Application Modernization Workshop

Mangesh Patankar
– IBM Developer Advocate

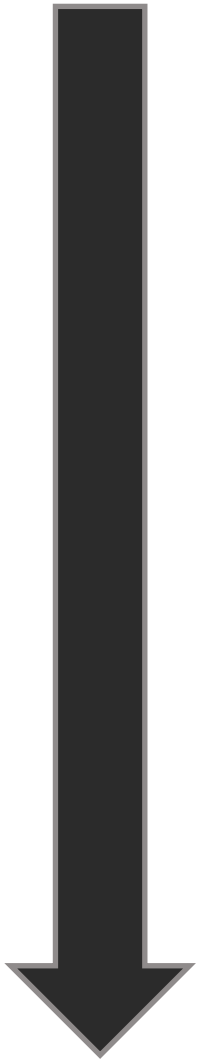
IBM
CODE

Agenda

- Evolution of Application Architectures
- Microservices – Key Tenets, Do’s Don’ts
- App Modernization – Things to know
- IBM Approach
- IBM Cloud Native Stack
- Lab: Deploy Microservices application on k8s

IBM CODE

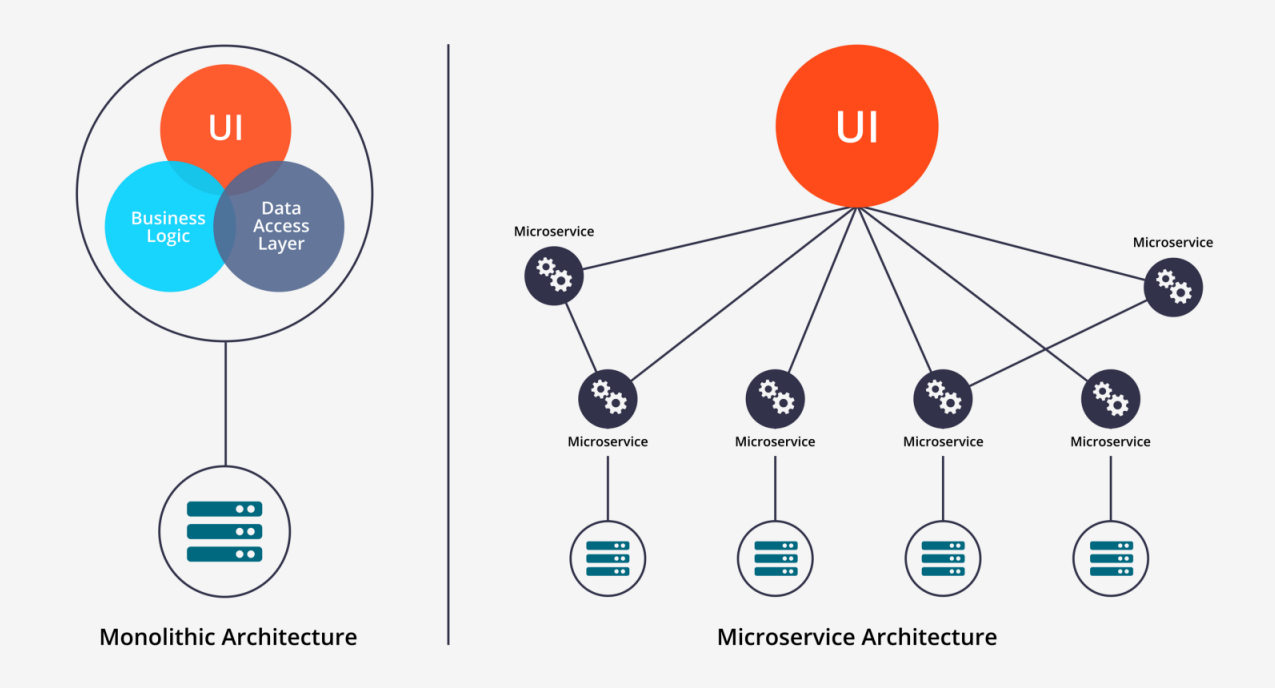
Evolution of application architectures



Late 90’s	**Enterprise Application (EAI) Services and Models** Addressed integration and transactional challenges primarily by using message oriented middleware. Mostly proprietary systems needing a proliferation of custom interfaces.
Mid 00’s	**Service Oriented Architectures** Based on open protocols like SOAP and WSDL making integration and adoption easier. Usually deployed on an Enterprise ESB which is hard to manage and scale.
Early 10’s	**API Platforms and API Management** REST and JSON become the defacto standard for consuming backend data. Mobile apps become major consumers of backend data. New Open protocols like OAuth become available further simplifying API development .
2015 and beyond	**Microservice Architecture** Applications are composed of small, independently deployable processes communicating with each other using language-agnostic APIs and protocols.

IBM CODE

Microservices



An engineering approach focused on decomposing an application into single-function modules with well defined interfaces which are independently deployed and operated by a small team who owns the entire lifecycle of the service.

Microservices accelerate delivery by minimizing communication and coordination between people while reducing the scope and risk of change.

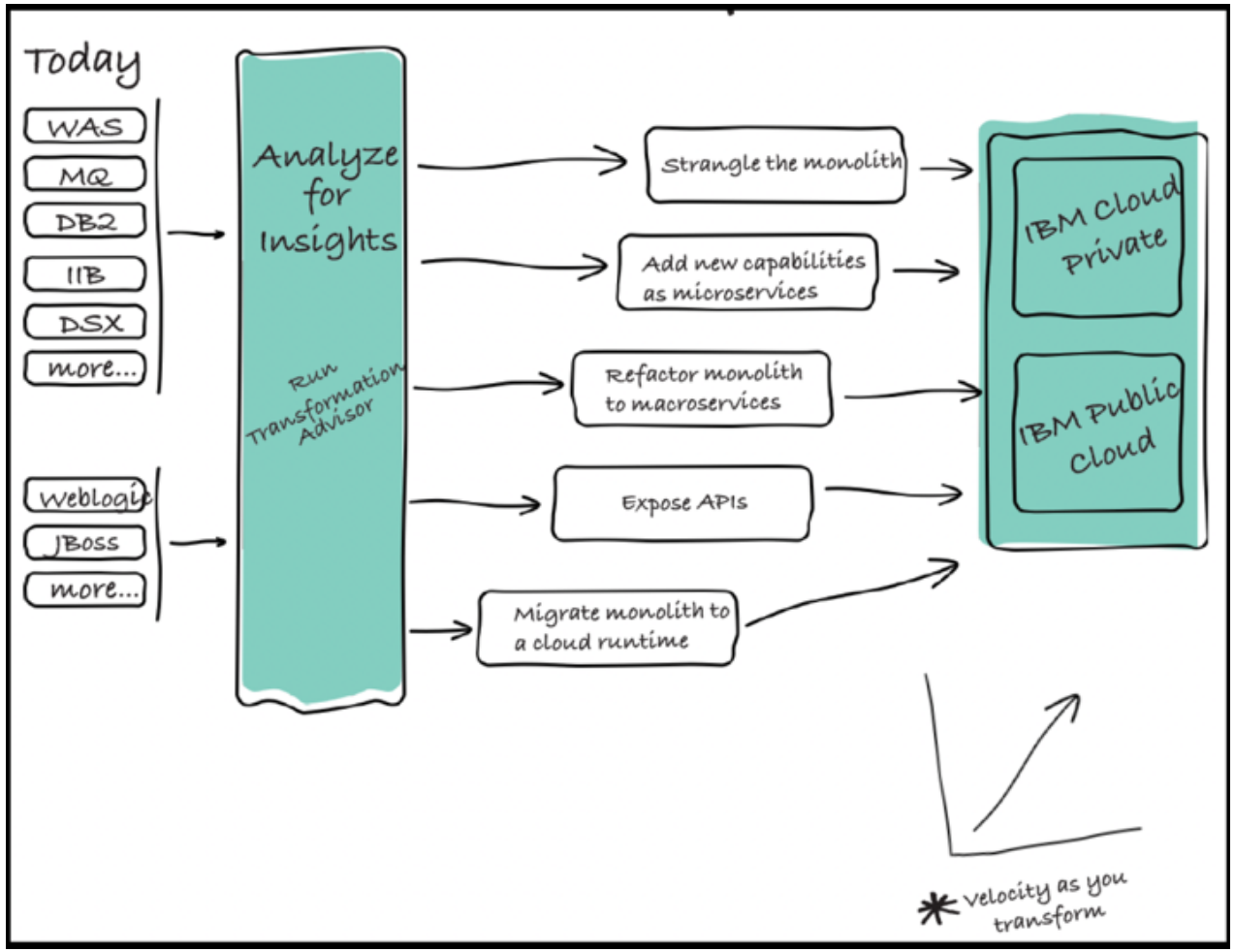
IBM CODE

Microservice versus Monolith

	Monolith	**Microservice**
Architecture	Built as a single logical executable (typically the server-side part of a three tier client-server-database architecture)	Built as a suite of small services, each running separately and communicating with lightweight mechanisms
Modularity	Based on language features	Based on business capabilities
Agility	Changes to the system involve building and deploying a new version of the entire application	Changes can be applied to each service independently
Scaling	Entire application scaled horizontally behind a load- balancer	Each service scaled independently when needed
Implementation	Typically written in one language	Each service implemented in the language that best fits the need
Maintainability	Large code base intimidating to new developers	Smaller code base easier to manage

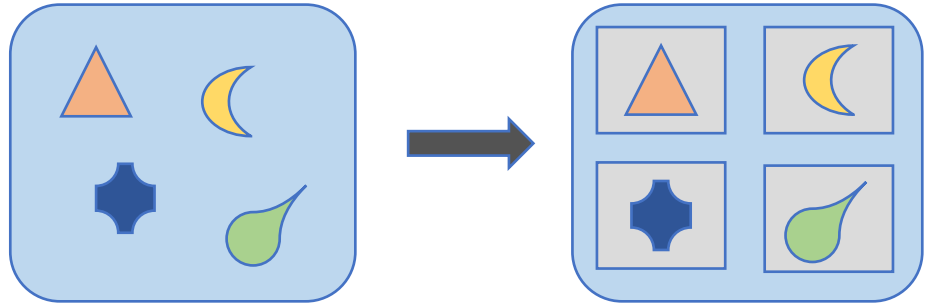
What Does App Modernization Mean Anyway ?

- Containerize an Existing Application / Workload ?
- Refactor Applications into Microservices ?
- Strangle Monolith over time with new Microservices ?
- Completely rewrite into new microservices ?
- Automate deployment ?
- Lift and Shift into a Cloud ?
- Expose Applications through API's ?
- Augment Old Code with new microservices ? APIs / Services (AI, Data Science) ?
- What About My Data !!!!!!!!



Key tenets of a microservices architecture

1. **Large monoliths are broken down into many small services**
   - Each service runs into its own process
   - Generally accepted rule is one service per container
2. **Services are optimized for a single function**
   - One business function per service
     - The service will have only one reason to change

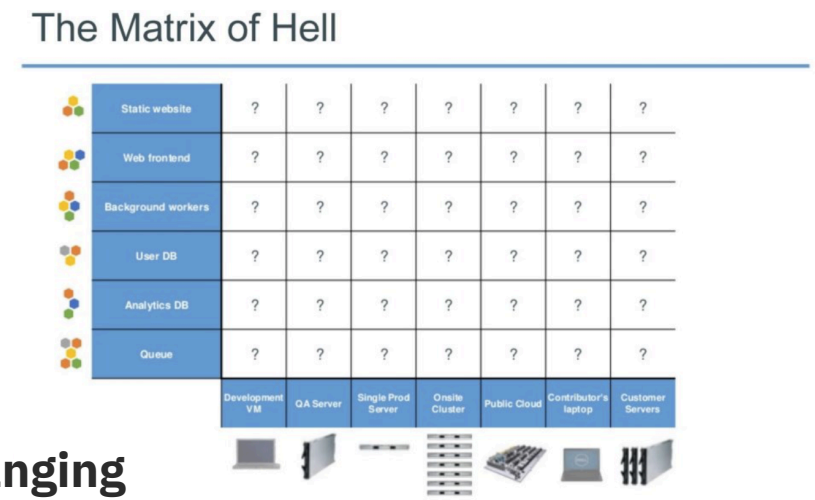


3. **Services are tightly encapsulated behind concrete programming interfaces**
   - Have to balance between evolving the interface and maintaining backward compatibility
4. **Communication via REST API and/or message brokers**
   - Avoids tight coupling and allows for flexibility of synchronous and asynchronous access
5. **Per-service continuous integration and continuous deployment (CI/CD)**
   - Services can evolve at different rates
6. **Per-service HA and clustering**
   - Services can be scaled independently at different rates as needed

Microservices Dos and Don’ts

1. **Don’t assume all monolithic apps are microservice candidates**
   - Stable monolithic applications that aren’t subject to frequent updates will not provide the return on investment. Good candidate applications have multiple upgrades per year, lots of business rules requiring complex regression testing and require extended service outages to implement
2. **Don’t even think about microservices without DevOps**
   - Microservices cause an explosion of moving parts. It is not a good idea to attempt to implement microservices without serious deployment and monitoring automation
3. **Don’t try to migrate all external dependencies at once**
   - Microservices often introduce multiple databases, message brokers, data caches, and similar services that all need to be maintained, clustered, and kept in top shape. It really helps if your first attempt at microservices is free from such concerns
4. **Don’t create too many microservices**
   - Each new microservice uses resources. Cumulative resource usage might outstrip the benefits of the architecture if you exceed the number of microservices that your DevOps organization, process, and tooling can handle
5. **Don’t forget to keep an eye on the potential latency issue**
   - Making services too granular or requiring too many dependencies on other microservices can introduce latency. Care should be taken when introducing additional microservices

Microservices Dos and Don’ts (cont’d)



6. **Do manage the dependency matrix**

7. **Do realize you’re dealing with an ever changing deployment environment**

   – Questions like the following may have different answers at any given point in time

   » “where are my applications running?!”

   » “where is my data/metrics/logs??!?!”

8. **Do stick with Java if you’re an experienced Java developer**

   – Learning a new language introduces a new layer of complexity as you’re getting familiar with a new development paradigm

Why microservices ?

Efficient teams	Simplified deployment	Right tools for the job	Improved application quality	Scalability
• End to end team ownership of relatively small codebases	• Each service is individually changed, tested, and deployed without affecting other services	• Teams can use best of breed technologies, libraries, languages for the job at hand	• Services can be tested more thoroughly in isolation	• Services can be scaled independently at different rates as needed
➢ Teams can innovate faster and fix bugs more quickly	➢ Time to market is accelerated.	➢ Leads to faster innovation	➢ Better code coverage	➢ Leads to better overall performance at lower cost

IBM CODE

IBM Cloud Sign up

https://ibm.biz/BdzvrK

Workshop Document at

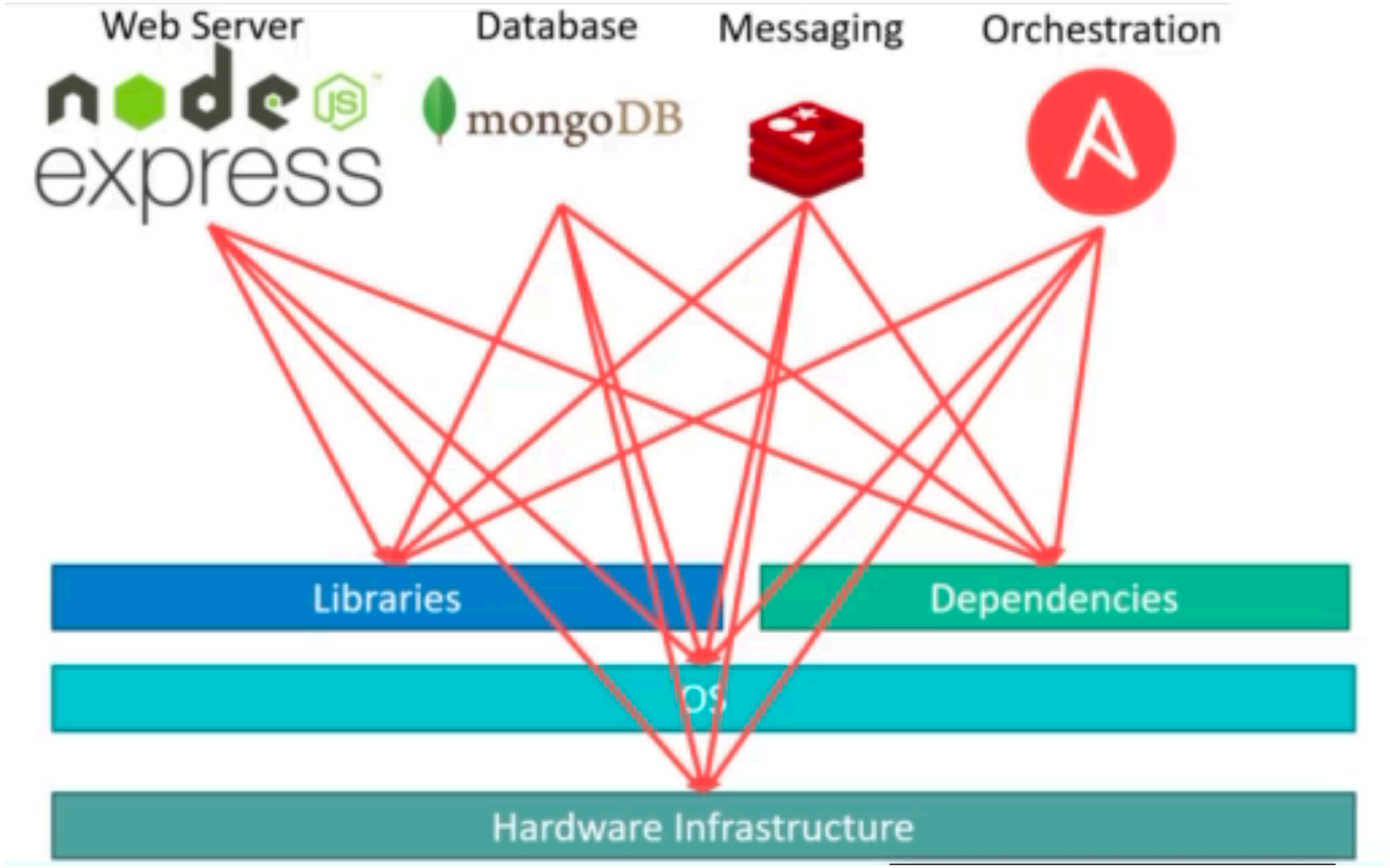
bit.ly/infykube

IBM CODE

Lets understand Container, K8s

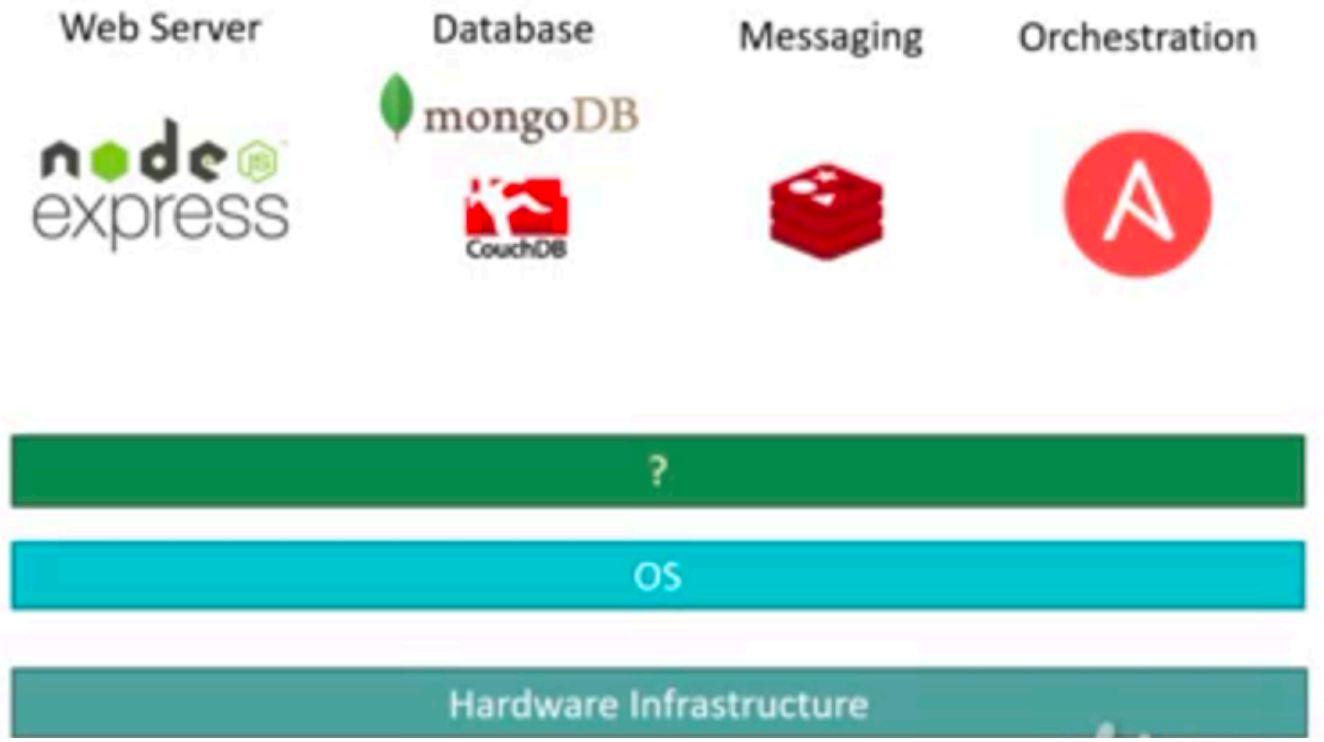
Why do you need containers?

- Compatibility/ Dependency
- Long setup time
- Different Dev/Test/Prod



What can we do to ?

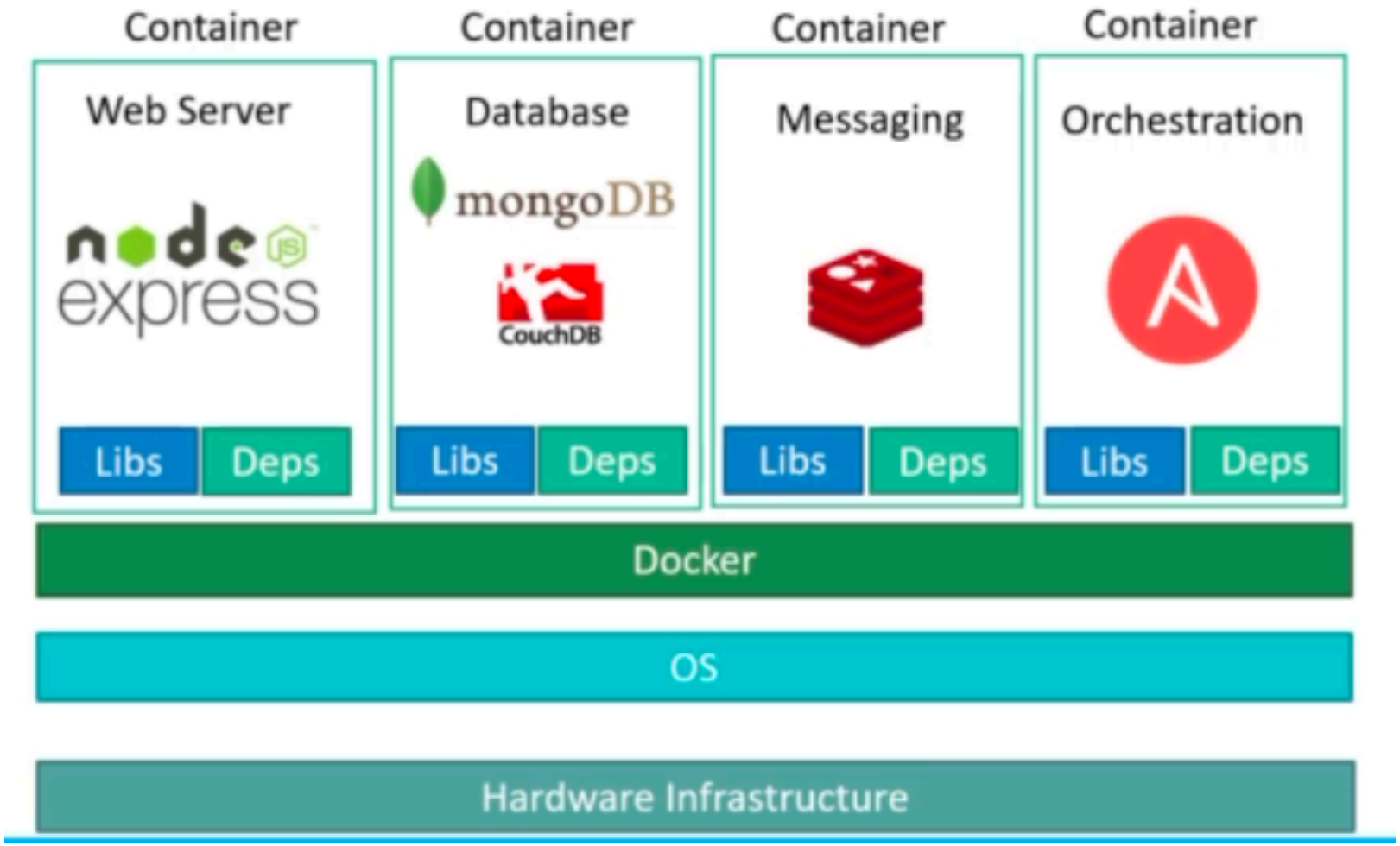
- Remove Compatibility/ Dependency
- Different Dev/Test/Prod



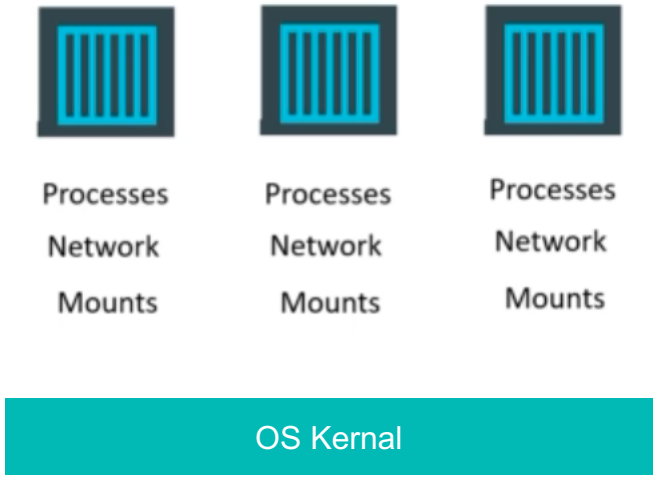
Why do you need containers?

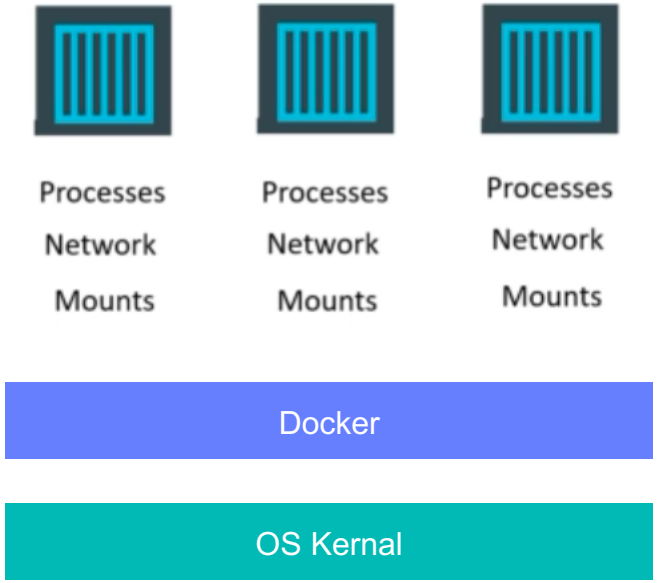
Docker
Docker Configuration

- Need docker installed



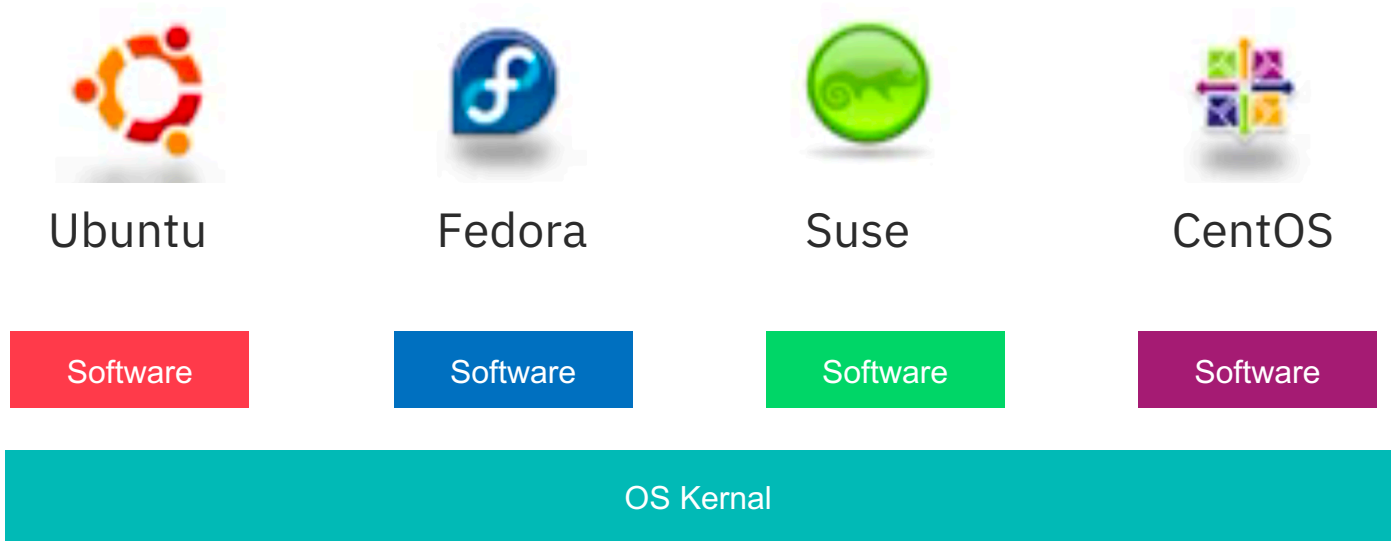
What is container?



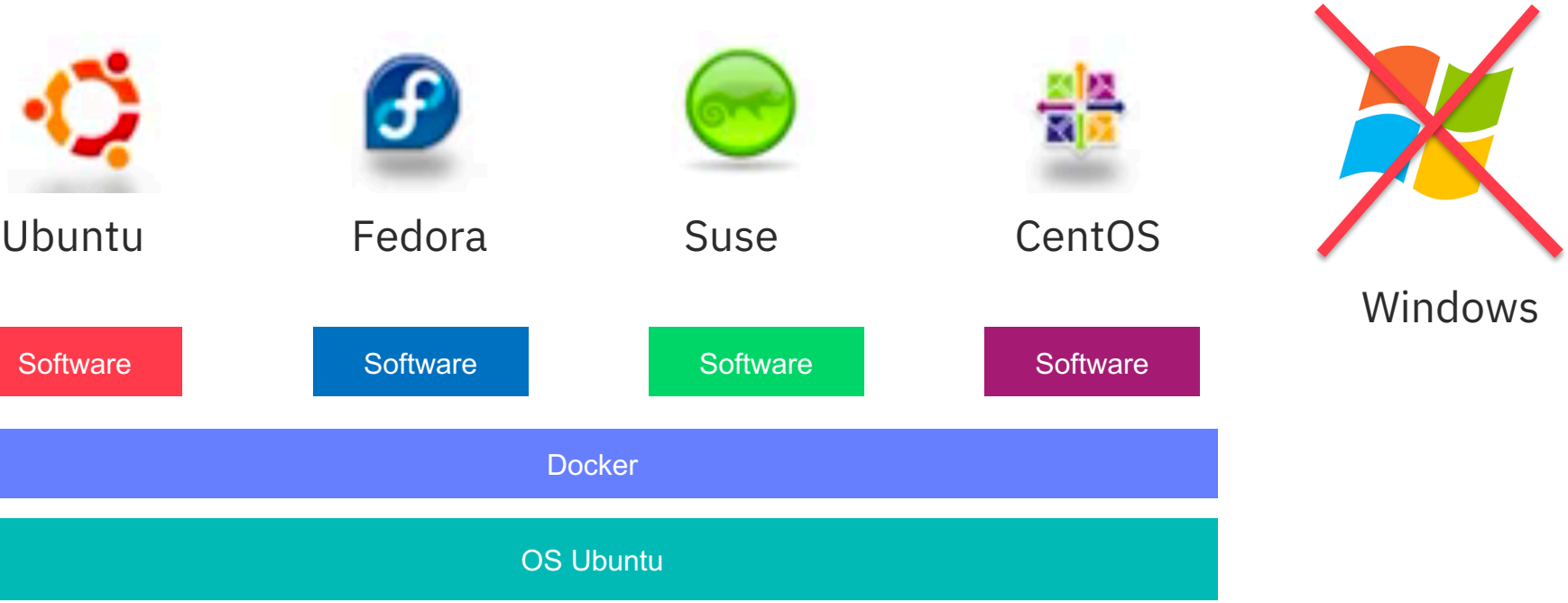


Lxc, lxd, lxcfs

Operating System



Sharing the kernal

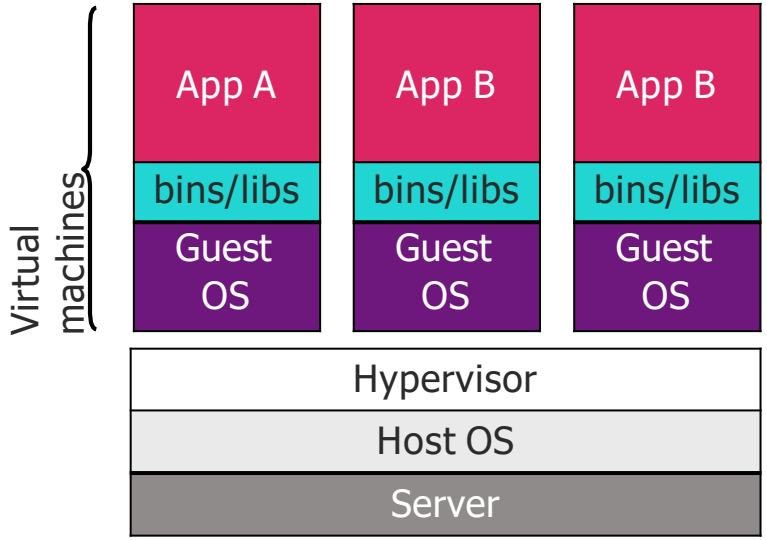


What are containers?

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

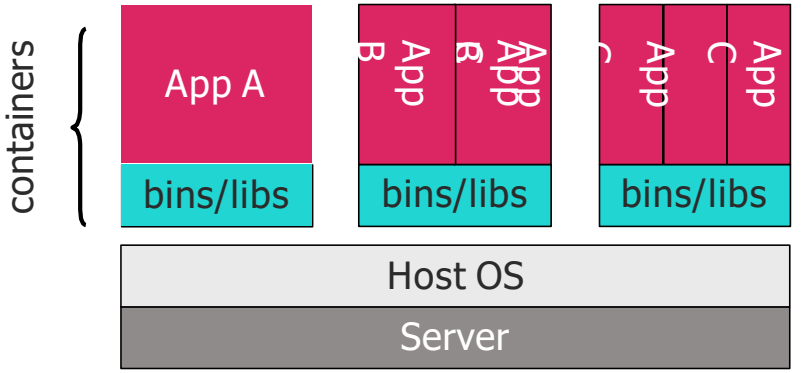
Work by hiding the differences between applications inside the container so that everything outside the container can be standardized

Docker: provides a standard way to create images for Linux Containers



Benefits of using containers

- Can run on many different platforms, e.g. **linux based.**
- Processes share OS resources, but remain segregated
- **Isolate** the different requirements between the applications that run inside the container, and the operations that run outside the container
- **Boot up Fast** : Quick and easy to create, delete, start, stop, download, and share
- Use hardware resources more efficiently than virtual machines, and are more lightweight
- Can be treated as unchangeable

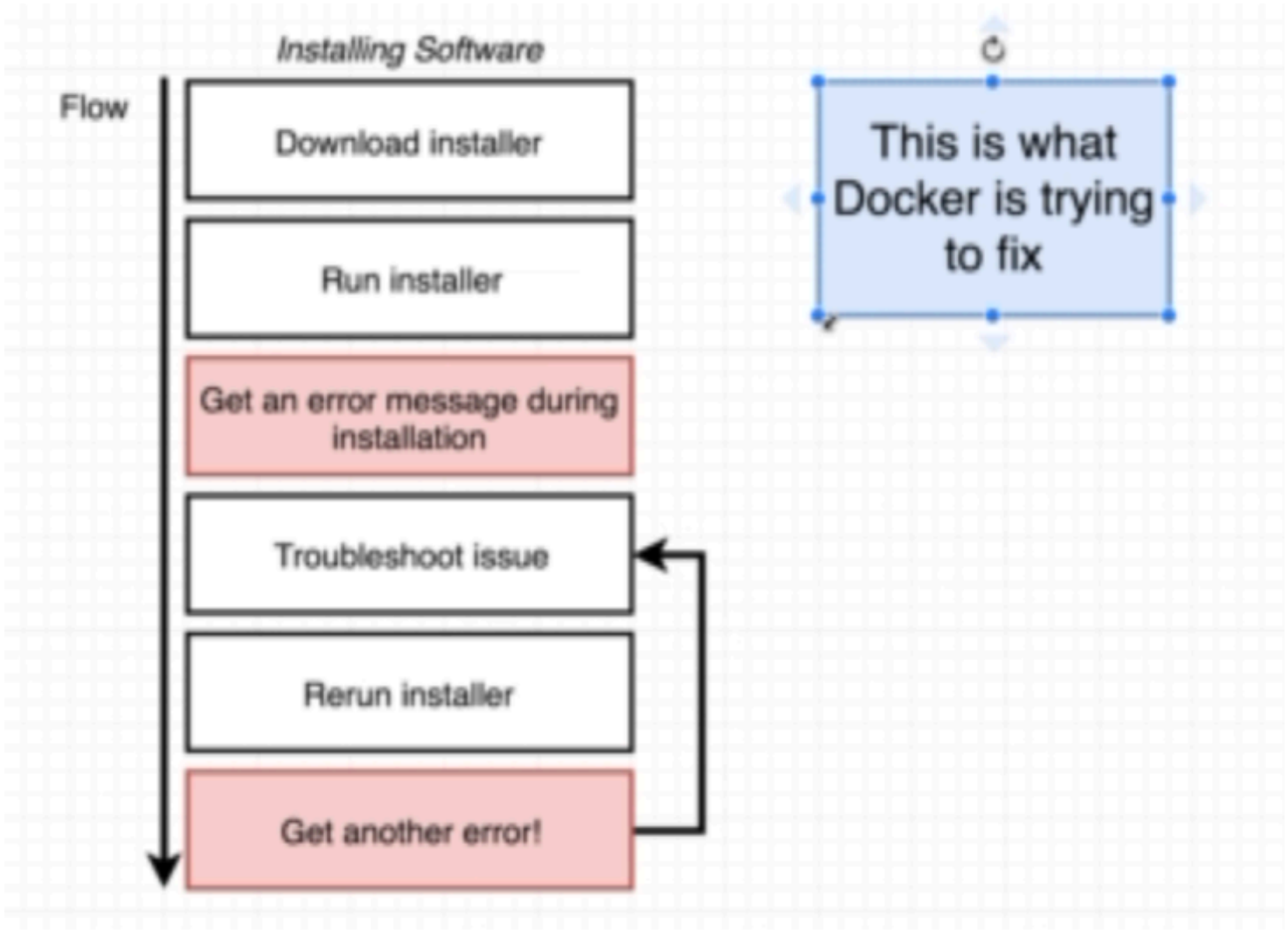


Containers are isolated, but share OS and, where appropriate, bins/libraries

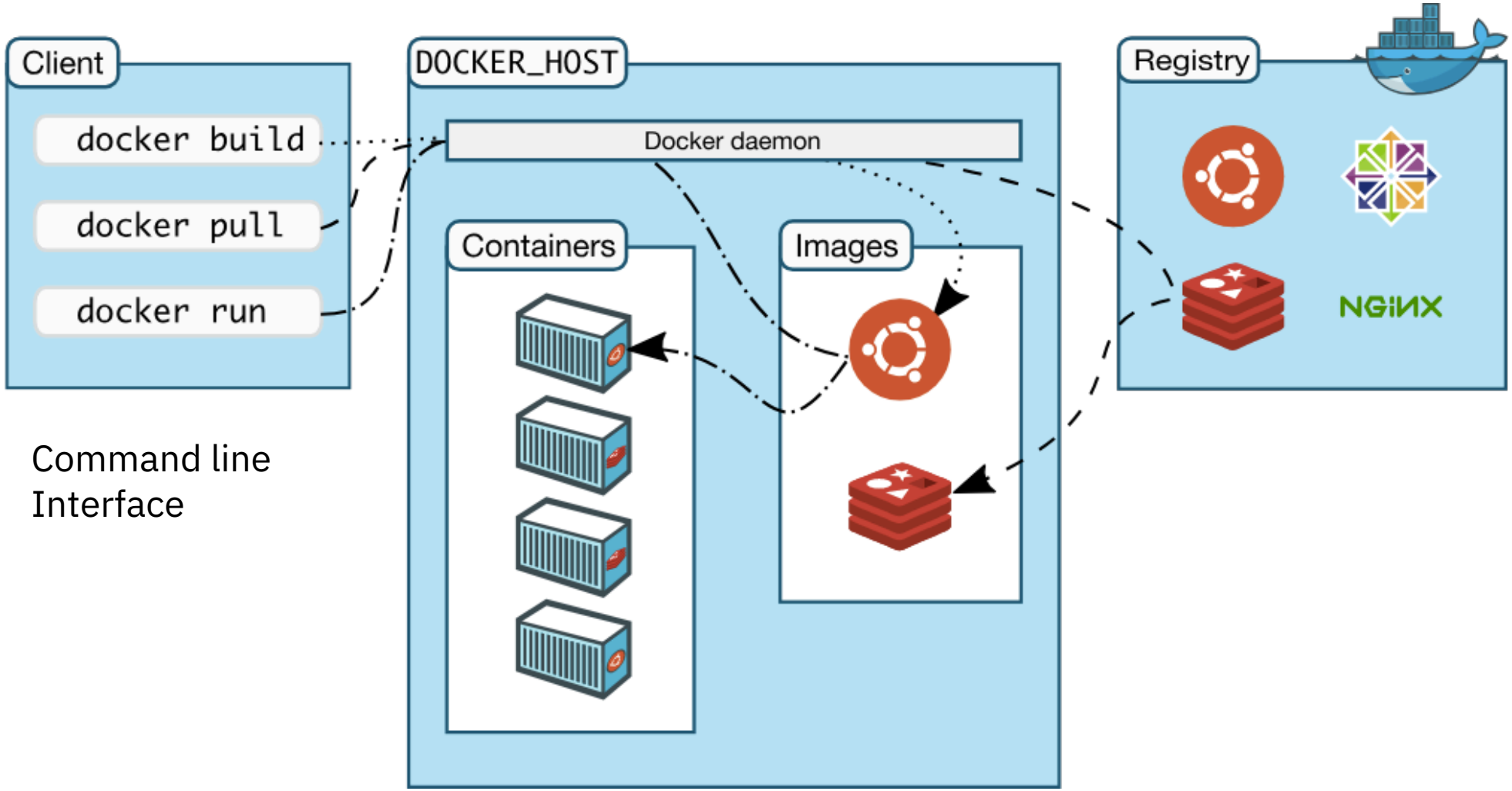
Container : Right Platform for running microservices

- Finer-grained execution environments
- Better isolation
- Faster initialization

Why use Docker?



Docker Flow - Demo

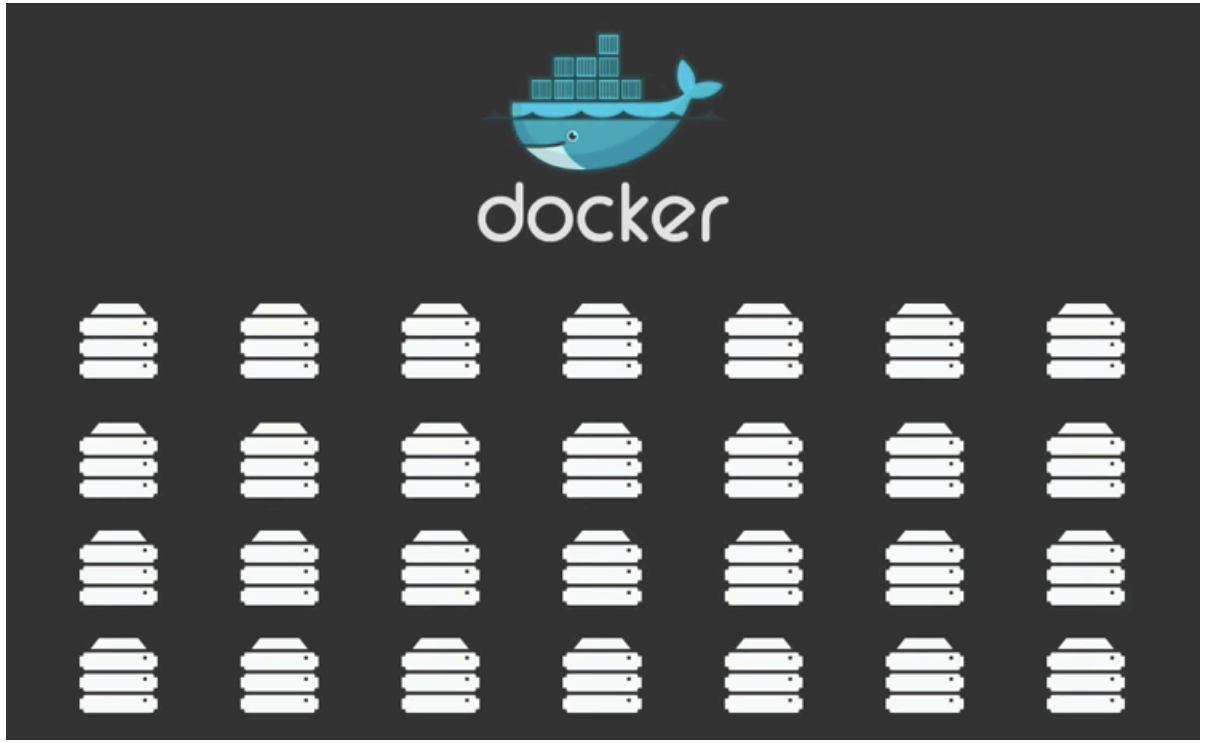


Command line Interface

Docker Engine

Deployed on 3 servers – Simple

Jump from 3 – 40,50 servers



- Scale out
- Keep track
- Where to put your containers
- What container – Where?
- May be containers other than docker as well

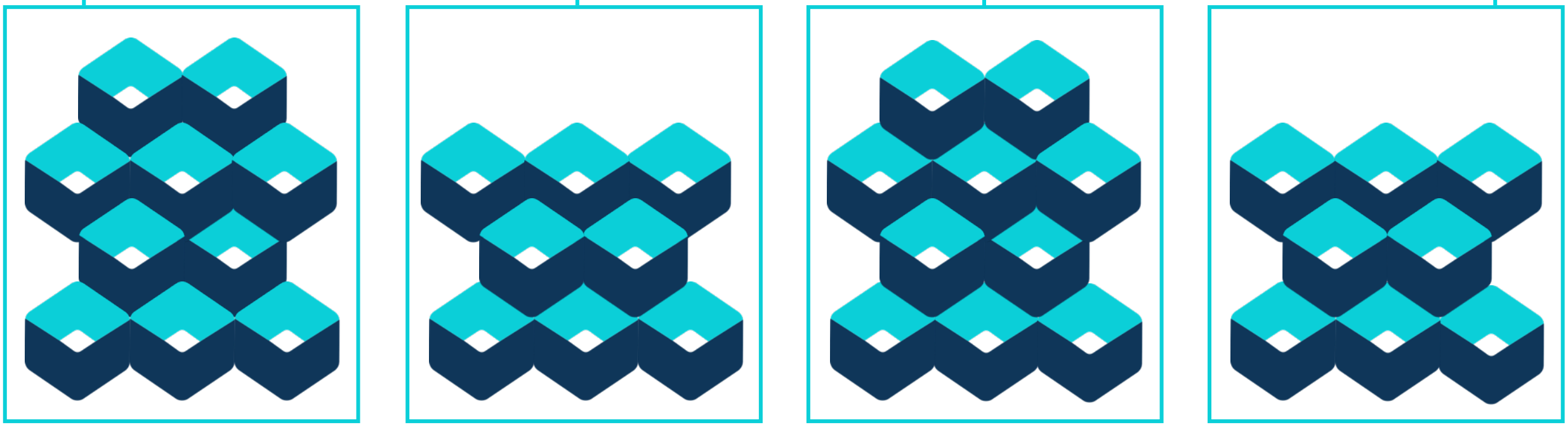
Containers are great but … can lead into lack of control & chaos

IBM CODE

Kubernetes – (Κυβερνήτης - Captain in Greek)

Regain control with Containers and Kubernetes

- Organize and Govern the Container Chaos



IBM CODE

What is K8s?

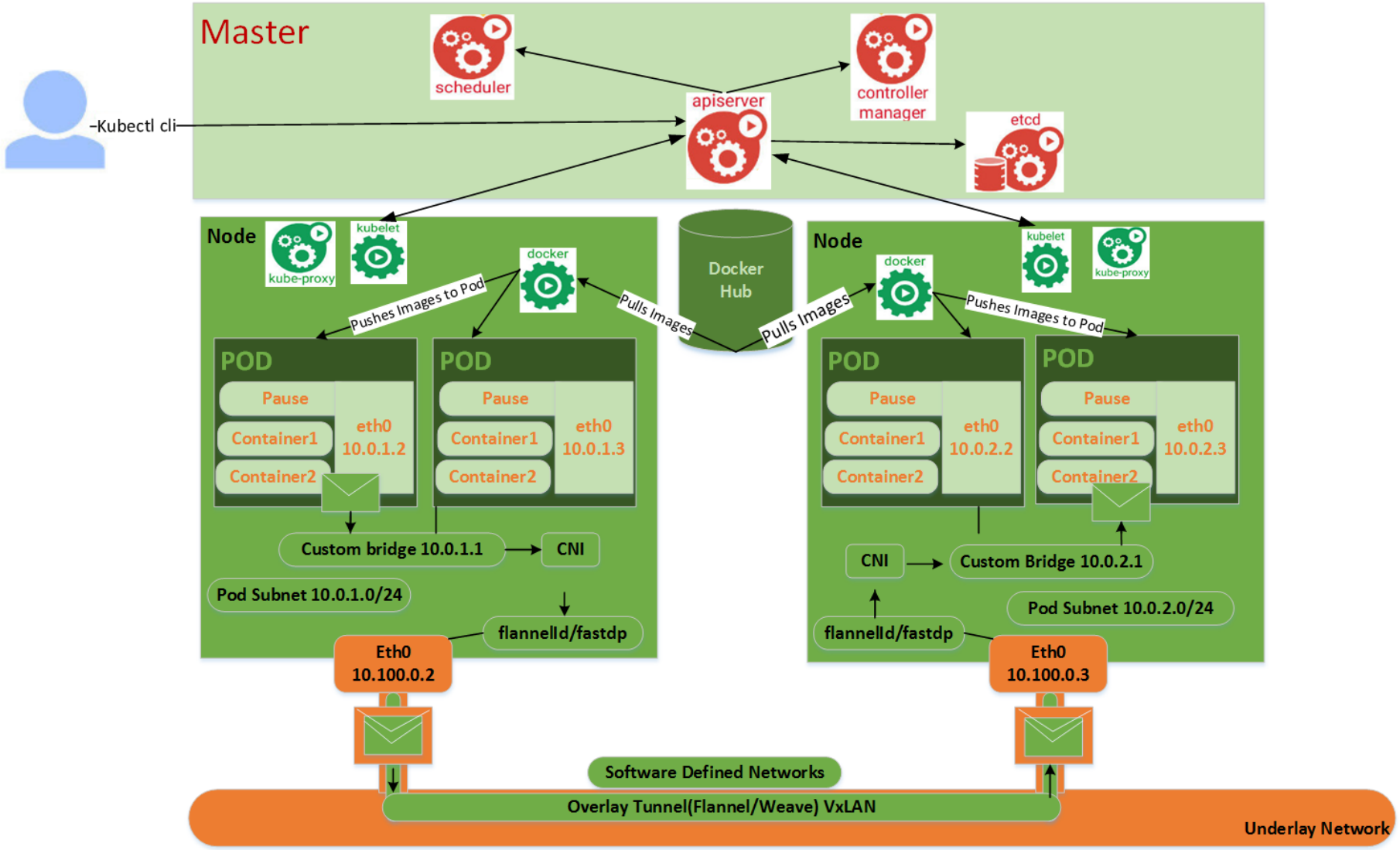
- Based on Google's Borg & Omega - Open Source - Container Orchestrator
- Open Governance

  Cloud Native Compute Foundation
- Adoption by Enterprise

  RedHat, Microsoft, IBM and Amazon

- Help to automate DevOps – Deployment, scaling and management of containerized apps
- Helps organize container in logical units(pods, nodes)
- Helps in resource monitoring and logging

Kubernetes Architecture



IBM Cloud Kubernetes Service

A **managed Kubernetes service** providing an intuitive user experience with simplified cluster lifecycle management.

Built-in **security and isolation** to enable rapid delivery of apps, while leveraging IBM Cloud Services including Weather data, IoT, Analytics, or **AI capabilities with Watson**. Available in six IBM regions WW, including **25+ datacenters**.

HIPAA Health Insurance Portability & Accountability Act

AICPA SOC

ISAE 3402

https://www.ibm.com/cloud/container-service

Kubernetes Certified Service Provider
Kubernetes Technology Partner
Kubernetes Service Providers

certified kubernetes 1.10

certified kubernetes 1.11

certified kubernetes 1.12

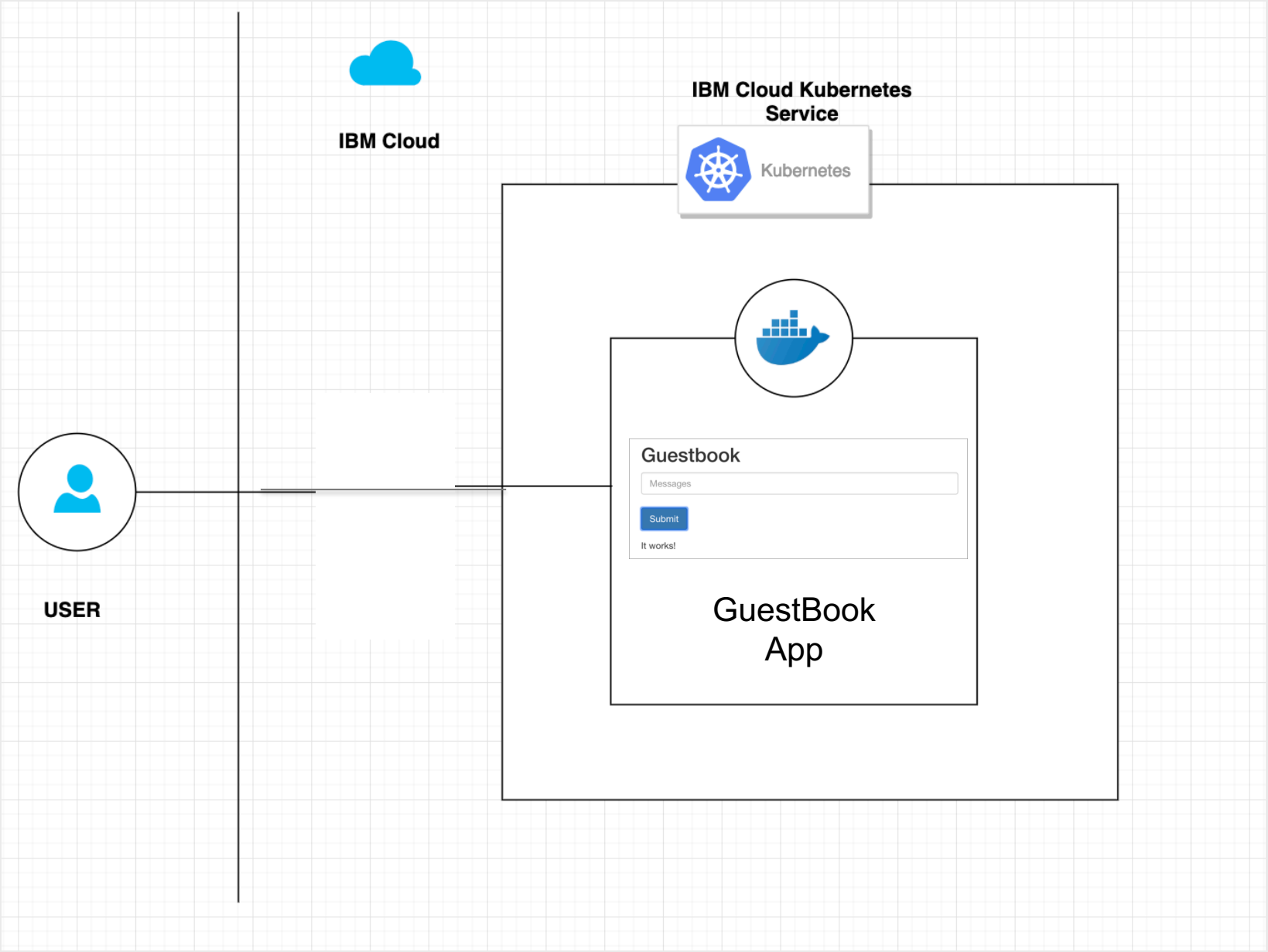
certified kubernetes 1.13

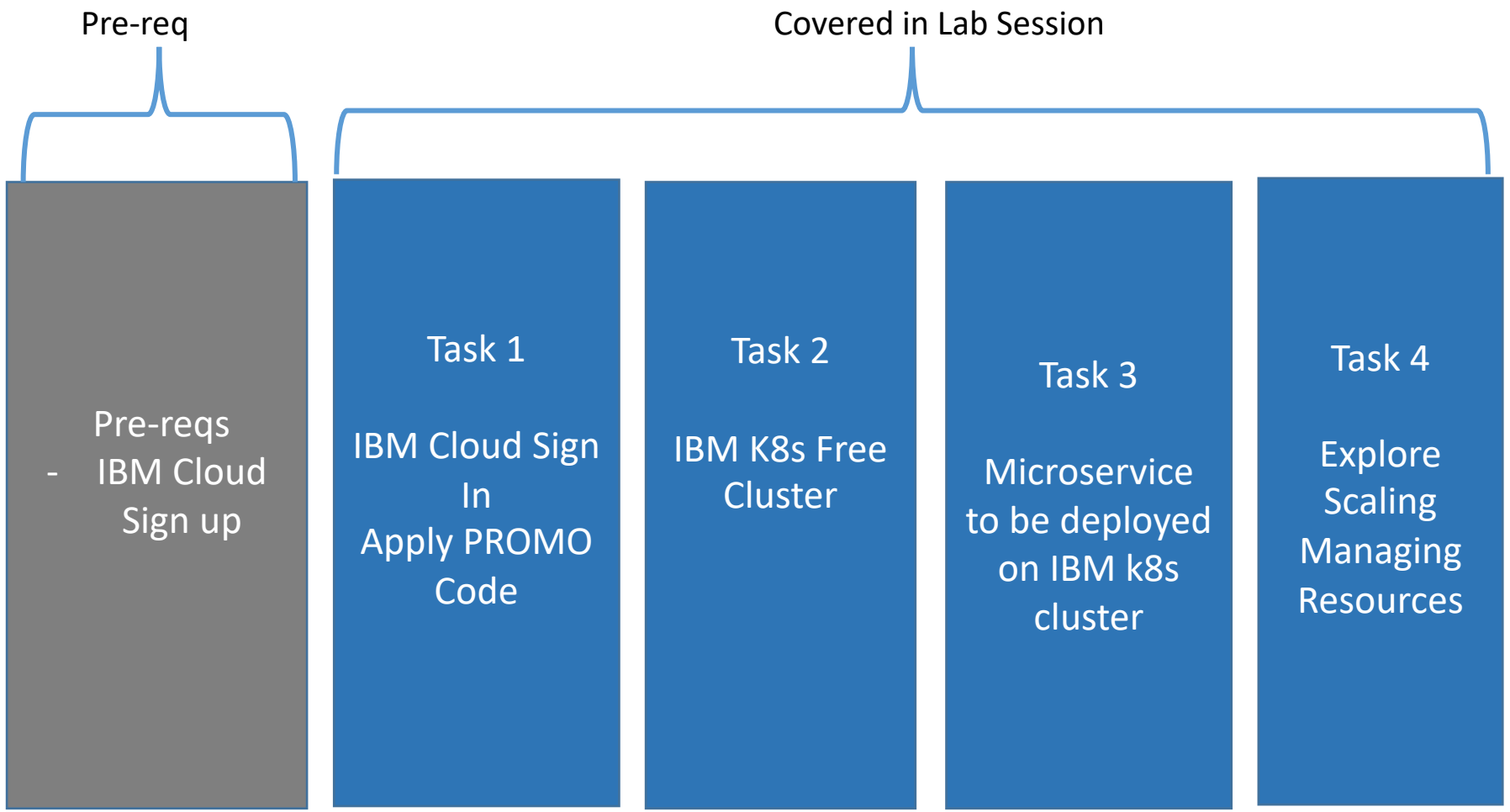
K8s - Key Features

Hands On Deploy microservice on IBM Cloud Kubernetes Cluster

Architectural Flow:
IKS (k8s Managed Cluster) on IBM Cloud
Use Client (Kubectl) to Deploy App

Technology Stack (IBM Cloud):
Docker Container
Kubernetes
IBM Cloud - IKS




Pre-req
Covered in Lab Session
Pre-reqs
- IBM Cloud Sign up
Task 1
IBM Cloud Sign In
Apply PROMO Code
Task 2
IBM K8s Free Cluster
Task 3
Microservice to be deployed on IBM k8s cluster
Task 4
Explore Scaling Managing Resources

Pre-requisite

IBM Cloud account Sign up

IBM Cloud Access

You should have access the following URLs

- https://cloud.ibm.com
- https://github.com

IBM Cloud Sign up

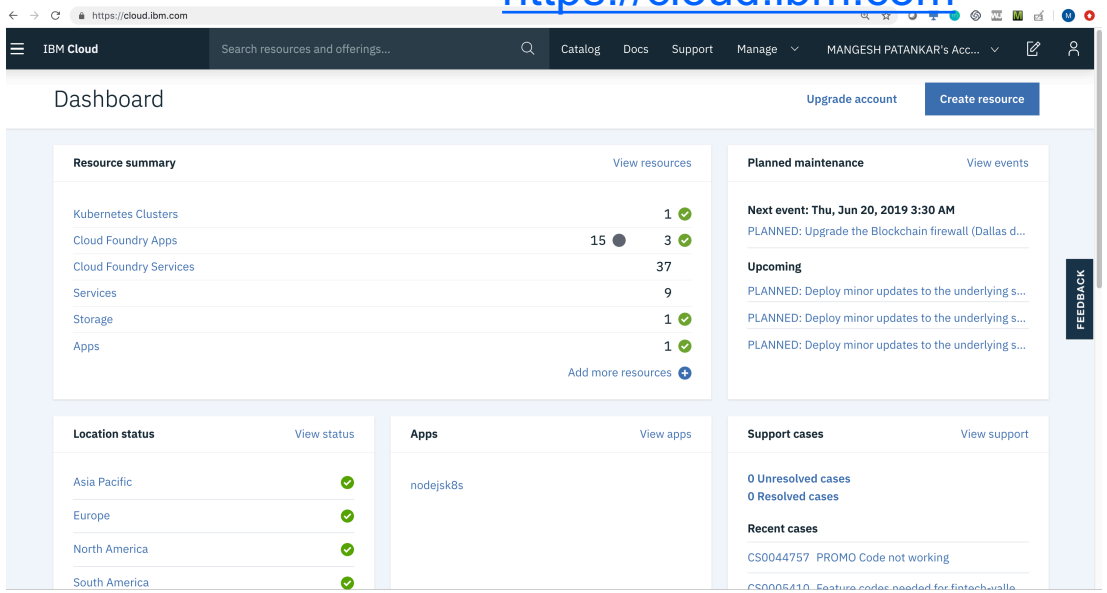
https://ibm.biz/BdzvrK

Workshop Document at

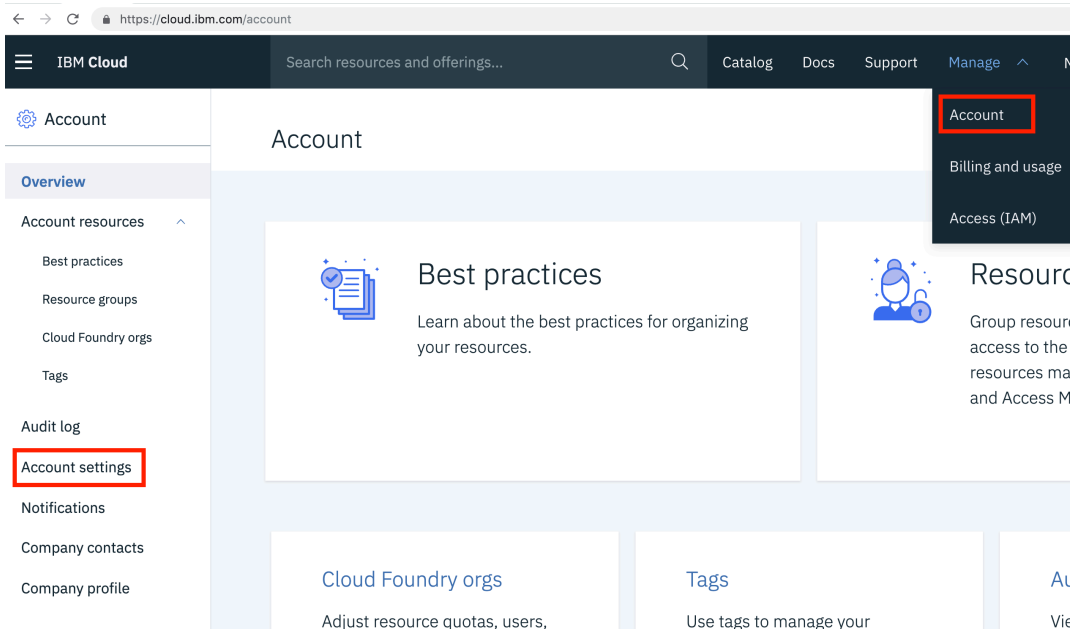
bit.ly/wiprokube

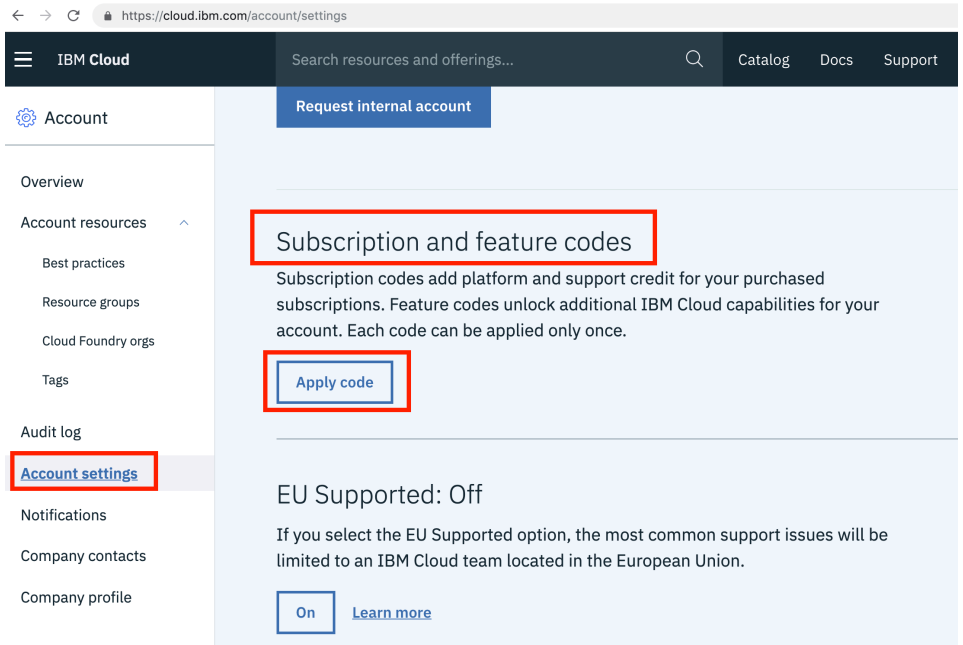
Task 1

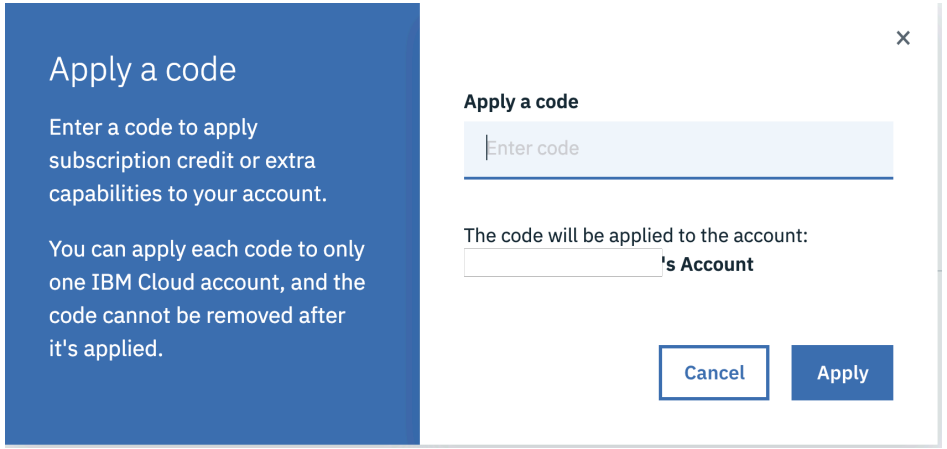
Login to the IBM cloud account
https://cloud.ibm.com



Apply PROMO Code

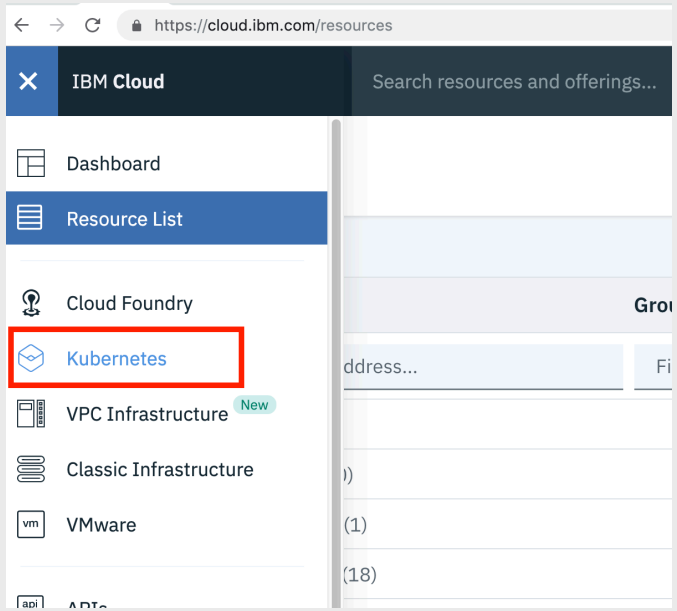


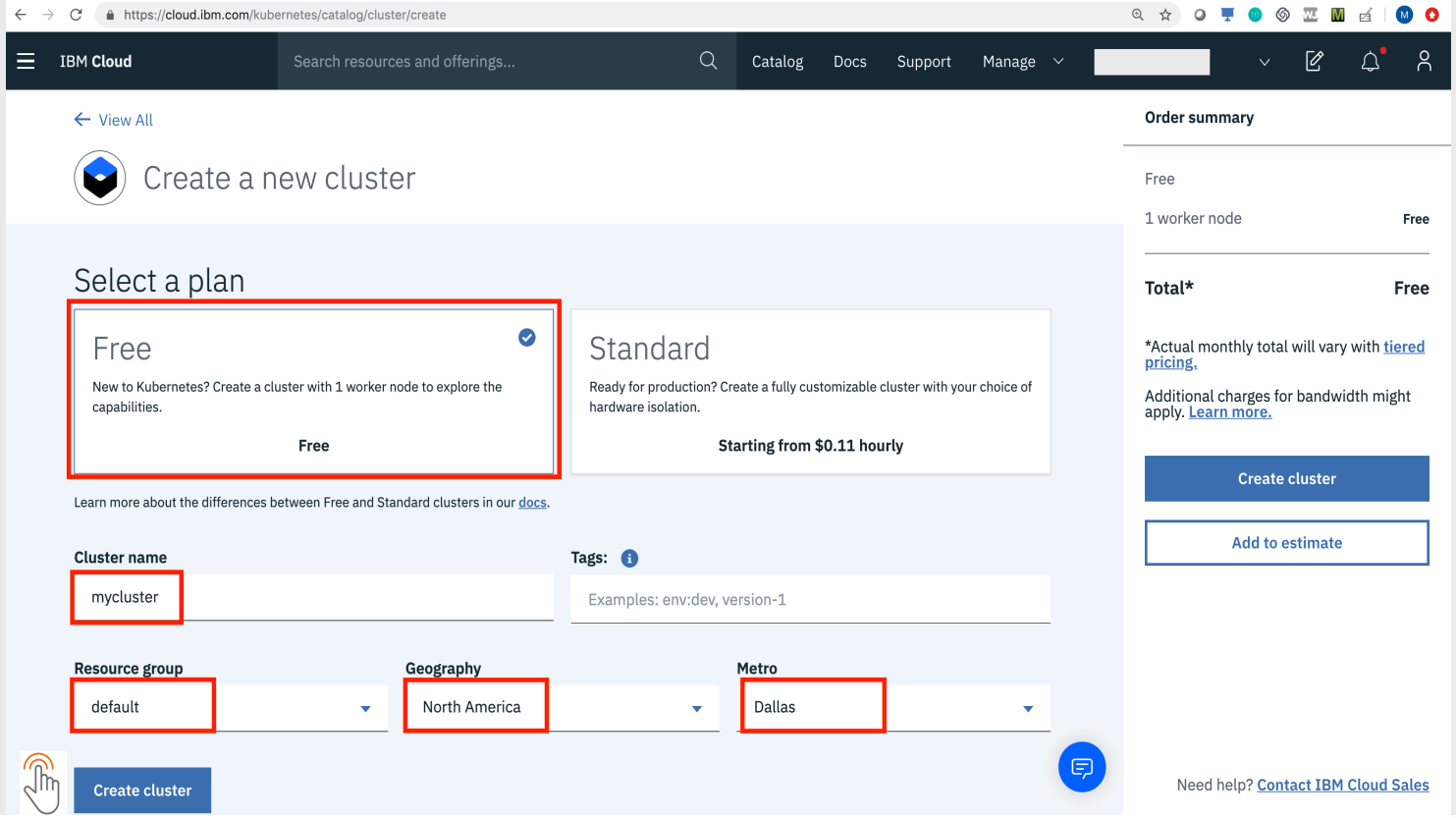


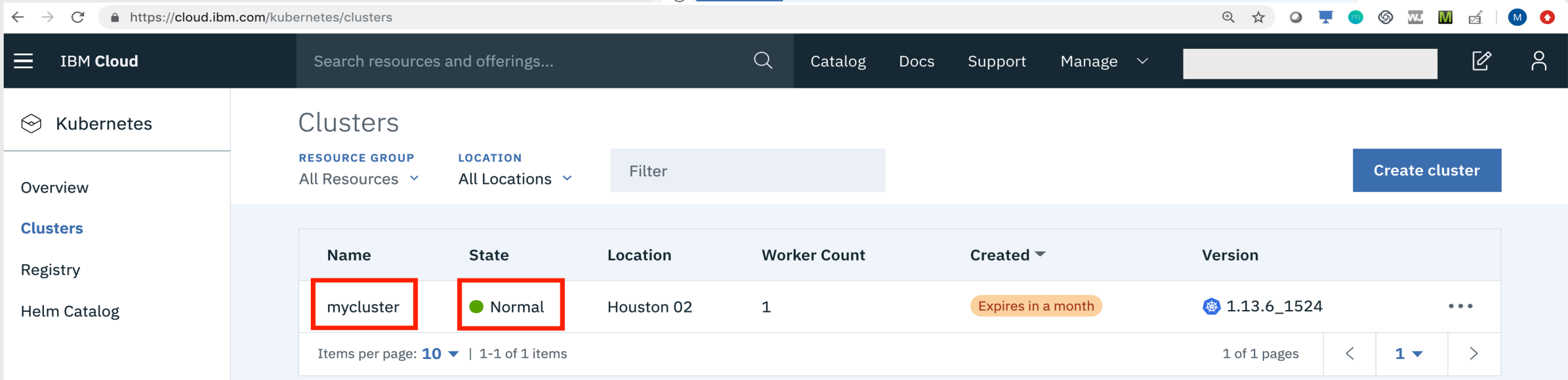


Task 2

Create K8s Free Cluster







You have your kubernetes cluster up and running

Task 3 Deploy a microservice on K8s Cluster created

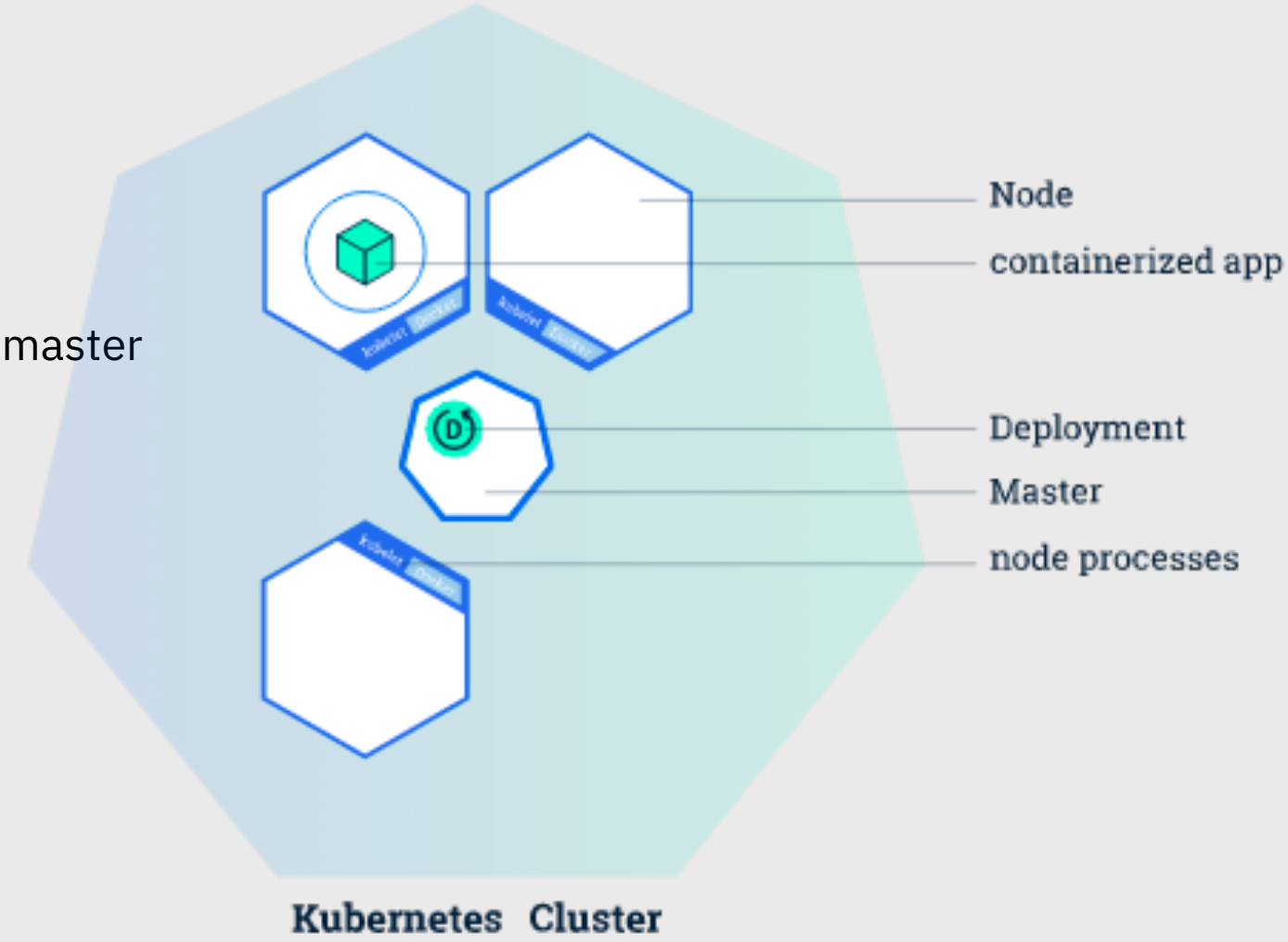
- Client to Co-ordinate with Cluster - Kubectl

(For this lab we will use Web Terminal)

- Use Kubectl commands to create k8s Objects
- Or YAML/JSON – Language to communicate to master
- Create a Deployment using command

  - kubectl create deployment

- Check your application is deployed



Task 4 Scaling, Clean up

Scale the instances to required number, through client.

- *kubectl scale --replicas=10 deployment guestbook*

$ kubectl delete deployment guestbook

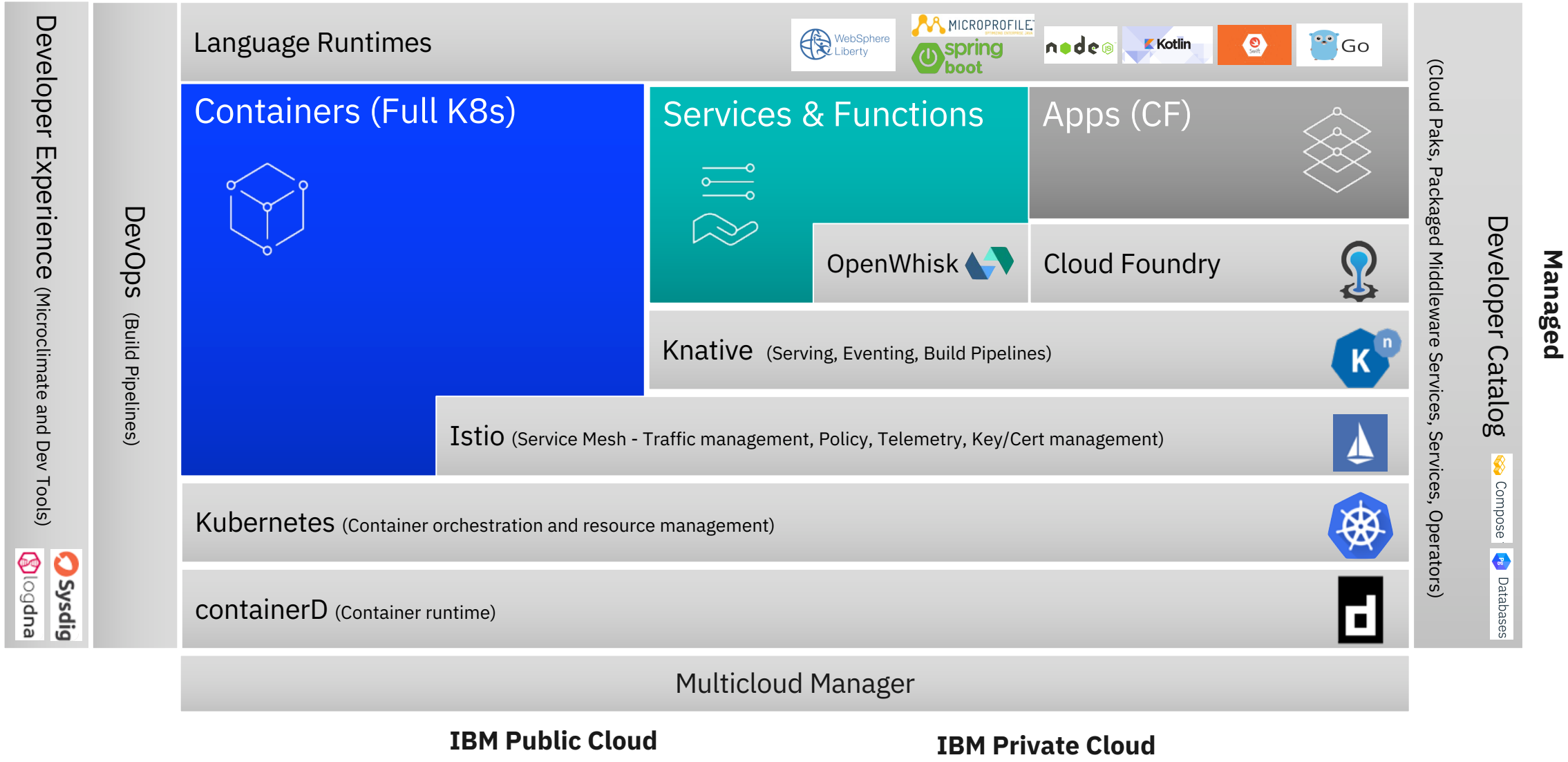
Remove the deployment

- *kubectl delete deployment guestbook*

Remove the service

- *kubectl delete service guestbook*

Industry Cloud Native Platform



IBM CODE

Educational Courses - Certify your Knowledge

- Docker essentials: Extend your apps with containers
  - https://developer.ibm.com/courses/all/docker-essentials-extend-your-apps-with-containers/
- Get started with Kubernetes and IBM Cloud Kubernetes Service
  - https://developer.ibm.com/courses/all/get-started-kubernetes-ibm-cloud-container-service/
- Get started with Istio and IBM Cloud Kubernetes Service
  - https://developer.ibm.com/courses/all/get-started-istio-ibm-cloud-container-service/
- Use the “Develop and test microservices with Kubernetes and Helm” toolchain
  - https://www.ibm.com/cloud/garage/tutorials/use-develop-test-microservices-with-kubernetes-and-helm-toolchain

- Using Istio across public and private clusters - https://developer.ibm.com/patterns/istio-for-multi-clusters-across-iks-and-icp/

- https://cognitiveclass.ai/learn/containers-k8s-and-istio-on-ibm-cloud/

Stay Connected and continue coding !

IBM CODER

IBM Coder Platform
https://ibmcoders.influitive.com/

Code & instructions available here
https://github.com/IBMDevConnect

IBM DEV ELO PER

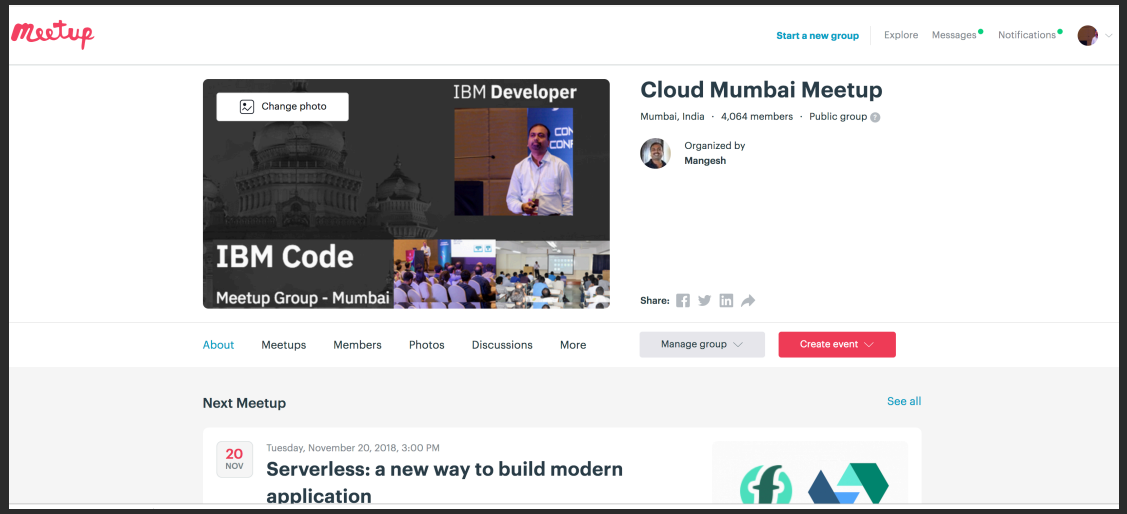
Check out the code patterns
https://developer.ibm.com/patterns/

Join our Slack team and stay in touch with the experts
https://ibmdevconnect.slack.com
Send in your request to -
http://ibm.biz/slackrequest

Call for Code 2019
https://callforcode.org/

Meetup

Join our Meetup groups
Mumbai :
https://www.meetup.com/Cloud-Mumbai-Meetup/
Hyderabad:
https://www.meetup.com/Hyderabad-Cognitive-with-Cloud
Bangalore :
https://www.meetup.com/IBMDevConnect-Bangalore
Chennai:
https://www.meetup.com/Chennai-CodeWeekend-Meetup/



@MangeshPatank

/mdpatankar

in/mangesh-patankar-1961a019

mapatank@in.ibm.com

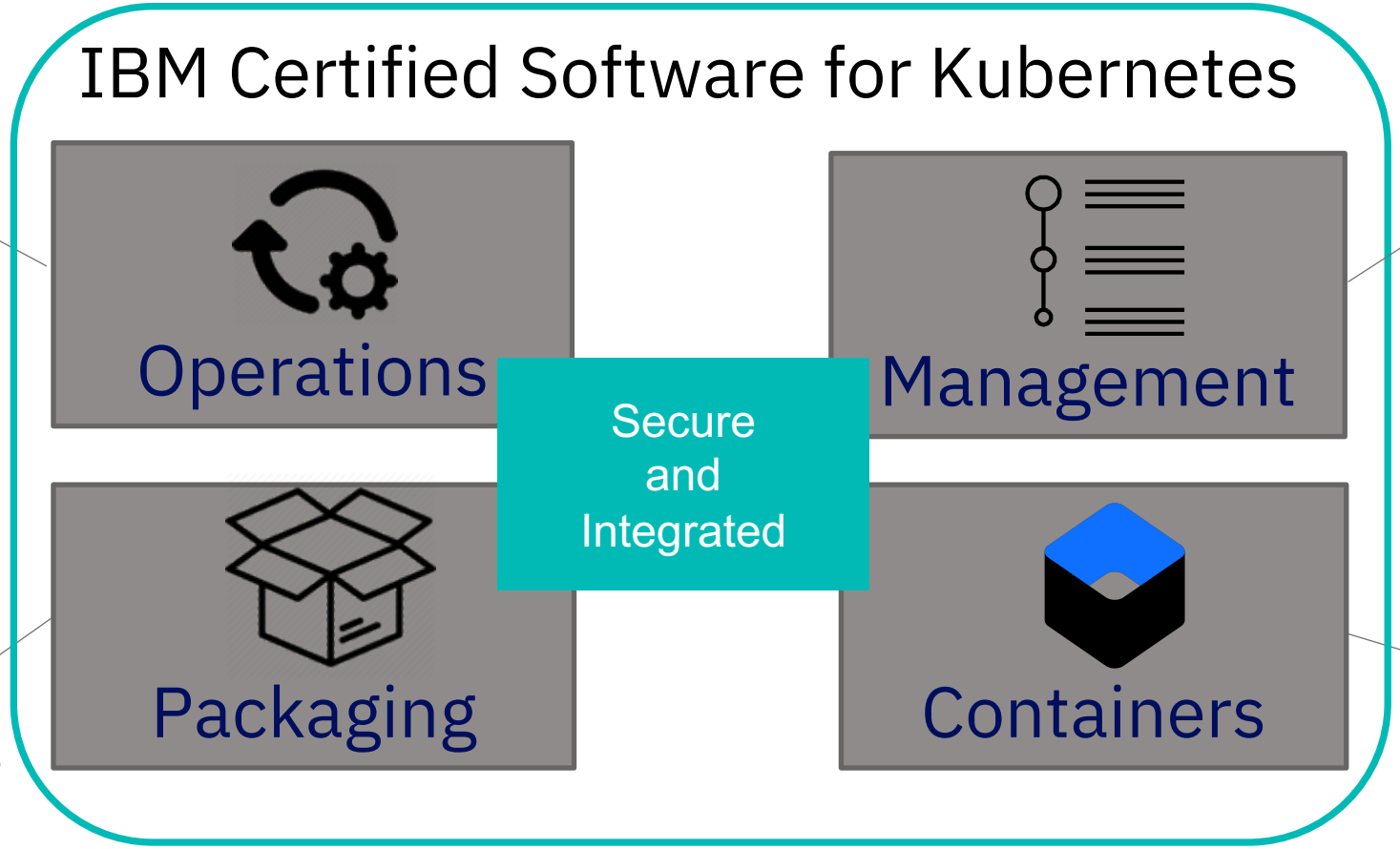
IBM CODE

Backup Slides

App Modernization is more than wrapping your app in a Docker Image

Upgrade & Rollback
consistent across all IBM Software built for Kubernetes

Enterprise Ready and Simple to Deploy
-Orchestrated by the product experts
-Integrated catalog experience
-Open standards packaging
-Secured by ICP IAM



Pre-integrated:
-Logging (Debug)
-Monitoring (Alerting)
-Usage Metering
-License Management

-**Scanned for Vulnerabilities**
-**Extendable** to Redhat Certified with RHEL base image

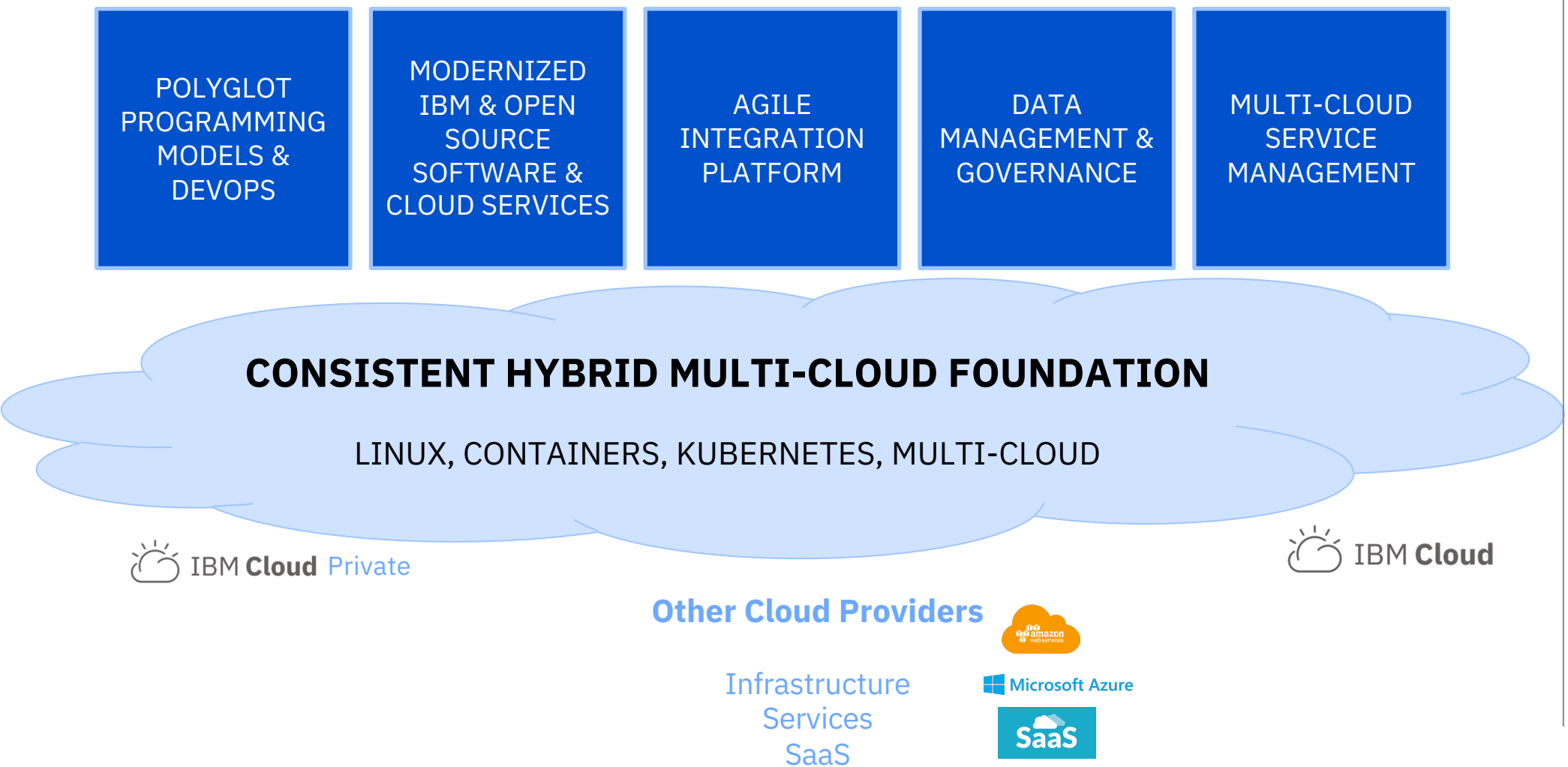
The entire package is certified to support IBM Cloud Private, not the individual pieces

IBM Cloud Private
The Universal Platform for IBM Middleware and Containers

IBM CODE

IBM’s Approach to Application Modernization

An Integrated Enterprise Hybrid Cloud Platform



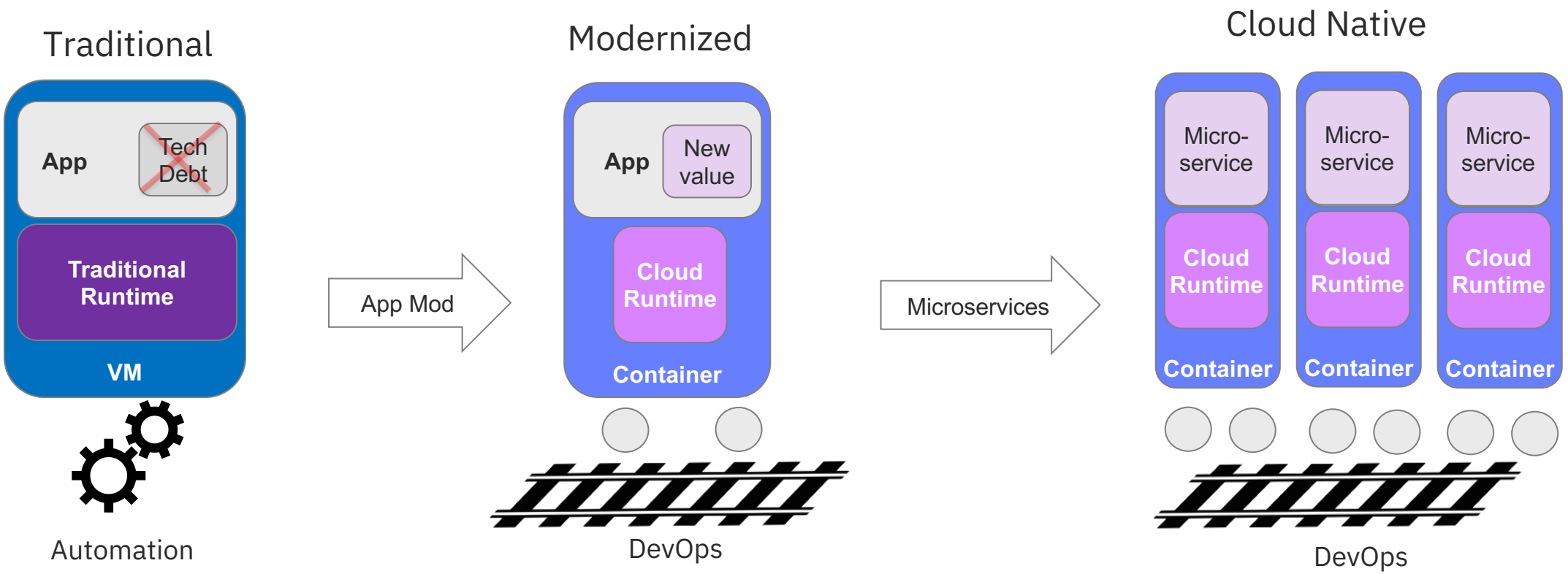
Migration & Transformation Tools

Methods & Reference Architectures

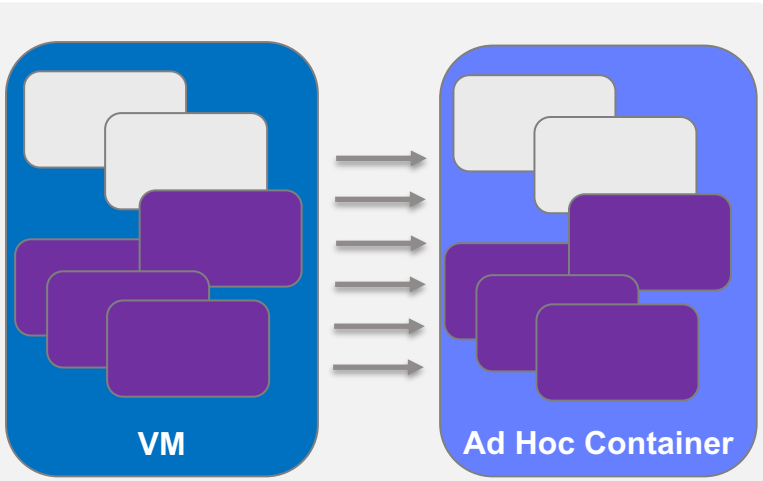
Innovation Garages

Professional Application & Managed Services

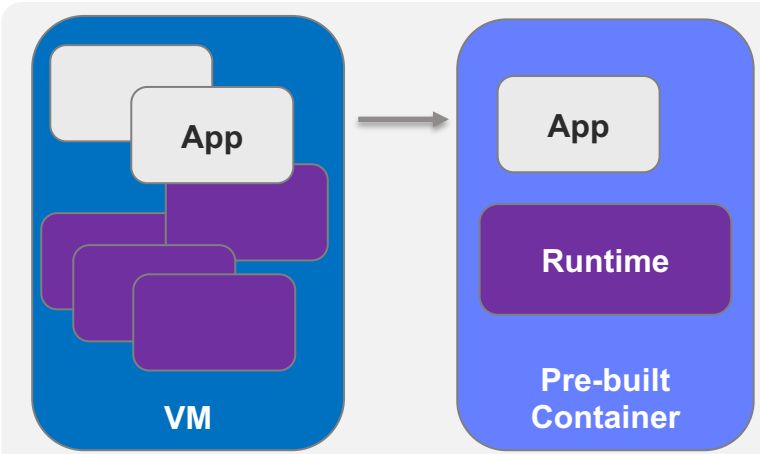
Traditional Java Application Modernization



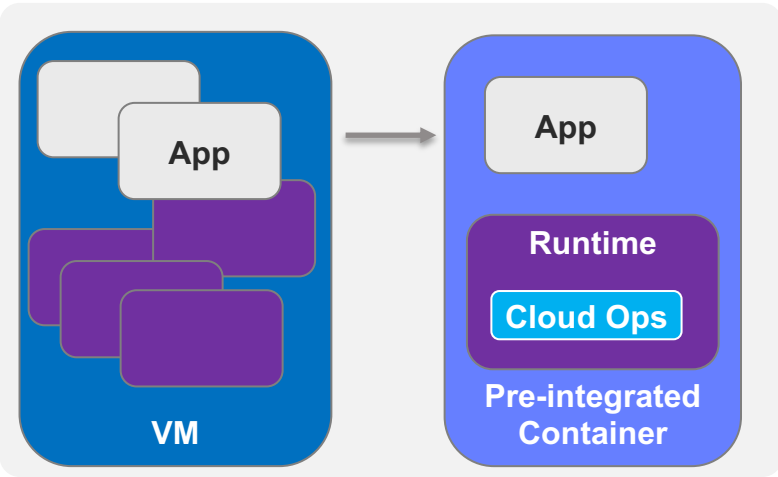
Containerization Approaches



“Lift & Shift” Containerization - DIY

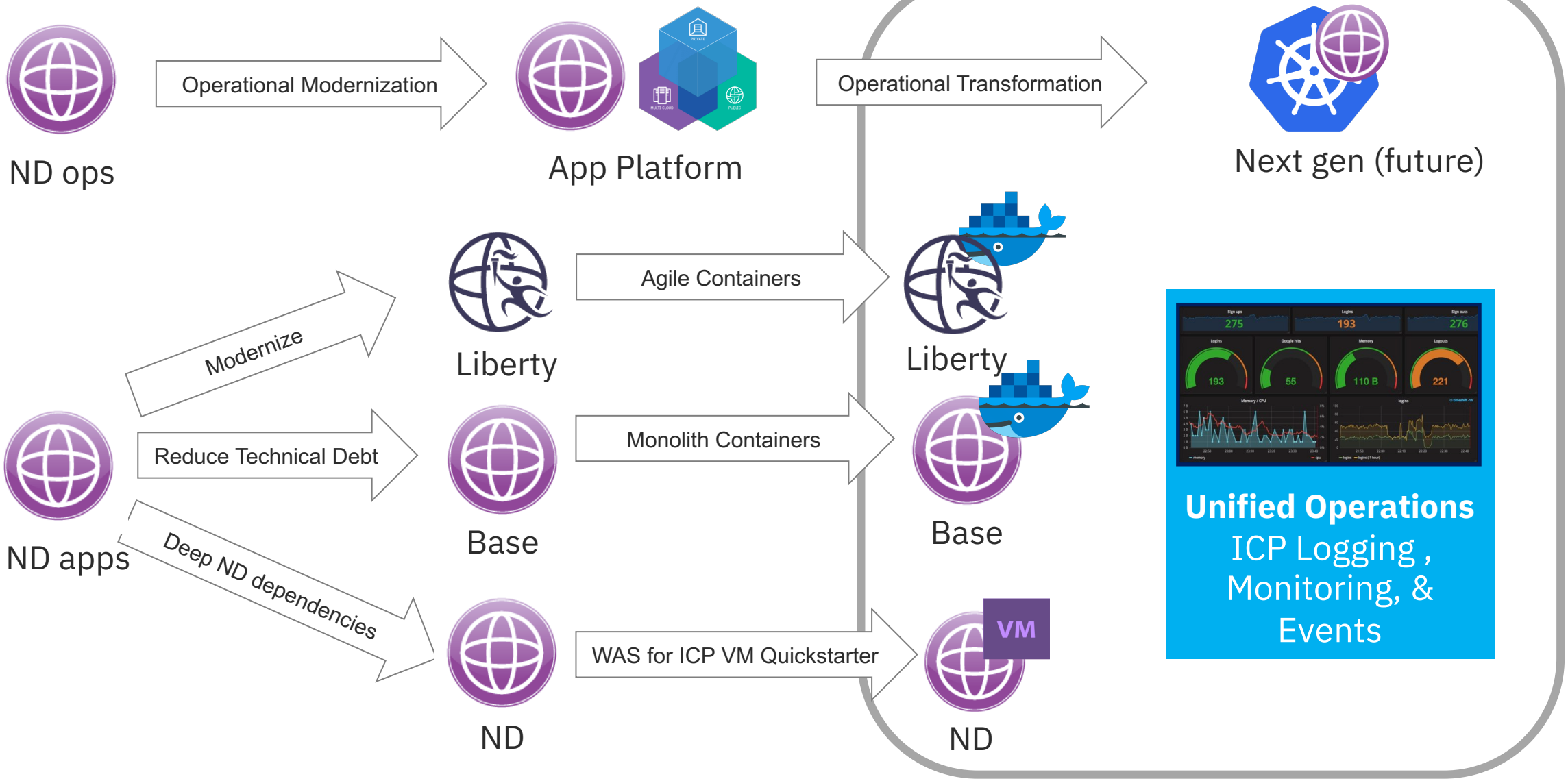


Vendor-provided containers



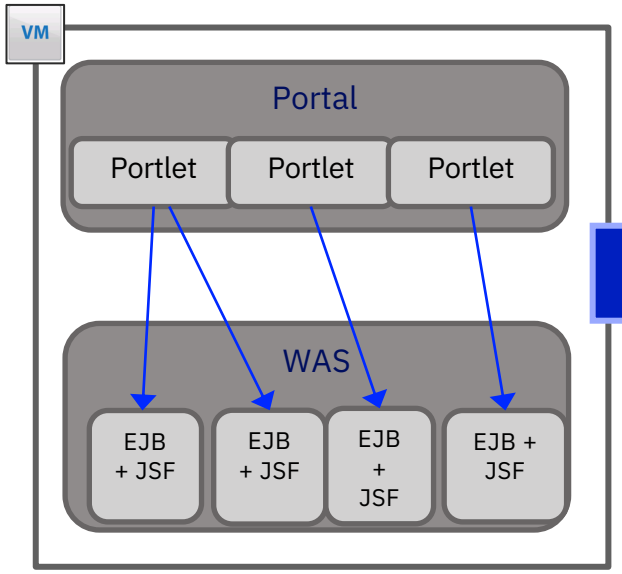
Built-in opinionated Cloud Ops

AppMod for WebSphere



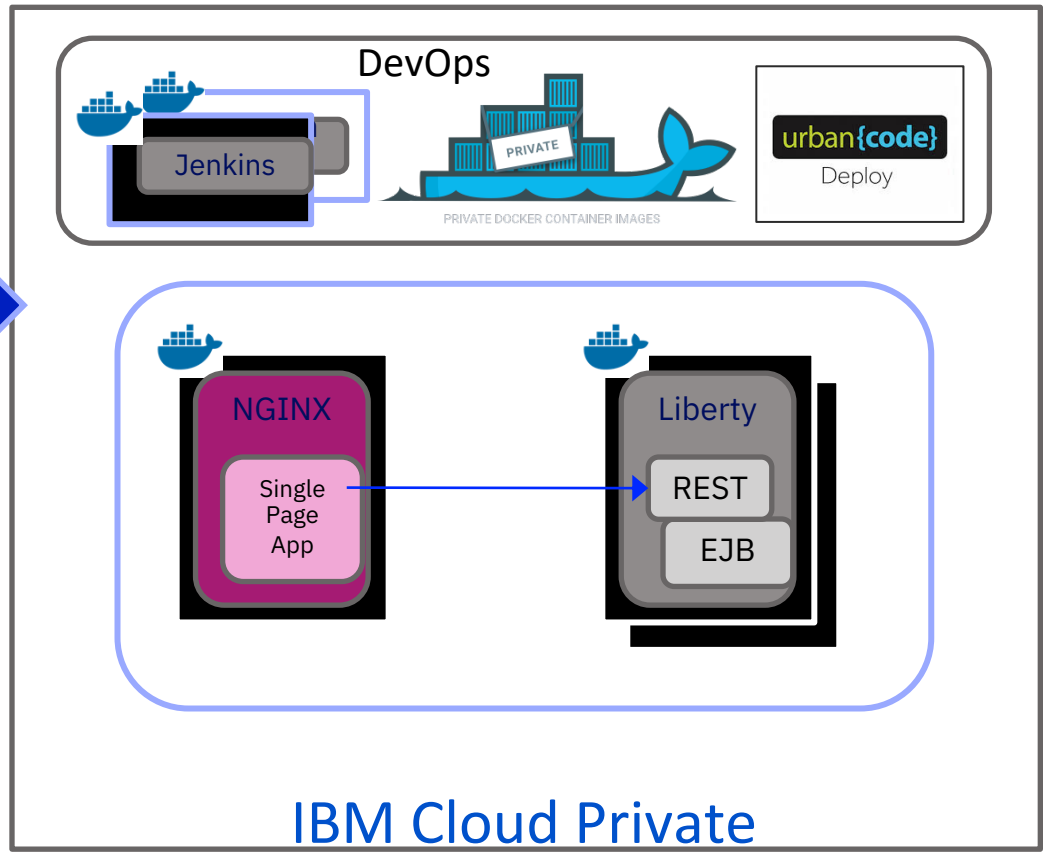
Modernize Mission-Critical Applications

Large US Insurance provider needed to modernize mission critical application for managing insurance claims and processing



Modernization Journey

- Modernize EJBs to run on Liberty and with REST interface
- Create new frontend
- Automate CI/CD with Jenkins and IBM UrbanCode



BENEFITS:

#1 - Modern, supported runtime with zero migration

#2 - Automated provisioning of test environments and automated testing

#3 - Reduced business risk by reusing EJB layer

Customer implementation led by IBM Cloud Garage

Microservice Solution Overview

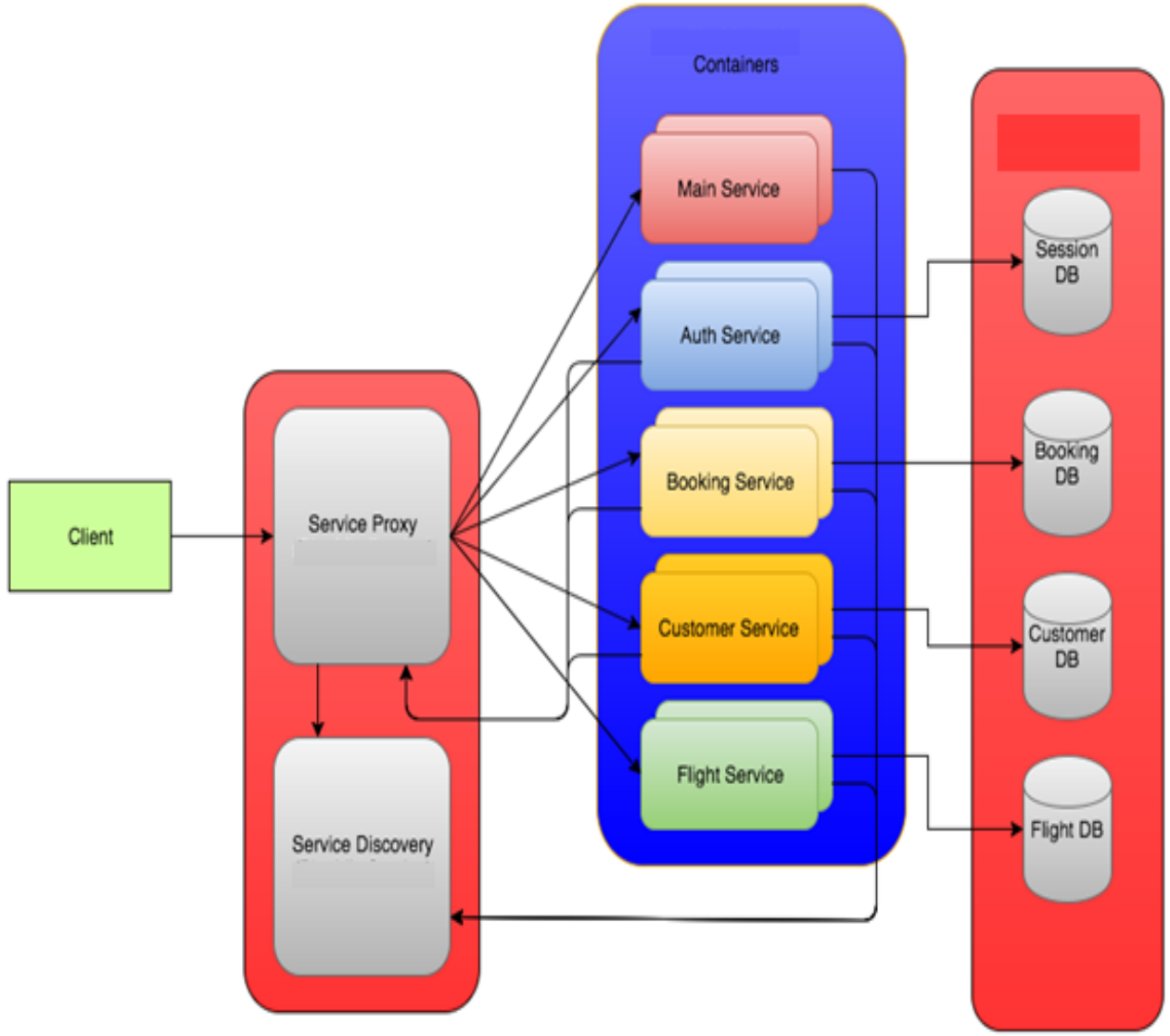
Here is a hypothetical Airline solution broken up into the following standalone services.

•Main app - presentation layer
•Auth Service - manages sessions (login, logout, validation)
•Customer Service - manages customer Info (get, update, validate password)
•Flight Service - manages flight data
•Booking Service - manages booking data
•Support Service - a simple dummy websocket chatroom (ma add a cognitive service here)

•Multiple instances of each service can be created.
•A service proxy is setup in front of the services to route/load balance them.

- The Auth, Booking, and Customer services also need to know how to reach the proxy because they call other services.
- Service Discovery manages the micro-services.

•The DBs can be separate instances (recommended) or one instance.



IBM CODE

Technologies for microservice transformation

- Containers (Docker)
- Container orchestration (Kubernetes)
- IBM Cloud Private
- Transformation Advisor
- 12-Factor Best Practices
- CI/CD tools (e.g Jenkins)

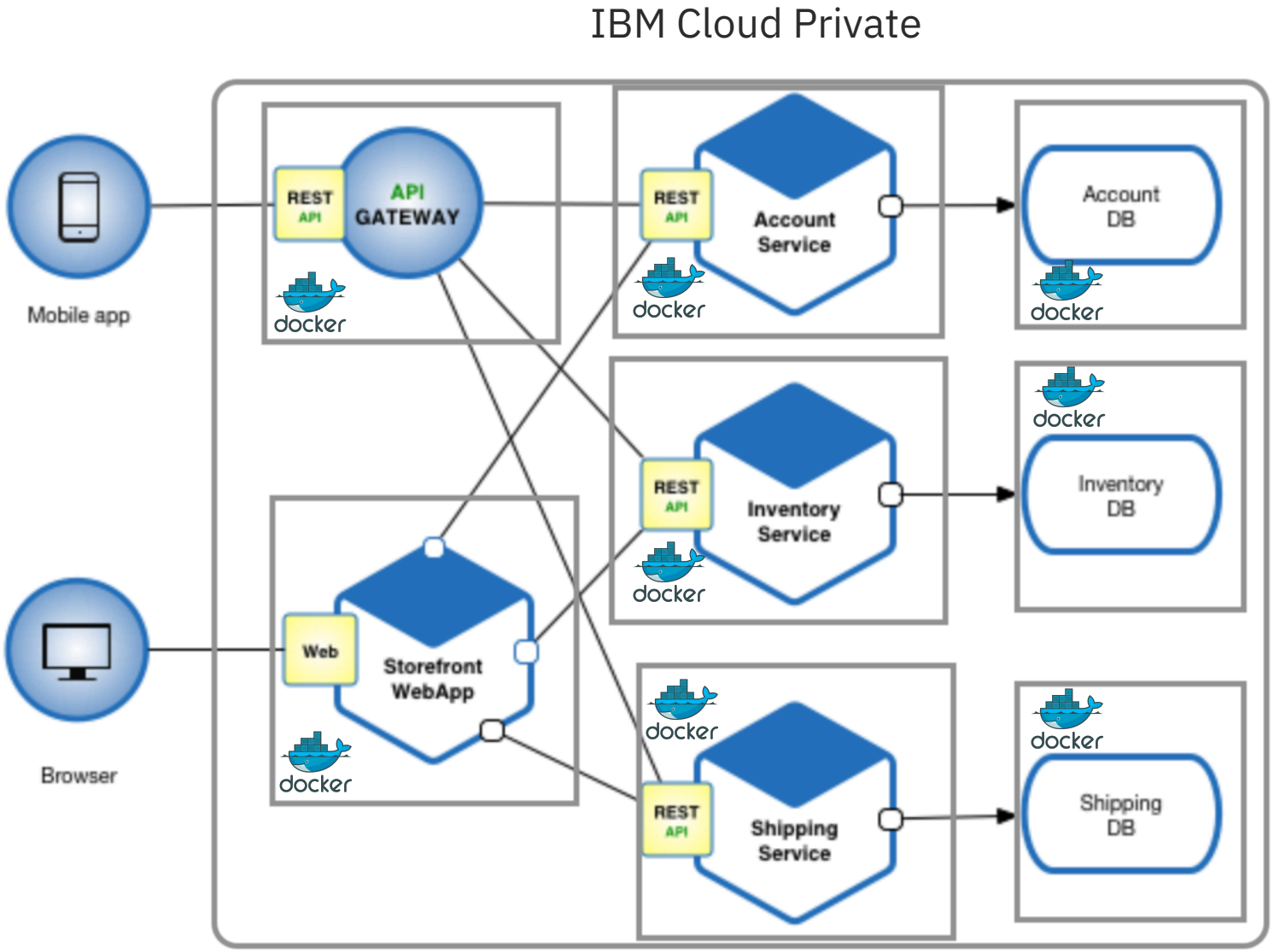
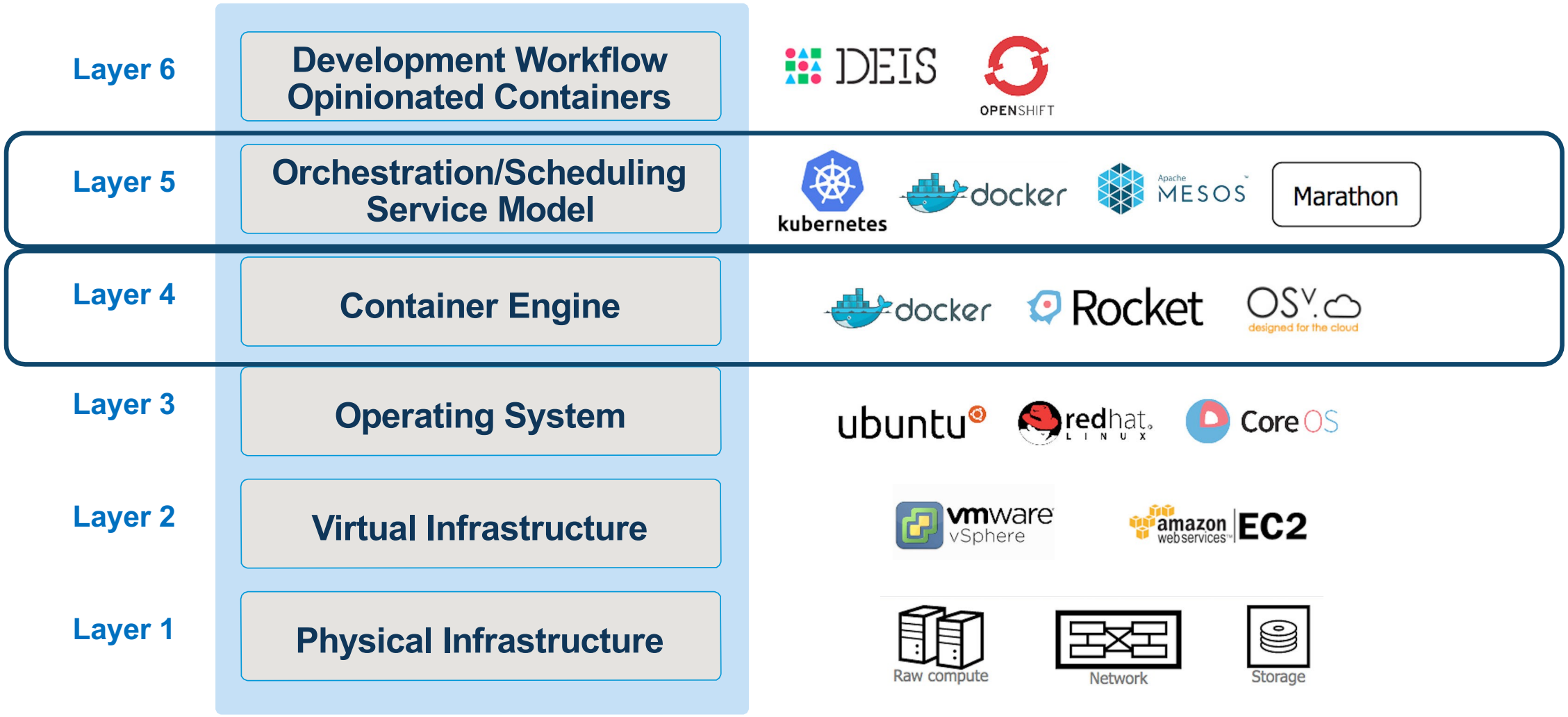


Fig 1. An eCommerce Java microservices app on an ICP Kubernetes cluster

Layers


Layer 6
Development Workflow Opinionated Containers
DEIS
OPENSHIFT
Layer 5
Orchestration/Scheduling Service Model
kubernetes
docker
Apache MESOS
Marathon
Layer 4
Container Engine
docker
Rocket
OSv designed for the cloud
Layer 3
Operating System
ubuntu
redhat LINUX
CoreOS
Layer 2
Virtual Infrastructure
vmware vSphere
amazon web services EC2
Layer 1
Physical Infrastructure
Raw compute
Network
Storage