Container Virtualization

Diogo Gomes

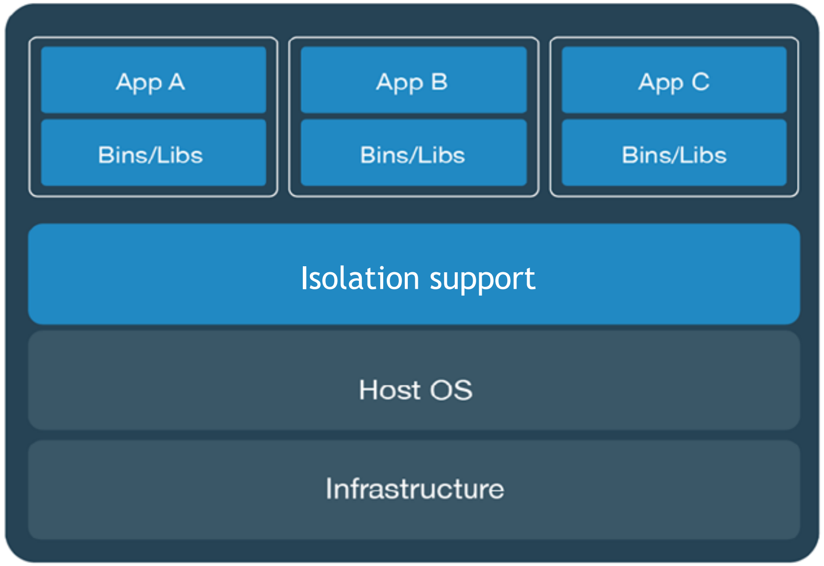
OS Level Virtual Machines

- Do not provide a full virtualized hardware
  - Only OS services are virtualized
  - Host kernel: virtualize its services in a isolated manner to different guests
- Container: isolated execution environment to encapsulate one or more processes

- Relevant aspects:
  - Isolation/Visibility: limit what can be seen by tenants
  - Resource control: limit resource consumption
  - Portability: reconstruct the same environment in multiple hosts

Container based virtualization

- Container based virtualization relies on OS mechanisms for enhanced isolation



OS Support

- Linux has no support for containers!
  - That is: the Linux OS does not know what is a container

- Containers are a set of configurations, creating an environment, applied to a process (and its children)
  - Namespaces: isolation and virtualization of each tenant
  - Cgroups: resource control of CPU, RAM, IO of each tenant

Underlying technologies

- Namespaces
  - (mnt, pid, net, ipc, uts/hostname, user ids)
- cgroups
  - (cpu, memory, disk, i/o - resource management)
- AppArmor, SELinux
  - (security/access control)
- seccomp
  - (computation isolation)
- chroot
  - (file system isolation)

Namespaces

- Mechanism used to isolate and virtualize system resources
  - Processes in a namespace are unable to see remaining resources
    - Their namespace looks like the entire host
  - Processes in a namespace access resources as if there is no concurrency
- Network namespace:
  - Mechanism to create a somewhat independent set of networking resources
    - Network interfaces, routing tables
  - Network interfaces can only belong to a single namespace
- PID namespace:
  - Restricted namespace with private process identifiers
    - Host processes are invisible

cgroups

- Mechanism used to restrict (limit, control) or monitor the amount of resources used by “groups of processes”
  - Processes can be organized in groups, to control their accesses to resources
- Example: CPU control groups for scheduling
  - Limit the amount of CPU time that processes can use, etc...
- Similar cgroups for other resources
  - memory, IO, pids, network, ..

Setting up a container

1. Setup all the needed namespaces and control groups
   - Usually according to some template
2. Create a “disk image” for the container
   - directory containing the container’s filesystem
   - will exist in the host (it’s a directory)
3. Chroot to the container filesystem
   - Must contain all the libraries/files needed to execute the program
4. Start the program to containerize
   - This process it will have PID 1 in the container
   - Note: this process can mount procfs or other pseudo-filesystems
     - Namespaces allow to control the information exported in those pseudofilesystems

Setting up a container

- Due to network namespace, containerized processes do not see the host’s network interfaces
  - But we usually do not assign a real network interface to a namespace
    - Would have impact to the host

- Networking with containers:
  - Create a virtual ethernet pair: Two virtual interfaces, connected point-to-point
    - Packets sent on one interface are received on the other, and vice-versa
  - Associate one virtual interfaces to the network namespace of the container
  - Bind the other one to a software bridge

Containers are not Hardware VMs

- Core idea: a container is a **Process in a sandbox**, not a fully emulated virtual hardware

Virtual Machine	Container
Each VM runs a different OS	All containers share the same OS
Booting the VM involves the Full boot of the guest HW	Containers boot in seconds (start service)
VM snapshots are made on demand	Containers use multiple images in layers
VM description is a XML document	Container description is a sequence of instructions
VM will pre-consume RAM and CPU	Containers have almost zero overhead
A VM description will result in one VM	A dockerfile may be used to start many containers
Has limits of tens of VMs per CPU core	Can have hundreds of containers per CPU core

Tools

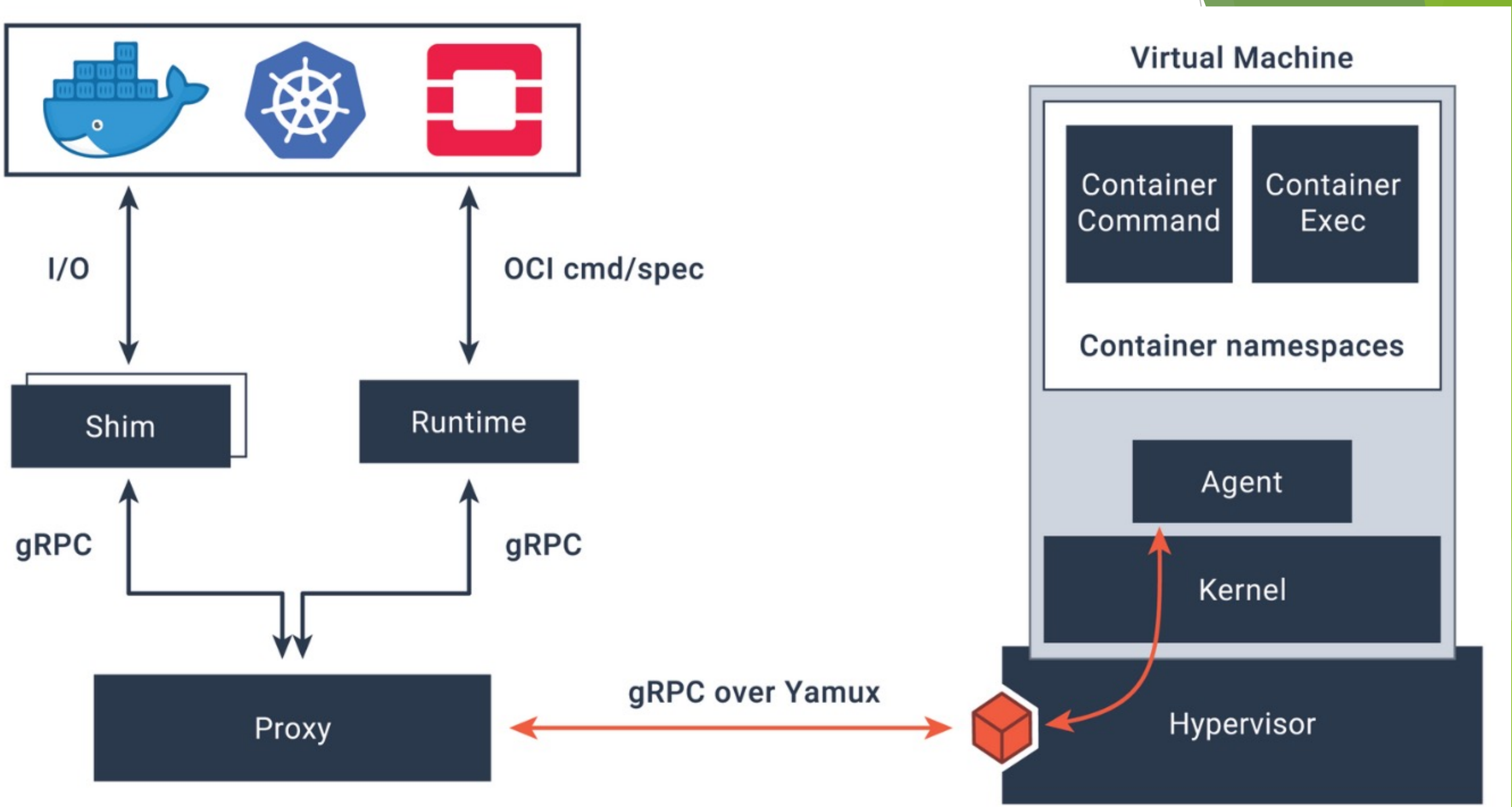
- Creating containers by hand is a difficult tasks
  - Reasonably high set of instructions to create namespaces, cgroups, chroots, interfaces, bridges
- User space tools facilitate manipulation of containers
  - LXC: Linux Containers
  - Docker
  - Singularity
  - Kubernetes

Portability

- Containers are all about creating replicable execution environments
  - Same environment (namespaces, chroot, cgroups) that can be used in different hosts
- Important to have a common description of containers
- Open Container Initiative (OCI): https://www.opencontainers.org/
  - Defines standards for user-space tools
    - Runtime specification: configuration, execution, and lifecycle of a container
    - Image specification: how to represent the data of a container

Portability

- Containers are evolving beyond OS level virtualization
  - If runtime and images are standardized, implementation doesn’t matter.
    - It is not relevant if the container will run on OS Level Virtualization or on a HW VM
- Containers are evolving to Light Weight Virtual Machines
  - Namespaces: can be implemented as a Virtual Machines
  - Cgroups: can be implemented as the Virtual Machines with control tools
  - Images: can be implemented as virtual disks
  - Example: Kata containers


Virtual Machine
Container Command
Container Exec
Container namespaces
Agent
Kernel
Hypervisor
I/O
OCI cmd/spec
Shim
Runtime
gRPC
gRPC
Proxy
gRPC over Yamux

Docker

- Commercial Product
  - Evolution from Solaris Containers and Linux Containers (LXC)
    - At a conceptual level
  - Released as Open Source in 2013

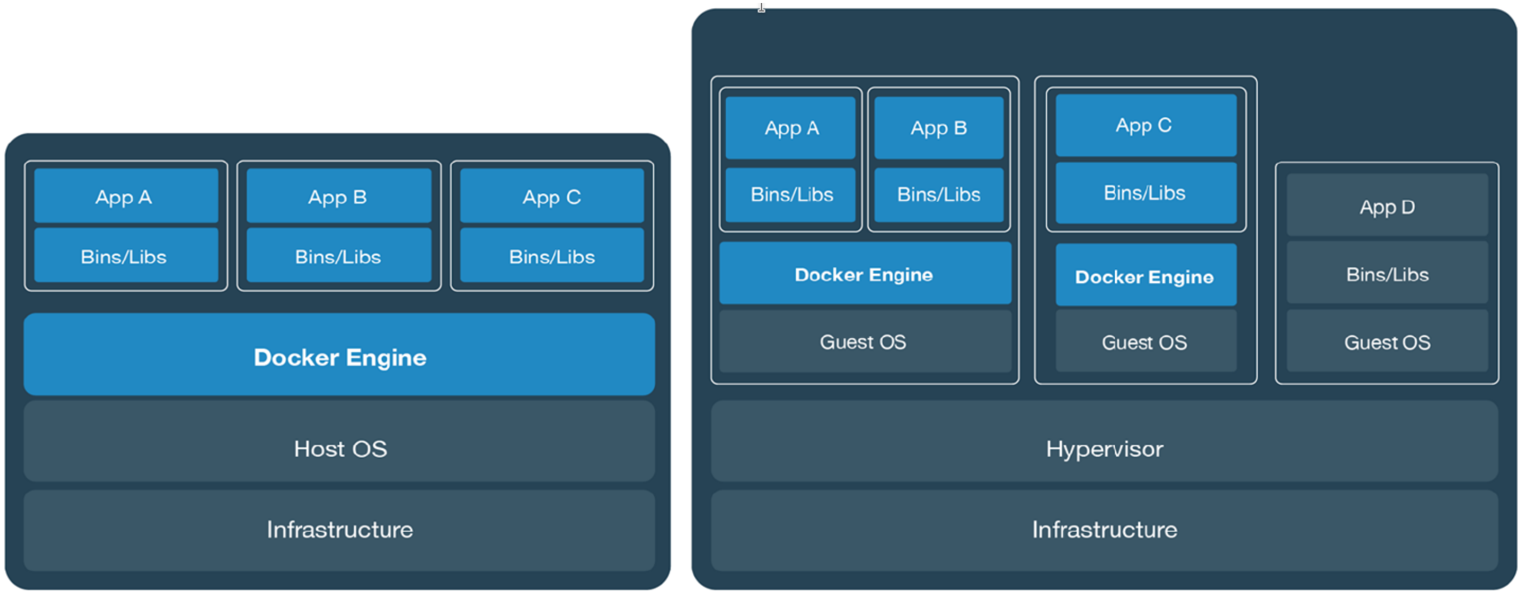
- Core Concept: Container
  - App level construct (unrelated to infrastructure)
  - Composed by multiple functionality (namespaces) in coordination
  - One Container runs One Application (which can spawn child apps)

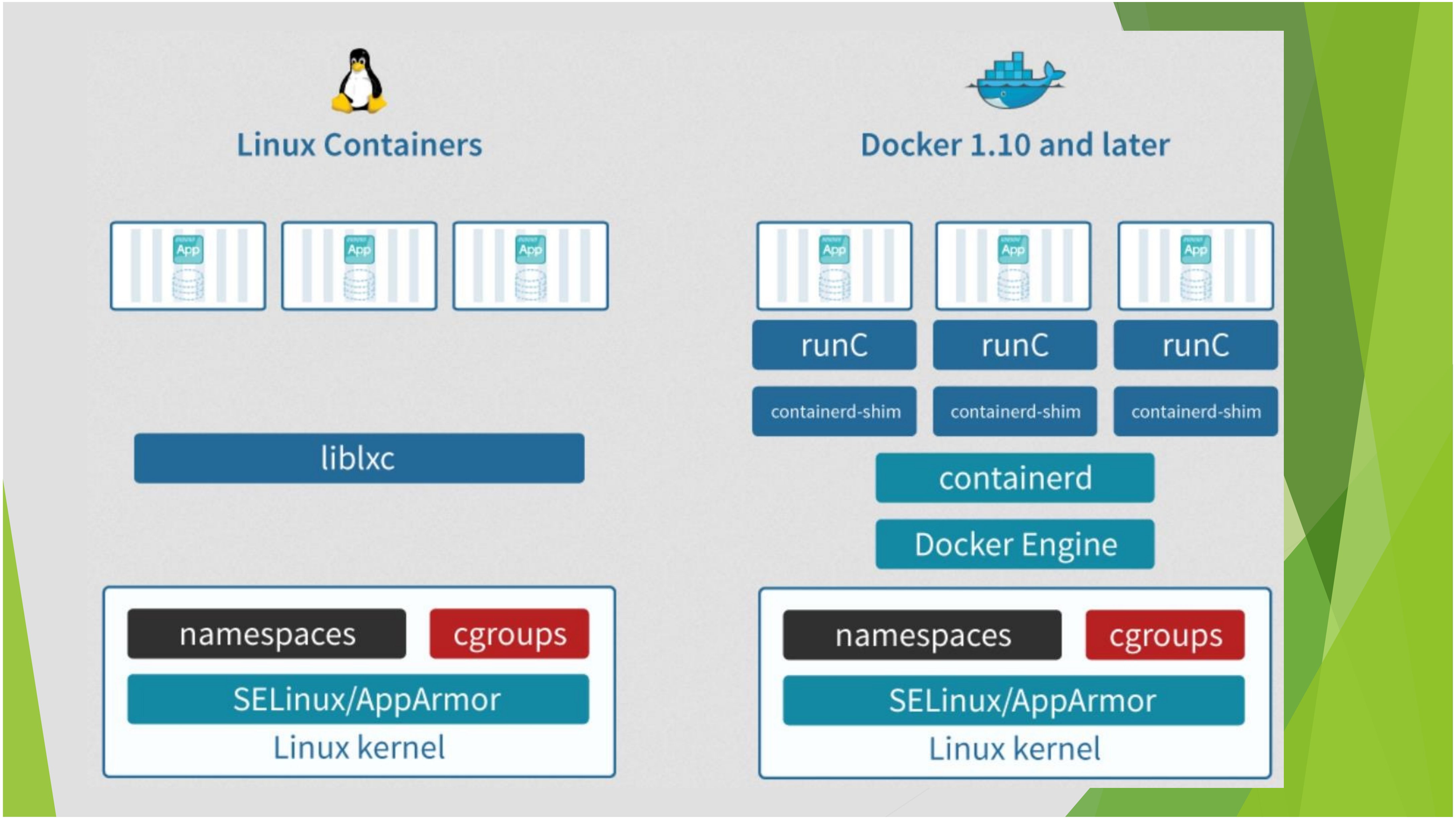
- **Containers pack an application to run in any underlying hardware**
  - including configurations, dependencies, auxiliary data, etc...

Docker Concepts

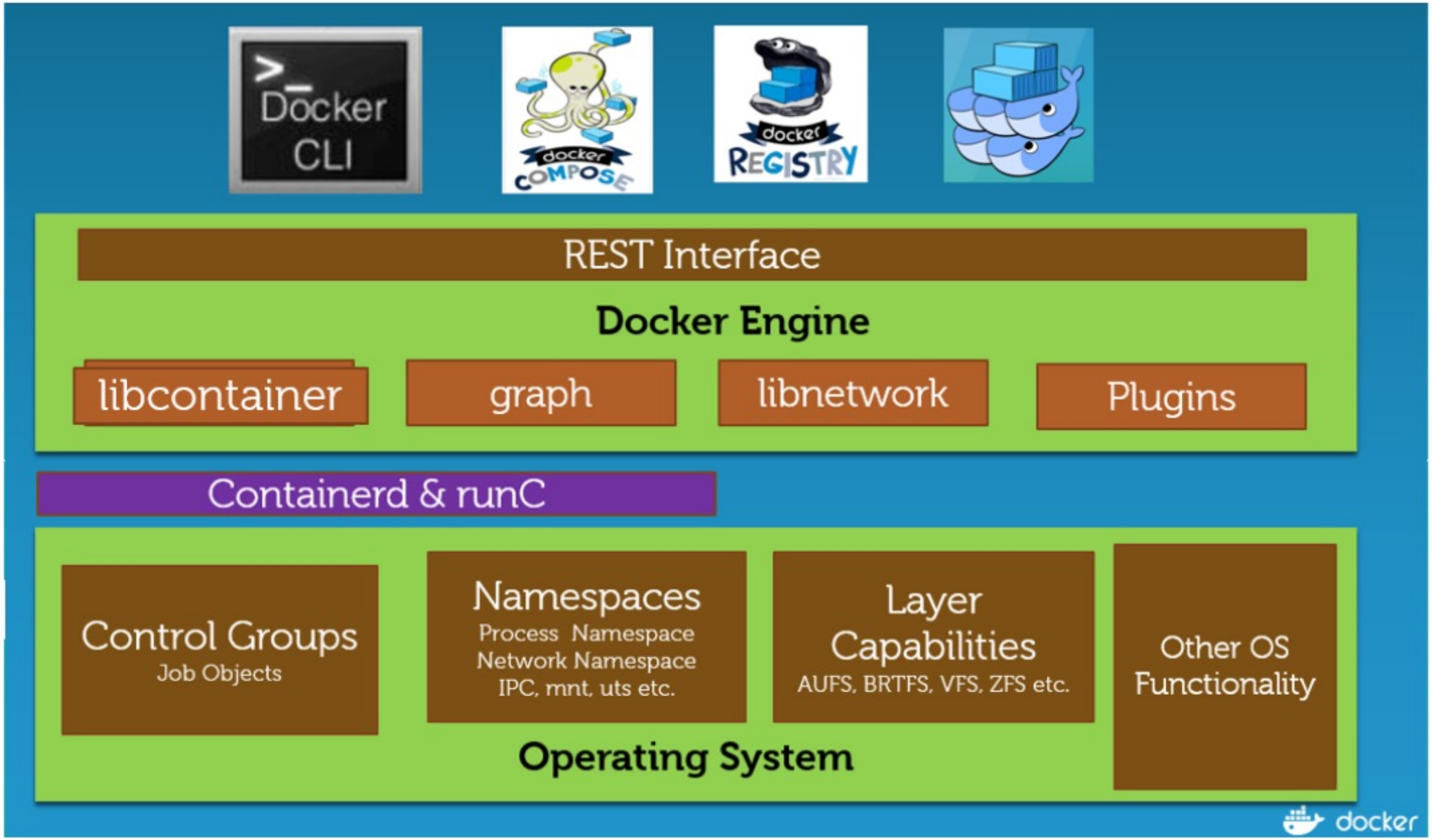
- **Image:** the data of a container (application, libraries, images, etc...)
- **Container:** A running instance of an application.
  - Composed by one or more images. Each image adds a specific functionality
- **Engine:** Software the executes commands for containers
- **Registry:** Repository of docker images
- **Control Plane:** infrastructure to manage containers and images

Containers can run with VMs

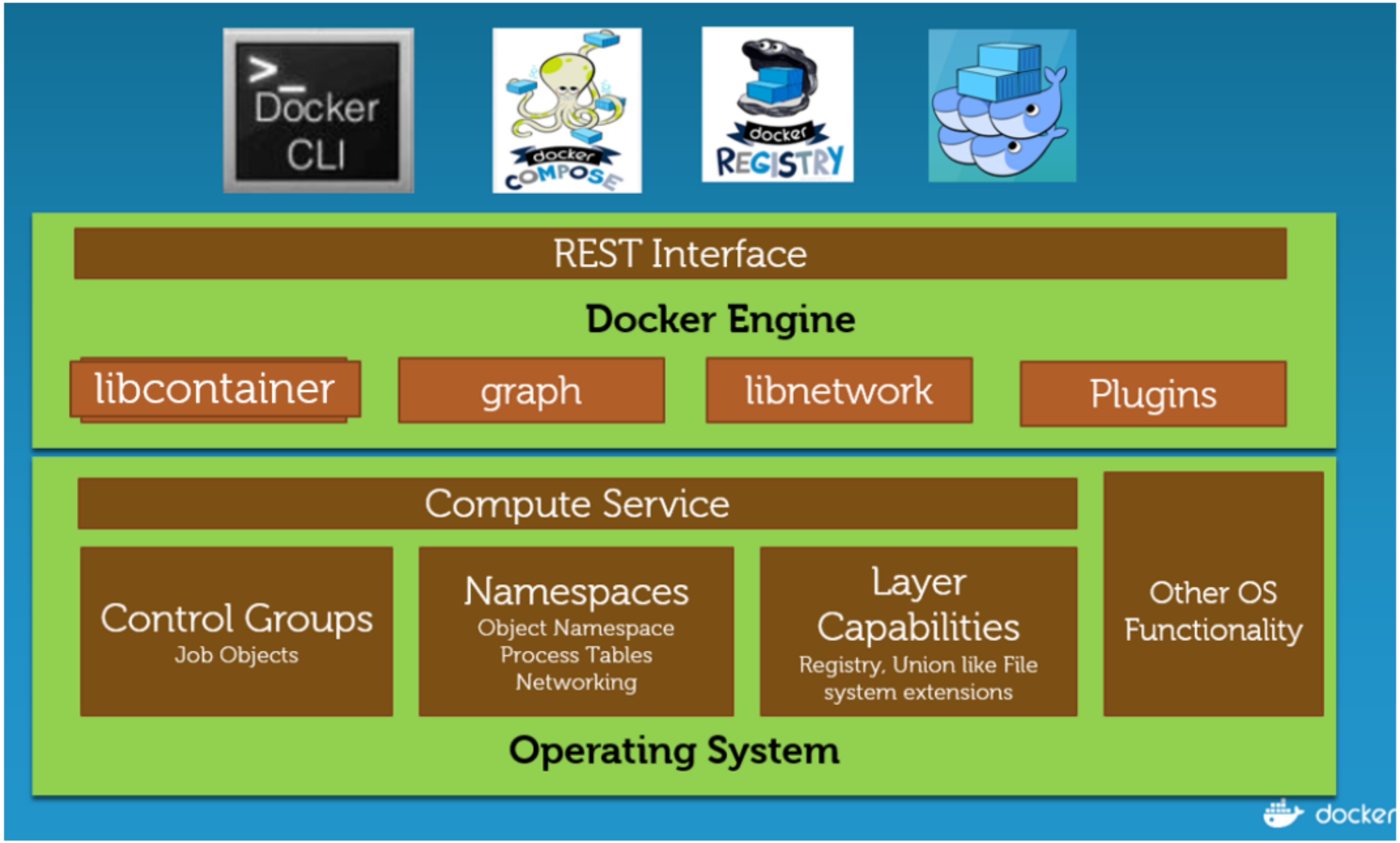



Linux Containers
App
App
App
liblxc
namespaces
cgroups
SELinux/AppArmor
Linux kernel
Docker 1.10 and later
App
App
App
runC
runC
runC
containerd-shim
containerd-shim
containerd-shim
containerd
Docker Engine
namespaces
cgroups
SELinux/AppArmor
Linux kernel

Docker Engine on Linux Platform



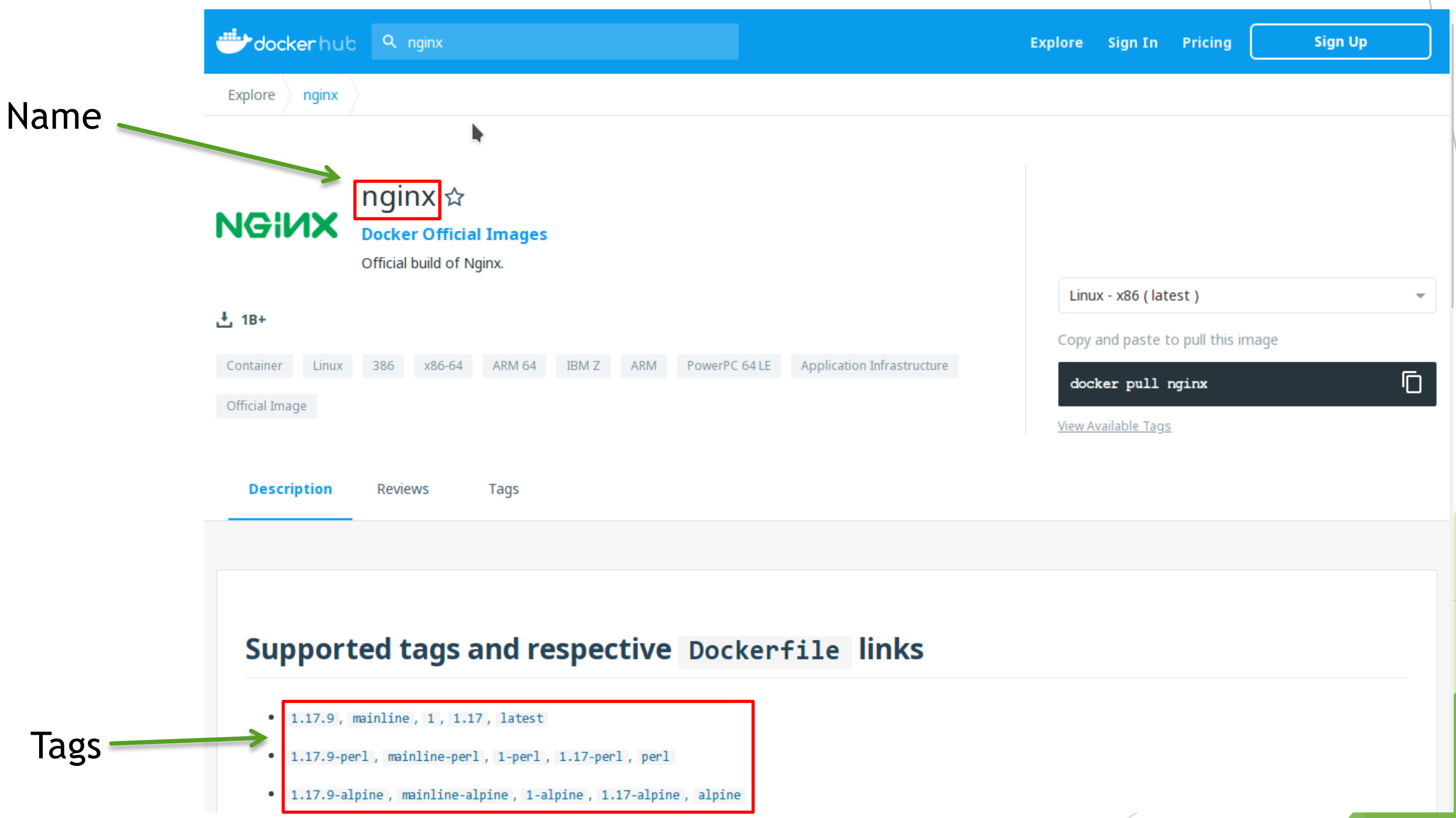
Docker Engine on Windows Platform



Docker Registry

- Central repository to store and deliver images
  - indexed by name and tag
  - can be private to an infraestruture or public (Docker Hub)
- Clients push images to the registry
  - Local client
- Docker Engine pulls images and executes the container
  - Running on servers
- Layers are hashed and indexed allowing reuse
  - Pulling an image only requires the layers that do not exist locally

Docker Hub - Largest Public Registry



```
root@server:~# docker search nginx
NAME                               DESCRIPTION                                     STARS               OFFICIAL            AUTOMATED
nginx                              Official build of Nginx.                        12788               [OK]
jwilder/nginx-proxy                Automated Nginx reverse proxy for docker con…   1750                                    [OK]
richarvey/nginx-php-fpm            Container running Nginx + PHP-FPM capable of…   758                                     [OK]
linuxserver/nginx                  An Nginx container, brought to you by LinuxS…   95
bitnami/nginx                      Bitnami nginx Docker Image                      77                                      [OK]
tiangolo/nginx-rtmp                Docker image with Nginx using the nginx-rtmp…   64                                      [OK]
jc21/nginx-proxy-manager           Docker container for managing Nginx proxy ho…   45
nginxdemos/hello                   NGINX webserver that serves a simple page co…   41                                      [OK]
nginx/unit                         NGINX Unit is a dynamic web and application …   36
jlesage/nginx-proxy-manager        Docker container for Nginx Proxy Manager        35                                      [OK]
nginx/nginx-ingress                NGINX Ingress Controller for Kubernetes         28
privatebin/nginx-fpm-alpine        PrivateBin running on an Nginx, php-fpm & Al…   22                                      [OK]
schmunk42/nginx-redirect           A very simple container to redirect HTTP tra…   18                                      [OK]
blacklabelops/nginx                Dockerized Nginx Reverse Proxy Server.          13                                      [OK]
nginxinc/nginx-unprivileged        Unprivileged NGINX Dockerfiles                  13
centos/nginx-112-centos7           Platform for running nginx 1.12 or building …   12
raulr/nginx-wordpress              Nginx front-end for the official wordpress:f…   12                                      [OK]
centos/nginx-18-centos7            Platform for running nginx 1.8 or building n…   12
nginx/nginx-prometheus-exporter    NGINX Prometheus Exporter                       9
sophos/nginx-vts-exporter          Simple server that scrapes Nginx vts stats a…   7                                       [OK]
mailu/nginx                        Mailu nginx frontend                            6                                       [OK]
bitnami/nginx-ingress-controller   Bitnami Docker Image for NGINX Ingress Contr…   4                                       [OK]
ansibleplaybookbundle/nginx-apb    An APB to deploy NGINX                          1                                       [OK]
wodby/nginx                        Generic nginx                                   0                                       [OK]
centos/nginx-110-centos7           Platform for running nginx 1.10 or building …   0
```

Workflow

- Search docker image

$ docker search nginx

- Pull docker image

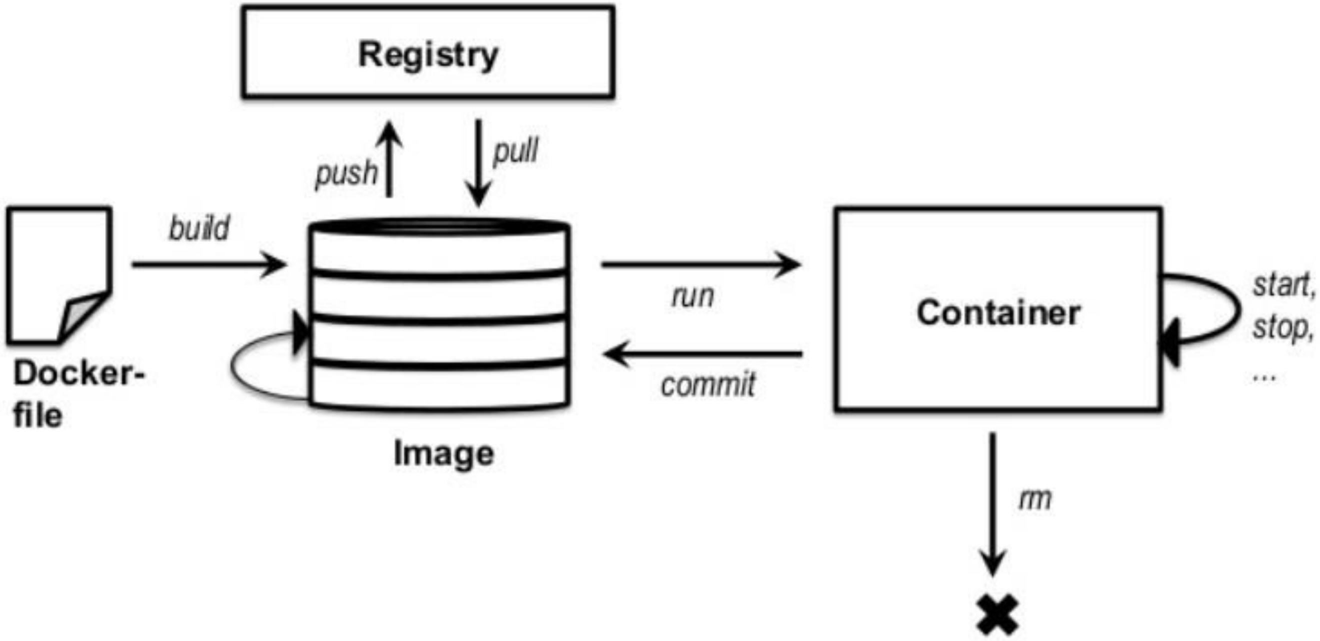
$ docker pull nginx

- List local docker images

$ docker image ls

- Run container

$ docker run -d --name frontend-server nginx



Docker file

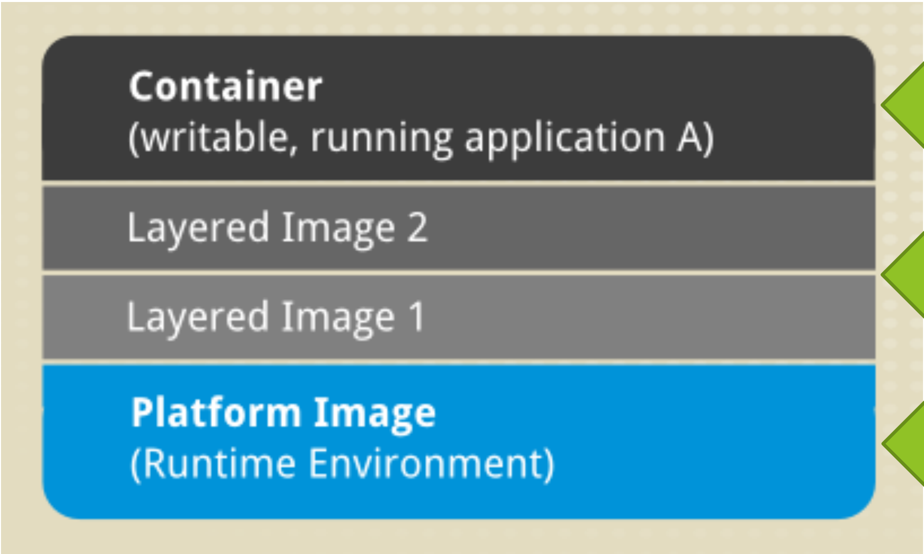
- **Sequence of commands** to prepare the container for execution
  - Lists dependencies
  - Lists commands to be executed inside container
  - Sets the command to execute on start
- Used locally or distributed in registry
  - Public registry: Docker hub
- May define versions for same software
  - nginx:latest, nginx:perl, nginx:alpine…

```
FROM debian:stretch-slim

LABEL maintainer="NGINX Docker Maintainers <docker-maint@nginx.com>"

ENV NGINX_VERSION 1.15.9-1~stretch
ENV NJS_VERSION   1.15.9.0.2.8-1~stretch

RUN set -x \
        && apt-get update \
        && apt-get install --no-install-recommends --no-install-
suggests -y gnupg1 apt-transport-https ca-certificates \
        && \

...OMMITED...

RUN ln -sf /dev/stdout /var/log/nginx/access.log \
        && ln -sf /dev/stderr /var/log/nginx/error.log

EXPOSE 80

STOPSIGNAL SIGTERM

CMD ["nginx", "-g", "daemon off;"]
```

Images

- Containers have their initial data in images
  - Contain data, applications, libraries
  - Contain an entry point to be executed when the container is initiated
- Composed by multiple layers
  - A base image
  - Several images, changing the content (with differences)
  - Exploits overlayfs (a unionfs similar to AUFS)
    - Can use others like btrfs, ZFS
- Read Only and Non persistent
  - Serve as templates for containers to start
  - Multiple containers will use the same images (containers are ephemeral)

Image layering



Writable, **non-persistent**, layer, always added when running a container

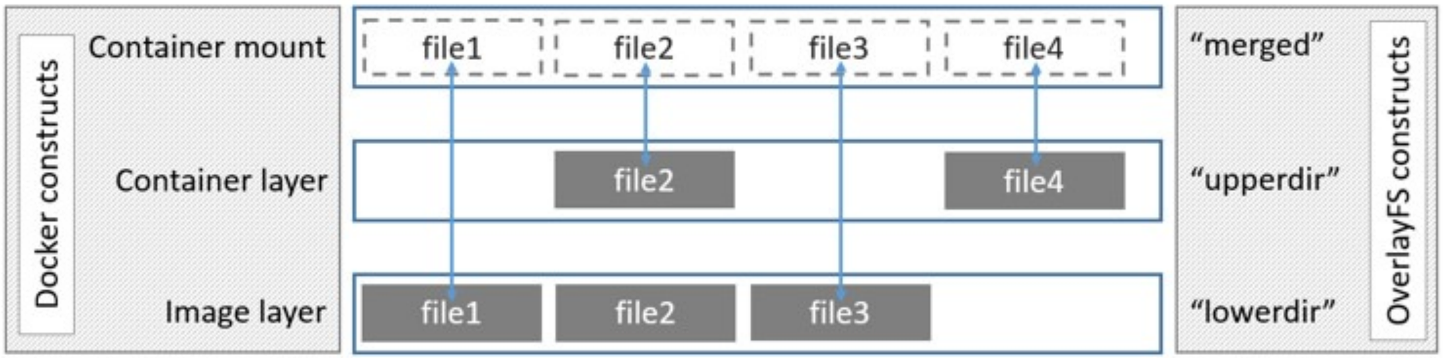
Read-Only layers with dependencies and

Read-Only base image, defining the runtime,
packages and base utilities for the

Available at /var/lib/docker/…

Images

- It’s important to keep images small
  - Dont: base your image on a full blown distro (e.g. debian)
  - Do: base your image on a small distro (e.g. Alpine)
- Presented on the host
  - /var/lib/docker/overlay2/<ID>/
    - /merged: FS seem inside container
    - /diff: differences from base



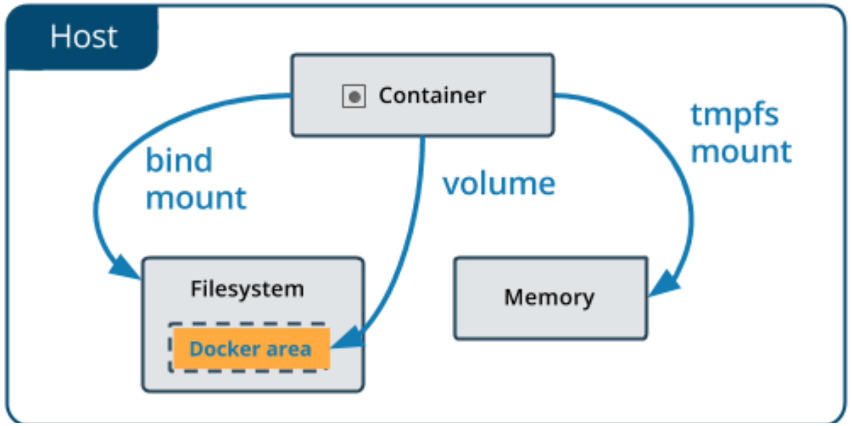
Images

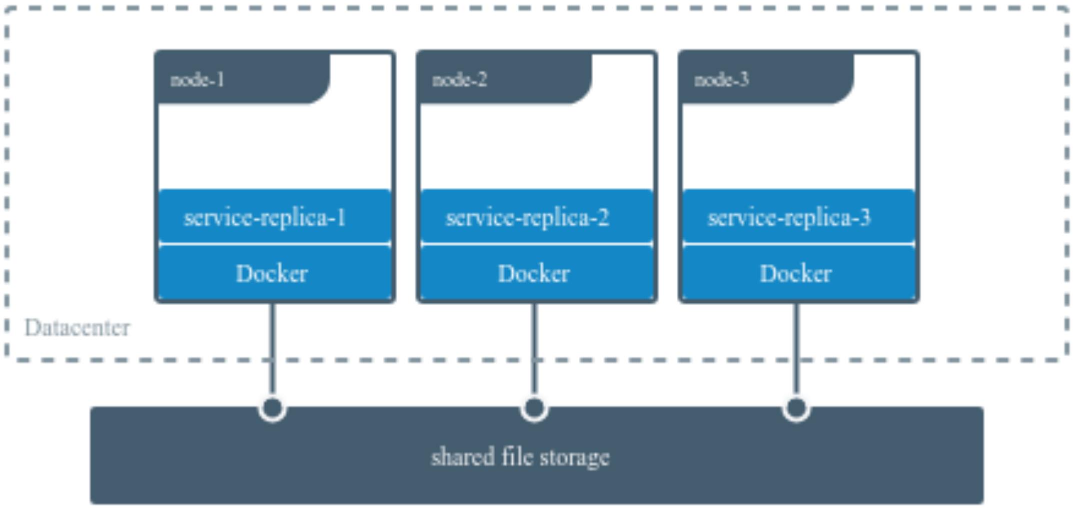
- OverlayFS has limitations!
  - Only a subset of the calls is supported
  - May create issues with some applications or decrease performance

- **Issues:**
  - open: opening a file RW after the same file is open as R will fail.
    - first open (R) will refer to a file in the base image
    - second open (RW) will result in a copy of the file being created (stored in the diff folder)
    - File descriptions will not point to the same file
  - rename: not supported
    - applications must copy and then unlink
    - much slower and will take additional disk space

Container Persistence

- Top layer is writable, but not persisted!
  - Stopping the container will result in data loss
- Persistence must be achieved through external resources
  - Bind mount: Host path
  - Volume: Docker managed resource
  - ... or external, database cluster





Container Persistence

- Bind mounts:
  - mount an existing path into a path inside the container
- Example: mount *app* dir from host, into */var/www* inside the container
  - Usefulness: persistence or add shared/specific data into container (ex, web page, static resources)

```
$ docker run -d\
    --mount type=bind,source=/app,target=/var/www\
    -p 8080:80
    --name frontend-server
    nginx
```

Container Persistence

- Volume mounts:
  - mount a specific storage object (managed by docker) at a given path
  - volumes are not deleted with containers
- Example: mount *app* volume, into */var/www* inside the container as read-only
  - Usefulness: persistence or add shared/specific data into container (ex, web page, static resources)

```
$ docker run -d\
    --mount type=volume,source=app:/var/www:ro\
    -p 8080:80
    --name frontend-server
    nginx
```

Container Network

- Containers run with specific network configuration
  - No ingress network access added by default
- Standard physical networks (others can be created)
  - bridge: virtual switch
  - host: network only connected to host OS
  - none: isolated network (only with loopback, and can be customized)
- Others
  - overlay: used to provide connectivity over multiple engines (swarm mode)
  - macvlan: assigns specific MAC address to container

Container Network Ports

- Services inside container cannot provide services to other applications
  - No network services by default
- Run command must **expose ports**
  - Outside ports mapped to inside ports
- E.g. map external port 8000 to internal (inside container) port 80

```
$ docker run -d -p 8000:80 --name frontend-server nginx
```

Sensitive data

- Frequently, there is sensitive data that must be available to containers
  - Database/ext service access credentials, certificates, admin credentials
  - Sensitive data is container specific, or shareable among several containers
- Direct (**no so correct**) approach: Build images with credentials
  - must build different images for each credential pair
  - must rebuild images when credentials change
  - credentials frequently end in the teams source control (e.g. Git repository)

Docker Compose

- Compose services created from multiple containers
  - that is... consider the specification of a complete service, and not
  - docker-compose.yml file

- Example:
  - specifies service name: nginx
  - container name and build ref
  - volumes to add
    - uses an environmental variable
  - ports to expose to outside

```yaml
version: ‘3’
services:
    nginx:
        container_name: nginx
        build:
            context: ./nginx
            dockerfile: Dockerfile
        volumes:
            - app:/var/www
            - ${NGINX_LOG_PATH}:/var/log/nginx
        ports:
            - 80:80
            - 443:443
```

Sensitive data: Docker Secrets

- Allows creating blobs of data with restricted access
  - Exist independently of the containers
  - Allow changing the secret without changing the container
    - ex: database access creds for dev, stagging and prod
  - Only accessed inside specific containers
    - at /run/secrets/secret_name

- Manage secret: docker secret create/inspect/list/rm
- Add to service: docker service... --secret secret_name
  - Will expose secret_name inside container

Docker Compose

```
version: ‘3’
services:
    nginx:
        container_name: nginx
        build:
            context: ./nginx
            dockerfile: Dockerfile
        ...
        secrets:
            key_file:
                file: server-key.pem
            cert_file:
                file: server-cert.pem
```

Kubernetes

Kubernetes

- A platform for **orchestrating (Docker/Containerd) containers** in a **clustered environment, usually** with **multiple hosts**

- Provides:
  - **container grouping:** groups containers in same environment
  - **load balancing** distributes load across physical resources
  - **auto-healing:** recover from failures (auto start containers)
  - **scaling features:** Increase number of containers as required by load
  - …

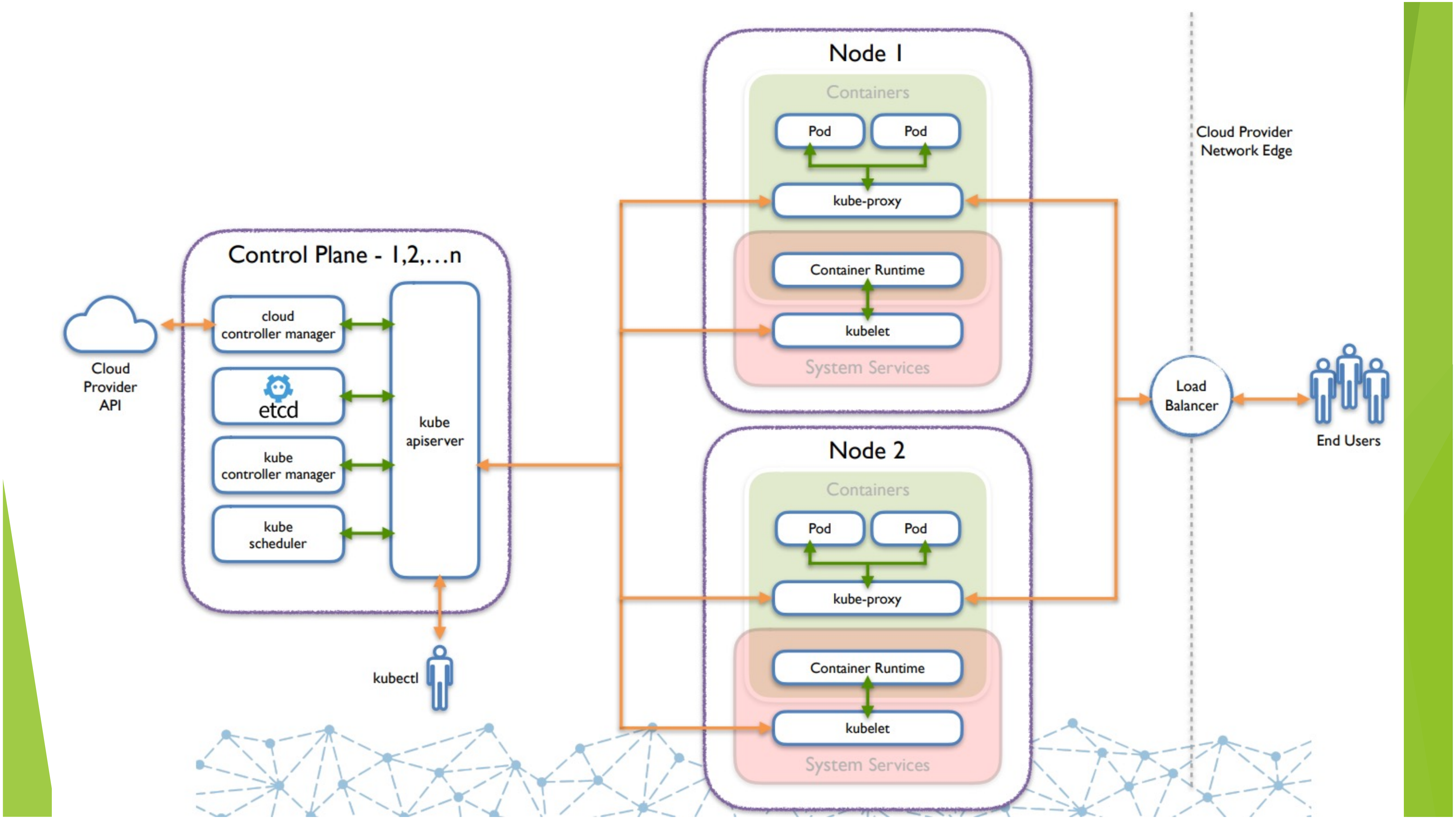
- Motivation: running individual containers is not enough. Kubernetes is an **orchestration solution that keeps services operating**

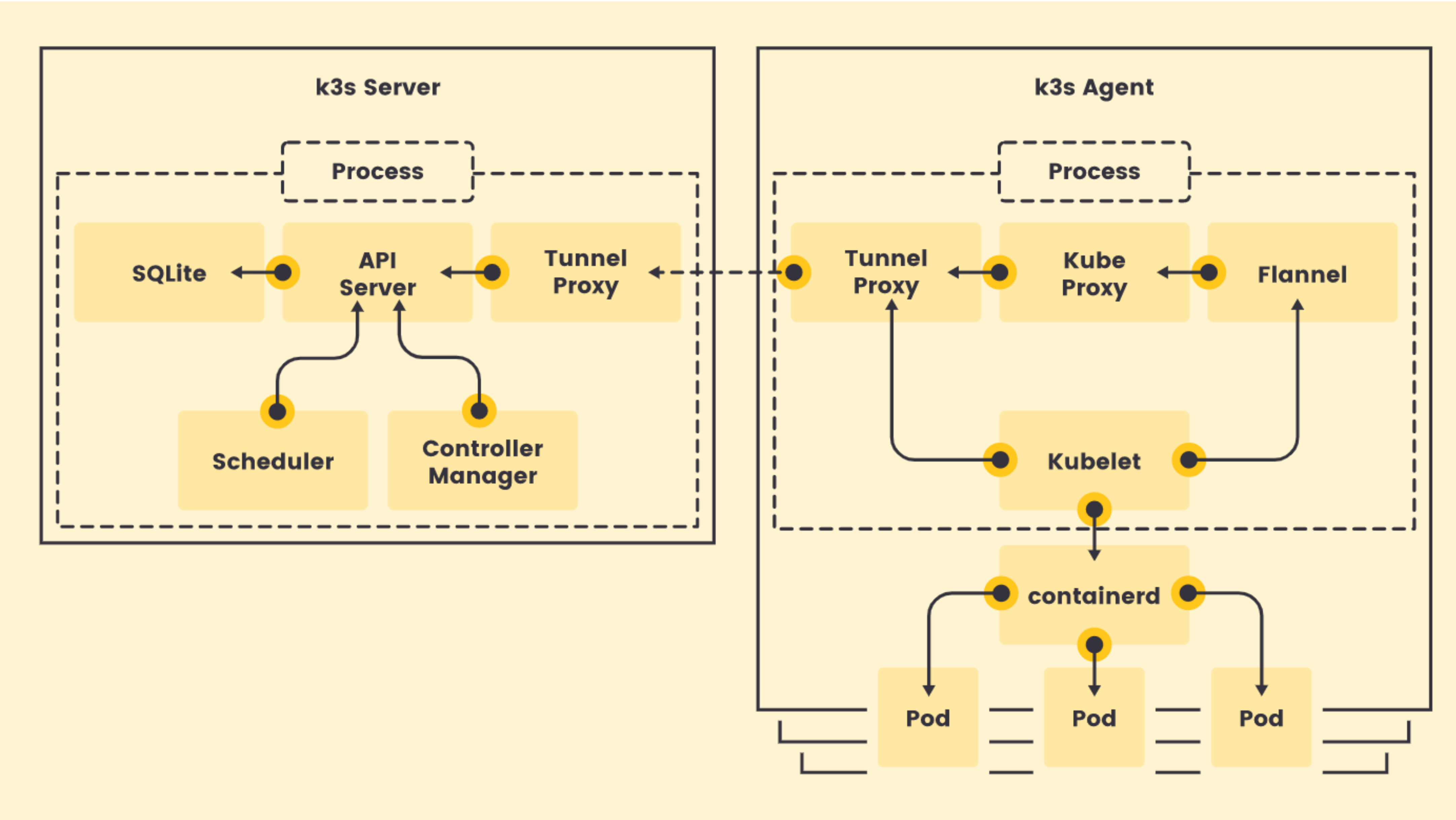
Kubernetes

- An orchestrator manages the lifecycle of an entity (VM, container, etc..)
  - In this case: the lifecycle of containers
  - How it is created
  - Where it is instantiated
  - How it is balanced to new nodes
- Orchestrators usually also manage related resources
  - Data volumes
  - Secrets
  - Configuration values
  - External resources and services
- Orchestrators are not simple automatic deployment tools
  - They have an integrated view of the resources and make decisions to optimize placement

Kubernetes

- Focus on managing hundreds/thousands of containers, on a fleet of thousands of nodes
  - While dealing with rollout of new versions, failure, security, …
- Implement a software defined platform: given the specification, the software will be kept running, as long as there are resources


Node 1
Containers
Pod
Pod
kube-proxy
Container Runtime
kubelet
System Services
Cloud Provider
Network Edge
Control Plane - 1,2,...n
Cloud
Provider
API
cloud
controller manager
etcd
kube
controller manager
kube
scheduler
kube
apiserver
Load
Balancer
End Users
Node 2
Containers
Pod
Pod
kube-proxy
Container Runtime
kubelet
System Services
kubectl


k3s Server
Process
SQLite
API Server
Tunnel Proxy
Scheduler
Controller Manager
k3s Agent
Process
Tunnel Proxy
Kube Proxy
Flannel
Kubelet
containerd
Pod
Pod
Pod

Key software components

- Kube-apiserver: provides a REST API for clients to interact with
  - Control plane
  - Data Storage
- Etcd: The Cluster Datastore, with a consistent and highly available key value storage
  - For configuration data
- Kube-Controller-Manager: Manages all component control loops
  - Ensures the cluster is at the desired state
- Kube-Scheduler: Places new Pods into Nodes
  - Takes in consideration MANY scheduling decisions (affinity, anti-affinity, constraints, data locality….)

Key software components

- Kube-Proxy: Manages network rules at each node, and does connection forwarding / load balancing
  - Similar to the docker proxy
- Kubelet: takes in consideration the description of each Pod (PodSpecs) and ensures (locally in the node) that the containers are running as expected
- Container Runtime Engine: A software that runs containers, many supported
  - Containerd (docker)
  - Rkt
  - cri-o
  - Virtlet
  - Kata

Networking

- Pods have an internal network namespace
  - You have multiple pods with the same networks!
- Pod Network: Cluster wide network for Pod-to-Pod communication
- Service Network: Virtual IPs (durable), managed by kube-proxy for service discovery
  - Because Pod IP address are volatile, services provide anchor points for communication

Namespaces

- Logical Clusters inside the Real Cluster
- Provide the means for partitioning the cluster to multiple teams/projects
- Objects are created under a namespace and deletion of the namespace may also delete all objects
  - Exceptions are the resources related StatefulSets (later)
- Some namespaces are always present (default, kube-system,…)
  - Typical namespaces: production, development, monitoring

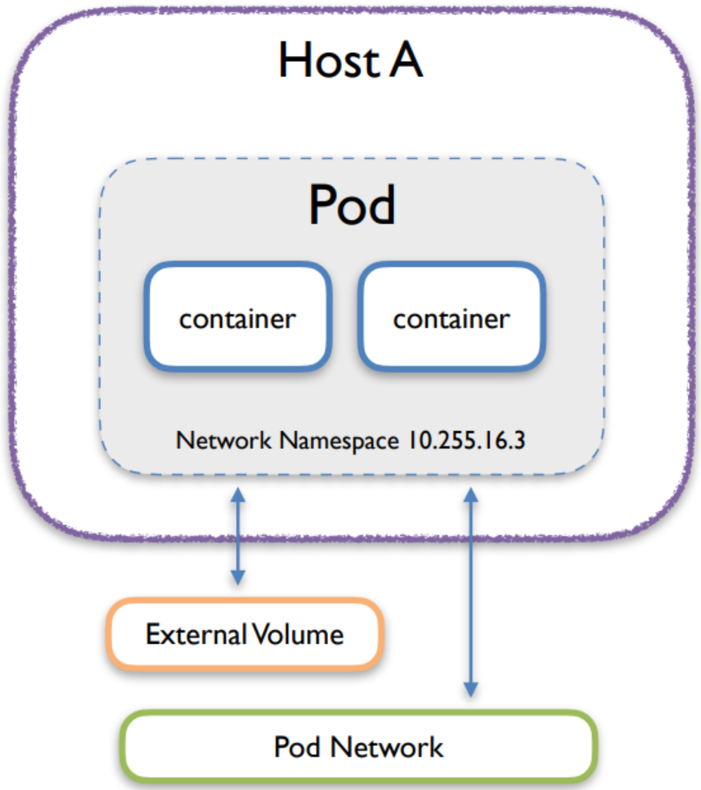
Pod

- A group of Containers. A basic management unit.
  - Includes information about what to run, and how to run something
  - Mimics a local host with applications (containers)

- **<u>Containers are ephemeral</u>**

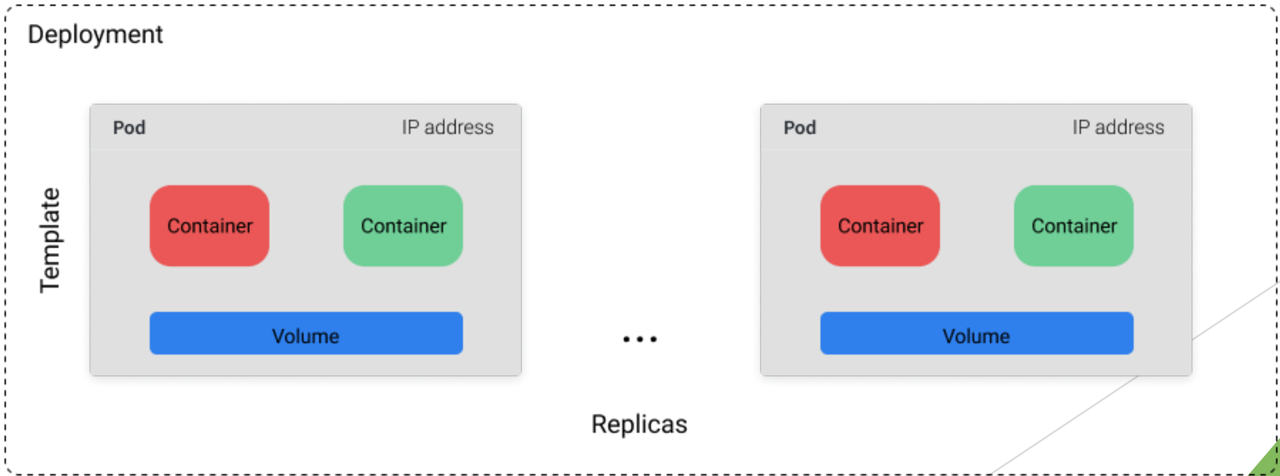
- Containers are tagged to versions and we go changing version as the infrastructure evolves

- Pod contents are always co-located and co-scheduled



Deployment

- A set of pods that should operate together
  - Can be a software, but also includes volumes, networks, secrets, etc..
- Defined using a declarative format (yaml file) stating the desired state of the pods



Deployment

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: app
  namespace: diving-into-k3s
spec:
  replicas: 1
  selector:
    matchLabels:
      app: app
  template:
    metadata:
      labels:
        app: app
    spec:
      containers:
      - name: app
        image: 10.110.0.3:5000/app:v1
        resources:
          requests:
            memory: "32Mi"
            cpu: "10m"
          limits:
            memory: "128Mi"
            cpu: "500m"
        ports:
        - containerPort: 8080
```

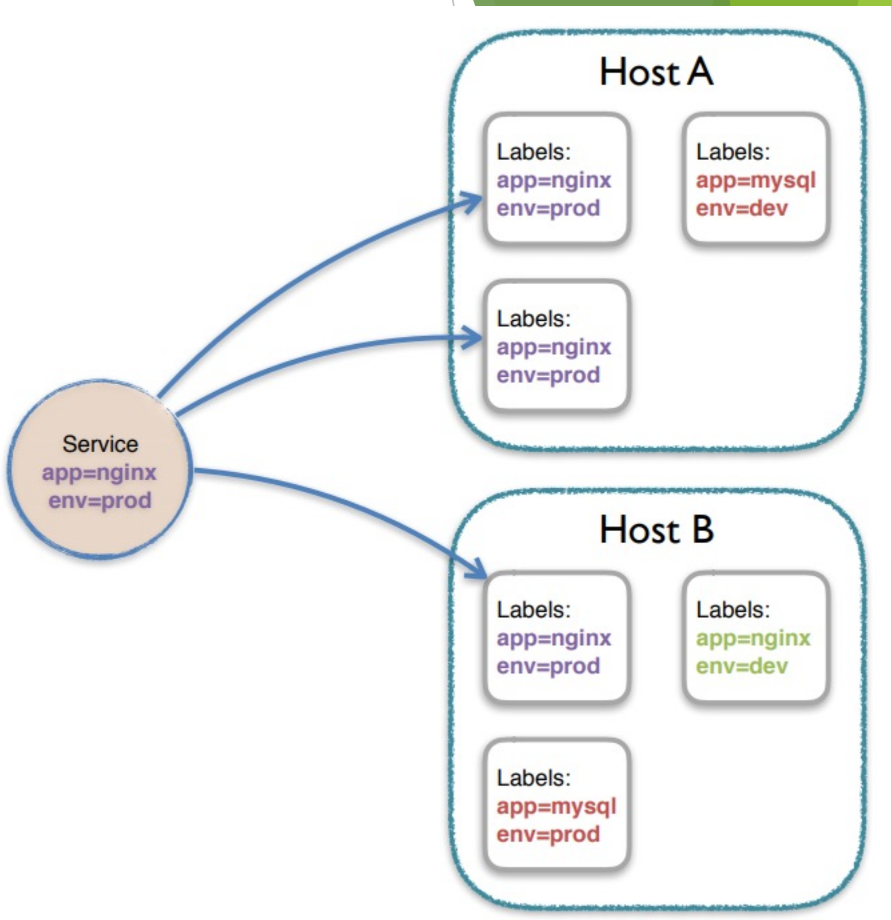
- What is the context
- How many instances
- How to find pods in an infrastructure
- What to run. In this case, a Docker container
- Resource limits
- How to expose

StatefulSet

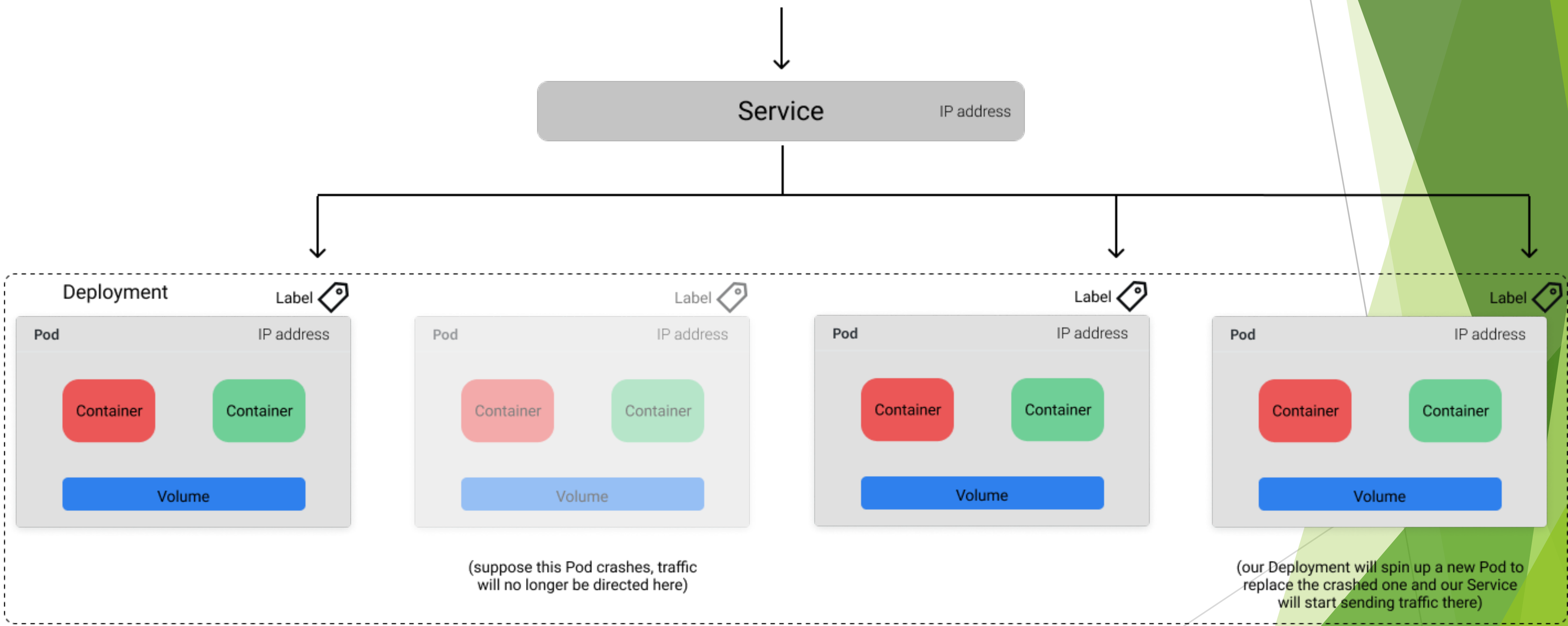
- Similar to a deployment but Pods are stateful
  - Have persistent and unique IDs
  - Are started by a specific order
  - Storage is persistent
    - Requires a StorageClass to actually store data
  - Graceful deployment and scaling
- Useful for stateful software, whose instances have unique roles
  - e.g a Database

Service

- K8s abstracts internal aspects from users, such as the IP addresses of replicas
  - Also, number and location of said replicas
- A service provides the visibility to reach a deployment **from inside the cluster**
  - Allows Pods to reach other pods
  - Service provides a stable endpoint, no matter how many pods are available (or where)
- Persistent
  - Static cluster IP
  - Static DNS Name



Service



Service
IP address
Deployment
Label
Pod
IP address
Container
Container
Volume
Label
Pod
IP address
Container
Container
Volume
Label
Pod
IP address
Container
Container
Volume
Label
Pod
IP address
Container
Container
Volume
(suppose this Pod crashes, traffic will no longer be directed here)
(our Deployment will spin up a new Pod to replace the crashed one and our Service will start sending traffic there)

Service

- A service named “app” allows Pods to access other pods by DNS, through the service
  - e.g. curl http://app would be redirected to one pod (round robin)

```yaml
apiVersion: v1
kind: Service
metadata:
  name: app
  namespace: diving-into-k3s
spec:
  ports:
  - port: 80
    targetPort: 8080
  selector:
    app: app
```

Name and context

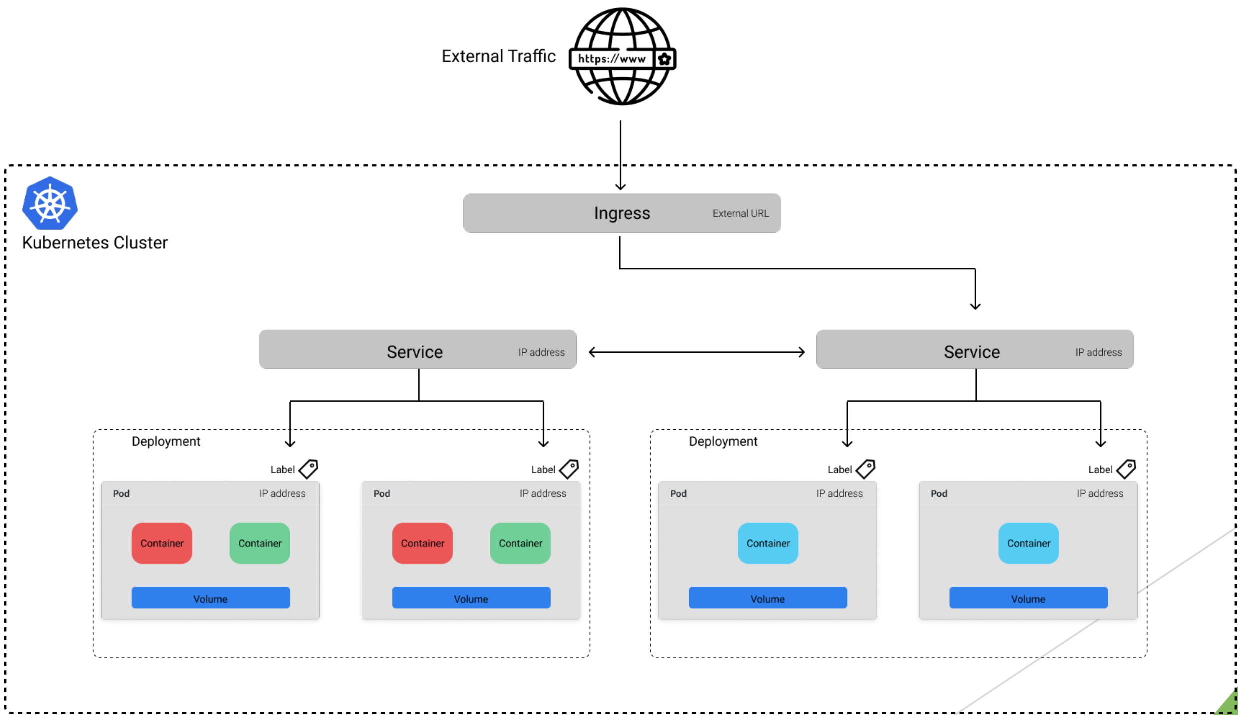
Which port is exposed

Which port used to reach Pods

Which pods are used

Ingress

- Similar to a Service, makes some services visible to the outside world
- Frequently uses a proxy software (Nginx, Traefik)



Ingress with Traefik

```
---
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: app-k3s
  namespace: diving-into-k3s
  annotations:
    kubernetes.io/ingress.class: traefik
    traefik.ingress.kubernetes.io/frontend-entry-points:
http,https
    traefik.ingress.kubernetes.io/redirect-entry-point:
https
    traefik.ingress.kubernetes.io/redirect-permanent:
"true"
spec:
  rules:
  - host: app.k3s
    http:
      paths:
      - path: /
        backend:
```

Name and Namespace

Traefik Specific Configuration for This ingress

How to expose the Pods (the hostname and port)

What Pods to use (will use round robin)

Job

- An ephemeral execution to run one time
- In contrast to a pod in a deployment, which are expected to be running, jobs are one-time tasks.
- Kubernetes ensures the job runs as stated, dealing with availability o

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: model-train-job
spec:
  template:
    spec:
      containers:
      - name: ml-model-train
        image: ml-development:1.2.1
        command: ["python", "train.py"]
      restartPolicy: Never
  backoffLimit: 5
```

How should we create the Pod for this Job?

How many times should we try to complete this Job before quitting?

ConfigMaps

- Provide the means to create generally available configurations for your entities
  - Used for non-confidential data
  - Key-Value pairs
  - Can be injected using configuration files in volumes, command line arguments or environmental variables
- Allows decoupling the software from the execution environment specific configuration
  - Usually, different environments (dev and prod?) may have different configurations
- ConfigMaps are created, and then attributed to a Pod in the deployment
  - Only the required Pods will access the configuration

ConfigMaps

A subPath (the Item) in a Volume is mapped to the Containers.

The config file (nginx.conf) will be available at /etc/nginx/nginx.conf

ConfigMap Items are mapped to a Volume.

Definition of the ConfigMap
(notice that it is restricted to the namespace)

ConfigMap has a name, and items inside it

```
# Create nginx deployment
---
...
    containers:
     - image: nginx:alpine
      name: nginx
      ports:
       - containerPort: 80
      resources: {}
      volumeMounts:
       - name: nginx-conf
        mountPath: /etc/nginx/nginx.conf
        subPath: nginx.conf
        readOnly: true
    volumes:
     - name: nginx-conf
      configMap:
       name: nginx-conf
       items:
        - key: nginx.conf
         path: nginx.conf
---
apiVersion: v1
kind: ConfigMap
metadata:
 name: nginx-conf
 namespace: diving-into-k3s
data:
 nginx.conf: |
  …content of the nginx.conf configuration
```

Secrets

- Similar to ConfigMaps, but aims to store a small, sensitive information blob (password, token, key)
  - By default are stored as a base64 string, with access limited only by the API
  - Kubernetes allows both Encryption and Role Based Access Control (RBAC)
    - Secrets will be strongly encrypted and only provided to allowed users
  - Differ from ConfigMaps as:
    - They can only be used by kubelet, an environmental variable or a volume
      - Cannot pass it through arguments
    - They have a type, defining its purpose

Secrets

Builtin Type	Usage
Opaque	arbitrary user-defined data
kubernetes.io/service-account-token	service account token
kubernetes.io/dockercfg	serialized ~/.dockercfg file
kubernetes.io/dockerconfigjson	serialized ~/.docker/config.json file
kubernetes.io/basic-auth	credentials for basic authentication
kubernetes.io/ssh-auth	credentials for SSH authentication
kubernetes.io/tls	data for a TLS client or server
bootstrap.kubernetes.io/token	bootstrap token data

- The Opaque type has no restrictions on content
  - Meant for arbitrary user data loaded from a file, or specified interactively
- The remaining types impose specific content format