AGENDA

- Brief History – Infrastructure shifts over the decades
- VMs vs Containers
- What are containers and what problem does it solve?
- What is Docker?
- Deep dive into Docker Internals
- Demo

BRIEF HISTORY – INFRASTRUCTURE SHIFTS OVER THE DECADES

Let's Recap

MAJOR INFRASTRUCTURE SHIFTS

Mainframe to PC

90's

Baremetal to Virtual

00's

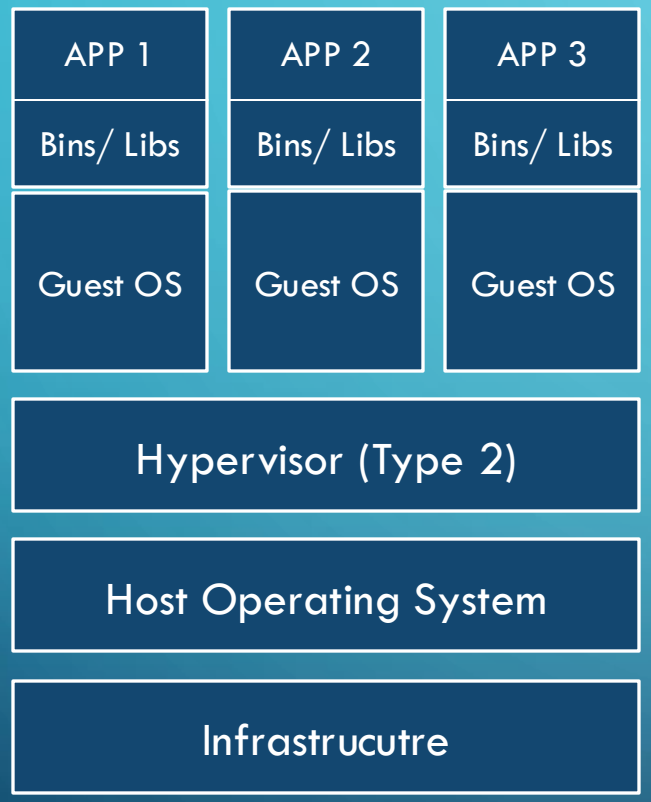
Datacenter to Cloud

10's

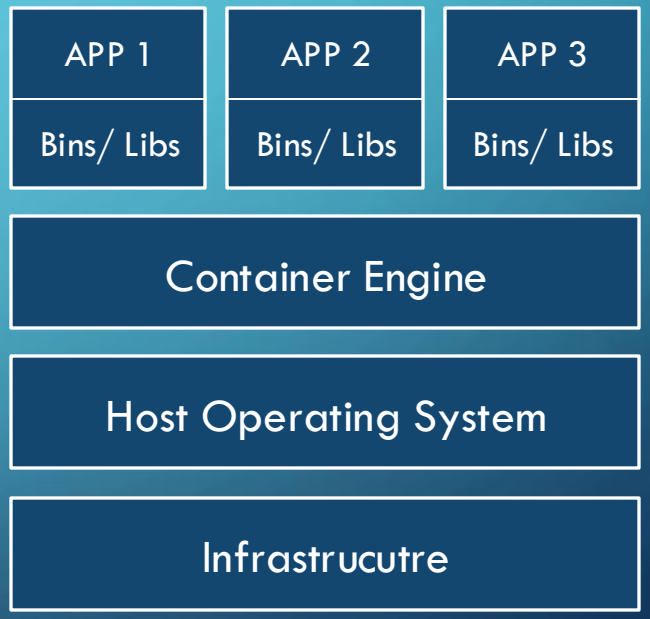
Host to Container (Serverless)

Now

VMs vs CONTAINERS



-Virtual Machines-



-Containers-

WHAT ARE CONTAINERS AND WHAT PROBLEMS DOES IT SOLVE?

MATRIX FROM HELL INCREASES THE COMPLEXITY

	Desktop	Test/QA Cluster	Production Cluster	Public Cloud	Data Center	Mainframe	Windows Server	Edge Device
Static Website	?	?	?	?	?	?	?	?
Web Frontend	?	?	?	?	?	?	?	?
Background Workers	?	?	?	?	?	?	?	?
User DB	?	?	?	?	?	?	?	?
Analytics DB	?	?	?	?	?	?	?	?
Queue	?	?	?	?	?	?	?	?

CONTAINERS REDUCES THE COMPLEXITY

IN SUMMARY A CONTAINER IS,

- Just an isolated process running on the host machine. And a restricted process.
- Will share OS and, where appropriate, bins/ libraries and limited to what resources it can access.
- It exits when the process stops.

“Containers are the next once-in-a-decade shift in IT infrastructure and process”

WHAT IS DOCKER?

- So what's Docker? – In 2024, Docker means lot's of things, let's just clear things out.
  - Docker as a "Company"
  - Docker as a "Product"
  - Docker as a "Platform"
  - Docker as a "CLI tool"
  - Docker as a "Computer Program"

WHAT IS DOCKER?

- Docker provides the ability to package and run applications within a loosely isolated environment which is a a container. Simply it's a container engine (runtime + tool for managing containers and images).
- It provides tooling and a platform to manage lifecycle of your containers,
  - Develop your apps and supporting components using containers
  - Distribute and test your apps as a container
  - Ability to deploy your app as a container or an orchestrated service, in whatever environment which supports Docker installation
- It shares the same OS kernel
- It works on all major Linux Distributions and containers native to Windows Server (specific versions)

UNDERLYING TECHNOLOGY IN DOCKER

- Docker is an extension of LXC's (Linux Containers) capabilities and packaged it in a way which is more developer friendly.
- It was developed in Go language and utilizes LXC, namespaces, cgroups and the linux kernel itself. Docker uses namespaces to provide the isolated workspace called a container. Each aspect of a container runs in a separate namespace and its access is limited to this namespace.

Static Website

Web Frontend

Background Workers

User DB

Analytics DB

Queue

Desktop

Test/QA Cluster

Production Cluster

Public Cloud

Data Center

Mainframe

Windows Server

Edge Device

BASIC DOCKER COMMANDS

- Docker CLI structure,
  - Old (Still works as expected) docker <command> options
  - New – docker <command> <sub-command> (options)
- Pulling Docker Image
  - docker pull nginx
- Running a Docker Container
  - docker run –p 80:80 --name web-server nginx
- Stopping the Container
  - docker stop web-server (or container id)

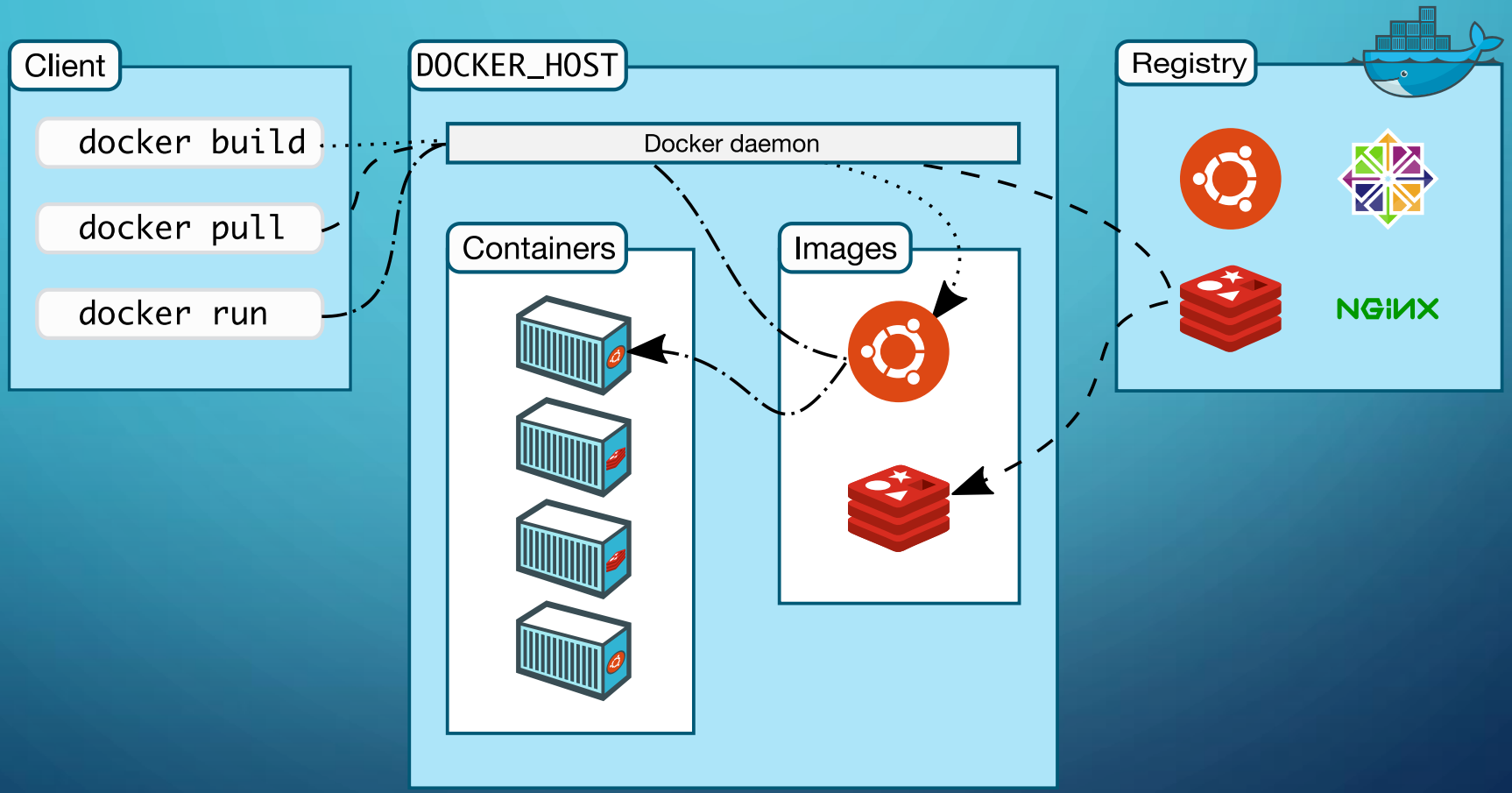
- Check what's happening in a containers,
  - docker container top web-server – Process list in 1 container
  - docker container inspect web-server – Details of one container config
  - docker container stats – Performance stats for all containers
- Getting a shell inside containers,
  - docker container run –it – Start a new container interactively
  - docker container exec –it <container_id_or_name> echo "I'm inside the container" – Run additional commands in the container
- Listing, removing containers and images
  - docker images
  - docker container ls | docker ps
  - docker <object> rm <id_or_name>

ARE THERE SOLUTIONS OTHER THAN DOCKER?

- Docker – Container runtime + Tool for managing containers and images
- Containerd – Container runtime only
- Podman – Tool for managing containers and images

DEEP DIVE INTO DOCKER INTERNALS

DOCKER ARCHITECTURE



WHAT HAPPENS WHEN YOU RUN A CONTAINER?

- ***docker run –p 80:80 nginx | docker container run –p 80:80 nginx***
  1. Looks for that particular image locally in image cache, if its not found pulls it from the configured registry (image repository). Downloads the latest version by default (nginx:latest)
  2. Creates a new container based on that image and prepares to start
  3. Docker allocates read write filesystem to the container, as its final layer. This allows running container to modify files and directories in its local filesystem.
  4. Gives it a virtual IP on a private network inside docker engine
  5. Opens up port 80 on host and forwards to port 80 in container.
  6. Starts container by using the CMD in the image Dockerfile.

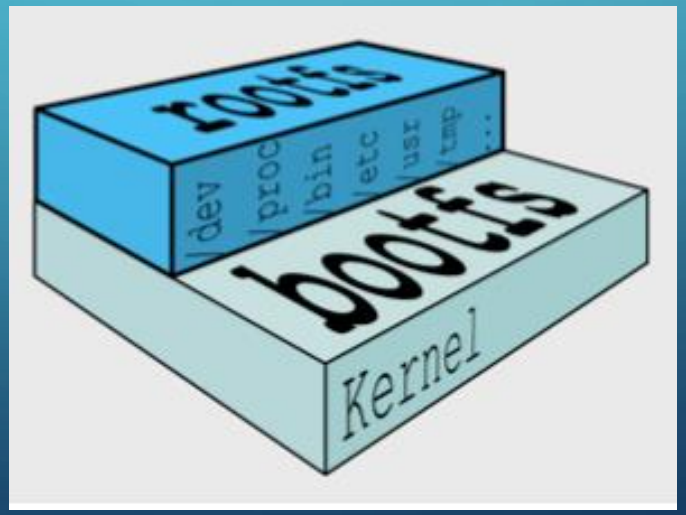
DOCKER OBJECTS

- Docker Images
  - A read-only template with instructions/ metadata for creating a Docker container.
  - Can create your own image or use images created and published in a registry by others.
  - Dockerfile can be used to define steps required to create and run the image.
  - Each instruction in Dockerfile creates a layer in the image, only those layers which changes each time are rebuilt – What makes images so lightweight, small and fast.

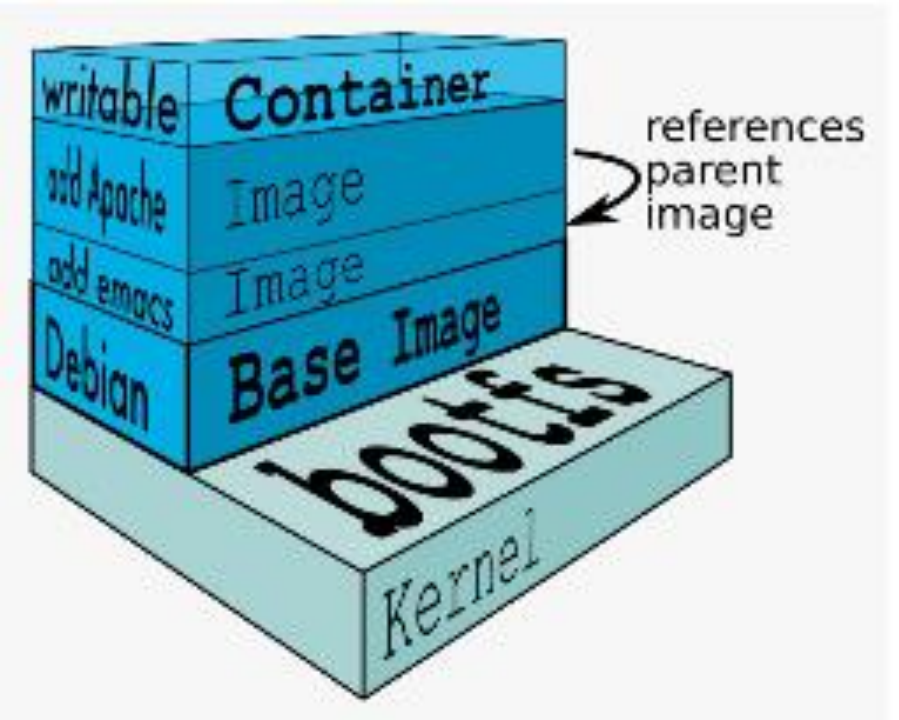
- Docker Containers
  - Runnable instance of an image.
  - Can create, start, stop, move, or delete a container using the Docker API or CLI.
  - Can connect it to one or more networks, attach storage to it, or even create a new image based on its current status.
  - A container is defined by its image as well as any config options provided to it when you create or start it. Note that when the container is removed any data associated with it will be deleted unless those are not stored in a persistent storage.

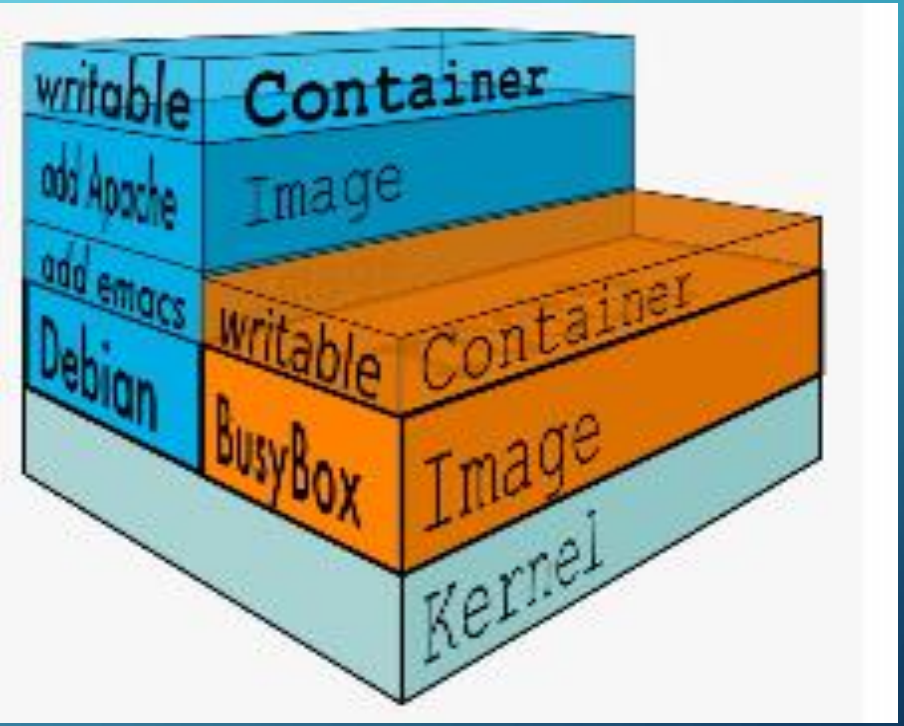
UNDERSTANDING DOCKER IMAGES/ CONTAINERS INTERNALS

- Docker Filesystem
  - Boot file system (bootfs) – Contains the bootloader and the kernel. User never touches this.
  - Root file system (rootfs) – Includes the typical directory structure we associate with Unix-like OS.



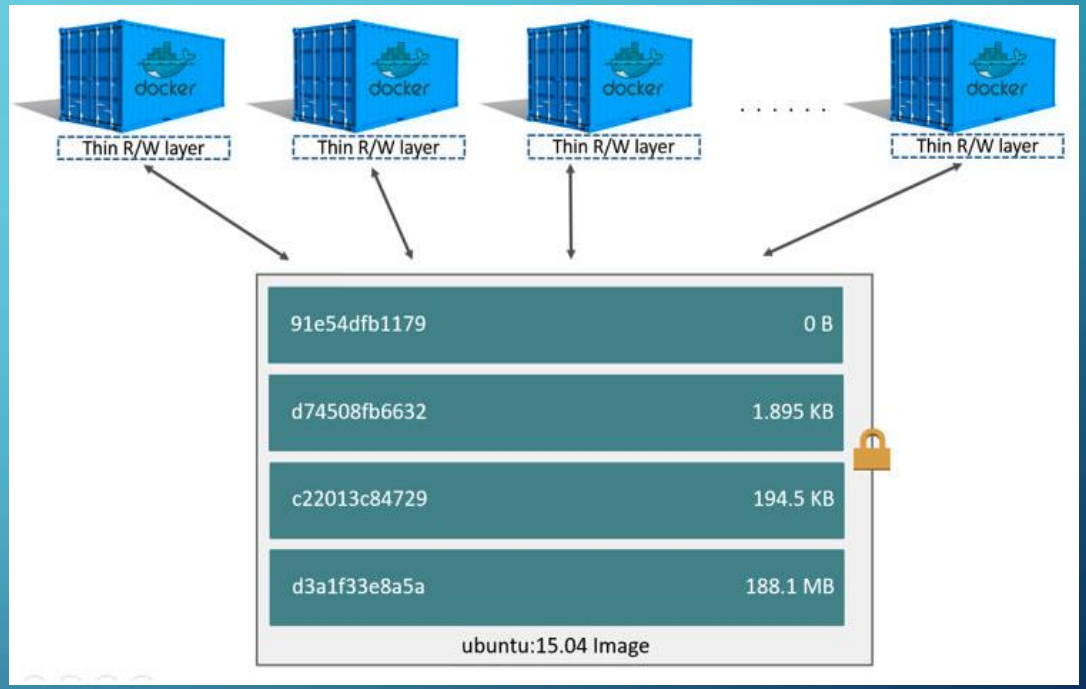
- In traditional Linux boot, kernel first mounts the rootfs as read-only, checks its integrity, and then switches the rootfs volume to read-write mode.
- Docker mounts the rootfs and **instead of changing the file system to read-write mode, it then takes advantage of union mounts service to add a read-write filesystem over the read-only file system.**
- In Docker terminology, a read-only layer is called an image. An image never changes and is fixed.
- Each image depend on one more image which creates the layer beneath it. The lower image is the parent of the upper image. Image without a parent is a base image.
- When you run a container, Docker fetches the image and its Parent Image, and repeats the process until it reaches the Base Image. Then the Union File System adds a read-write layer on top.
- That read-write layer, plus the information about its Parent Image and some additional information like its unique id, networking configuration, and resource limits is called a **container**


writable
Container
add Apache
Image
add emacs
Image
Debian
Base Image
boots
Kernel
references
parent
image


writable
Container
add Apache
Image
add emacs
writable
Container
Debian
BusyBox
Image
Kernel

- A container can have two states, it may be running or exited.
- When a container is exited the state of the file system and its exit value is saved.
- You can start, stop, and restart a container. The processes of restarting a container from scratch will preserve its file system is just as it was when the container was stopped. But the memory state of the container is not preserved.
- You can also remove the container permanently.
- A container can also be promoted directly into an image using the docker commit command. Once a container is committed as an image, you can use it to create other images on top of it.
  - docker commit <container-id> <image-name:tag>

- Based from the UFS, Docker uses a strategy called Copy on Write to improve the efficiency by minimizing I/O and the size of each subsequent layers,
  - If a file or directory exists in a lower layer within the image, and another layer (including the writable layer) needs read access to it, it just uses the existing file.
  - The first time another layer needs to modify the file (when building the image or running the container), the file is copied into that layer and modified.

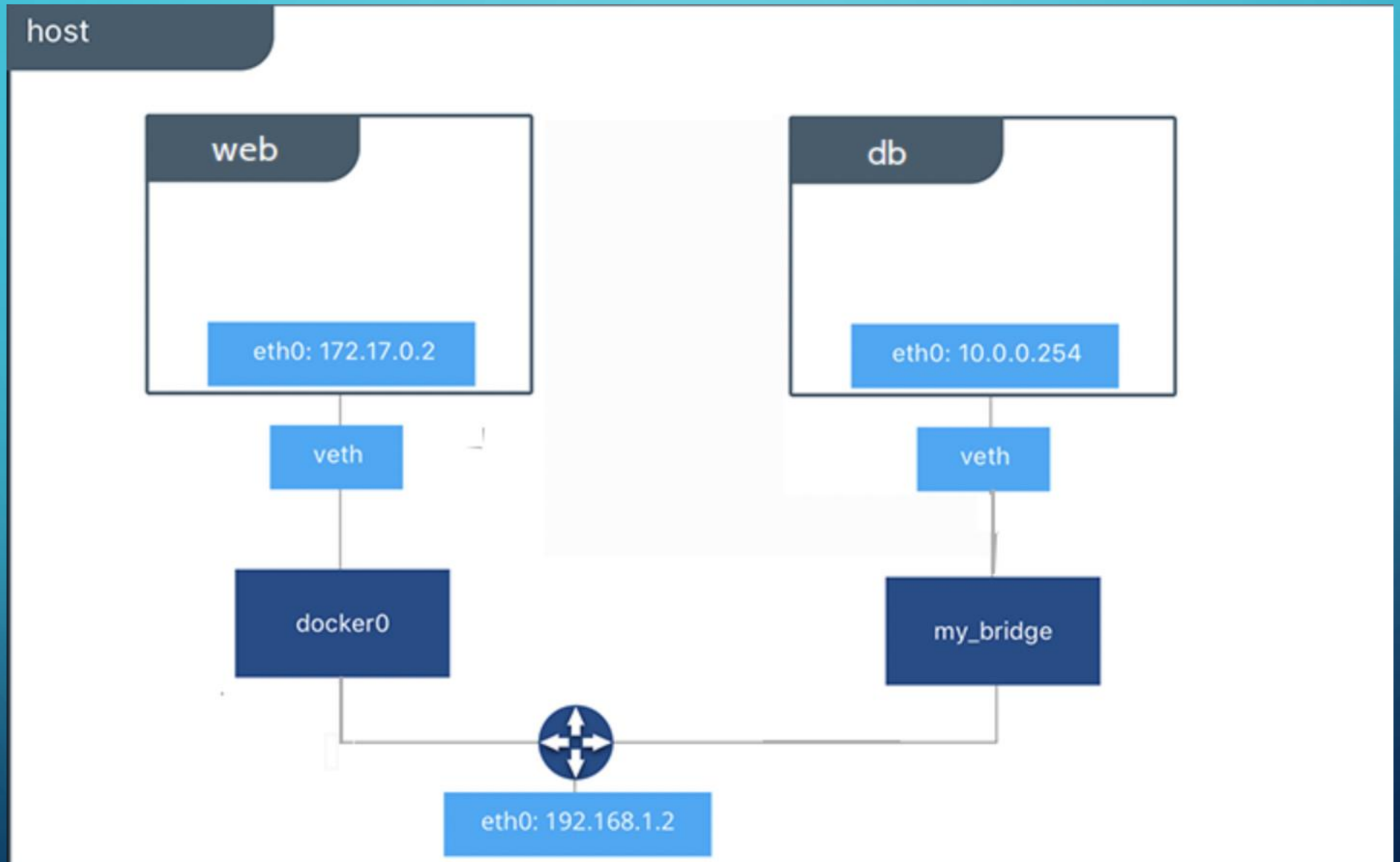


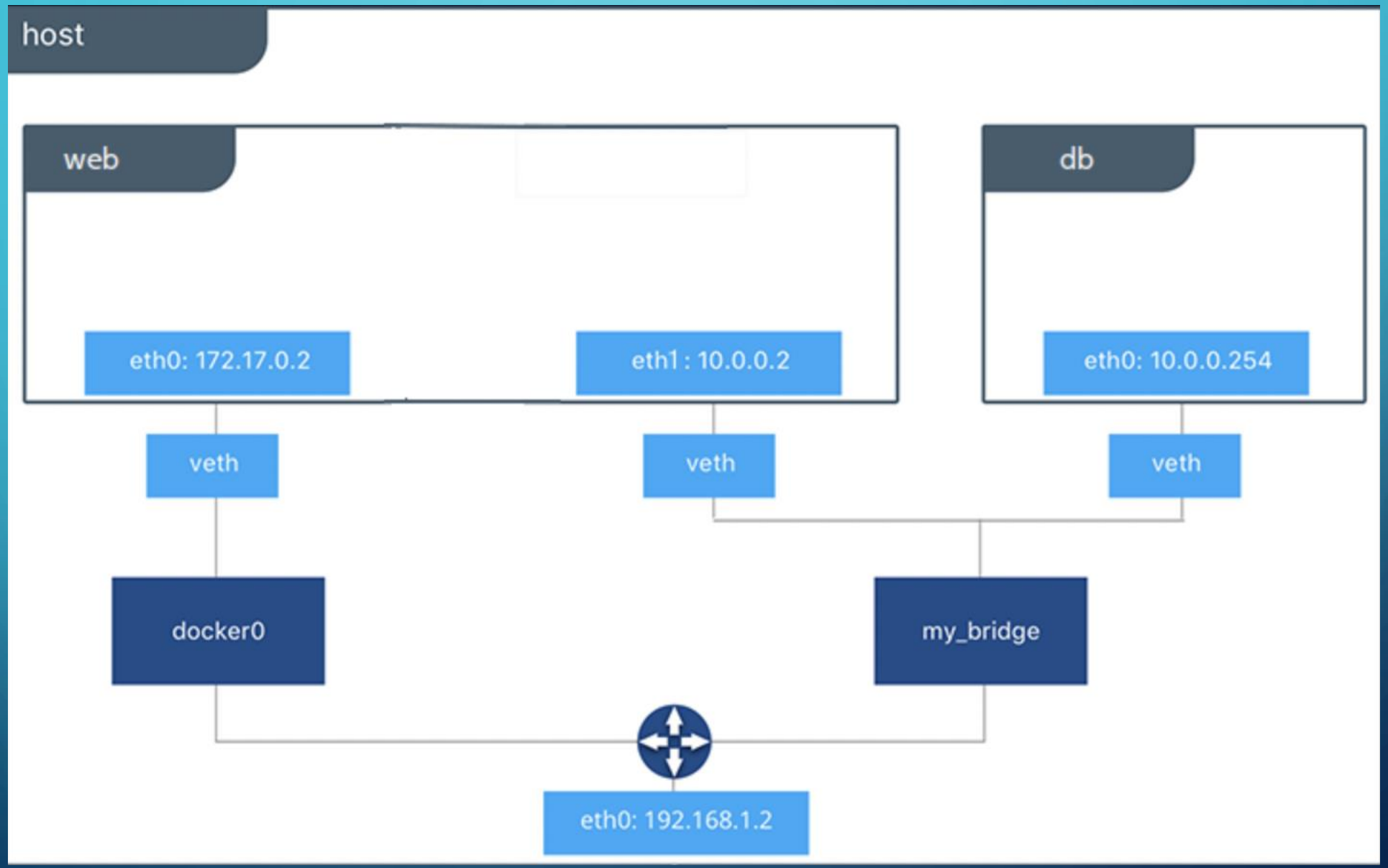
- Docker Image Creation and Storage
  - You can create an image using a Dockerfile or by committing a container's changes back to an image.
  - Once you create an image, it will be stored in the Docker host's local image cache.
  - In order to move images in/out of the local image cache,
    - Export/ Import it as a tarball
    - Push/ pull to a remote image registry (ex - DockerHub)

DOCKER OBJECTS CONT…

- Docker Networks
  - Each container is connected to a private virtual network called “bridge”.
  - Each virtual network routes through the NAT firewall on the host IP.
  - All containers on a virtual network can talk to each other without exposing ports.
  - Best practice is to create a new virtual network for each app.

- Docker enables to:
  - Create new virtual networks.
  - Attach container to more than one virtual network (or none)
  - Skip virtual networks and use host IP (--net=host)
  - Use different Docker network drivers to gain new abilities.
    - Docker Engine provides support for different network drivers – bridge (default), overlay and macvian etc.. . You can even write your own network driver plugin to create your own one.
- Docker Networking – DNS
  - Docker deamon has a built in DNS, which consider container name as equivalent hostname of the container.


host
web
eth0: 172.17.0.2
veth
docker0
db
eth0: 10.0.0.254
veth
my_bridge
eth0: 192.168.1.2


host
web
db
eth0: 172.17.0.2
eth1 : 10.0.0.2
eth0: 10.0.0.254
veth
veth
veth
docker0
my_bridge
eth0: 192.168.1.2

PERSISTENCE DATA

- If we want to use persistence data as in like databases or unique data in containers, Docker enables that using two ways,
  - Volumes – Make a location outside of container UFS.
  - Bind Mounts - Link host path to the container path.

DOCKER COMPOSE

- Another Docker client, that lets you work with apps consisting of a set of containers.
  - This saves docker container run settings in easy to read file, which can be committed to VCS.
  - Can use this to create one-line development environments
- Consists of two components
  - YAML formatted file that describes – Images, Containers, Networks, Volumes etc…
  - A CLI tool docker-compose used to automate/manage those YAML files

HOW CAN WE RUN CONTAINERS AT SCALE?

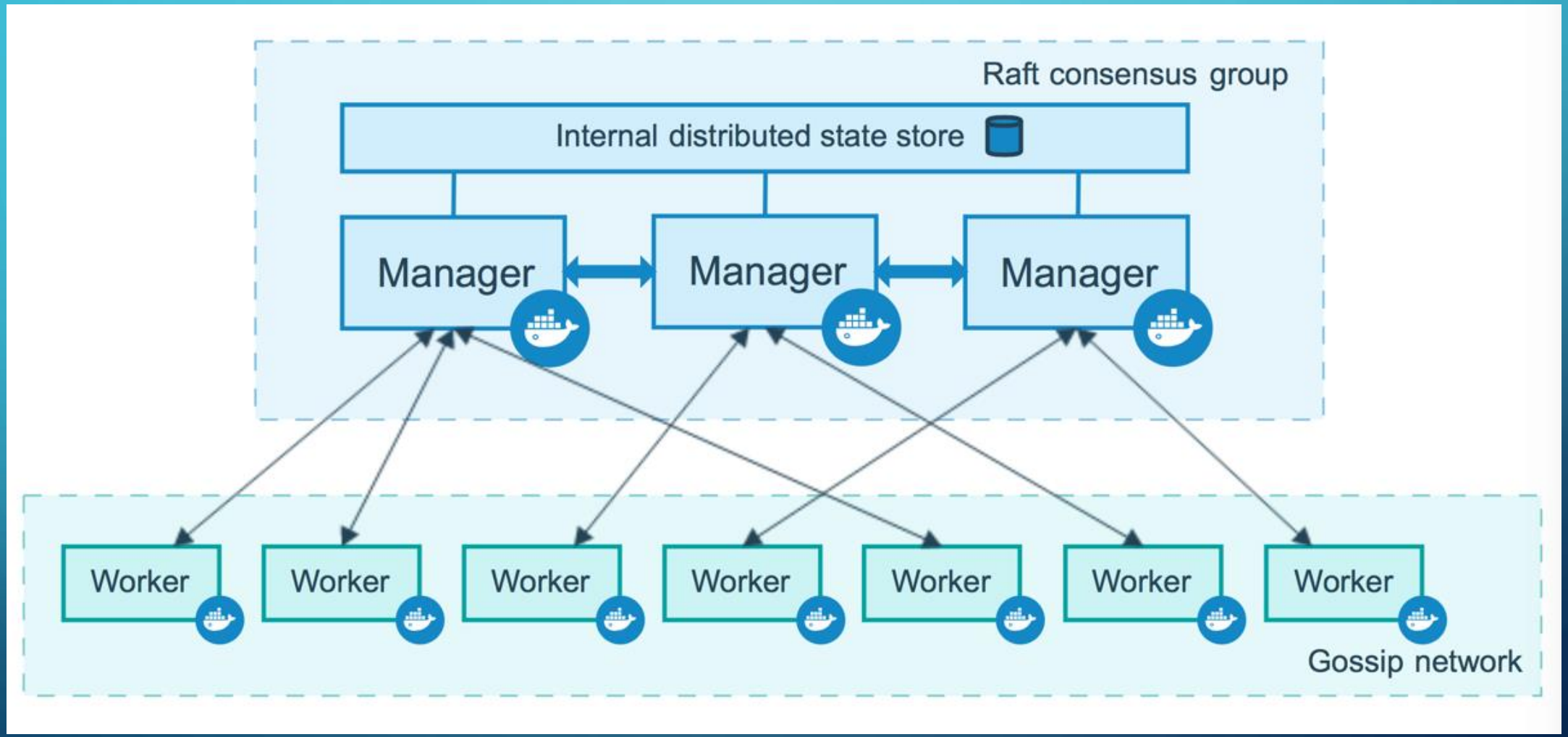
CONTAINER ORCHESTRATION

- Container orchestration automates the deployment, management, scaling, and networking of containers.
- Container orchestration can be used in any environment where you use containers. It can help you to deploy the same application across different environments without needing to redesign it.

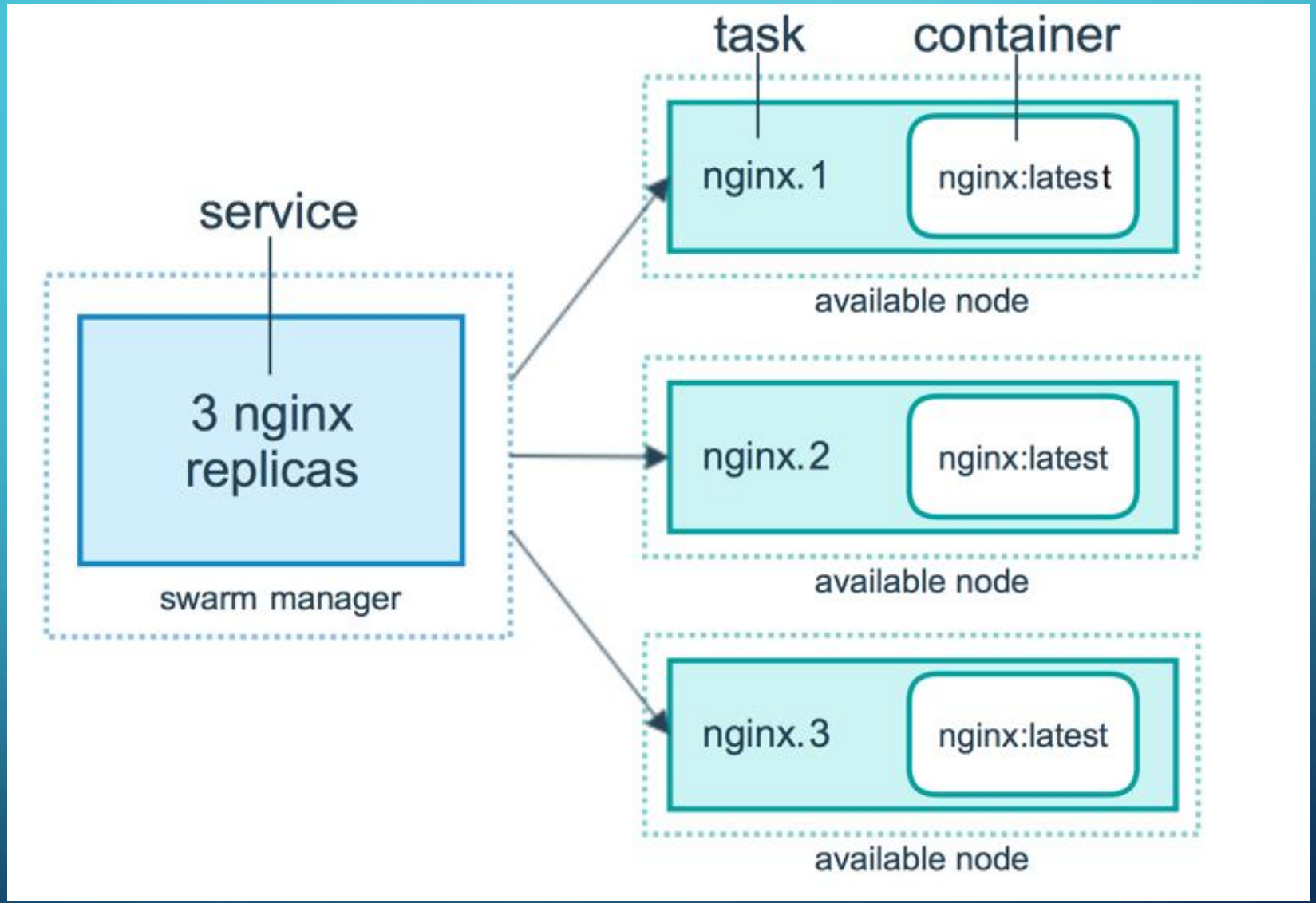
INTRODUCTION TO DOCKER SWARM

DOCKER SWARM KEY CONCEPTS

- Docker Swarm provides the cluster management and orchestration features of Docker Engine 1.12
- Nodes - A **node** is an instance of the Docker engine participating in the swarm. You can also think of this as a Docker node.
  - Manager Node - To deploy your application to a swarm, you submit a service definition to a **manager node**. These nodes are also responsible for perform the orchestration and cluster management functions required to maintain the desired state of the swarm.
  - Worker Nodes - Receive and execute tasks dispatched from manager nodes. An agent which is running within the worker nodes report the current status of the tasks assigned to it which allows manager node to keep the desired state for each worker node.


Raft consensus group
Internal distributed state store
Manager
Manager
Manager
Worker
Worker
Worker
Worker
Worker
Worker
Worker
Gossip network

- Task - It is the atomic scheduling unit of swarm. Manager nodes assign tasks to worker nodes according to the number of replicas set in the service scale.
- Service - the definition of the tasks to execute on the manager or worker nodes. Here is where you specify which container image to use and which commands to execute inside running containers.
  - Replicated services model - the swarm manager distributes a specific number of replica tasks among the nodes based upon the scale you set in the desired state.
  - Global services model - the swarm runs one task for the service on every available node in the cluster.


task
container
service
nginx.1
nginx:latest
available node
3 nginx
replicas
nginx.2
nginx:latest
available node
swarm manager
nginx.3
nginx:latest
available node

- *docker swarm init --advertise-addr <MANAGER-IP>*
- Join a worker node - *docker swarm join --token SWMTKN-1 49nj1cmql0jkz5s954yi3oex3nedyz0fb0xx14ie39trti4wxv-8vxv8rssmk743ojnwacrr2e7c 192.168.99.100:2377*
- *docker service create --replicas 1 --name helloworld alpine ping docker.com*
- *docker service scale <SERVICE-ID>=<NUMBER-OF-TASKS>*

INTRODUCTION TO KUBERNETES

WHAT IS KUBERNETES?

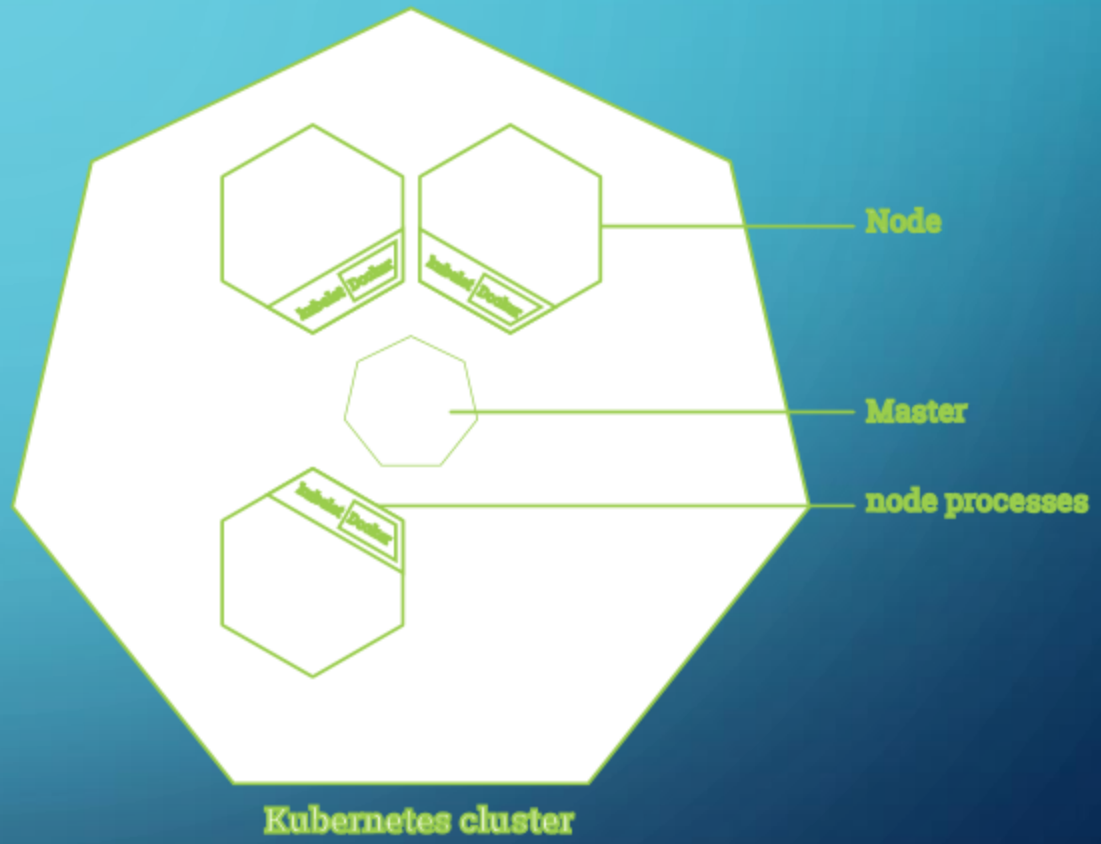
- “Kubernetes (k8s) is an open source platform for automating deployment, scaling and management of containers at scale”
- Project that was created by Google as an open source container orchestration platform. Born from the lessons learned and experiences in running projects like Borg and Omega @ Google
- It was donated to CNCF (Cloud Native Computing Foundation) who now manages the Kubernetes project
- Current Kubernetes stable version – 1.29

IT'S CAPABLE OF…

- Horizontal scaling
- Load distribution
- Service discovery
- Health monitoring
- Deploying new versions, rollbacks
- Handling hardware faliures

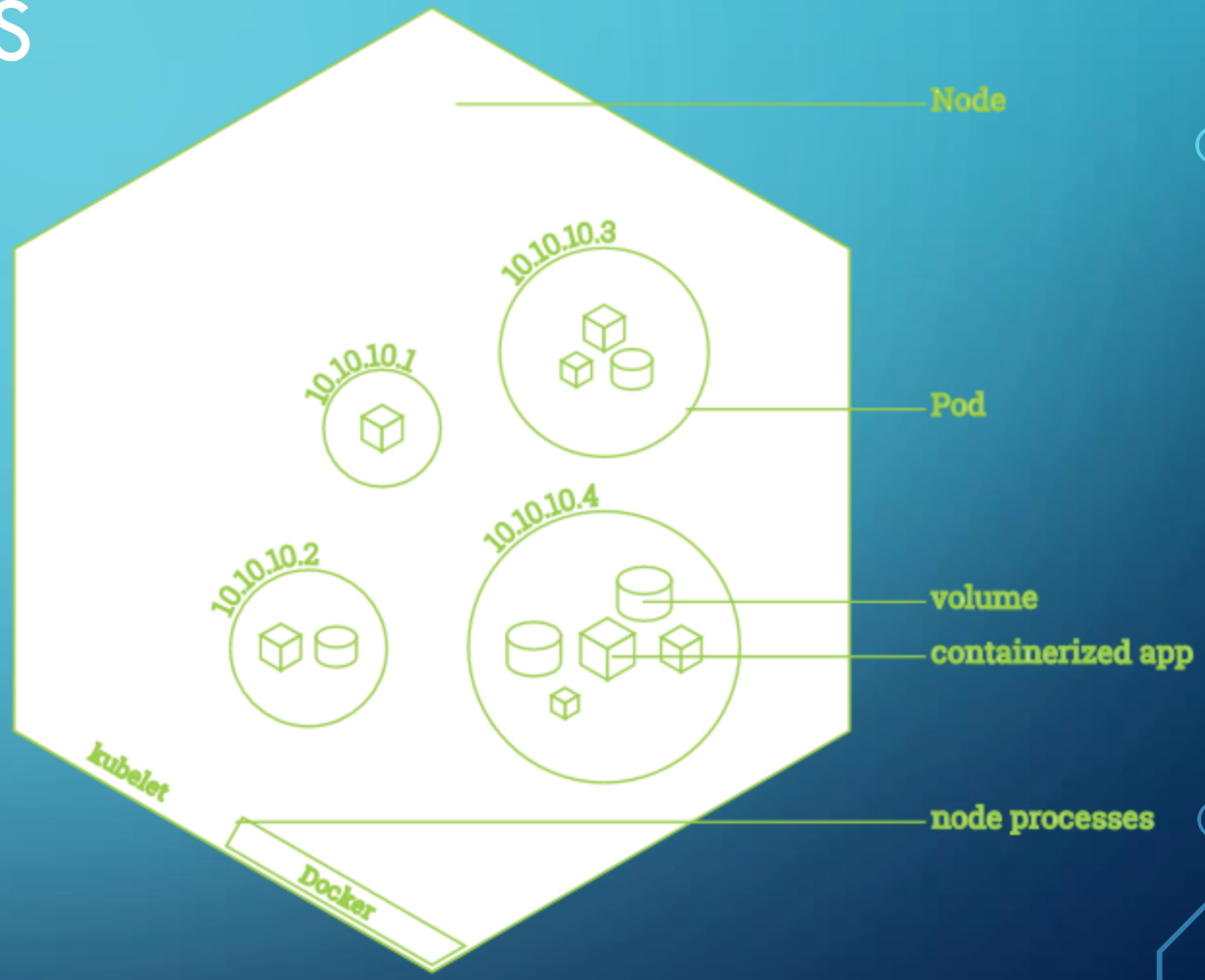
HIGH LEVEL ARCHITECTURE

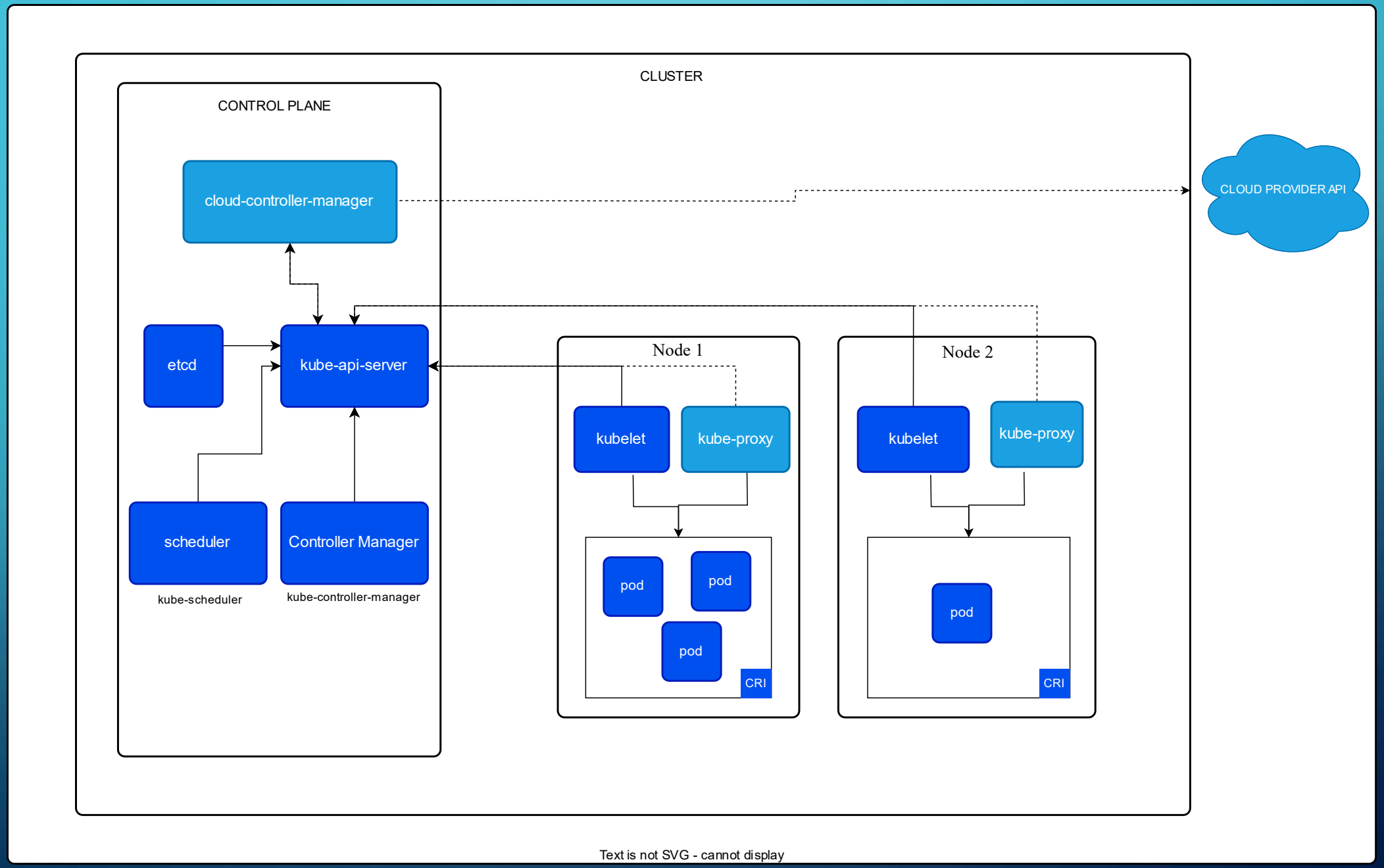
- Master [Control Plane]
  - coordinates all activities in the cluster
- Nodes:
  - virtual or physical machines
  - actual workers
  - runs processes:
    - kubelet
    - kube-proxy
    - container runtime



BASIC BUILDING BLOCKS

- Containers
  - Define single running process*
  - Eg docker container
- Pods
  - the way of running containers in kubernetes
  - basic deployable and scaling unit
  - defines one or more containers
  - containers are co-located on a node
  - flat network structure
- Nodes:
  - physical worker machines
  - can run multiple pods
  - pods running within single node don't know about each other




CLUSTER
CONTROL PLANE
cloud-controller-manager
CLOUD PROVIDER API
etcd
kube-api-server
Node 1
Node 2
kubelet
kube-proxy
kubelet
kube-proxy
scheduler
Controller Manager
kube-scheduler
kube-controller-manager
pod
pod
pod
pod
CRI
CRI
Text is not SVG - cannot display

RUNNING THINGS LOCALLY

- Minikube:
  - single node cluster
  - running in a VM
  - supports linux, windows and macOS
  - mature project
- kind:
  - Requires you to have Docker or Podman installed in your local computer
- Docker Desktop with built-in kubernetes:
  - single node cluster
  - running in a VM
  - windows and macOS
  - drag & drop installation
  - bound to specific kubernetes version

https://kubernetes.io/docs/tasks/tools/

MANAGING CLUSTER RESOURCES

- **Create resource from file** - *kubectl create -f resource_file.yml*
- **Change existing (or create) resource based on file** - *kubectl apply -f resource_file.yml*
- **Delete existing resource** - *kubectl delete resource_type resource_name*
- **List resources of type** - *kubectl get resource_type*
- **Edit resource on the server** - *kubectl edit resource_type resource_name*

DEBUGGING CLUSTER RESOURCES

- **Execute command on the container -** *kubectl exec [-it] pod_name process_to_run*
- **Get container logs -** *kubectl logs pod_name [-c container_name]*
- **Forward port from a pod -** *kubectl port-forward pod_name local_port:remote_port*
- **Print detailed description of a resource -** *kubectl describe resource_type resource_name*

FEW RESOURCE OBJECTS IN K8S

- **Replica Sets -** Ensures desired number of pods exist by: scaling up or down and running new pods when nodes fail
- ***Deployment***
  - A *Deployment* provides declarative updates for Pods and ReplicaSets. You describe a *desired state* in a Deployment, and the Deployment Controller changes the actual state to the desired state at a controlled rate.
- ***Service***
  - A method for exposing a network application that is running as one or more Pods in your cluster.
    - Cluster IP/ NodePort/ Load Balancer/ Ingress

Feature	ClusterIP	NodePort	LoadBalancer
Exposition	Exposes the Service on an internal IP in the cluster.	Exposing services to external clients	Exposing services to external clients
Cluster	This type makes the Service only reachable from within the cluster	A NodePort service, each cluster node opens a port on the node itself (hence the name) and redirects traffic received on that port to the underlying service.	A LoadBalancer service accessible through a dedicated load balancer, provisioned from the cloud infrastructure Kubernetes is running on
Accessibility	It is **default** service and Internal clients send requests to a stable internal IP address.	The service is accessible at the internal cluster IP-port, and also through a dedicated port on all nodes.	Clients connect to the service through the load balancer's IP.
Yaml Config	`type: ClusterIP`	`type: NodePort`	`type: LoadBalancer`
Port Range	Any public ip form Cluster	30000 - 32767	Any public ip form Cluster

Image Credit: StackOverflow

DEMO

- *Defining a Pod*
- *Creating a ReplicaSet*
- *Creating a Deployment*
- *Creating a Service and exposing it*

Q/A

REFERENCES

- https://docs.docker.com/get-started/overview/
- https://www.docker.com/blog/containers-and-vms-together/
- https://www.redhat.com/en/topics/containers/containers-vs-vms
- Docker Storage Drivers - https://docs.docker.com/storage/storagedriver/
- https://docs.docker.com/storage/storagedriver/select-storage-driver/
- https://www.youtube.com/watch?v=cjXI-yxqGTI
- Docker Buildx - https://docs.docker.com/buildx/working-with-buildx/
- Jiang Huan BuildKit timings - https://medium.com/titansoft-engineering/docker-build-cache-sharing-on-multi-hosts-with-buildkit-and-buildx-eb8f7005918e
- What is Docker BuildKit - https://brianchristner.io/what-is-docker-buildkit/

THANK YOU!

LinkedIn - https://lk.linkedin.com/in/ravindufernando