



DevOps – Lecture 3

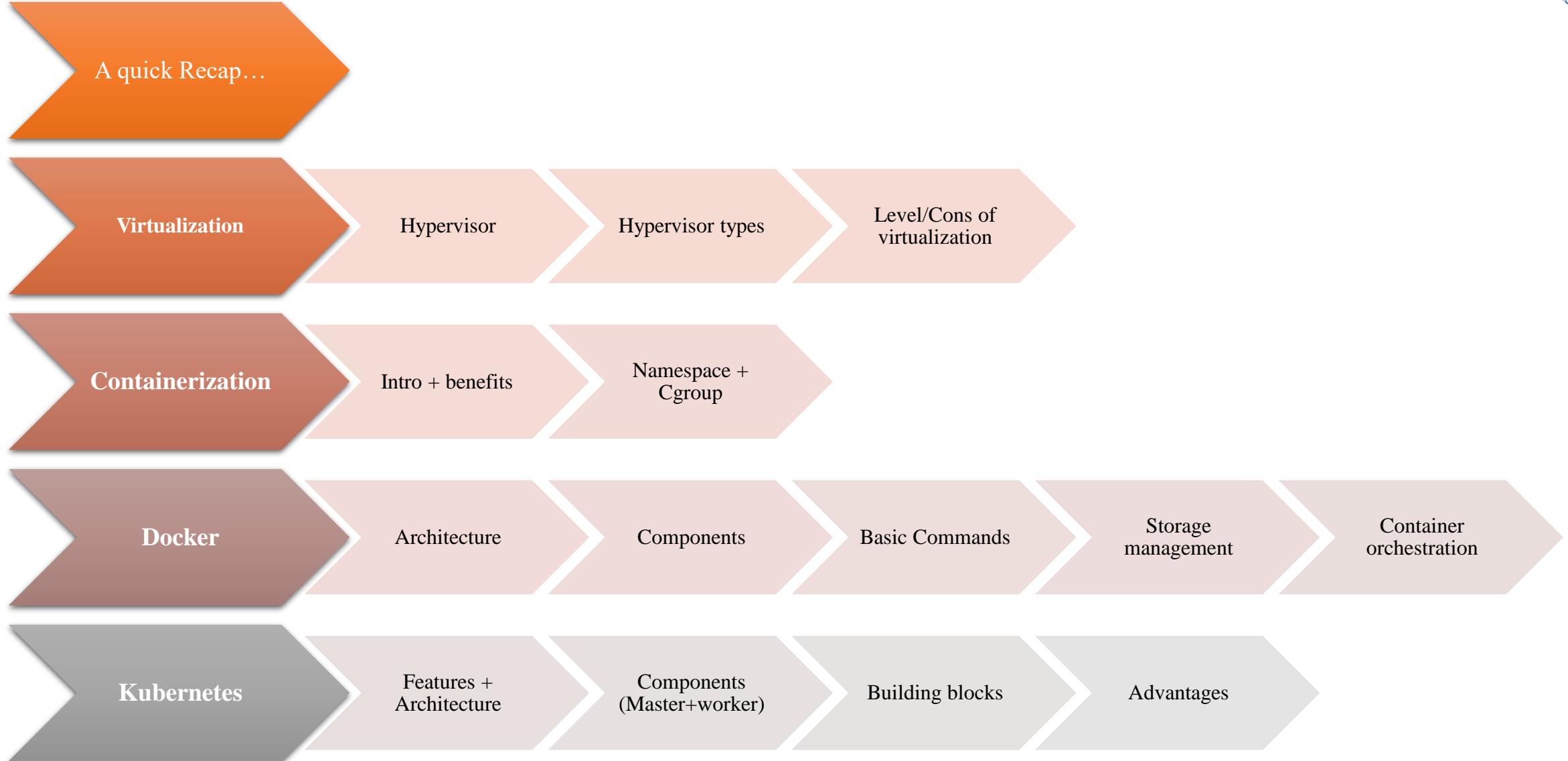
Containerization

22 Sept 2021

Chinmaya Kumar Dehury

Chinmaya.Dehury@ut.ee

OUTLINE



Updated Schedule

Next week, no session

- Lec-04: 29 Sept: No Session for UniTartuCS Day
 - Lab: No Session

Exam schedule

Final Exam

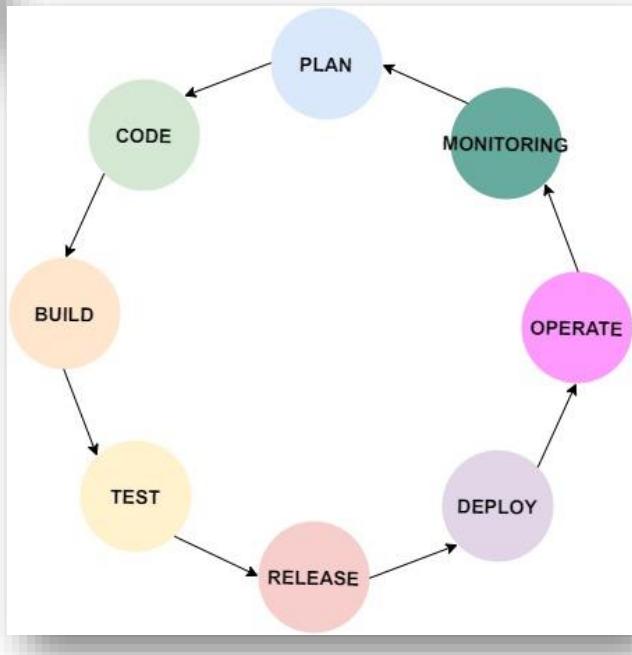
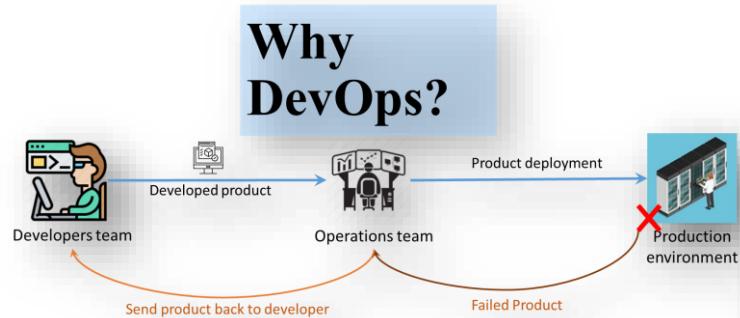
Student may choose any one date based on their convenience:

- Option 1: ~15 Dec 2021
- Option 2: ~05 Jan 2022

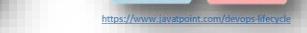
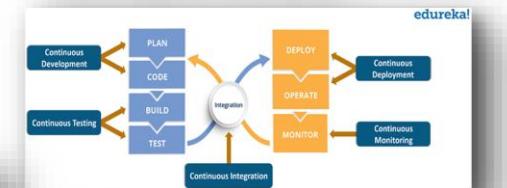
Should submit the deliverable by 5PM of the next Friday

e.g. For *Practice Session 3*, the deadline is *1 Oct 2021, 5PM Estonia time*

A quick recap... on DevOps



DevOps Phases



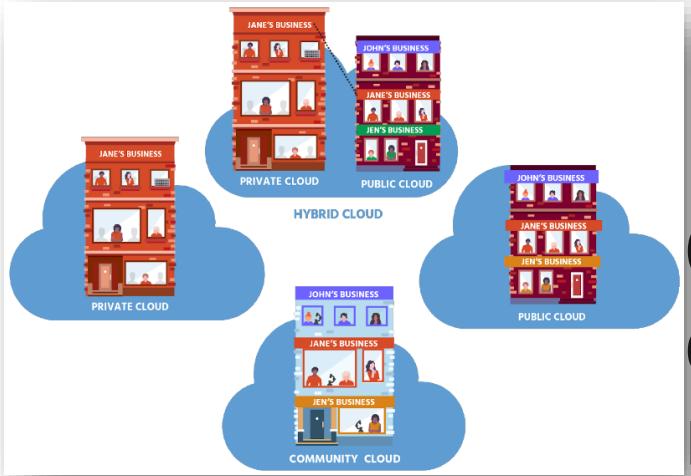
DevOps Benefits

What is a Culture?

What is a silo?

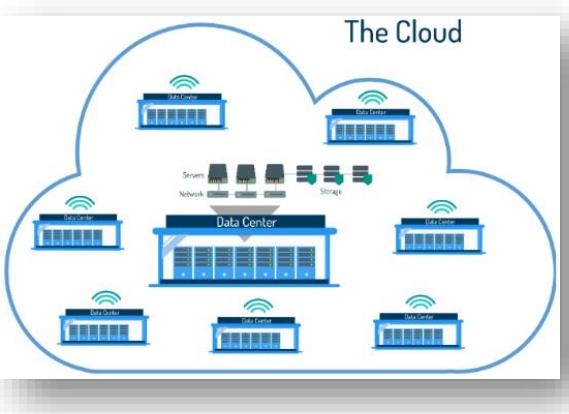
DevOps Goals

A quick recap... on Cloud computing

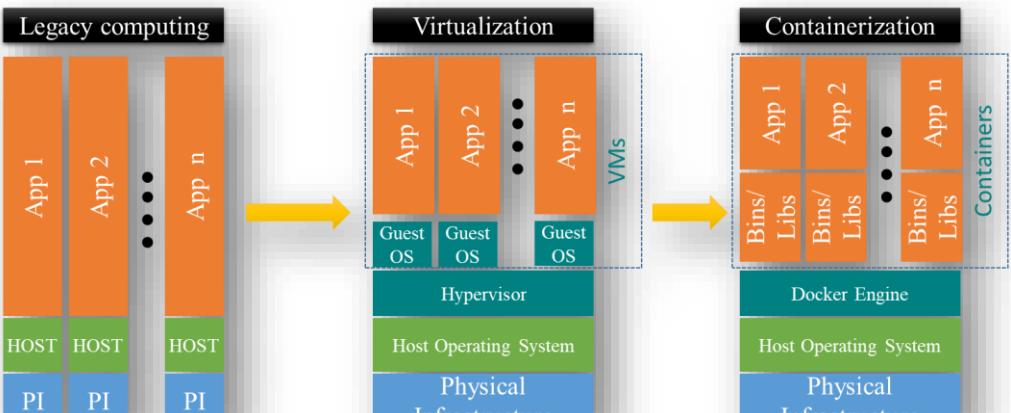
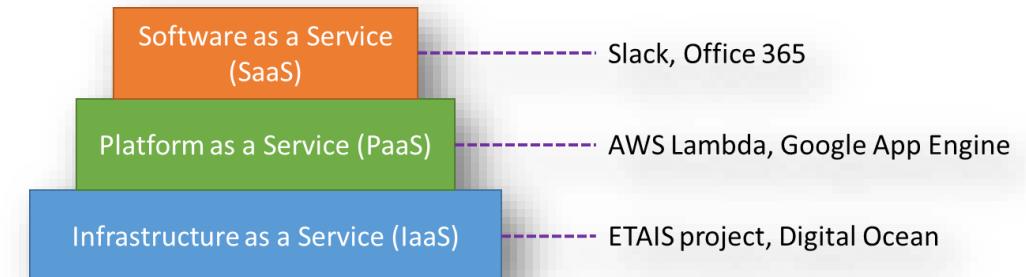


Cloud deployment model

What is Cloud Computing?

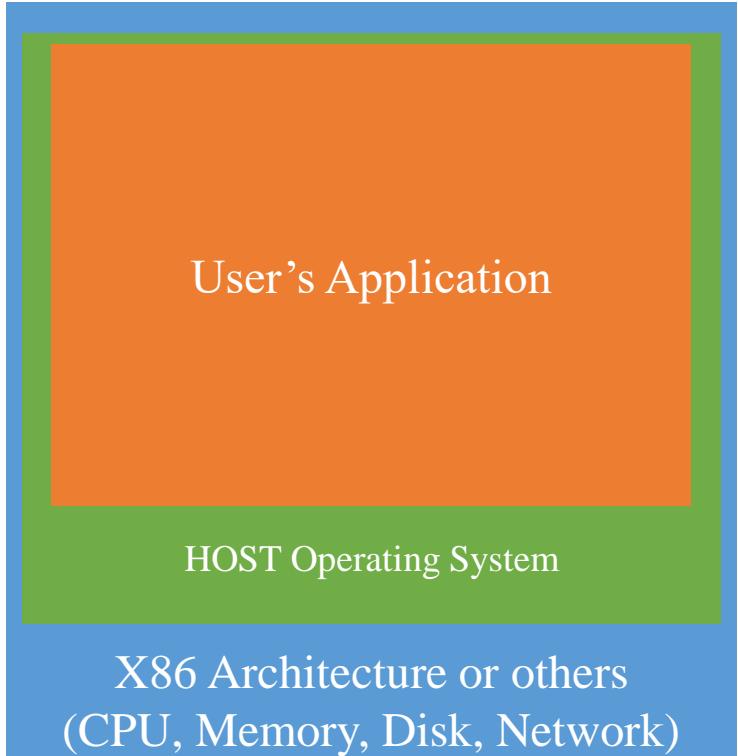


Types of Cloud Computing service models



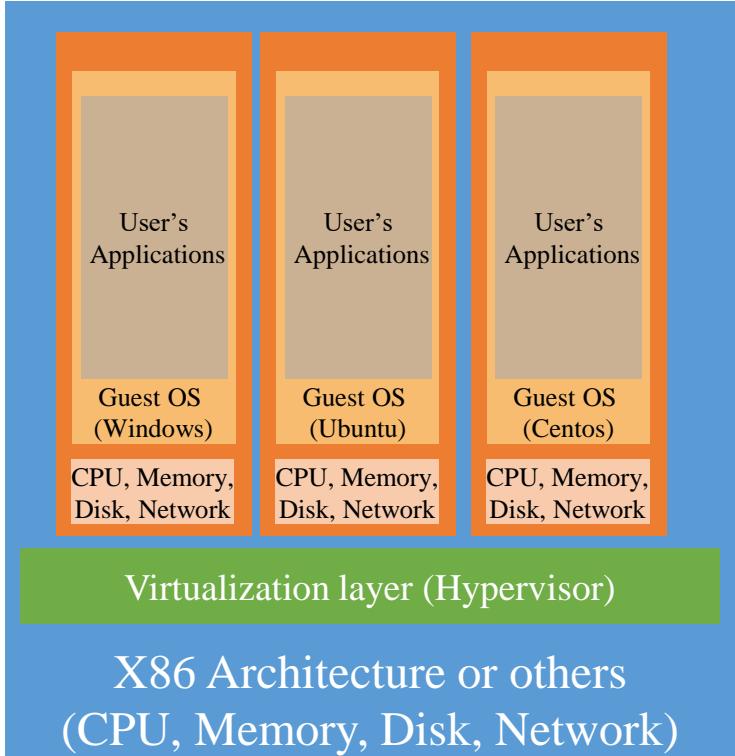
Past, Present & Future of Virtualization

Without Virtualization – traditional physical server



- Single OS per machine
- Single/multiple application per machine
- Hardware components connected directly to operating system
 - **CPU**
 - **Memory**
 - **Disk**
 - **Network Card**

With Virtualization



Why Virtualization ?

- Increased **performance** and computing **capacity** relevance to hardware manufacturing.
- **Underutilized** Hardware and software Resources
- Lack of Space
- Greening Initiatives
- Rise of administrative costs
 - Administrative tasks include: labor intensive
 - hardware monitoring
 - defective hardware replacement
 - server setup and updates
 - resource monitoring backups

Src: https://courses.cs.ut.ee/LTAT.06.008/2021_spring/uploads/Main/L2_CloudComputing2021.pdf

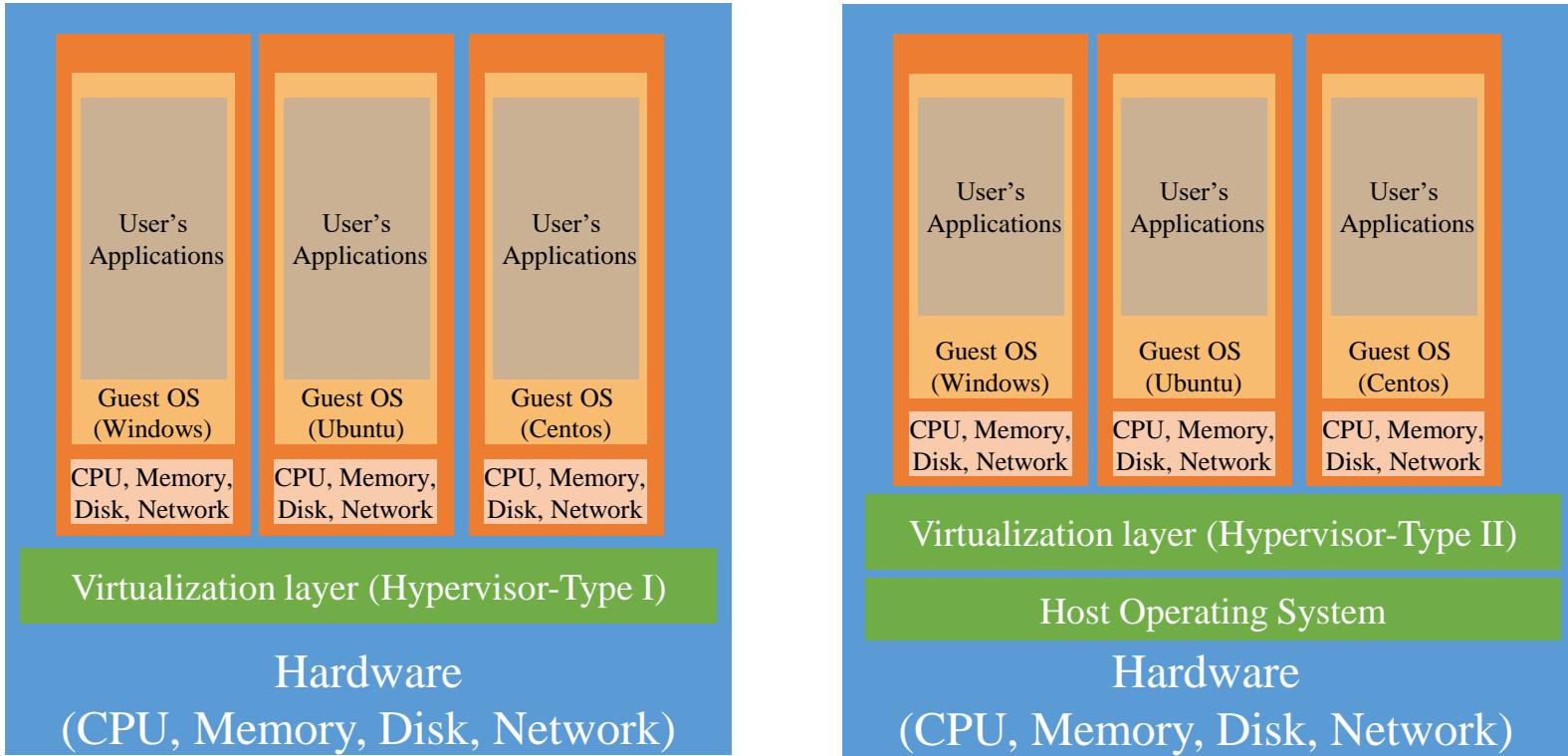
Virtualization

- To define it in a general sense,
 - virtualization encompasses any technology - either software or hardware - that adds an extra layer of isolation or extra flexibility to a standard system.
- Virtualization is mainly used to **emulate execution environment**, storage and networks.
- Execution Environment classified into two :
 - Process-level –implemented on top of an existing operating system.
 - System-level –implemented directly on hardware and do not or minimum requirement of existing operating system
- Familiar examples include
 - virtual memory for memory management,
 - virtual disks to allow for partitioning
 - "virtual machines" (e.g. for Java and .net) to allow for better software portability.

Hypervisor

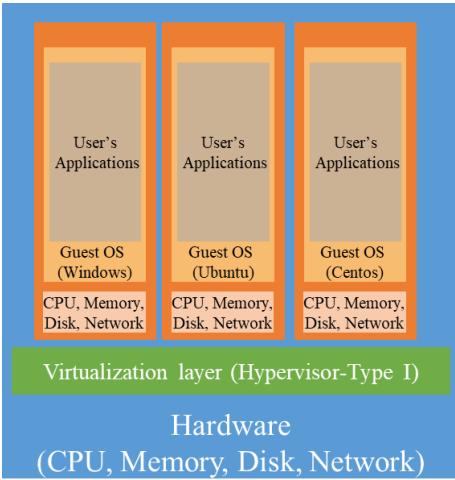
- a program used to run and manage one or more virtual machines on a computer. [[src](#)]
- It recreates a h/w environment.

Hypervisor type



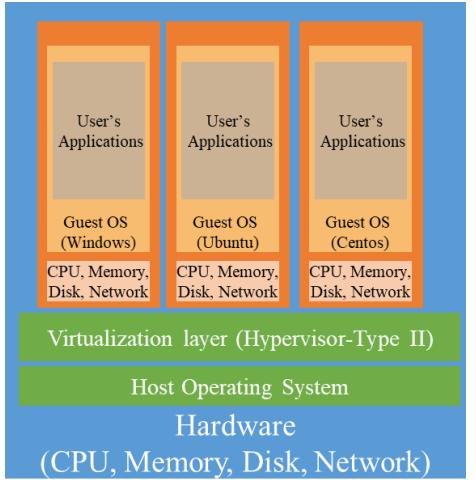
Src: <http://player.slideplayer.com/download/15/4802616/l13LuM2ipqUnDgvIrlExpjw/1632128184/4802616.ppt>

Hypervisor type



Type-I

- It runs directly on top of the hardware.
- Takes place of OS.
- Directly interact with the ISA exposed by the underlying hardware.
- Also known as *native virtual machine*. Example: VmWare ESXi, MS HyperV



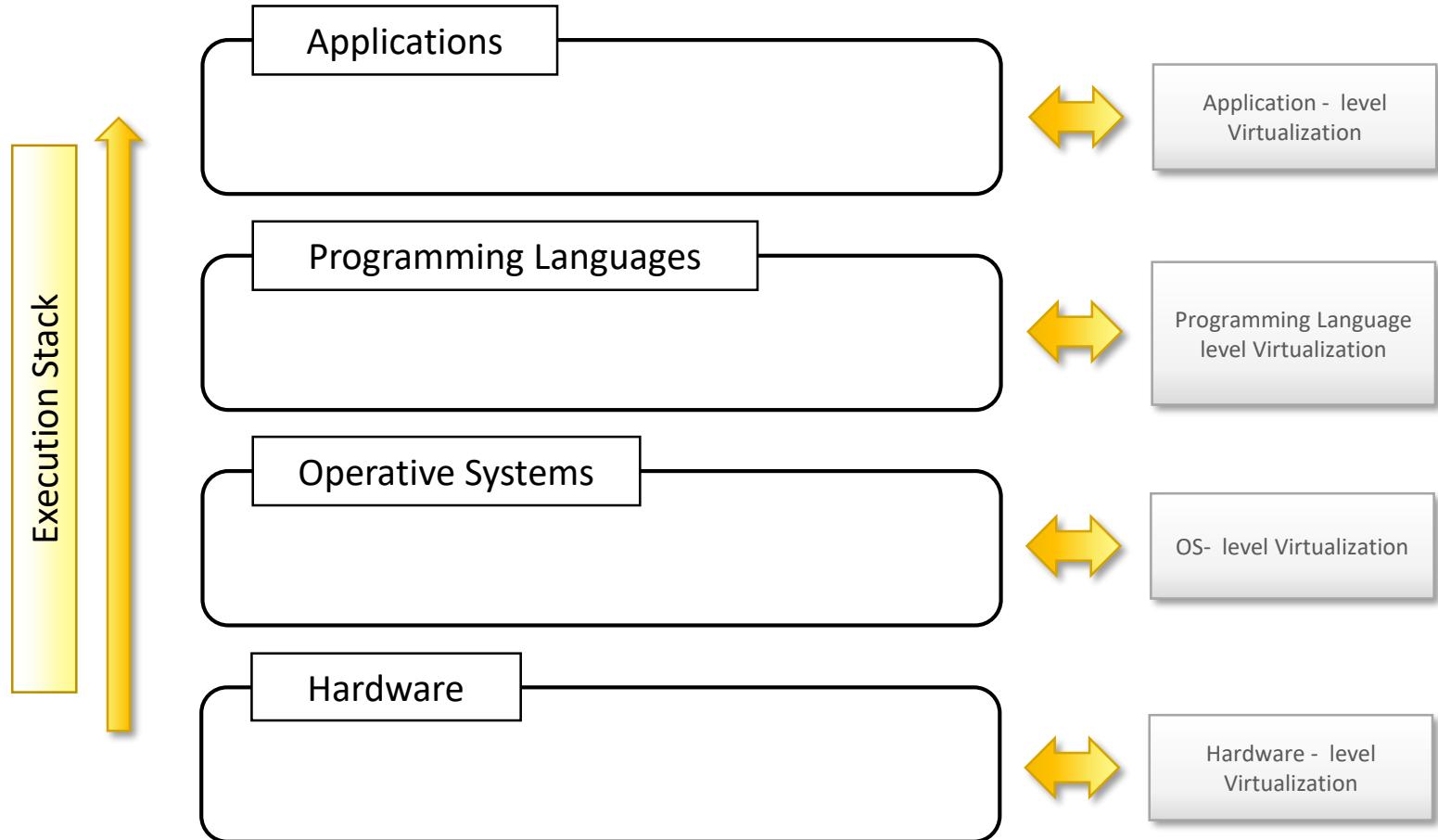
Type-II

- It require the support of an operating system to provide virtualization services.
- Programs managed by the OS.
- Emulate the ISA of virtual h/w.
- Also called hosted virtual machine.

Example: KVM, Virtual Box

an [instruction set architecture \(ISA\)](#), aka [computer architecture](#), defines the supported [instructions](#), [data types](#), [registers](#), the hardware support for managing [main memory](#), fundamental features (such as the [memory consistency](#), [addressing modes](#), [virtual memory](#)), and the [input/output](#) model of a family of implementations of the ISA.

Different levels of Virtualization

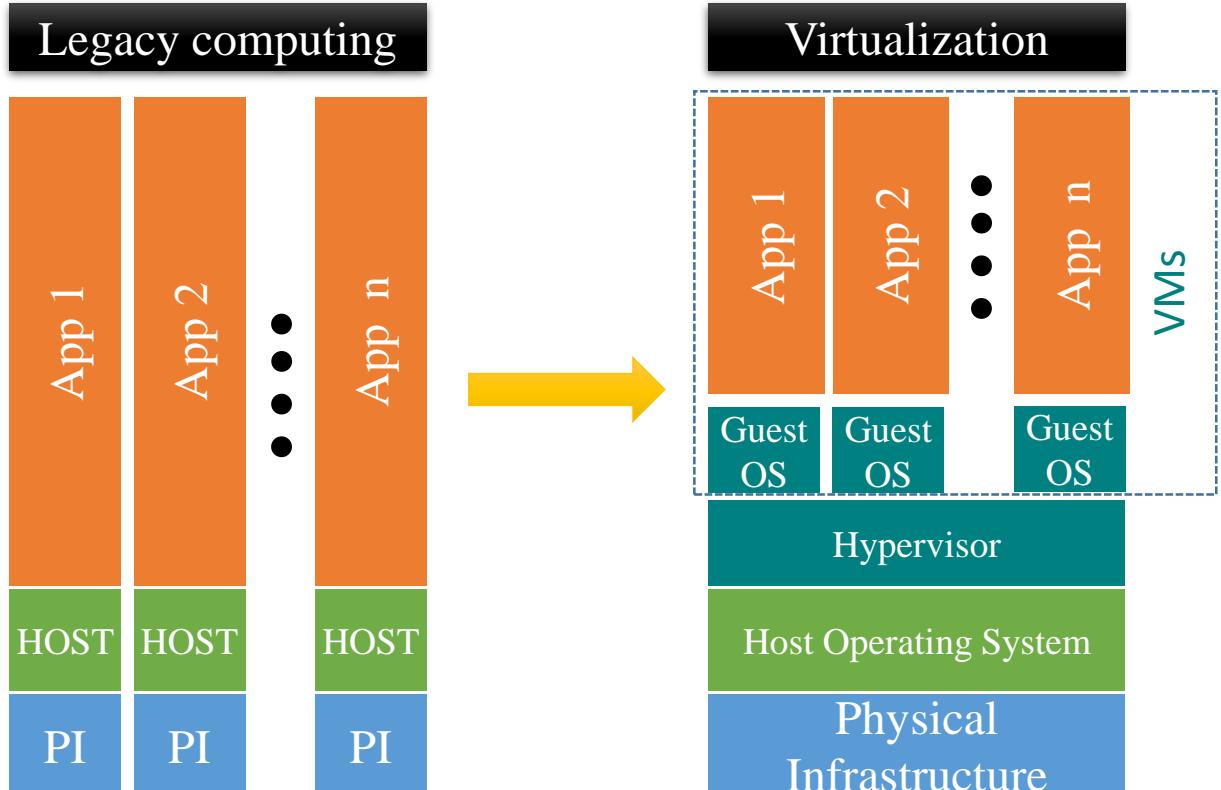


Src: <http://buyya.com/>

Cons of Virtualization....

- Cold start
- Need more storage
 - Less number of virtual machines per PI
- Performance degradation
- Inefficiency and degraded user experience
- As it interposes and abstraction layer between guest & host.
- Some of specific features of the host are unexposed. Security holes and new threats
 - Case 1 - emulating a host in a completely transparent manner.
 - Case 2 - H/w virtualization , malicious programs can preload themselves before the OS and act as a thin VMM.

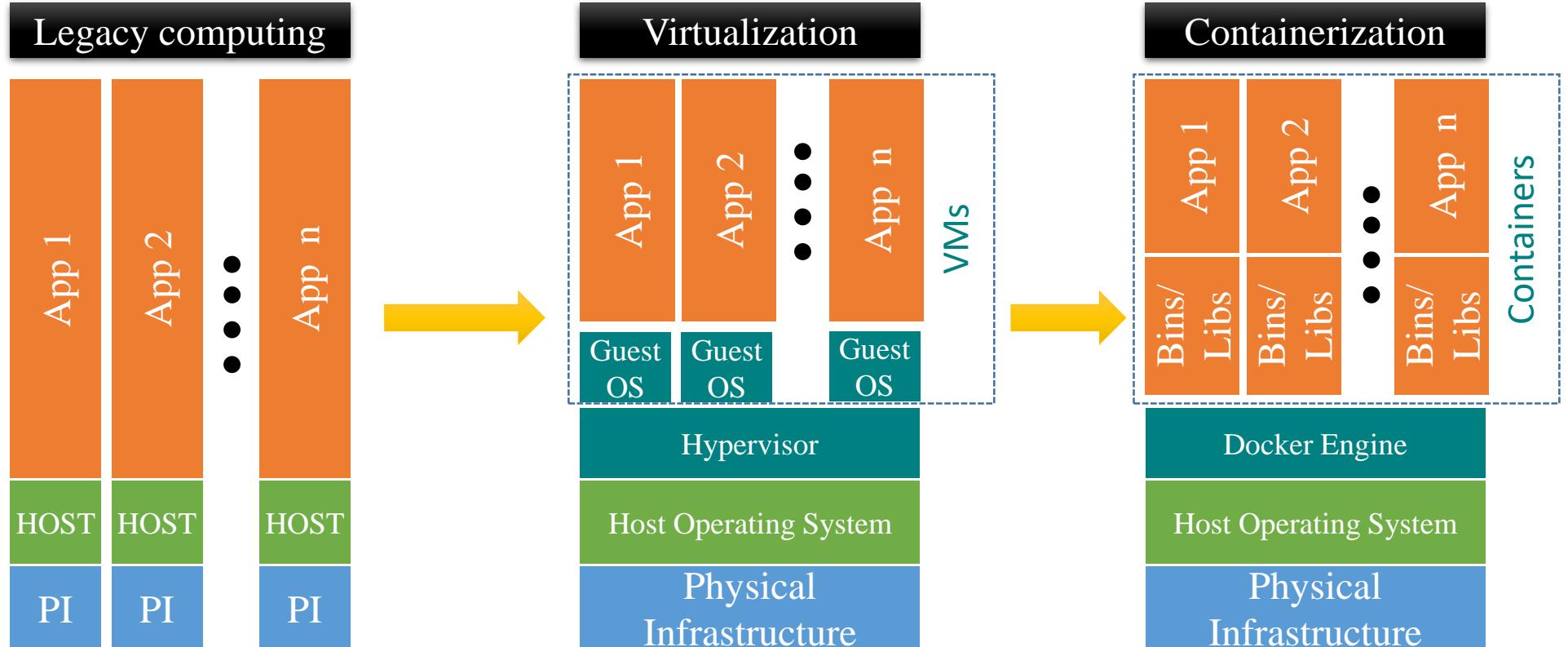
Containerization...



What we need

- Lightweight
- Require less memory space
- Fast launch time
- better resource utilization

Containerization...



- Containerized apps share Host OS's kernel to execute work
- Workload in Containers use Host OS kernel

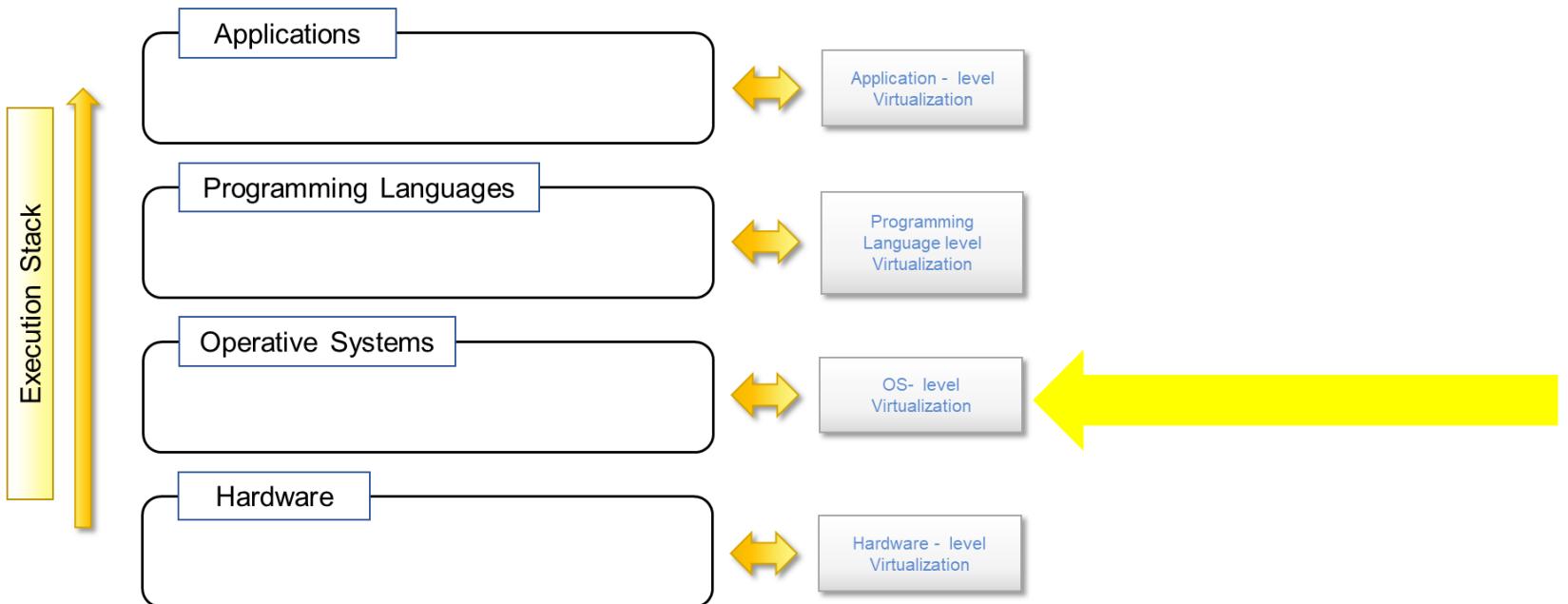
What we need

- Lightweight
- Require less memory space
- Fast launch time
- better resource utilization
- 1000s of containers can be loaded onto a Host

Containers

What is a container?

- LXC, a Linux container, is a Linux operating system-level virtualization method for running multiple isolated Linux based systems on single host.



Containers Benefits

Portability: not tied to or dependent upon the host operating system

Agility: capability to manage various kinds of changes during the development process. Able to respond quickly to the changes...

Speed: less start-up time, most lightweight...

Fault isolation: does not affect other containers...

Efficiency: less start-up time, smaller in capacity, allow more container to run

Ease of management: A container orchestration platform automates the installation, scaling, and management of containerized workloads and services.

Security: due to isolation of applications

Src: <https://www.ibm.com/cloud/learn/containerization>

Containers Benefits - VMs vs Containers

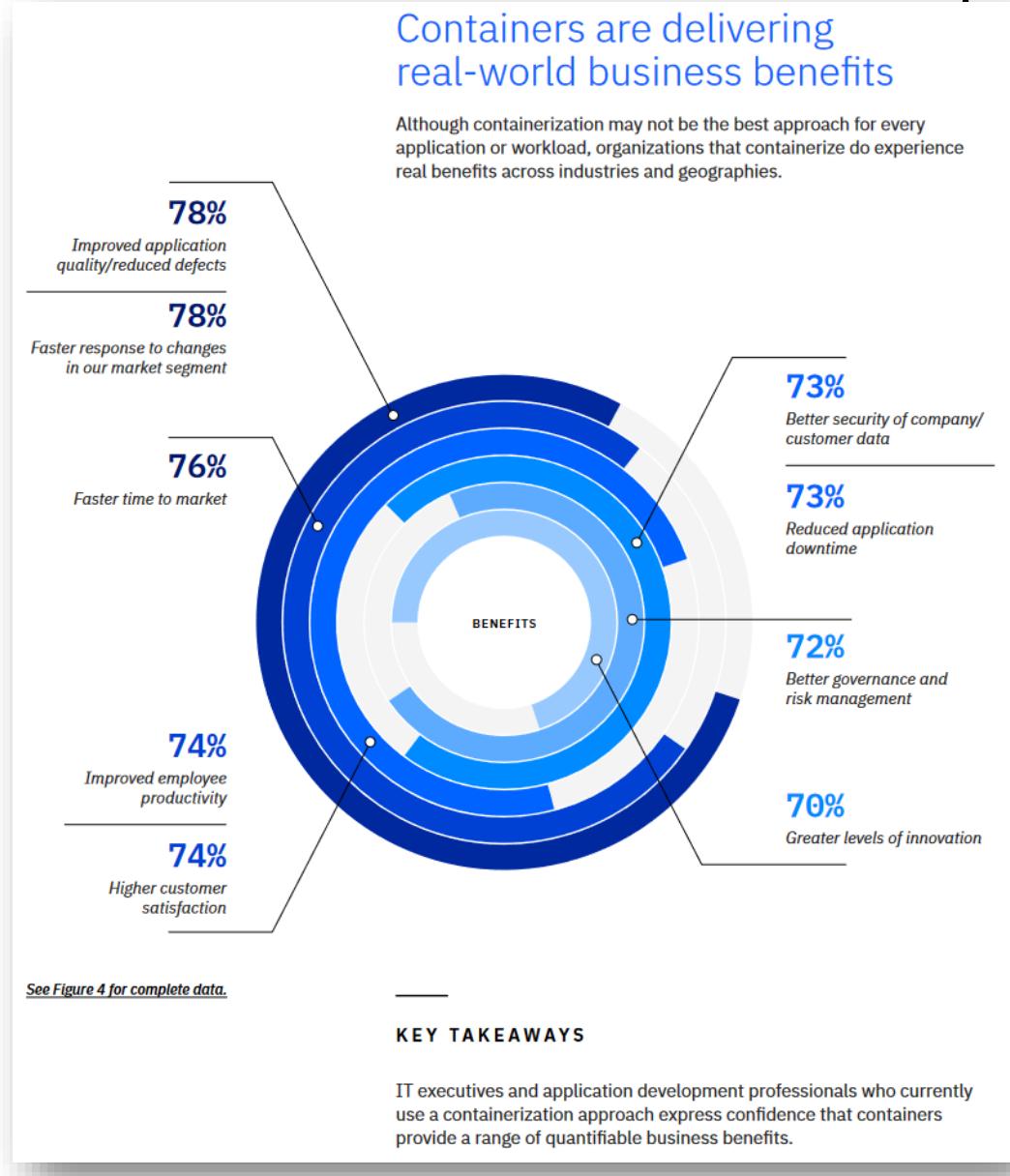
Virtual Machines	Containers
Heavyweight	Lightweight
Limited performance	Native performance
Each VM runs in its own OS	All containers share the host OS
Hardware-level virtualization	OS virtualization
Startup time in minutes	Startup time in milliseconds
Allocates required memory	Requires less memory space
Fully isolated and hence more secure	Process-level isolation, possibly less secure

Docker containers are NOT VMs

- Easy connection to make
- Fundamentally different architectures
- Fundamentally different benefits

Src: <https://www.backblaze.com/blog/vm-vs-containers/>

Containers Benefits - Containers in the enterprise



Report: Containers in the enterprise

url: <https://www.ibm.com/downloads/cas/VG8KRPRM>

Containers

What is a container?

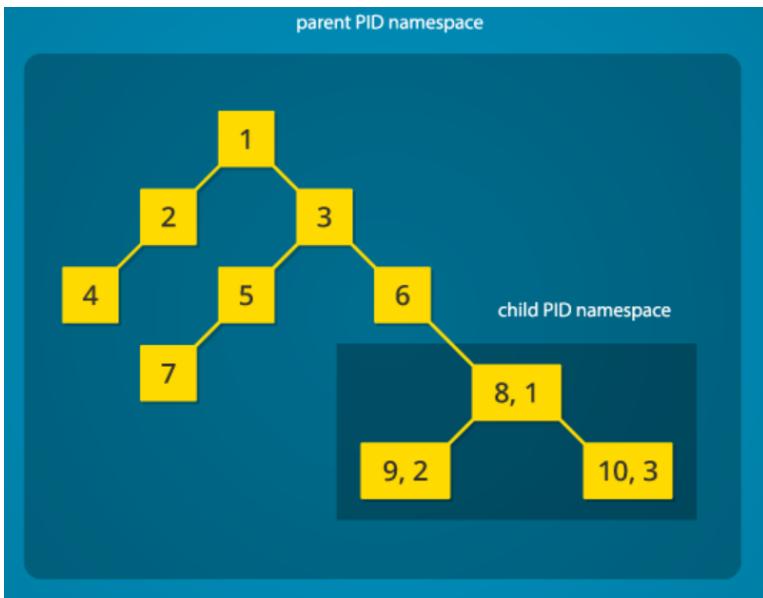
- LXC, Linux container, is a Linux operating system-level virtualization method for running multiple isolated linux based systems on single host controlled and managed by ***Namespaces*** and ***Cgroups***.
- ***Namespaces***: Linux namespace partitions processes and system resources so that only processes in the same namegroup get access to namegroup resources and processes.
- ***Cgroups***: Originally contributed by Google, Cgroups is a Linux kernel concept that governs the isolation and usage of system resources, such as CPU & memory, for a group of processes.

Containers – *Namespaces*

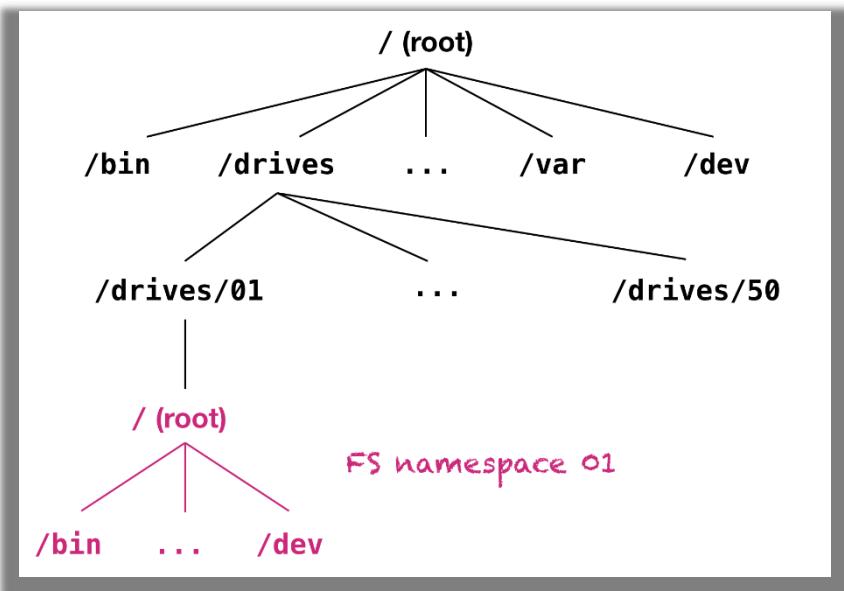
- Namespaces are a feature of the Linux kernel that partitions kernel resources such that one set of processes sees one set of resources while another set of processes sees a different set of resources.

Examples: PID(Process Id), MNT(Mount file/folder), IPC,NET(Individual port and IP)

Process Id namespace



Filesystem namespace

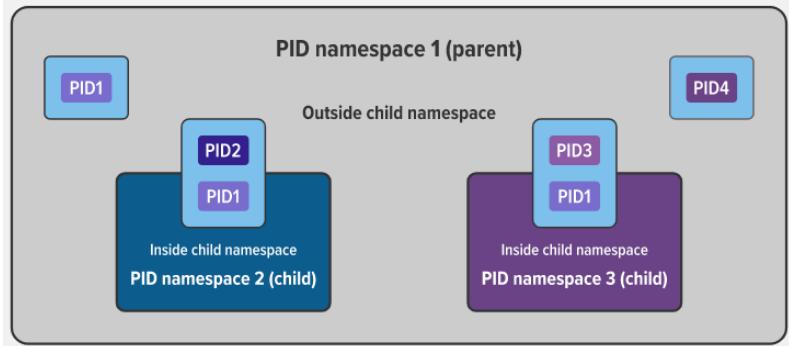
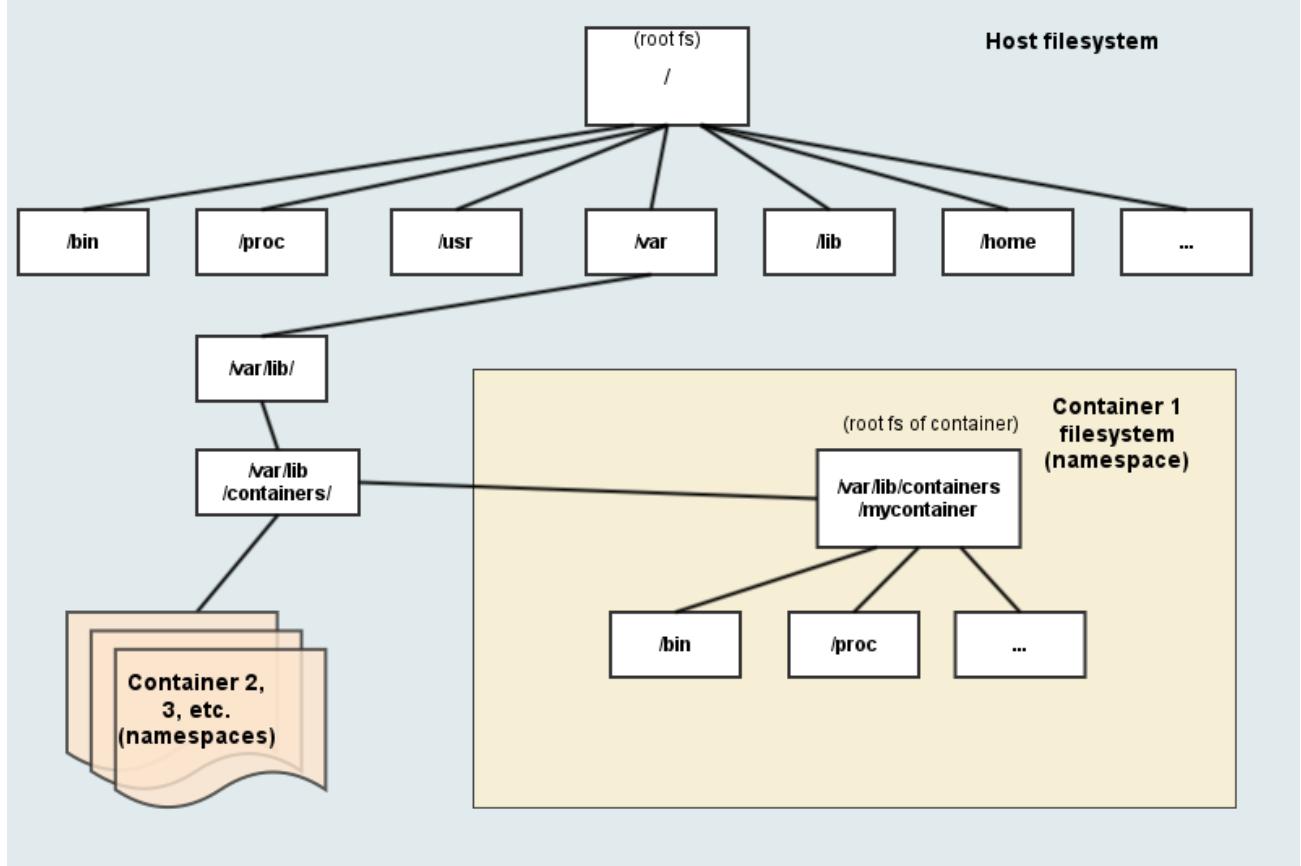


Src:

<https://www.toptal.com/linux/separation-anxiety-isolating-your-system-with-linux-namespaces>

<https://blog.codecentric.de/en/2019/06/docker-demystified/>

Broad view of Filesystem namespace



<https://www.nginx.com/blog/what-are-namespaces-cgroups-how-do-they-work/>

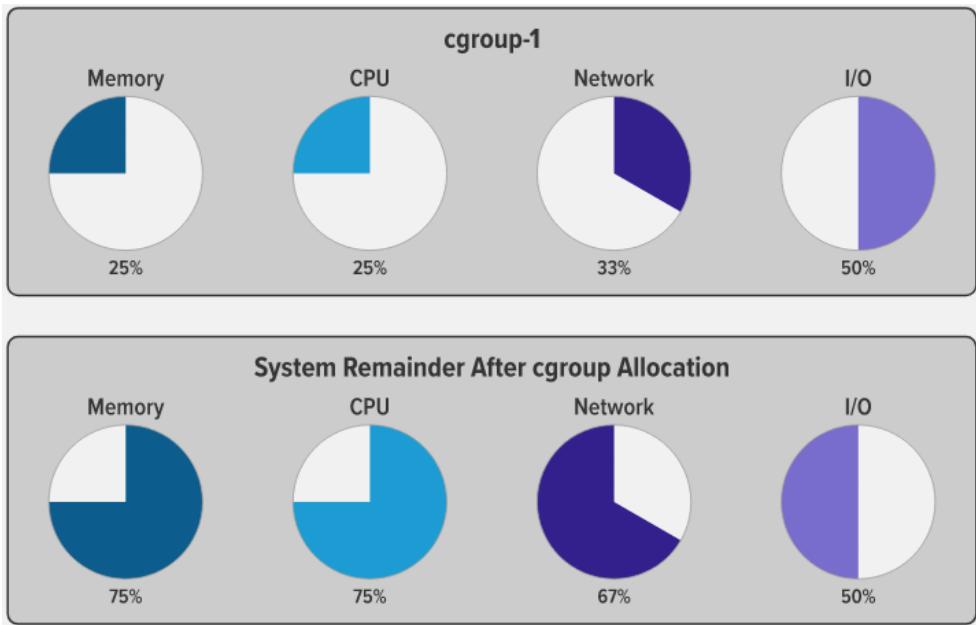
Src:

<https://www.toptal.com/linux/separation-anxiety-isolating-your-system-with-linux-namespaces>

<https://blog.codecentric.de/en/2019/06/docker-demystified/>

Containers – *Cgroups*

“... cgroups, a feature for isolating and controlling resource usage (e.g., how much CPU and RAM and how many threads a given process can access) within the Linux kernel. ...”



Cgroups provide the following features:

Resource limits – limit how much of a particular resource (memory or CPU, for example) a process can use.

Prioritization – control how much of a resource a process can use compared to processes in another cgroup...

Accounting – Resource limits are monitored and reported at the cgroup level.

Control – You can change the status (frozen, stopped, or restarted) of all processes in a cgroup with a single command.

Some container runtime platforms...

- Docker
- CoreOS rkt
- Mesos
- LXC
- OpenVZ
- Containerd
- Windows Server Containers
- Linux VServer
- Hyper-V Containers
- Unikernels
- Java containers

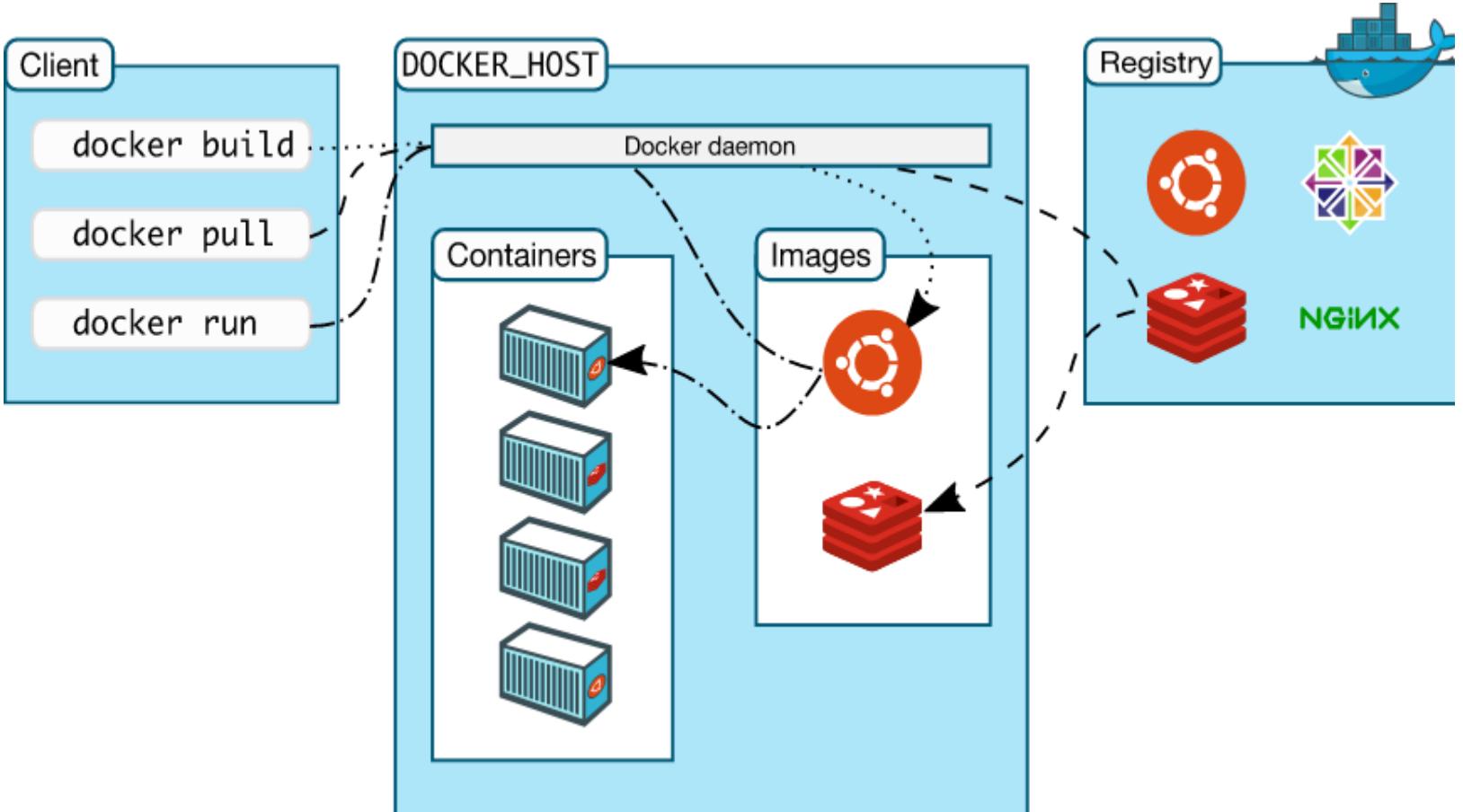
Some container runtime platforms...

- **Docker**
- CoreOS rkt
- Mesos
- LXC
- OpenVZ
- Containerd
- Windows Server Containers
- Linux VServer
- Hyper-V Containers
- Unikernels
- Java containers

Docker

- Docker container technology was launched in 2013 as an *open source Docker Engine*.
- Docker enterprise edition introduced in 2016 as a first commercial product.
- Docker is written in the **Go programming language**
- When you run a container, Docker creates a set of *namespaces* for that container.

Docker Architecture

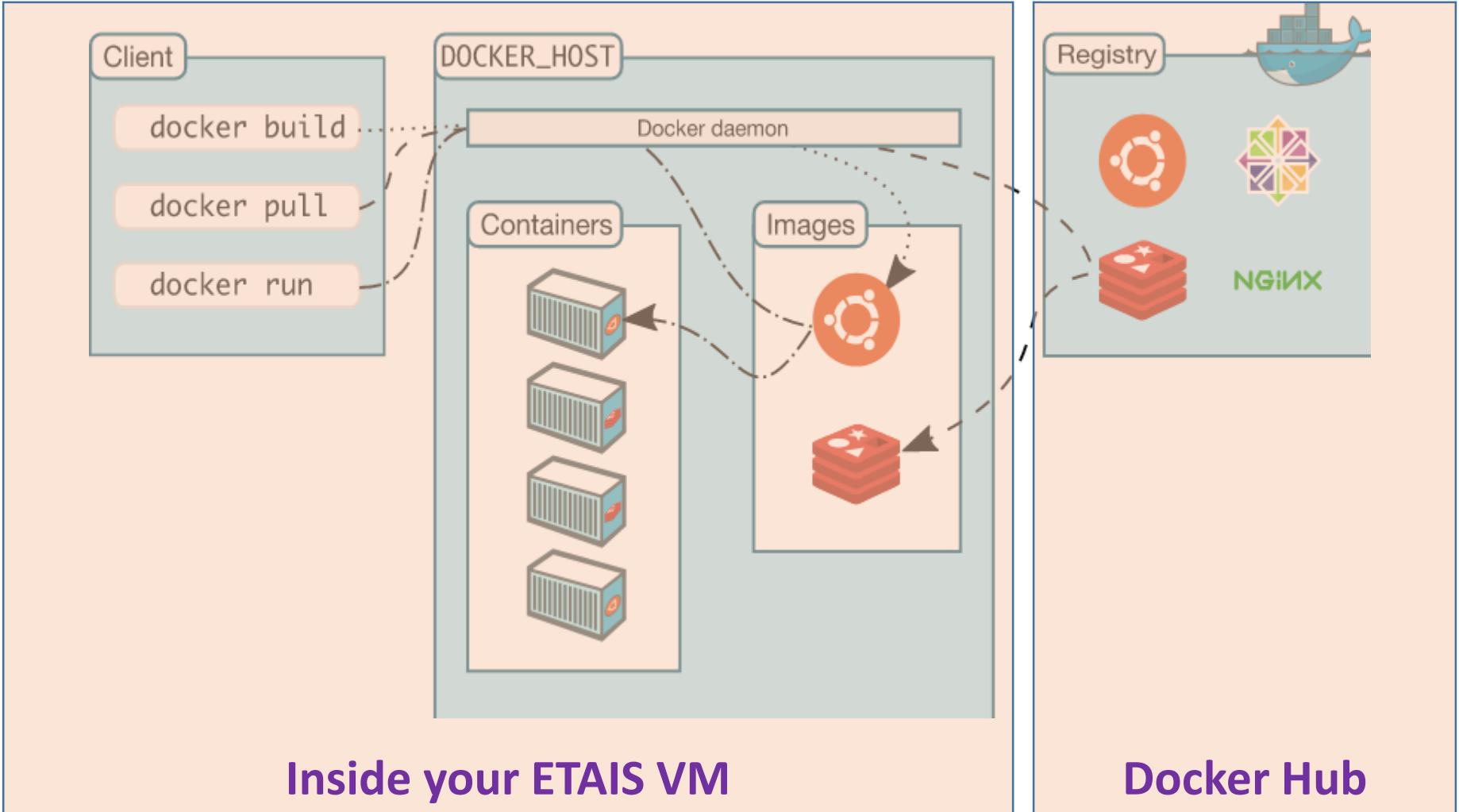


Src: <https://docs.docker.com/get-started/overview/>

LTAT.06.015 : Lec-03 : Containerization

27

Docker Architecture



Src: <https://docs.docker.com/get-started/overview/>

LTAT.06.015 : Lec-03 : Containerization

28

Docker – terminologies/vocabulary

The Role of Images and Containers



Docker Image

Docker Image

- The basis of a Docker container
- Images are read only templates build from Dockerfile.
- Docker uses Union File System.
 - Duplication-free
 - Layer segregation



Docker Container

Docker Container:

- The image when it is running
- The standard unit for application service.
- Runs your application.

Example: Ubuntu with Node.js and Application Code

Src: <https://f.ch9.ms/thumbnail/ff8e0db3-f25d-4e9f-b68e-5ae38b411d79.pptx>

Docker – terminologies/vocabulary



Docker Engine

Creates, ships and runs Docker containers deployable on a physical or virtual, host locally, in a datacenter or cloud service provider



Registry Service (Docker Hub(Public) or Docker Trusted Registry(Private))

Cloud or server based storage and distribution service for your images

Basic Docker Commands

- docker --version
- docker pull <image name>
- docker run -it -d <image name>
- docker ps
- docker image ls
- docker exec -it <container id> <command>
- docker stop <container id>
- docker kill <container id>
- docker login

Basic Docker Commands



Build

Build an image from the Dockerfile in the current directory and tag the image

```
docker build -t myimage:1.0 .
```

List all images that are locally stored with the Docker Engine

```
docker image ls
```

Delete an image from the local image store

```
docker image rm alpine:3.4
```



Run

Run a container from the Alpine version 3.9 image, name the running container "web" and expose port 5000 externally, mapped to port 80 inside the container.

```
docker container run --name web -p 5000:80 alpine:3.9
```

Stop a running container through SIGTERM

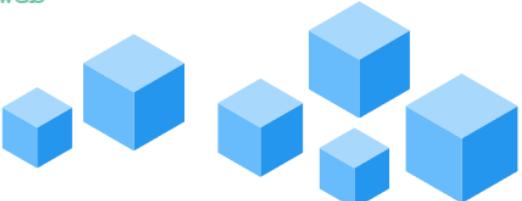
```
docker container stop web
```

Stop a running container through SIGKILL

```
docker container kill web
```

List the networks

```
docker network ls
```



Share

Pull an image from a registry

```
docker pull myimage:1.0
```

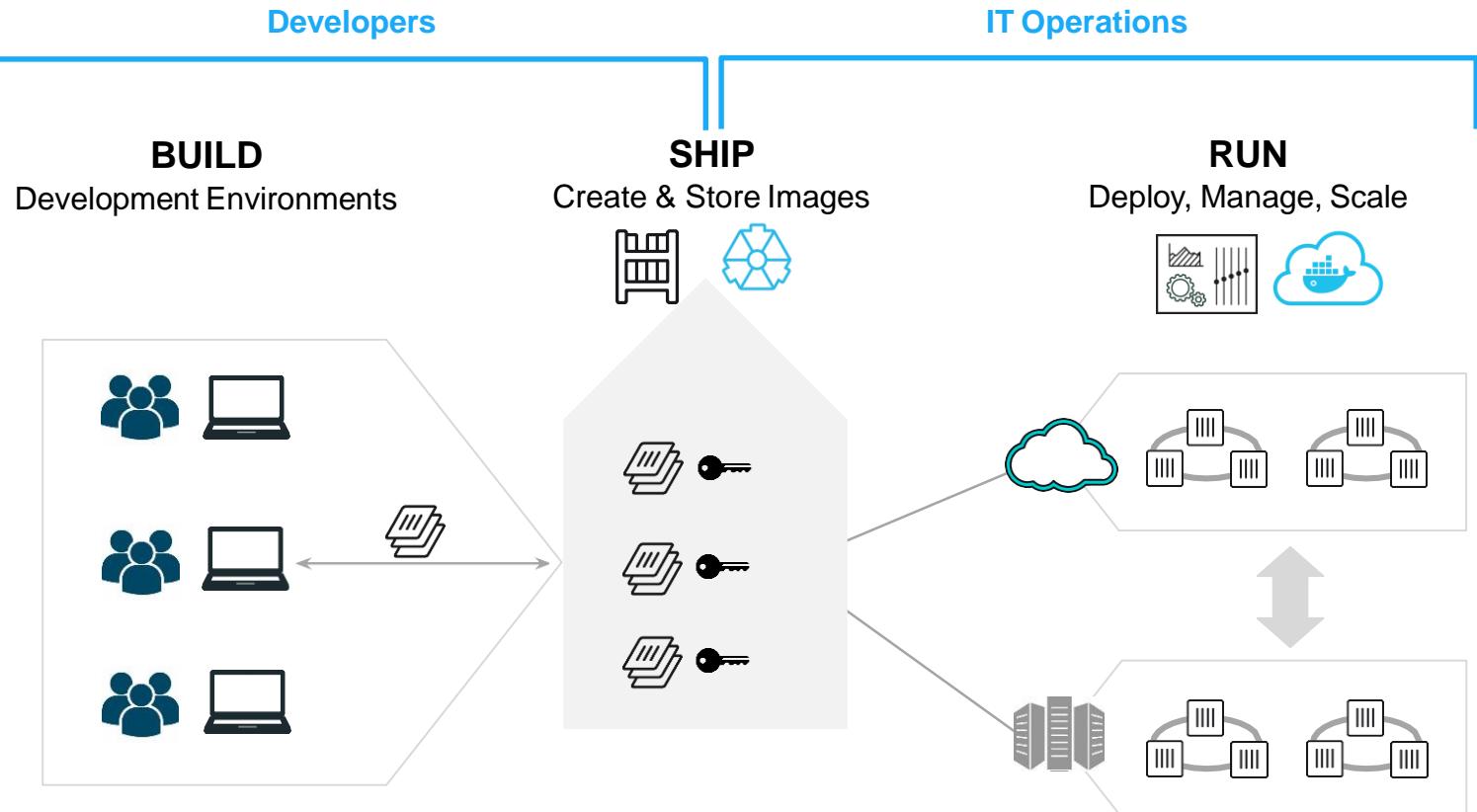
Retag a local image with a new image name and tag

```
docker tag myimage:1.0 myrepo/myimage:2.0
```

Push an image to a registry

```
docker push myrepo/myimage:2.0
```

Docker – in DevOps



Dockerfile basic

- *Dockerfile*, a text file that includes specific keywords that dictate how to build a specific image.
- Docker builds images automatically by reading the instructions from a *Dockerfile*.
- An Example of *dockerfile*

```
FROM python:3.8-slim-buster
WORKDIR /app
COPY requirements.txt requirements.txt
RUN pip3 install -r requirements.txt
COPY .
CMD [ "python3", "-m" , "flask", "run", "--host=0.0.0.0"]
```

FROM : creates a layer from the ubuntu:18.04 Docker image.

WORKDIR : sets the path where the command, defined with CMD, is to be executed.

COPY : adds files from your Docker client's current directory.

RUN : builds your application with make.

CMD : specifies what command to run within the container.

Src: https://docs.docker.com/develop/develop-images/dockerfile_best-practices/

Dockerfile basic

- An Example of *dockerfile*

```
FROM python:3.8-slim-buster
WORKDIR /app
COPY requirements.txt requirements.txt
RUN pip3 install -r requirements.txt
COPY . .
CMD [ "python3", "-m", "flask", "run", "--host=0.0.0.0" ]
```

FROM : creates a layer from the ubuntu:18.04 Docker image.

WORKDIR : sets the path where the command, defined with **CMD**, is to be executed.

COPY : adds files from your Docker client's current directory.

RUN : builds your application with make.

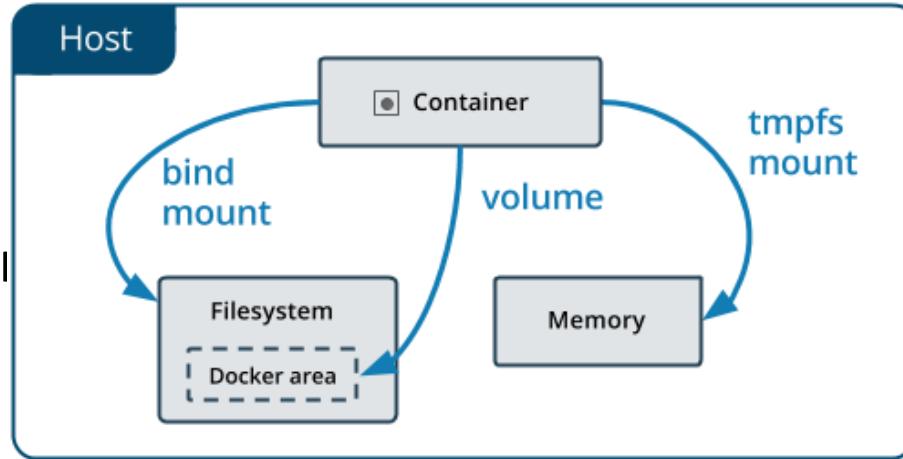
CMD : specifies what command to run within the container.

Build the docker file: `docker build -t myflask .`

Src: https://docs.docker.com/develop/develop-images/dockerfile_best-practices/

Manage data in Docker

- By default all files created inside a container are stored on a writable container layer
 - The data doesn't persist when that container no longer exists
 - A container's writable layer is tightly coupled to the host machine
- Docker has two options for containers to store files in the host machine
 - *Volumes*
 - *bind mounts*



Manage data in Docker - Volumes

- *Volumes* are the preferred mechanism for persisting data generated by and used by Docker containers.
- Volumes are completely managed by Docker.

Create a volume:

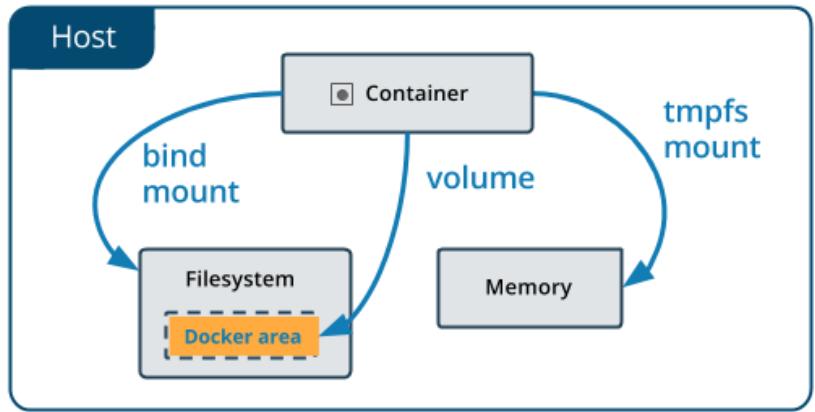
```
docker volume create vol2
```

Remove a volume:

```
docker volume rm vol2
```

List volumes:

```
docker volume ls
```



Start a container with a volume

```
docker run -d --name mywebapp -v vol2:/app myFlask:latest
```

OR

```
docker run -d --name mywebapp --mount source=vol2,target=/app myFlask:latest
```

Manage data in Docker - bind mounts

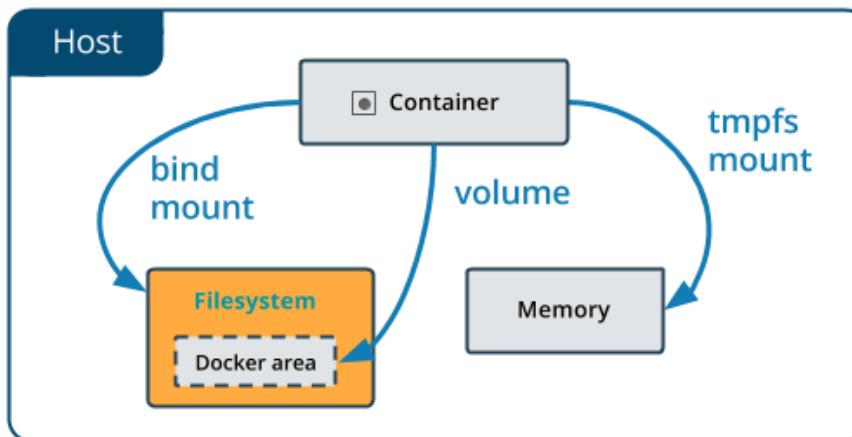
- When you use a bind mount, a file or directory on the *host machine* is mounted into a container.
- The file or directory is referenced by its absolute path on the host machine.

Start a container with a bind mount

```
docker run -d --name mywebapp -v "$(pwd)"/target:/app myFlask:latest
```

OR

```
docker run -d --name mywebapp --mount type=bind,source="$(pwd)"/target,target=/app myFlask:latest
```



Manage data in Docker - tmpfs mounts

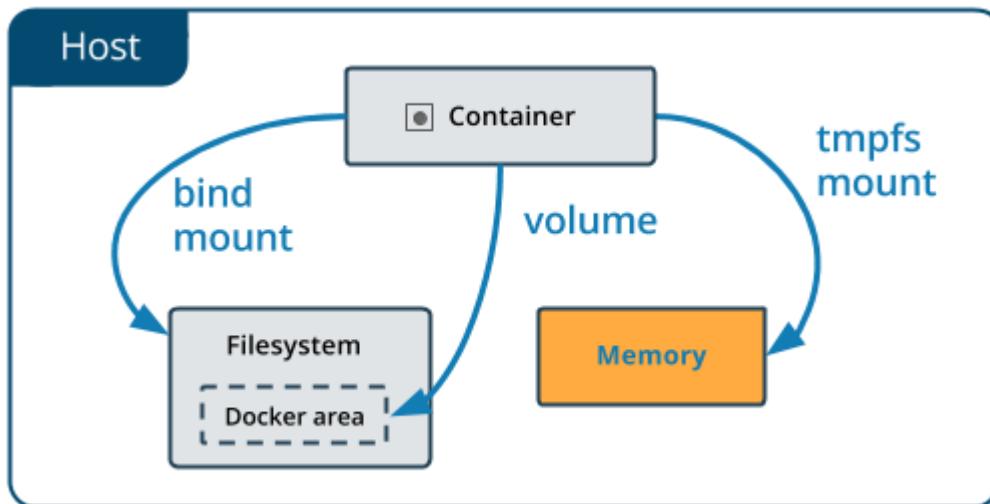
- *Volumes* and *bind mounts* let you share files between the host machine and container so that you can persist data even after the container is stopped.
- The third option *tmpfs mounts* option is only available to *Docker on Linux* [[src](#)].
- As opposed to volumes and bind mounts, a tmpfs mount is *temporary*
- When the container stops, the tmpfs mount is removed, and files written there won't be persisted.
- *Useful* to temporarily store sensitive files

Start a container with a bind mount

```
docker run -d -it --name mywebapp --tmpfs /app myFlask:latest
```

OR

```
docker run -d --name mywebapp \
--mount type=tmpfs,destination=/app \
myFlask:latest
```



Who is using Docker...



How would you manage large number of containers...?

- Manage state / health / lifecycle
- Manage networking, discoverability, etc.
- Manage sensitive data
- Manage scale

Container Orchestration

- Fault-tolerance
- On-demand scalability
- Optimal resource usage
- Auto-discovery to automatically discover and communicate with each other
- Accessibility from the outside world
- Seamless updates/rollbacks without any downtime.

How would you manage large number of container...?

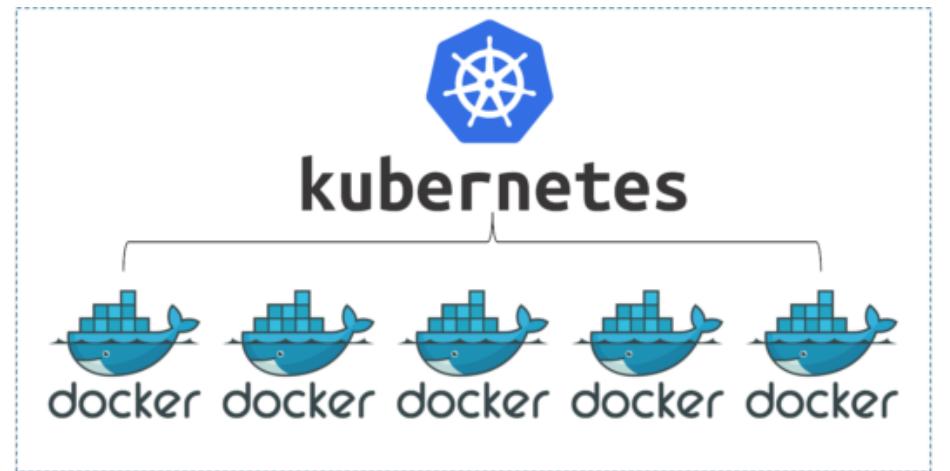
Container Management Solutions

Kubernetes

Kubernetes

[Kubernetes](#), also known as K8s, is an open-source system for automating deployment, scaling, and management of containerized applications.

- Matured open source project
- Based on learning from internal Google projects and the concept of Google Borg(2003-2004)
- Kubernetes initial release 7 June 2014
- Releases every 3 months
- Very large developer base and interest
- Not necessarily just about containers



Img src: <https://www.edureka.co/blog/kubernetes-tutorial/>

Some Kubernetes alternatives

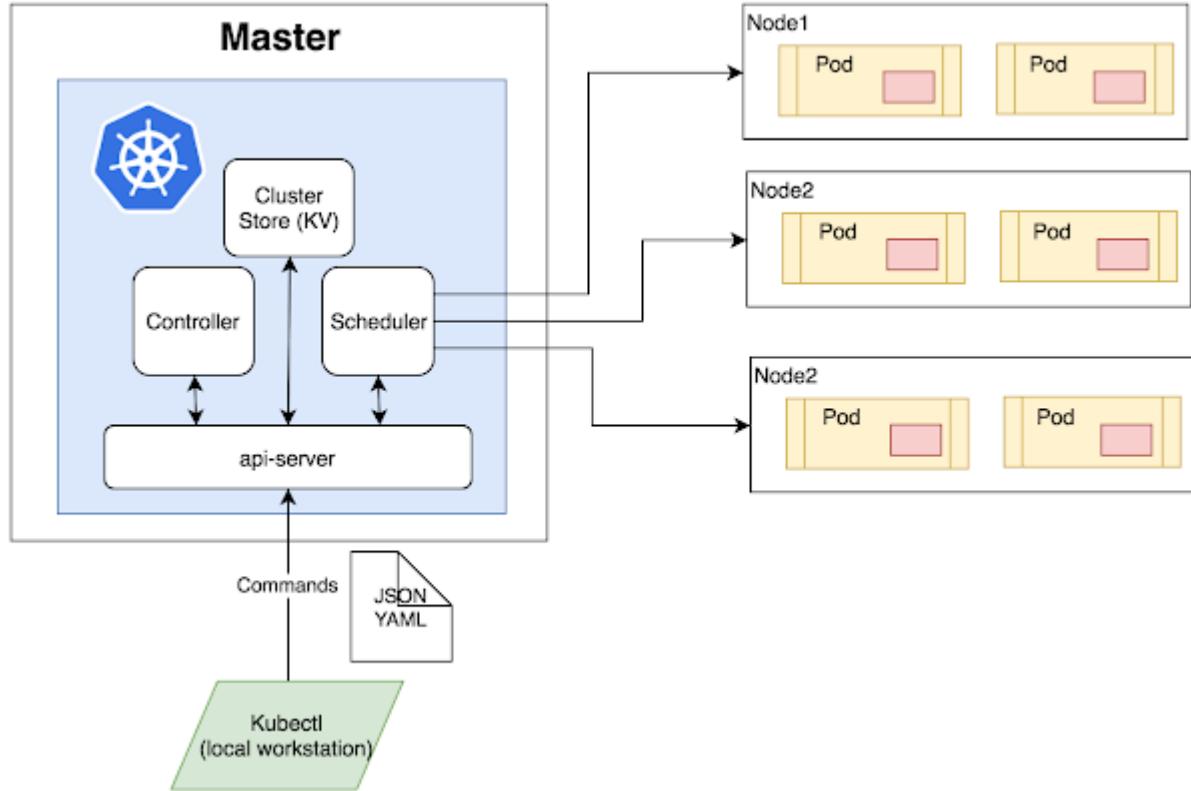
- **Amazon Elastic Container Service**
- **Azure Container Instances**
- **Azure Service Fabric**
- **Marathon**
- **Nomad**
- **Docker Swarm**

Kubernetes features

- ✓ **Service discovery and load balancing**
- ✓ **Storage orchestration**
- ✓ **Secret and configuration management**
- ✓ **Automatic bin packing**
- ✓ **Horizontal scaling**
- ✓ **Self-healing**
- ✓ **Designed for extensibility**

Src: <https://kubernetes.io/>

Kubernetes Components



Cluster:

It is a collection of hosts(servers) that helps you to aggregate their available resources. That includes ram, CPU, ram, disk, and their devices into a usable pool.

Master:

The master is a collection of components which make up the control panel of Kubernetes. These components are used for all cluster decisions. It includes both scheduling and responding to cluster events.

Node:

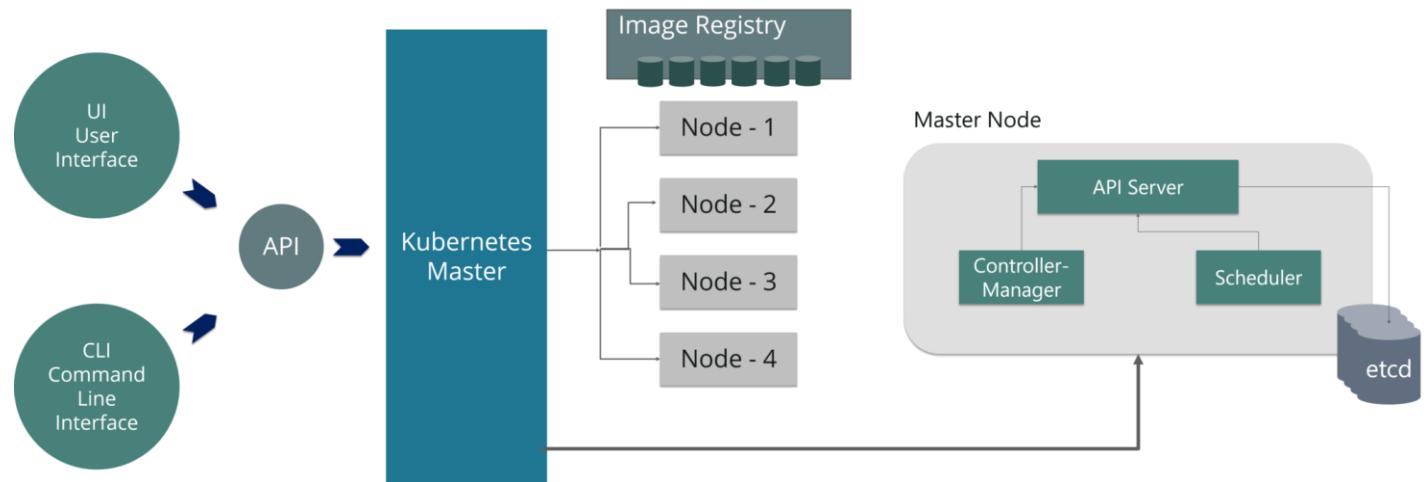
It is a single host which is capable of running on a physical or virtual machine. A node should run both kube-proxy and kubelet which are considered as a part of the cluster.

Namespace:

It is a logical cluster or environment. It is a widely used method which is used for scoping access or dividing a cluster.

Src: <https://www.guru99.com/kubernetes-tutorial.html>

Kubernetes - Master nodes



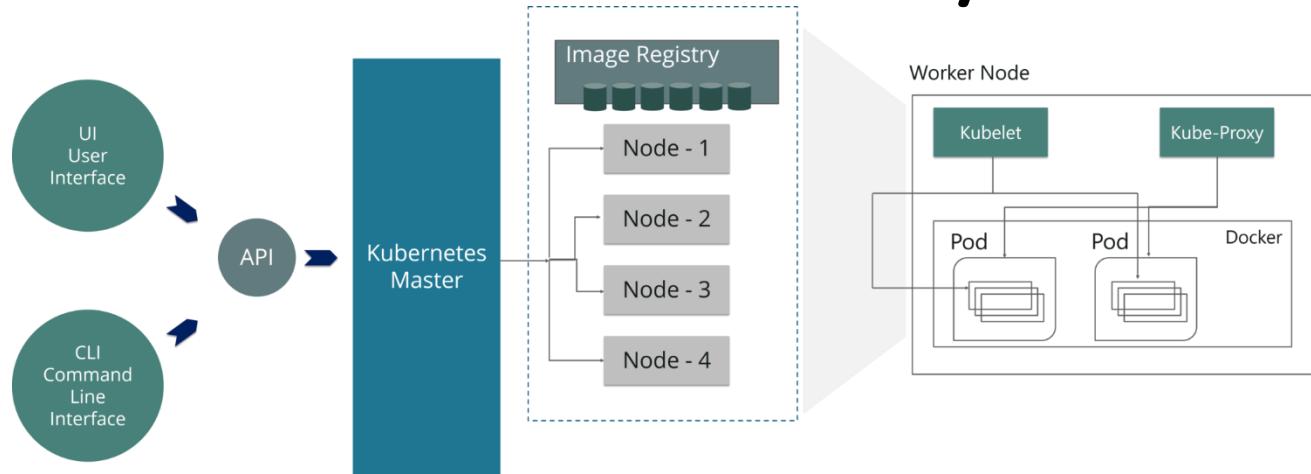
Master Node

- responsible for the management of Kubernetes cluster
- **API-Server:** entry point for all kind of administrative tasks
- >1 master node for fault tolerance
- **Scheduler:**
 - schedules the tasks to slave nodes.
 - stores the resource usage information for each slave node
- **ETCD:**
 - ETCD is a simple, distributed, consistent key-value store.
 - used for shared configuration and service discovery

Src: <https://www.edureka.co/blog/kubernetes-tutorial/>

- **Controller Manager:**
 - regulates the Kubernetes cluster
 - tells Nodes what to run, how to expose applications, how-to commit changes and so on

Kubernetes - Worker/Slave nodes



Kubelet: gets the configuration of a Pod from the API server and ensures that the described containers are up and running.

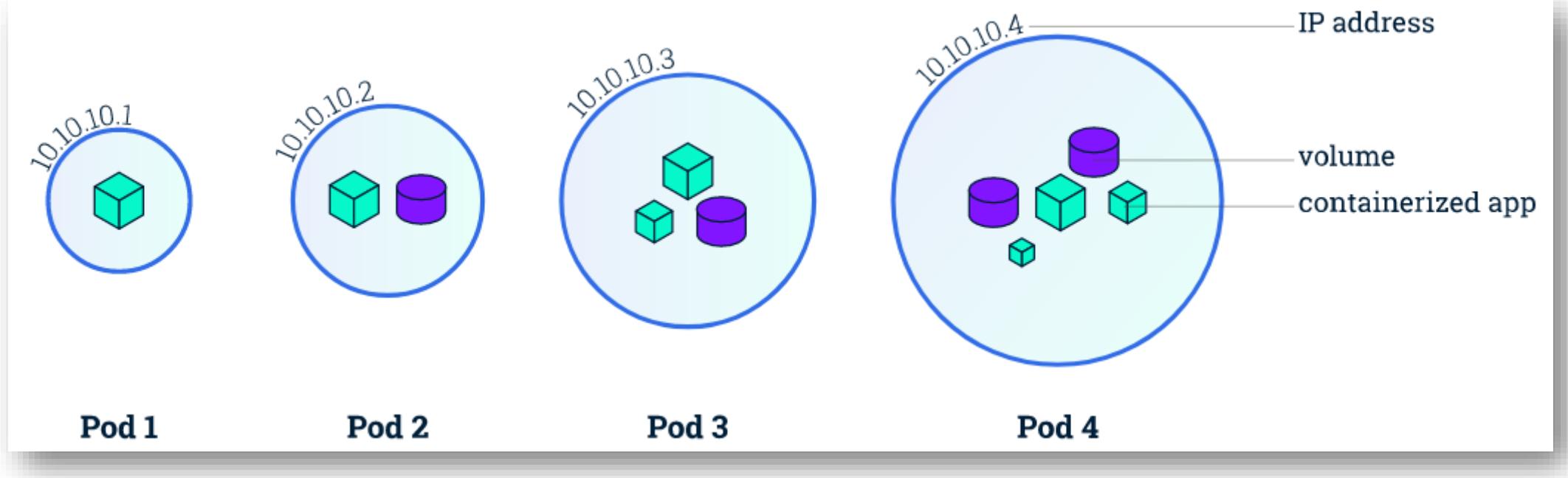
Docker Container: Docker container runs on each of the worker nodes, which runs the configured pods

Kube-proxy: Kube-proxy acts as a load balancer and network proxy to perform service on a single worker node

Pods: A pod is a combination of single or multiple containers that logically run together on nodes

Src: <https://www.edureka.co/blog/kubernetes-tutorial/>

Kubernetes - Pod



Pods run on worker nodes

- It can be a single container or multiple containers
- When we do scaling, we scale Pods
- When we do a rolling update then we update Pod by Pod
- One pod can only run on one worker node at a time
- Two pods of the same Deployment usually run on different worker nodes

Kubernetes - Pod

```

apiVersion: batch/v1
kind: Job
metadata:
  name: hello
spec:
  template:
    # This is the pod template
    spec:
      containers:
        - name: hello
          image: busybox
          command: ['sh', '-c', 'echo "Hello, Kubernetes!" && sleep 3600']
    restartPolicy: OnFailure
    # The pod template ends here
  
```

Example - 1

```

apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
  labels:
    app: nginx
spec:
  containers:
    - name: nginx
      image: nginx:1.15.11
      ports:
        - containerPort: 80
  
```

Example - 2

kubectl get pods # List all pods in the namespace

```
$ kubectl get pods
NAME                  READY   STATUS    RESTARTS   AGE
kubernetes-bootcamp-fb5c67579-pxstk   1/1     Running   0          5m58s
```

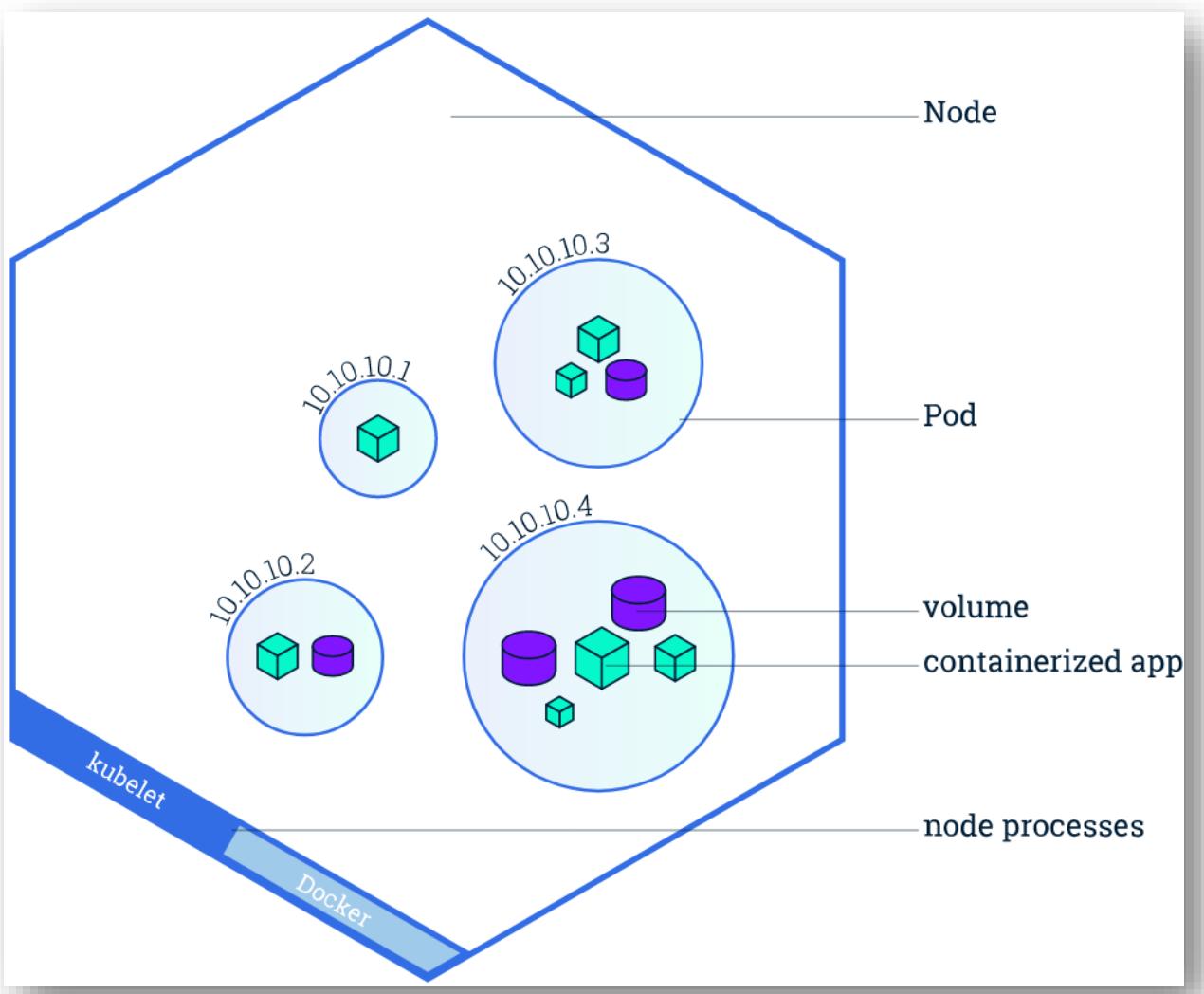
kubectl get pods -o wide # List all pods in the current namespace, with more details

Kubernetes - Pod

```
$ kubectl describe pods
Name:           kubernetes-bootcamp-fb5c67579-pxstk
Namespace:      default
Priority:       0
Node:           minikube/172.17.0.99
Start Time:     Tue, 21 Sep 2021 08:09:58 +0000
Labels:         app=kubernetes-bootcamp
                pod-template-hash=fb5c67579
Annotations:    <none>
Status:         Running
IP:             172.18.0.4
IPs:
  IP:          172.18.0.4
Controlled By: ReplicaSet/kubernetes-bootcamp-fb5c67579
Containers:
  kubernetes-bootcamp:
    Container ID:   docker://dc01480f02574e66127f7124580766332166d97f5982acd7de0e4f35e78979f1
    Image:          gcr.io/google-samples/kubernetes-bootcamp:v1
    Image ID:       docker-pullable://jocatalin/kubernetes-bootcamp@sha256:0d6b8ee63bb57c5f5b6156f446b3bc3b3c143d23303
    7f3a2f00e279c8fcc64af
    Port:          8080/TCP
    Host Port:     0/TCP
    State:         Running
    Started:       Tue, 21 Sep 2021 08:10:04 +0000
    Ready:          True
    Restart Count: 0
    Environment:   <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-p5k8h (ro)
Conditions:
```

Kubernetes - Node

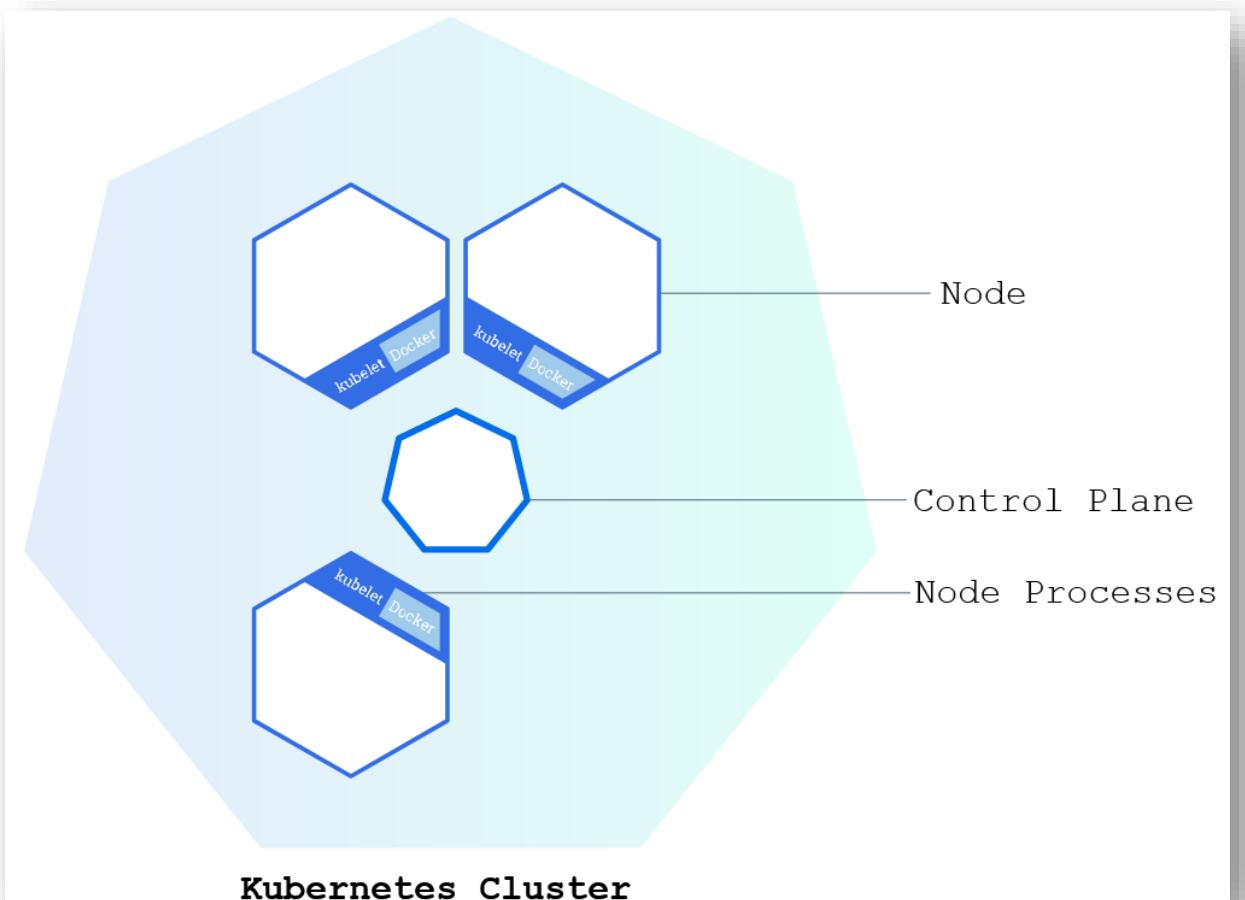
```
$ kubectl get nodes
NAME      STATUS    ROLES          AGE      VERSION
minikube  Ready     control-plane,master  15m     v1.20.2
```



<https://kubernetes.io/docs/tutorials/kubernetes-basics/explore/explore-intro/>

Src: <https://raw.githubusercontent.com/riigipilv/trainings/master/abc/Slaivid.pdf>

Kubernetes - Cluster



```
$ kubectl cluster-info
Kubernetes control plane is running at https://172.17.0.108:8443
KubeDNS is running at https://172.17.0.108:8443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy
```

Src: <https://kubernetes.io/docs/tutorials/kubernetes-basics/explore/explore-intro/>

LTAT.06.015 : Lec-03 : Containerization

55

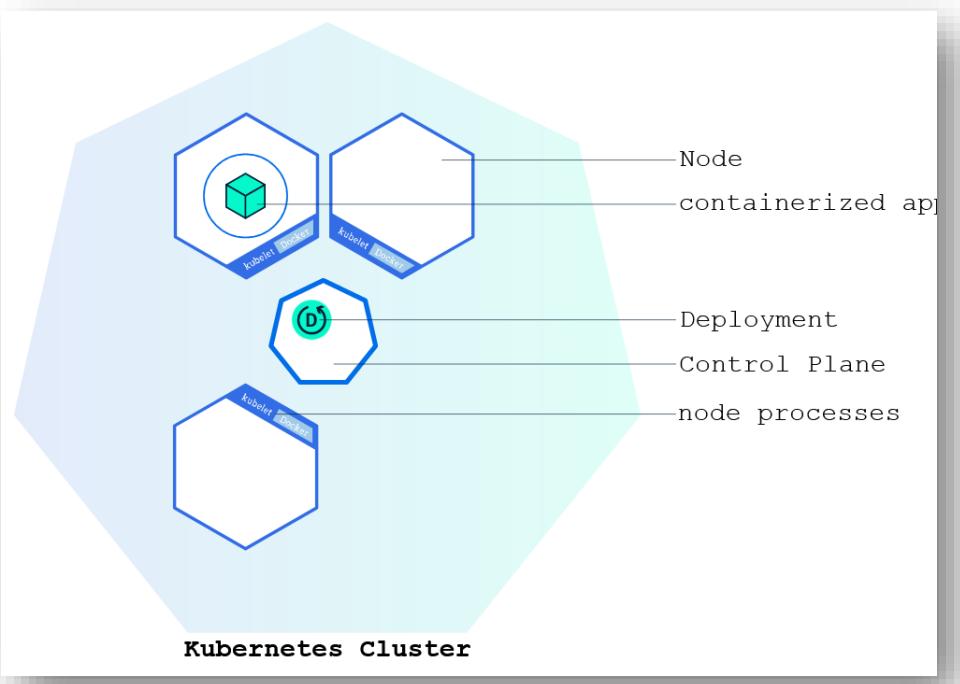


Kubernetes – App, Deployment, Services

App: Your application is expected to be containerized.

Deployment:

- allow updates to a set of pods at a specified rate. They are needed for rolling updates, rolling back, auto scaling. Manages ReplicaSets
- When we delete a Deployment, then the Pods of this Deployment get deleted



```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.14.2
          ports:
            - containerPort: 80
  
```

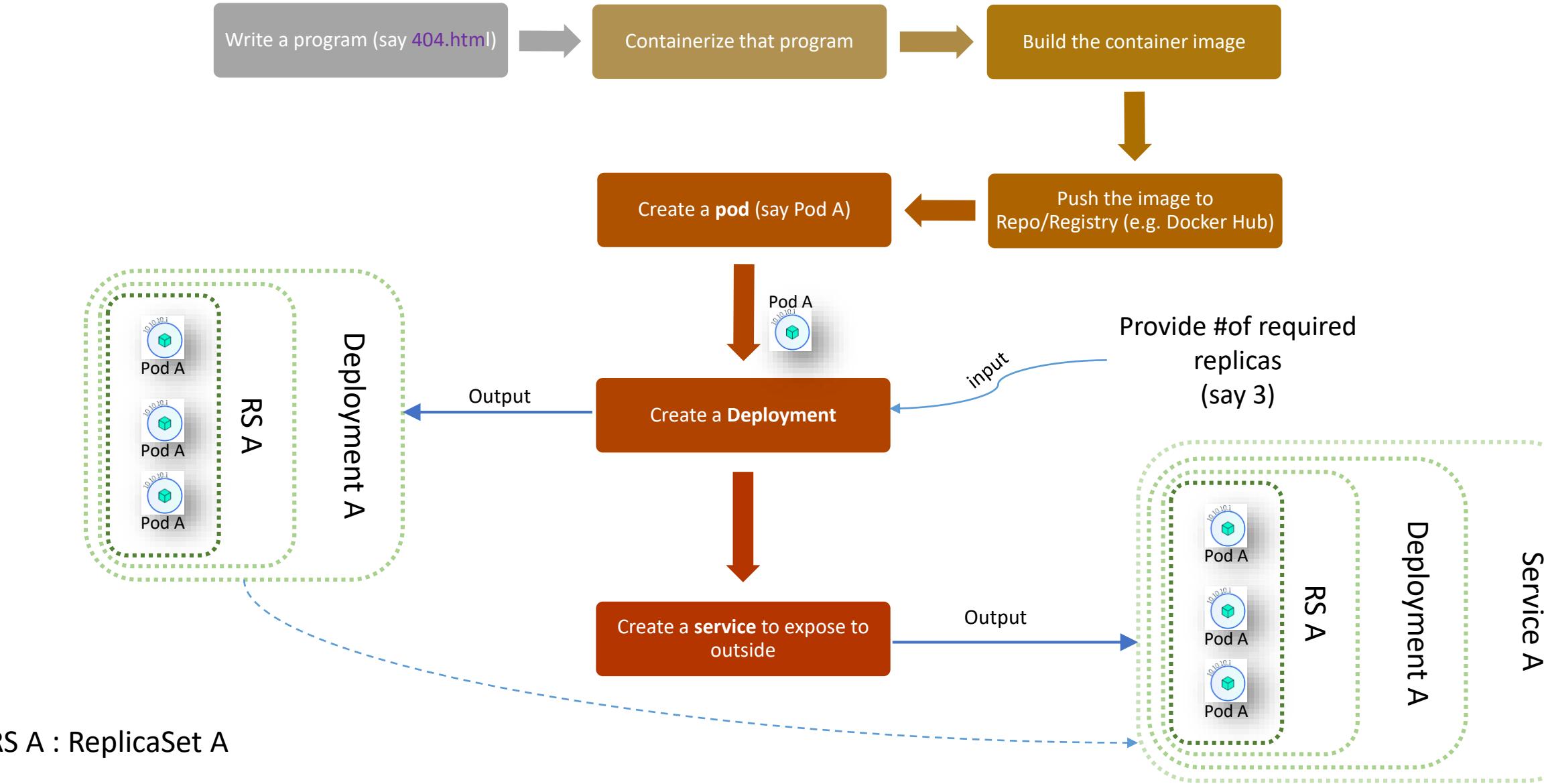
```
$ kubectl get deployments
NAME           READY   UP-TO-DATE   AVAILABLE   AGE
kubernetes-bootcamp   1/1     1           1          106s
```

```
$ kubectl create deployment kubernetes-bootcamp --image=gcr.io/google-samples/kubernetes-bootcamp:v1
deployment.apps/kubernetes-bootcamp created
$
```

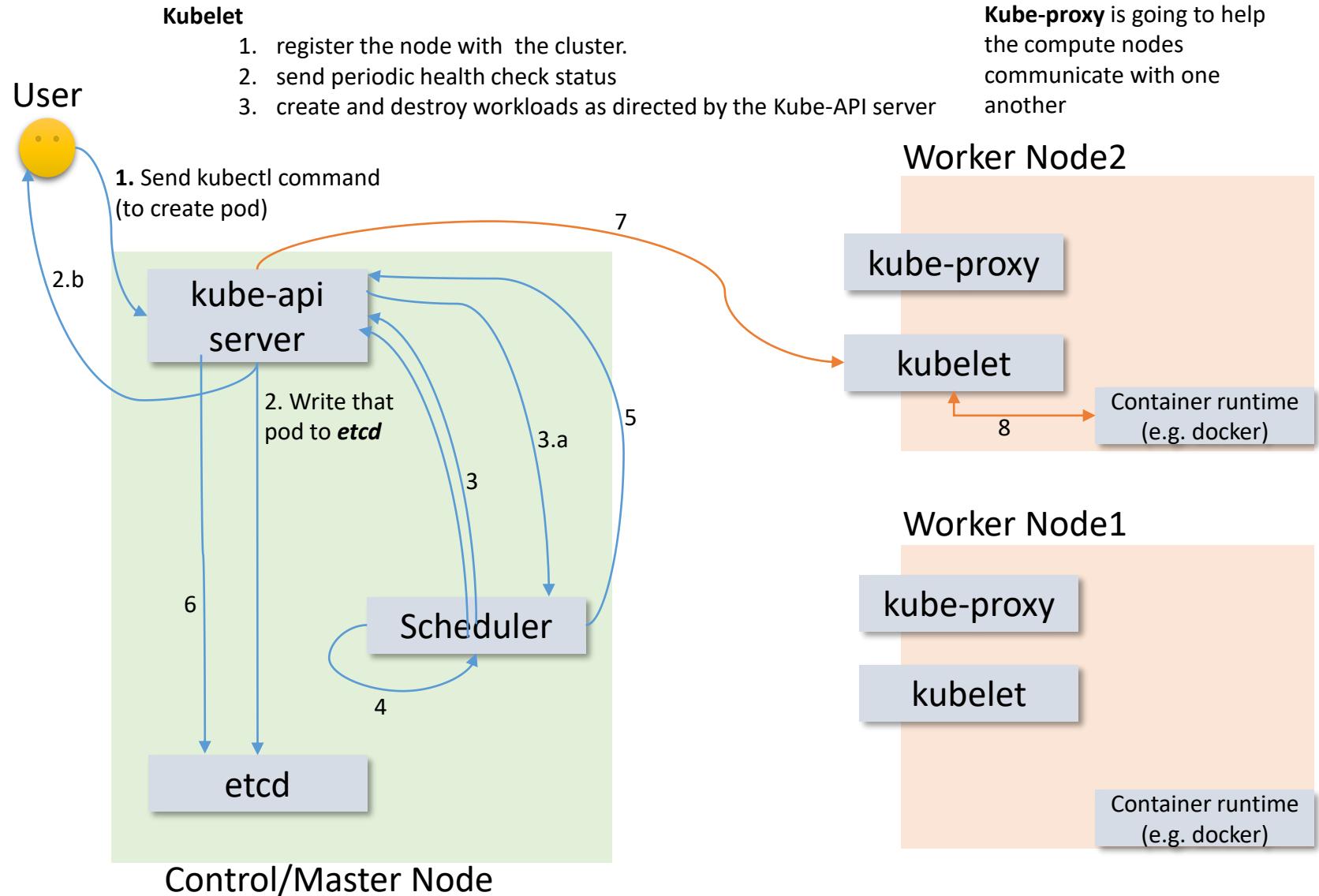
Kubernetes – App, Deployment, Services

- A **Service** in Kubernetes is an abstraction which defines a logical set of Pods and a policy by which to access them.
- **Expose Your App to public**
- When we create a Service, we provide access to pods

Kubernetes – App, Deployment, Services



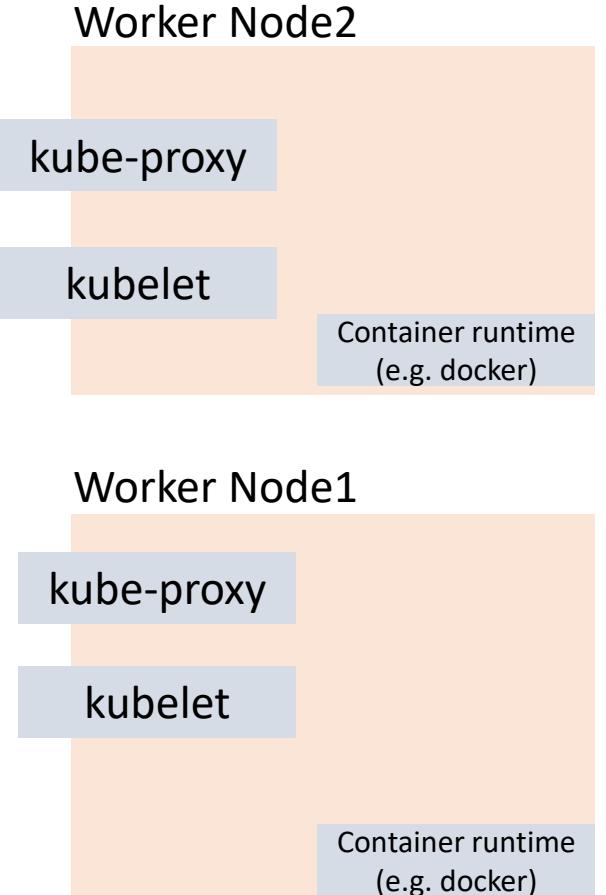
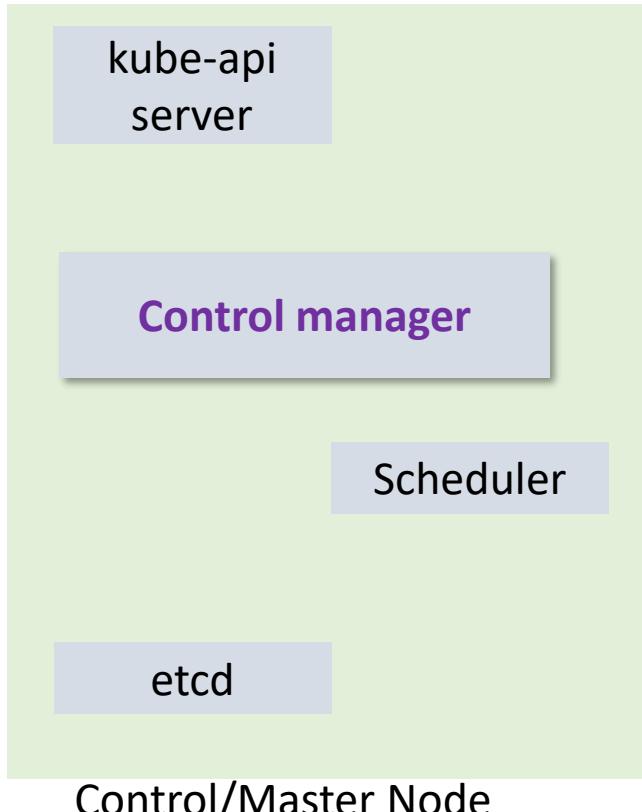
Kubernetes – Pod creation workflow



Kube-proxy is going to help the compute nodes communicate with one another

1. **User** Sends kubectl command (say to create pod)
 - a) **kube-api** server receive the command
 - b) Authorize the user
2. **kube-api** writes that pod request to **etcd**
 - a) Upon successful write..
 - b) kube-api acknowledge the User that the pod is create
 - c) Updated the desired state of whole system/cluster in the **etcd**
3. **Scheduler** pings the **kube-api** server regularly to know if there is any anything to schedule
 - a) Kube-api server send the task to create a pod, in our case.
4. Scheduler find available compute/worker node, based on health of compute node, resource availability, conditions etc.
 1. Find the best worker node (e.g. *Worker Node 2*)
5. Scheduler tells kube-api server where to create the pod
6. Kube-api server write that info (here it is *Worker Node 2*) to **etcd**.
 1. Updated the desired state of whole system/cluster in the **etcd**
7. **Kube-api instruct kubelet (in Worker Node 2) to create a pod.**
8. **Kubelet, with container runtime, create the pod.**

Kubernetes – Pod creation workflow

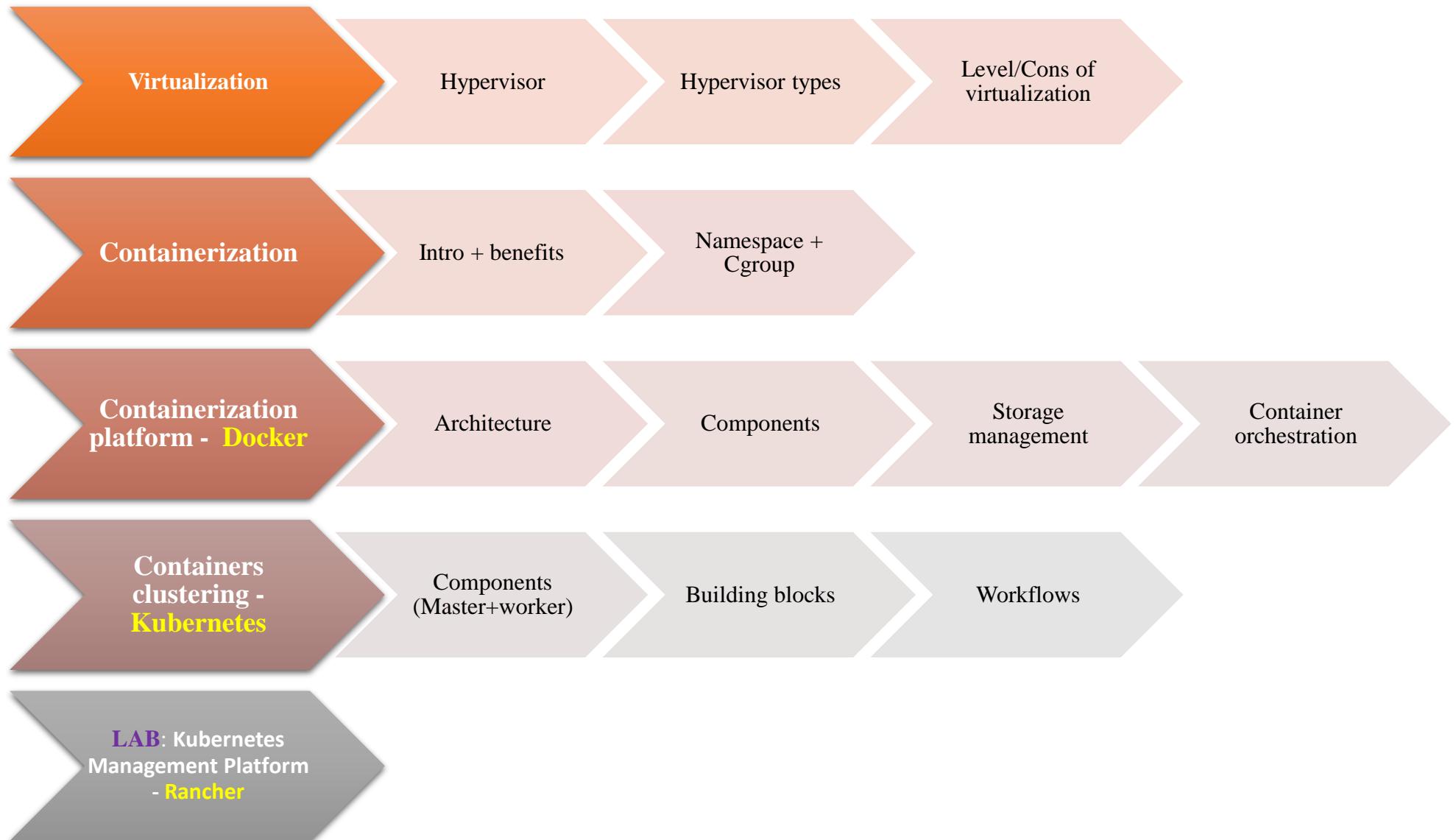


- **Controller Manager:**
 - regulates the Kubernetes cluster
 - tells Nodes what to run, how to expose applications, how-to commit changes and so on
 - *watching* the desired state - the actual state - and making sure it's the same as the desired state

Kubernetes - Advantages

- Easy organization of service with pods
- Largest community among container orchestration tools
- Kubernetes can run on-premises bare metal, OpenStack, public clouds
[Google](#), [Azure](#), [AWS](#), etc.
- Avoid vendor lock issues

Summary

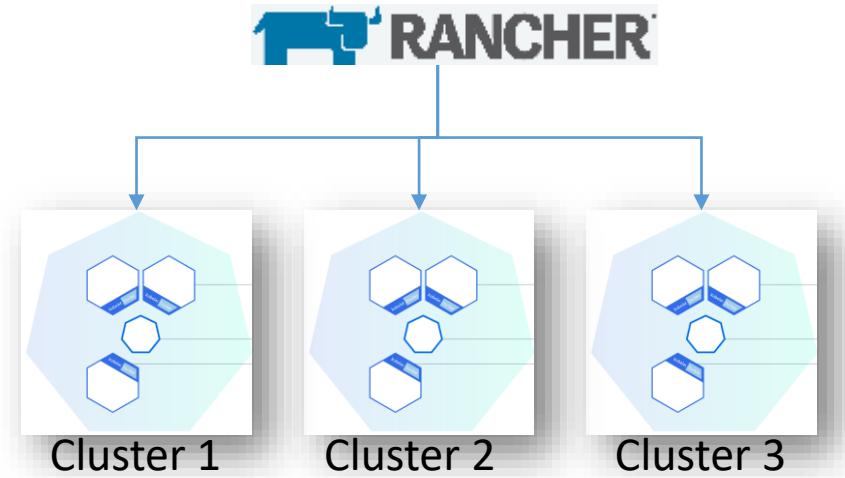


Lab Sessions

- Previous session (Prac-2):
 - Installation of Docker in an ETAIS Virtual Machine
 - Practicing Docker commands, Docker volumes
 - Building a *Dockerfile*
 - Shipping a Docker image to Docker hub
 - Visualize sensor data in Grafana: containerizing an application

• This session

- Setting up ETAIS virtual machines for Kubernetes Cluster
- Installation of Kubernetes cluster using Rancher
- Deploying service using a Docker image
- Working with Rancher dashboard
- ...



References

- [https://www.snia.org/sites/default/files/CSI/SNIA Intro to Containers Container Storage and Docker Final.pdf](https://www.snia.org/sites/default/files/CSI/SNIA%20Intro%20to%20Containers%20Container%20Storage%20and%20Docker%20Final.pdf)
- <https://www.freecodecamp.org/news/a-beginner-friendly-introduction-to-containers-vms-and-docker-79a9e3e119b/>
- <https://www.ibm.com/downloads/cas/VG8KRPRM>
- <https://docs.docker.com/storage/>
- [https://files.informatandm.com/uploads/2018/10/Introduction to Kubernetes Beyond Hello World NeilPeterso.pdf](https://files.informatandm.com/uploads/2018/10/Introduction%20to%20Kubernetes%20Beyond%20Hello%20World%20NeilPeterso.pdf)
- <https://github.com/riigipilv/trainings/tree/master/abc>
- <https://queue.acm.org/detail.cfm?id=2898444>
- <https://www.guru99.com/kubernetes-tutorial.html>
- <https://www.edureka.co/blog/kubernetes-tutorial/>
- [https://files.informatandm.com/uploads/2018/10/Introduction to Kubernetes Beyond Hello World NeilPeterso.pdf](https://files.informatandm.com/uploads/2018/10/Introduction%20to%20Kubernetes%20Beyond%20Hello%20World%20NeilPeterso.pdf)
- <https://ubuntu.com/tutorials/gitlab-cicd-pipelines-with-microk8s#1-overview>
- <https://medium.com/google-cloud/kubernetes-nodeport-vs-loadbalancer-vs-ingress-when-should-i-use-what-922f010849e0>

Any Question ?

THANK YOU

