

The Journey to Cloud

Kubernetes Basic Concepts

Niklaus Hirt

DevOps Architect / Cloud Architect

nikh@ch.ibm.com



The Journey to Cloud



IBM Cloud

Who am I?

Niklaus Hirt

Passionate about tech for over 35 years

- High-school in Berne
- Degree in Computer Science at EPFL
- ELCA
- CAST
- IBM

✉ nikh@ch.ibm.com

🐦 @nhirt



Agenda - Kubernetes Basic Concepts

Module - BASIC

Module 0: Prepare the Labs

Module 1: Microservices

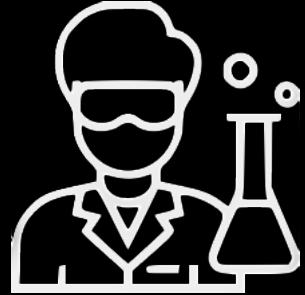
Module 2: Containers with Docker

Module 3: Kubernetes

BREAK

Module 4: Kubernetes Hands-On

END – 13:00

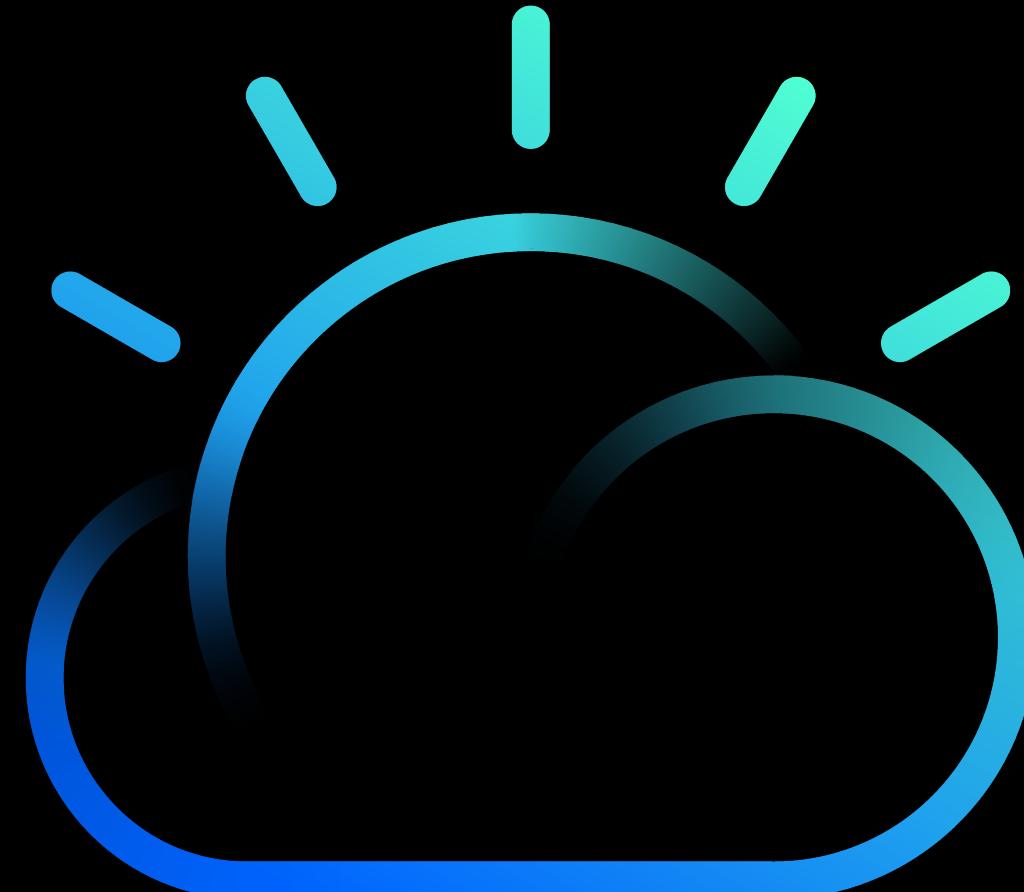


Sources and documentation will be available here:

https://github.com/niklaushirt/k8s_training_public



°° The Journey to Cloud
Prepare the Labs



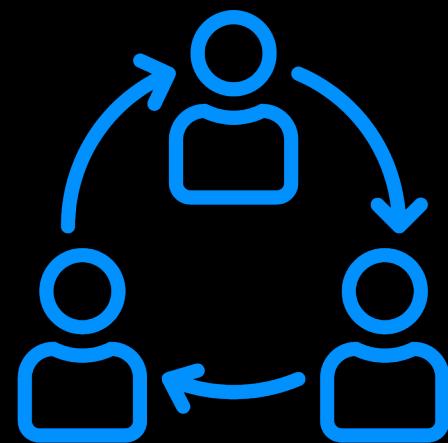
IBM Cloud

Session Objectives

Attendees will be grouped in **teams** wherever it makes sense to facilitate collaborative work.



Following some lectures will be **hands-on** work that each team collaborates to complete.



Teams

black 31701

olive 31711

peru 31715

white 31702

brown 31712

chocolate 31716

red 31703

lightblue 31713

orchid 31717

blue 31704

orange 31708

gold 31718

yellow 31705

purple 31709

pink 31719

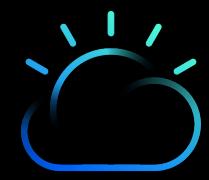
lime 31706

maroon 31710

violet 31720

cyan 31707

firebrick 31714



Collector - Accessing team web site

http://158.177.137.195:{port#}

Team name / color will be shown

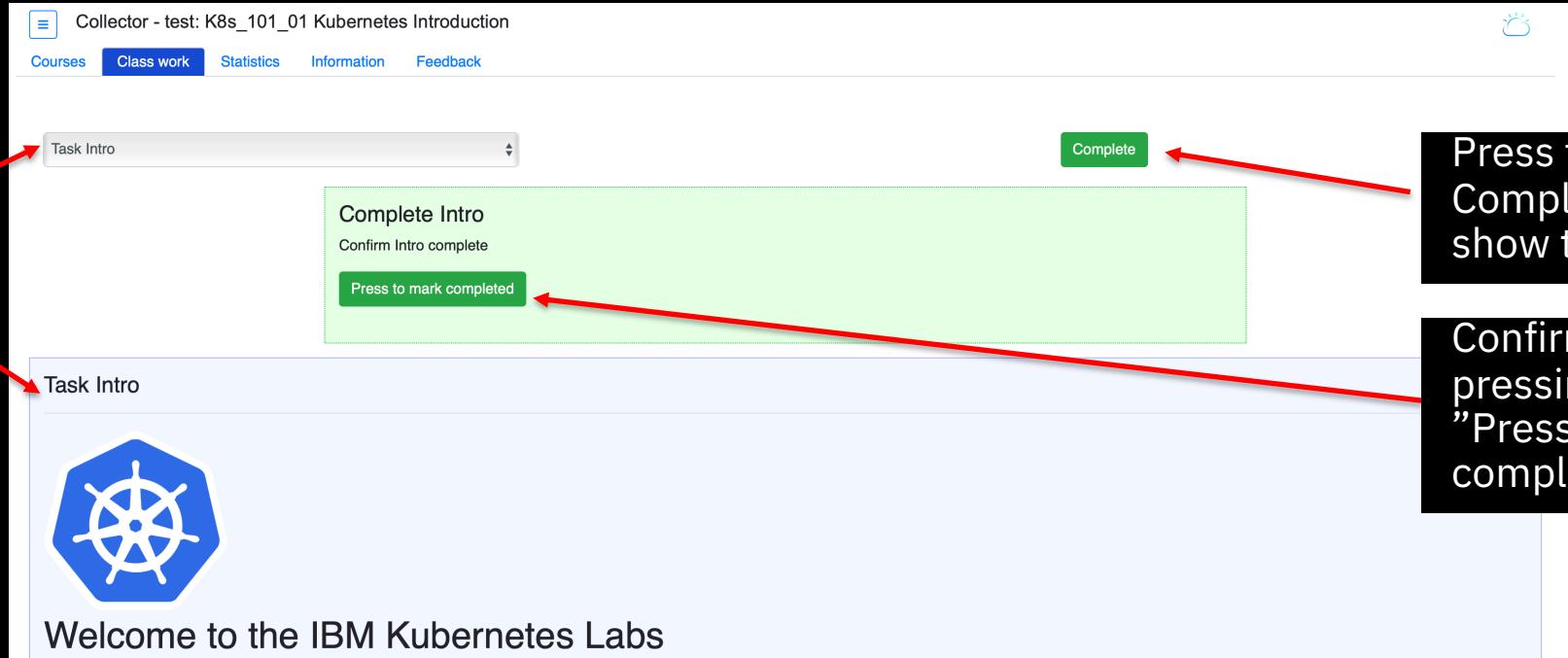
blue 31704

The screenshot shows a web browser window with the title "Collector - blue". The navigation bar includes links for "Courses", "Class work", "Statistics", "Information", and "Feedback". Below the navigation bar, a section titled "Catalog of courses" displays a list of courses. A dropdown menu is open over the first item, "KUB01 Lab Setup", with the text "select course" preceding the list. The list contains five items: "KUB01 Lab Setup", "KUB02 Kubernetes Introduction", "KUB03 Kubernetes Labs", and "KUBADV01 Istio". To the right of the dropdown, there is a button labeled "Begin course". A large callout box with a red border and white text, containing the instruction "Select course and press button to begin", points to the "Begin course" button.

Current course catalog

Collector – Class work

Select class work and the blue portion of the screen is shown



Press the green Complete button to show the green portion.

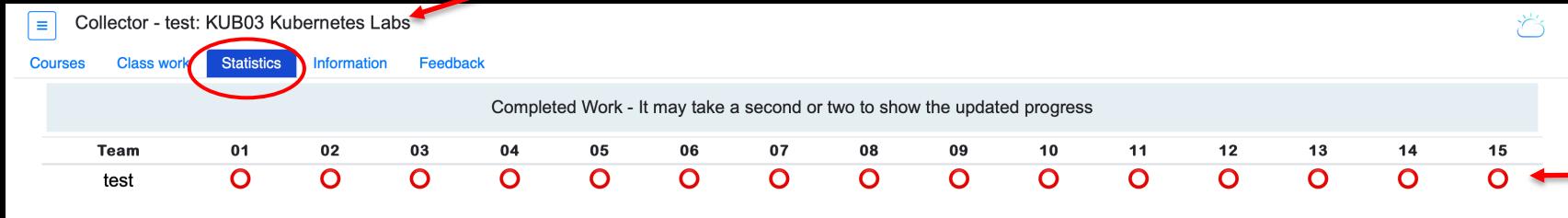
Confirm completion by pressing the green "Press to mark completed" button.



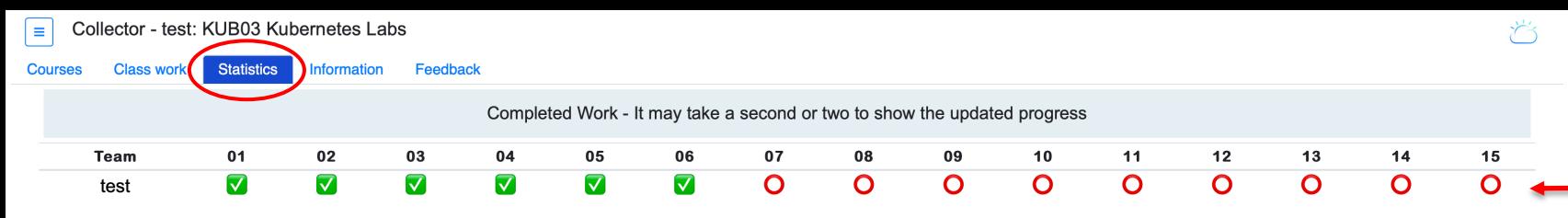
The Complete Button might not show instantly depending on the course settings

Collector – Track course completed work

Course title



The number of items tracked will change based on the current course selected.



Green checkmark - item is completed
Red circle - item is waiting to be completed

Collector – Instructor Dashboard

Remaining Time for the Lab

Collector - instructor: K8s_101_01 Kubernetes Introduction

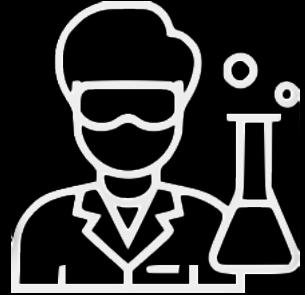
Remaining time: 0h 29m 50s

Courses Class work Statistics Information Feedback Insight

Completed Work - It may take a second or two to show the updated progress

Team	01	02	03	04	05	Cnt
instructor	✓	○	○	○	○	0
lightblue	✓	✓	✓	✓	✓	1
olive	✓	✓	○	○	○	2
peru	○	○	○	○	○	3
chocolate	○	○	○	○	○	4
pink	○	○	○	○	○	5
violet	○	○	○	○	○	6





JTC90 Lab Setup

EVERYBODY

Task 2: Setup Kubectl

OK

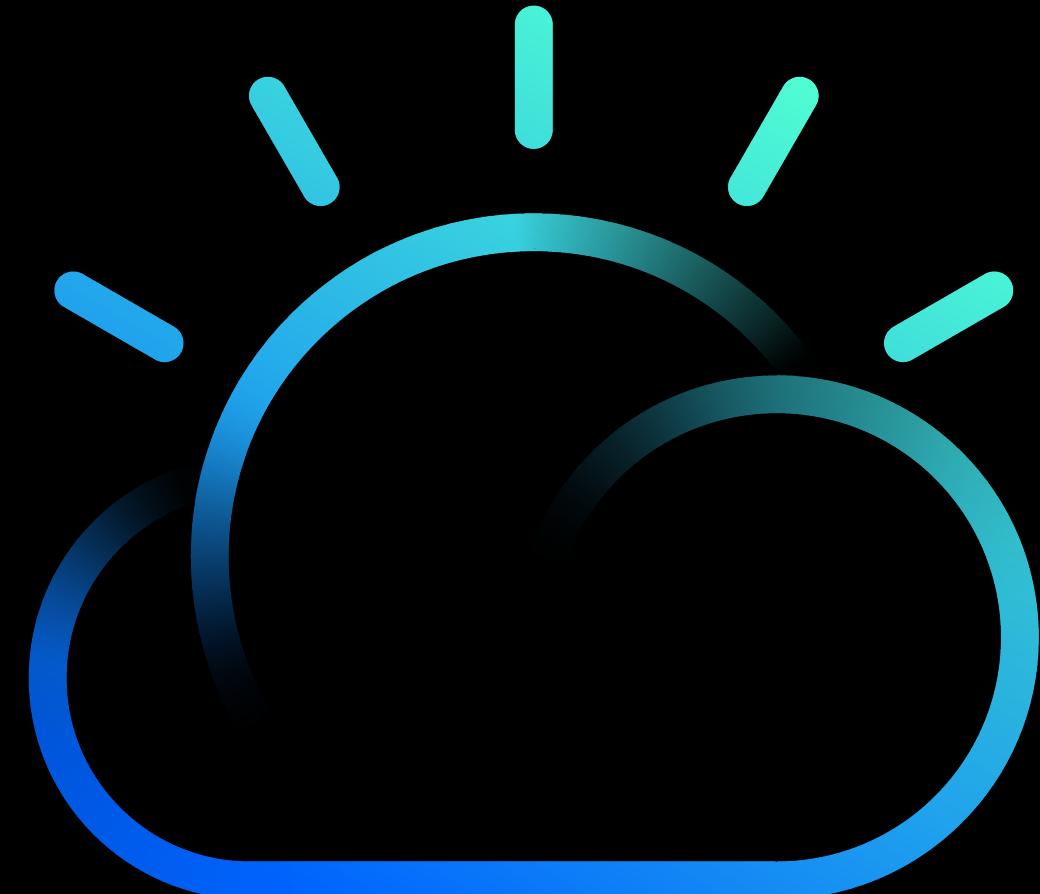
Task 3: Setup git

Task 4: Final Check

?

The Journey to Cloud **Microservices**

01



IBM Cloud

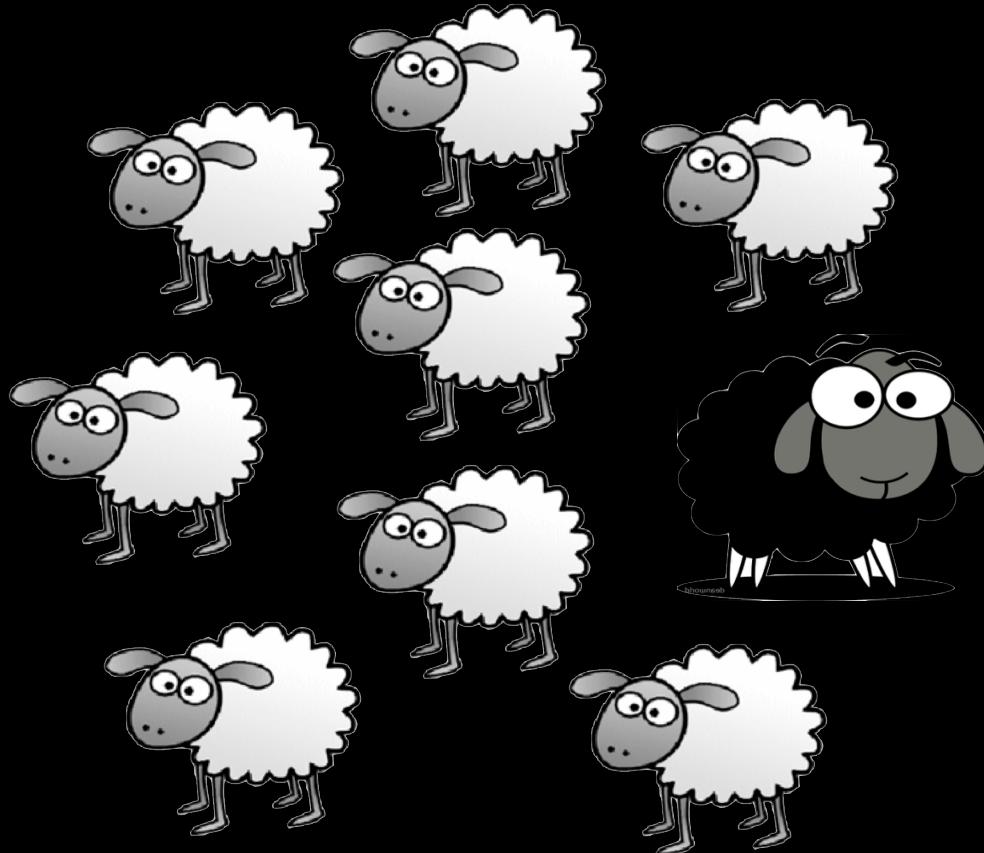


disruption

dɪs'ruptʃn/
noun

Business. a **radical change** in an industry, business strategy, etc., especially involving the introduction of a **new product or service** that creates a **new market**

Disruption is new reality



disruption

dɪs'ruptʃn/

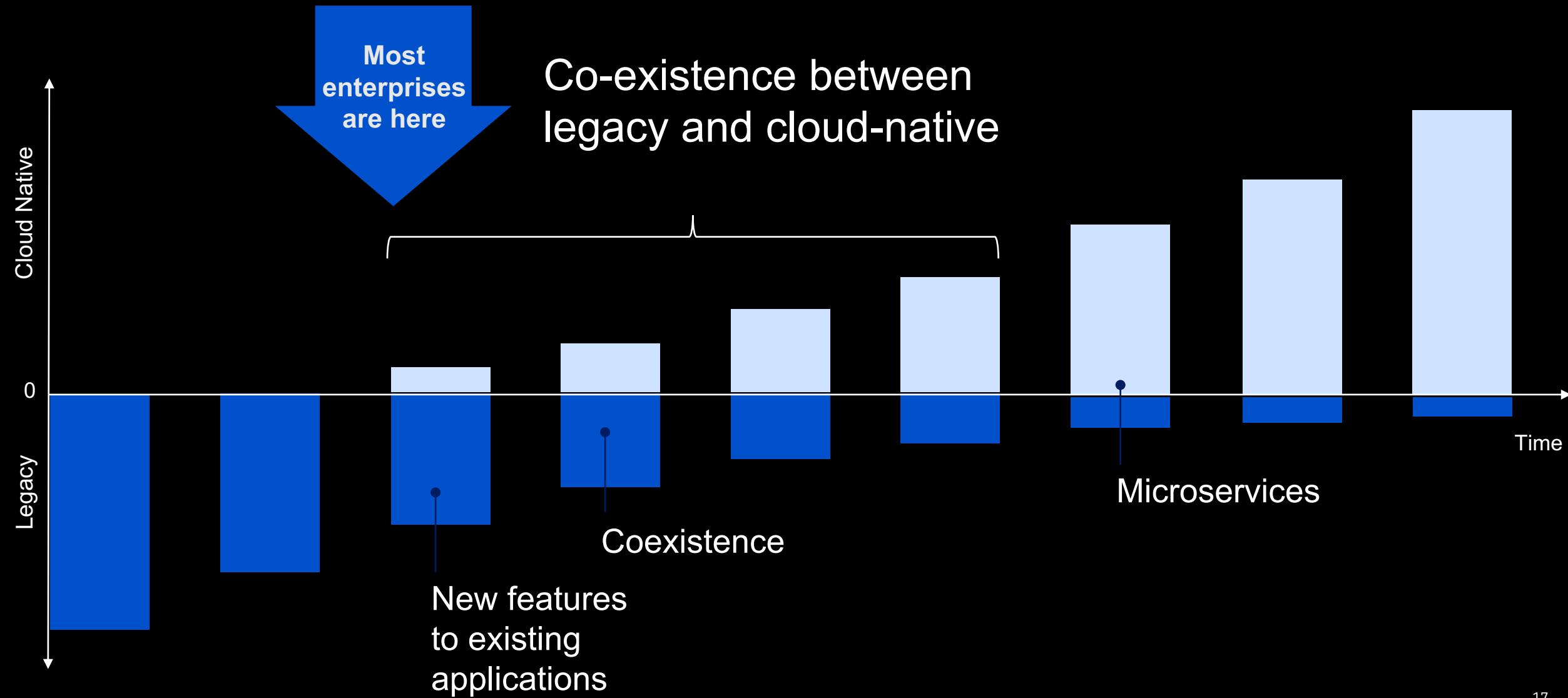
noun

Business. a **radical change** in an industry, business strategy, etc., especially involving the introduction of a **new product or service** that creates a **new market**

Digital Transformation is the new normal

Digital Transformation - Way to go

Cloud native and legacy apps will co-exist for the next 10+ years



How do you achieve digital transformation?

Adopt new Processes	Adopt new Technology	Adopt new Tools	Adopt Cloud
Continuous Integration	Architectural patterns (Microservices)	Git, Github, Gitlab	Docker, Kubernetes
Continuous Build	Frameworks (Java MicroProfile, Spring ...)	Jenkins	Virtualization, VMs
Continuous Deploy	Node.js , Swift	UrbanCode	Monitoring, Dashboards, Alerts
Agile Dev Models		Docker	
		Kubernetes	

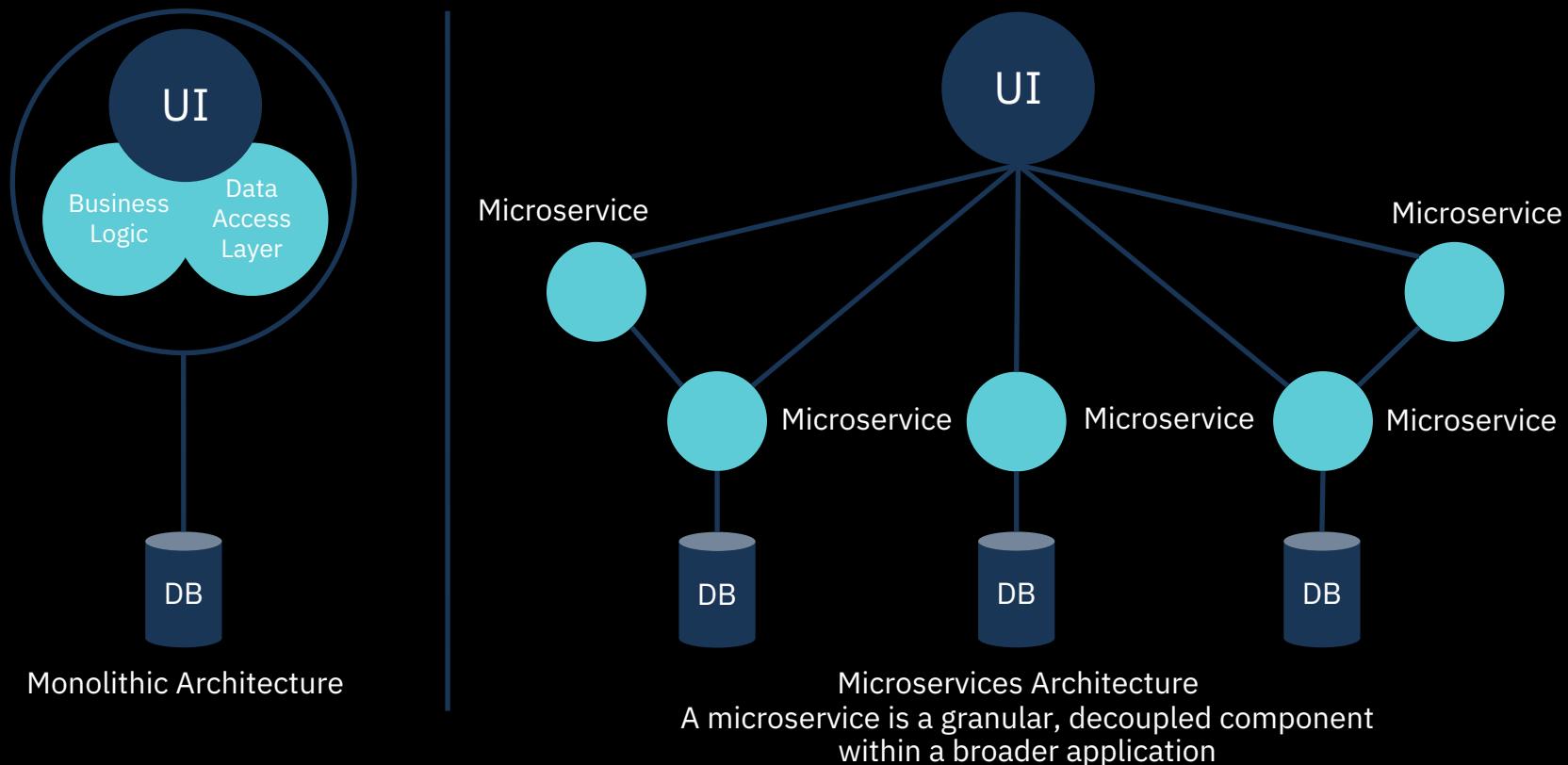
Microservices

Decomposing an application into **single function modules** which are **independently deployed and operated**

Accelerate delivery by minimizing communication and coordination between people

Microservices architecture

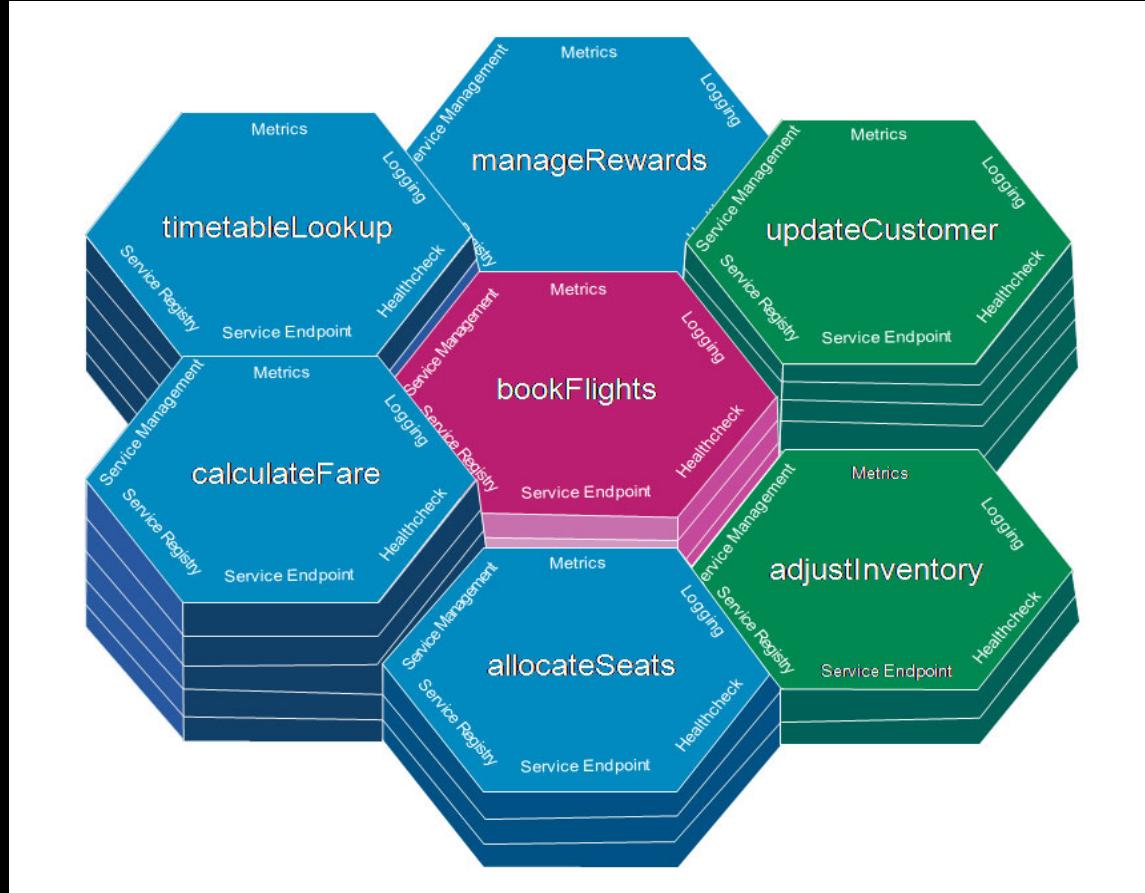
Simplistically, microservices architecture is about breaking down large silo applications into more manageable, fully decoupled pieces



Example application using microservices

Airline reservation application

- Book flights
- Timetable lookup
- Calculate fare
- Allocate seats
- Manage rewards
- Update customer
- Adjust inventory



Microservices – Key Tenets



Large monoliths are **broken down into many small services**

- Each service runs its own process
- There is one service per container



Services are **optimized for a single function**

- There is only one business function per service
- The Single-responsibility Principle: A microservice should have one, and only one, reason to change



Communication via **REST API** and **message brokers**

- Avoid tight coupling introduced by communication through a database



Per-service continuous delivery (CI/CD)

- Services evolve at different rates
- Let the system evolve, but set architectural principles to guide that evolution



Per-service high availability and clustering decisions

- One size or scaling policy is not appropriate for all
- Not all services need to scale; others require autoscaling up to large numbers

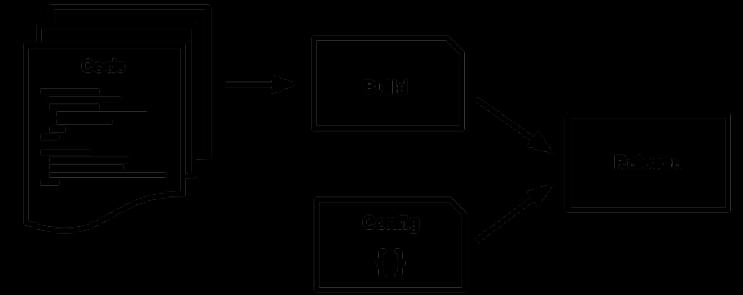
Microservices – Advantages

In a microservices architecture each component:

- Is **developed independently** and has **limited, explicit dependencies** on other services
- Is developed by a **single, small team** in which all team members can understand the entire code base
- Is developed on its **own timetable** so new versions are delivered independently of other services
- **Scales and fails independently** which isolates any problems
- Can be developed in a different **language**
- **Manages its own data** to select the best technology and schema

The 12 Factor App

1. Codebase - One codebase tracked in revision control, many deploys
2. Dependencies - Explicitly declare and isolate dependencies
3. Config - Store config in the environment
4. Backing services - Treat backing services as attached resources
5. Build, release, run - Strictly separate build and run stages
6. Processes - Execute the app as one or more stateless processes
7. Port binding - Export services via port binding
8. Concurrency - Scale out via the process model
9. Disposability - Maximize robustness with fast startup and graceful shutdown
10. Dev/prod parity - Keep development, staging, and production as similar as possible
11. Logs - Treat logs as event streams
12. Admin processes - Run admin/management tasks as one-off processes



Not a methodology to do cloud-native design!

<https://12factor.net/>

7

missing factors

13. Observable

Apps should provide visibility about current health and metrics

14. Schedulable

Apps should provide guidance on expected resource constraints

15. Upgradable

Apps must upgrade data formats from prior generations

16. Least privileged

Apps should provide guidance on expected resource constraints

17. Auditable

Apps should provide appropriate audit logs for compliance needs

18. Access Control (Identity, Network, Scope, Certificates)

Protect app and resources from the world

19. Measurable

Apps usage should be measurable for quota or chargebacks

That said....

Microservices

are
hard

Microservice transformations – The good, the bad, the ugly

The good

IF implemented correctly Microservices improve scalability and reliability.

- Scale easily and very efficiently
- No single point of failure by isolation
- Be modified without compromising the whole system
- Use of languages and frameworks that best suit the purpose of each service
- Natively ready for CaaS and PaaS

Microservice transformations – The good, the bad, the ugly

The bad

- Added management **complexity**
- **Expertise** required to maintain a microservice-based application
- No or **reduced benefits** if application doesn't need scaling or cloud transformation
- **No greenfield** options because microservices need to connect to existing systems
- Needs more **robust testing** in order to cover all APIs

Microservice transformations – The good, the bad, the ugly

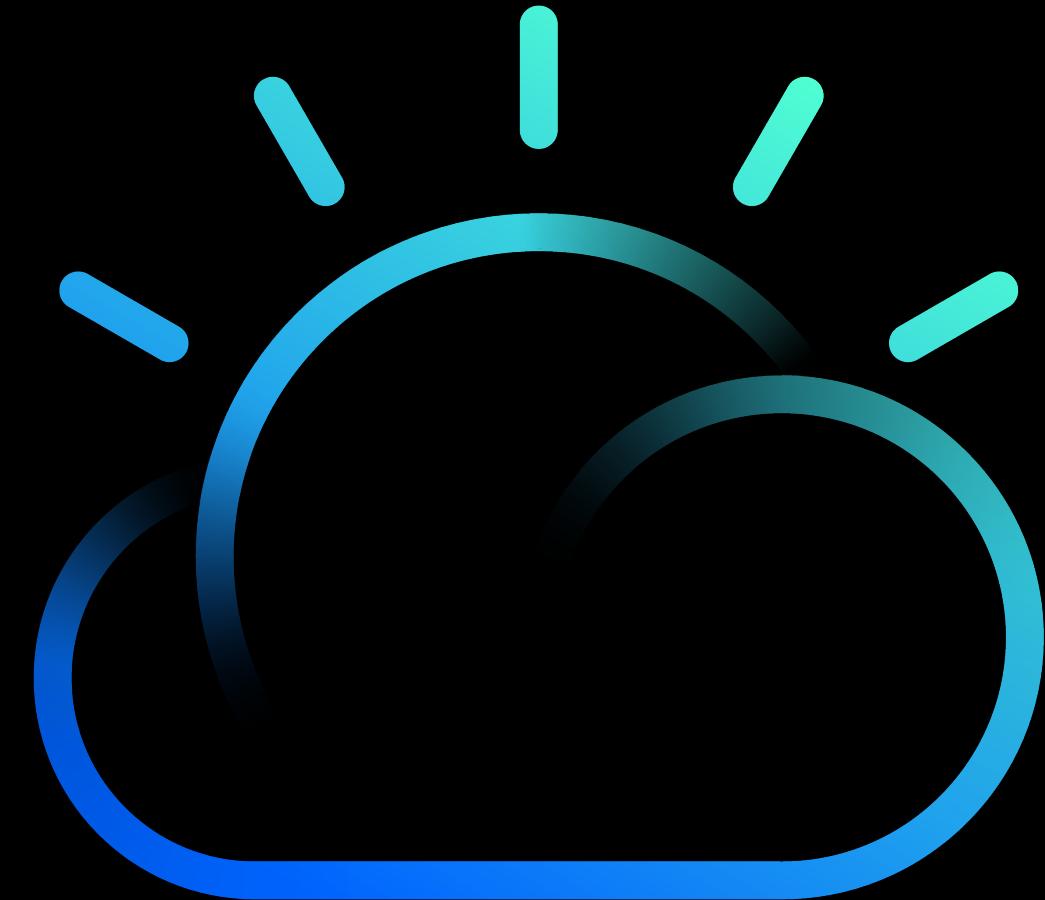
... and **the downright ugly**

- May require **high initial investment** to run for Dev, deployment and Ops overhead
- Higher **operational and monitoring costs** through shift in paradigms
- **Non-uniform** application design and architecture through free choice of technologies
- **Overload of documentation** for every individual component app

Microservices, when
implemented incorrectly,
can make poorly written
applications even more
dysfunctional

..but we're
getting ahead
of ourselves

QUESTIONS?



IBM Cloud

The Journey to Cloud **Docker**

02



IBM Cloud

Everybody Loves Containers



A **standard way to package an application and all its dependencies** so that it can be moved between environments and run without changes

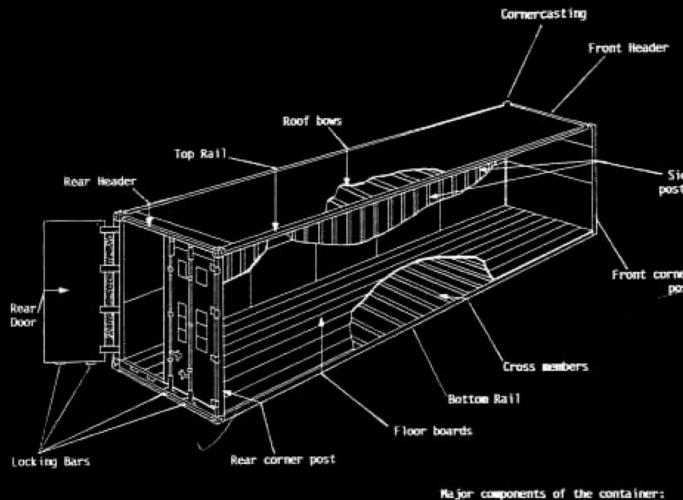
Containers work by **isolating the differences between applications** inside the container so that everything outside the container can be standardized

Microservices implementation with Containers

Why it works – separation of concerns

Development

- Worries about what's “**inside**” the container
 - Code
 - Libraries
 - Package Manager
 - Apps
 - Data
- All Linux servers look the same

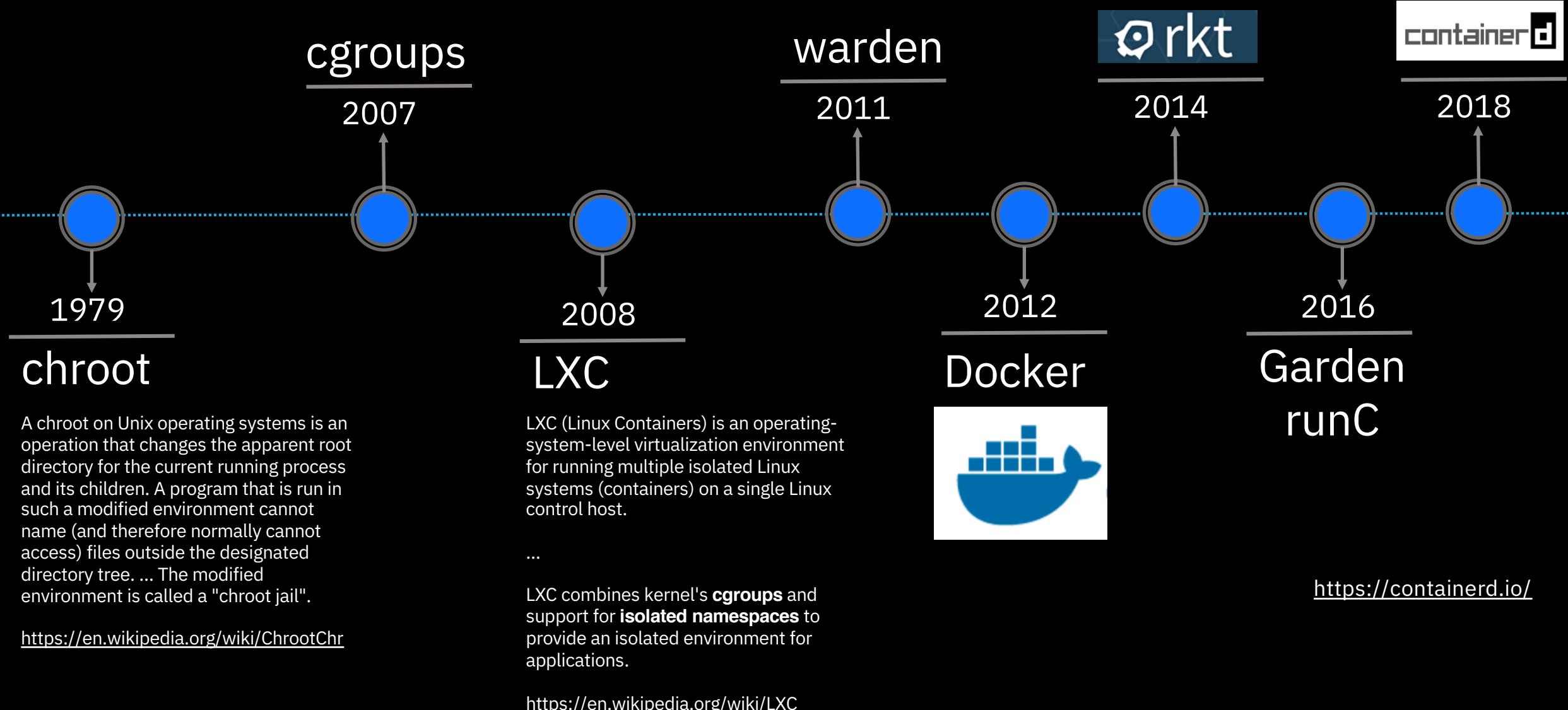


Operations

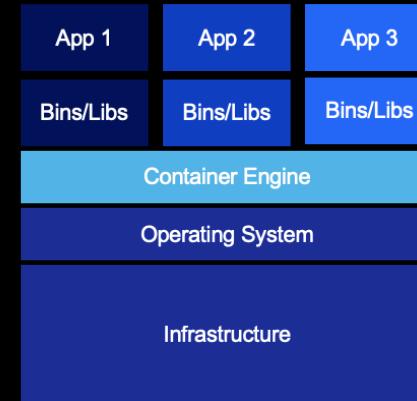
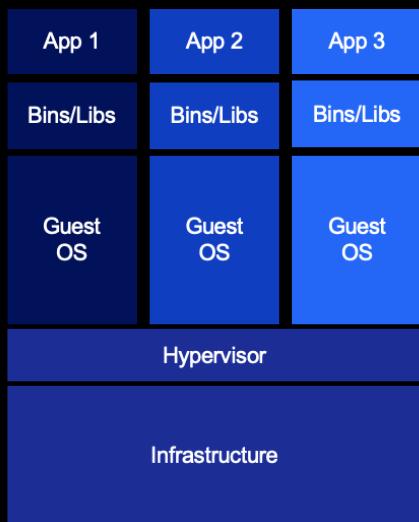
- Worries about what's “**outside**” the container
 - Logging
 - Remote Access
 - Monitoring
 - Network Config
- All containers start, stop, copy, attach, migrate, etc... the same way

Clear ownership boundary between
Dev and IT Ops drives DevOps adoption and fosters agility

Container History



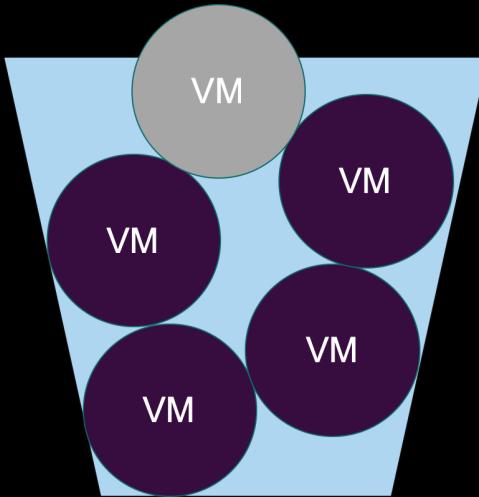
VMs vs. Containers



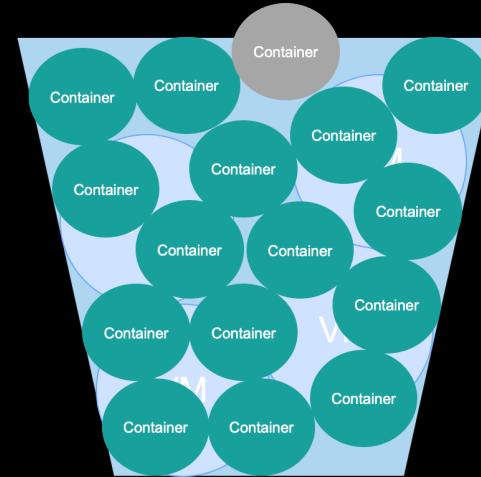
- + VM Isolation
- Complete OS
- Static Compute
- Static Memory

- + Container Isolation
- + Shared Kernel
- + Burstable Compute
- + Burstable Memory

VMs vs. Containers



- + VM Isolation
- Complete OS
- Static Compute
- Static Memory
- Low Resource Utilization



- + Container Isolation
- + Shared Kernel
- + Burstable Compute
- + Burstable Memory
- + High Resource Utilization

Advantages of Containers



Containers are **portable**



Containers are **easy to manage**



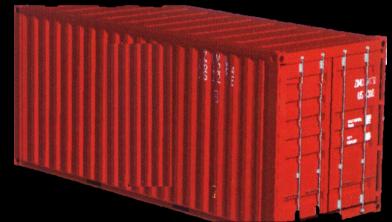
Containers provide “**just enough**” isolation



Containers use hardware **more efficiently**



Containers are **immutable**



Docker Components

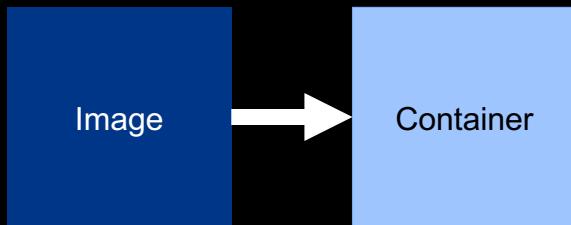
Container

Smallest compute unit



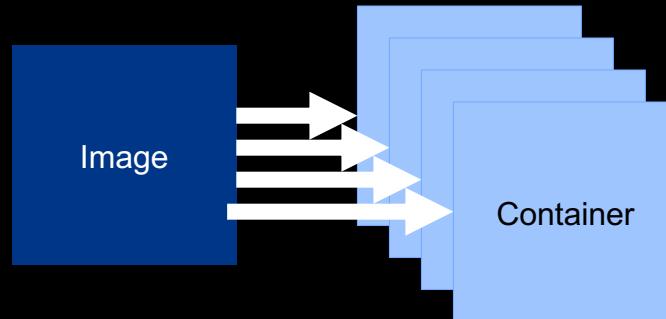
Docker Components

Containers
are created from
Images



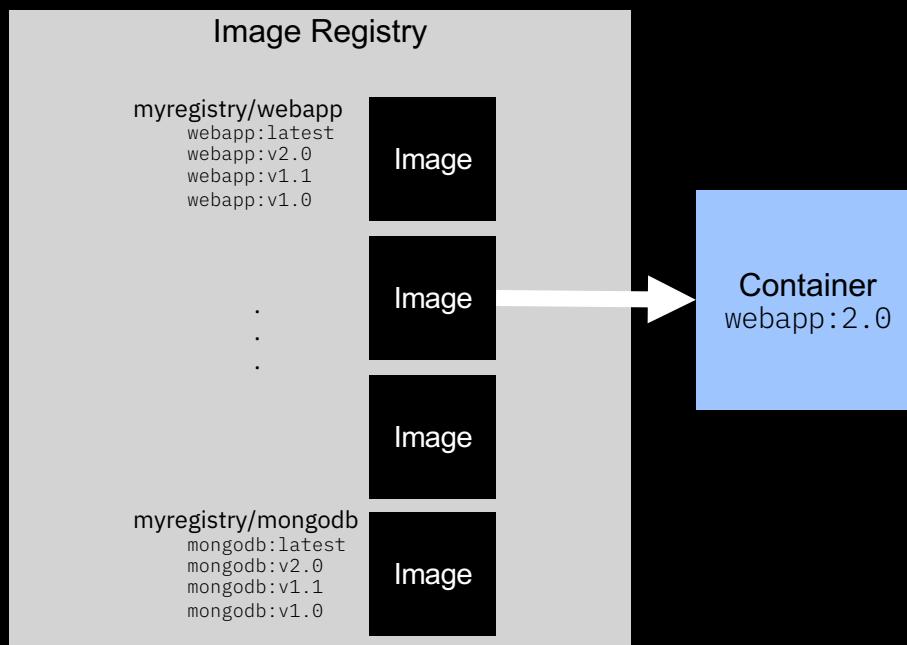
Docker Components

As **many Containers**
as needed can be created from
Images



Docker Components

The **Image Registry**
stores the versioned
Images
to create
Containers



Docker Components



Image

A read-only snapshot of a container stored in Docker Hub to be used as a template for building containers



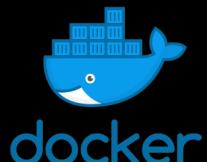
Container

The standard unit in which the application service resides or transported



Docker Hub/Registry

Available in SaaS or Enterprise to deploy anywhere you choose
Stores, distributes, and shares container images



Docker Engine

A program that creates, ships, and runs application containers
Runs on any physical and virtual machine or server locally, in private or public cloud. Client communicates with Engine to execute commands

Why Containers

Docker Layers

Inheritance

- Build on the work of those who came before you

```
# Pull base image
FROM tomcat:8-jre8

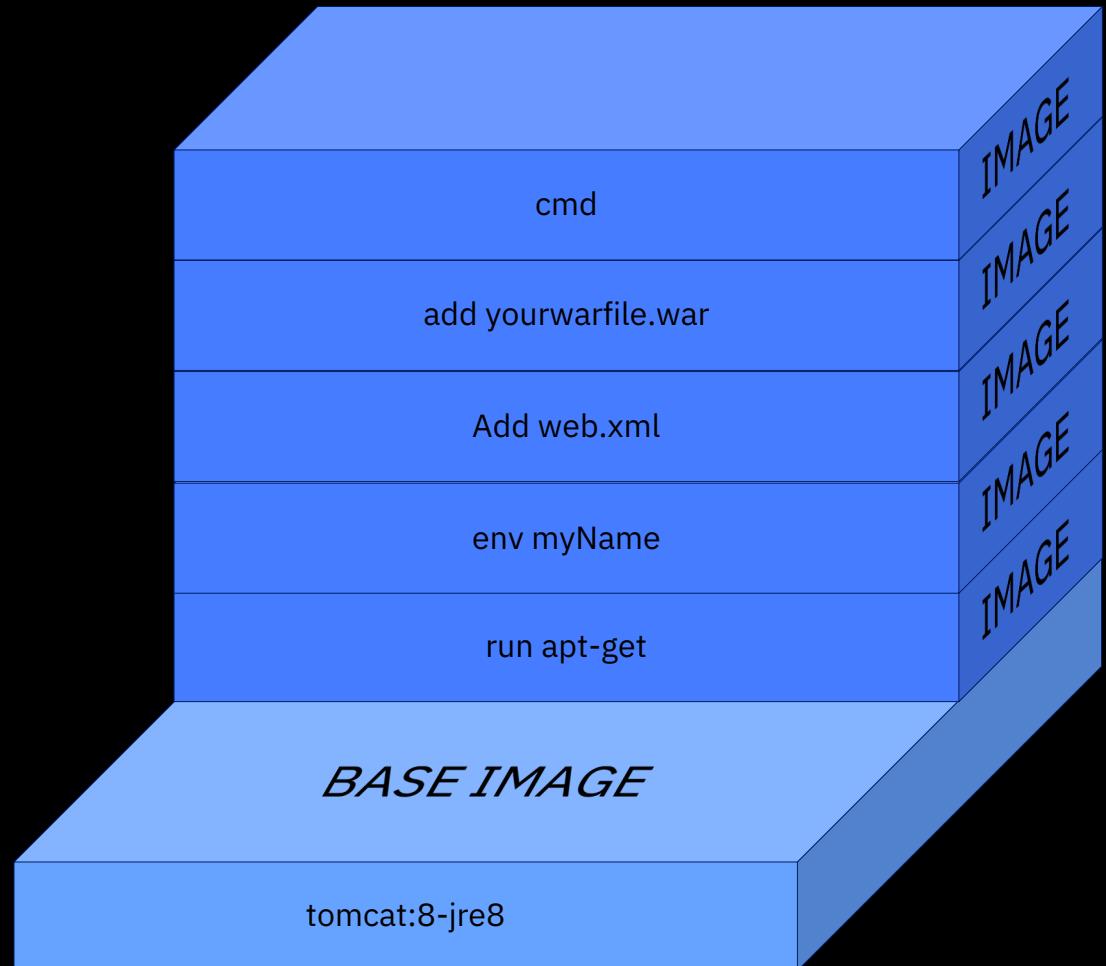
# Maintainer
MAINTAINER youremailaddress"

# Run command
RUN apt-get update && apt-get -y upgrade

# Set variables
ENV myName John Doe

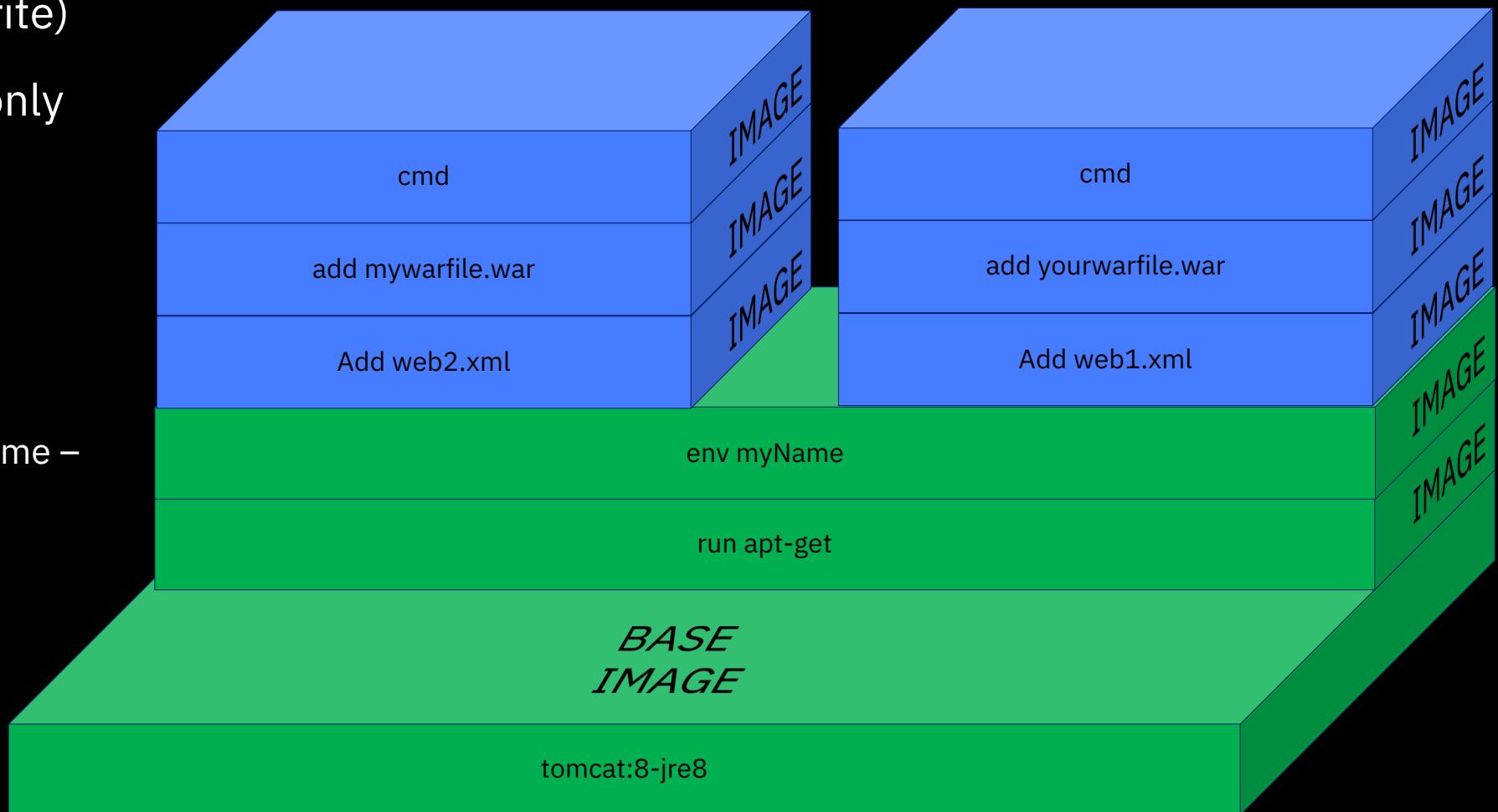
# Copy to images tomcat path
ADD web.xml /usr/local/tomcat/conf/
ADD yourwarfile.war /usr/local/tomcat/webapps/

# Run server
CMD ["catalina.sh", "run"]
```

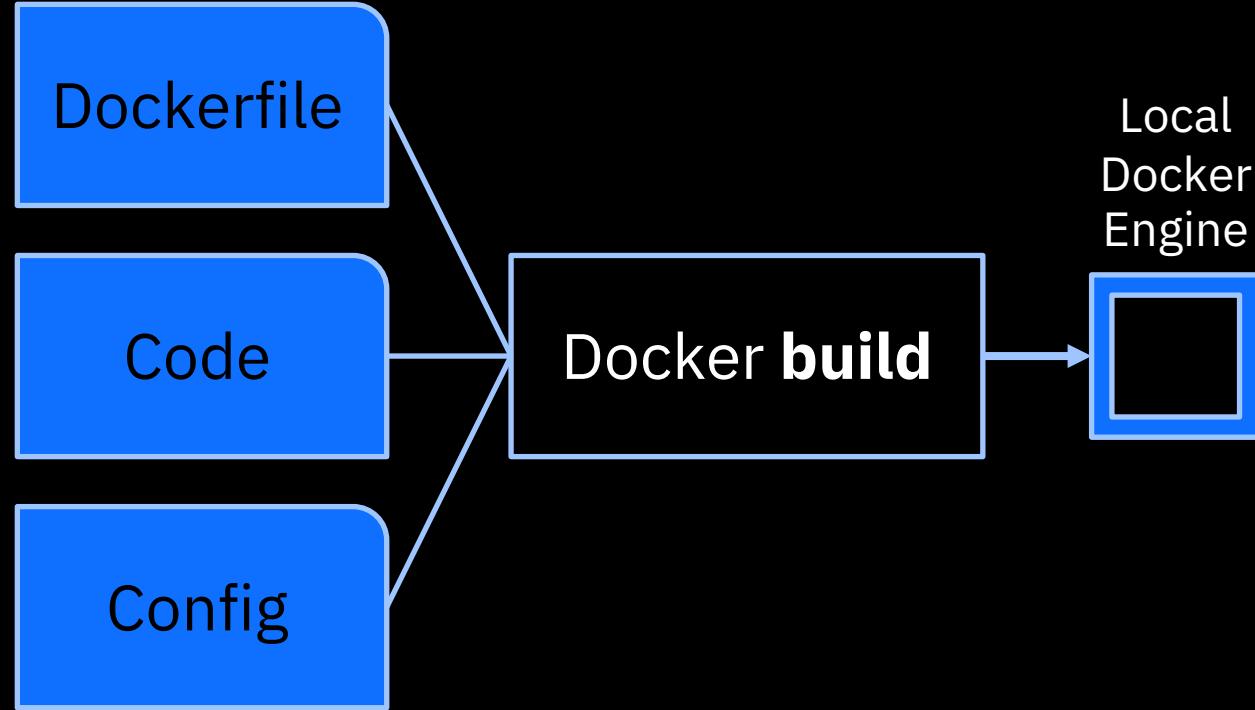


Why Containers

- Docker uses a **layered filesystem** (copy-on-write)
- New files (& edits) are only visible to current/above layers
- Layers allow for **reuse**
 - More containers per host
 - Faster start-up/download time – base layers are "cached"



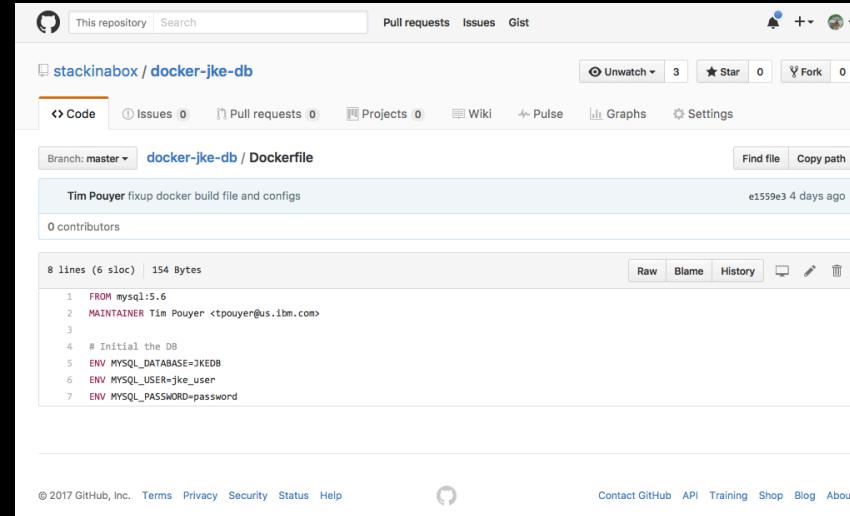
Docker Basics – Build



Docker Basics - Build

Dockerfile

- Text based file describing:
 - Previous Layer
 - Environment Variables
 - Commands used to populate data/software/frameworks/etc...
 - Command to run when executed



The screenshot shows a GitHub repository page for 'stackinabox / docker-jke-db'. The 'Dockerfile' tab is selected, displaying the following code:

```
FROM mysql:5.6
MAINTAINER Tim Poyer <tpouyer@us.ibm.com>
# Initial the DB
ENV MYSQL_DATABASE=jKEdb
ENV MYSQL_USER=jke_user
ENV MYSQL_PASSWORD=password
```

```
# Pull base image
FROM tomcat:8-jre8

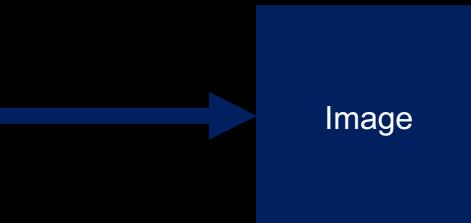
# Maintainer
MAINTAINER "youremailaddress"

# Run command
RUN apt-get update && apt-get -y upgrade

# Set variables
ENV myName John Doe

# Copy to images tomcat path
ADD web.xml /usr/local/tomcat/conf/
ADD yourwarfile.war /usr/local/tomcat/webapps/

# Run server
CMD ["catalina.sh", "run"]
```



Docker Basics – Build -Dockerfile

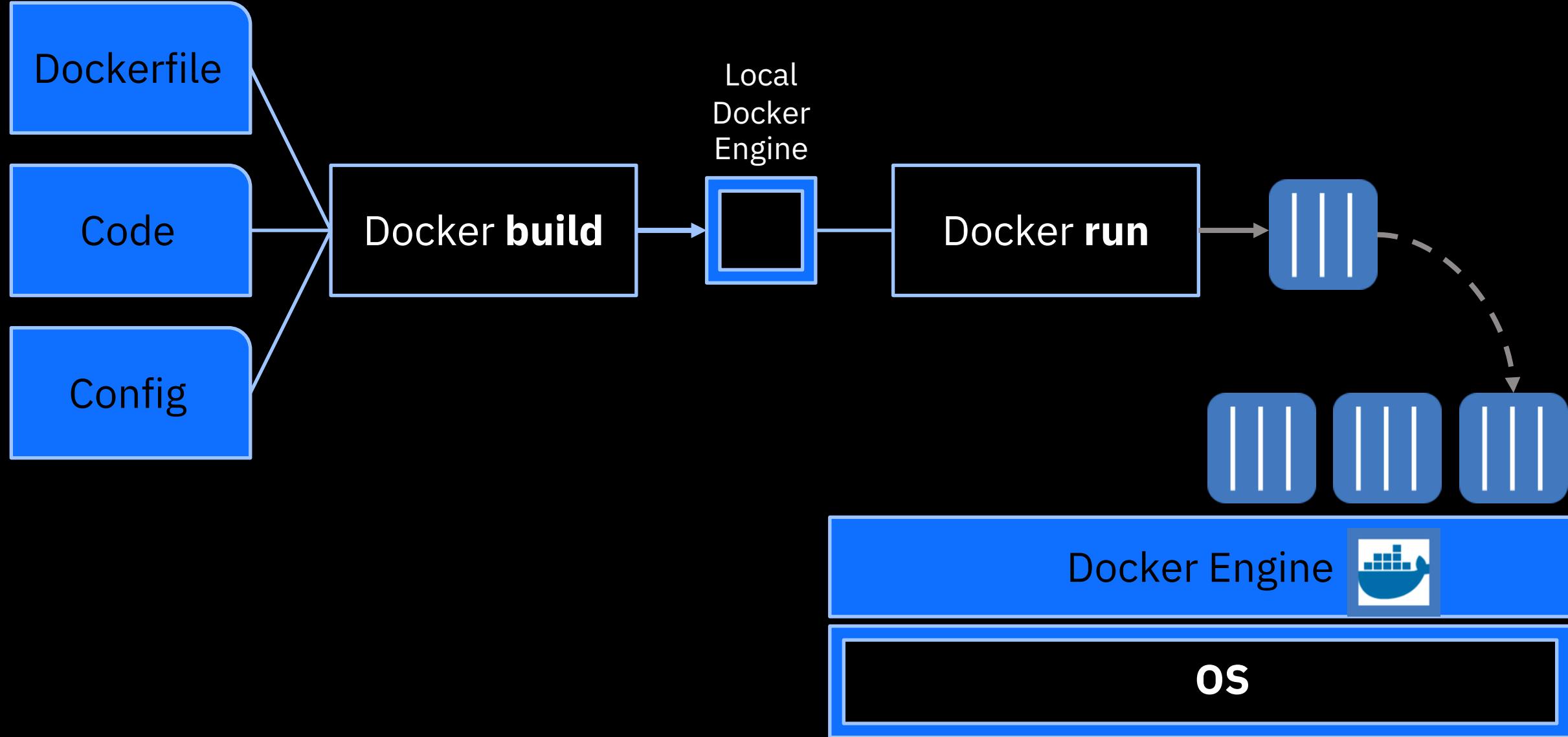
one process per container

- A text file that builds an image using Docker directives

- FROM # Pull base image
FROM tomcat:8-jre8
- RUN # Maintainer
MAINTAINER "youremailaddress"
- COPY # Run command
RUN apt-get update && apt-get -y upgrade
- ENTRYPOINT # Set variables
ENV myName John Doe
- LABEL # Copy to images tomcat path
ADD web.xml /usr/local/tomcat/conf/
ADD yourwarfile.war /usr/local/tomcat/webapps/
- EXPOSE # Run server
CMD ["catalina.sh", "run"]

```
docker build -t myimage ./Dockerfile
```

Docker Basics – Run

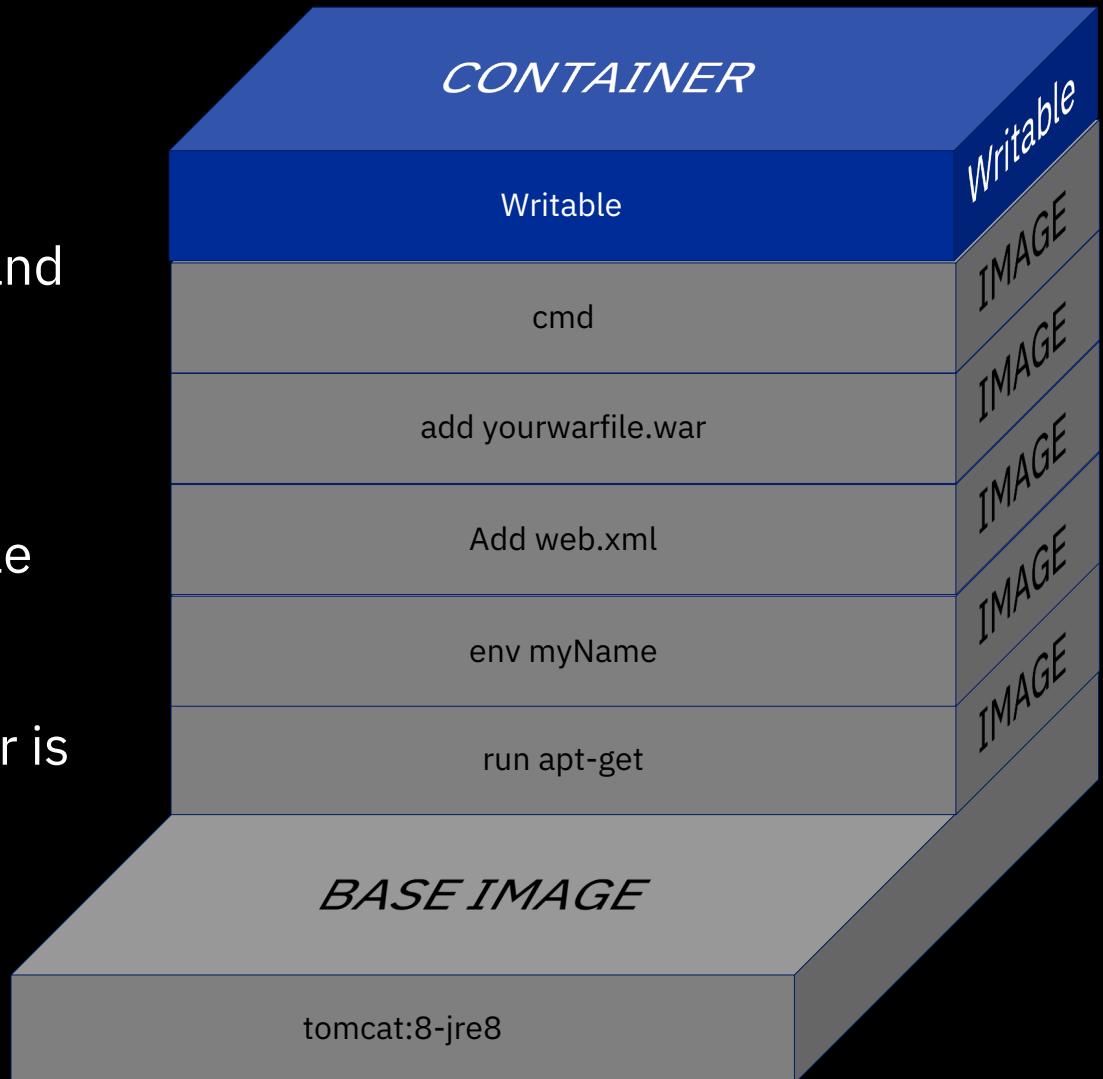


Docker Basics – Run

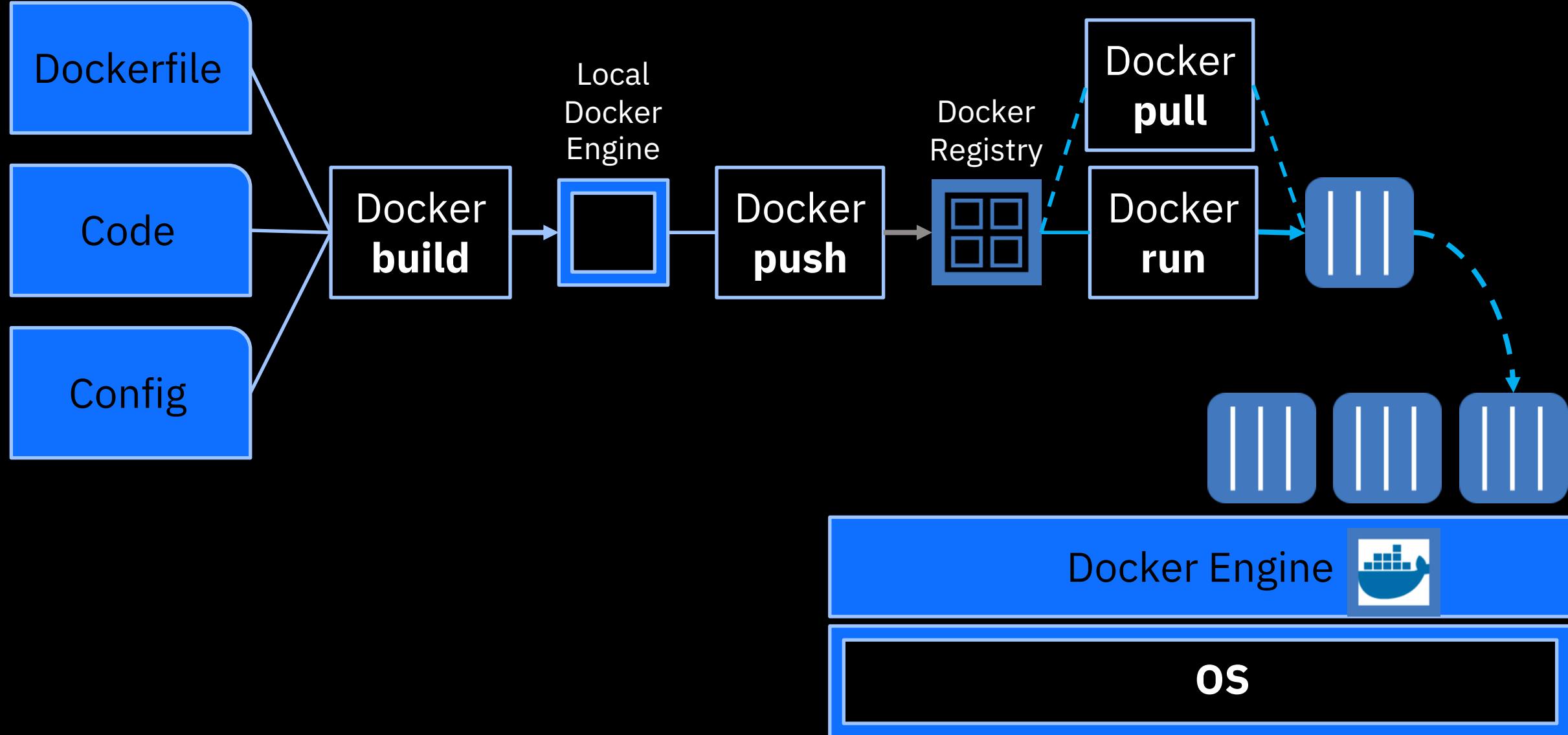
Docker Layers

Inheritance

- The biggest difference between a **container** and an **image** is the **top writable layer**.
- All writes to the container that add new or modify existing data are stored in this writable layer.
- When a container is deleted, its writable layer is also deleted. The underlying image is unchanged.



Docker Basics – Store, Retrieve & Run with registry



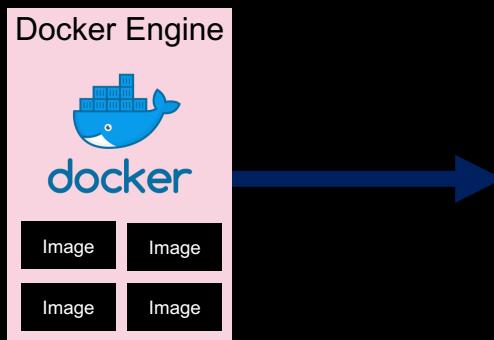
Docker Basics – Store & Retrieve

Docker Registry

- Private Local
- Public Docker Hub
- Private Shared

docker images

```
tpoyer at Laptop in ~/Development/workspace/docker/docker-jke-db on master
$ docker images
REPOSITORY          TAG      IMAGE ID      CREATED       SIZE
stackinabox/demo-docker   dev      4e2a1f6cc1a4    25 hours ago  528.3 MB
stackinabox/demo-jke     1.1      a596fbf2c3b7    3 days ago   672 MB
stackinabox/demo-jke     dev      a596fbf2c3b7    3 days ago   672 MB
stackinabox/demo-jke     latest   a596fbf2c3b7    3 days ago   672 MB
stackinabox/demo-jke     1.1      a596fbf2c3b7    3 days ago   672 MB
stackinabox/demo-jke     latest   a596fbf2c3b7    3 days ago   672 MB
stackinabox/jke-web      latest   a4be0cdad8bc    4 days ago   462.3 MB
stackinabox/demo-jke-db   dev      54bdbf42b999    4 days ago   327.5 MB
stackinabox/jke-db        ddc     54bdbf42b999    4 days ago   327.5 MB
stackinabox/urbanbane-deploy-client  6.2.2.0  53c878fbf034    7 days ago   477 MB
websphere-liberty        crosservices: 7e27c3fb9c96   10 days ago  445.7 MB
mysql                   architecture 5.6      a896fd82dcd5   13 days ago  327.5 MB
mysql                   principles   latest   f3694c67abdb   13 days ago  400.1 MB
[...]
$ ls
[...]
Line 31, Column 17
Tab size: 4
```



myregistry/webapp

webapp:latest
webapp:v2.0
webapp:v1.1
webapp:v1.0

myregistry/mongodb

mongodb:latest
mongodb:v2.0
mongodb:v1.1
mongodb:v1.0

Docker Basics – Run

Docker run

- Local (containerd)

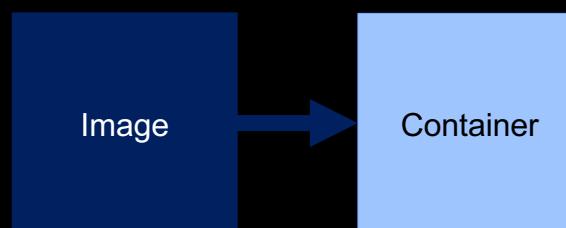
```
tpouyer at Laptop in ~/Development/workspace/docker/docker-jke-db on master
$ docker run -d postgres:latest
4eec29ae5eaa20798b1af8fa37873f7fc20cbb7cb789986b44f66cd94a784e0a

tpouyer at Laptop in ~/Development/workspace/docker/docker-jke-db on master
$ docker ps
CONTAINER ID        IMAGE               COMMAND      CREATED             STATUS              PORTS
ORTS                NAMES
4eec29ae5eaa        postgres:latest     "/docker-entrypoint.s"   5 seconds ago    Up 4 seconds          5432/tcp

tpouyer at Laptop in ~/Development/workspace/docker/docker-jke-db on master
$ 
```

```
docker run -d -e MYVAR=foo myimage:1.0.0
docker run -ti myimage:1.0.0 /bin/bash
```

...



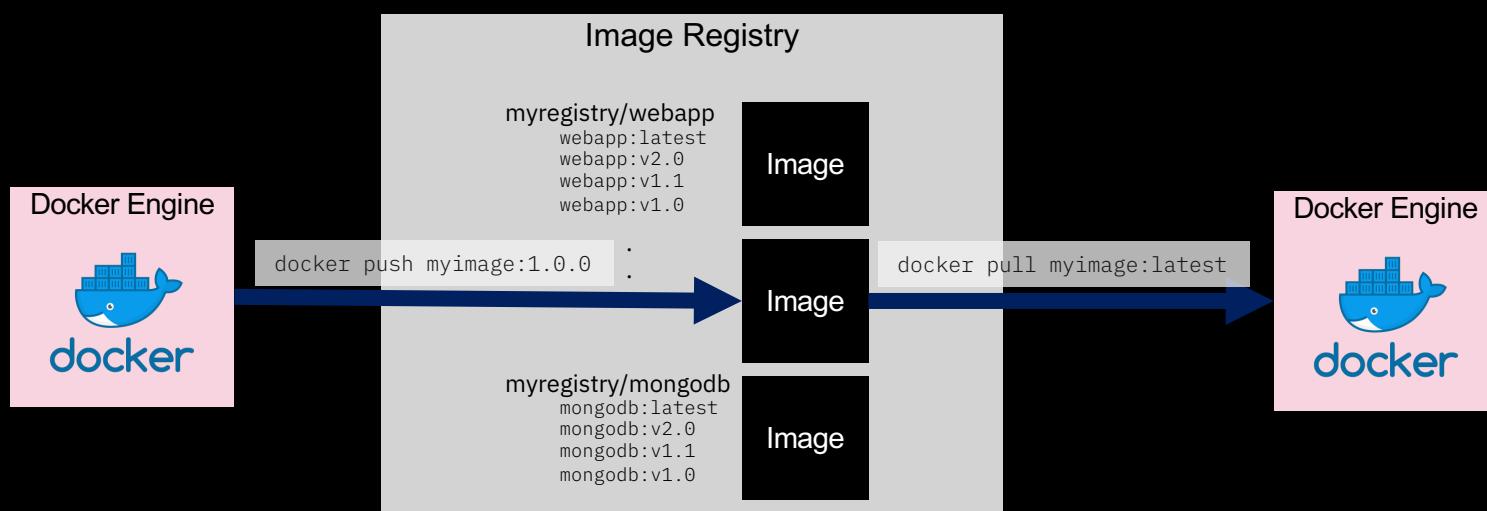
Docker Basics – Store & Retrieve with registry

Docker Registry

- Private Local
- Public Docker Hub
- Private Shared

```
docker pull myimage:latest  
docker push myimage:1.0.0
```

```
tpouyer at Laptop in ~/Development/workspace/docker/docker-jke-db on master  
$ docker pull postgres:latest  
latest: Pulling from library/postgres  
...  
Digest: sha256:0842a7ef786aa2658623085160cb38451eb3d40856e7d222ae0069b6e6296877  
Status: Downloaded newer image for postgres:latest  
  
tpouyer at Laptop in ~/Development/workspace/docker/docker-jke-db on master  
$
```



Docker Basics – Registries

Hosting image repositories

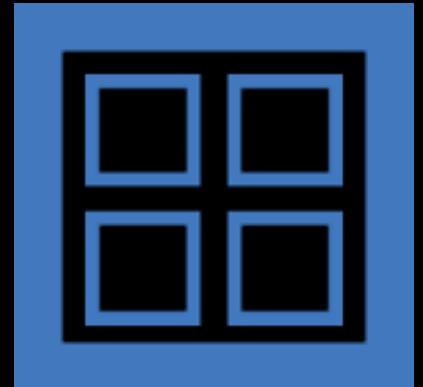
- You can define your own registry
- A registry is managed by a registry container

Public and Private registries

- Public Registry like **Docker Hub**
- <https://hub.docker.com>

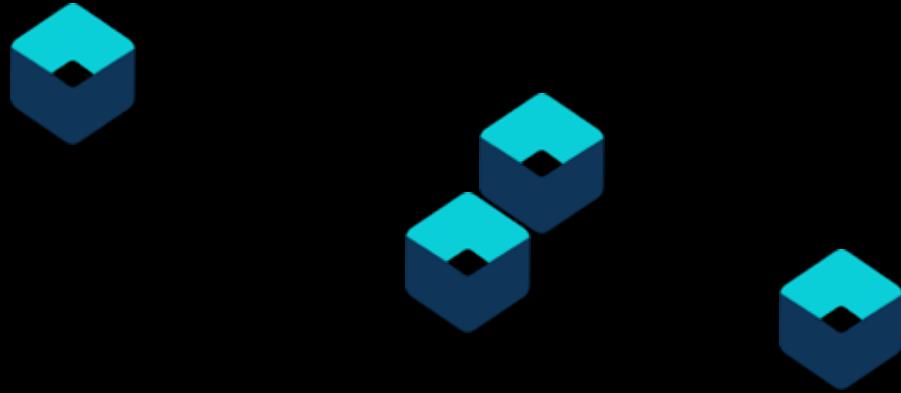
Login into the registry

- Docker login domain:port





Everyone's container journey starts with one container....



At first the growth is easy to handle....

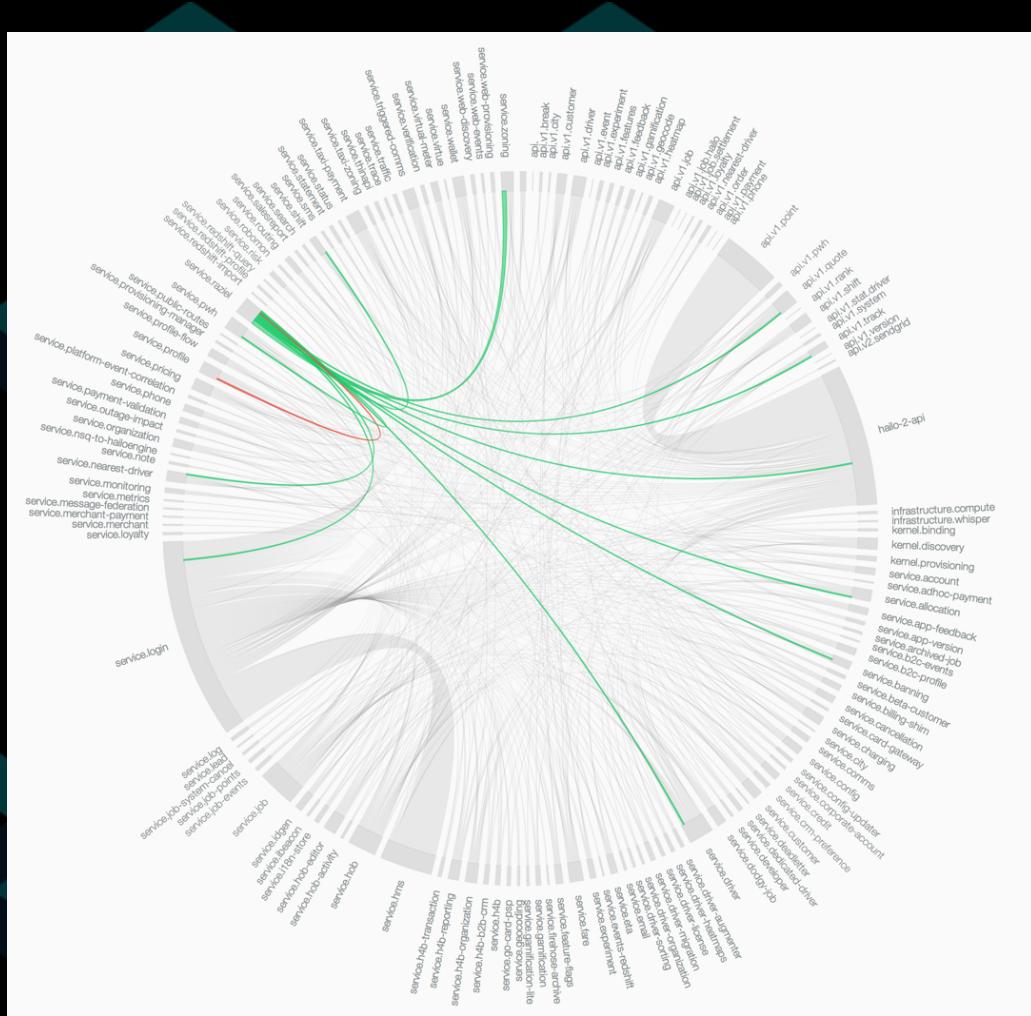


Pets vs Cattle

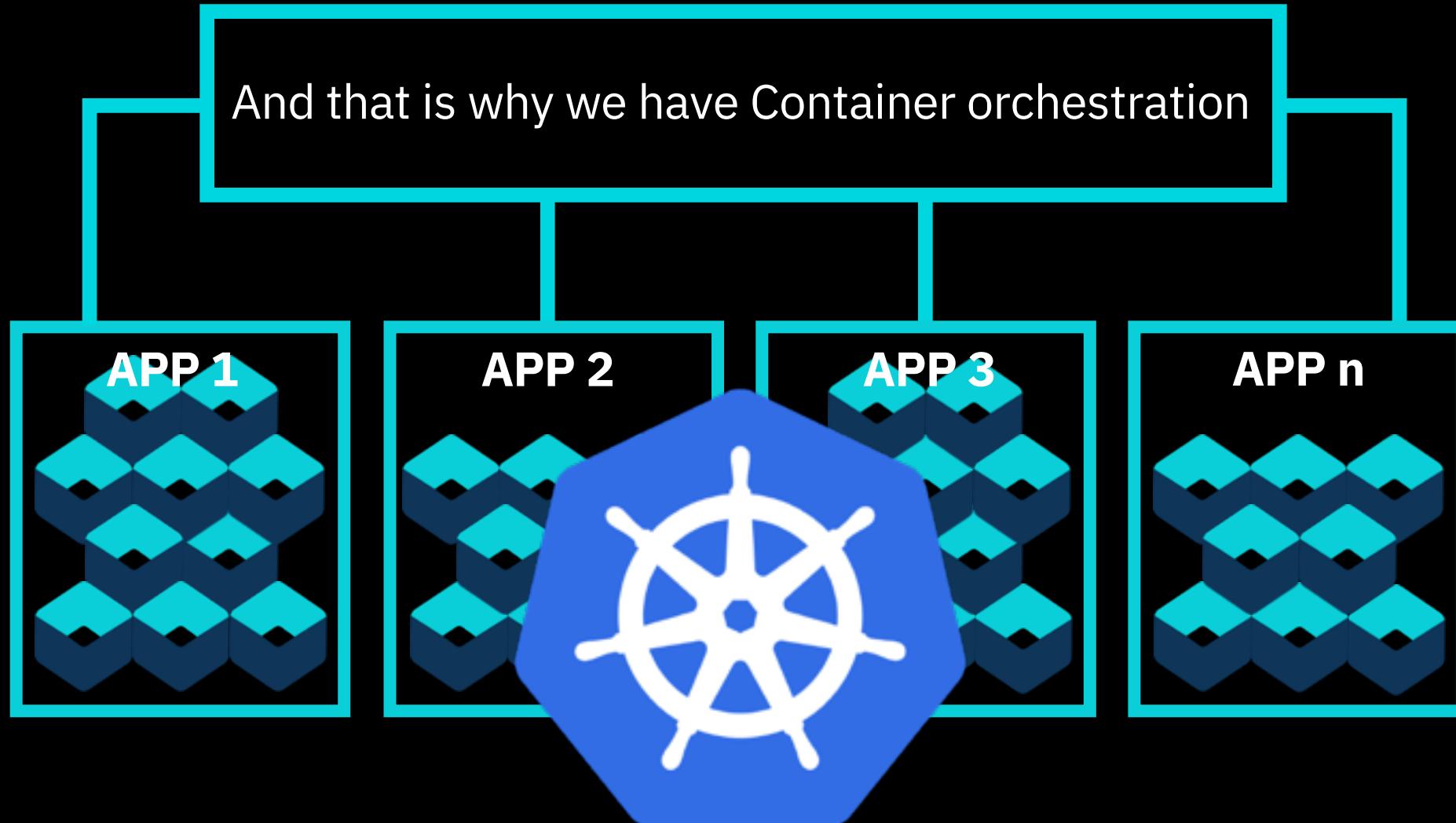
But soon you have many applications, many instances...

The trade off

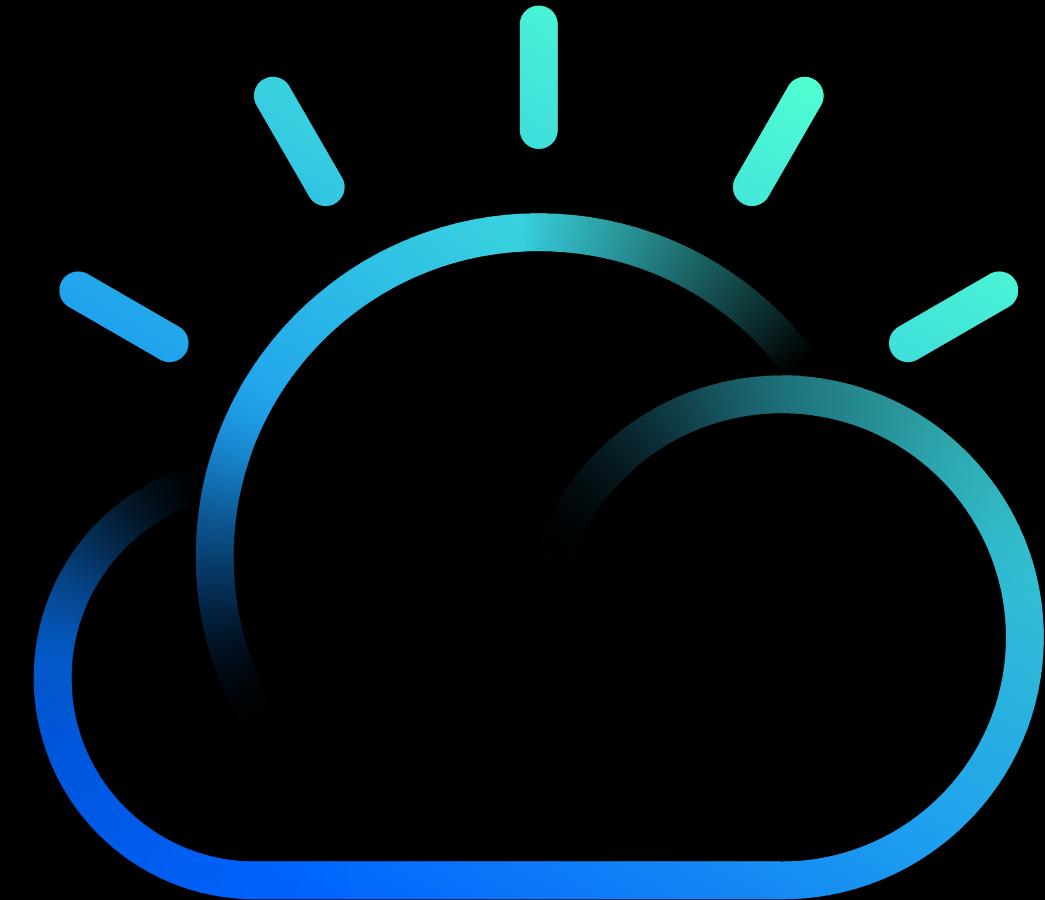
Improved delivery velocity
in exchange for
increased operational complexity



Enter Kubernetes (K8s)



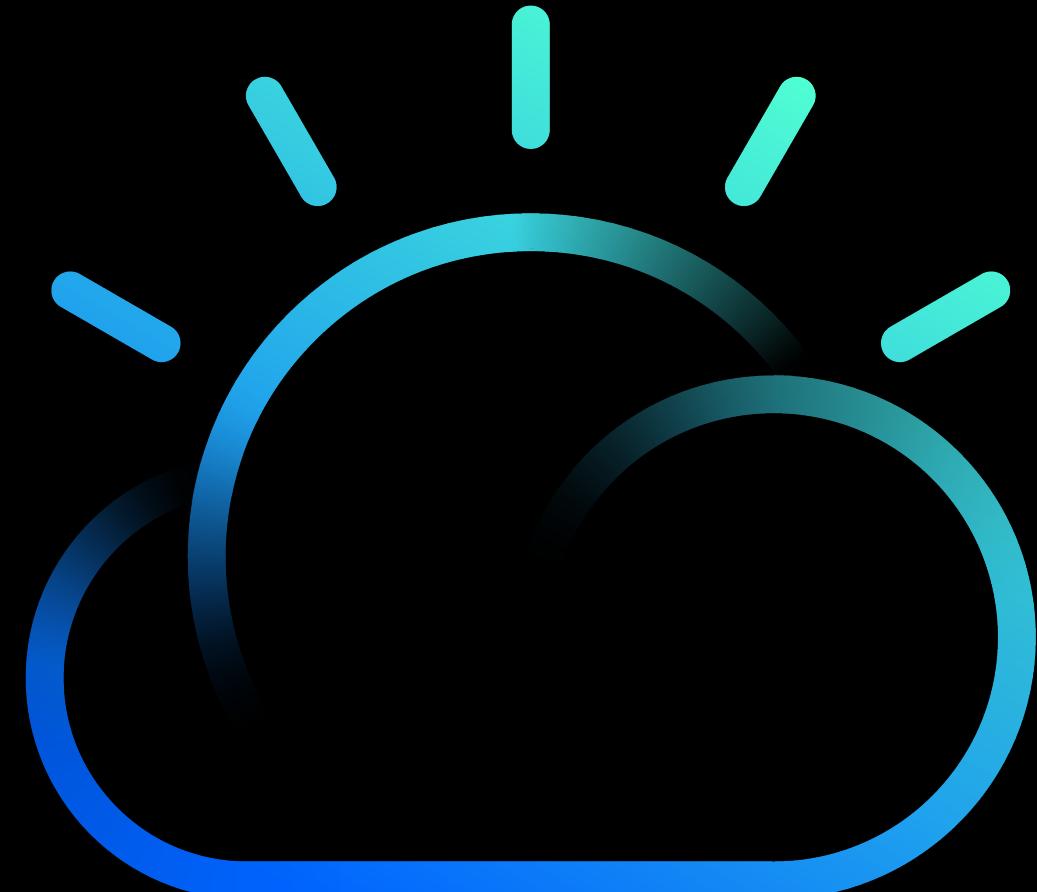
QUESTIONS?



IBM Cloud

The Journey to Cloud **Kubernetes**

03



IBM Cloud

Kubernetes – Declarative System



Imperative Systems

In an imperative system, **the user** knows the desired state and **the user** determines the sequence of commands to transition the system to the desired state.

Declarative Systems

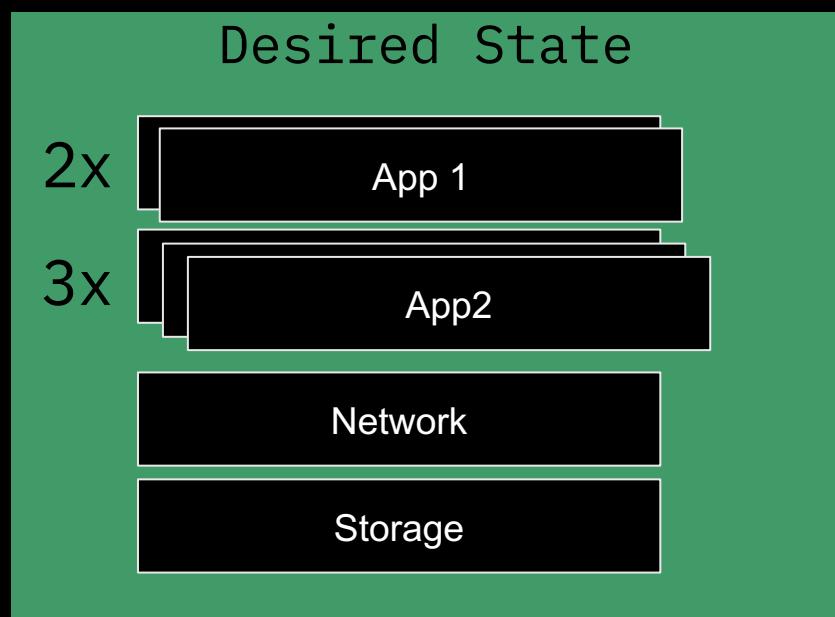
By contrast, in a declarative system, **the user** knows the desired state, supplies a representation of the desired state to the system, then **the system** reads the current state and determines the sequence of commands to transition the system to the desired state.

Kubernetes – Declarative System

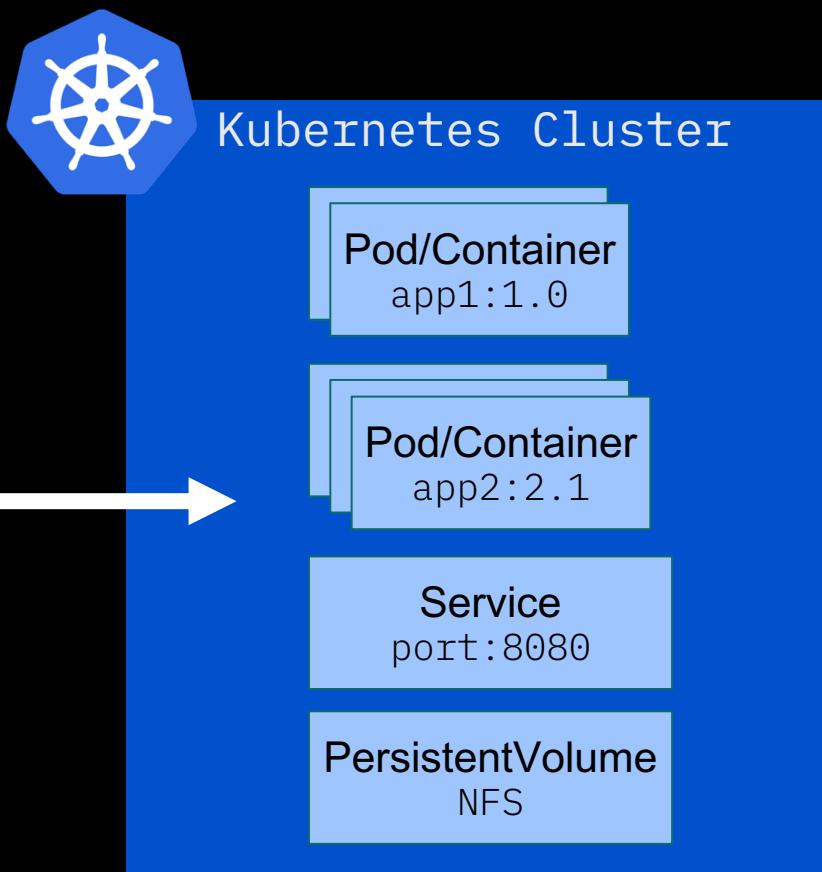


The Desired State

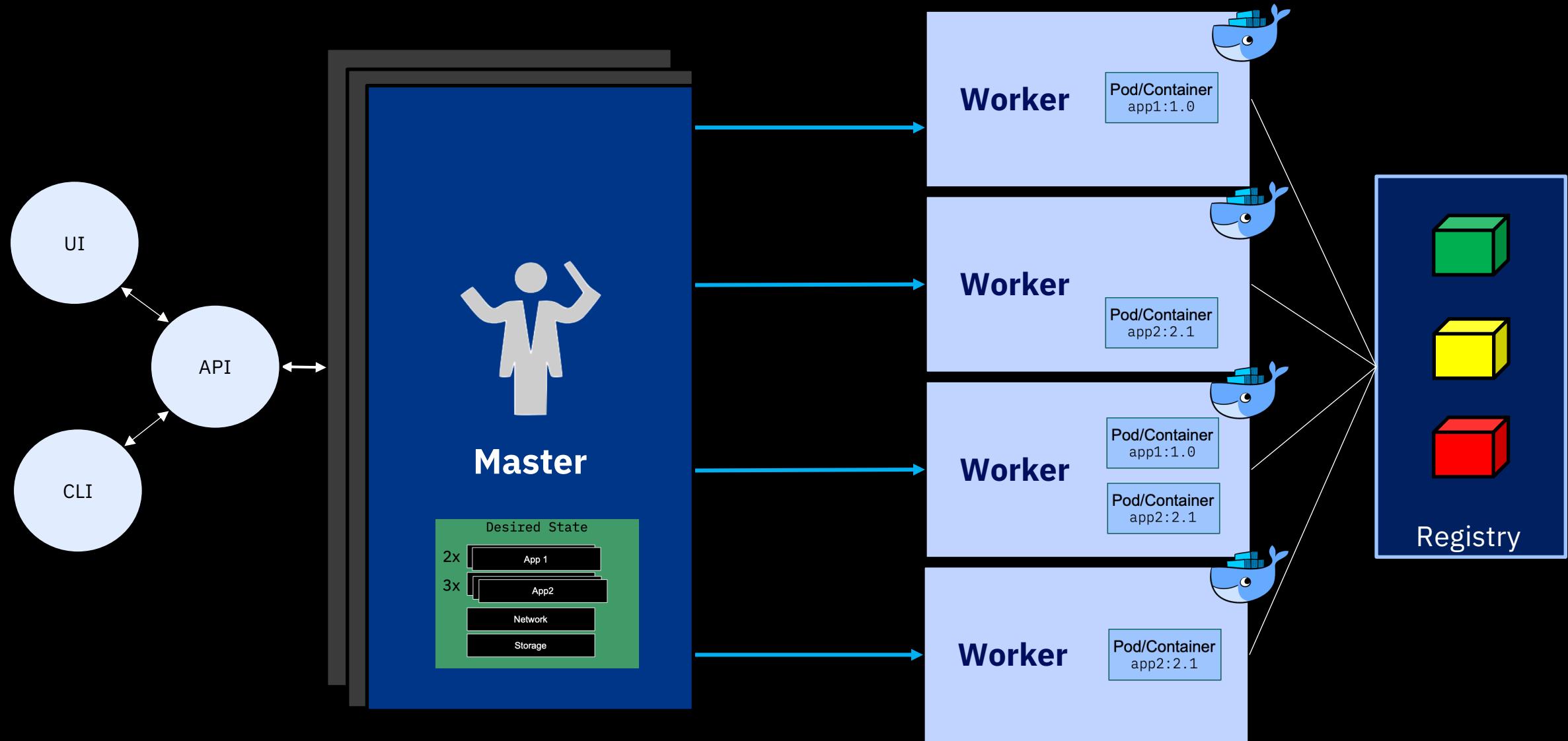
Kubernetes ensures that all the containers running across the cluster are in the desired state at any moment.



```
kind: Deployment
apiVersion: extensions/v1beta1
metadata:
  name: mcmk-ibm-mcmk-prod-klusterlet
  labels:
    app: ibm-mcmk-prod
    chart: ibm-mcmk-prod-3.1.2
    component: "klusterlet"
    release: mcmk
    heritage: Tiller
spec:
  replicas: 1
  selector:
    matchLabels:
      app: ibm-mcmk-prod
      component: klusterlet
      release: mcmk
  template:
    metadata:
      labels:
        app: ibm-mcmk-prod
        component: "klusterlet"
        release: mcmk
        heritage: Tiller
        chart: ibm-mcmk-prod-3.1.2
      annotations:
        productName: "IBM Multi-cloud Manager - Klusterlet"
        productID: "354b8990aab44c9988a0edfd101b128"
        productVersion: "3.1.2"
    spec:
```



Kubernetes Management Architecture





What is Kubernetes?

Container orchestrator

- Runs and manages containers
- Unified API for deploying web applications, batch jobs, and databases
- Maintains and tracks the global view of the cluster
- Supports multiple cloud and bare-metal environments

Manage applications, not machines

- Rolling updates, canary deploys, and blue-green deployments

Designed for extensibility

- Rich ecosystem of plug-ins for scheduling, storage, networking

Open source project managed by the Linux Foundation

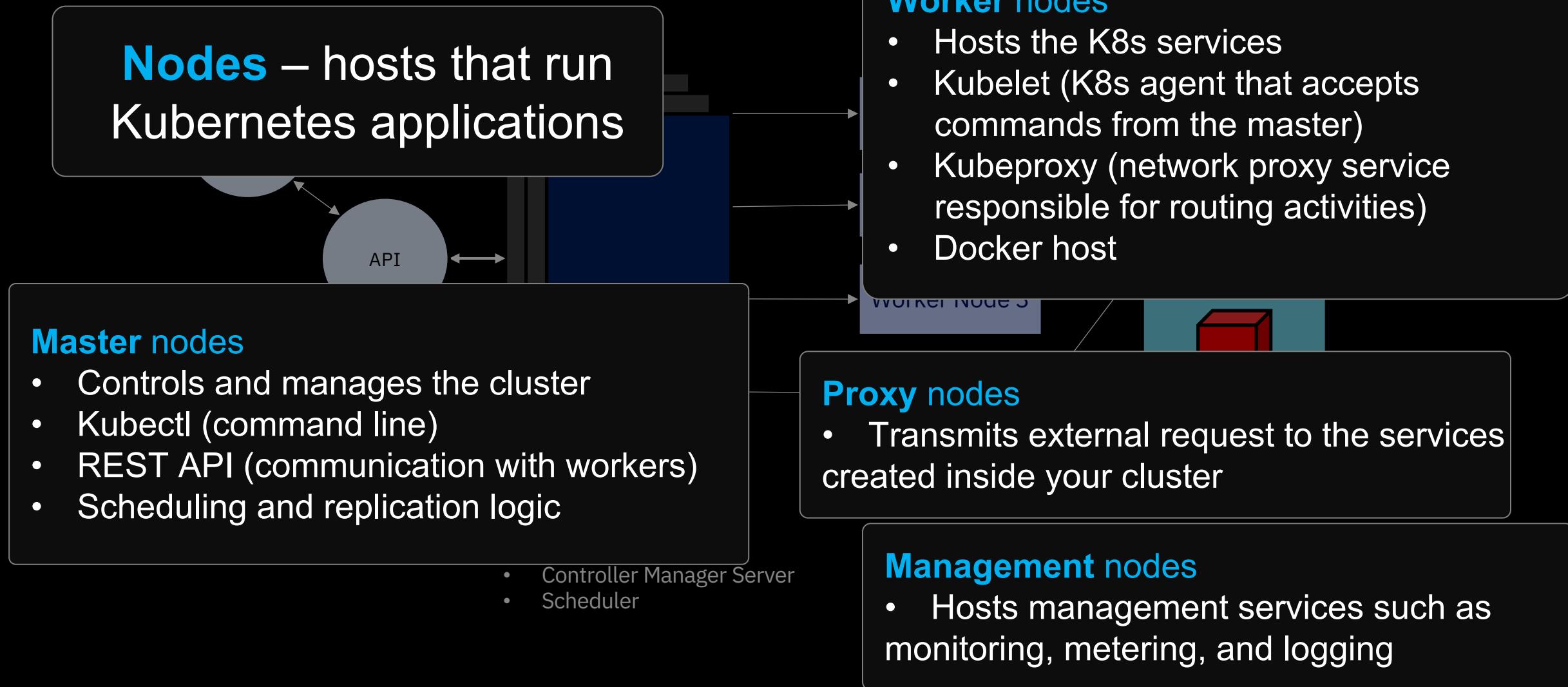
- Inspired and informed by Google's experiences and internal systems
- 100% open source, written in Go

Kubernetes Strengths



- Kubernetes has a **clear governance model** managed by the Linux Foundation
- A growing and **vibrant** Kubernetes **ecosystem**
- Kubernetes **avoids dependency and vendor lock-in**
- Kubernetes supports a **wide range of deployment options**

Kubernetes Management Architecture

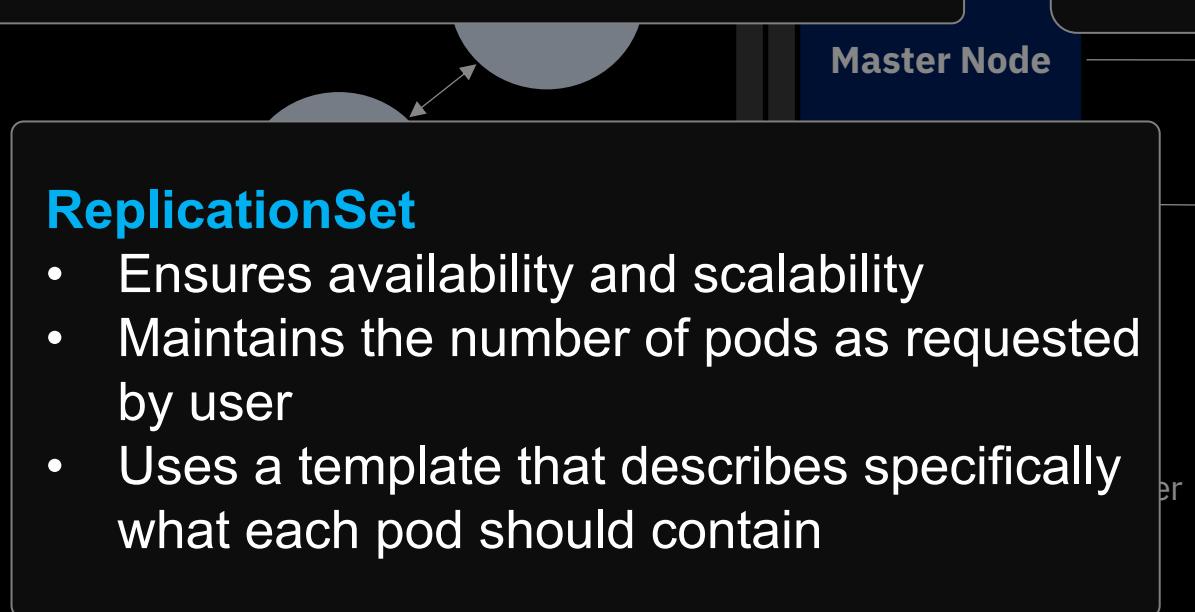


Kubernetes Management Architecture



Pods

- Smallest deployment unit in K8s
- Collection of containers that run on a worker node
- Each has its own IP
- Pod shares a PID namespace, network, and hostname



ReplicationSet

- Ensures availability and scalability
- Maintains the number of pods as requested by user
- Uses a template that describes specifically what each pod should contain

Deployment

- A set of pods to be deployed together
- Declarative - creates a ReplicaSet describing the desired state
- Rollout/ Rollback - Deployment controller changes the actual state to the desired state
- Scale and autoscale: A Deployment can be scaled



Service

- Collections of pods exposed as an endpoint
- A service provides a way to refer to a set of Pods (selected by labels) with a single static IP address

Kubernetes Management Architecture



ConfigMap

- Configuration values to be used by containers in a pod
- Stores configuration outside of the container image, making containers more reusable

Labels

- Metadata assigned to K8s resources
- Key-value pairs for identification
- Critical to K8s as it relies on querying the cluster for resources that have certain labels

Secrets

- Sensitive info that containers need to consume
- Encrypted in special volumes mounted automatically

- Etcd
- API Server
- Controller Manager Server

API Server – Kubernetes API server

Worker Node 3

etcd - a highly-available key value store which K8s uses for persistent storage of all of its REST API objects

Scheduler – schedules pods in worker nodes

Kubernetes Cluster Architecture

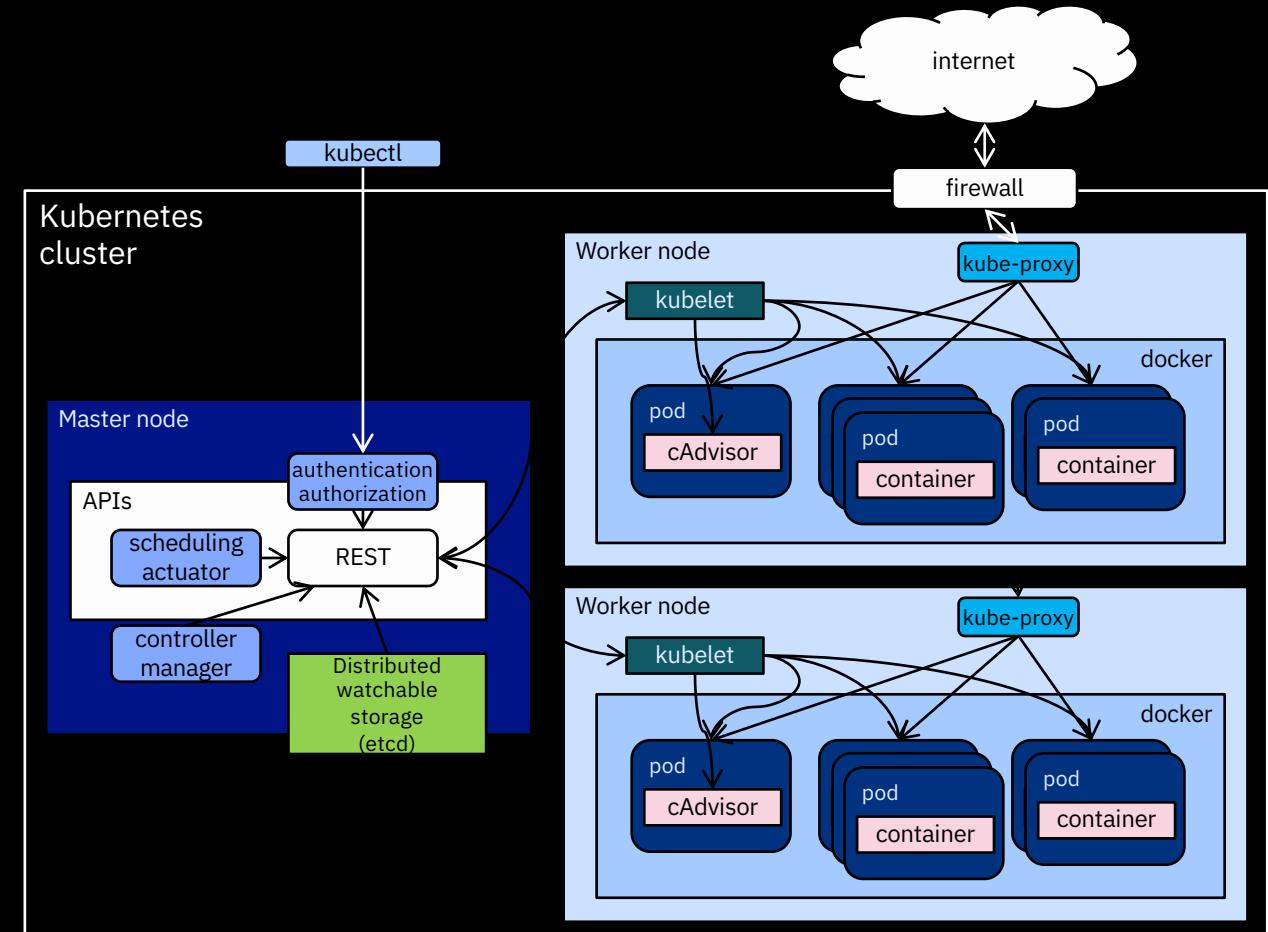


Master node

- Node that manages the cluster
- Scheduling, replication & control
- Multiple nodes for HA

Worker nodes

- Node where pods are run
- Docker engine
- kubelet agent accepts & executes commands from the master to manage pods
- cAdvisor – Container Advisor provides resource usage and performance statistics
- kube-proxy – routes inbound or ingress traffic



kubectl – talking to the Cluster



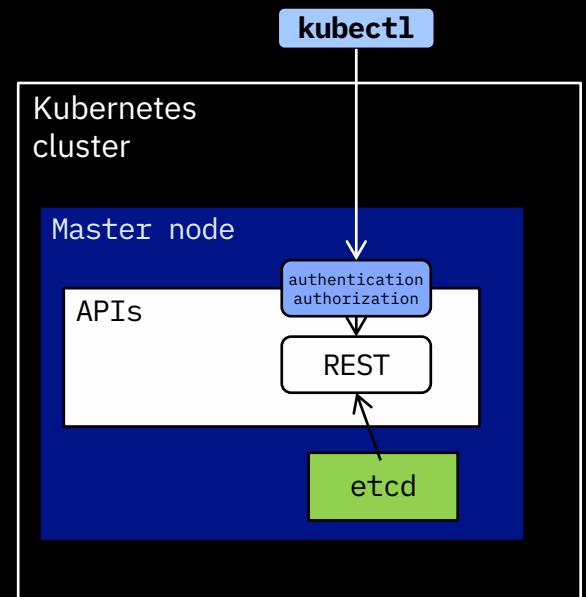
kubectl is a command line interface for running commands against Kubernetes clusters (read state, create objects, ...).

kubectl looks for a file named config in the \$HOME/.kube directory.

Kubernetes uses etcd as a key-value database store.
It stores the configuration of the Kubernetes cluster in etcd.
It also stores the *actual* state of the system and the *desired* state of the system in etcd.

Anything you might read from a `kubectl get xyz` command is stored in etcd.

Any change you make via `kubectl create` will cause an entry in etcd to be updated.



kubectl – talking to the Cluster



kubectl is a command line interface for running commands against Kubernetes clusters.

kubectl looks for a file named config in the \$HOME/.kube directory.

`kubectl [command] [TYPE] [NAME]`

`kubectl create -f example.yaml`

Create objects in yaml file

`kubectl apply -f example.yaml`

Modify objects in yaml file

`kubectl delete -f example.yaml`

Delete objects in yaml file

`kubectl get pods`

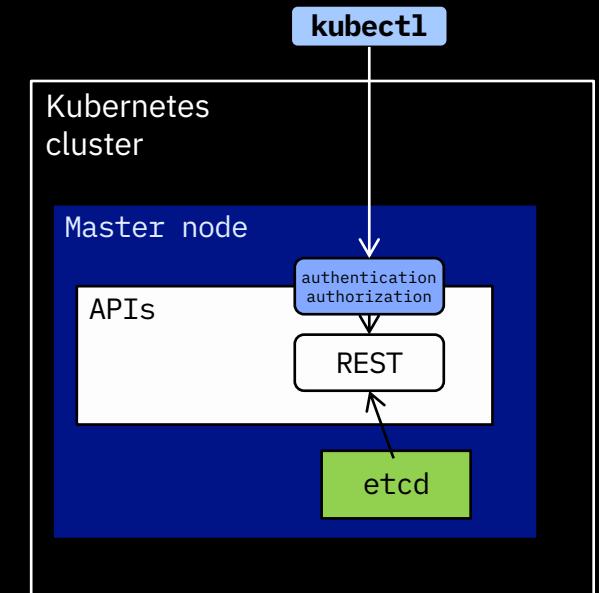
List Pods in Namespace

`kubectl describe nodes <node-name>`

Details about K8s object

`kubectl logs <pod-name>`

Get the logs for a Pod



Pod



- A group of one or more containers is called a pod.
- Containers in a pod are deployed together, and are started, stopped, and replicated as a group.
- Containers in pod share the same network interface.
- Applications in the same pod
 - Share IP Address and port space
 - Share the same hostname
 - Can communicate using native IPC
 - Can share mounted storage
- Applications in different pods
 - Have different IP Addresses
 - Have different hostnames
 - Pods running on the same node might as well be on different servers
- **When designing pods ask, “Will these containers work correctly if they land on different machines?”**

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
    - name: nginx
      image: nginx:1.7.9
      ports:
        - containerPort: 80
```



10.0.0.1



10.0.0.2

Creating a Pod



```
kubectl run nginx --image=nginx:1.7.9  
or  
kubectl create -f example.yaml
```

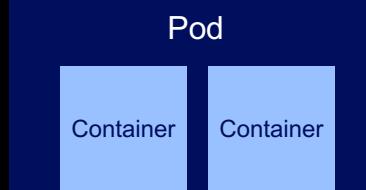
```
kubectl get pods  
NAME          READY  STATUS    RESTARTS  AGE  
nginx-5bd87f76c-vxc79  1/1    Running   0          29s
```

example.yaml

```
apiVersion: v1  
kind: Pod  
metadata:  
  name: nginx  
spec:  
  containers:  
    - name: nginx  
      image: nginx:1.7.9  
      ports:  
        - containerPort: 80
```



10.0.0.1

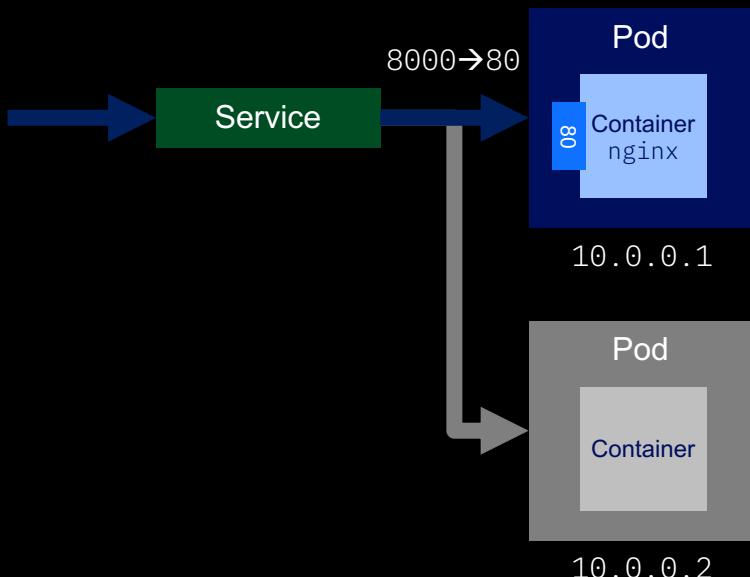


10.0.0.2

Service



- A service provides a way to refer to a set of Pods (selected by labels) with a single static IP address.
- Also provide load balancing

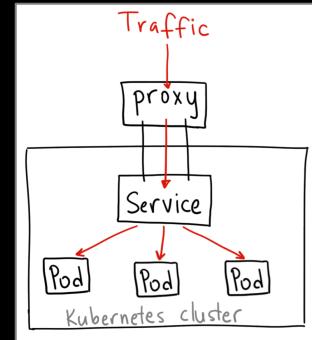


```
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  ports:
    - port: 8000
      targetPort: 80
      protocol: TCP
  selector:
    app: nginx
```

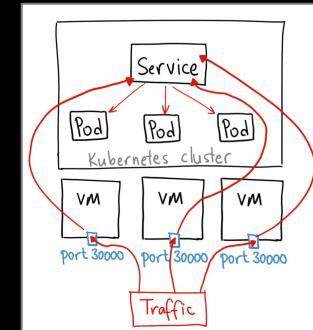
Service



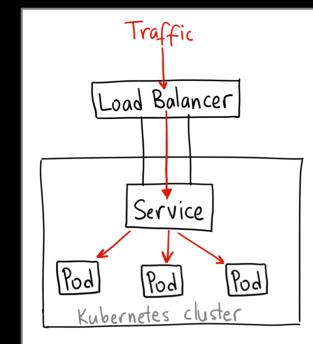
ClusterIP: This type exposes the service on the cluster internal IP. This means that the service is only reachable from within the cluster.



NodePort: This type exposes the service on each Node's static IP address.



LoadBalancer: This service type exposes a service using the cloud provider load balancer.



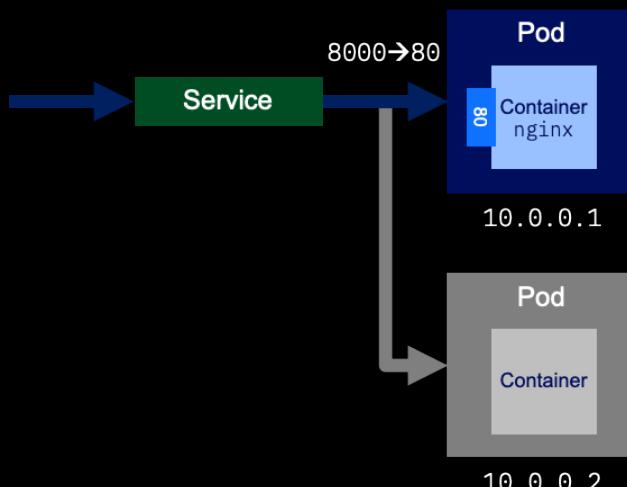
Creating a Service



```
kubectl expose deployment nginx --type="NodePort" --port=8000 --target-port=80  
or  
kubectl create -f example.yaml
```

```
kubectl get services
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
nginx	NodePort	10.106.115.135	<none>	8000:32499/TCP	6s



example.yaml

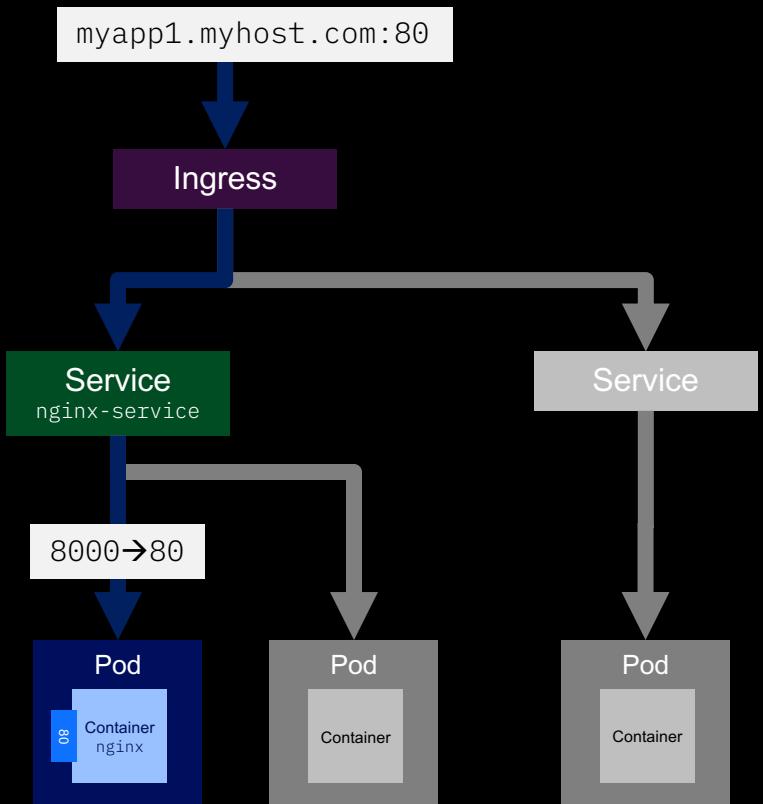
```
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  ports:
    - port: 8000
      targetPort: 80
      protocol: TCP
  selector:
    app: nginx
```

Ingress



An ingress can be configured to give services externally-reachable URLs, load balance traffic, terminate SSL, and offer name based virtual hosting. An ingress controller is responsible for fulfilling the ingress, usually with a loadbalancer.

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: nginx-ingress
  annotations:
    ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - host: myapp1.myhost.com
    http:
      paths:
      - backend:
          serviceName: nginx-service
          servicePort: 80
```

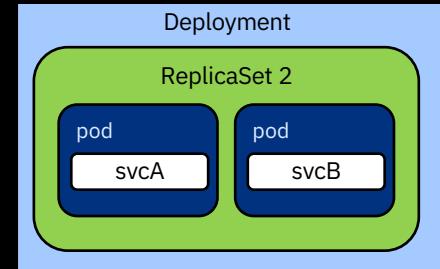


Deployments & ReplicaSets



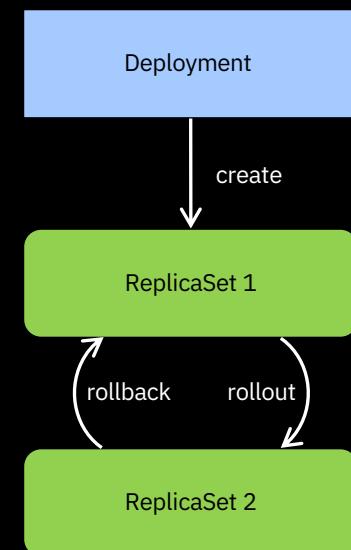
Deployment

- A set of **pods to be deployed together**, such as an application
- **Declarative**: Revising a Deployment **creates a ReplicaSet** describing the desired state
- **Rollout**: Deployment controller changes the actual state to the desired state at a controlled rate
- **Rollback**: Each Deployment revision can be rolled back
- **Scale** and autoscale: A Deployment can be scaled



ReplicaSet

- Cluster-wide pod manager that **ensures the proper number of pods are running** at all times.
- A set of pod templates that describe a set of pod replicas
- Uses a template that describes specifically what each pod should contain
- Ensures that a specified number of pod replicas are running at any given time



Deployment



- A Deployment object defines a Pod creation template and **desired replica count**.
- Create or delete Pods as needed to meet the replica count.
- Manage safely **rolling out changes** to your running Pods.

```
kubectl create -f example.yaml
```

example.yaml

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 2
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.7.9
      ports:
        - containerPort: 80
```

StatefulSet, DaemonSet, Job...



StatefulSet

- Intended to be used with stateful applications and distributed systems.
 - Pods are created sequentially
 - Ordinal index and stable network identity

DaemonSet

- Ensures that all (or some) nodes run a copy of a pod
 - running a cluster storage daemon
 - running a logs collection daemon
 - running a node monitoring daemon

Job

- Creates one or more pods and ensures that a specified number of them successfully terminate

Cron Job

- Manage time based jobs, once at a specified in time, or repeatedly at a specified time point

Naming



Name

- Each resource object by type has a unique name

Namespace

- **Resource isolation:** Each namespace is a virtual cluster within the physical cluster
 - Resource objects are scoped within namespaces
 - Low-level resources are not in namespaces: nodes, persistent volumes, and namespaces themselves
 - Names of resources need to be unique within a namespace, but not across namespaces
- **Resource quotas:** Namespaces can divide cluster resources
- Initial namespaces
 - **default** – The default namespace for objects with no other namespace
 - **kube-system** – The namespace for objects created by the Kubernetes system

Resource Quota

- Limits resource consumption per namespace
- Limit can be number of resource objects by type (pods, services, etc.)
- Limit can be total amount of compute resources (CPU, memory, etc.)
- Overcommit is allowed; contention is handled on a first-come, first-served basis

Configuring Resources and Containers



ConfigMap

- Configuration values to be used by containers in a pod
- Stores configuration outside of the container image, making containers more reusable

Secret

- Sensitive info that containers need to read or consume
- Encrypted in special volumes mounted automatically

Configuring Resources and Containers



```
apiVersion: extensions/v1beta1
kind: ConfigMap
apiVersion: v1
metadata:
  name: example-config
data:
  allowed: '"true"'
  enemies: aliens
  lives: "3"
```

```
kind: Deployment
spec:
  containers:
    - name: test-container
      image: xxx
      ...
  env:
    - name: SPECIAL_LEVEL_KEY
      valueFrom:
        configMapKeyRef:
          name: example-config
          key: enemies
```

Can be used as normal environment variable

Kubernetes Autoscaling



Horizontal Pod Autoscaling (HPA)

- Automatically scales the number of pods in a replication controller, deployment, or replica set
- Matches the observed average CPU utilization to the specified target
- Fetches metrics in two different ways: direct Heapster access and REST client access
- Kubernetes Heapster enables container cluster monitoring and performance analysis
- Default config: query every 30 sec, maintain 10% tolerance, wait 3 min after scale-up, wait 5 min after scale-down

```
$ kubectl autoscale deployment <deployment-name> --cpu-percent=50  
--min=1 --max=10 deployment "<hpa-name>" autoscaled
```

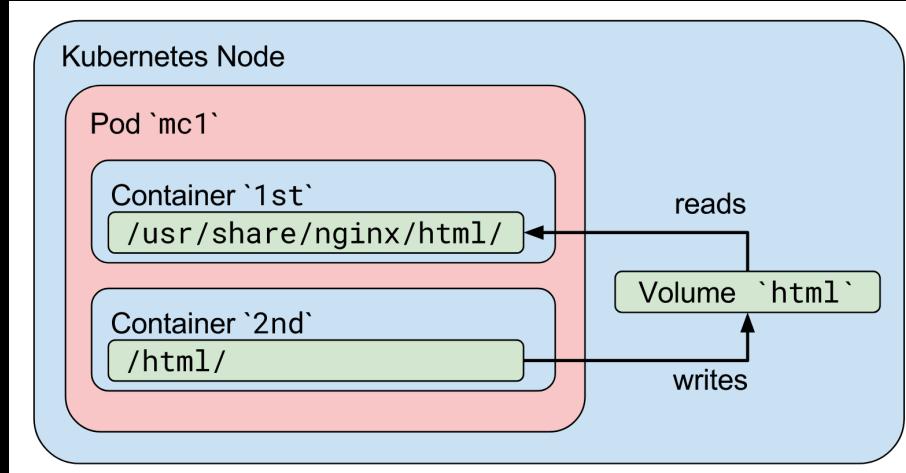
Creates a horizontal pod autoscaler

- An HPA instance
- Maintains between 1 and 10 replicas of the pods controlled by the deployment
- Maintains an average CPU utilization across all pods of 50%

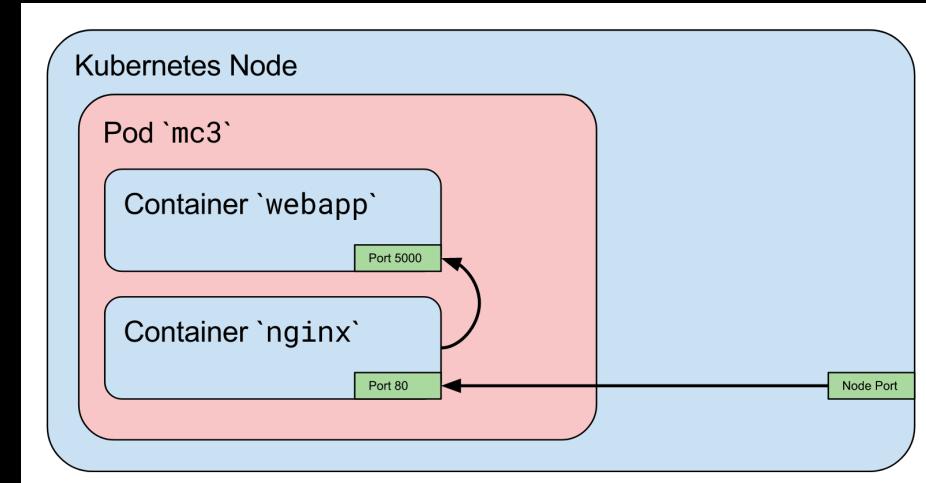
Multi-Container Pod Design Patterns



Shared Volumes

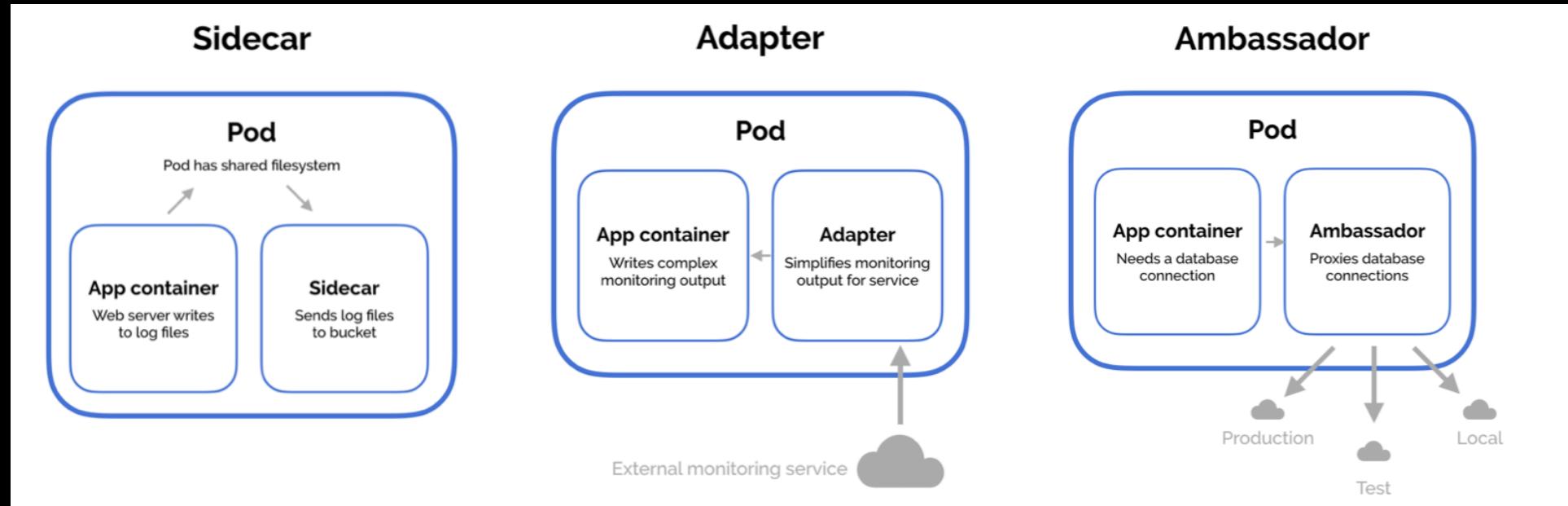


No Network isolation



In what order containers are being started in a Pod?

Multi-Container Pod Design Patterns



Sidecar pattern

The sidecar pattern consists of a main application—i.e. your web application—plus a helper container with a **responsibility that is essential to your application**, but is not necessarily part of the application itself.

Adapter pattern

The adapter pattern is used to standardize and normalize application output or **monitoring data** for aggregation.

Ambassador pattern

The ambassador pattern is a useful way to connect containers with the outside world (**proxy**).

HELM Charts



- Helm is the open standard for **Application Packaging and Deployment** for Kubernetes (like npm, yum or apt-get)
- Helm charts **automate the deployment of resources** and prerequisites including locations of Docker images



Catalog

All Categories > Search items Filter

Category	Chart Name	Description	Status
Blockchain	acs-engine-autoscaler	DEPRECATED Scales worker nodes within agent pools	KUBE Deprecated
Business Automation	aerospike	A Helm chart for Aerospike in Kubernetes	KUBE
Data	airflow	Airflow is a platform to programmatically author, schedule and monitor workflows	KUBE
Data Science & Analytics	anchore-engine	Anchore container analysis and policy evaluation engine service	KUBE
DevOps	apm-server	The server receives data from the Elastic APM agents and stores the data into a datastore li...	KUBE
Integration	ark	A Helm chart for ark	KUBE
IoT	artifactory	DEPRECATED Universal Repository Manager supporting all major packaging formats, build tools	KUBE Deprecated
Network	artifactory-ha	Universal Repository Manager supporting all major packaging formats, build tools	KUBE Deprecated
Operations	artifactory-ha	Universal Repository Manager supporting all major packaging formats, build tools	KUBE Deprecated
Runtimes & Frameworks	audit-beat	A lightweight shipper to audit the activities of users and processes on your systems	KUBE
Security	auth-apikeys	ICP IAM Token Service	KUBE
Storage	audit-logging	Audit logging storage and search management solution	mgmt-charts
Tools	auth-idp	ICP Security Authentication Provider	mgmt-charts
Other	auth-pap	ICP IAM Policy Administration	mgmt-charts
	auth-pdp	ICP IAM Policy Decision	mgmt-charts
	aws-cluster-autoscaler	Scales worker nodes within autoscaling groups.	KUBE Deprecated
	bitcoind	Bitcoin is an innovative payment network and a new kind of money.	KUBE
	bookstack	BookStack is a simple, self-hosted, easy-to-use platform for organising and storing informat...	KUBE

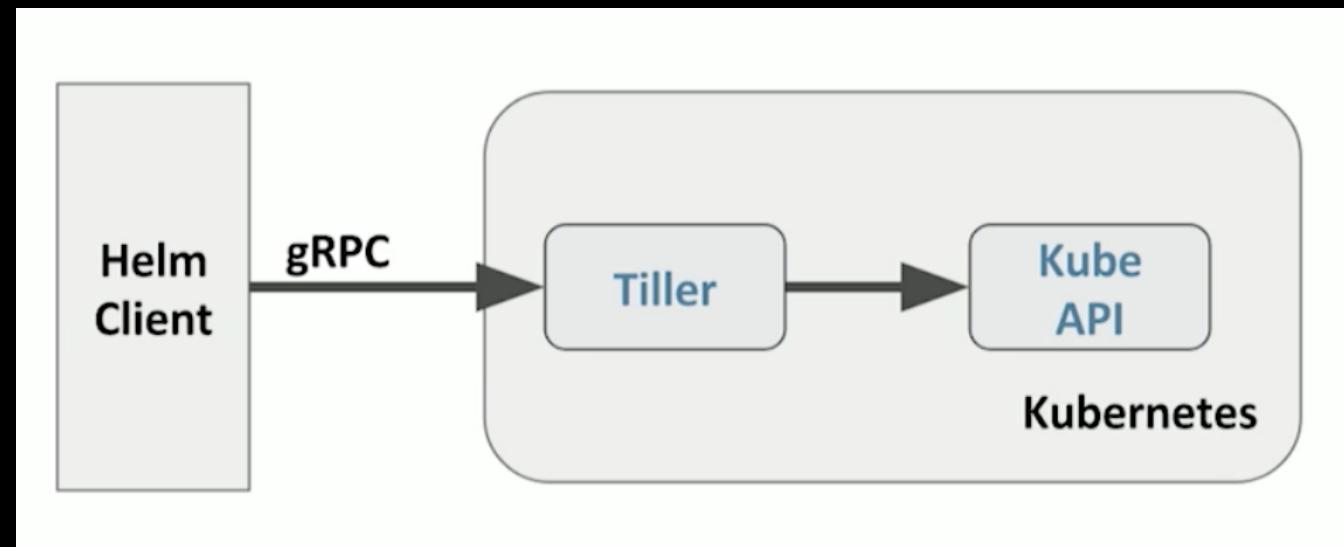
HELM Example



Helm has 2 Components :

Tiller : Tiller is the in-cluster component of Helm. It interacts directly with the Kubernetes API server to install, upgrade, query, and remove Kubernetes resources. It also stores the objects that represent releases.

Client : The Helm client interacts with the Tiller and stores information in a local directory.



HELM Example



- **Chart :**

A Helm package that contains information sufficient for installing a set of Kubernetes resources into a Kubernetes cluster.

📁 templates
📄 .helmignore
📄 Chart.yaml
📄 LICENSE
📄 README.md
📄 values.yaml

- **Chart.yaml**

Basic information about the chart (name, versions, links, icon, ...)

- **Templates (yaml)**

Kubernetes deployment manifests (yaml) with placeholders for external parameters.

- **values.yaml**

A set of variables that will show up in the UI (or customized at the helm command line) and be passed on at deploy time to the deployment manifests.

Deployment Chart

```
1 apiVersion: v1
2 kind: PersistentVolumeClaim
3 metadata:
4   name: {{ template "fullname" . }}-pv-claim
5 spec:
6   accessModes:
7     - ReadWriteMany
8   resources:
9     requests:
10    storage: {{ .Values.storage.size }}
```

Values.yaml file

```
storage:
  size: 2Gi
```



IBM Helm Charts

<https://github.com/ibm/charts>

A screenshot of the GitHub repository page for "IBM / charts". The page shows the repository's landing page with the title "IBM / charts", navigation links for "Code", "Pull requests 0", "Projects 0", "ZenHub", "Wiki", and "Insights", and a description stating "The IBM/charts repository provides helm charts for IBM and Third Party middleware." Below the description are several tags: "kubernetes", "helm", "helm-charts", "ibm", "ibm-bluemix", and "docker".

The IBM/charts repository provides helm charts for IBM and Third Party middleware.

kubernetes helm helm-charts ibm ibm-bluemix docker

Resources



Linux Foundation – Introduction to Kubernetes

- <https://courses.edx.org/courses/course-v1:LinuxFoundationX+LFS158x+2T2017/course/>

Kubernetes tutorial

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

Introduction to container orchestration

- <https://www.exoscale.ch/syslog/2016/07/26/container-orch/>

TNS Research: The Present State of Container Orchestration

- <https://thenewstack.io/tns-research-present-state-container-orchestration/>

Large-scale cluster management at Google with Borg

- <https://research.google.com/pubs/pub43438.html>

Benefits of using a Kubernetes Based Platform

Speed

- Fast boot-time
- Easy scalability
- Rapid deployment

Portability

- Between different environments
- Between private and public cloud

Efficiency

- Better usage of computer resources

Automation

- Next level standardization and automation

13x

More software releases

Eliminate

“works on my machine” syndrome

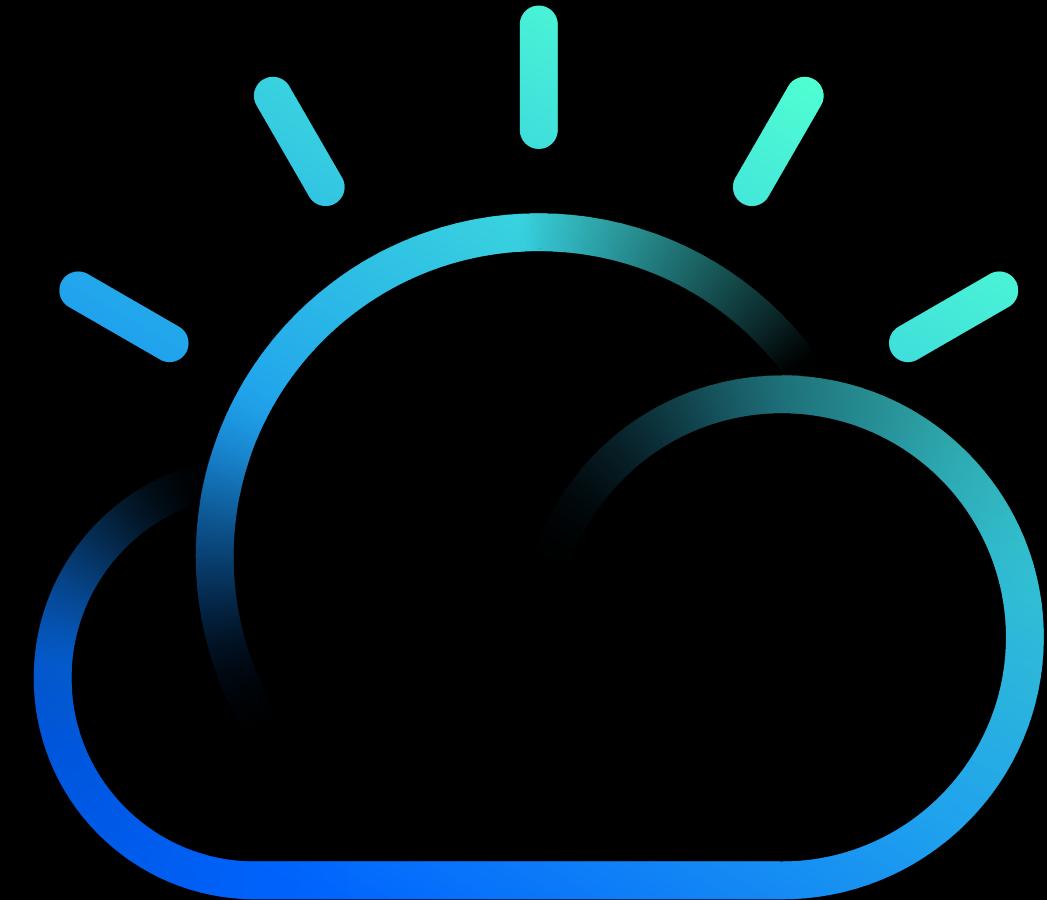
~47%

Reduction of VMs,
OS licensing and
server cost

Reduce

operation cost

QUESTIONS?



IBM Cloud



°° The Journey to Cloud
Kubernetes - Hands-On

04



IBM Cloud

Remember your Team Color

black 31701

olive 31711

peru 31715

white 31702

brown 31712

chocolate 31716

red 31703

lightblue 31713

orchid 31717

blue 31704

orange 31708

gold 31718

yellow 31705

purple 31709

pink 31719

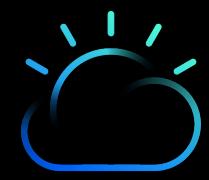
lime 31706

maroon 31710

violet 31720

cyan 31707

firebrick 31714



Collector - Accessing team web site

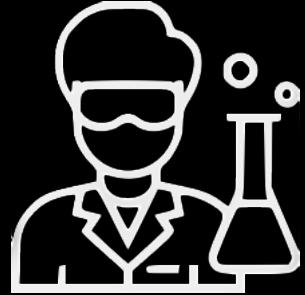
http://158.177.137.195:{port#}

Team name / color will be shown

blue 31704

The screenshot shows a web browser window with the title "Collector - blue". The navigation bar includes links for "Courses", "Class work", "Statistics", "Information", and "Feedback". Below the navigation bar, a section titled "Catalog of courses" displays a list of courses. A dropdown menu is open over the first item, "KUB01 Lab Setup", with the text "select course" preceding the list. The list contains five items: "KUB01 Lab Setup", "KUB02 Kubernetes Introduction", "KUB03 Kubernetes Labs", and "KUBADV01 Istio". To the right of the dropdown, there is a button labeled "Begin course". A large callout box with a red border and white text, containing the instruction "Select course and press button to begin", points to the "Begin course" button.

Current course catalog



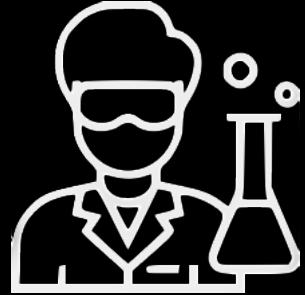
JTC01 Labs

Lab 0 : Preparation

Lab 1: Set up and deploy your first application

Lab 2 : Scale and Update Deployments

Lab 3 : Scale and update apps natively

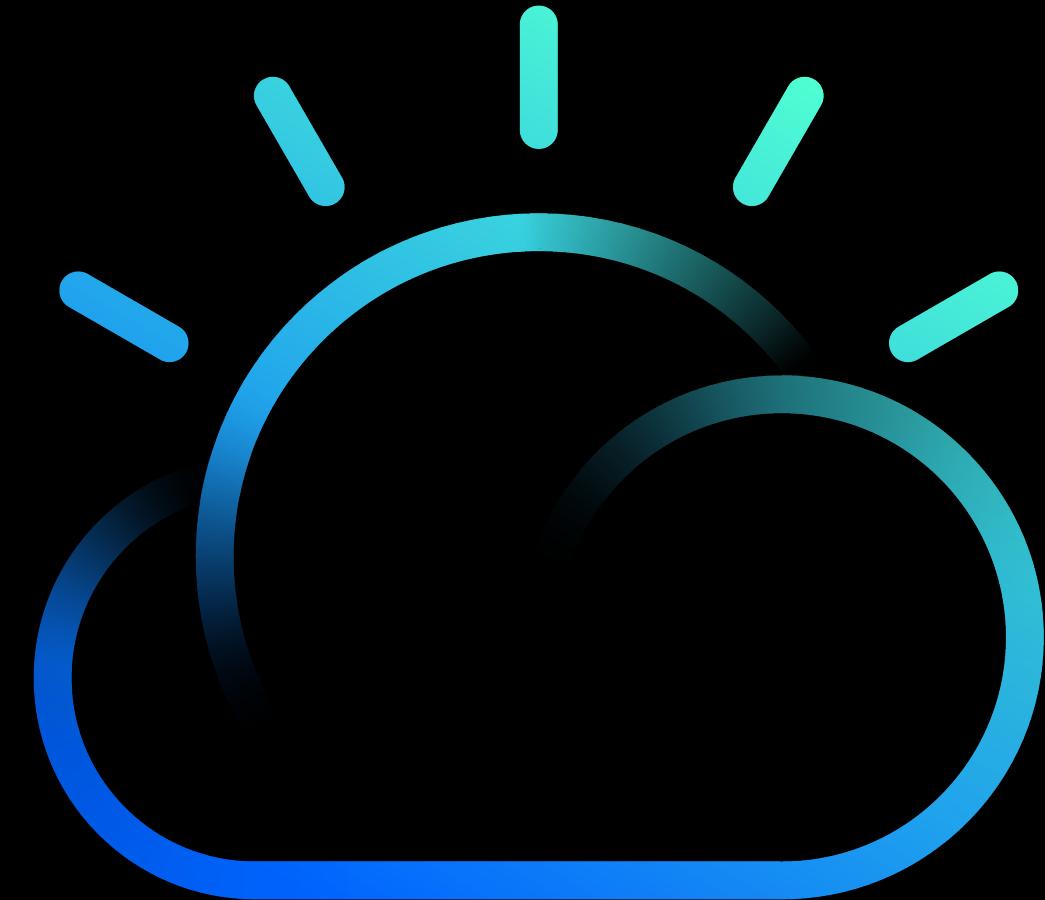


READY
SET
GO!!!!

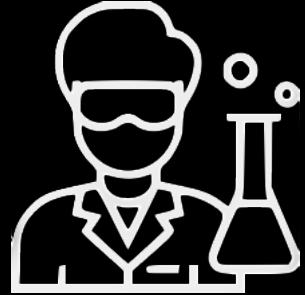
<http://158.177.137.195:{port#}>

Duration: 90 mins

QUESTIONS?



IBM Cloud



Sources and documentation will be available here:

https://github.com/niklaushirt/k8s_training_public

IBM