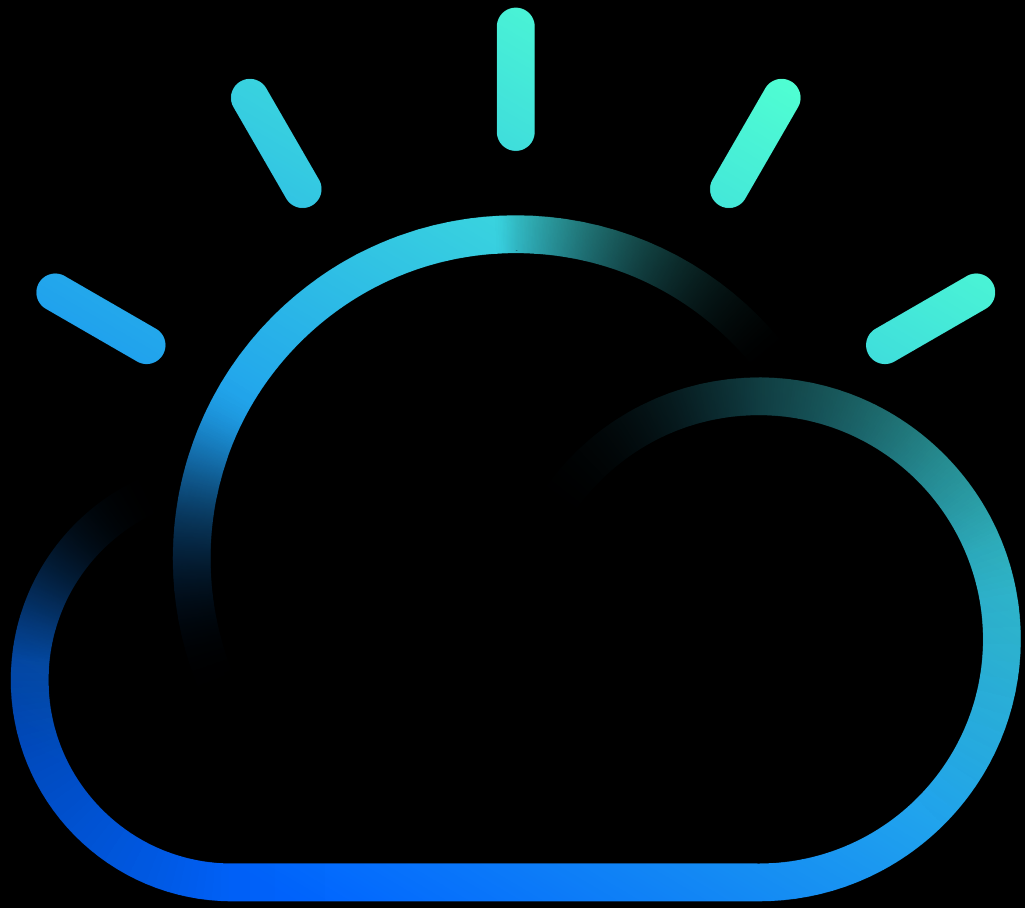
The Path to Cloud

Kubernetes Basic Concepts

Niklaus Hirt
DevOps Architect / Cloud Architect
nikh@ch.ibm.com

The path 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 0: Prepare the Labs

Module 1: Microservices

Module 2: Containers with Docker

Module 3: Kubernetes

Module 4: Kubernetes Hands-On

Agenda - Kubernetes Advanced Concepts

Module 5: Mesh Networking with ISTIO

Module 6: Mesh Networking Hands-On

Module 7: Serverless with Knative

Module 8: Serverless with Knative Hands-On (Optional)

Module 9: GitOps with ArgoCD (Optional)

Module 10: Multi-Cloud Management (Optional)

Wrap-up

Agenda - Kubernetes Basic Concepts

Module 0: Prepare the Labs

Module 1: Microservices

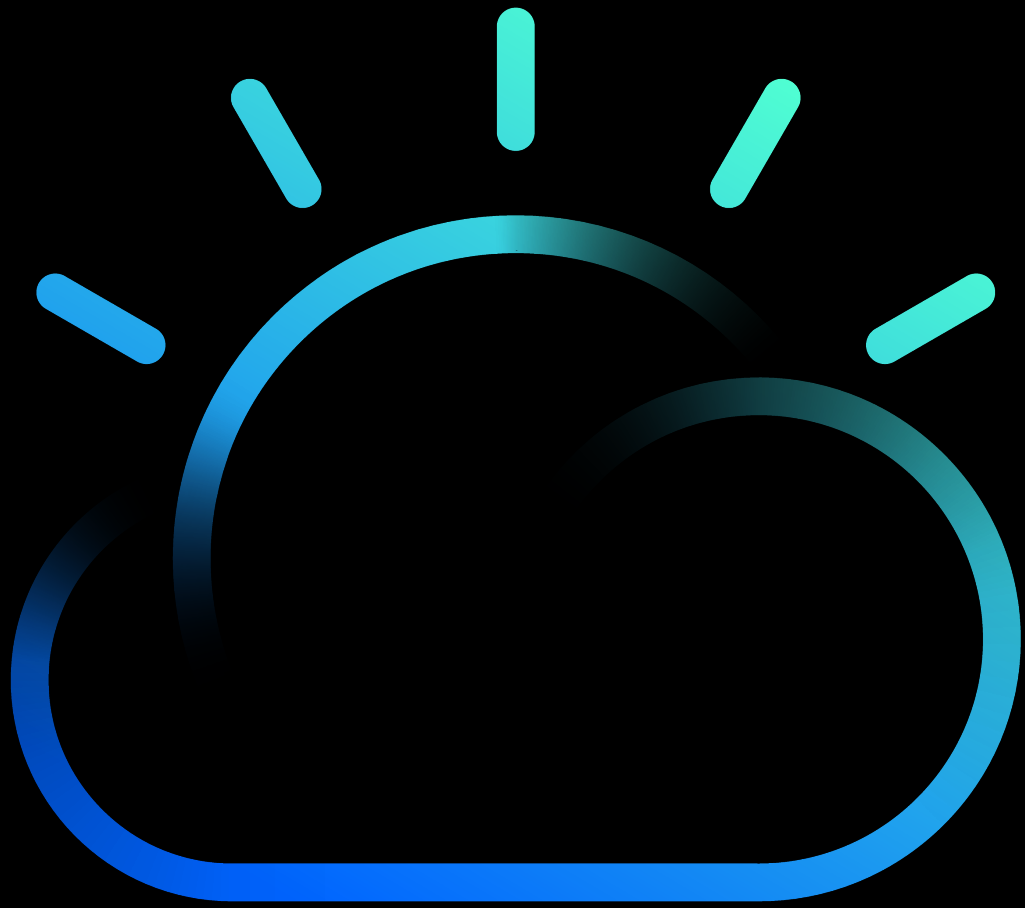
Module 2: Containers with Docker

Module 3: Kubernetes

Module 4: Kubernetes Hands-On

The path to Cloud
Prepare the Labs

00

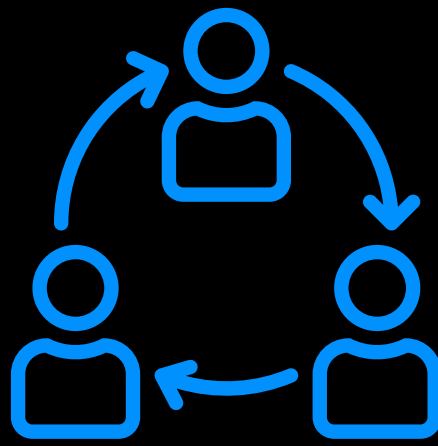


IBM **Cloud**

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

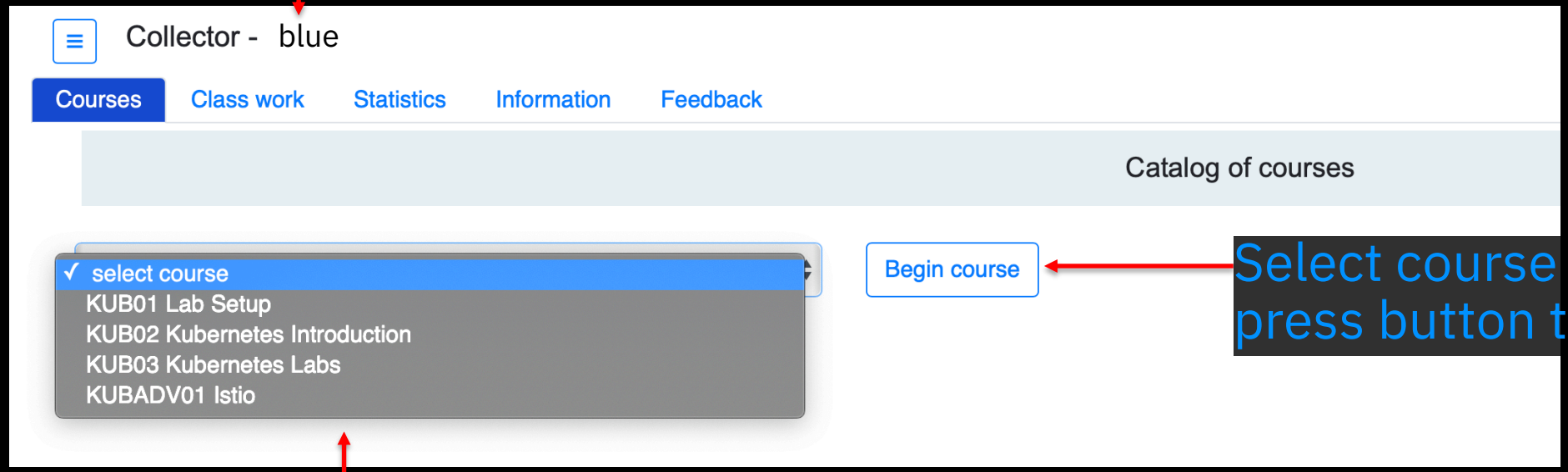
black 31701	olive 31711	peru 31715
white 31702	brown 31712	chocolate 31716
red 31703	lightblue 31713	orchid 31717
blue 31704	orange 31708	gold 31718
yellow 31705	purple 31709	pink 31719
lime 31706	maroon 31710	violet 31720
cyan 31707	firebrick 31714	

Collector - Accessing team web site

http://158.177.137.195:{port#}

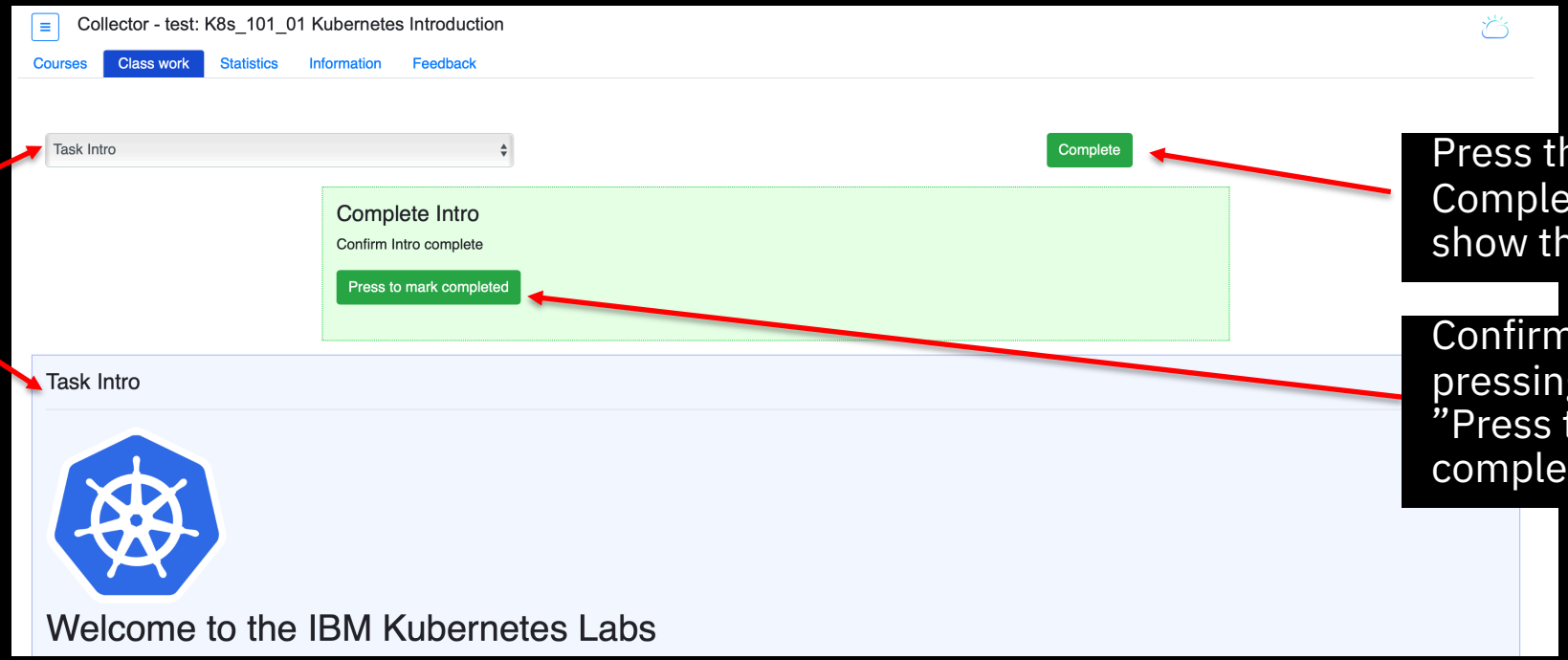
blue 31704

Team name / color will be shown



Collector – Class work

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



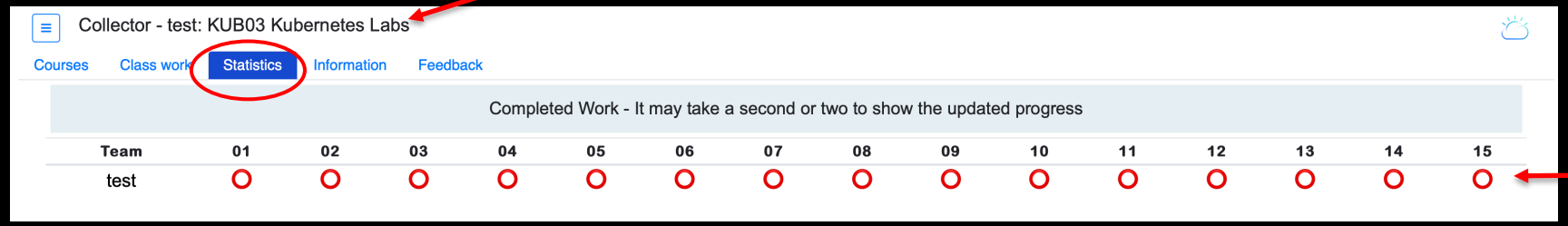
Press the green Complete button to show the green portion.

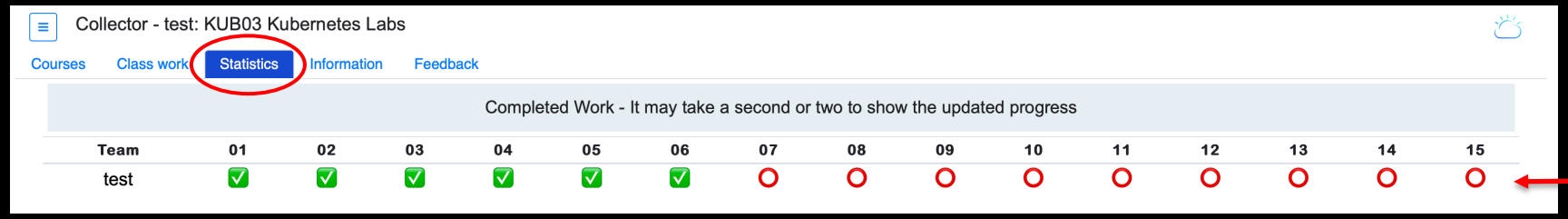
Confirm completion by pressing the green ”Press to mark completed” button.

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

Collector – Track course completed work

Course title



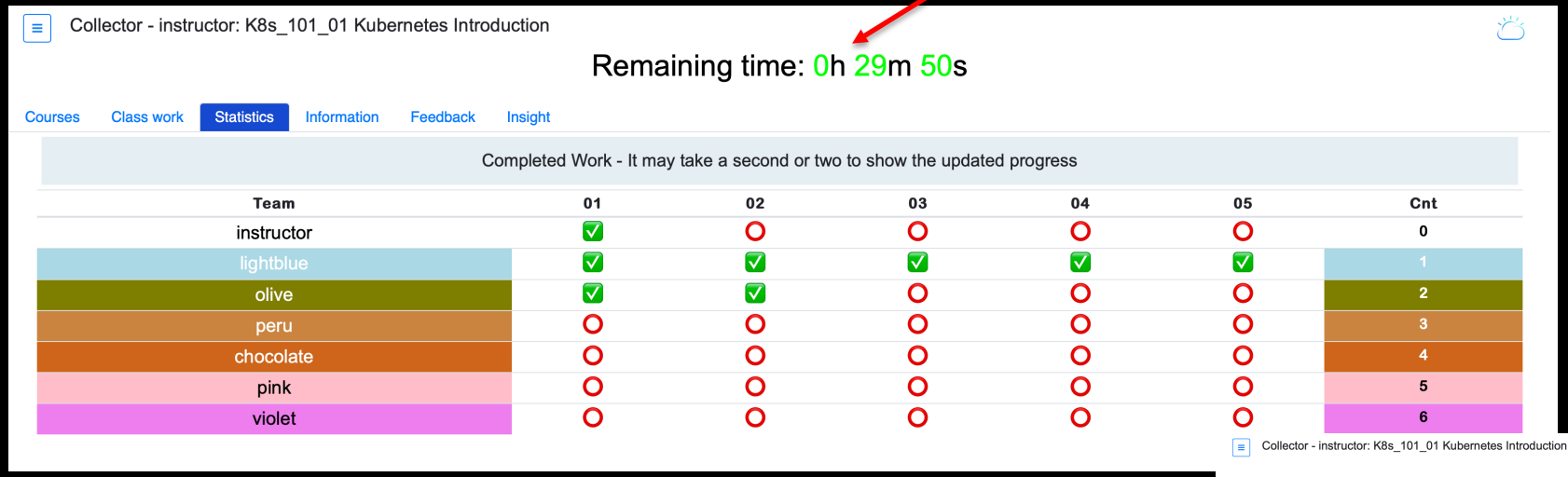


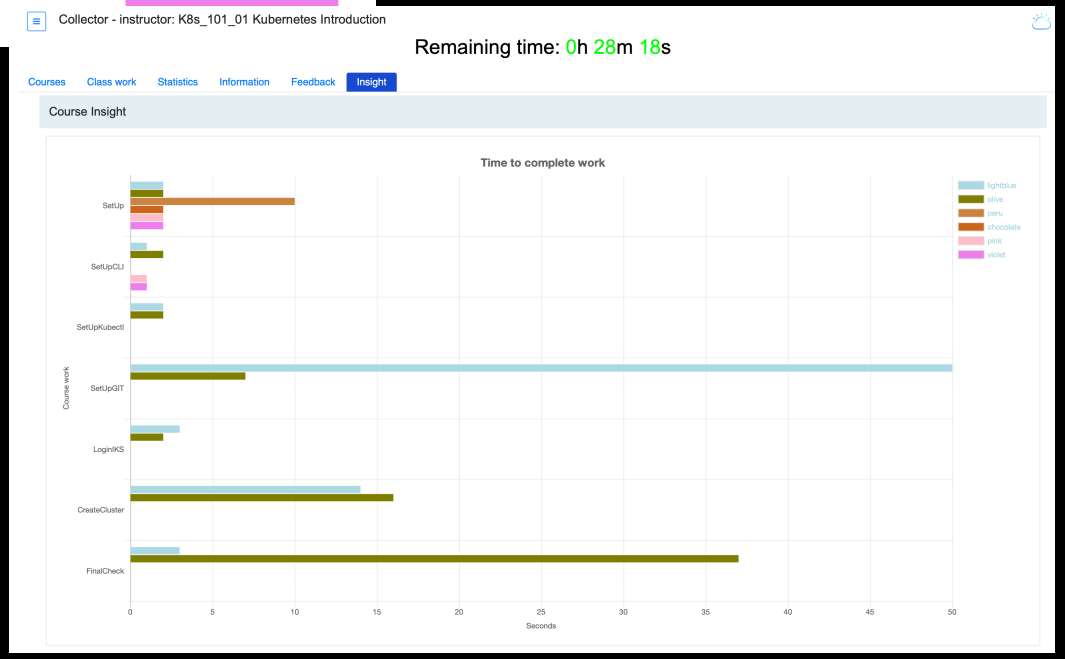
The number of items tracked will change based on the current course selected.

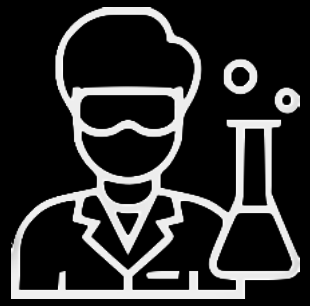
Green checkmark - item is completed
Red circle - item is waiting to be completed

Collector – Instructor Dashboard

Remaining Time for the Lab







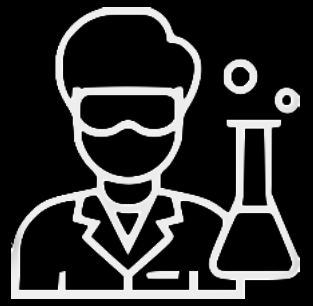
KUB01 Lab Setup

Task 1: Setup Minikube

Task 2: Setup kubectl

Task 3: Setup git

Task 4: Final Check



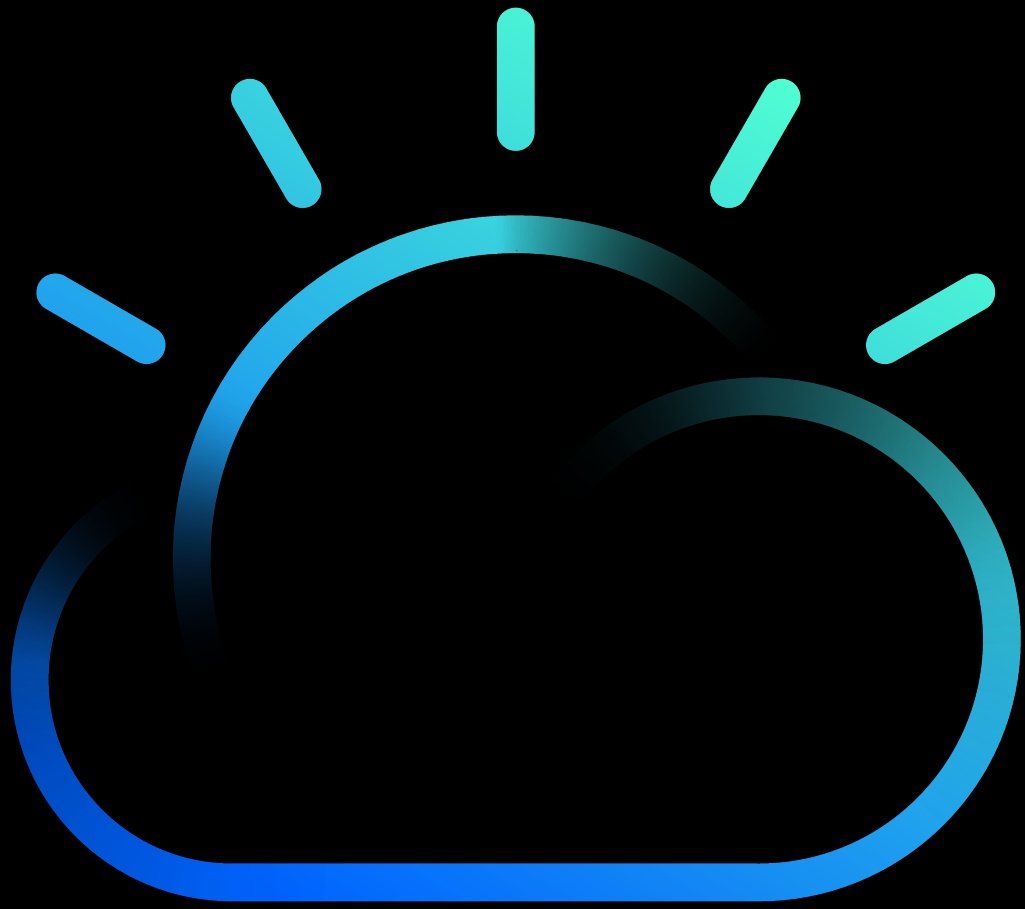
READY SET GO!!!!

http://158.177.137.195:{port#}

Duration: 30 mins

The path to Cloud
Microservices

01



IBM **Cloud**

Kodak

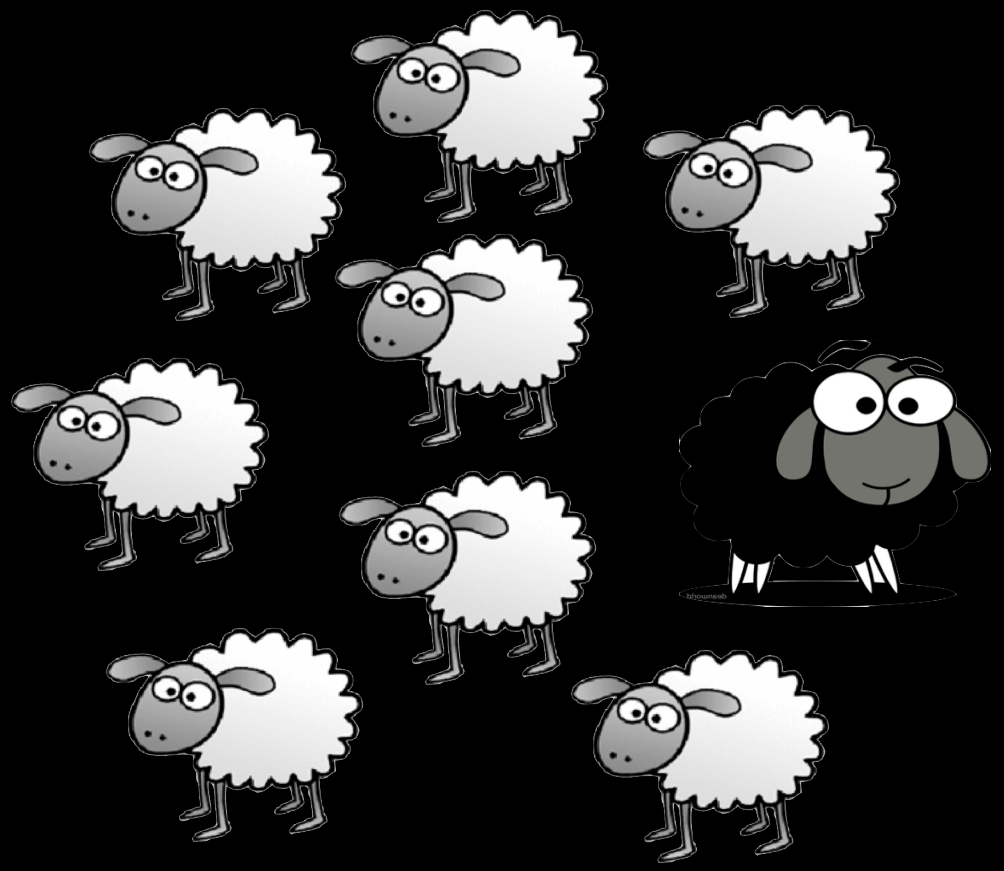
disruption

dɪsˈrʌpʃn/

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

dɪsˈrʌpʃn/

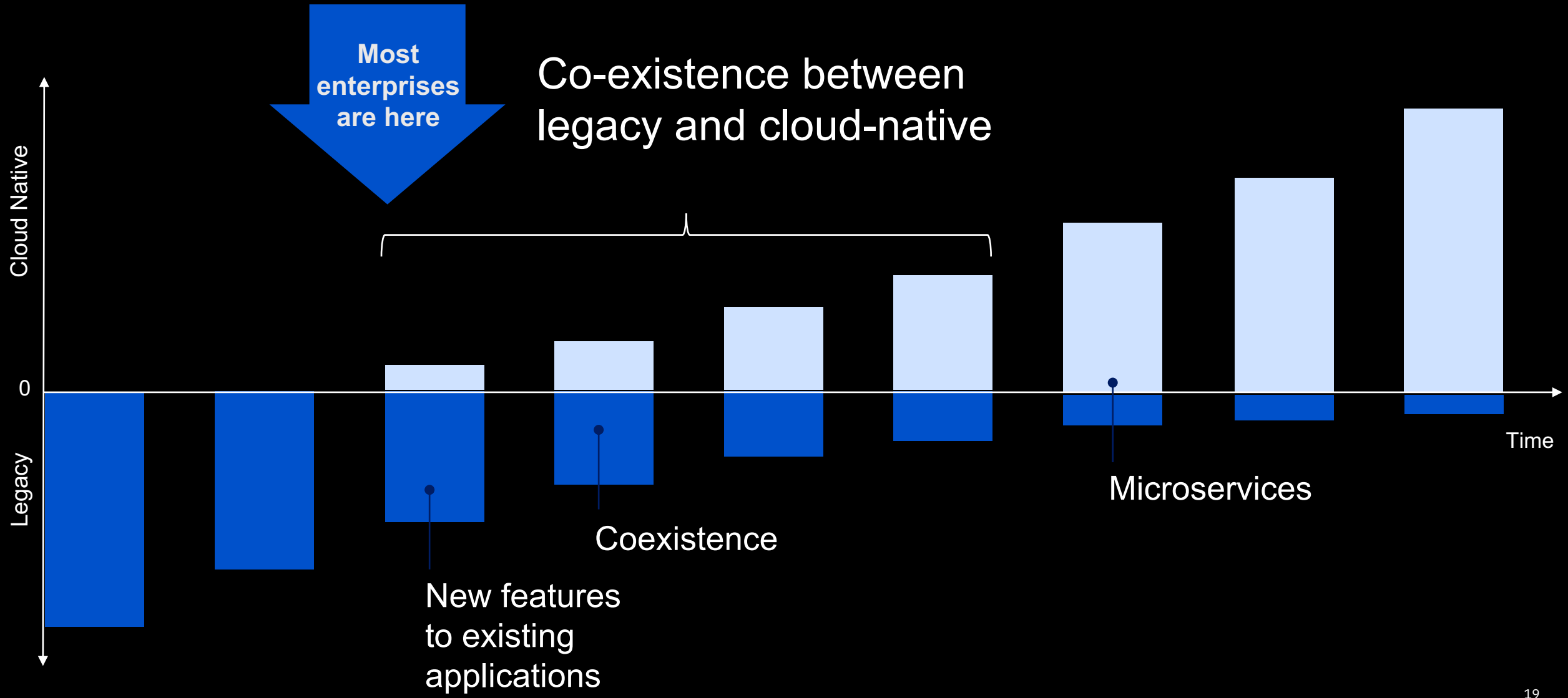
noun

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	

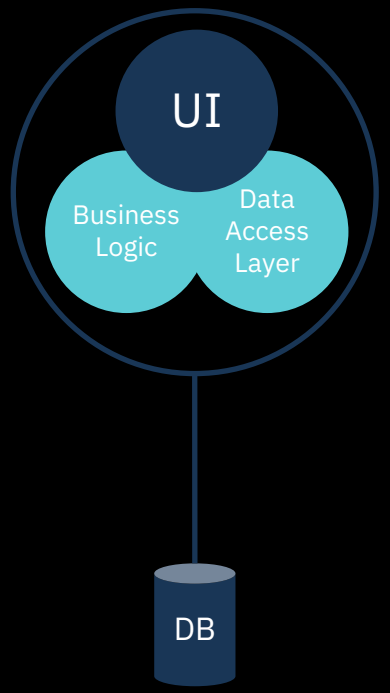
Microservices

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

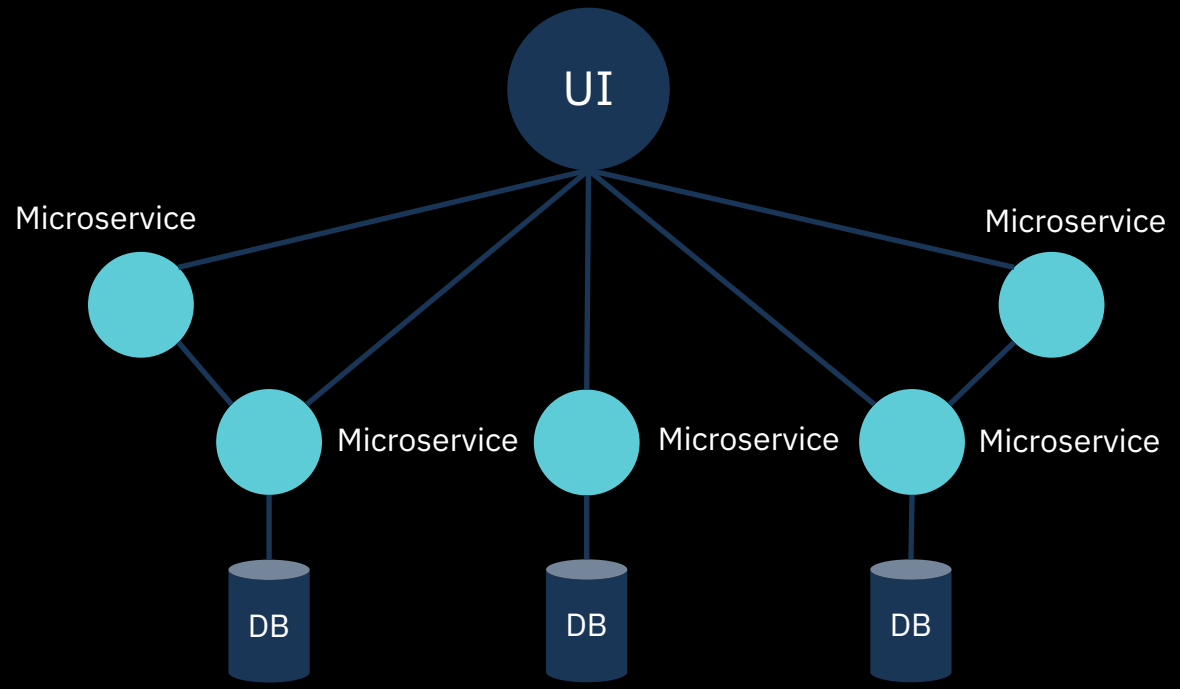
Accelerate delivery by minimizing communication and coordination between people

Microservices architecture

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



Monolithic Architecture

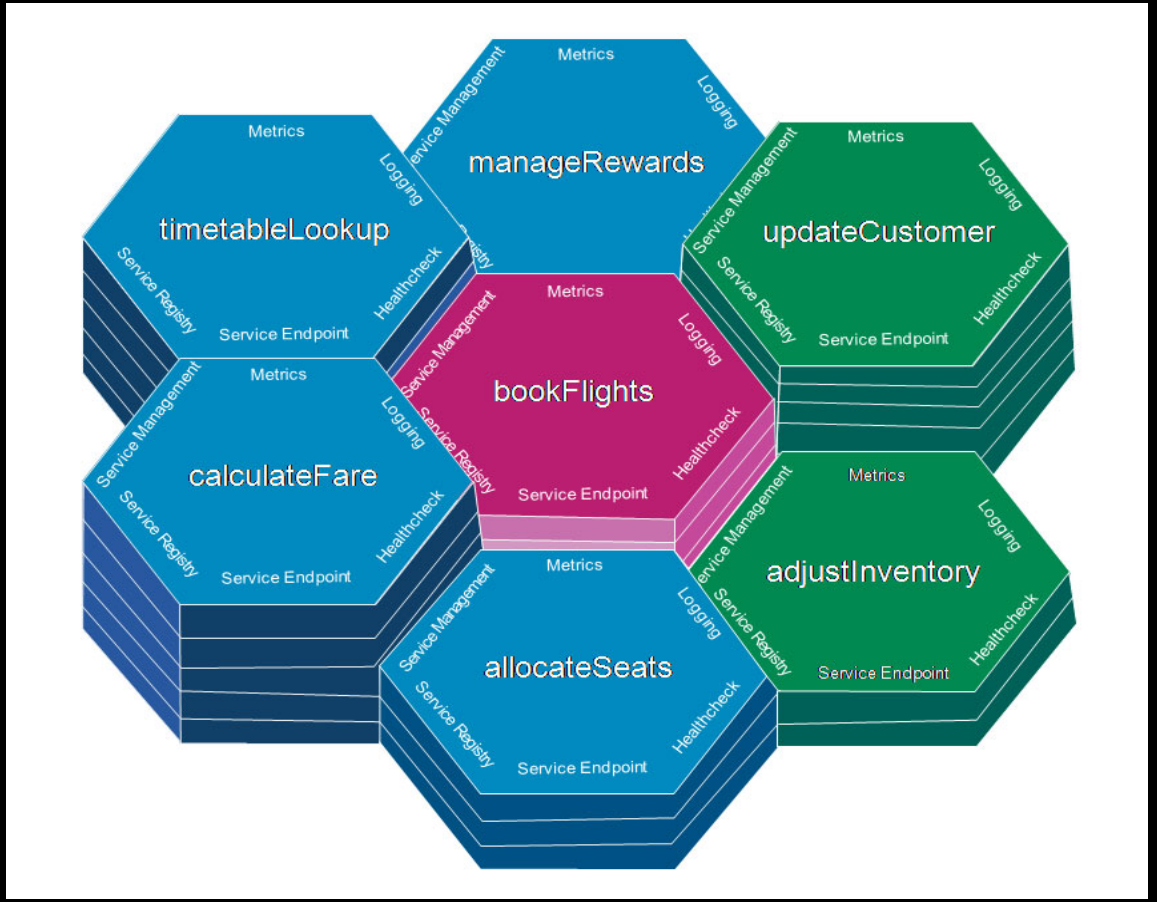


Microservices Architecture
A microservice is a granular, decoupled component within a broader application

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**

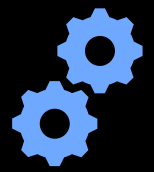
- Each service runs its own process
- There is one service per container

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

Source: IBM Architecture Center

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

Source: IBM Architecture Center

Microservices - principles

One job

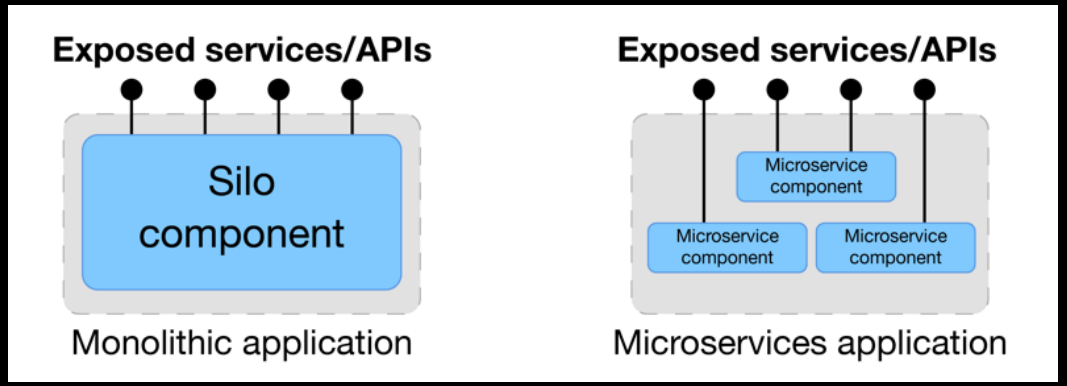
Each microservice must be **optimized for a single function**. Each service is smaller and simpler to write, maintain, and manage. Robert Martin calls this principle the "single responsibility principle."

Separate processes

Communication between microservices must be conducted through **REST API and message brokers**. All communication from service to service must be through the service API or must use an explicit communication pattern, such as the Claim Check Pattern from Hohpe.

Execution scope

Although microservices can expose themselves through APIs, the focus is not on interfaces, but on the running components. The granularity of a microservices application is highlighted in this figure:



Source: IBM Architecture Center

Microservices principles

CI/CD

Each microservice can be continuously integrated (CI) and continuously delivered (CD). When you build a large application that is composed of many services, you soon realize that different services evolve at different rates. If each service has a unique continuous integration or continuous delivery pipeline, the service can proceed at its own pace. In the monolithic approach, different aspects of the system are all released at the speed of the slowest moving part of the system.

Resiliency

You can apply high availability and clustering decisions to each microservice. When you build large systems, another realization that you have is that when it comes to clustering, one size does not fit all. The monolithic approach of scaling all the services in the monolith at the same level can lead to the overuse or underuse of services. Even worse, when shared resources are monopolized, services might be neglected. In a large system, you can deploy services that do not need to scale to a minimum number of servers to conserve resources. Other services require scaling up to large numbers.

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

https://medium.com/@ssriva/7-missing-factors-from-12-factor-application-2a3e1169bd9d

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

Source: https://caylent.com/microservices-good-bad-ugly/

Microservice transformations – The good, the bad, the ugly

The bad

- Added management complexity
- Expertise required to maintain a microservice-based application
- No 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

Source: https://caylent.com/microservices-good-bad-ugly/

Microservice transformations – The good, the bad, the ugly

… and the downright ugly

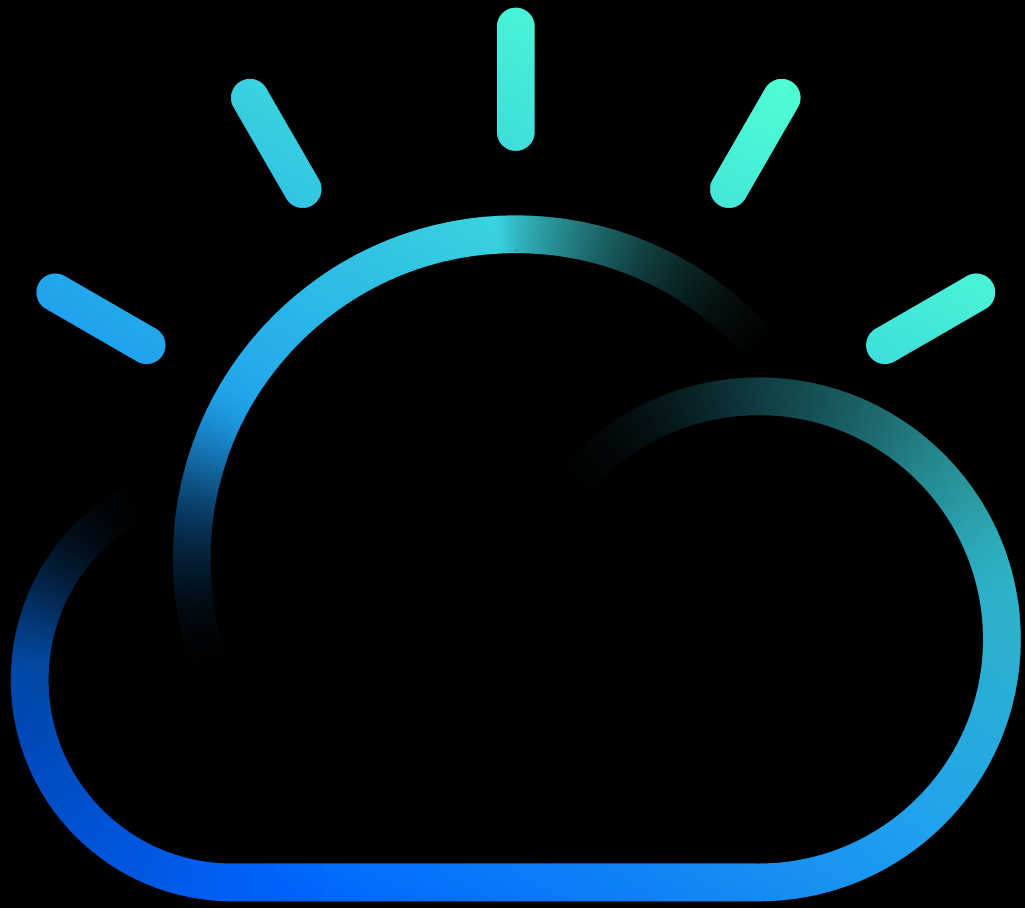
- May require high initial investment to run for Dev, deployment and Ops overhead
- Non-uniform application design and architecture through free choice of technologies
- Overload of documentation for every individual component app
- Higher operational and monitoring costs through shift in paradigms

Source: https://caylent.com/microservices-good-bad-ugly/

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

..but we're getting ahead of ourselves

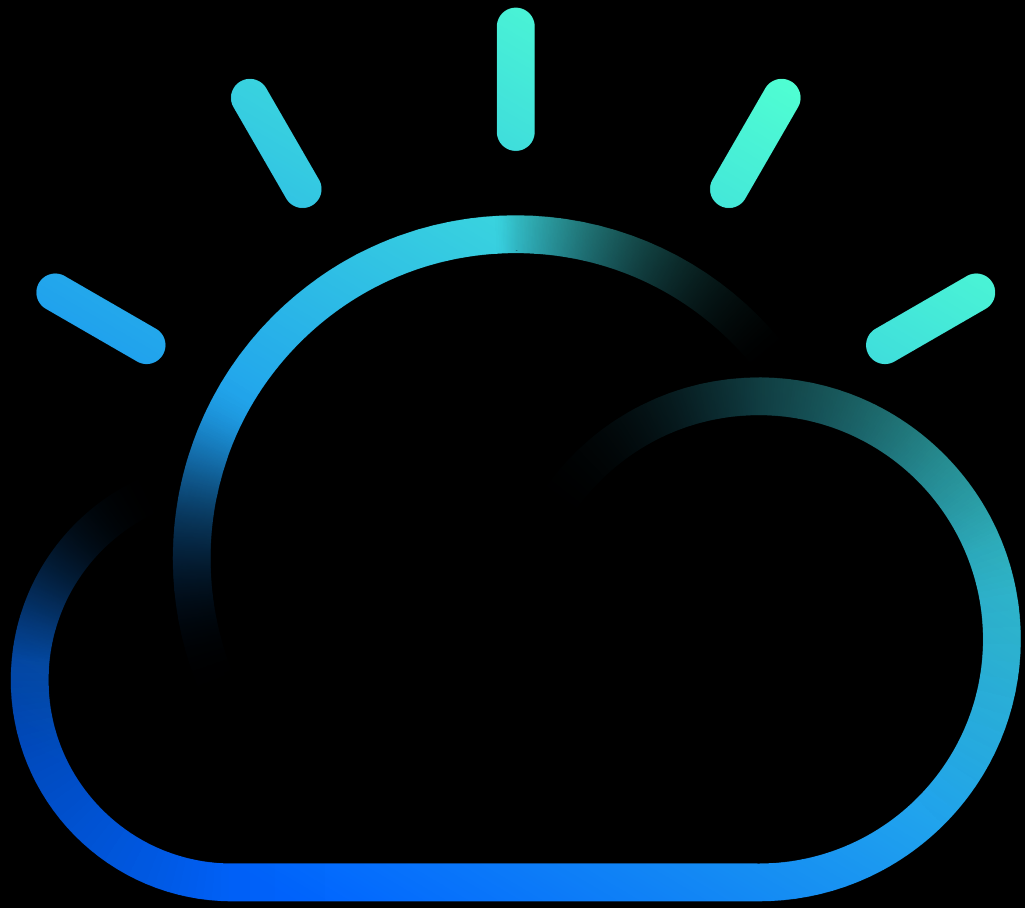
QUESTIONS?



IBM **Cloud**

IBM

The path 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

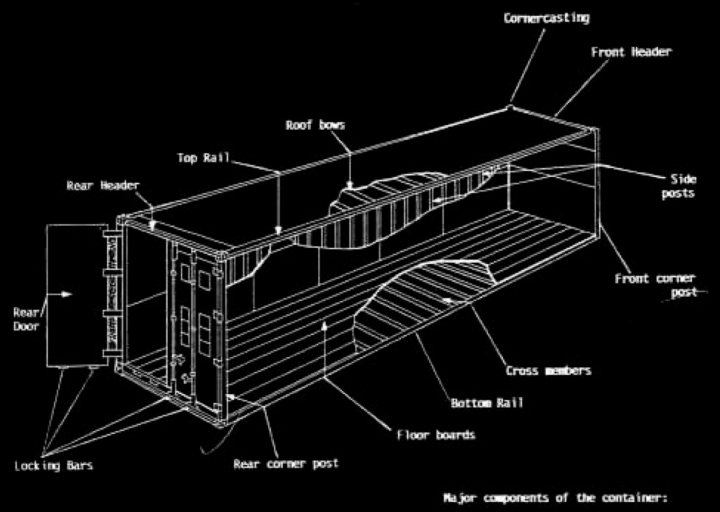
Containers work by **isolating the differences between applications** inside the container so that everything outside the container can be standardized

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



Major components of the container:

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

1979
chroot

2007
cgroups

2008
LXC

2011
warden

2012
Docker

2014
rkt

2016
Garden
runC

2018
containerd

A chroot on Unix operating systems is an operation that changes the apparent root directory for the current running process and its children. A program that is run in such a modified environment cannot name (and therefore normally cannot access) files outside the designated directory tree. ... The modified environment is called a "chroot jail".

https://en.wikipedia.org/wiki/ChrootChr

LXC (Linux Containers) is an operating-system-level virtualization environment for running multiple isolated Linux systems (containers) on a single Linux control host.

...

LXC combines kernel's **cgroups** and support for **isolated namespaces** to provide an isolated environment for applications.

https://en.wikipedia.org/wiki/LXC

https://containerd.io/

Why Containers

Why Should You Care

General Adoption Statistics

- There are 460K Dockerized applications, at 3100% growth in last 2 years
- Over 4 billion containers have been pulled so far
- Docker is supported by a large and fast growing community of contributors and users
- There are over 125K Docker Meetup members worldwide. That is about 40% of the population of Iceland!

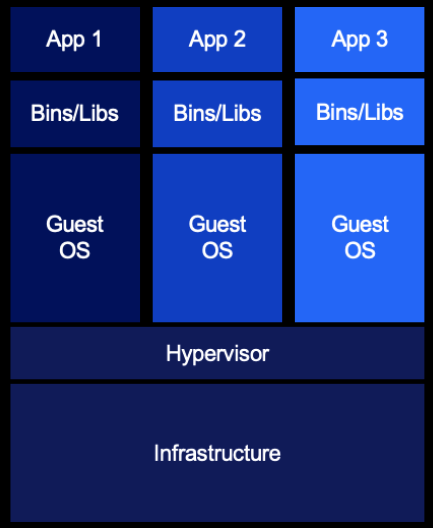
Why Containers

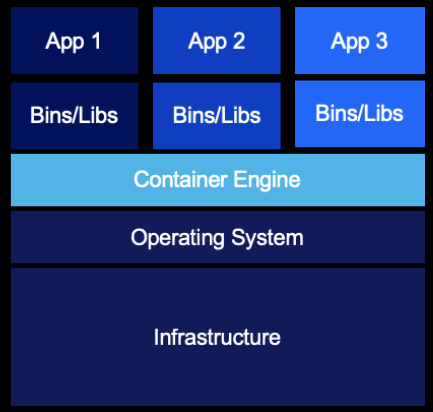
Why Should You Care

Post-Docker Introduction to Organization Statistics

- 30% increase in Docker adoption in first year
- Docker running on 10% of hosts
- Docker is mostly used by large companies with a large number of hosts
- Number of containers running in production quintuples (=5X) 9 months after initial deployment
- Average container lifespan decreases from 13 hours to 9.25 hours
- PHP, Ruby, Java and Node are the main programing frameworks used in containers

VMs vs. Containers

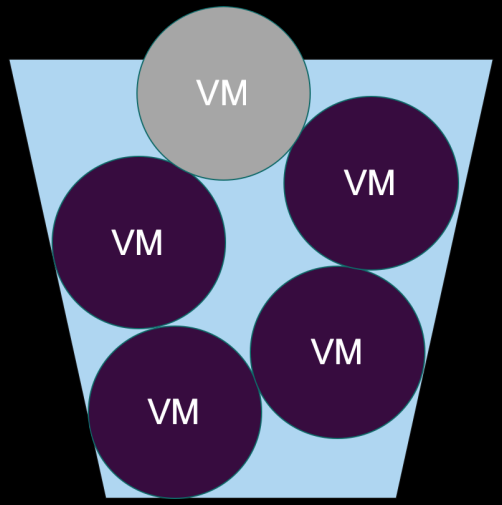


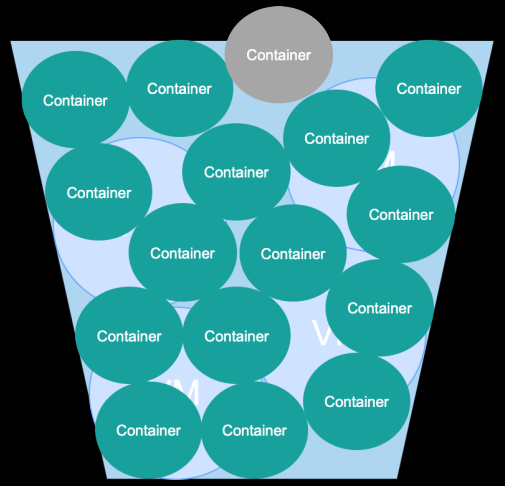


+ VM Isolation
- Complete OS
- Static Compute
- Static Memory

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

VMs vs. Containers





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

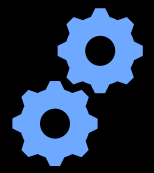
+ Container Isolation
+ Shared Kernel
+ Burstable Compute
+ Burstable Memory
+ High Resource Utilization

VM vs. Containers

Features	Virtual Machines (VMs)	Application Containers	Application Container Benefits
Virtualization	In VMs, the virtualization occurs in server hardware.	In containers, the virtualization occurs at the OS level.	Faster time-to-market
Standardization	VMs are standardized systems that have capabilities similar to that of bare metal computers.	Containers are not standardized, as their kernel OS has varying degrees of complexity.	Makes applications more portable by isolating them from the underlying infrastructure
Resource management	Each VM has its own kernel OS, binary, and library.	Containers share the same host OS, and if needed, both binary and library.	Lower overhead costs
Density	VMs require a few gigabytes (GB) of space that limits them into a single server.	Containers require a few megabytes (MB) of space that enables many containers to fit into a single server.	Lighter weight than VMs
Boot time	Minutes	Seconds	Easy to update and release newer versions of applications

Advantages of Containers

Containers are **portable**

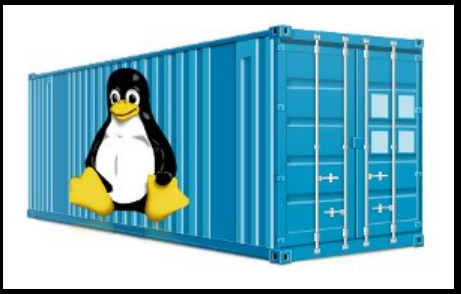


Containers are **easy to manage**

Containers provide **“just enough” isolation**

Containers use hardware **more efficiently**

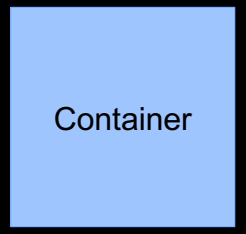
Containers are **immutable**



Docker Components

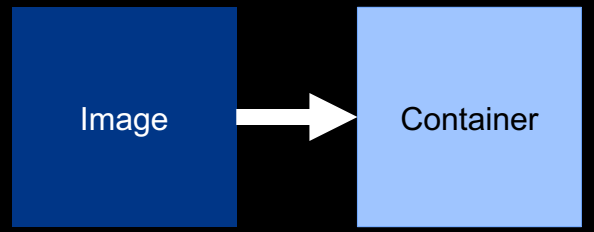
Container

Smallest compute unit



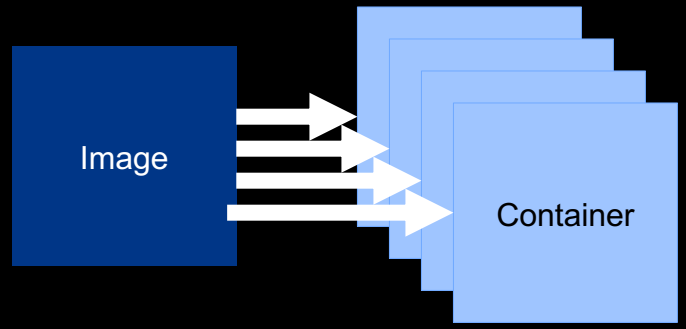
Docker Components

Containers
are created from
Images



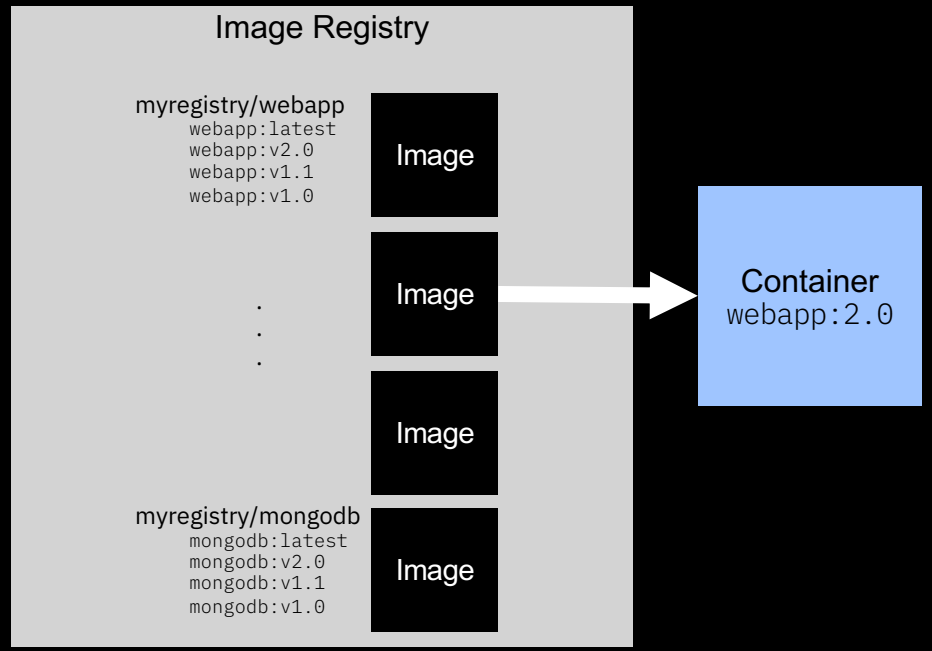
Docker Components

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

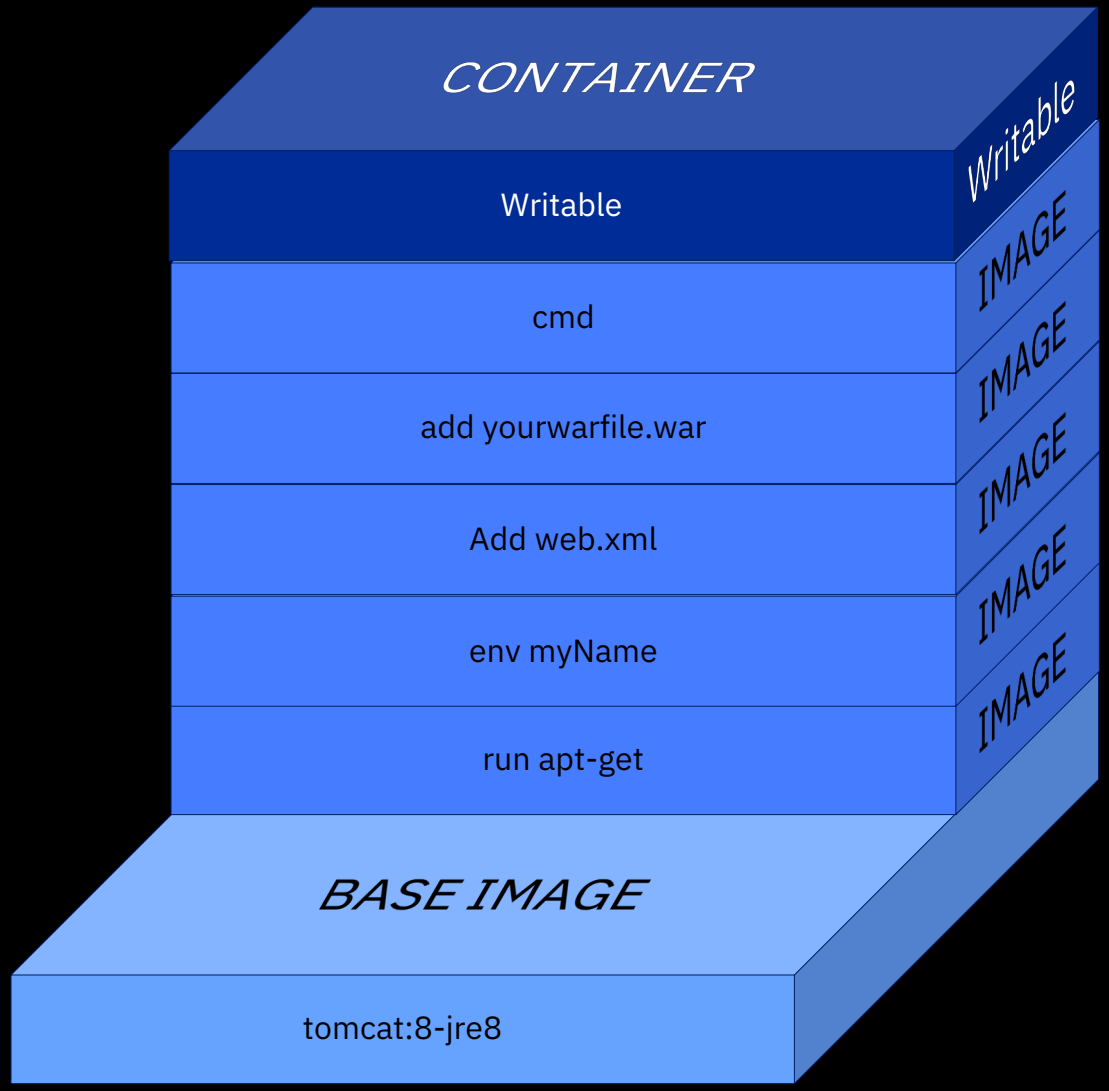
Why Containers

Docker Layers

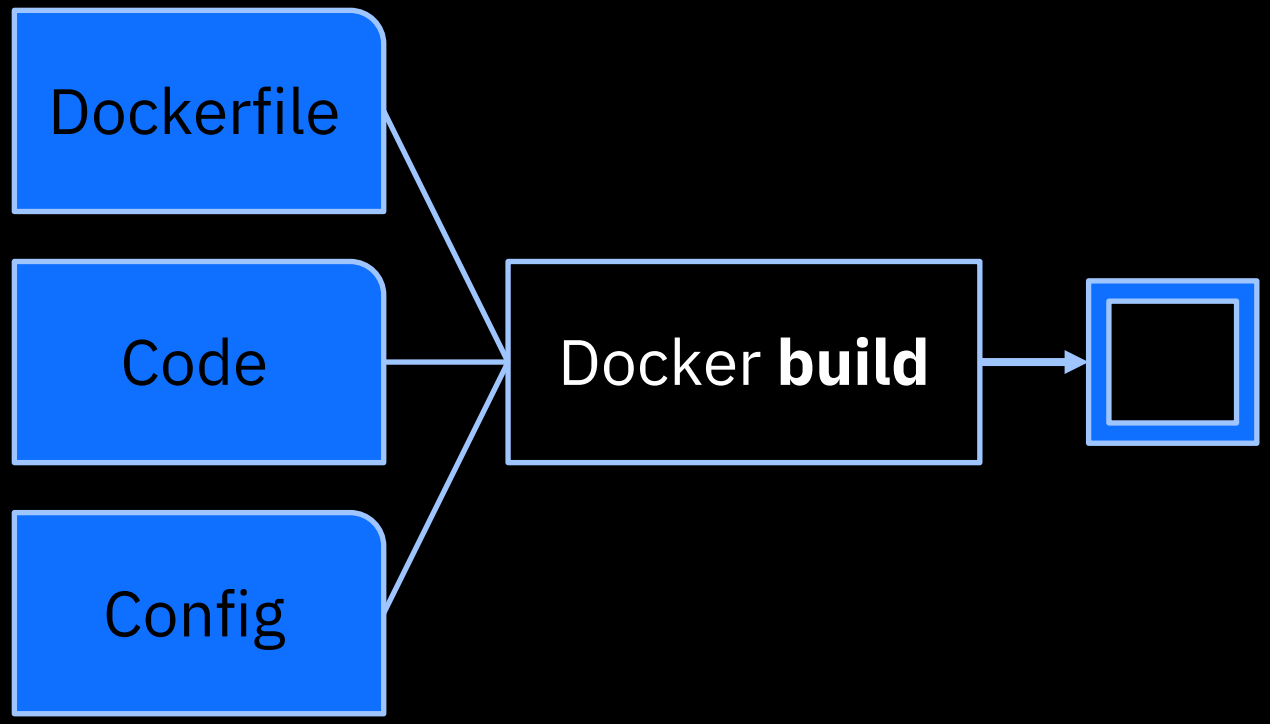
Inheritance

- 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"]
```



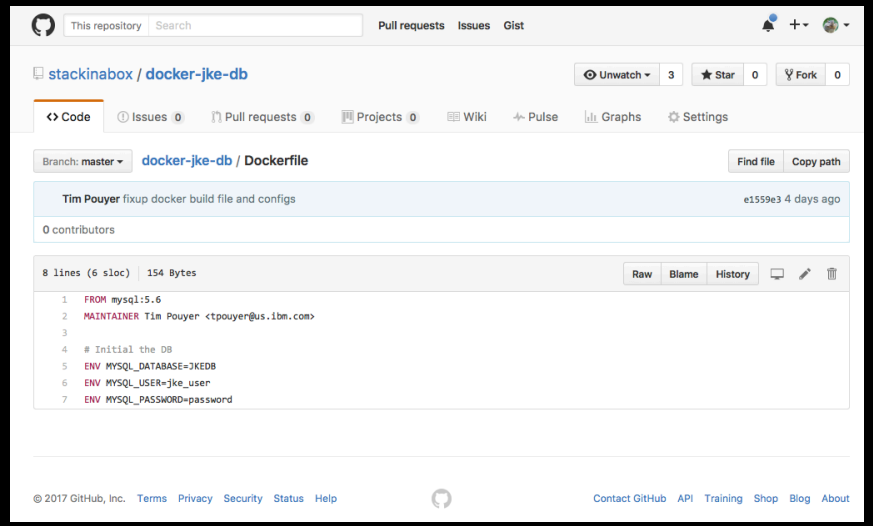
Docker Basics – Build



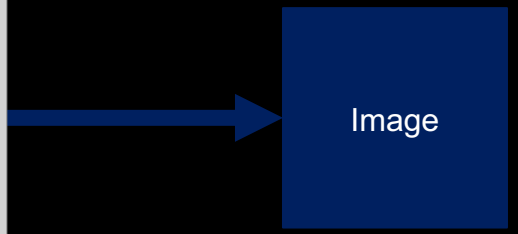
Docker Basics - Build

Dockerfile

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




```
# 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

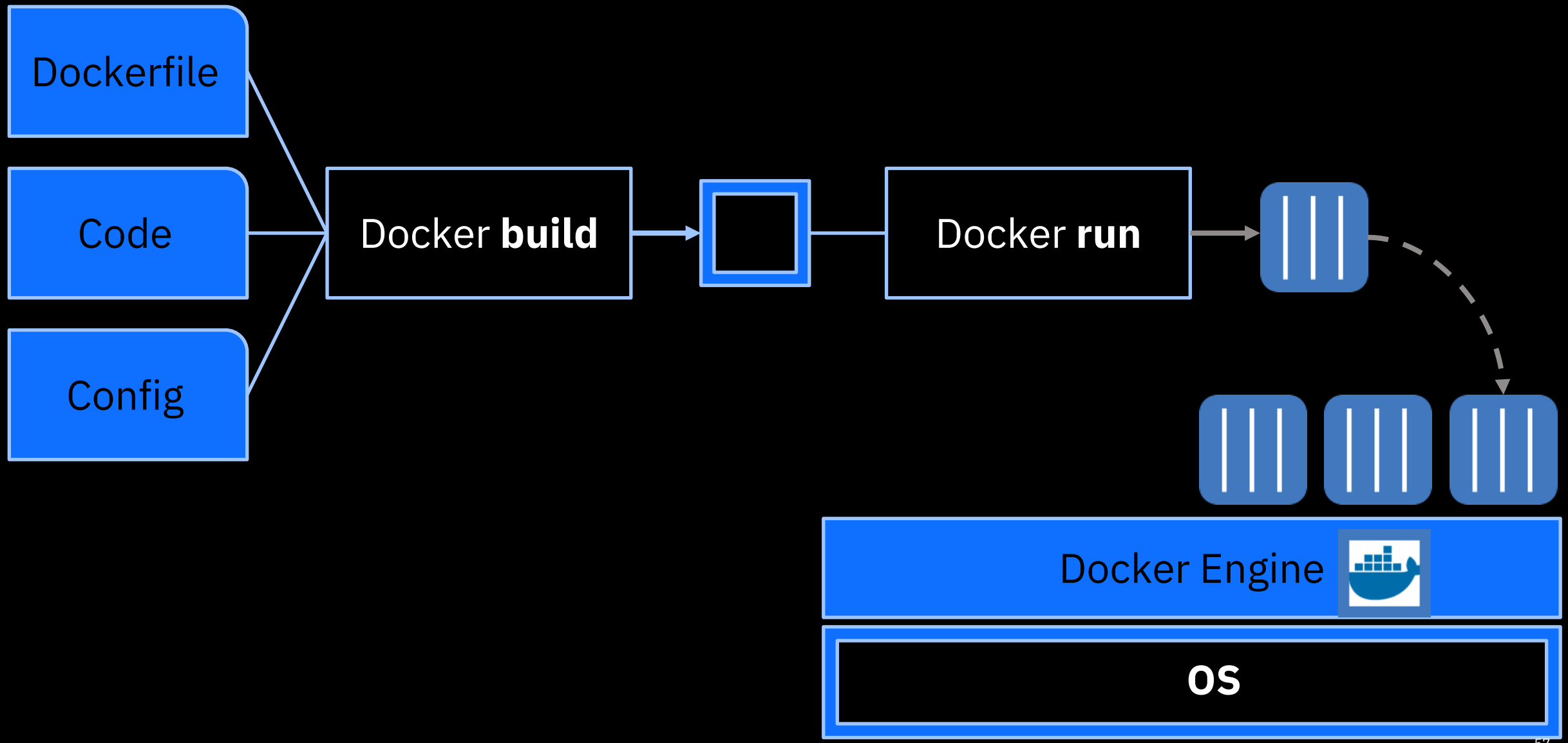
- A text file that builds an image using Docker directives

- FROM
- RUN
- COPY
- ENTRYPOINT
- LABEL
- ENV
- ARG
- USER
- EXPOSE

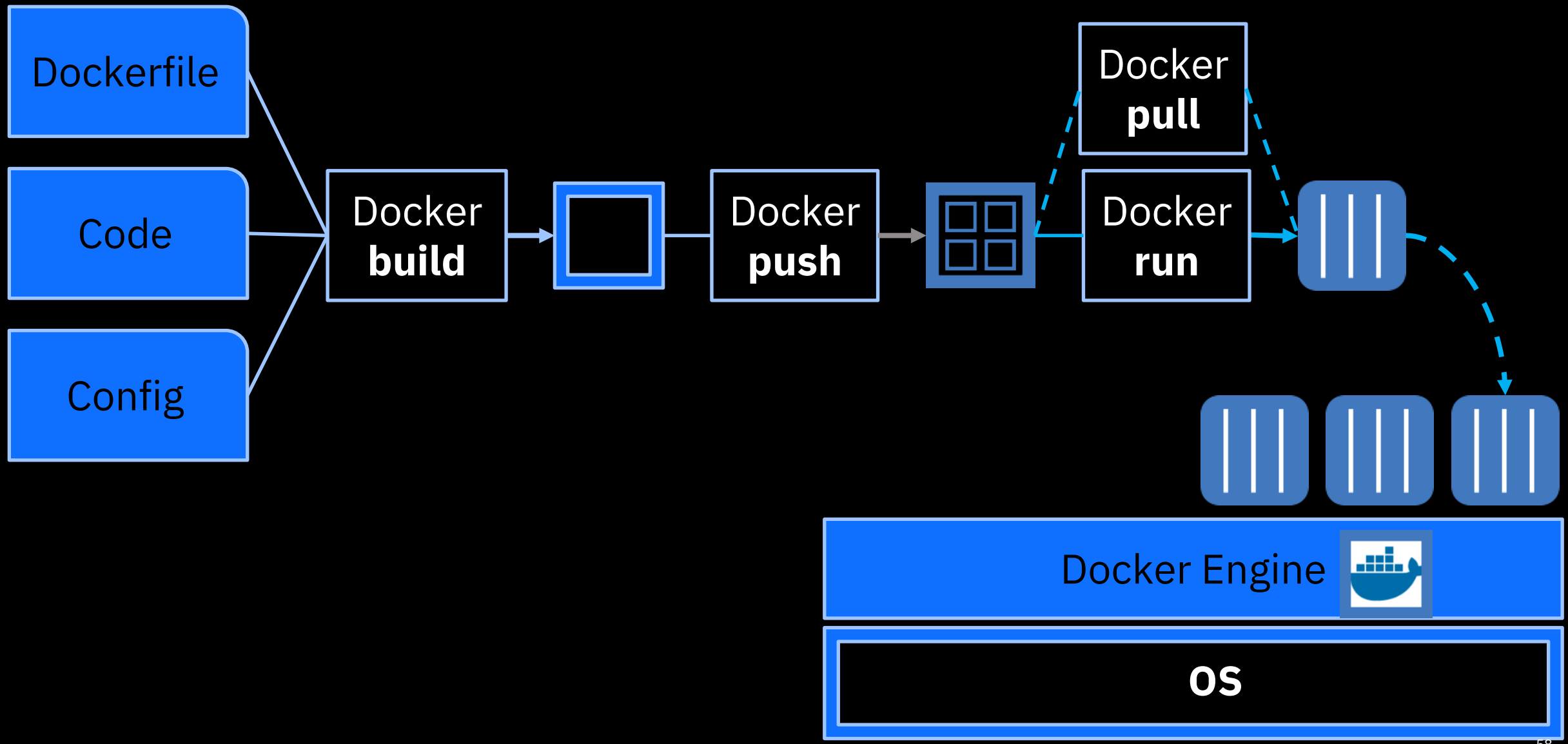
```
# Pull base image
FROM tomcat:8-jre8

# 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 build –t myimage ./Dockerfile
```

Docker Basics – Run



Docker Basics – Store, Retrieve & Run with registry

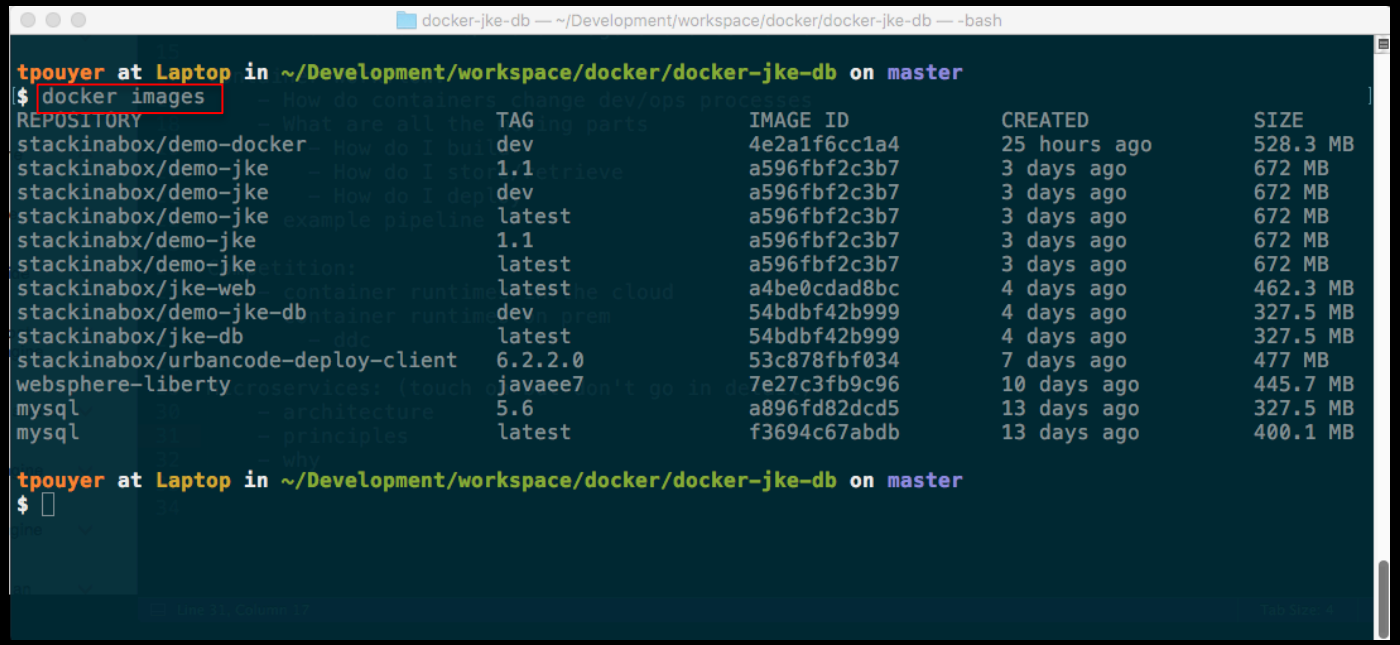


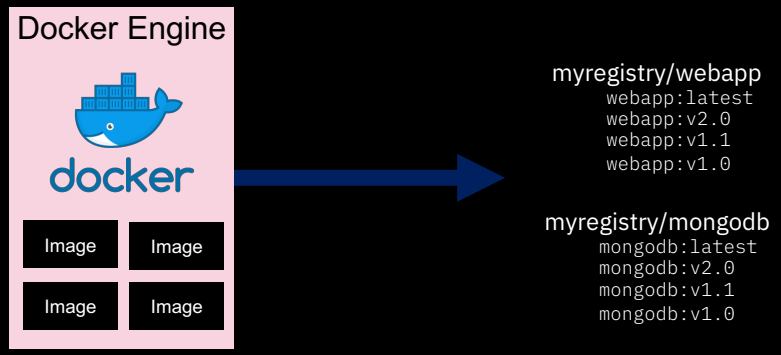
Docker Basics – Store & Retrieve

Docker Registry

- Private Local
- Public Docker Hub
- Private Shared

```
docker images
```





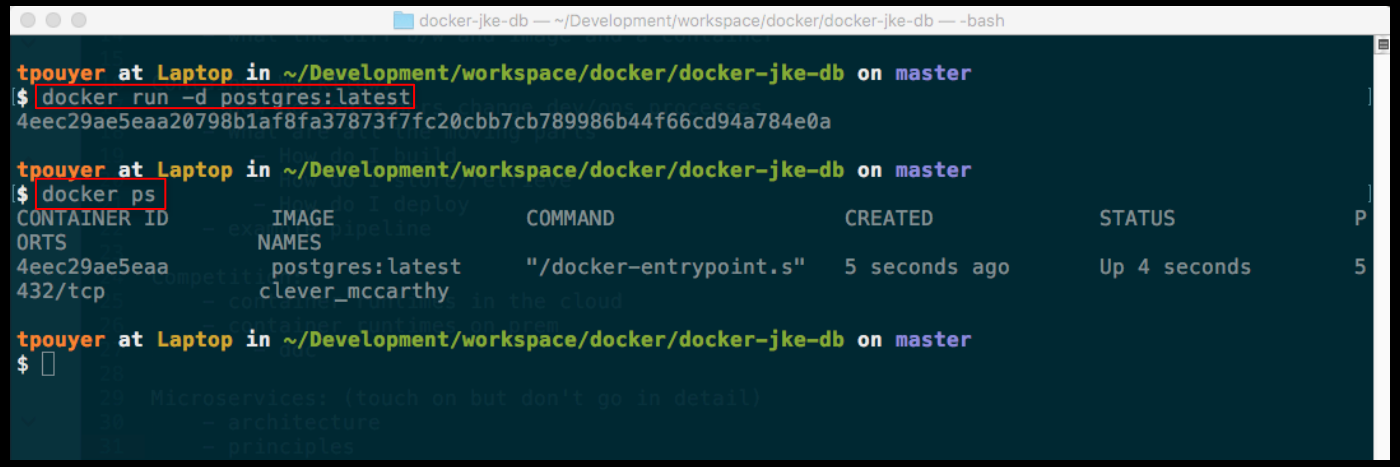
myregistry/webapp
webapp:latest
webapp:v2.0
webapp:v1.1
webapp:v1.0

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

Docker Basics – Run

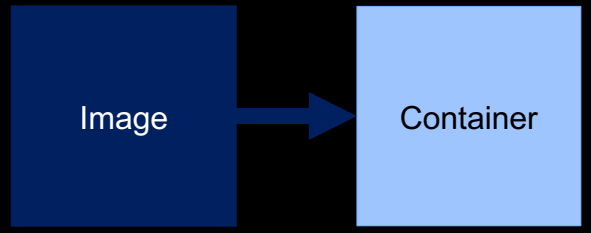
Docker run

- Local (containerd)




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

…



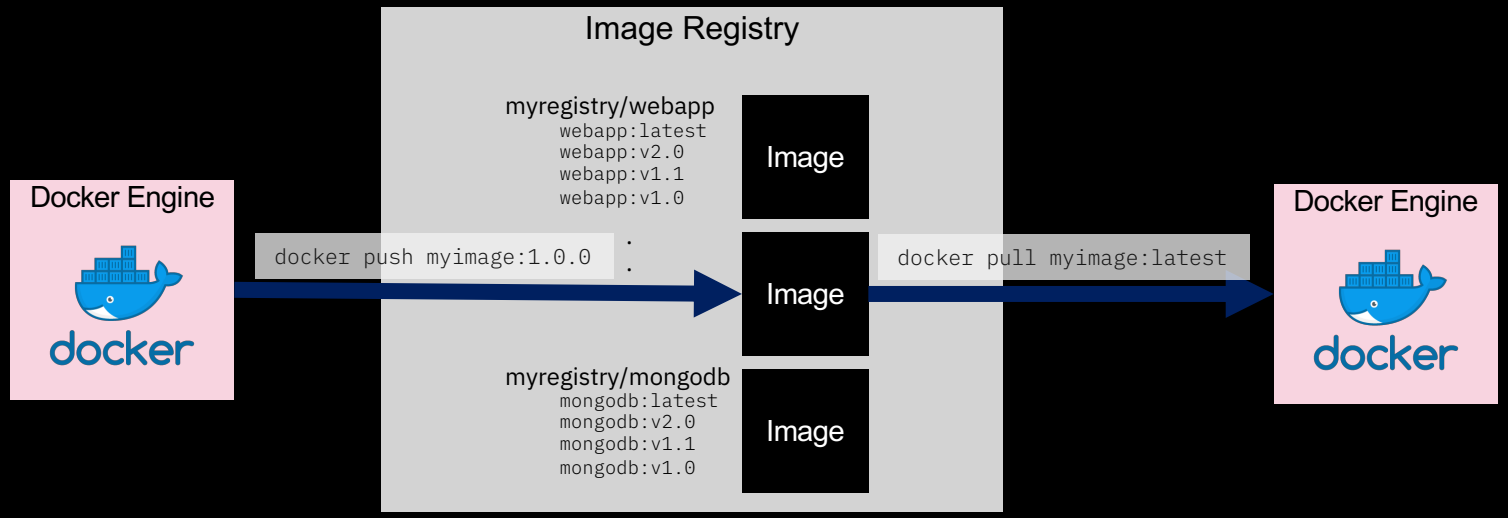
Docker Basics – Store & Retrieve with registry

Docker Registry

- Private Local
- Public Docker Hub
- Private Shared

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

```
docker-jke-db — ~/Development/workspace/docker/docker-jke-db — -bash
tpouyer at Laptop in ~/Development/workspace/docker/docker-jke-db on master
$ docker pull postgres:latest
latest: Pulling from library/postgres

5040bd298390: Already exists
f08454c3c700: Pull complete
4db038cdfe03: Pull complete
e1d9ba315f03: Pull complete
25e0ee93170e: Pull complete
3f28084c3f51: Pull complete
78c91f0aedcd: Pull complete
93ab52dbcbb8: Pull complete
27ec75825613: Pull complete
28ef691a9920: Pull complete
0f0dd20755c9: Pull complete
2a4a824861f7: Pull complete
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**

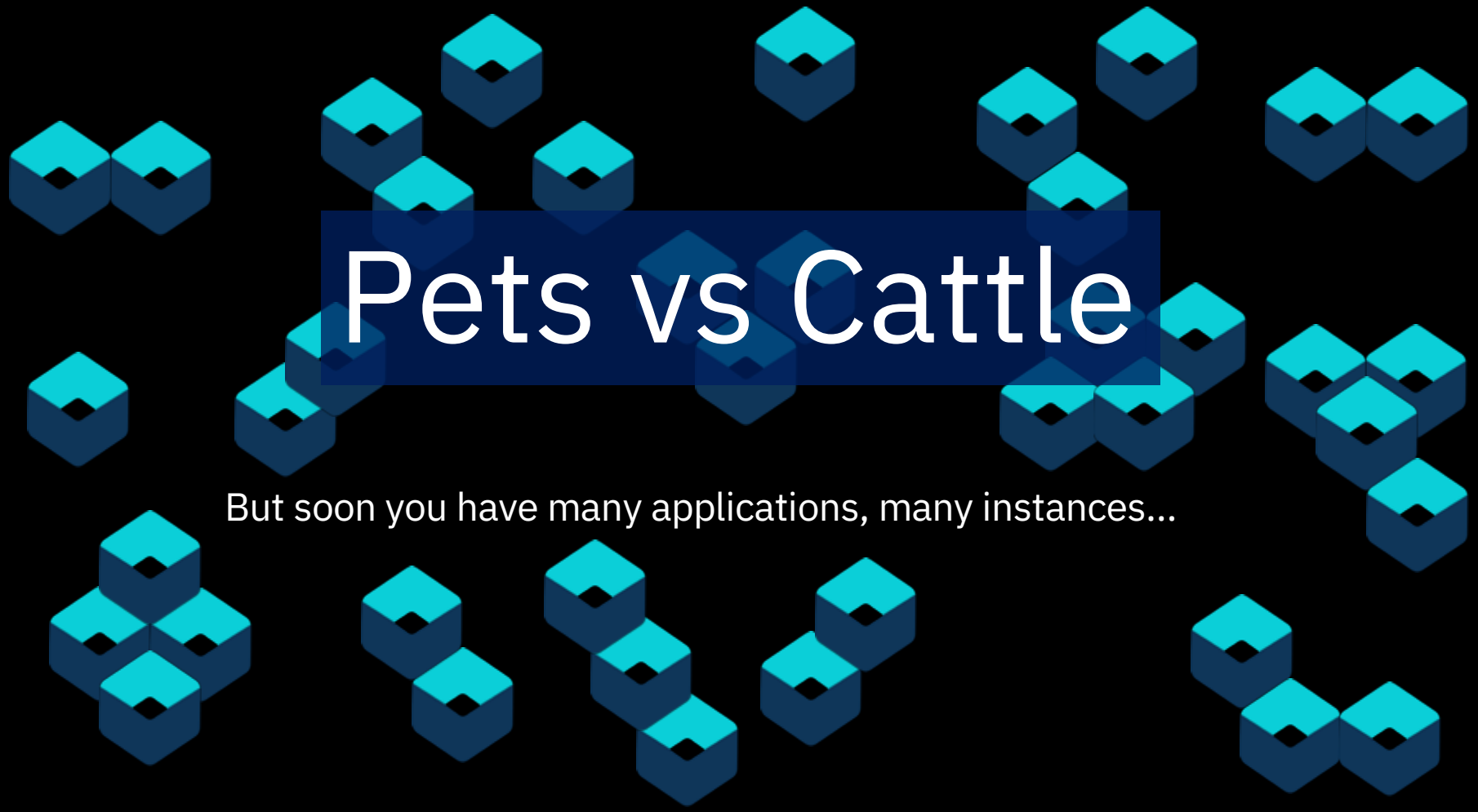
Login into the registry

- Docker login domain:port

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

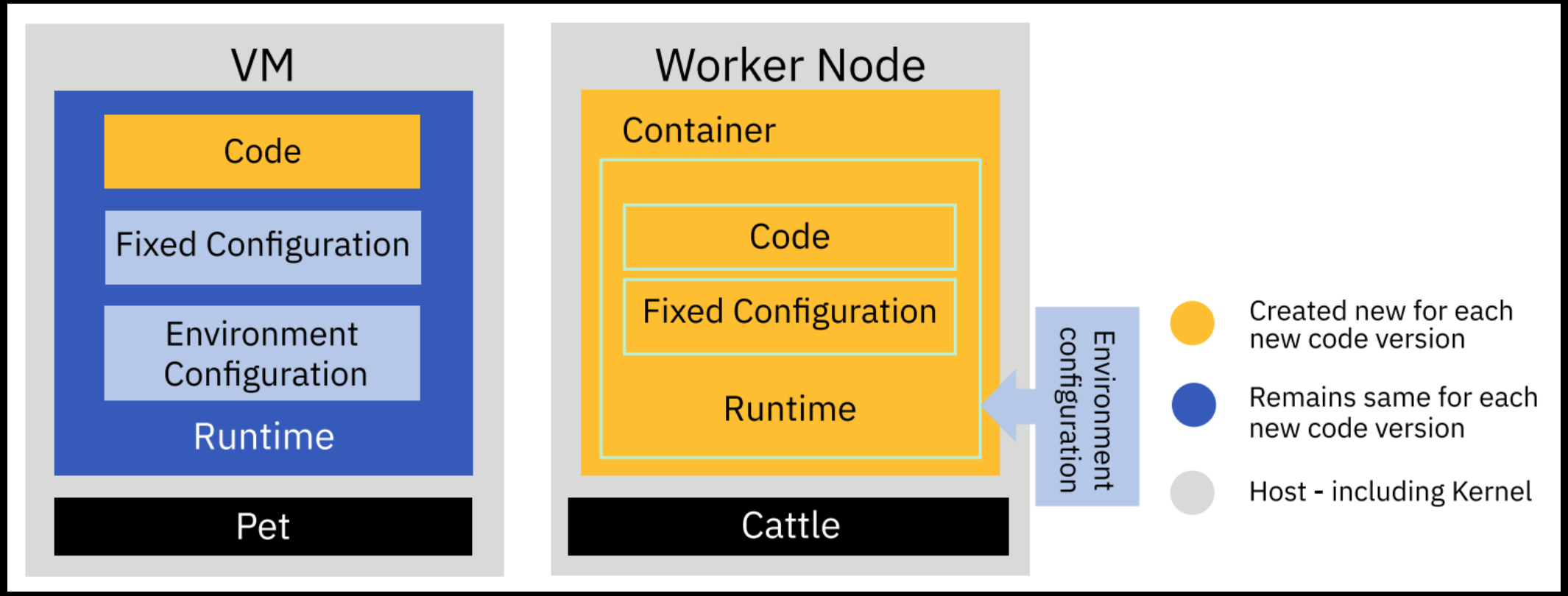
At first the growth is easy to handle....

Pets vs Cattle



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

Pets vs Cattle



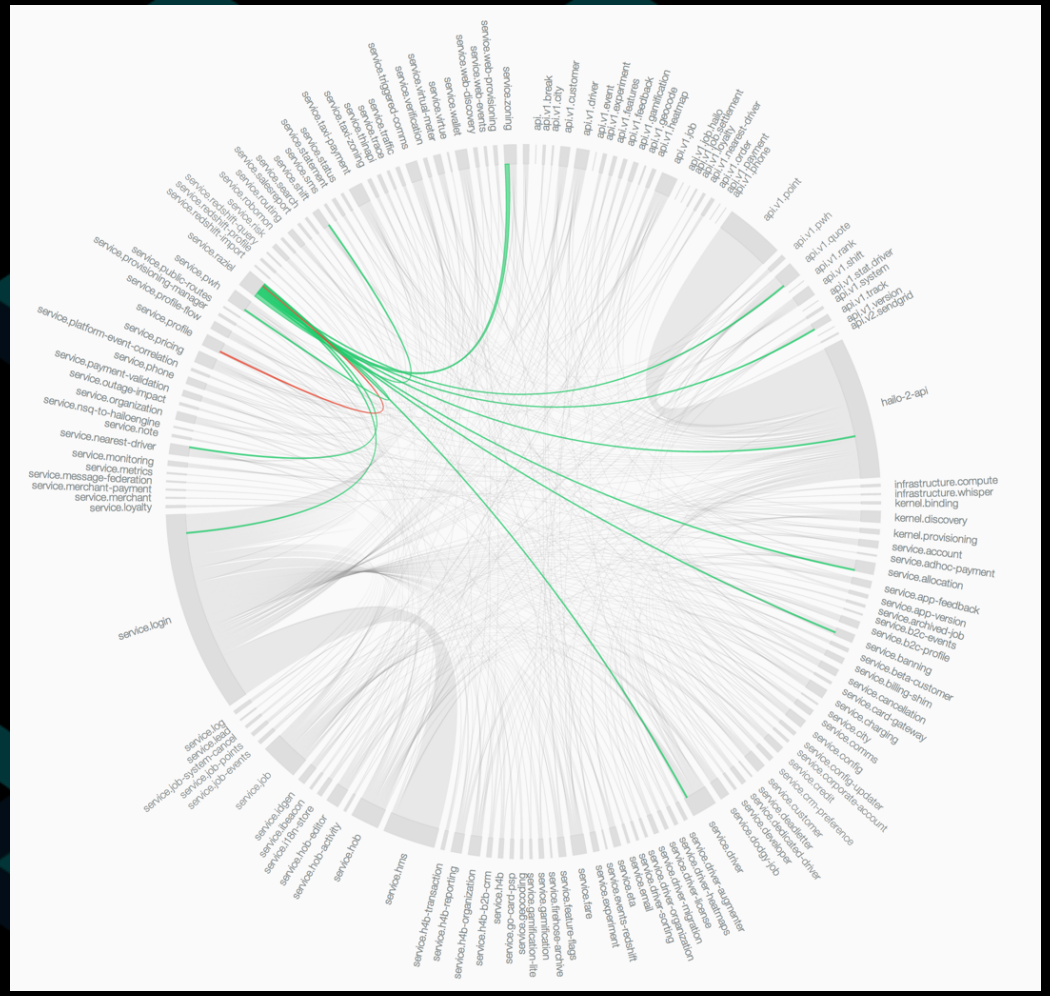
https://developer.ibm.com/tutorials/get-started-with-agile-integration-architecture-integration-modernization-in-action/

The trade off

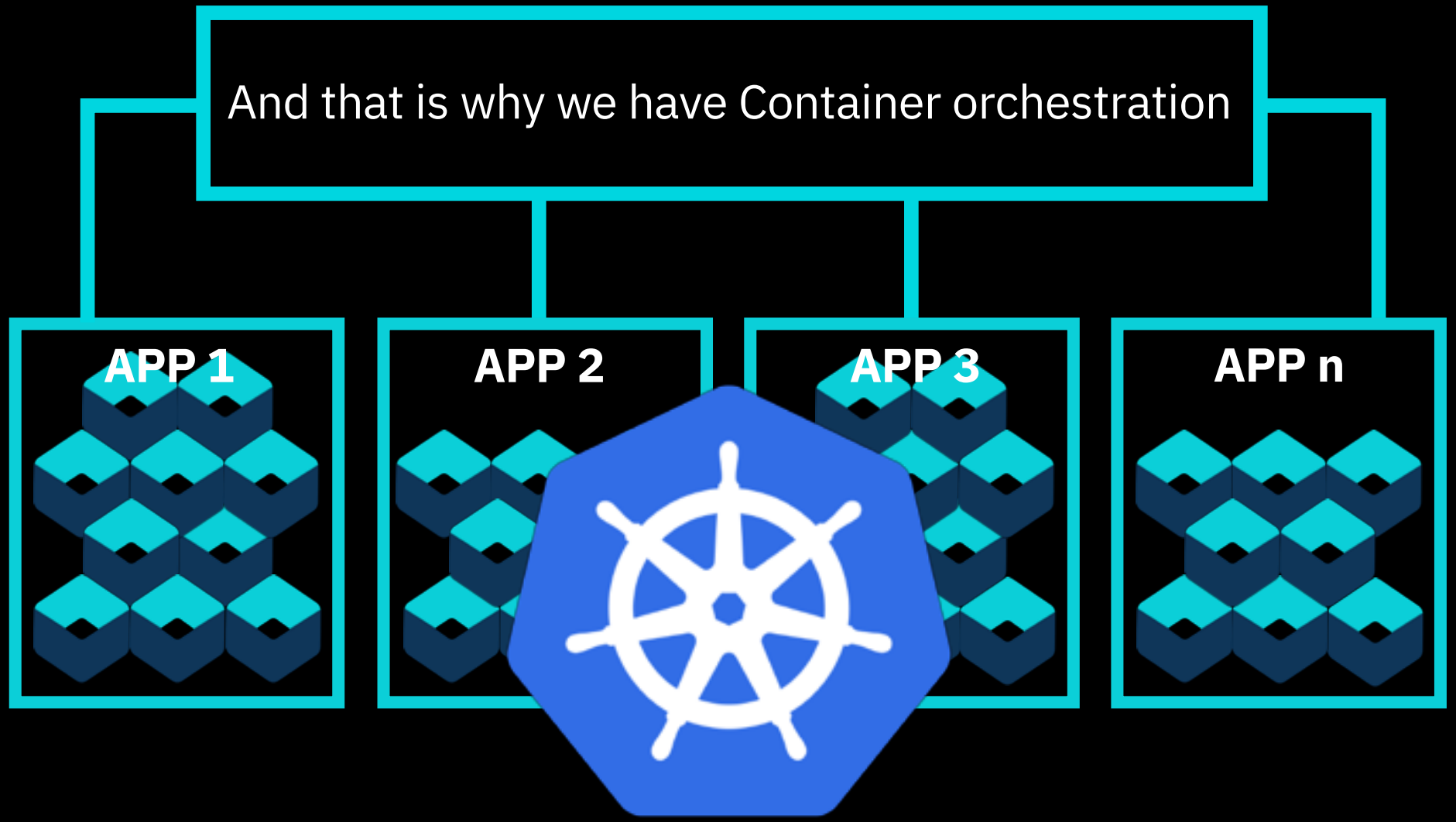
Improved delivery velocity

in exchange for

increased operational complexity



Enter Kubernetes (K8s)



Seeded with runC from Docker

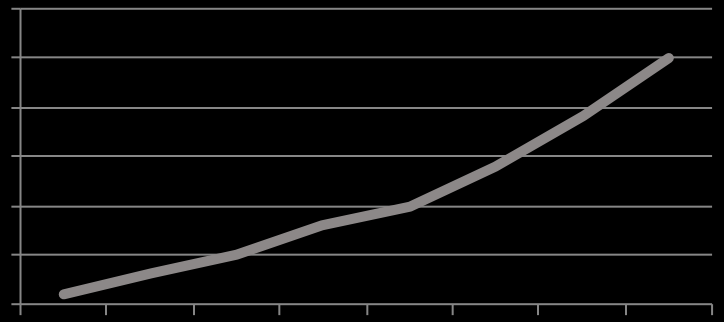
IBM Contributions to Docker

1 upstream contributor in August 2015 has grown to 25 today

IBM now has 2 core maintainers in the Docker engine

IBM has contributions in:

- Docker engine
- Compose
- Machine
- Swarm
- Distribution (Docker registry)
- libnetwork
- libcontainer (prior to OCI)
- Opencontainers/runC
- Docker-py

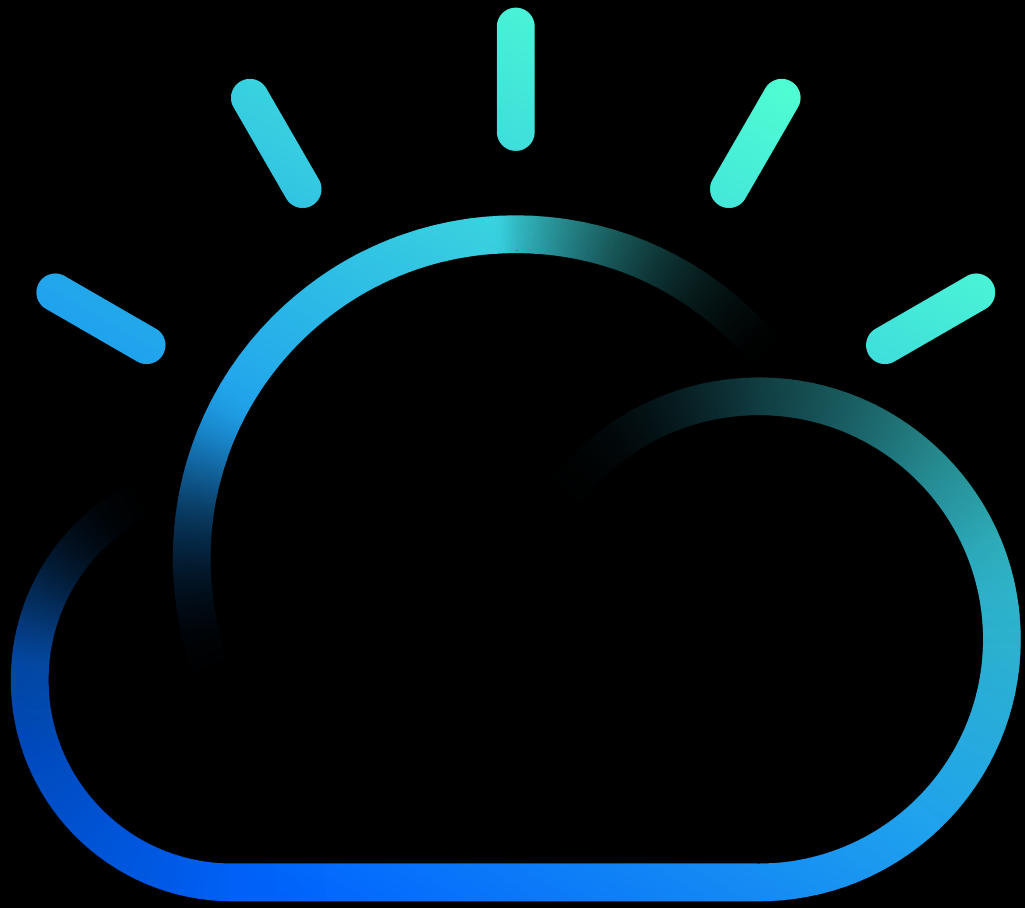


- #3 in all-time non-Docker employee commits
- Contributed SoftLayer, Object Storage, Bluemix integrations to Docker ecosystem
- Enabling other (non-x86) architectures for Docker engine, registry, and CI workflow

Docker Recap

- **Containers are not VMs**
- **Containers provide many benefits:**
  - Efficiency
  - Portability
  - Consistency
- **New challenges with containers:**
  - Production apps dependent on open-source projects
  - Existing tools may not be sufficient for container
  - Need to focus on business objectives

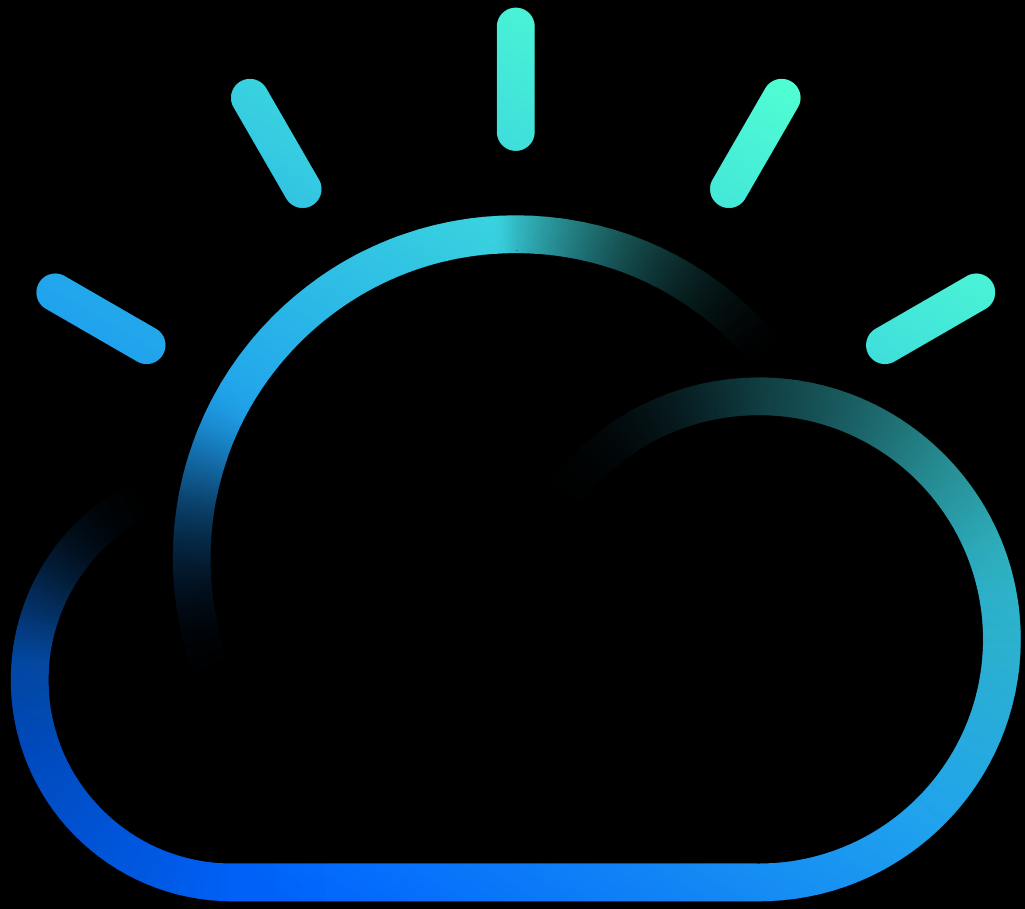
QUESTIONS?



IBM **Cloud**

IBM

The path 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.

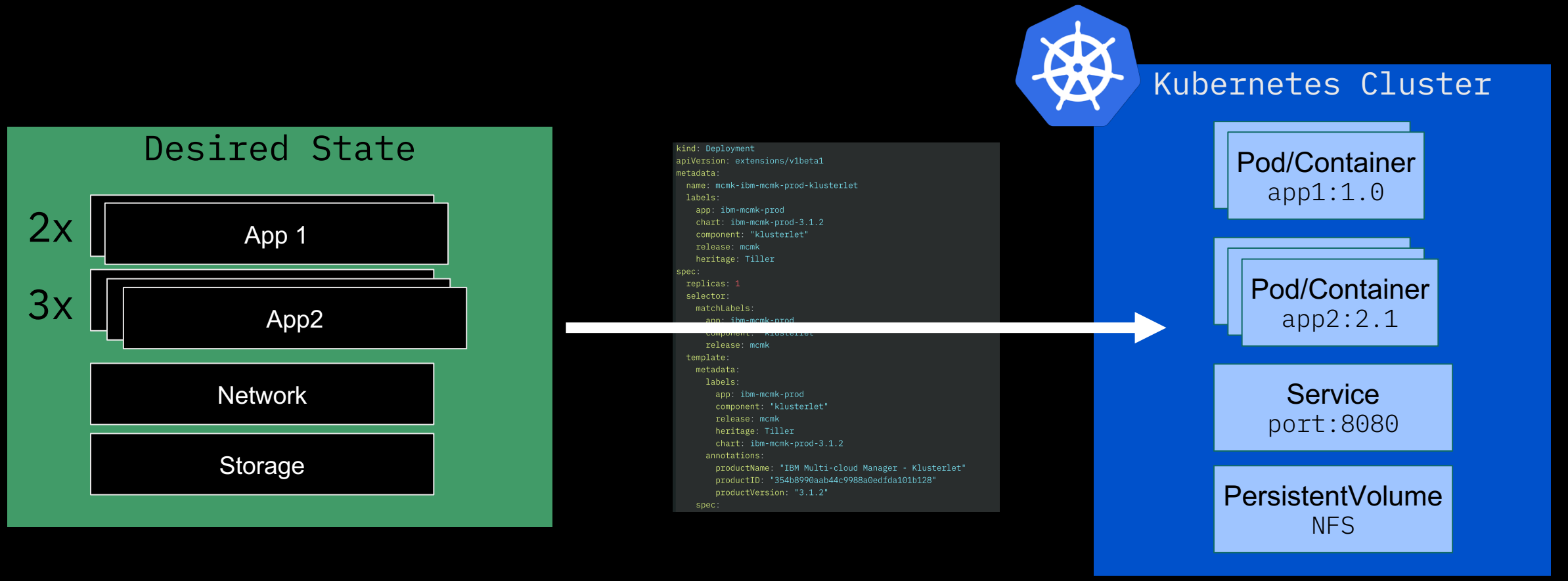
Declarative Systems

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.

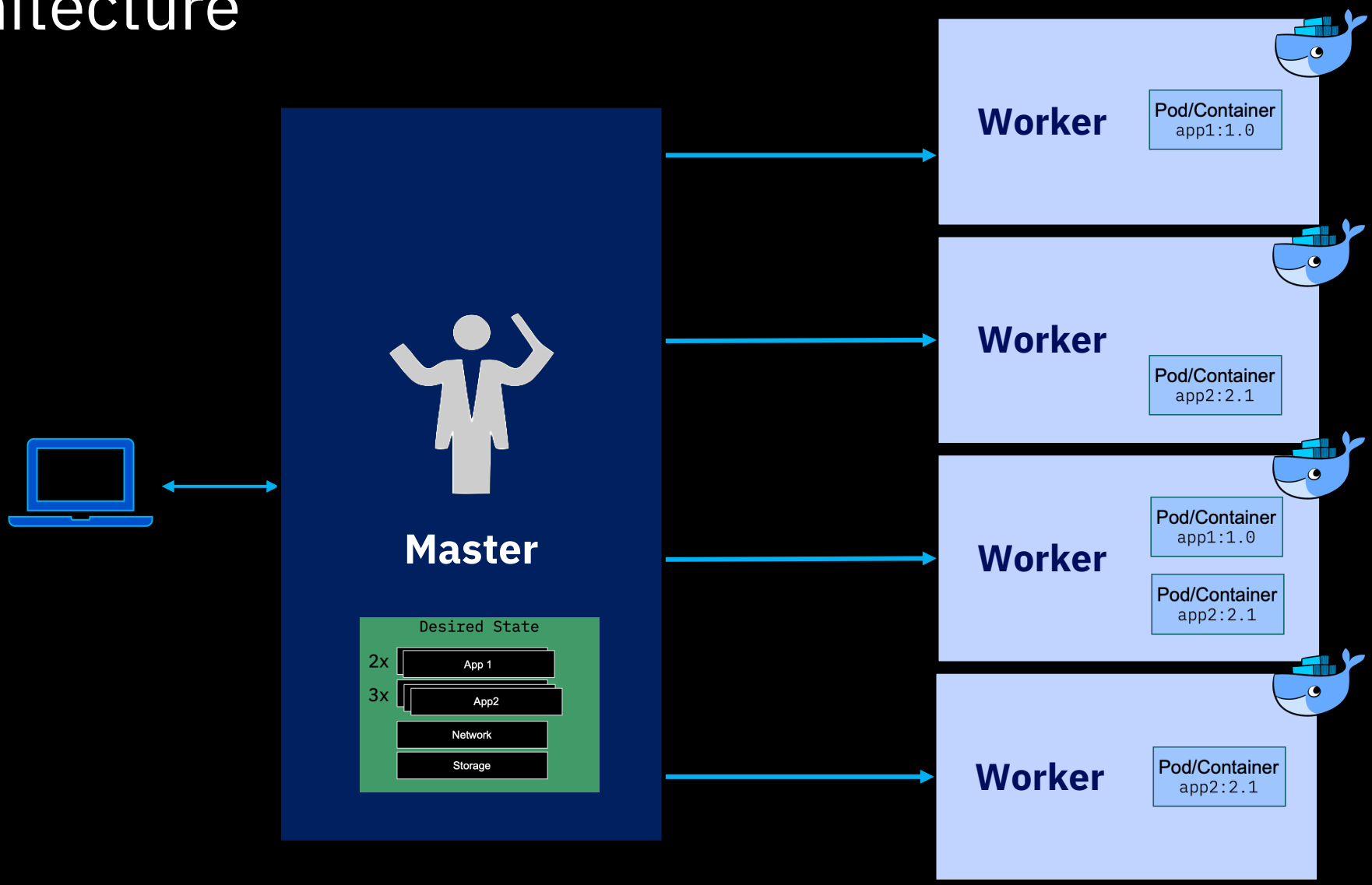
Kubernetes – Declarative System

The Desired State

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



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**

What do Kubernetes really offer ?

Intelligent Scheduling

Automatically places containers based on their resource requirements and other constraints, while not sacrificing availability. Mix critical and best-effort workloads in order to drive up utilization and save even more resources.

Self Healing

Restarts containers that fail, replaces and reschedules containers when nodes die, kills containers that don't respond to your user-defined health check, and doesn't advertise them to clients until they are ready to serve.

Horizontal Scaling

Scale your application up and down with a simple command, with a UI, or automatically based on CPU usage.

Service Discovery and Load Balancing

No need to modify your application to use an unfamiliar service discovery mechanism. Kubernetes gives containers their own IP addresses and a single DNS name for a set of containers, and can load-balance across them.

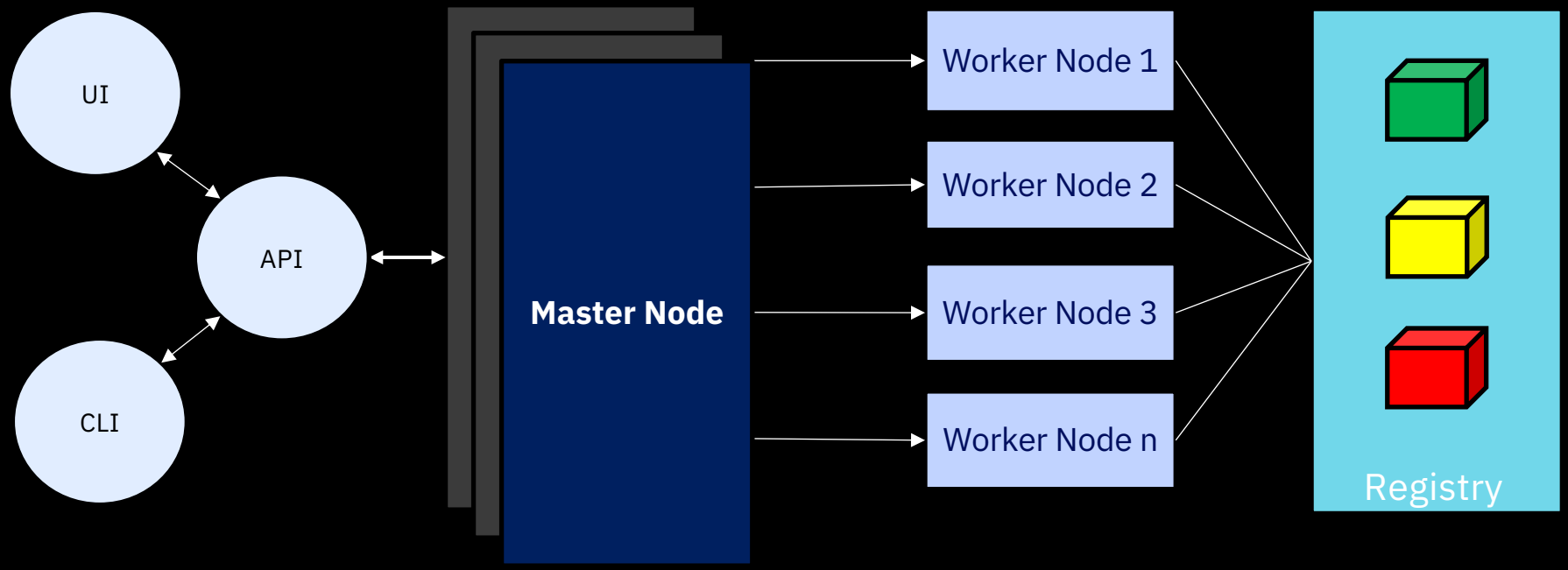
Automated rollout and rollback

Kubernetes progressively rolls out changes to your application, while monitoring application health to ensure it doesn't kill all your instances at the same time. If something goes wrong, Kubernetes will rollback the change for you. Take advantage of a growing ecosystem of deployment solutions.

Secret and configuration management

Deploy and update secrets and application configuration without rebuilding your image and without exposing secrets in your stack configuration.

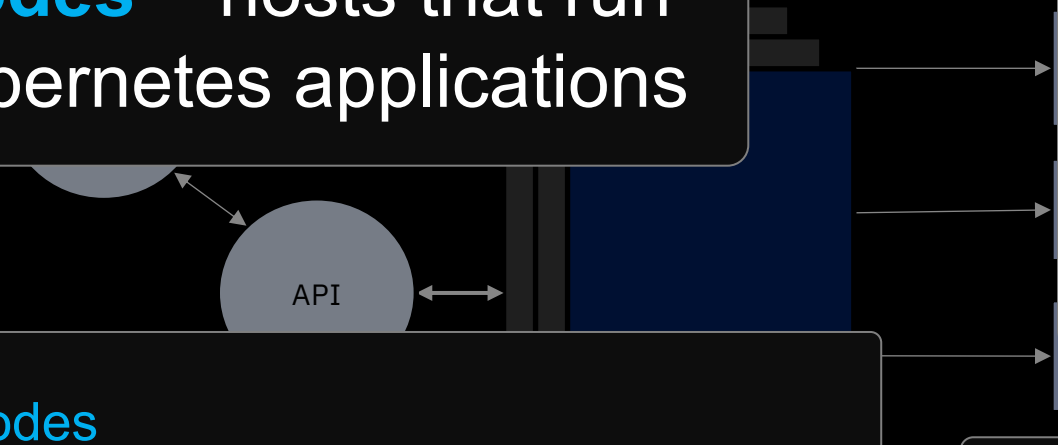
Kubernetes Management Architecture



- Etcd
- API Server
- Controller Manager Server
- Scheduler

Kubernetes Management Architecture

Nodes – hosts that run Kubernetes applications



Worker nodes
- Hosts the K8s services
- Kubelet (K8s agent that accepts commands from the master)
- Kubeproxy (network proxy service responsible for routing activities)
- Docker host

Master nodes
- Controls and manages the cluster
- Kubectl (command line)
- REST API (communication with workers)
- Scheduling and replication logic

- Controller Manager Server
- Scheduler

Proxy nodes
- Transmits external request to the services created inside your cluster

Management nodes
- Hosts management services such as monitoring, metering, and logging

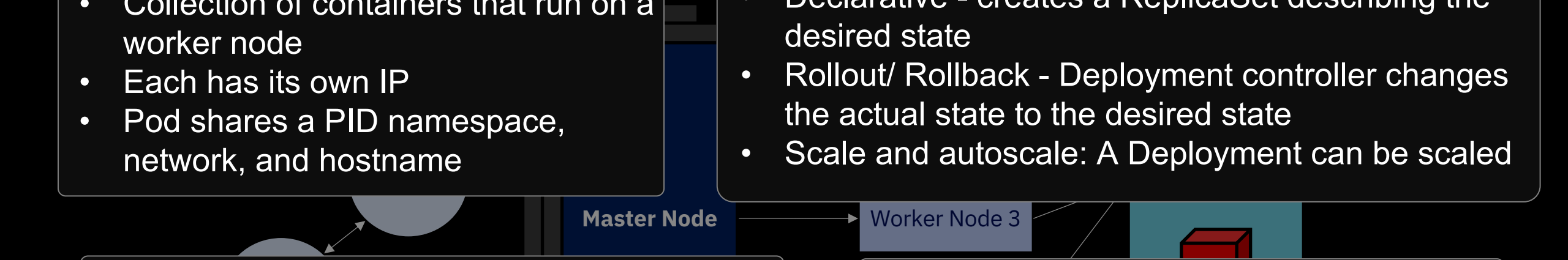
Kubernetes Management Architecture

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

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



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

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

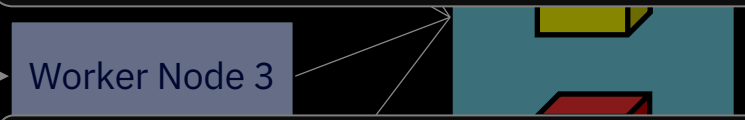
- Configuration values to be used by containers in a pod
- Stores configuration outside of the container image, making containers more reusable

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

- Etcd
- API Server
- Controller Manager Server

API Server – Kubernetes API server

Scheduler – schedules pods in worker nodes

What is container orchestration?

Container orchestration
- Manages the deployment, placement, and lifecycle of workload containers

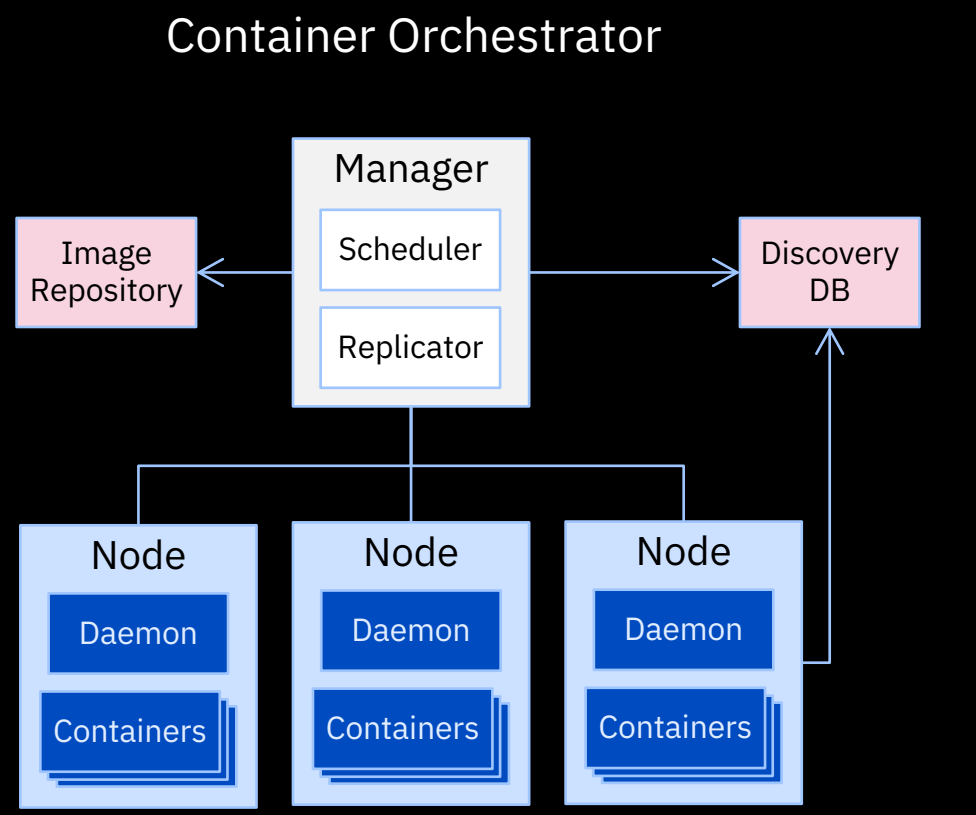
Cluster management
- Federates multiple hosts into one target

Scheduling
- Distributes containers across nodes

Service discovery
- Knows where the containers are located
- Distributes client requests across the containers

Replication
- Ensures the right number of nodes and containers

Health management
- Replaces unhealthy containers and nodes



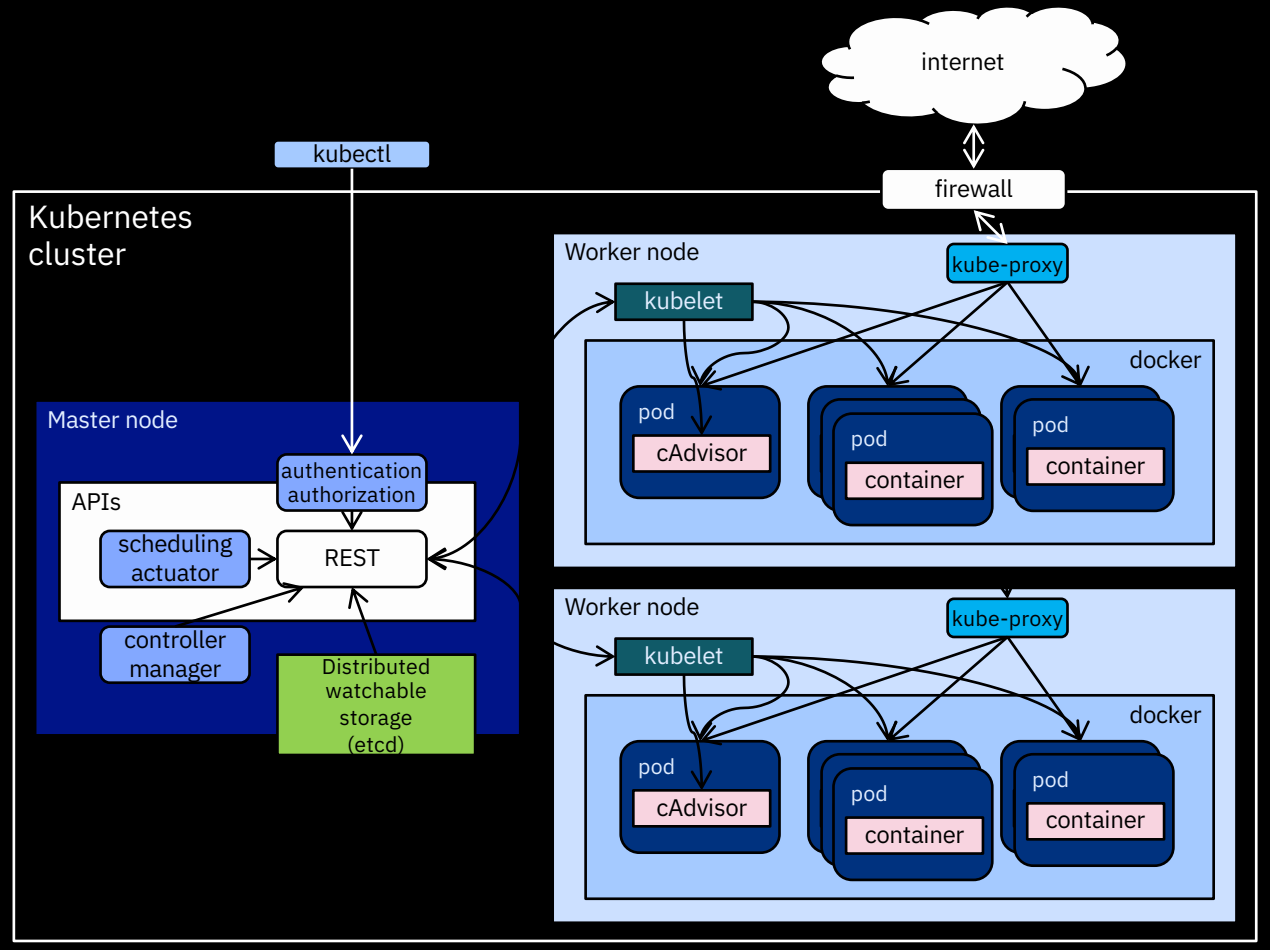
Kubernetes Cluster Architecture

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



Master Node Components

Etcd

- A highly-available key value store
- All cluster data is stored here

API Server

- Exposes API for managing Kubernetes
- Used by kubectrl CLI

Controller manager

- Daemon that runs controllers, which are the background threads that handle routine tasks in the cluster
- Node Controller – Responsible for noticing and responding when nodes go down
- Replication Controller – Replaced by ReplicaSet
- Endpoints Controller – Populates the Endpoints object (that is, joins services and pods)
- Service Account & Token Controllers – Create default accounts and API access tokens for new namespaces

Scheduler

- Selects the worker node each pods runs in

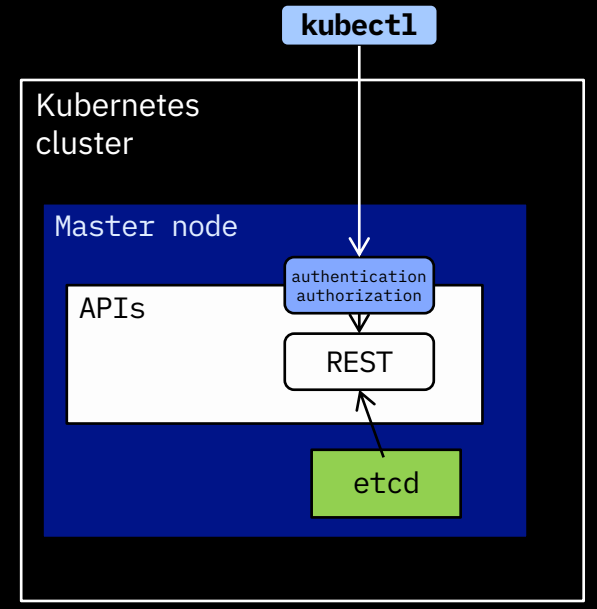
kubectl – talking to the Cluster

`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.



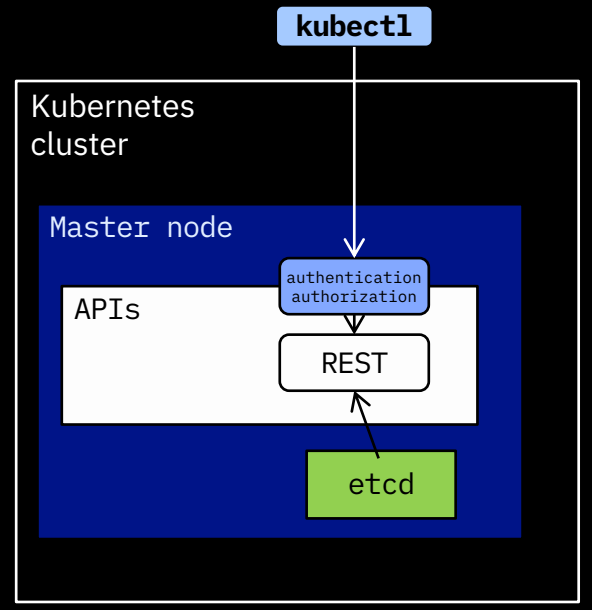
kubectl – talking to the Cluster

`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]

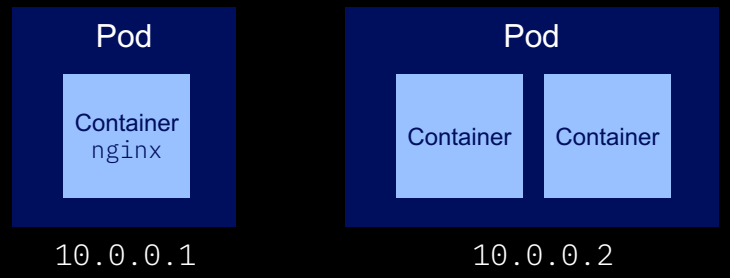
kubectl create -f example.yaml	Create objects in yaml file
kubectl apply -f example.yaml	Modify objects in yaml file
kubectl delete -f example.yaml	Delete objects in yaml file
kubectl get pods	List Pods in Namespace
kubectl describe nodes <node-name>	Details about K8s object
kubectl logs <pod-name>	Get 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?”**

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

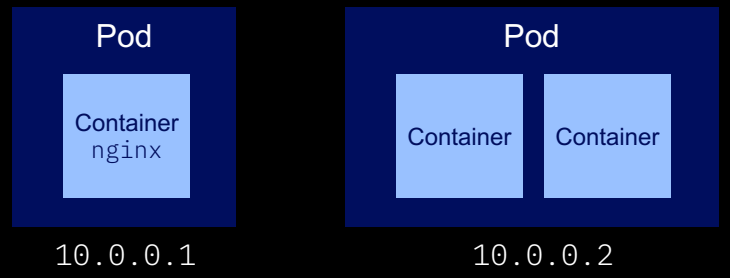


Pod

Features Provided by Kubernetes Pods:

- Creating, Listing, Deleting Pods
- Run commands in your pod's containers with exec
- Copy files to and from containers in your pods
- Port forwarding from your local machine to your pod
- Liveness Probes
- Readiness Probes
- Persistent Volume Storage

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



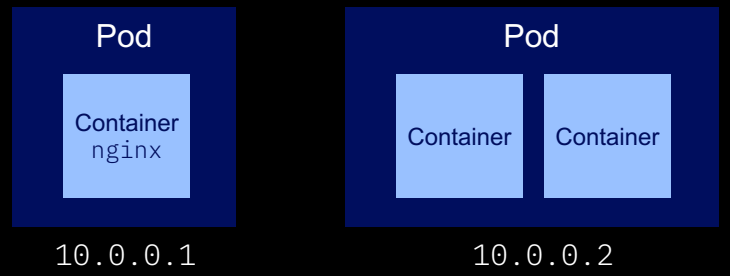
Creating a Pod

```
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

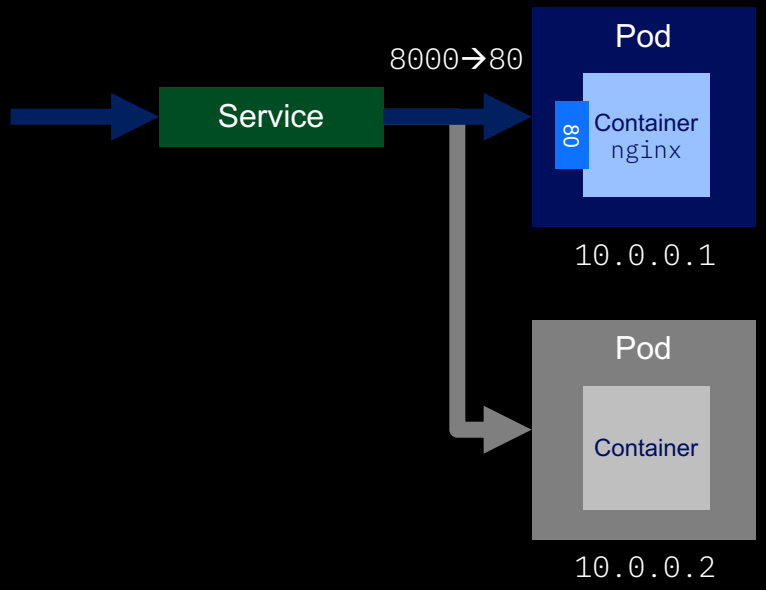
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx:1.7.9
    ports:
    - containerPort: 80
```



Service

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

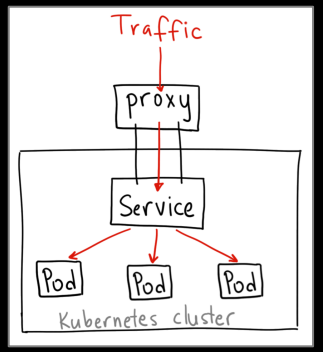
- Also provide load balancing



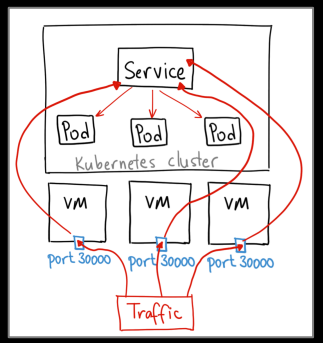

```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  ports:
  - port: 8000
    targetPort: 80
    protocol: TCP
  selector:
    app: nginx
```

Service

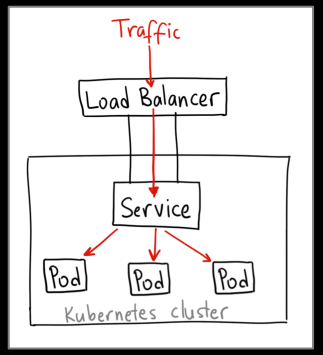
ClusterIP: This type exposes the service on the cluster internal IP. This means that the service is only reachable from within the cluster.



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



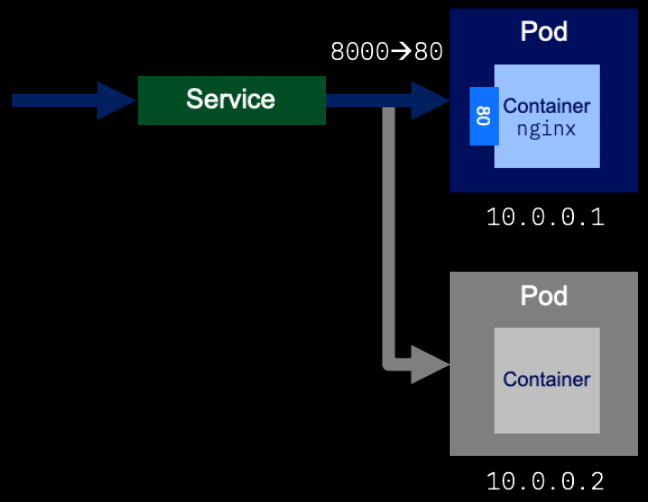
LoadBalancer: This service type exposes a service using the cloud provider load balancer.



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
```



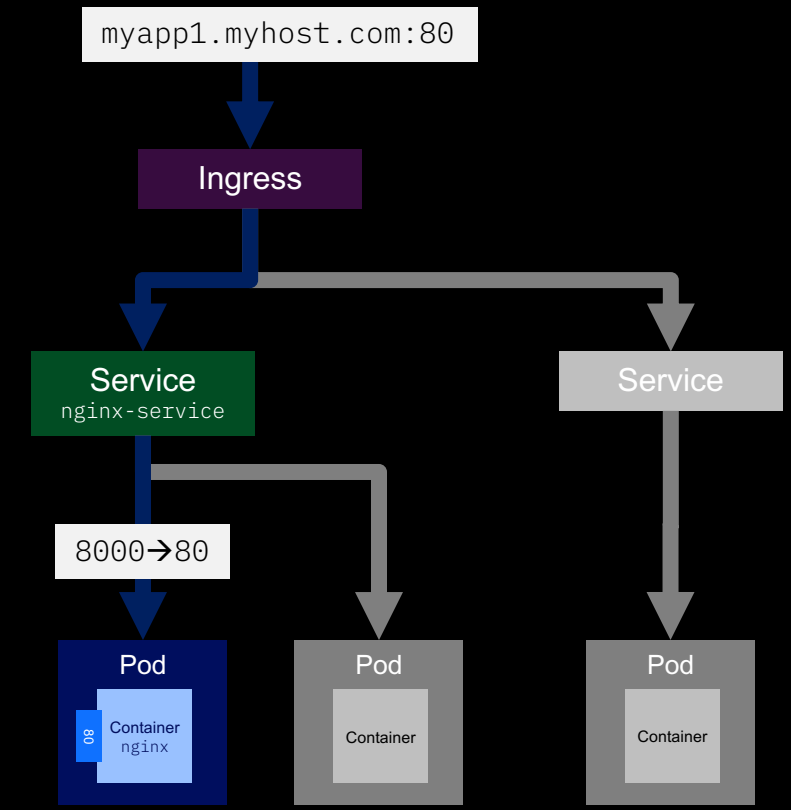
example.yaml

```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.

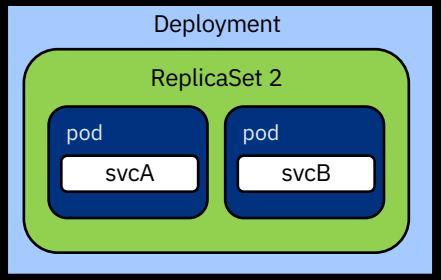
```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: nginx-ingress
  annotations:
    ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - host: myapp1.myhost.com
    http:
      paths:
      - backend:
          serviceName: nginx-service
          servicePort: 80
```



Deployments & ReplicaSets

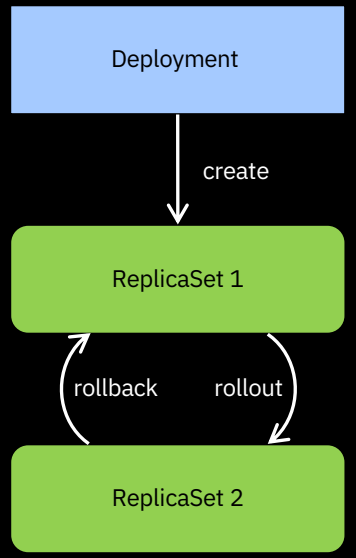
Deployment

- 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

```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

Kubernetes User Security

- **Authentication**
  - OIDC Tokens
  - ServiceAccount Tokens
- **Authorization**
  - RBAC
  - Role
  - RoleBinding
  - ClusterRole
  - ClusterRoleBinding
- **Namespaces**
  - Help to scope access control to a set of resources in a Kubernetes cluster

```
---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRole
metadata:
  name: viewer
rules:
- apiGroups:
  - apps
  - extensions
  resources:
  - deployments
  - replicasets
  verbs:
  - get
  - list
  - watch
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: cicd-viewer
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: viewer
subjects:
- kind: User
  name: IAM#cicd@us.ibm.com
  apiGroup: rbac.authorization.k8s.io
```

Configuring Resources and Containers

Label

- **Metadata** assigned to Kubernetes resources (pods, services, etc.)
- Key-value pairs for identification
- Critical to Kubernetes as it relies on querying the cluster for resources that have certain labels

Selector

- An expression that **matches labels** to identify related resources

Deployment details

Type	Detail
Name	cam-ui-basic
Namespace	services
Created	39 minutes ago
Labels	app=cam-ibm-cam,chart=ibm-cam-1.3.0,heritage=Tiller,name=cam-ui-basic,release=cam
Selector	name=cam-ui-basic
Replicas	Desired: 1 | Total: 1 | Updated: 1 | Available: 1
RollingUpdateStrategy	Max unavailable: 1 | Max surge: 1
MinReadySeconds	0

Service details

Type	Detail
Name	cam-ui-basic
Namespace	services
Created	40 minutes ago
Type	ClusterIP
Labels	app=cam-ibm-cam,chart=ibm-cam-1.3.0,heritage=Tiller,name=cam-ui-basic,release=cam
Selector	name=cam-ui-basic
Cluster IP	10.0.0.165
External IP	-
Port	cam-ui-basic 39002/TCP
Node port	None
Session affinity	None

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

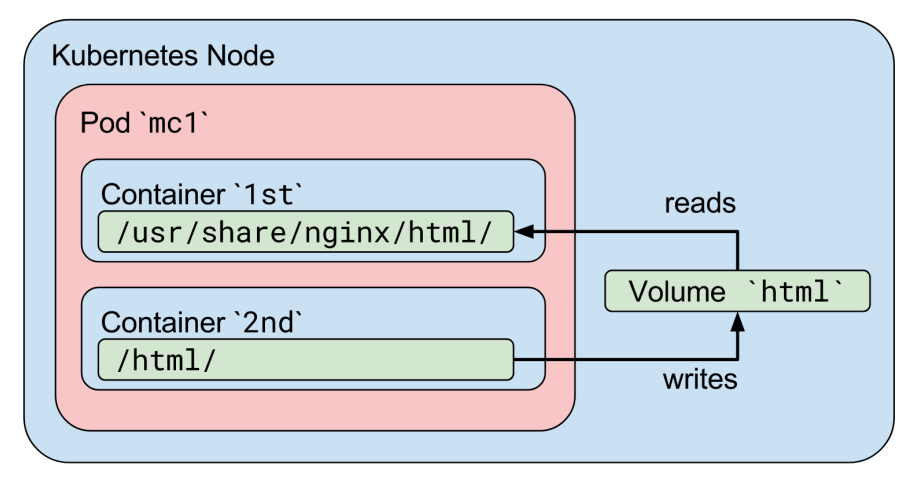
```
$ kubectl autoscale deployment <deployment-name> --cpu-percent=50
  --min=1 --max=10 deployment "<hpa-name>" autoscaled
```

Creates a horizontal pod autoscaler

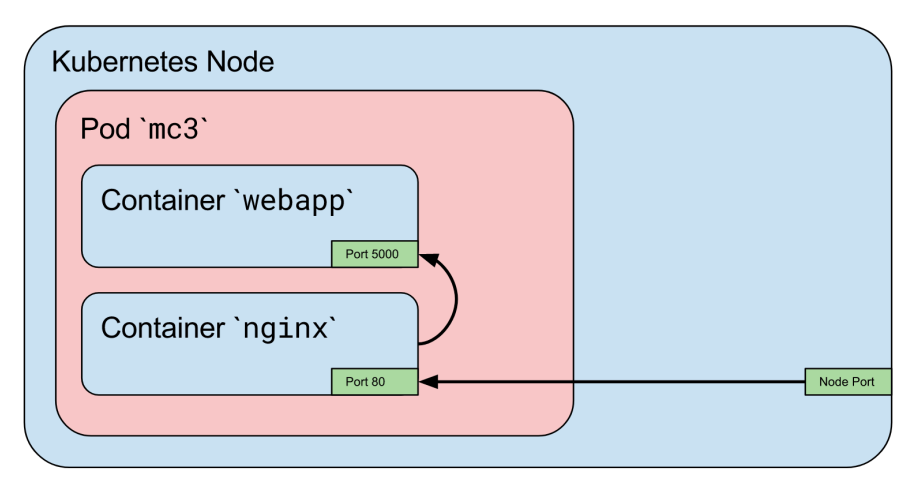
- An HPA instance
- Maintains between 1 and 10 replicas of the pods controlled by the deployment
- Maintains an average CPU utilization across all pods of 50%

Multi-Container Pod Design Patterns

Shared Volumes

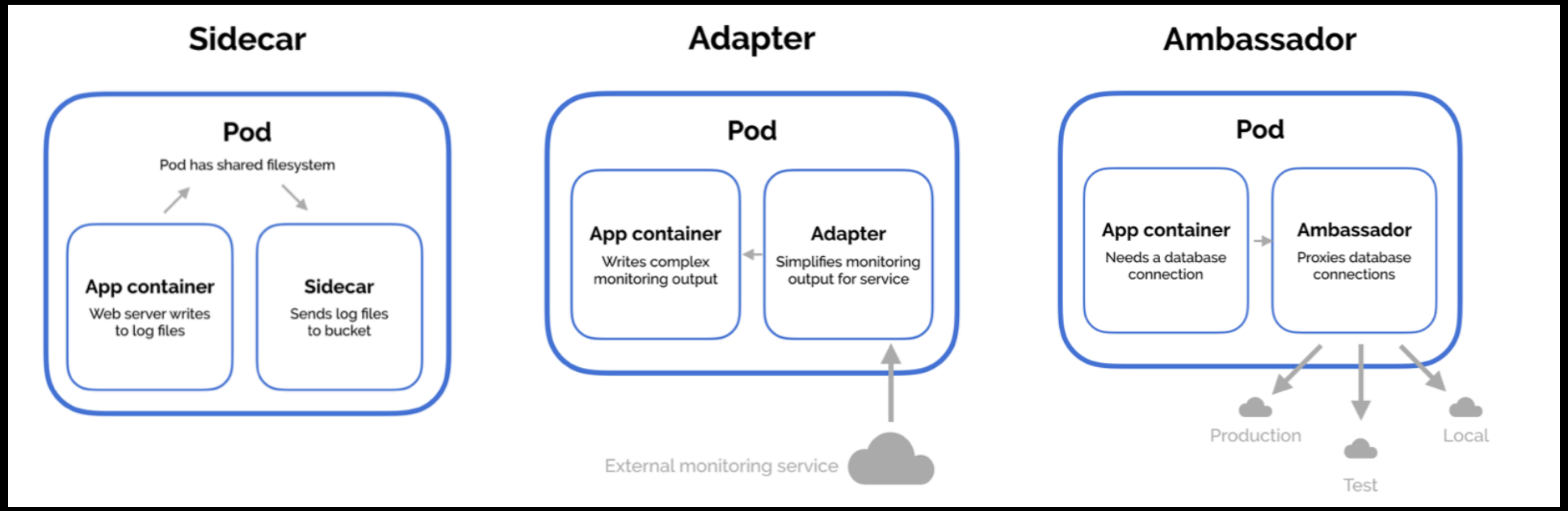


No Network isolation



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.

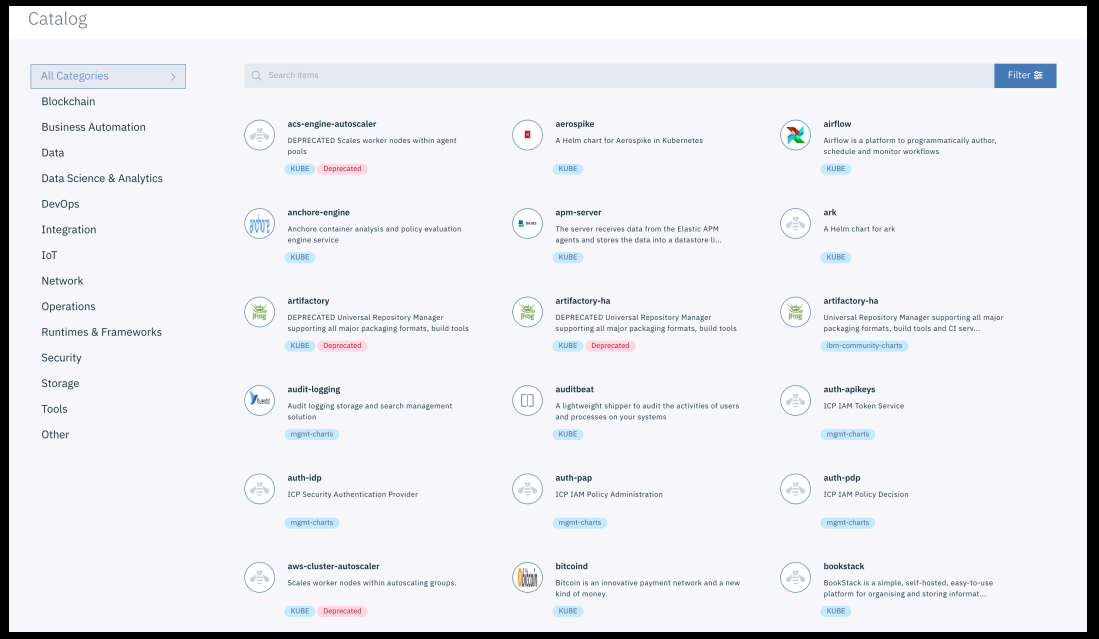
Adapter pattern
The adapter pattern is used to standardize and normalize application output or **monitoring data** for aggregation.

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

https://matthewpalmer.net/kubernetes-app-developer/articles/multi-container-pod-design-patterns.html

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

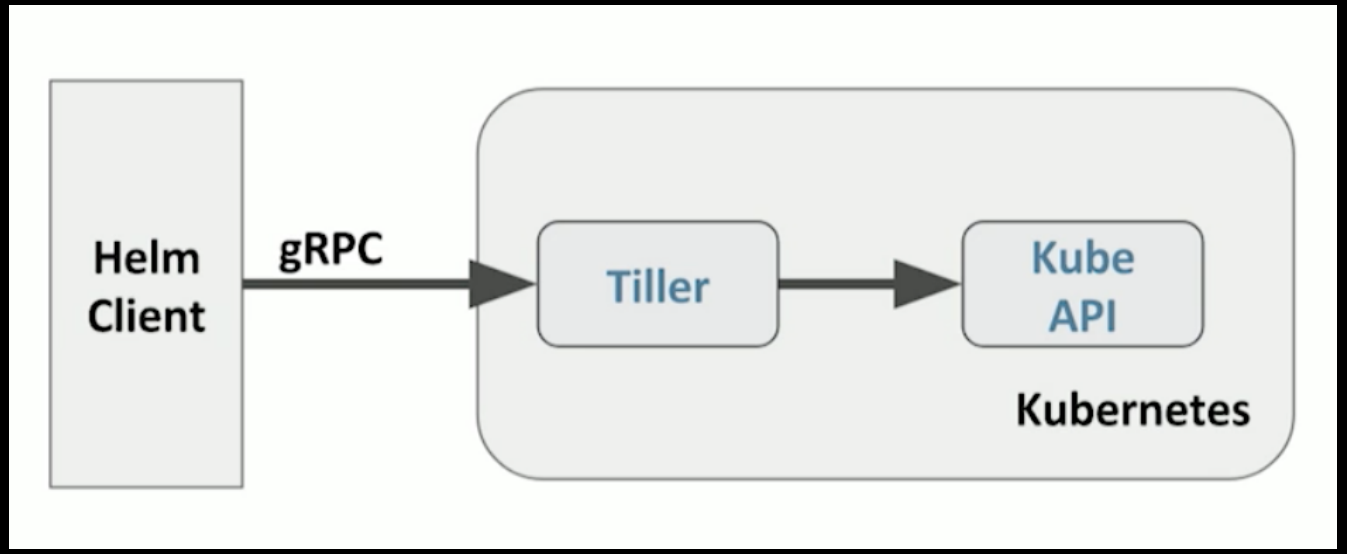


HELM Example

Helm has 2 Components :

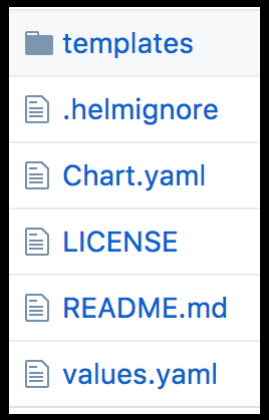
Tiller : Tiller is the in-cluster component of Helm.
It interacts directly with the Kubernetes API server to install, upgrade, query, and remove Kubernetes resources. It also stores the objects that represent releases.

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.
  - 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

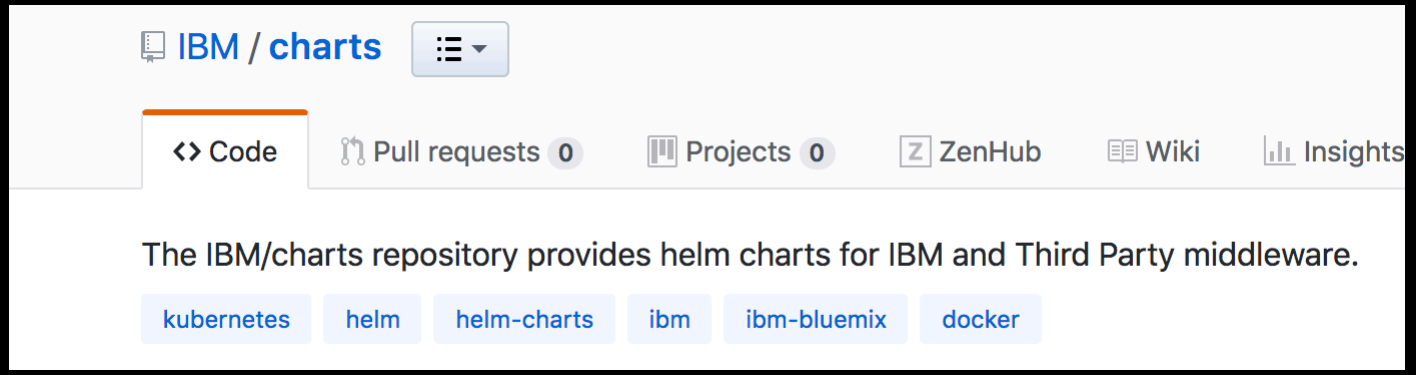
```
1  apiVersion: v1
2  kind: PersistentVolumeClaim
3  metadata:
4    name: {{ template "fullname" . }}-pv-claim
5  spec:
6    accessModes:
7      - ReadWriteMany
8    resources:
9      requests:
10       storage: {{ .Values.storage.size }}
```

Values.yaml file

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

IBM Helm Charts

https://github.com/ibm/charts



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

https://medium.com/@ssriva/7-missing-factors-from-12-factor-application-2a3e1169bd9d

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

7 missing factors

Application health / Probes

Resource requests, limits, & quotas

Rollout/Rollback

Least Privilege

CADF (Cloud Auditing Data Federation)

Access Control -Authentication and Authorization

Metering - Compute resources allocated to run the containers

https://medium.com/@ssriva/7-missing-factors-from-12-factor-application-2a3e1169bd9d

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

13x

More software releases

Portability

- Between different environments
- Between private and public cloud

Eliminate

“works on my machine” syndrome

Efficiency

- Better usage of computer resources

~47%

Reduction of VMs, OS licensing and server cost

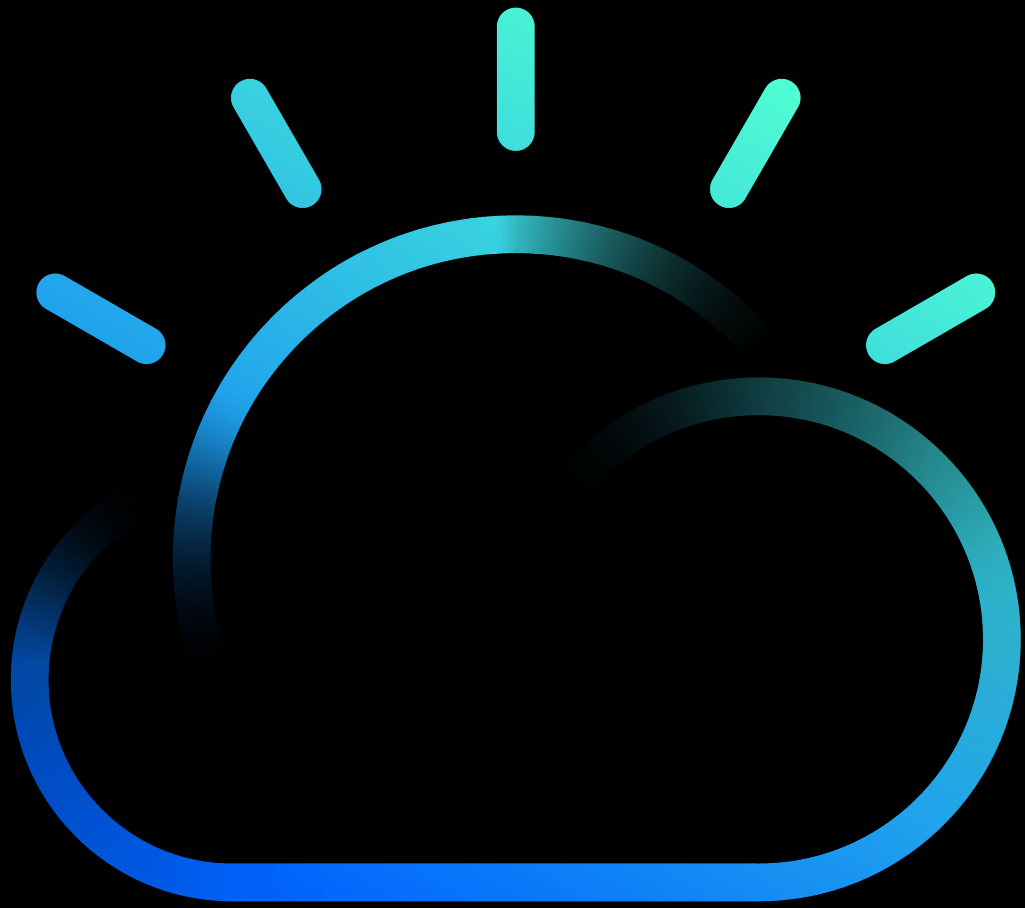
Automation

- Next level standardization and automation

Reduce

operation cost

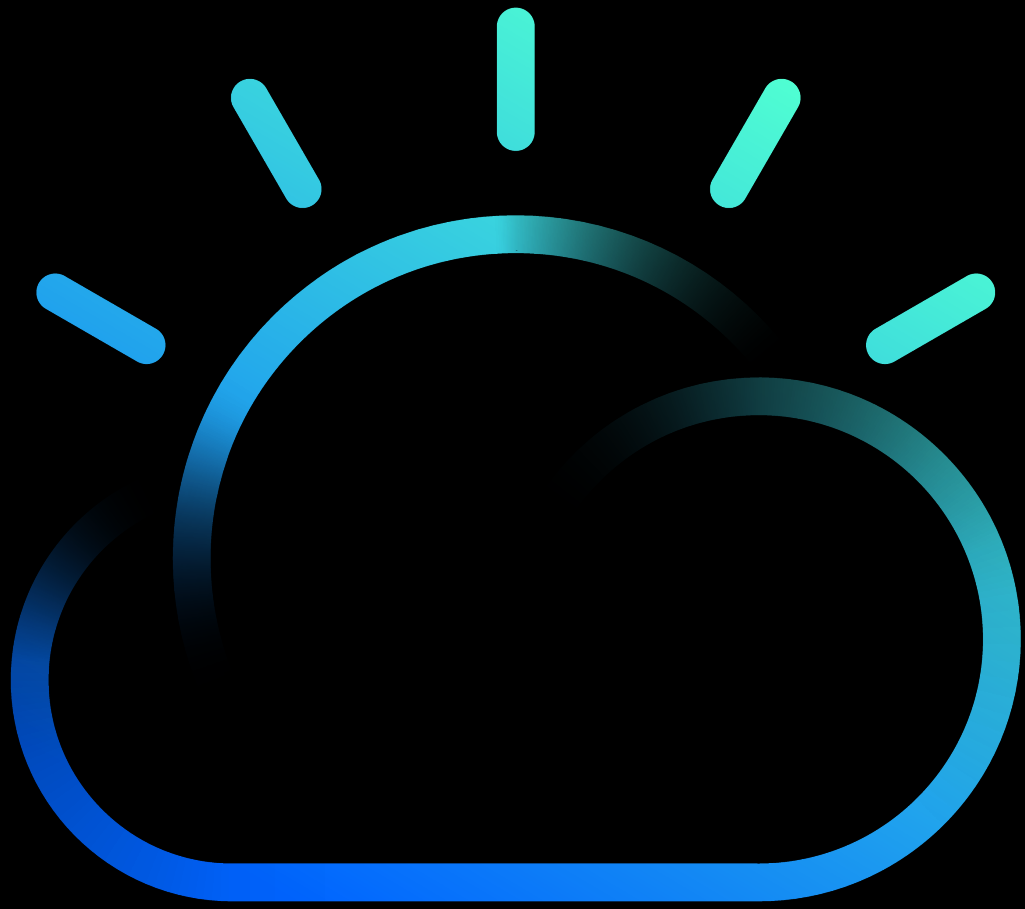
QUESTIONS?



IBM **Cloud**

The path to Cloud
Kubernetes - *Hands-On*

04



IBM **Cloud**

Remember your Team Color

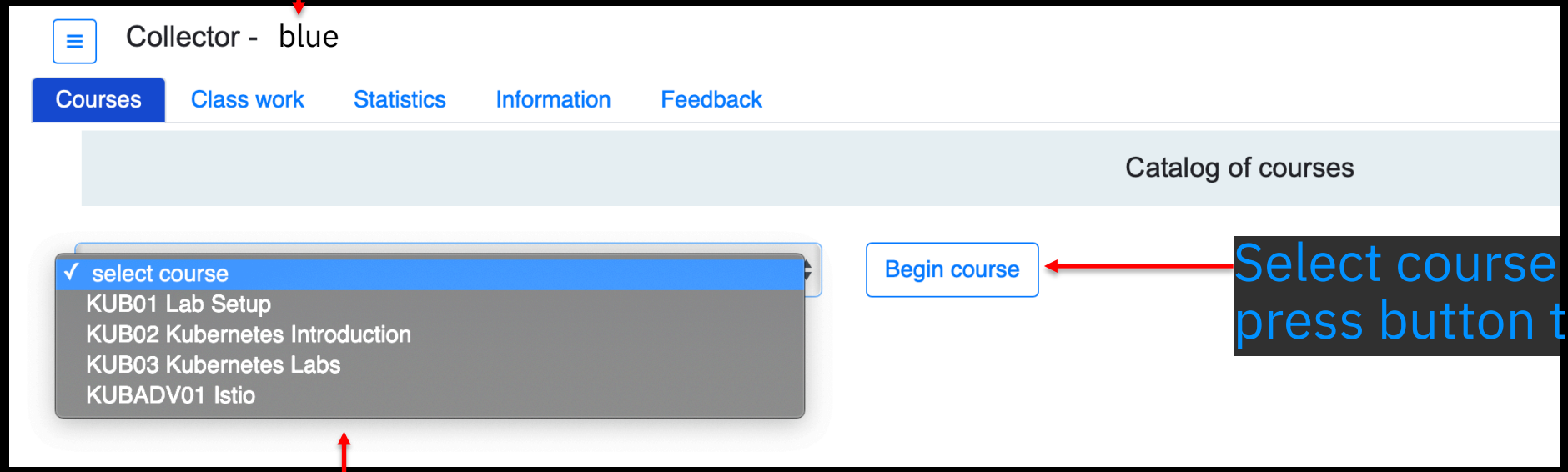
black 31701	olive 31711	peru 31715
white 31702	brown 31712	chocolate 31716
red 31703	lightblue 31713	orchid 31717
blue 31704	orange 31708	gold 31718
yellow 31705	purple 31709	pink 31719
lime 31706	maroon 31710	violet 31720
cyan 31707	firebrick 31714	

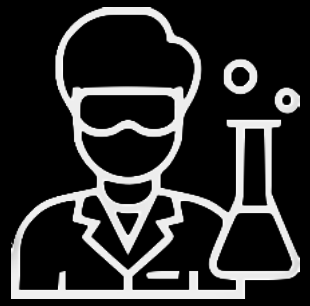
Collector - Accessing team web site

http://158.177.137.195:{port#}

blue 31704

Team name / color will be shown





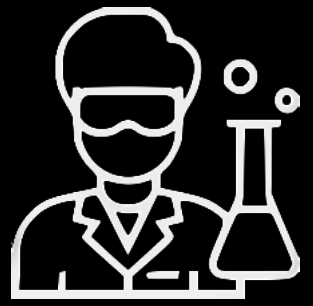
KUB03 Labs

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

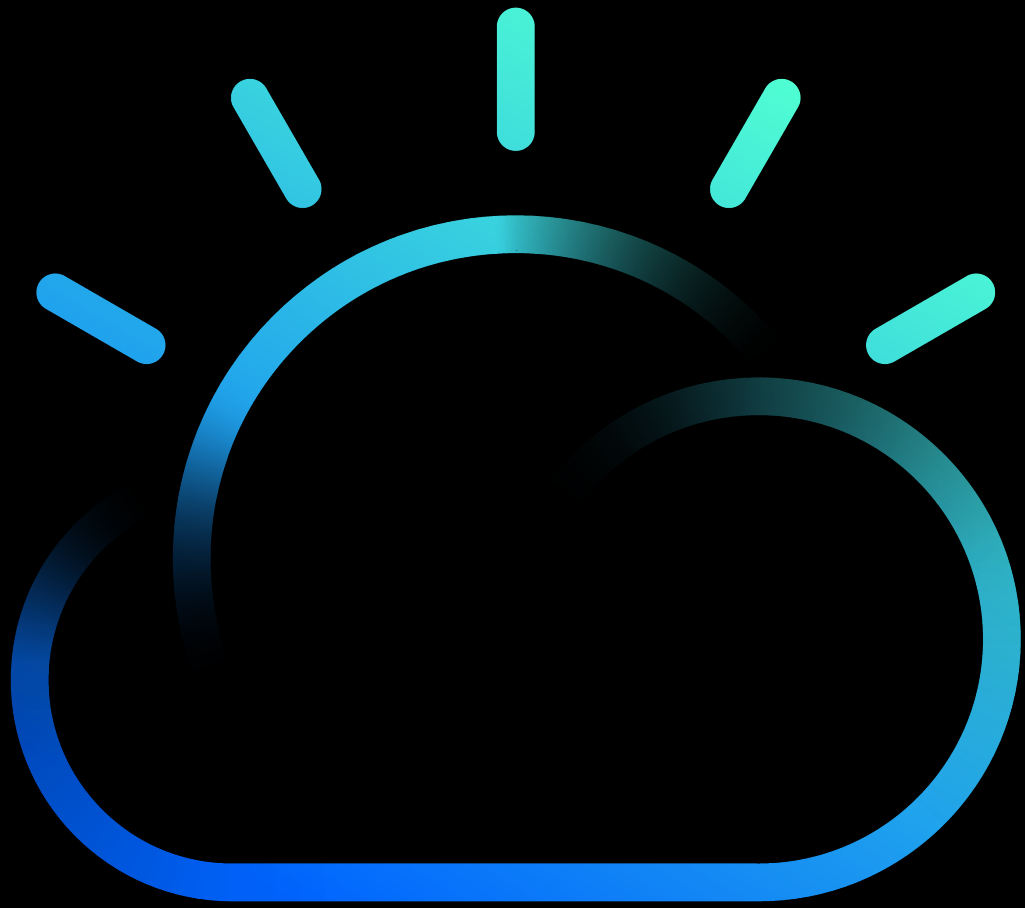


READY SET GO!!!!

http://158.177.137.195:{port#}

Duration: 90 mins

QUESTIONS?



IBM **Cloud**