cluster

- A Kubernetes cluster consists of two types of resources:
- **Master**: Which coordinates with the cluster
- The Master is responsible for managing the cluster. The master coordinates all activities in your cluster, such as scheduling applications, maintaining applications' desired state, scaling applications, and rolling out new updates.
- **Nodes**: Are the workers that run application
- A node is a VM or a physical computer that serves as a worker machine in a Kubernetes cluster.
- Masters manage the cluster and the nodes are used to host the running applications.
- **The nodes communicate with the master using the Kubernetes API**, which the master exposes.

kubeadm


Network Service
DNS Service
Proxy Service
</> kube-apiserver
etcd
node-controller
replica-controller
</> kubelet
Container Runtime
Master
Network Service
Proxy Service
</> kubelet
Container Runtime
Worker Node 1
Network Service
Proxy Service
</> kubelet
Container Runtime
Worker Node 2

Steps


6
5
4
3
2
1
Join Node
Join Node
POD Network
Initialize
kubeadm
kubeadm
kubeadm
docker
docker
docker
Master
Worker Node 1
Worker Node 2