# Kubernetes Workshop Series

# JTC01
# Microservices and Docker
# Basic
# Concepts

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

Welcome to the

# Kubernetes Workshop Series

# Housekeeping

Meeting is being recorded to be shared on Social Media

Meeting Mute All: Unmute to speak

Breaks: every 60mins (interrupt me if I forget ;-)

Questions:

- In Slack # (not in Webex!)
- Adressed at the end of the Module
- Additional questions: unmute to speak

We will monitor the Slack channel during the Labs
→ Feel free to answer other participants questions

# 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 – Microservices and Docker - Basic Concepts

Module 0: Prepare the Labs
Module 1: Microservices
Module 2: Containers with Docker
Module 3: Let’s get real
Module 4: Docker Hands-On

# Sources and documentation will be available here:

https://github.com/niklaushirt/k8s_training_public

https://github.com/niklaushirt/training

Kubernetes Workshop Series

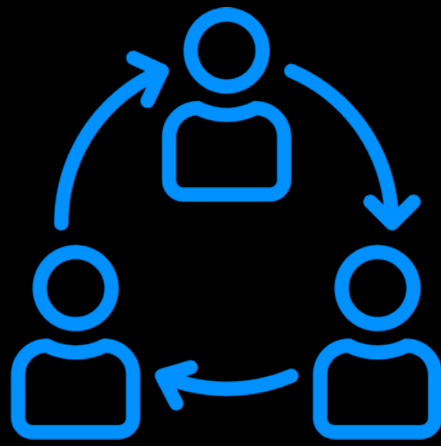
# Prepare the Labs

00

# Session Objectives

Attendees will run their own ***Personal Training Environment (PTE)*** in the VM.

Following the lectures there will be ***hands-on*** labs that each participant can complete in the PTE.



# JTC90 Lab Setup

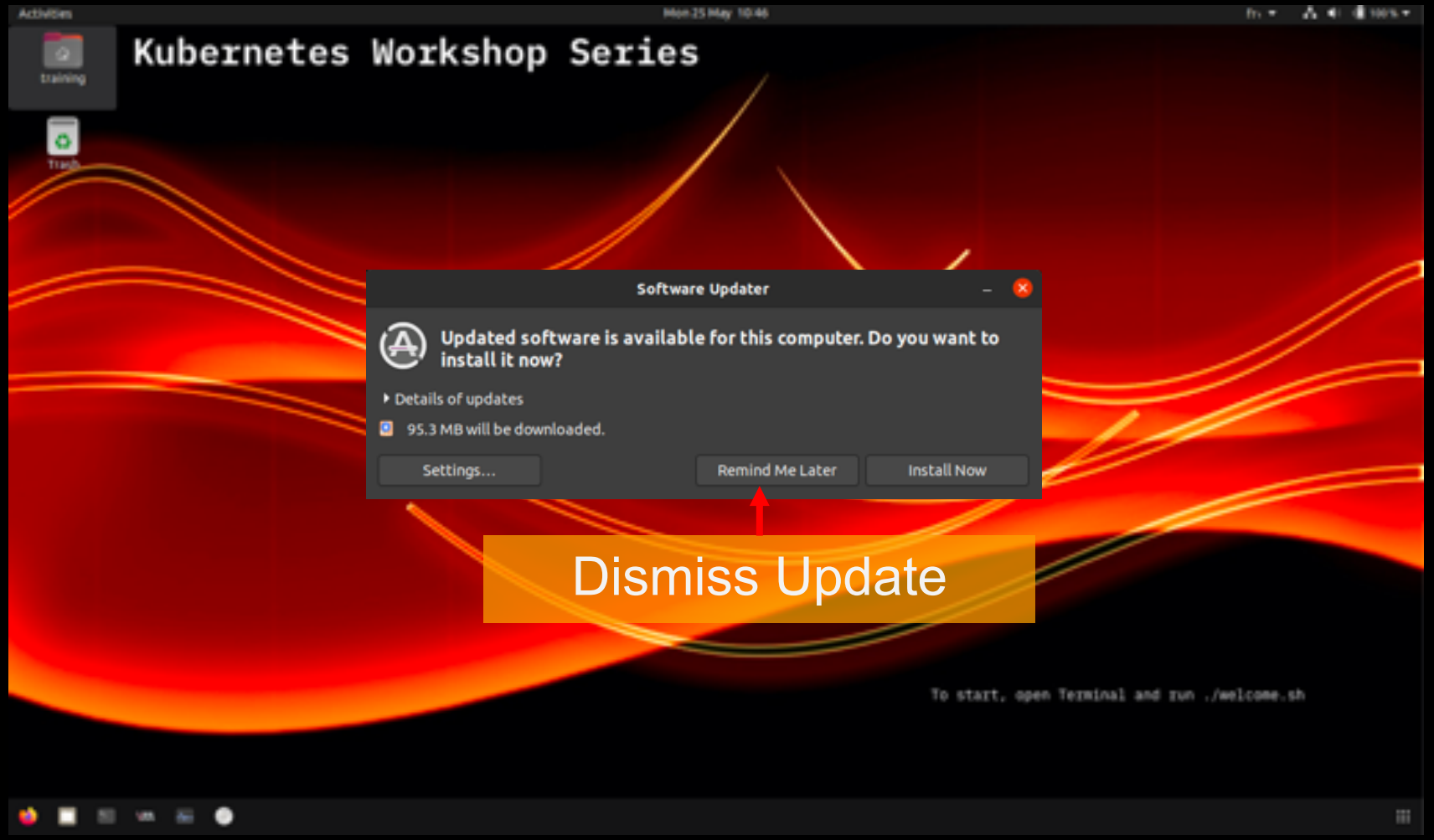
Task 1: Download Training VM

Task 2: Setup VMWare / VirtualBox

Task 3: Start Training VM

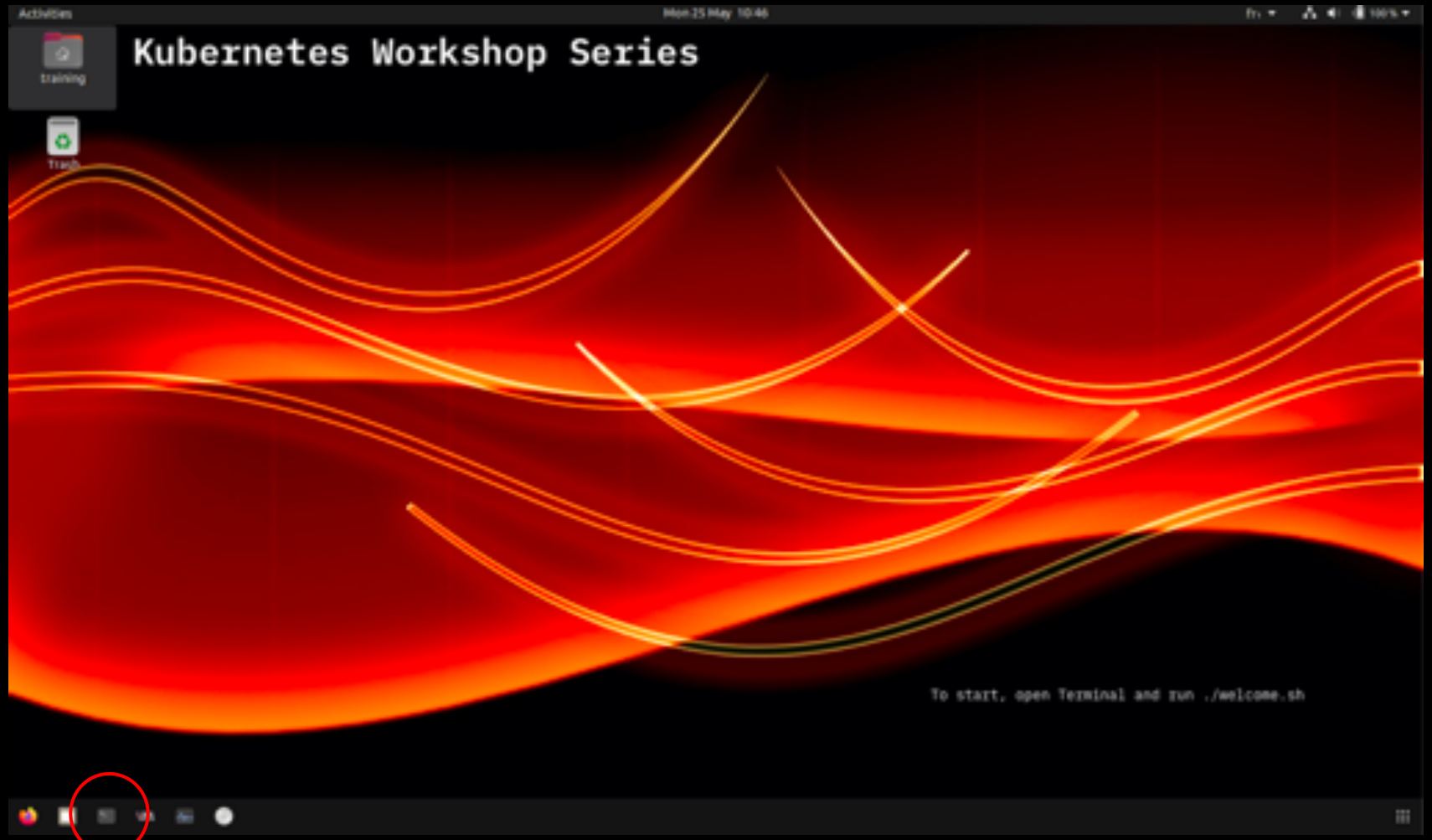
Task 4: Login / Check

# Accessing your Personal Training Environment



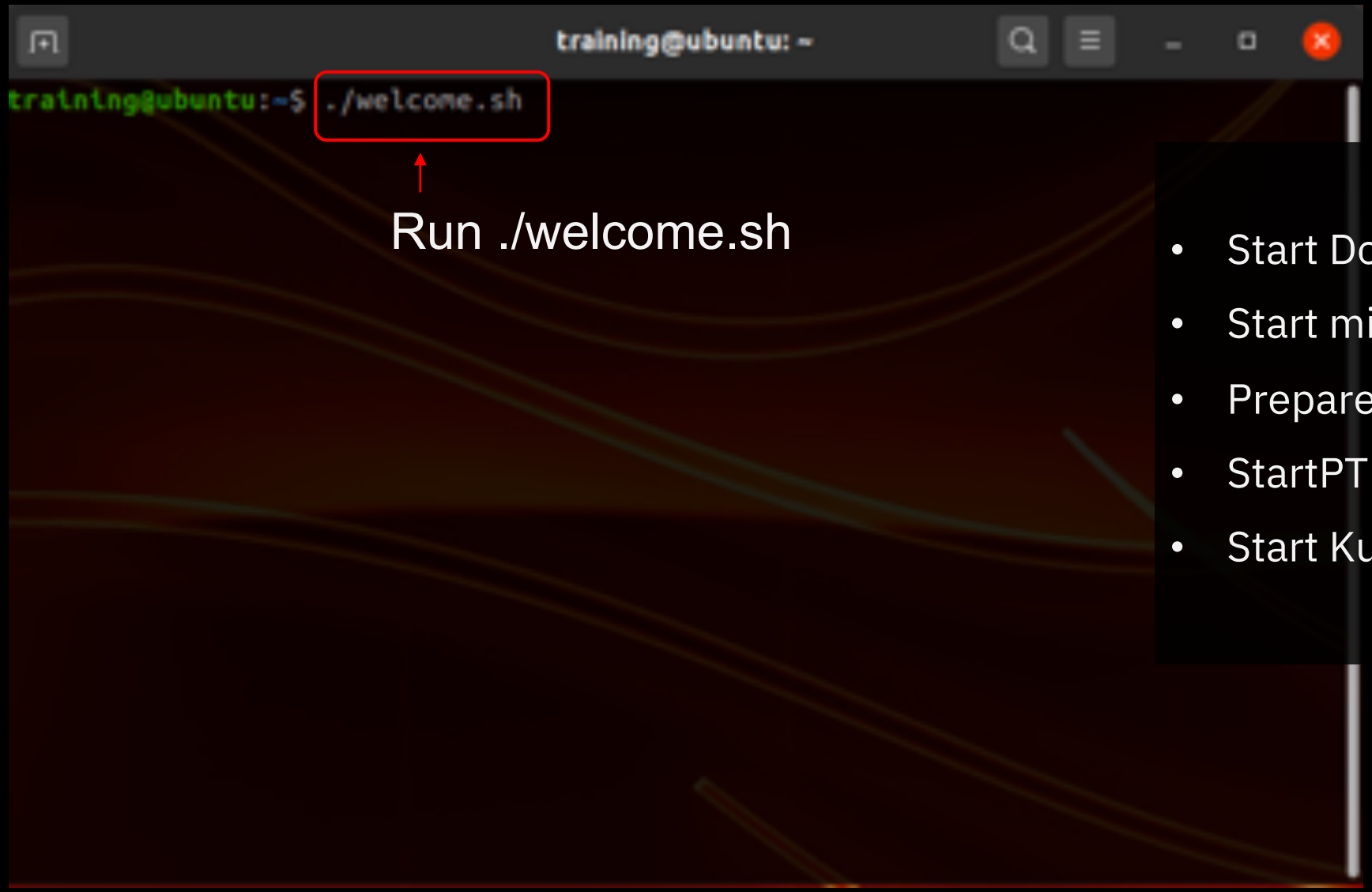
# Accessing your Personal Training Environment

# Accessing your Personal Training Environment



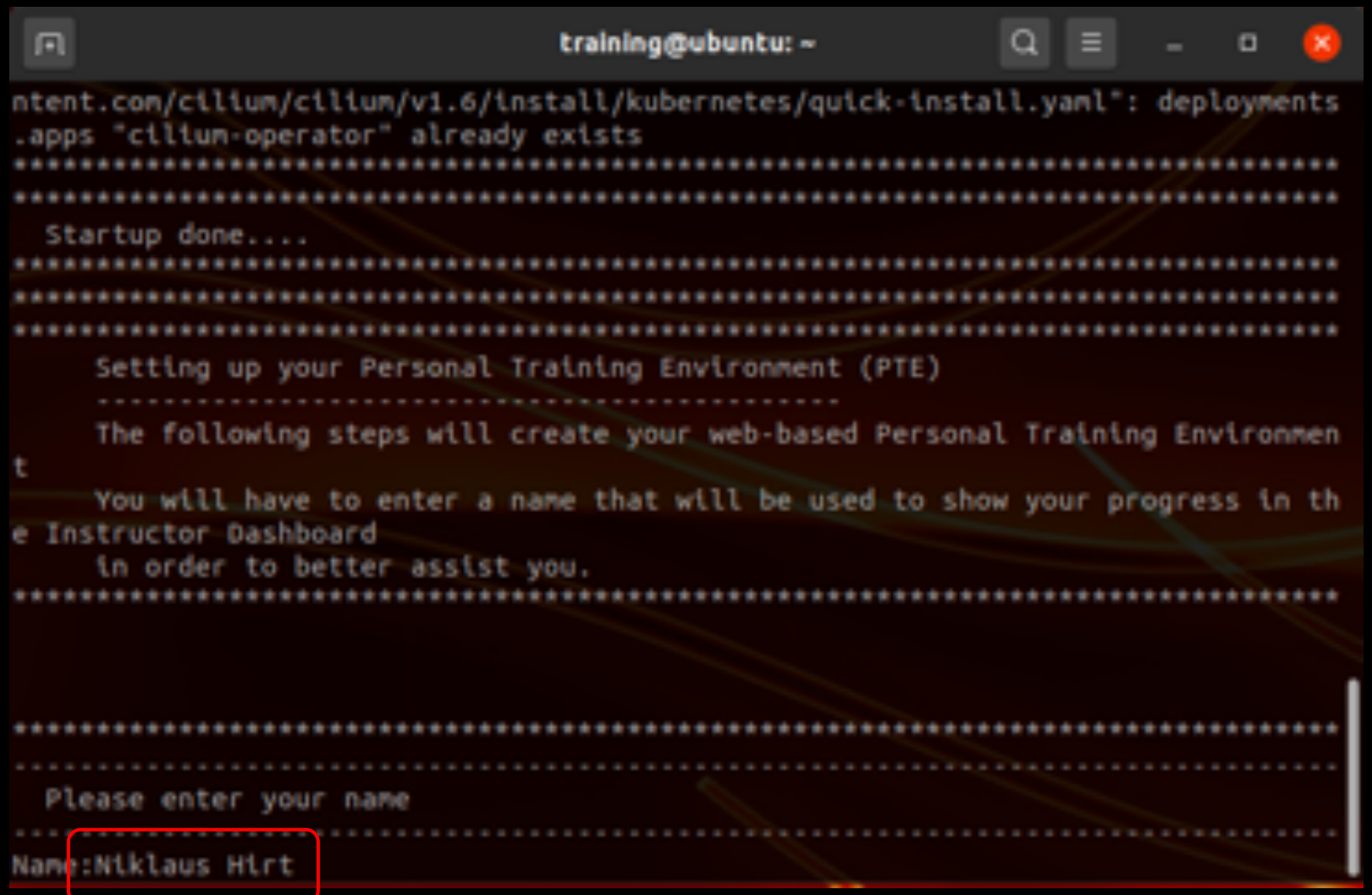

Start Terminal

# Accessing your Personal Training Environment




- Start Docker
- Start minikube
- Prepares networking
- StartPTE
- Start Kubernetes Dashboard

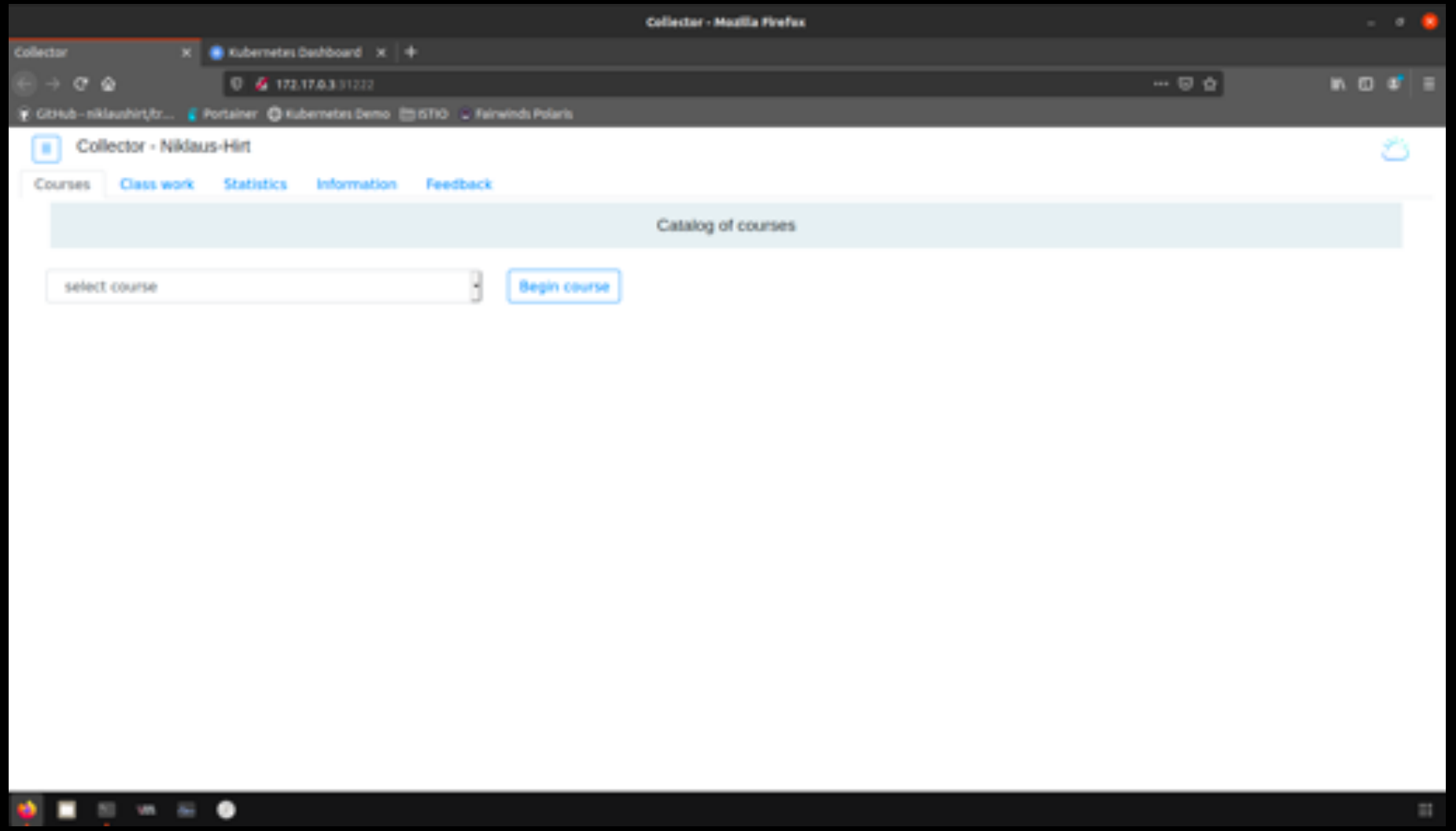
# Accessing your Personal Training Environment




Enter your name

Name will be used to show your progress in the Instructor Dashboard in order to better assist you

# Accessing your Personal Training Environment




When completed, your PTE and Kubernetes Dashboard will open automatically

# JTC90 Lab Setup

**EVERYBODY OK ?**

Task 1: Download Training VM

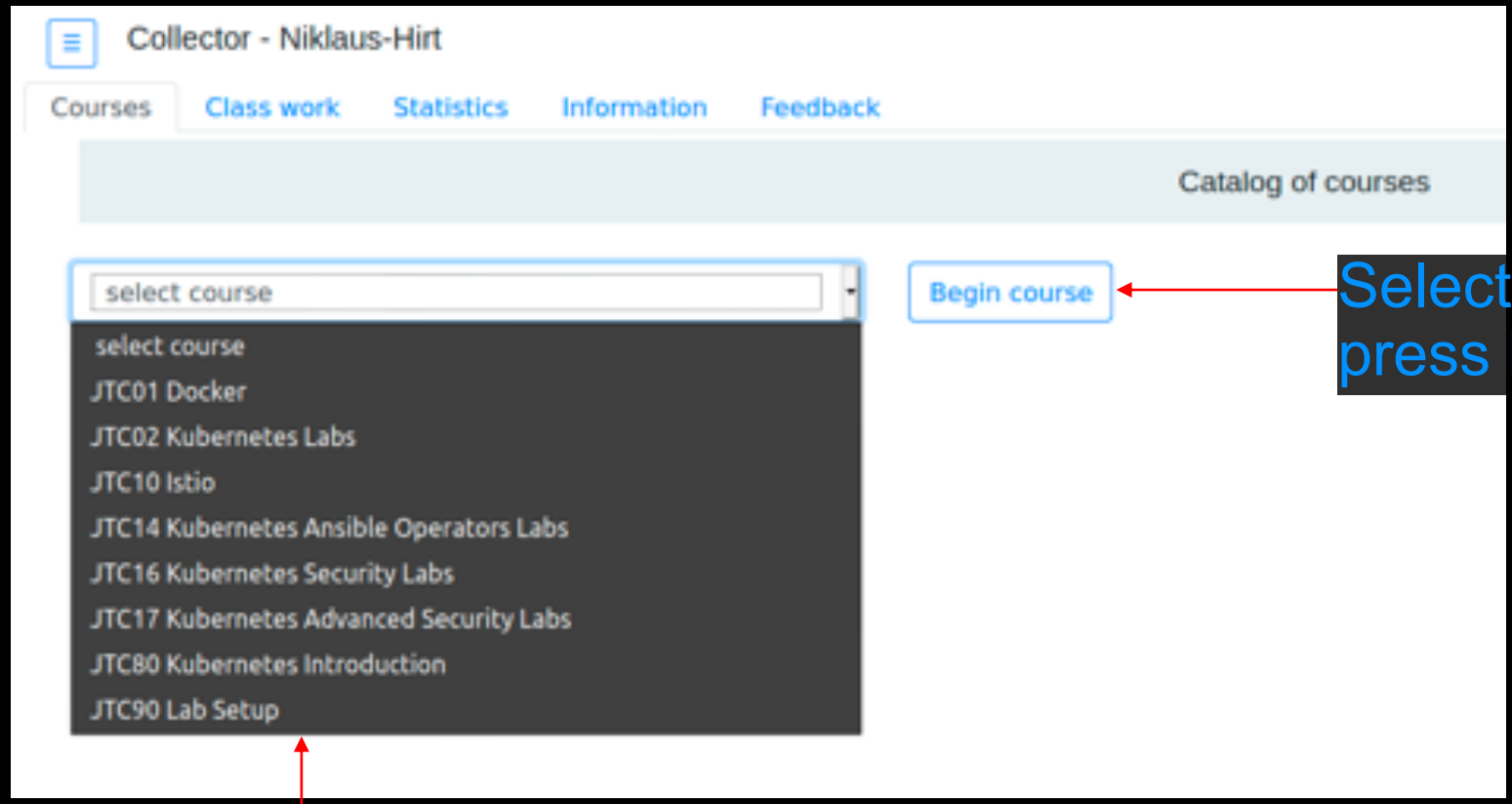
Task 2: Setup VMWare / VirtualBox

Task 3: Start Training VM

Task 4: Login / Check

# Accessing your Personal Training Environment

Name will be shown

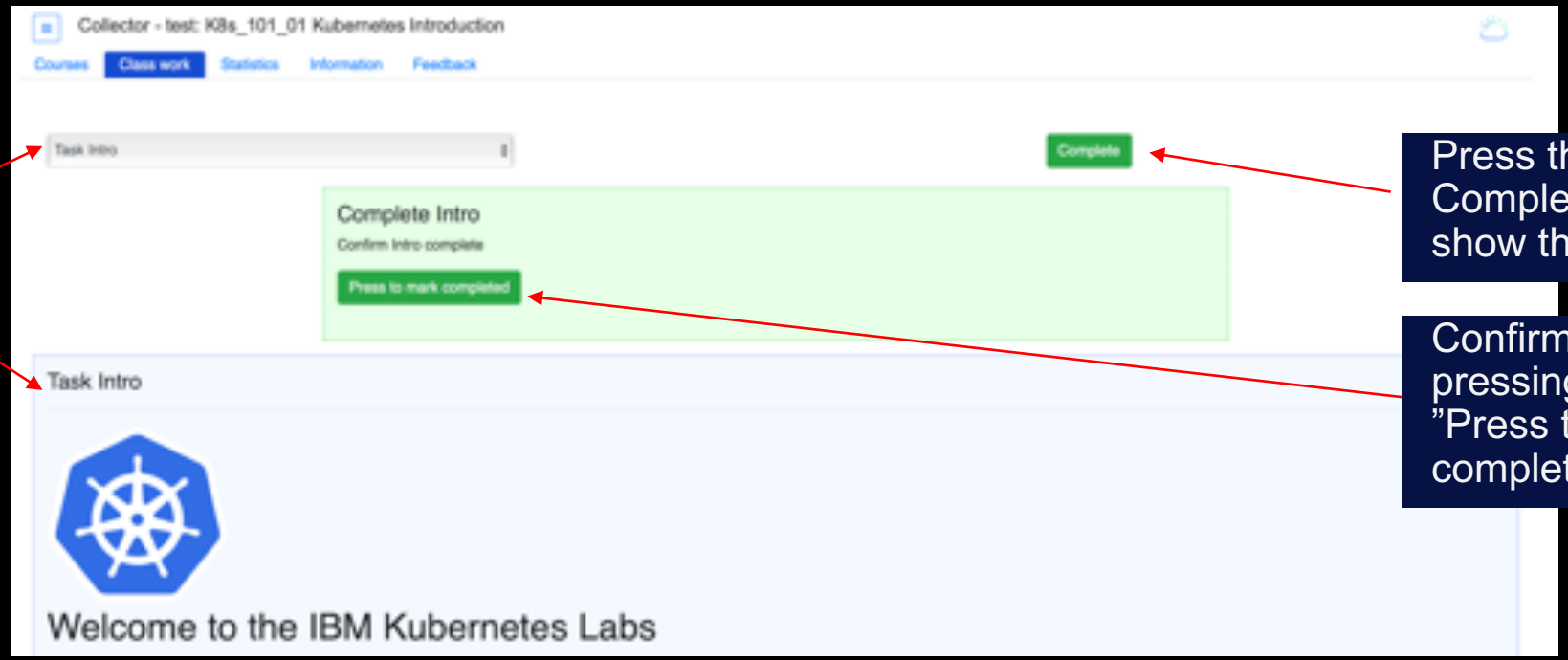



Select course and press button to begin

Current course catalog

# 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

# Following your progress

Course title

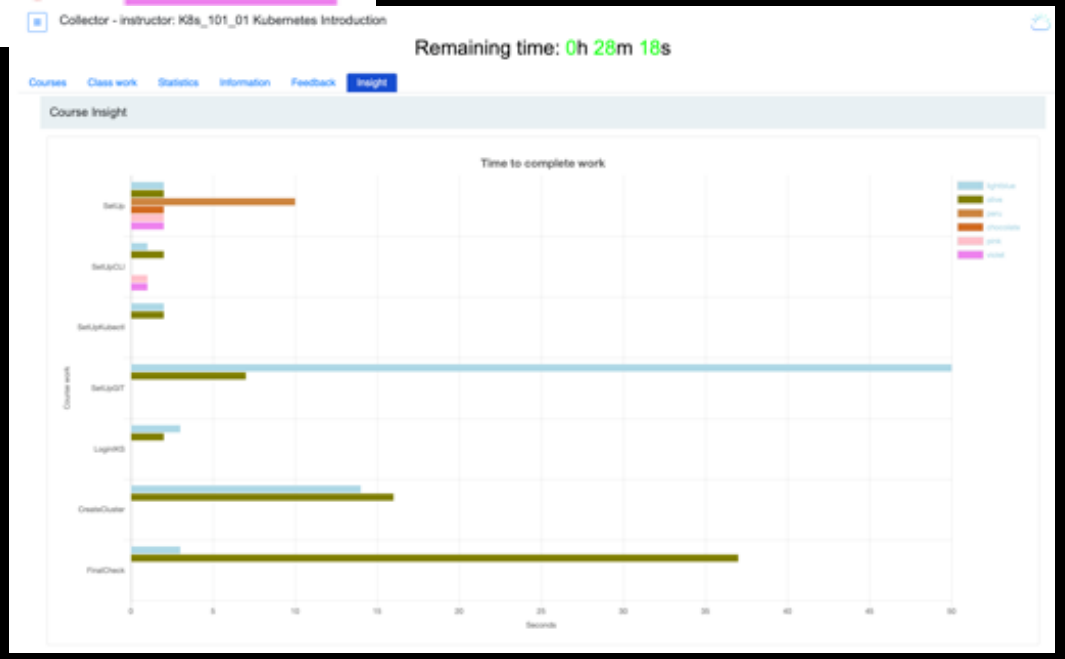


The number of items tracked will change based on the current course selected.

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

# Instructor Dashboard

Remaining Time for the Lab

# QUESTIONS?

Kubernetes Workshop Series

# **Microservices**

01

Kodak

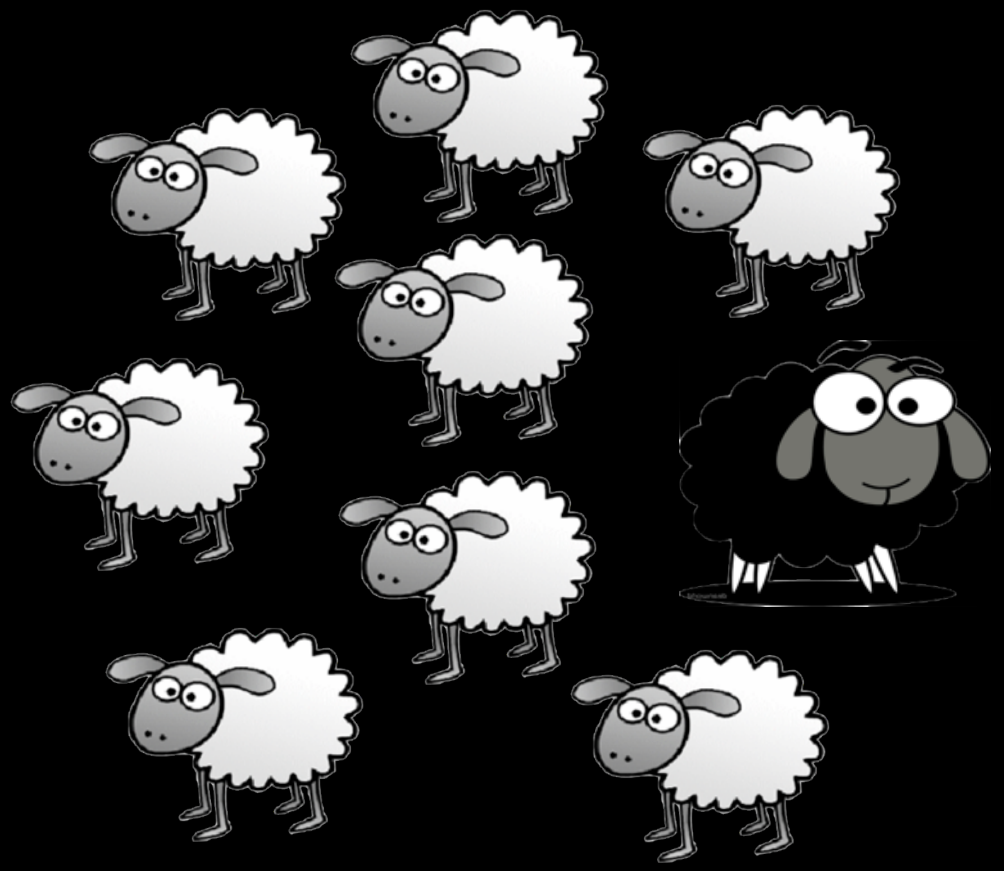

# disruption

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/

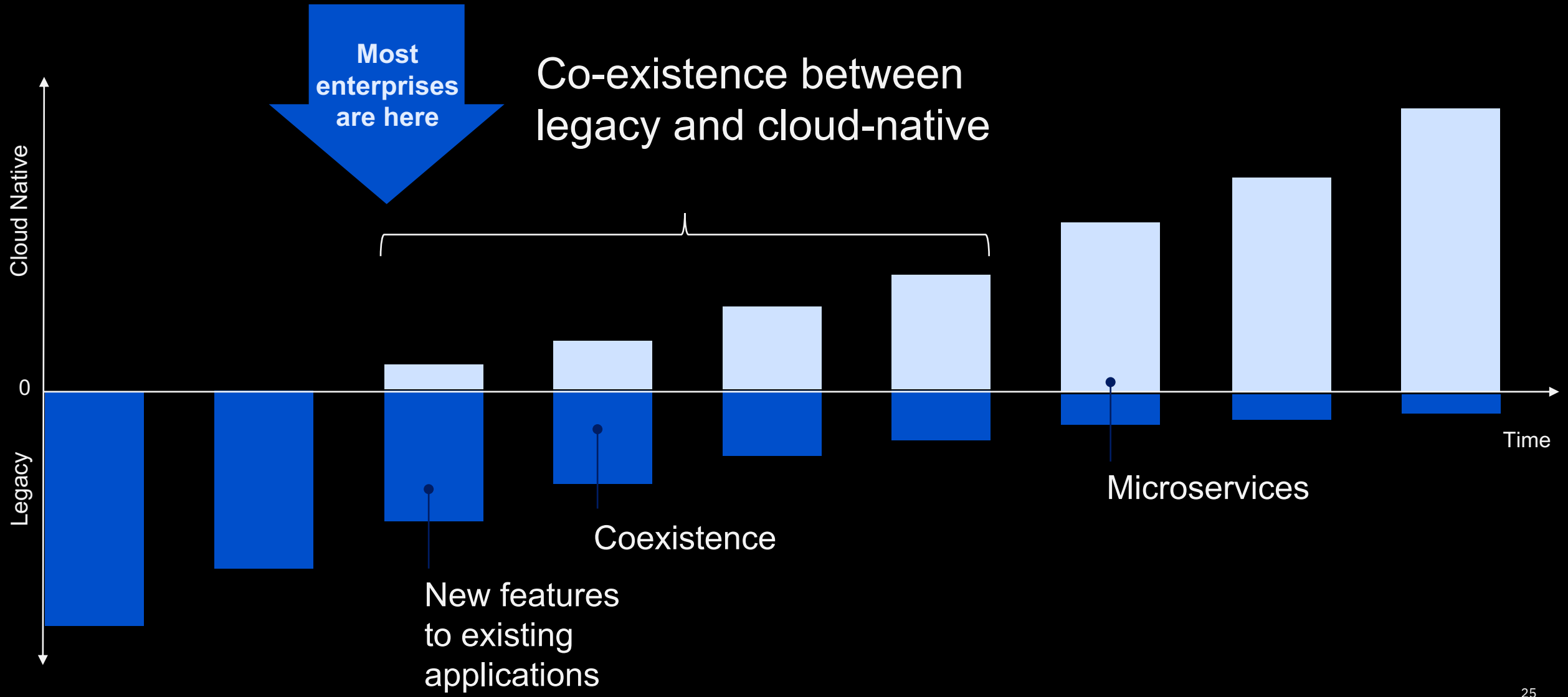
*noun*

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 | |

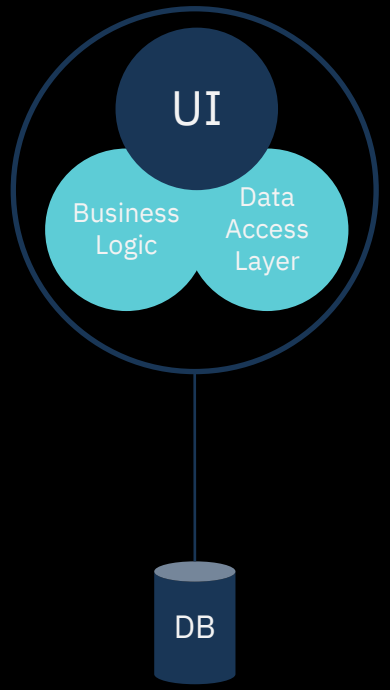
# Microservices

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

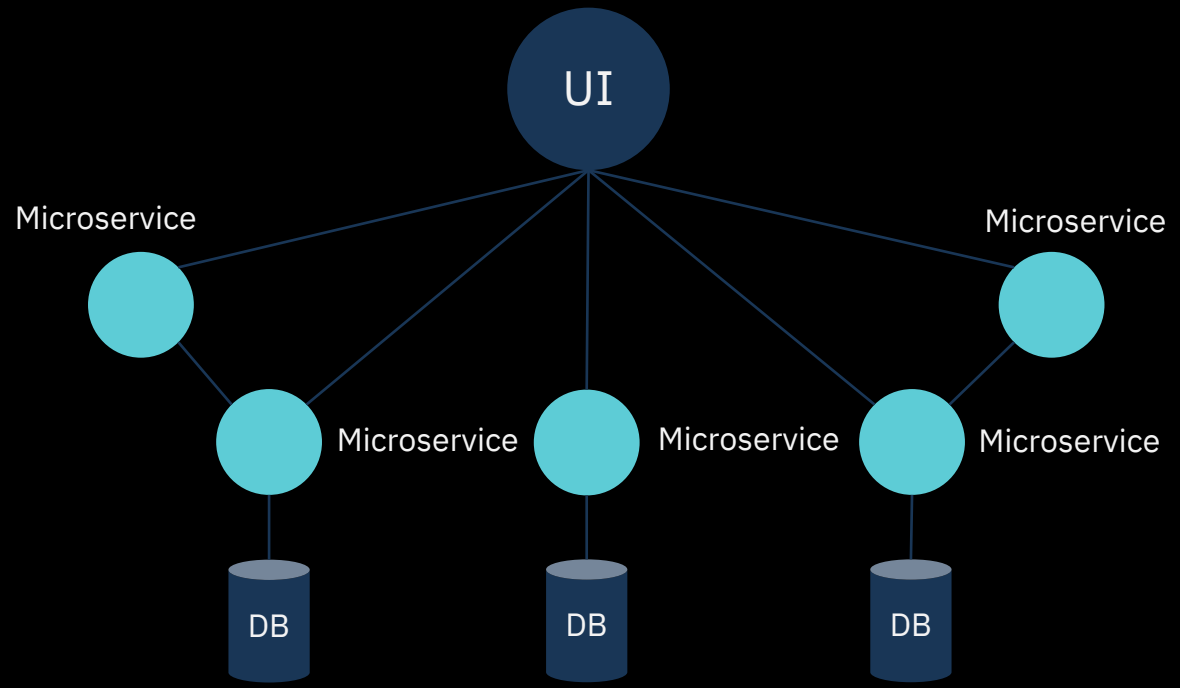
**Accelerate delivery** by minimizing communication and coordination between people

# Microservices architecture

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




Monolithic Architecture

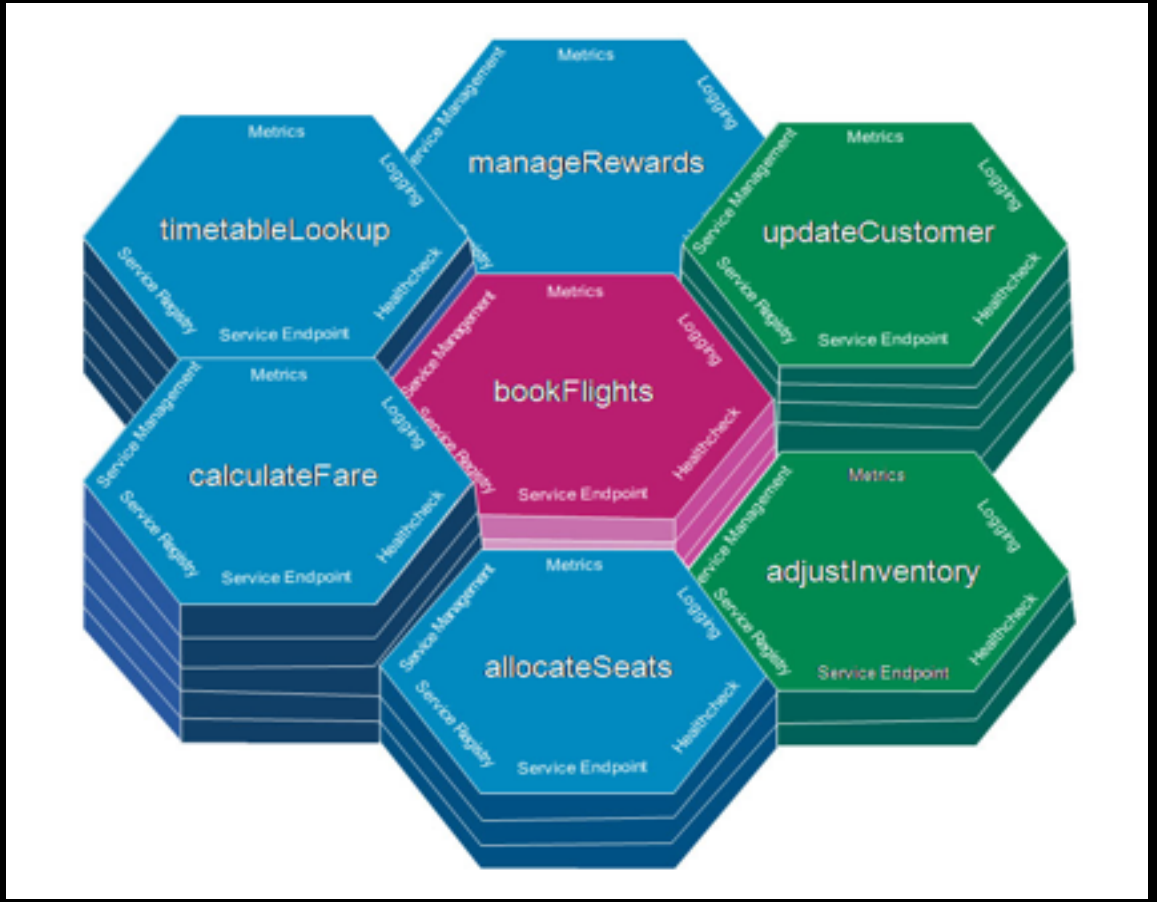



Microservices Architecture
A microservice is a granular, decoupled component within a broader application

# 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

# Obvious rules for developing cloud applications

1. Don’t code your application directly to a **specific topology**
2. Don’t assume the local **file system** is **permanent**
3. Don’t keep **session state** in your application
4. Don’t **log** to the file system
5. Don’t assume any specific **infrastructure dependency**
6. Don’t use **infrastructure APIs** from within your application
7. Don’t use **obscure protocols**
8. Don’t rely on **OS-specific features**
9. Don’t **manually install** your application

Read the article : http://www.ibm.com/developerworks/websphere/techjournal/1404_brown/1404_brown.html

# Cloud-Native Application Goals

**Horizontal scaling**

- Application runs in multiple runtimes spread across multiple hosts (VIII)

**Immutable deployment**

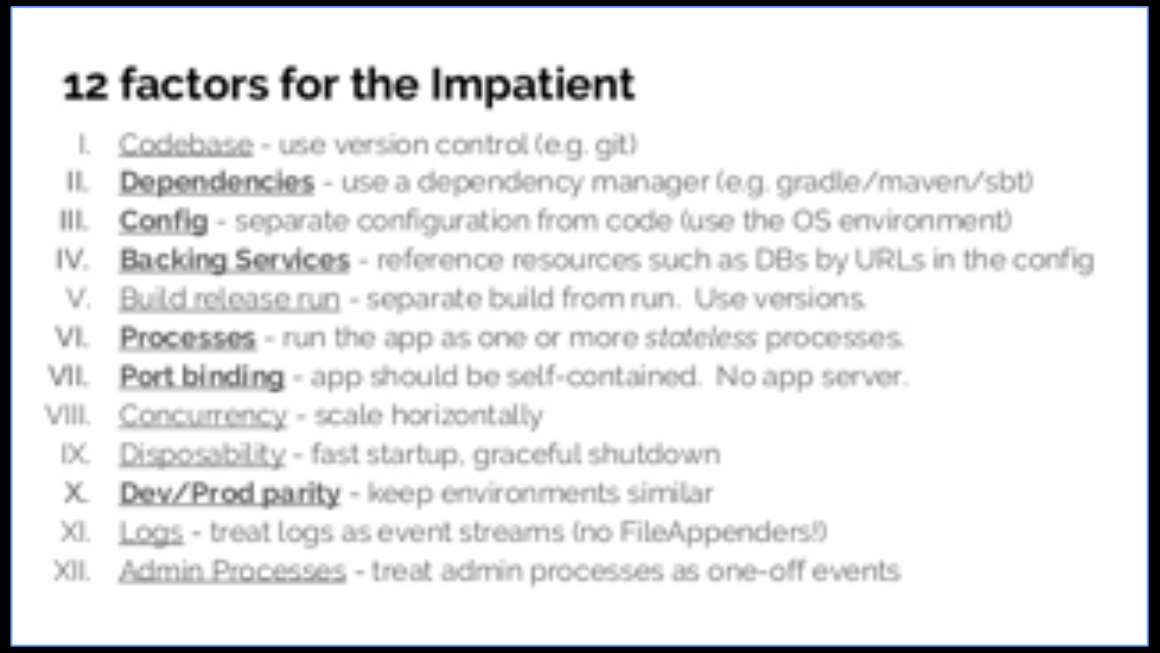
- A runtime is not patched, it's replaced (IX)
- A runtime is stateless (VI)
- Shared functionality in backing services ( IV)

**Elasticity**

- Automatic scale-out and scale-in to maintain performance
- Achieved via containerization

**Pay-as-you-go charging model**

- Pay for what you use




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

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

# ..but we're getting ahead of ourselves

# QUESTIONS?

Kubernetes Workshop Series

# Docker

02

# 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

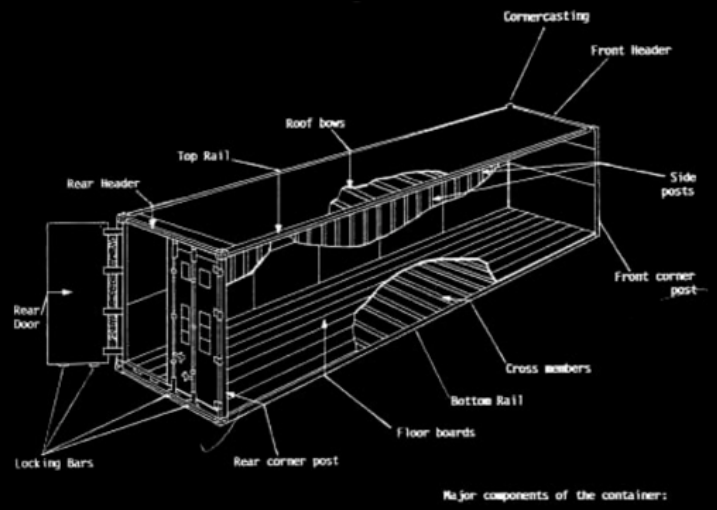
Containers work by **isolating the differences between applications** inside the container so that everything outside the container can be standardized

# 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



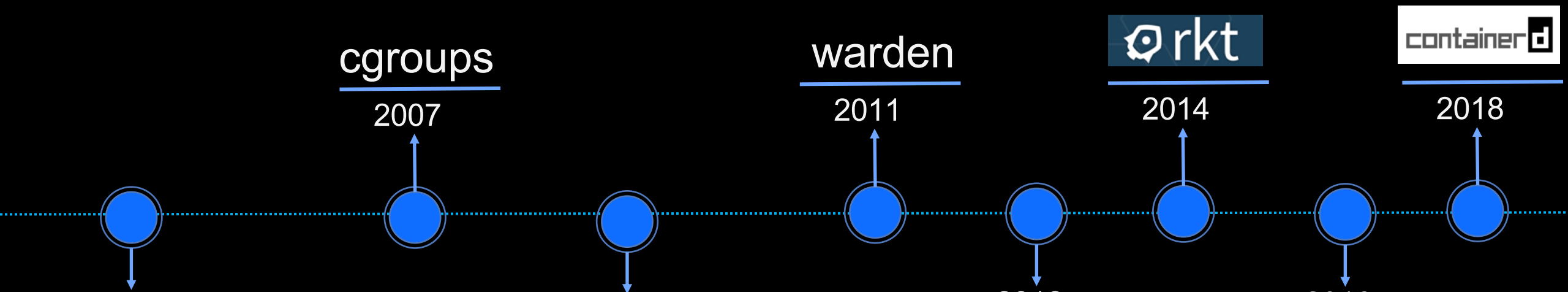
Major components of the container:

### 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




## chroot

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

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.

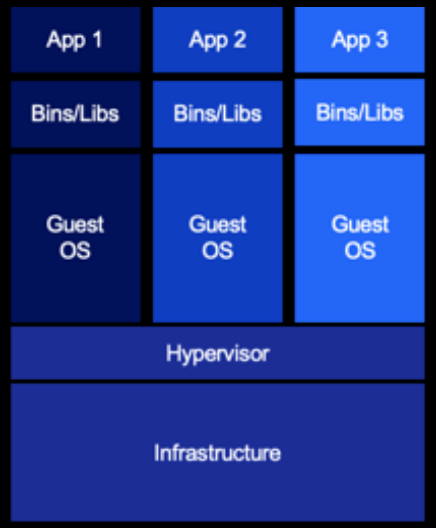
https://en.wikipedia.org/wiki/LXC

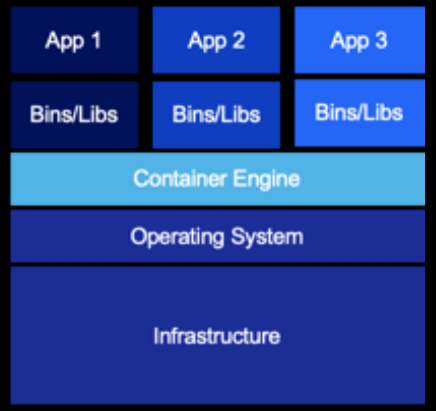
## Docker

## Garden runC

https://containerd.io/

# VMs vs. Containers

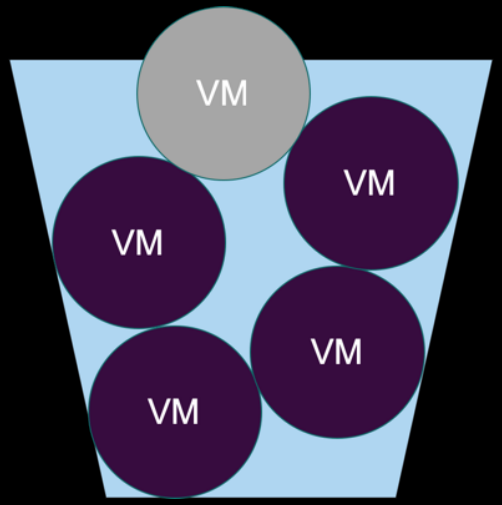


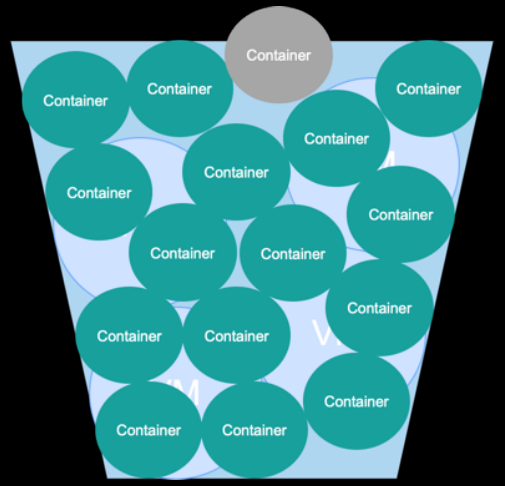




+ VM Isolation
- Complete OS
- Static Compute
- Static Memory

+ 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

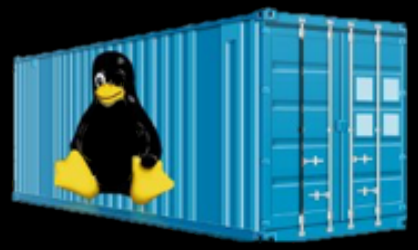
Containers are **portable**

Containers are **easy to manage**

Containers provide **“just enough” isolation**

Containers use hardware **more efficiently**

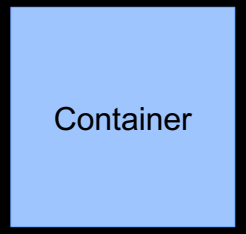
Containers are **immutable**



# Docker Components

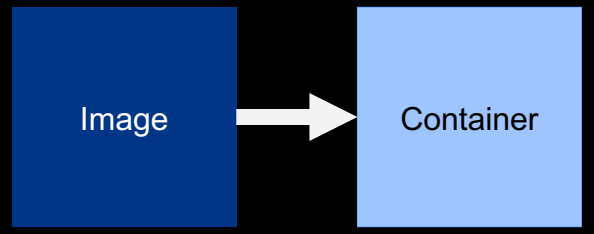
**Container**

Smallest compute unit



# Docker Components

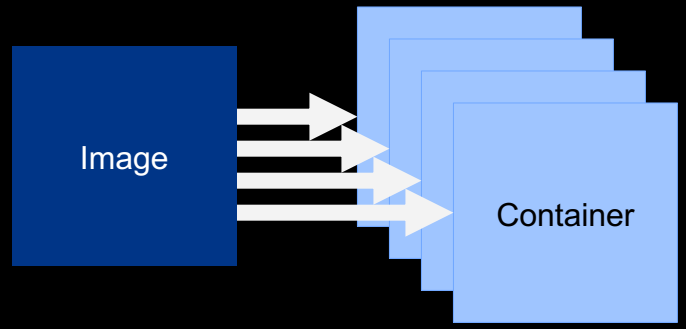
**Containers**
are created from
**Images**




one process per container

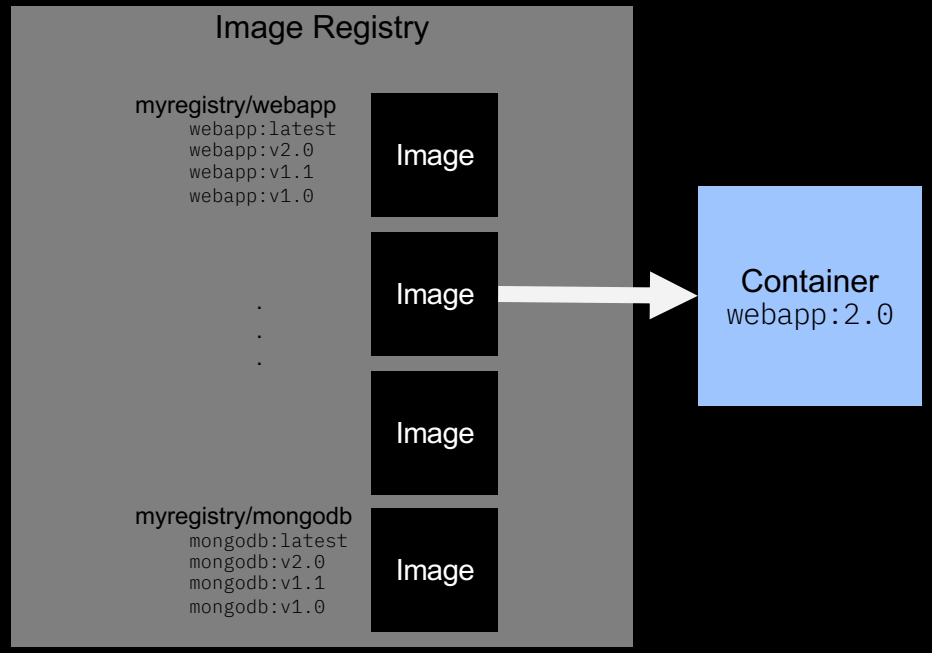
# Docker Components

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

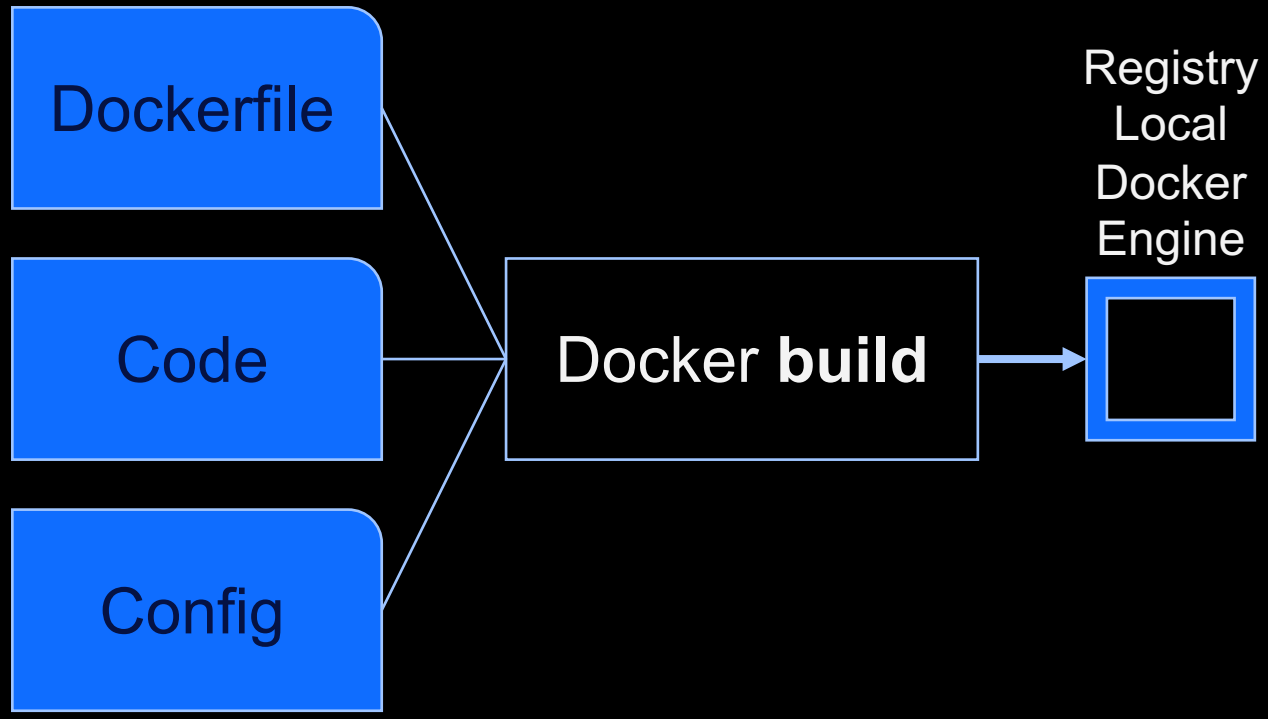

## Docker Hub/Registry

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

# Docker Basics – Build

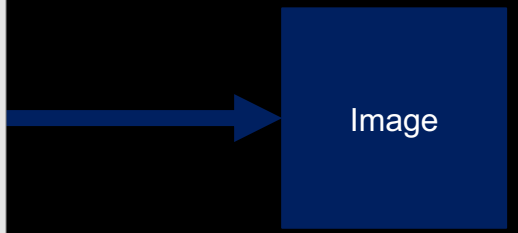


# Docker Basics – Build - Dockerfile

## 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

- 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/

# Expose the port of the web application
EXPOSE 80

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

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

Base Image from:
- hub.docker.com
- quay.io
- your own

Run commands
apt-get, yum, chown, mv, cp, ...

Set environment variables
Can be overriden when running

Add assets to the image
Jar, sources, golang app, ...

Expose a port of the process running in the container

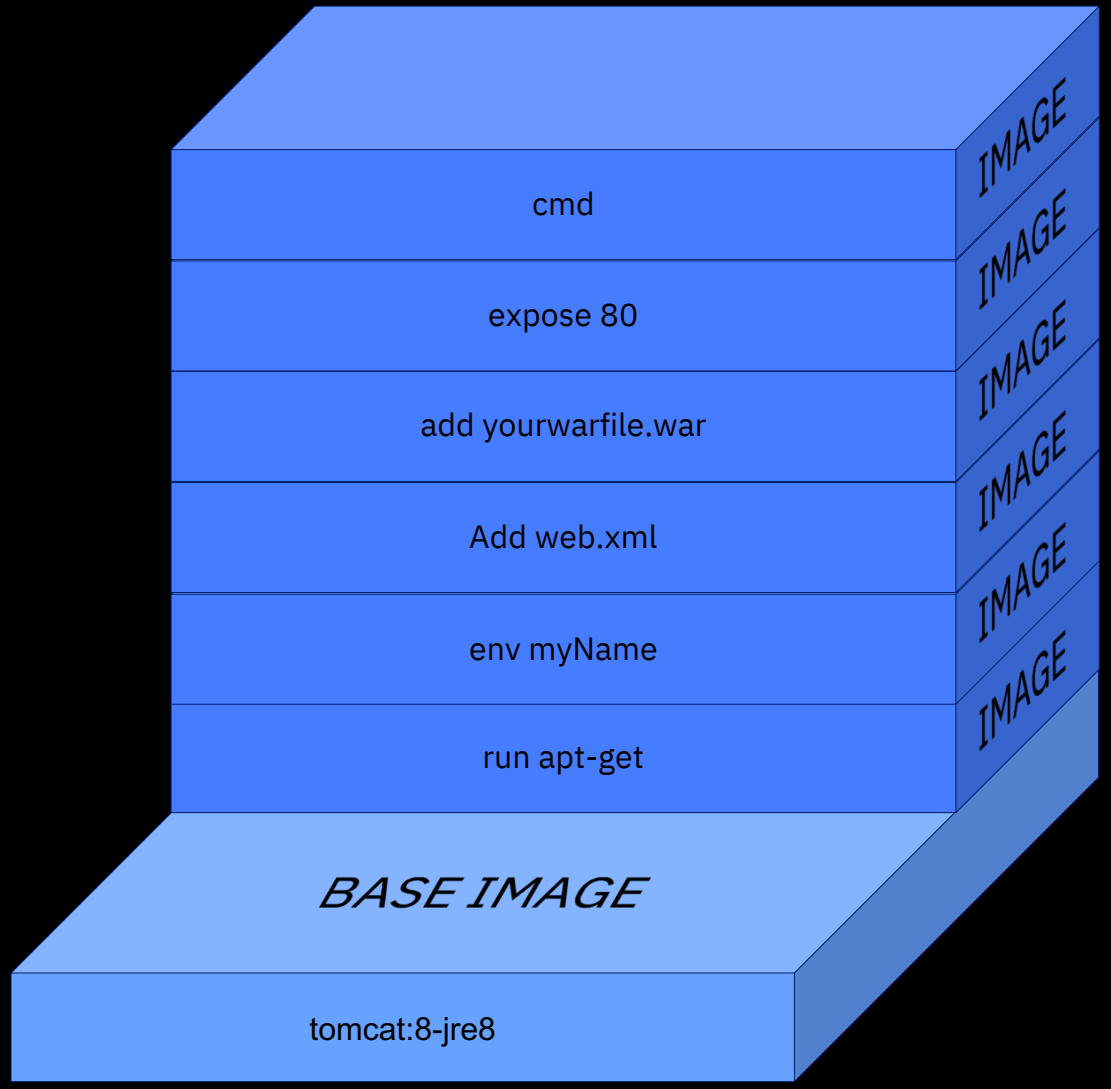
Startup command when image is being run

# Docker Basics – Layered File System

## Docker Layers
### Inheritance

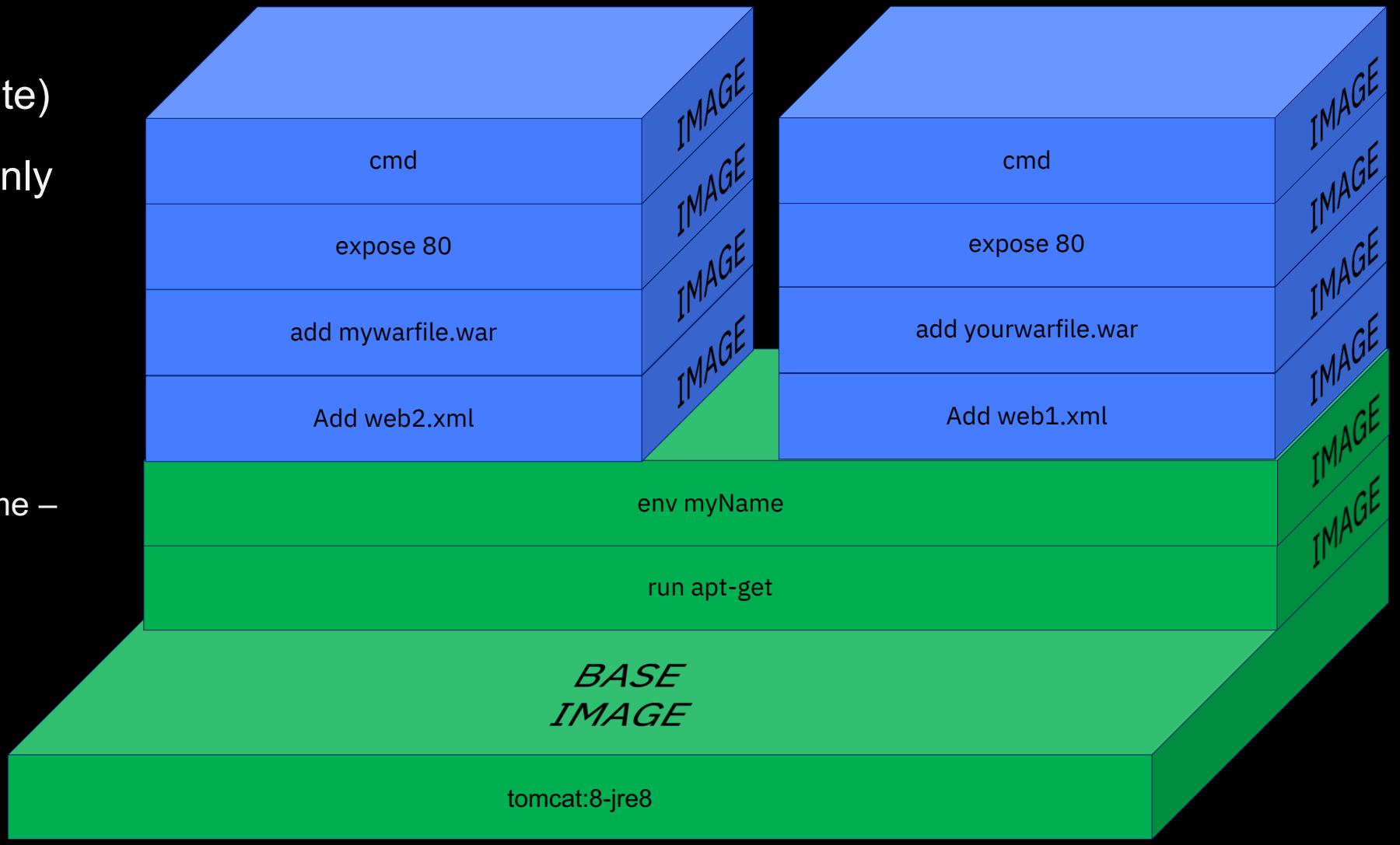
- Build on the work of those who came before you

```
# 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/

# Expose the port of the web application
EXPOSE 80

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

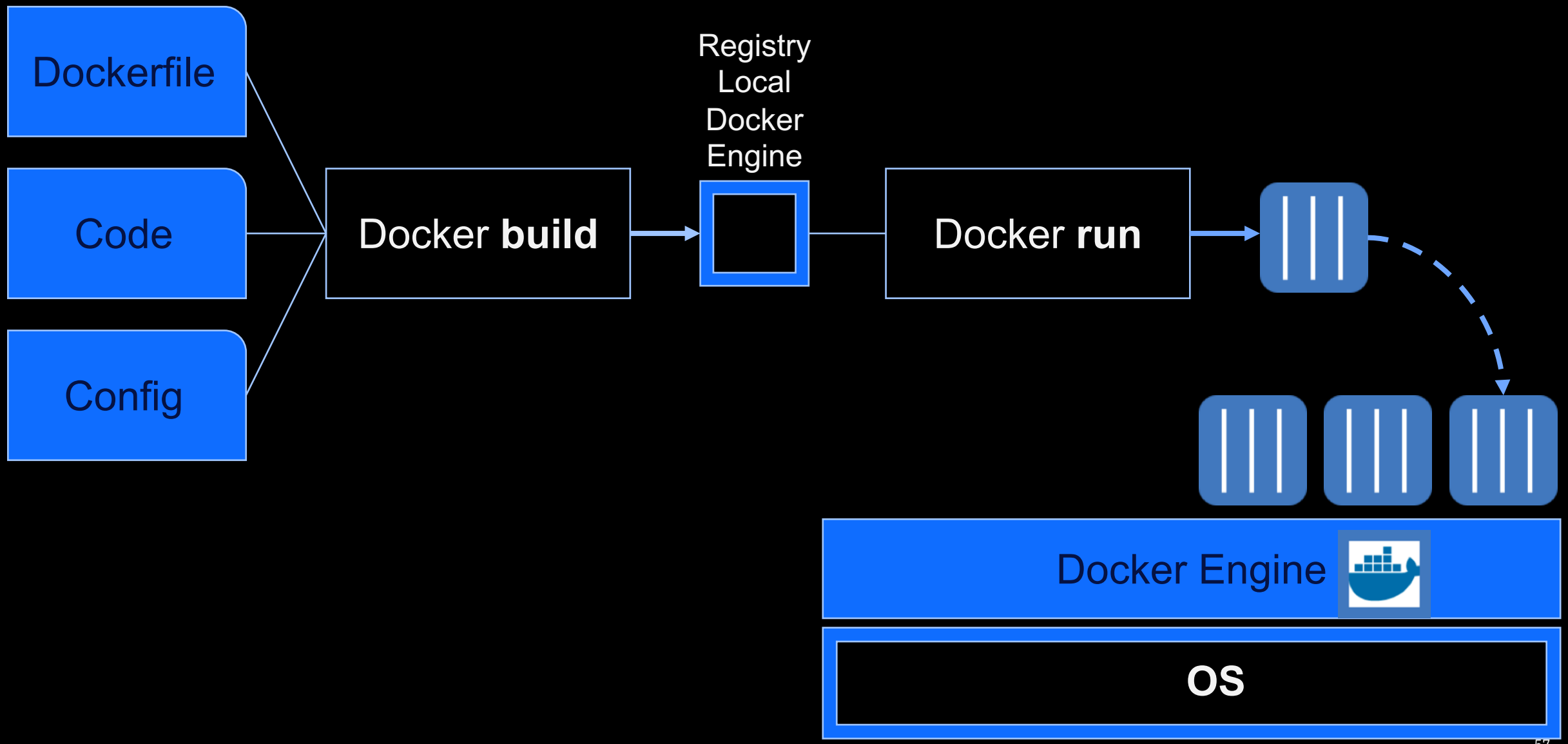


# Docker Basics – Layered File System

- 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 – Run



# Docker Basics – Run

## Docker Layers

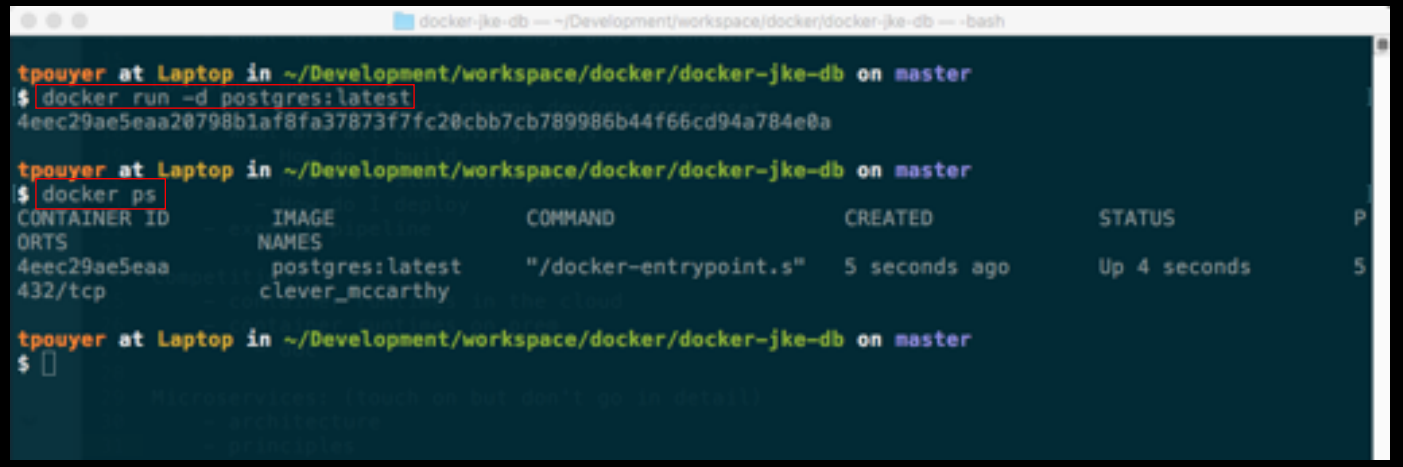
### Inheritance

- 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 – Run

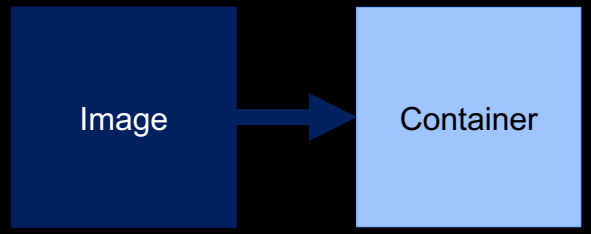
## Docker run

- Local (containerd)



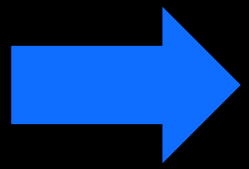

```
docker run -d -e MYVAR=foo -p 8080:80 myimage:1.0.0
docker run -ti myimage:1.0.0 /bin/bash
```

…



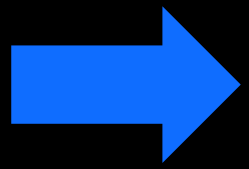
# Docker Basics – Run – CMD vs ENTRYPOINT

CMD echo “Hello World”



```
mac> sudo docker run [image_name]
mac> Hello World

mac> sudo docker run [image_name] hostname
mac> my-host-name
```

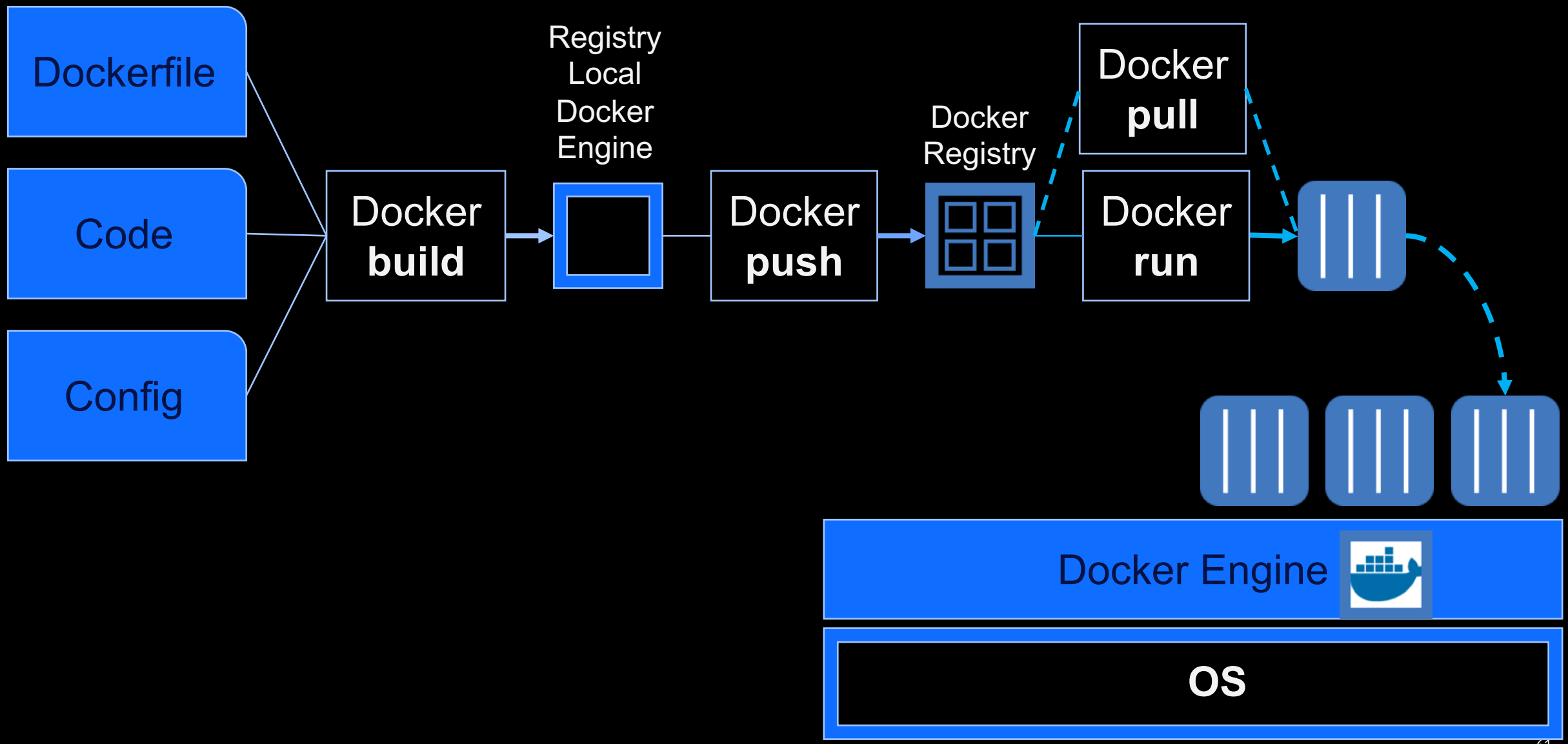
ENTRYPOINT echo "Hello World”



```
mac> sudo docker run [image_name]
mac> Hello World

mac> sudo docker run [image_name] hostname
mac> Hello World  hostname
```

https://phoenixnap.com/kb/docker-cmd-vs-entrypoint

# Docker Basics – Store, Retrieve & Run with registry



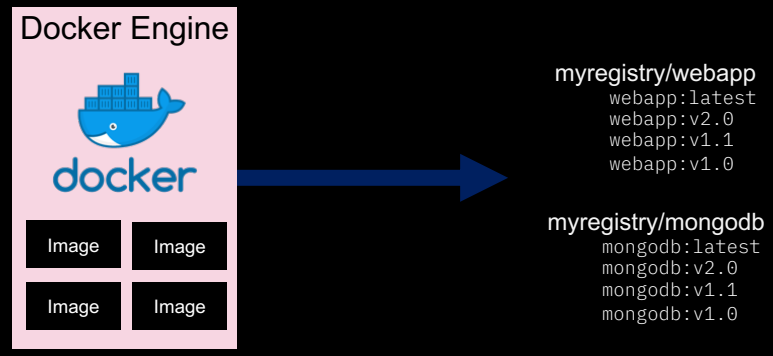
# Docker Basics – Store & Retrieve

## Docker Registry

- Private Local
- Public Docker Hub
- Private Shared

```
docker images
```

```
tpouyer 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
stackinabx/demo-jke                 1.1                 a596fbf2c3b7        3 days ago          672 MB
stackinabx/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                  latest              54bdbf42b999        4 days ago          327.5 MB
stackinabox/urbancode-deploy-client 6.2.2.0             53c878fbf034        7 days ago          477 MB
websphere-liberty                   javaee7             7e27c3fb9c96        10 days ago         445.7 MB
mysql                               5.6                 a896fd82dcd5        13 days ago         327.5 MB
mysql                               latest              f3694c67abdb        13 days ago         400.1 MB

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




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

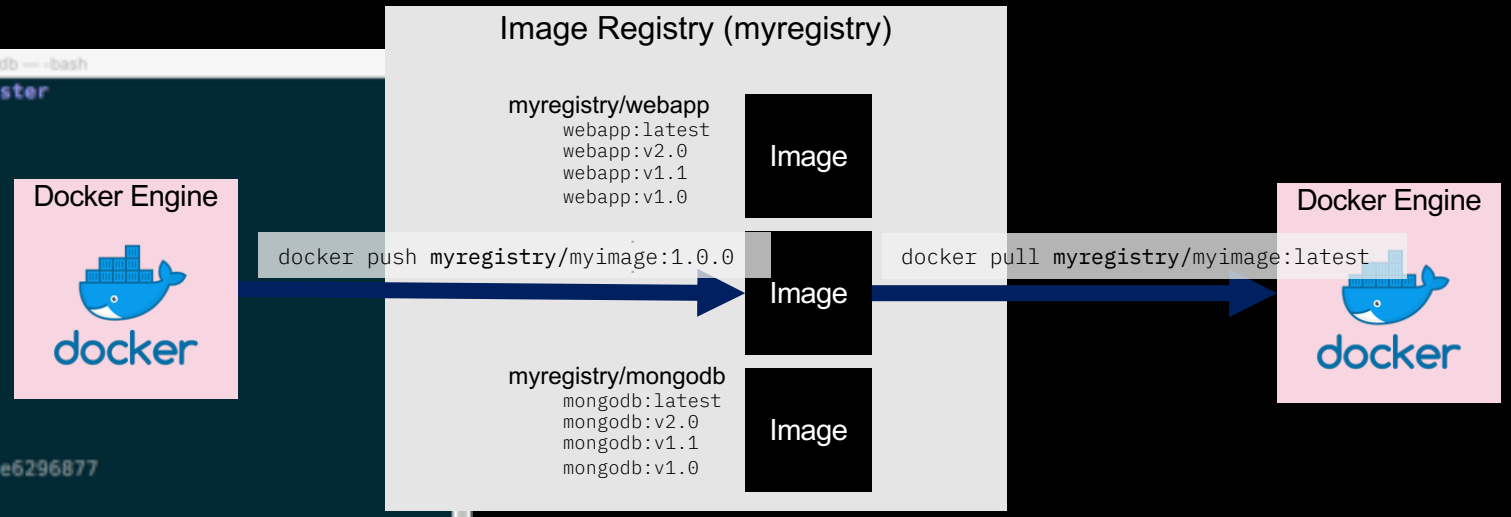
myregistry/mongodb
- mongodb:latest
- mongodb:v2.0
- mongodb:v1.1
- mongodb:v1.0

# Docker Basics – Store & Retrieve with registry

## Docker Registry

- Private Local
- Public Docker Hub
- Private Shared

```
docker tag myimage:1.0.0 myregistry/myimage:1.0.0
docker push myregistry/myimage:1.0.0

docker pull myregistry/myimage:latest
```

# 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 **Quay** or **Docker Hub**
- **https://quay.io**
- **https://hub.docker.com**

Login into the registry

- Docker login domain:port

# 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?

Kubernetes Workshop Series

# **Let‘s get real**

# 03

# 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 or reduced benefits if application doesn’t need scaling or cloud transformation
- 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
- Higher initial 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

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

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

# Microservices – breaking down the ~~monolith~~ beast

## How to identify candidates

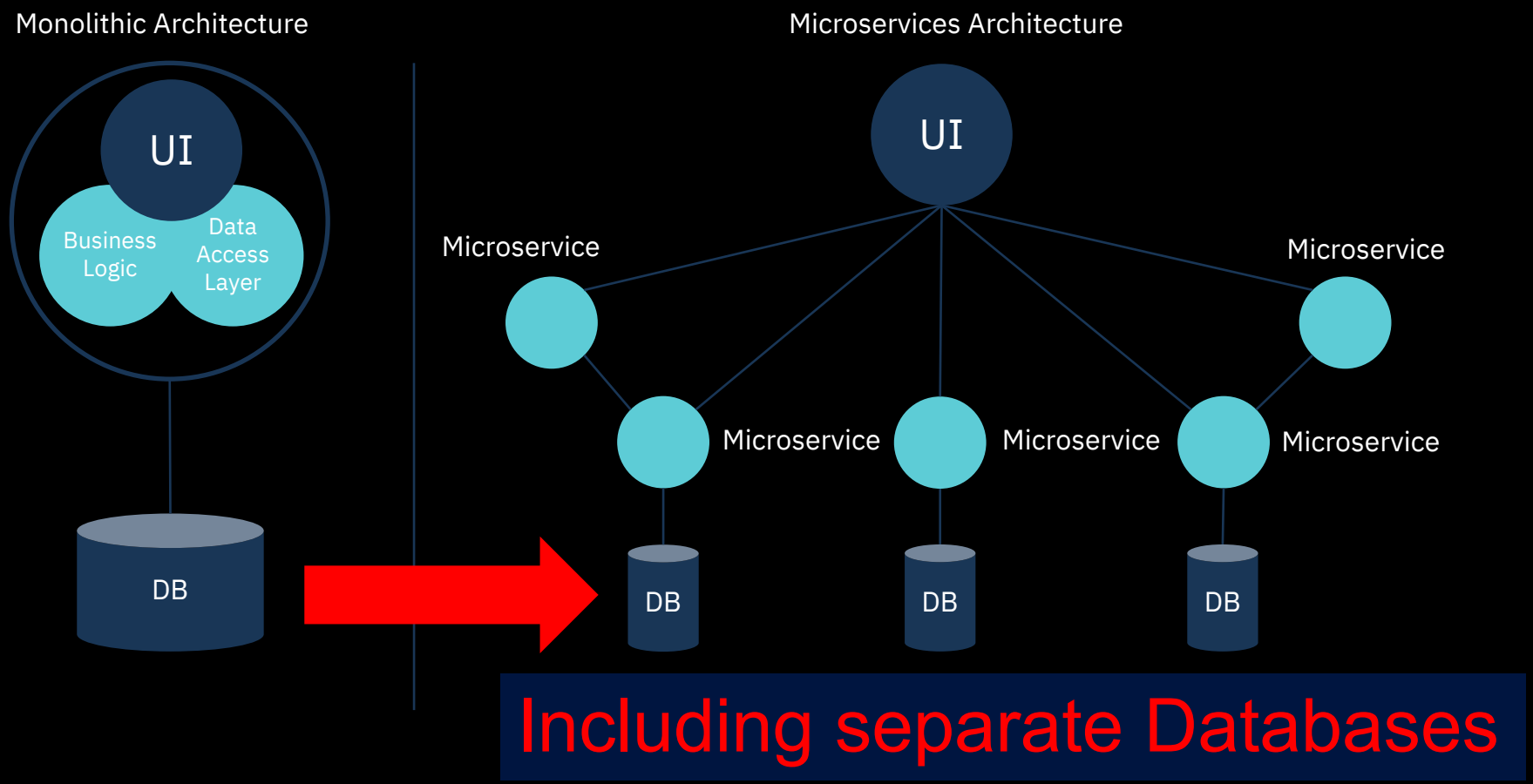
- Is there anything that is scaling differently than the rest of the system?
- Is there anything that feels “tacked-on”?
- Is there anything changing much faster than the rest of the system?
- Is there anything requiring more frequent deployments than the rest of the system?
- Is there a part of the system that a small team, operates independently?
- Is there a subset of tables in your datastore that isn’t connected to the rest of the system?

Source: https://www.simplethread.com/youre-not-actually-building-microservices/

# Databases

# Microservices – Database refactoring

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



# Microservices – Database refactoring

## Containerizing your WAR file doesn’t mean you’re doing microservices.

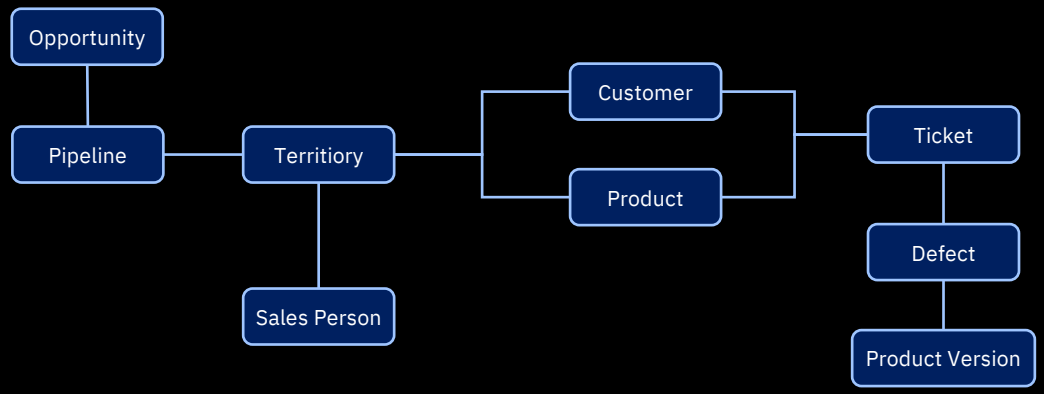
- What is the domain? What is reality?
- Where are the transactional boundaries?
- How should microservices communicate across boundaries?
- What if we just turn the database inside out?

Source: https://martinfowler.com/bliki/BoundedContext.html

# Microservices – Bounded Context

## Bounded Context  is a central pattern in Domain-Driven Design

- DDD deals with large models by dividing them into different Bounded Contexts

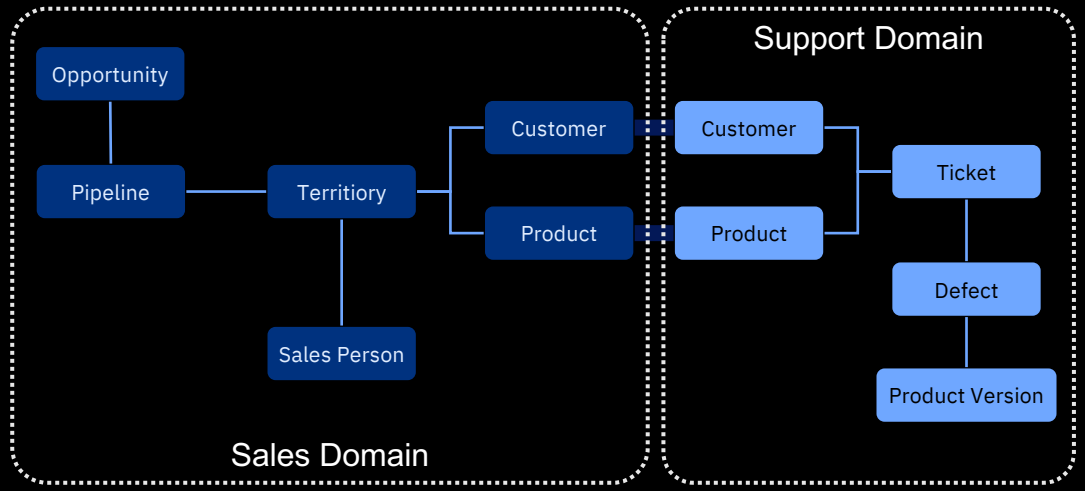



Source: https://martinfowler.com/bliki/BoundedContext.html

# Microservices – Bounded Context

## Bounded Context  is a central pattern in Domain-Driven Design

- DDD deals with large models by dividing them into different Bounded Contexts
- Helps to build and refine a model that represents our domains, contained within a boundary that defines our context.

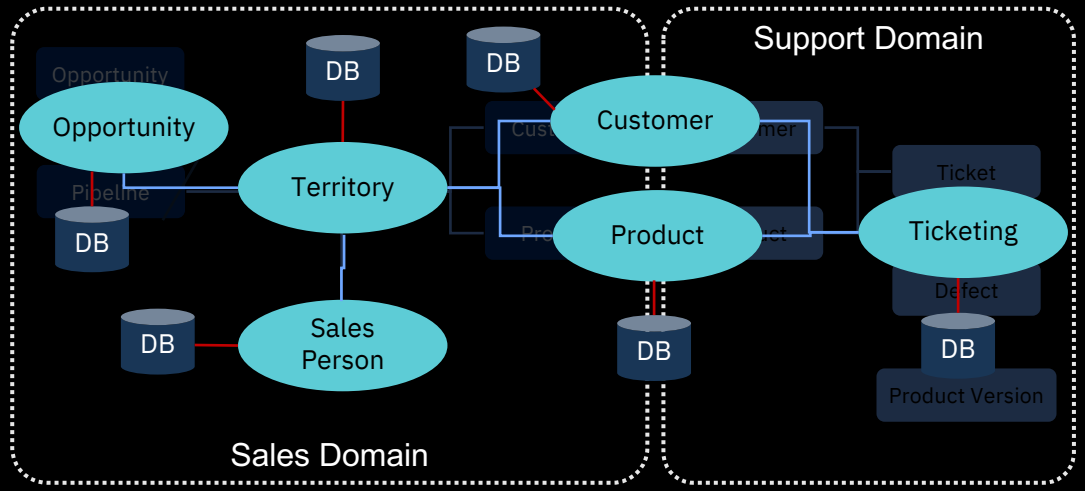



Source: https://martinfowler.com/bliki/BoundedContext.html

# Microservices – Bounded Context

## Bounded Context  is a central pattern in Domain-Driven Design

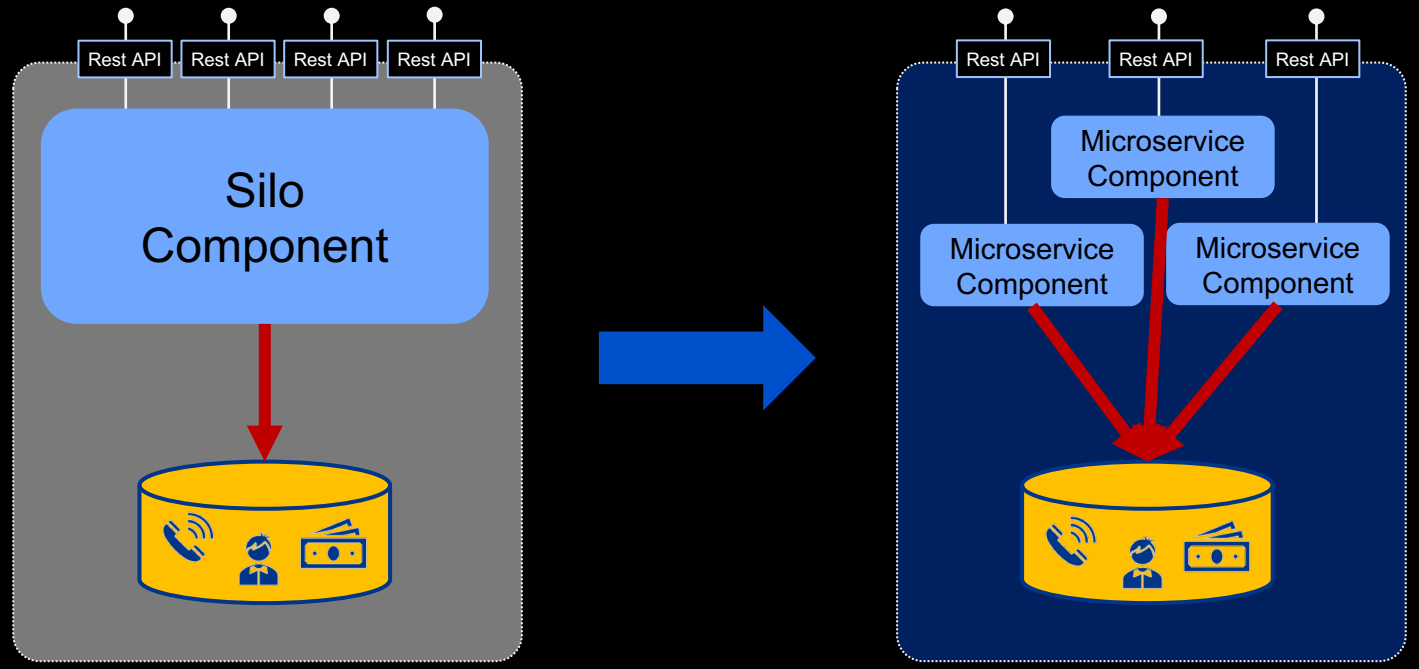
- DDD deals with large models by dividing them into different Bounded Contexts
- Helps to build and refine a model that represents our domains, contained within a boundary that defines our context.
- These boundaries end up being our microservices




Source: https://martinfowler.com/bliki/BoundedContext.html

# Microservices – Database refactoring

**Common mistake**

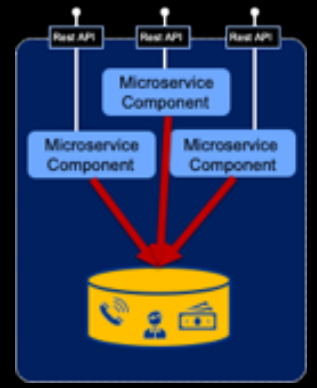



Source: IBM Architecture Center

# Microservices – monolithic database

## Challenges

- Inability to deploy your service changes independently through tight coupling.
- Schema changes need to be coordinated amongst all the services
- Difficult to scale individual services
- All your services have to use a relational database even when a no-SQL datastore would bring benefits
- Difficult to improve application performance (DB/Table size)

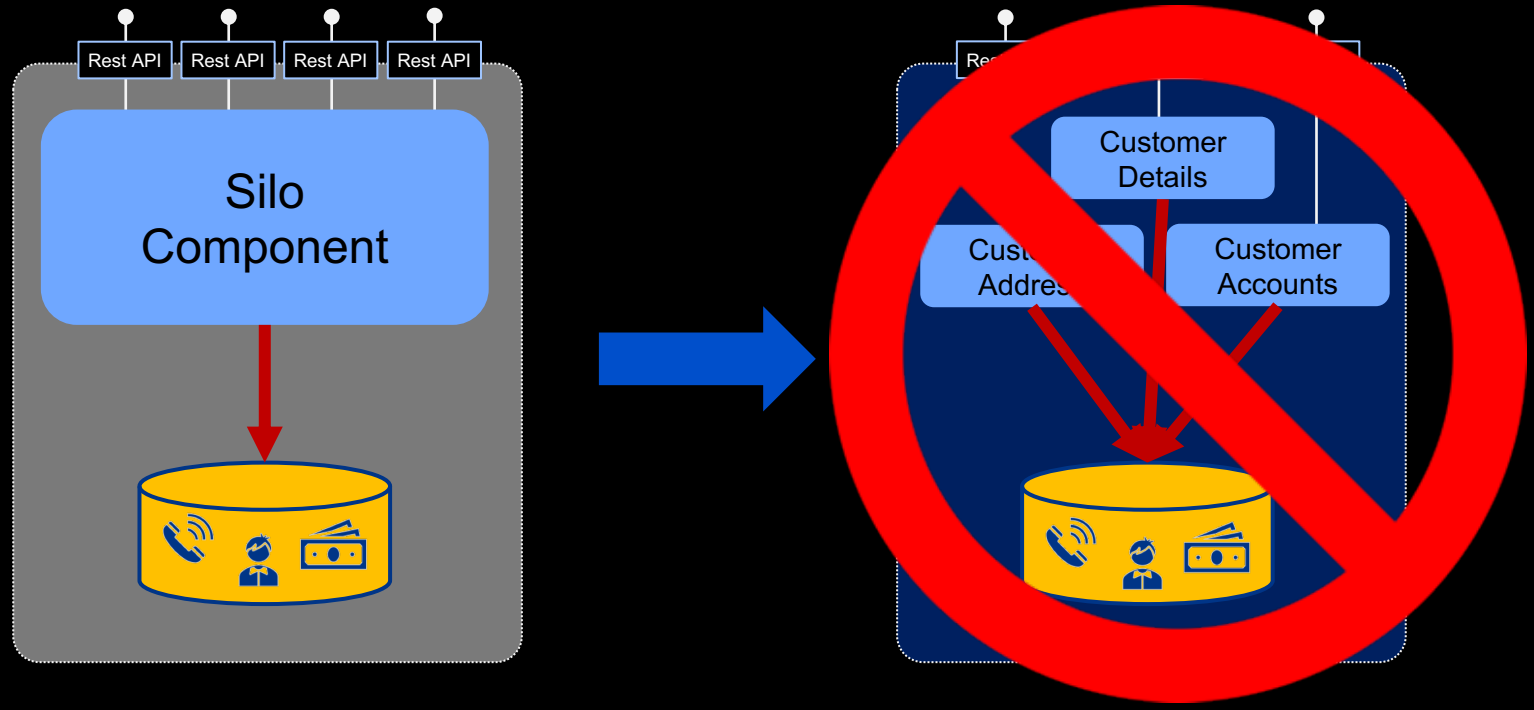



Source: https://dzone.com/articles/breaking-the-monolithic-database-in-your-microserv

# Microservices – Database refactoring

## Data Isolation

Each microservice must keep its own data. One typical error that happens in the beginning of a microservice transformation is the use of a centralized database, the same way it was used in the monolithic application. This creates a single point of failure and dependency between the microservices.

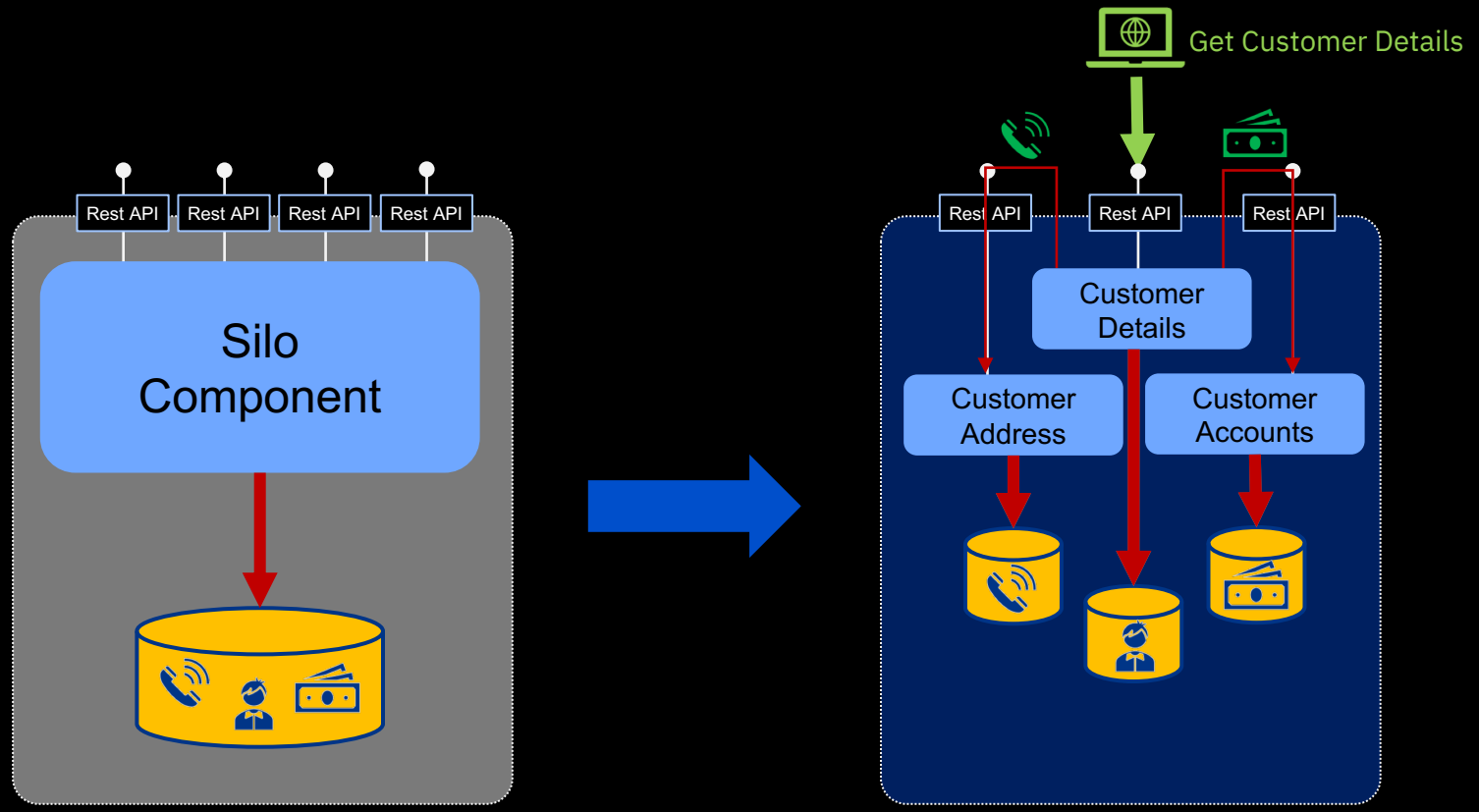



Source: IBM Architecture Center

# Microservices – Database refactoring

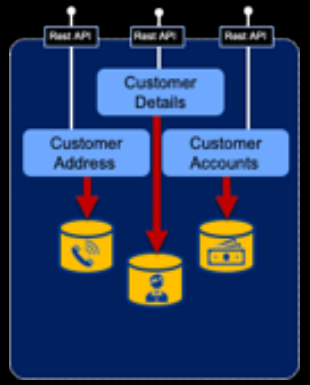
## Data Isolation

Each microservice must keep its own data. One typical error that happens in the beginning of a microservice transformation is the use of a centralized database, the same way it was used in the monolithic application. This creates a single point of failure and dependency between the microservices.




Source: IBM Architecture Center

# Microservices – per-service database

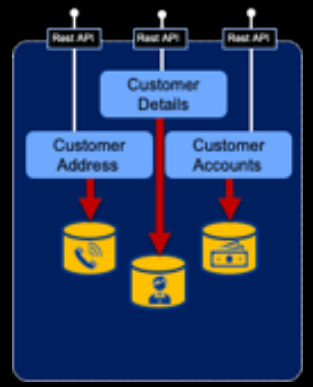



## Benefits

- Avoid unexpected data modification – the API enforces consitency
- Make changes to our system without blocking progress of other parts of the system
- Store the data in a project-owned databases, using the appropriate technology
- Make changes to the schema/databases at our leisure
- Become much more scalable, fault tolerant, and flexible

Source: https://blog.christianposta.com/microservices/the-hardest-part-about-microservices-data/

# Microservices – per-service database




## Drawbacks

- Costly to refactor
- Introduces latency
- Needs robust transaction handling by the service (cost!)
- More difficult to debug
- More difficult to operationalize
- More difficult to test (needs well designed API tests)

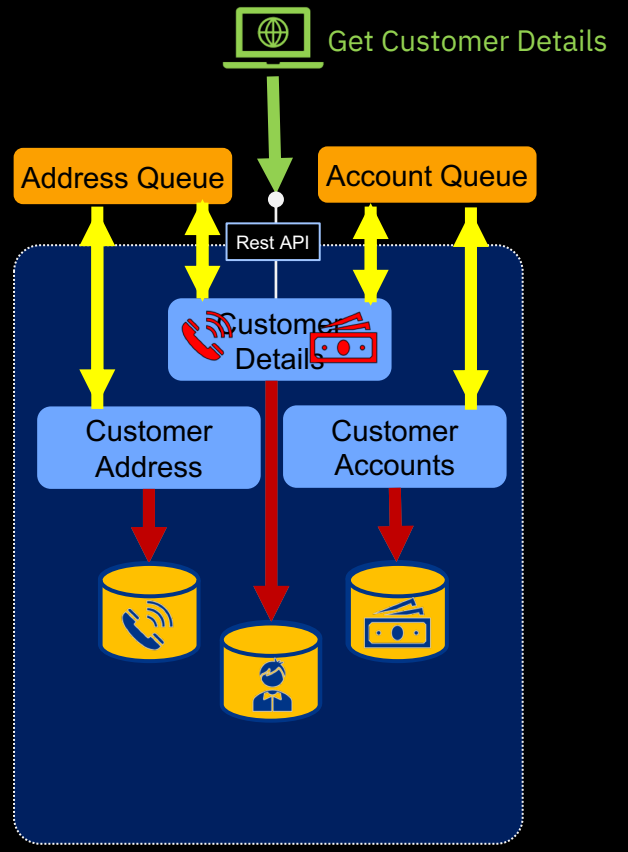
Source: https://blog.christianposta.com/microservices/the-hardest-part-about-microservices-data/

# Event Driven Architecture

# Microservices – one step further

## Event-Driven Architecture

- Common pattern to maintain data consistency across different services.
- Avoid waiting for ACID transactions to complete
- Makes your application more available and performant
- Provides loose coupling between services




Source: https://dzone.com/articles/breaking-the-monolithic-database-in-your-microserv

Microservices

# Good or bad idea?

# Microservices – reasons not to use them

## Some thoughts

- If speed is your goal, microservices aren't the solution (MVP, doesn’t have to scale)
- Applications that require tight integration between individual components and services (real-time processing, …)
- At what point is it in its lifecycle? Mission-critical?
- A moderately large, moderately complex application being maintained by a relatively small development and operations team
- Not all applications are large enough to break down into microservices

Source: https://blog.christianposta.com/microservices/when-not-to-do-microservices/

# Microservices – reasons not to use them

## How to detect antipatterns

- A change to one microservice often requires changes to other microservices
- Deploying one microservice requires other microservices to be deployed at the same time
- Your microservices are overly chatty
- The same developers work across a large number of microservices
- Many of your microservices share a datastore
- Your microservices share a lot of the same code or models

Source: https://www.simplethread.com/youre-not-actually-building-microservices/

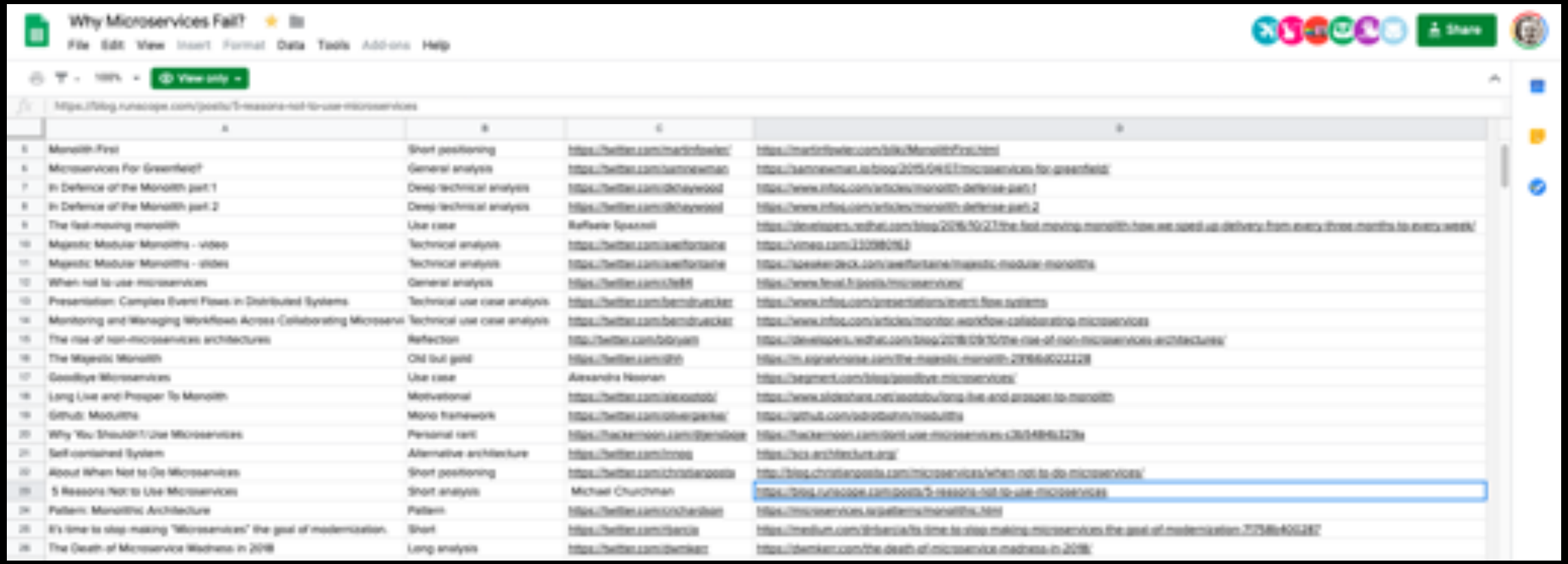
# Microservices – reasons not to use them

What works for **large and complex** applications
does not always work at a **smaller scale**

What makes sense for a **new application**
does not always make sense when maintaining or
updating **existing applications**

Source: https://blog.runscope.com/posts/5-reasons-not-to-use-microservices

# Microservices – reasons not to use them




https://docs.google.com/spreadsheets/d/1vjnjAII_8TZBv2XhFHra7kEQzQpOHSZpFIWDjynYYf0/edit?pli=1#gid=0

# QUESTIONS?

# Everybody Loves Containers

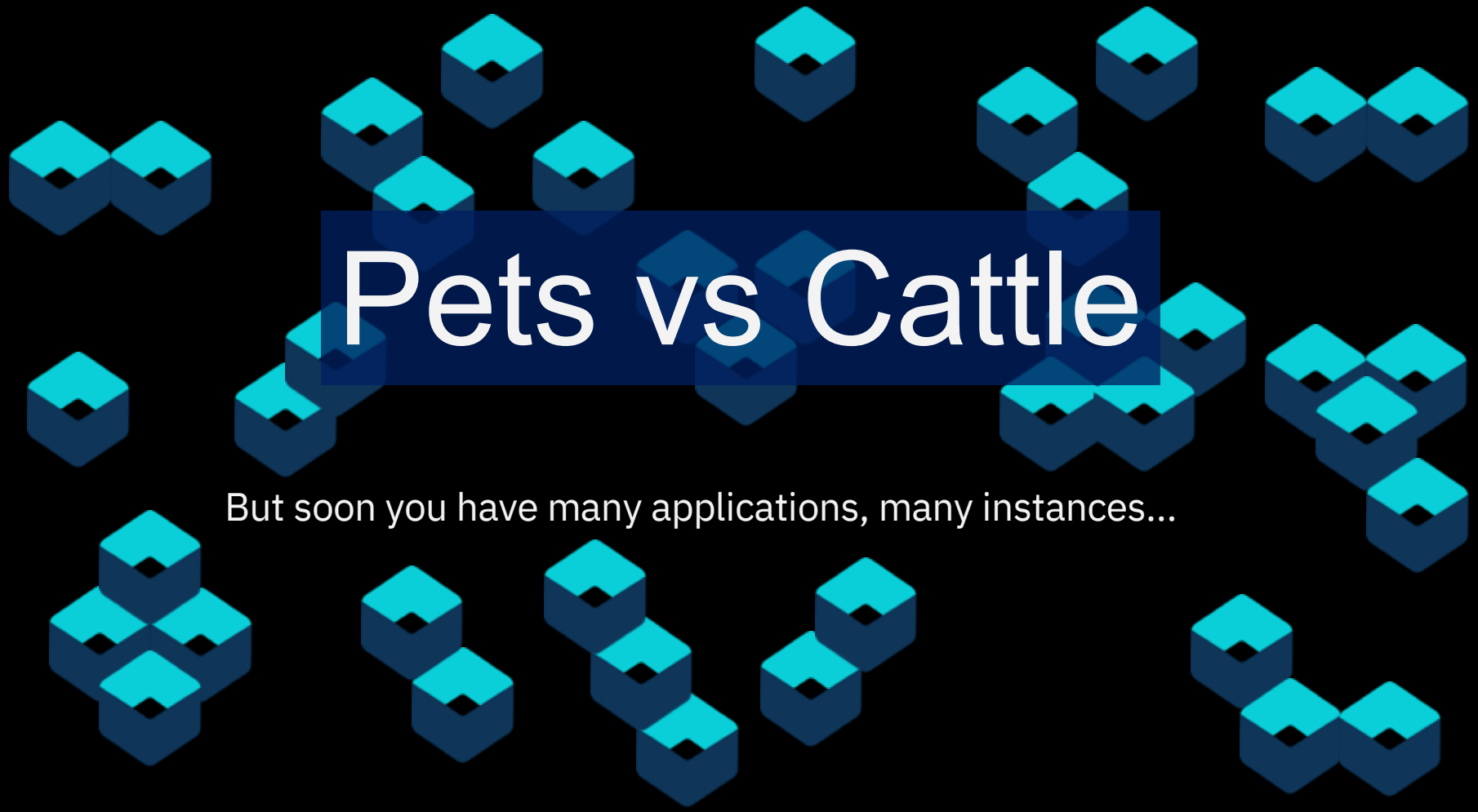
## But when you go

From this….

To this….

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...

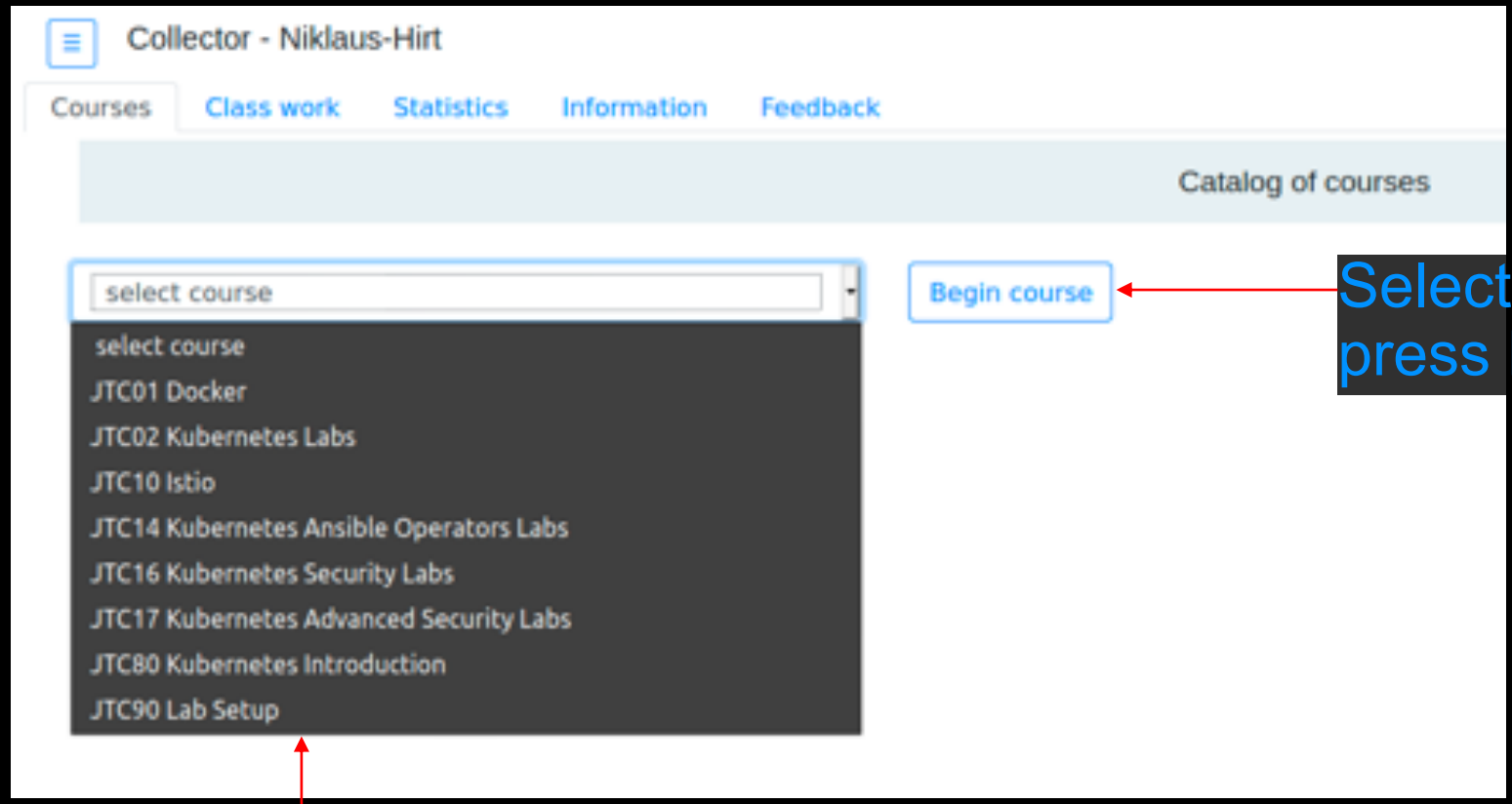
But more on
this next week.....

Kubernetes Workshop Series
**Docker -** *Hands-On*

# 04

# Starting Course JTC01 Docker

Name will be shown




Select course and press button to begin

Current course catalog

# JTC01 Labs

## Lab 1 : Docker basics

## Lab 2 : Docker – Working with Containers

- Build a Docker image
- Run a Docker image
- Use the Portainer tool
- Deploy a more complex Docker application
- Push a Docker image into a registry

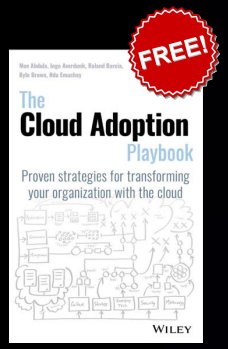
## Lab 3 : Docker internals

# READY SET GO!!!!

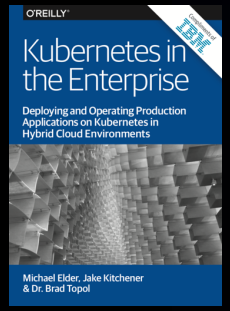
Duration: 60 mins

# QUESTIONS?

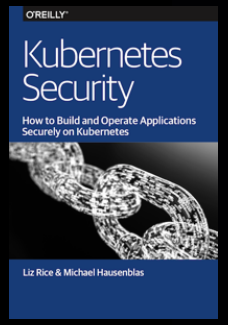
# Kubernetes – Some Reading Tips




The de facto guide to improving your enterprise with the cloud, created by distinguished members of our Solution Engineering team
http://ibm.biz/playbook




Deploying and Operating Production Applications on Kubernetes in Hybrid Cloud Environments
https://ibm.co/2LQketN (excerpt)




https://kubernetes-security.info/

# Sources and documentation will be available here:

https://github.com/niklaushirt/k8s_training_public

https://github.com/niklaushirt/training

**See you next week!**

- **Same place**
- **Same time**

Kubernetes Workshop Series

**Kubernetes**

# Niklaus Hirt

nikh@ch.ibm.com

@nhirt

# THANK YOU!!!!