Introduction to Containers, Kubernetes and ICP

Mangesh Patankar – Developer Advocate
mapatank@in.ibm.com
@MangeshPatank

Raghavendra Deshpande – Developer Advocate
ragdeshp@in.ibm.com
@ragdeshp

Karan Chaturvedi – Developer Advocate
karanchat@in.ibm.com
@Ec08Karan

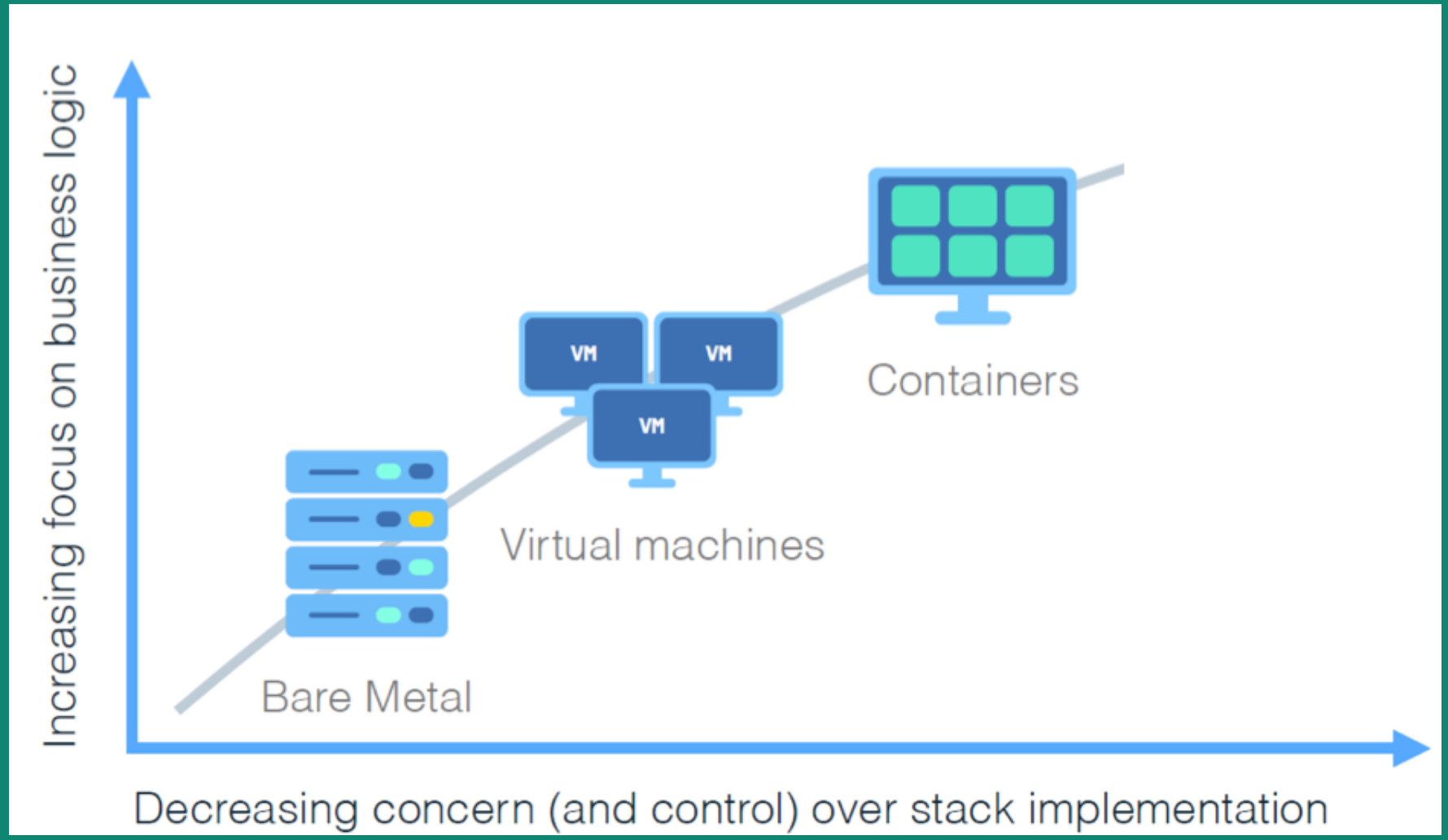
Agenda

- Introduction to Containers and Docker
- What is Kubernetes?
- Kubernetes Architecture – Resources
- Lab : Docker, Kubernetes Cluster – IBM Cloud

- Lunch Break

- YAML
- Lab : Kubernetes Application Deployment
- Kubernetes @ IBM
- ICP – IBM Cloud Private – Overview, Demo
- Helm - Overview, Demo
- Summary

Evolution of Servers

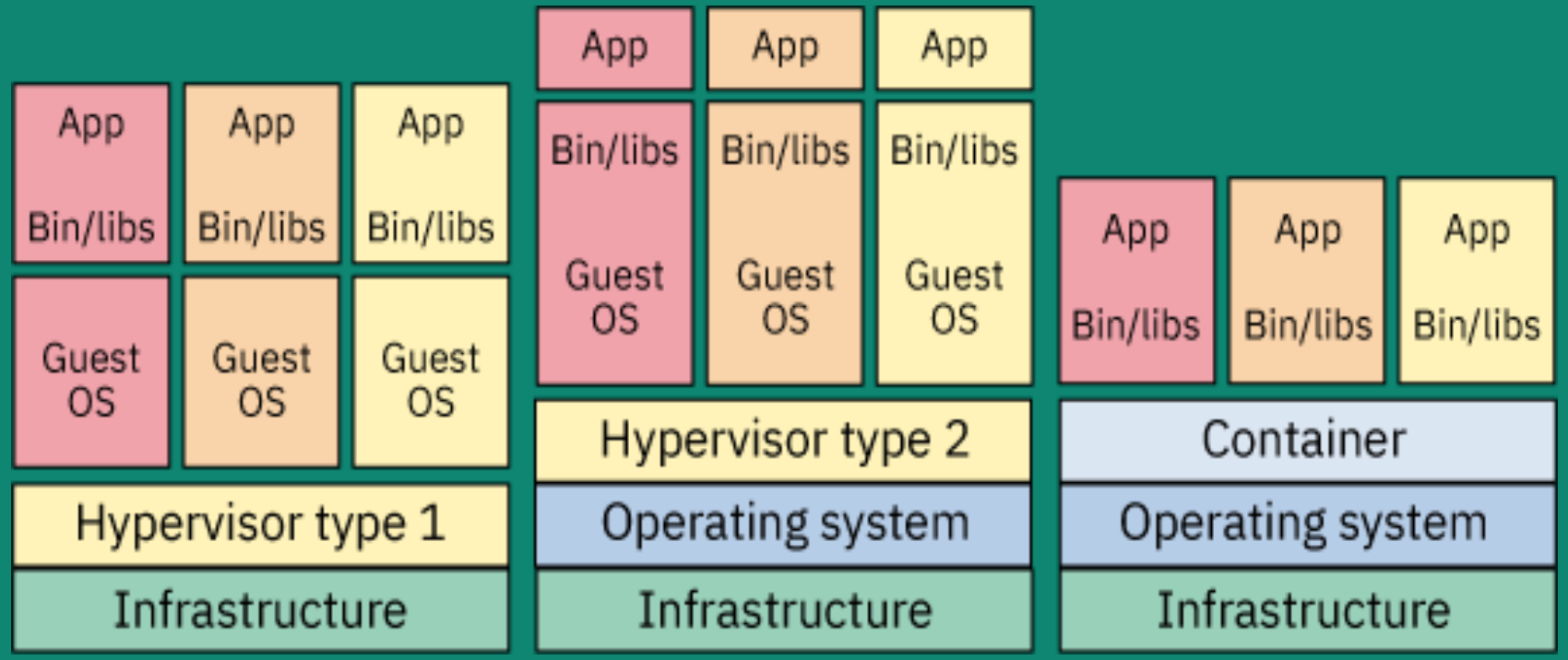


What are Containers?

- A group of processes run in isolation
  - Similar to VMs but managed at the process level
  - All processes MUST be able to run on the shared kernel

- Each container has its own set of "namespaces" (isolated view)
  - **PID** - process IDs
  - **USER** - user and group IDs
  - **UTS** - hostname and domain name
  - **NS** - mount points
  - **NET** - Network devices, stacks, ports
  - **IPC** - inter-process communications, message queues
  - **cgroups** - controls limits and monitoring of resources

VM vs Container



Each VM has its own OS

Containers share the same base Kernel

App, bins/libs/OS must all be runnable on the shared kernel

If OS files aren't needed they can be excluded.

Why Containers?

- Fast startup time - only takes milliseconds to:
  - Create a new directory
  - Lay-down the container's filesystem
  - Setup the networks, mounts, ...
  - Start the process

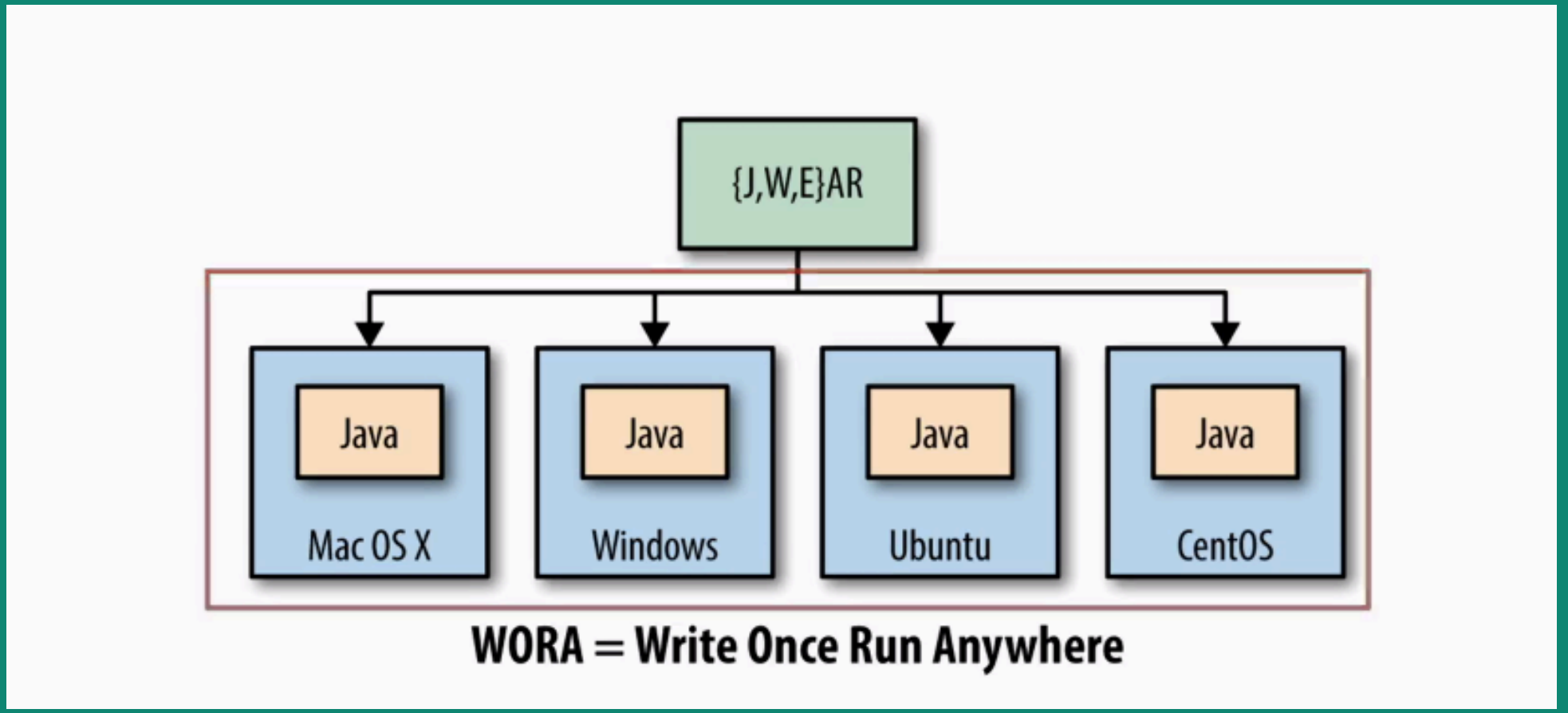
- Better resource utilization
  - Can fit far more containers than VMs into a host

	CONTAINER BENEFITS	VIRTUAL MACHINE BENEFITS
Consistent Runtime Environment	✓	✓
Application Sandboxing	✓	✓
Small Size on Disk	✓	
Low Overhead	✓	

Docker :

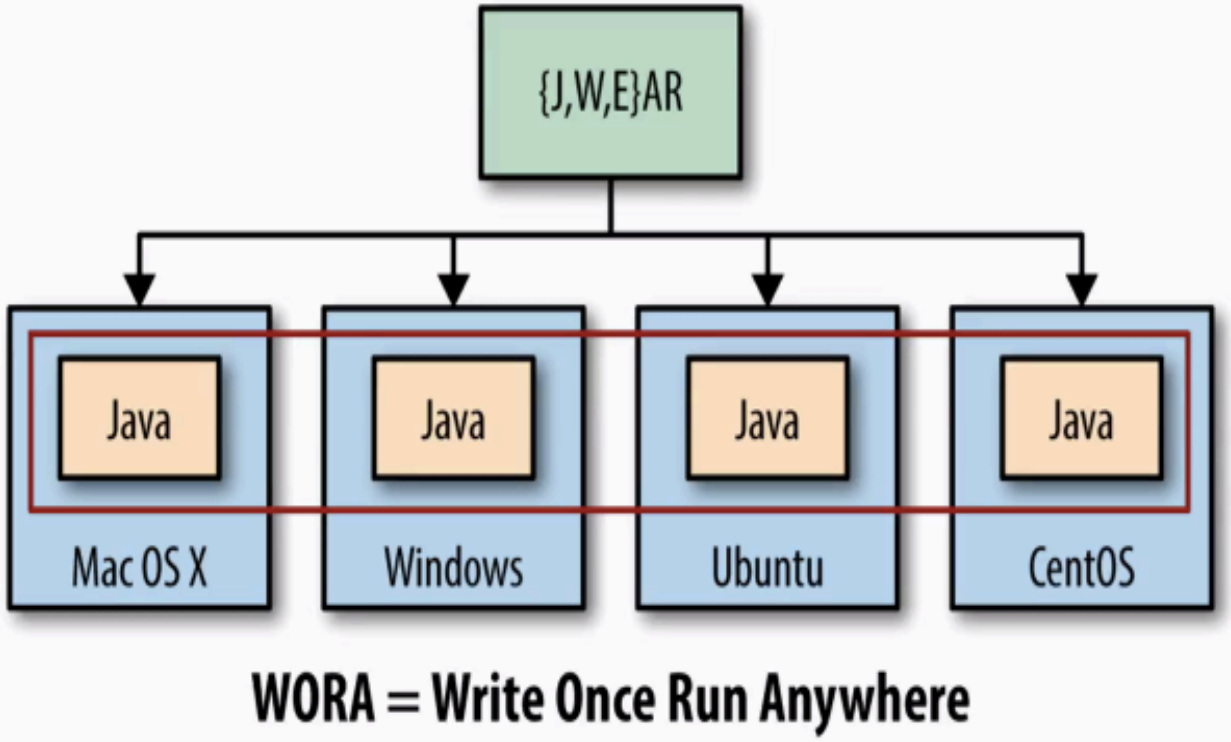
- Why we need Docker
- Package Once and Deploy Anywhere
- Docker Concepts
- Containerization
- Isolation

Java Application



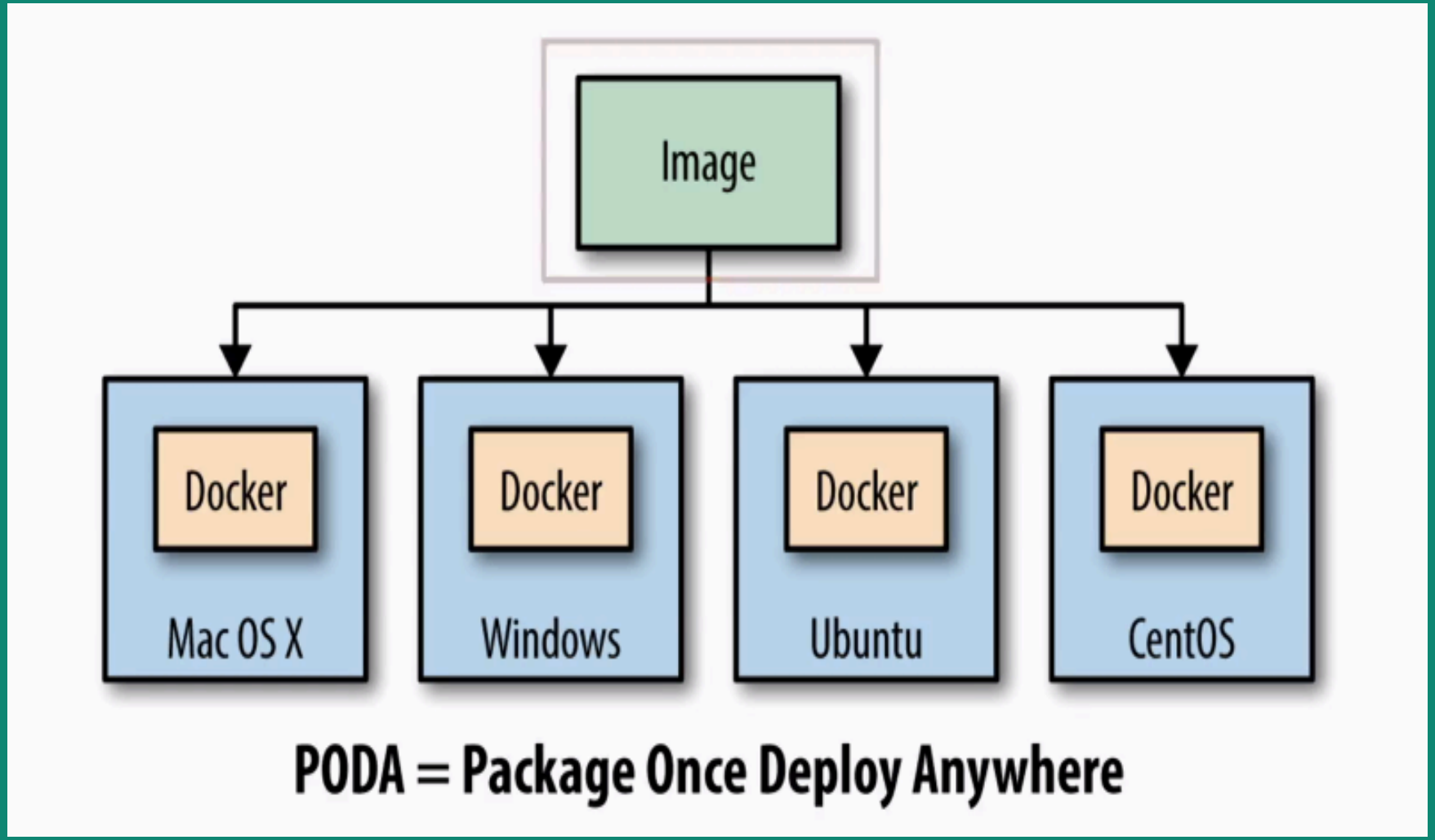
- Each OS has its Java Software
- With same version of Software

Dependencies



Java installed
(Proper version) –

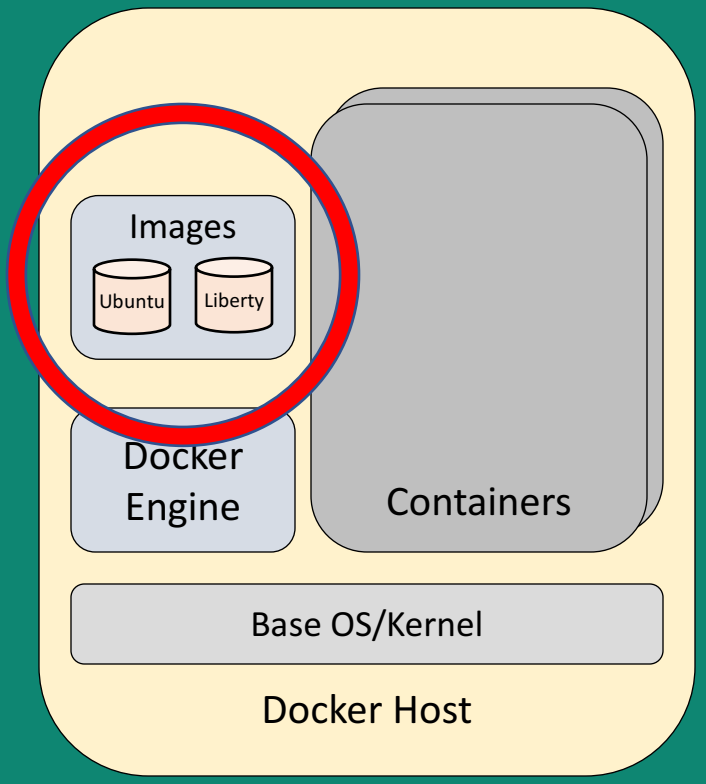
One Step Ahead : Docker



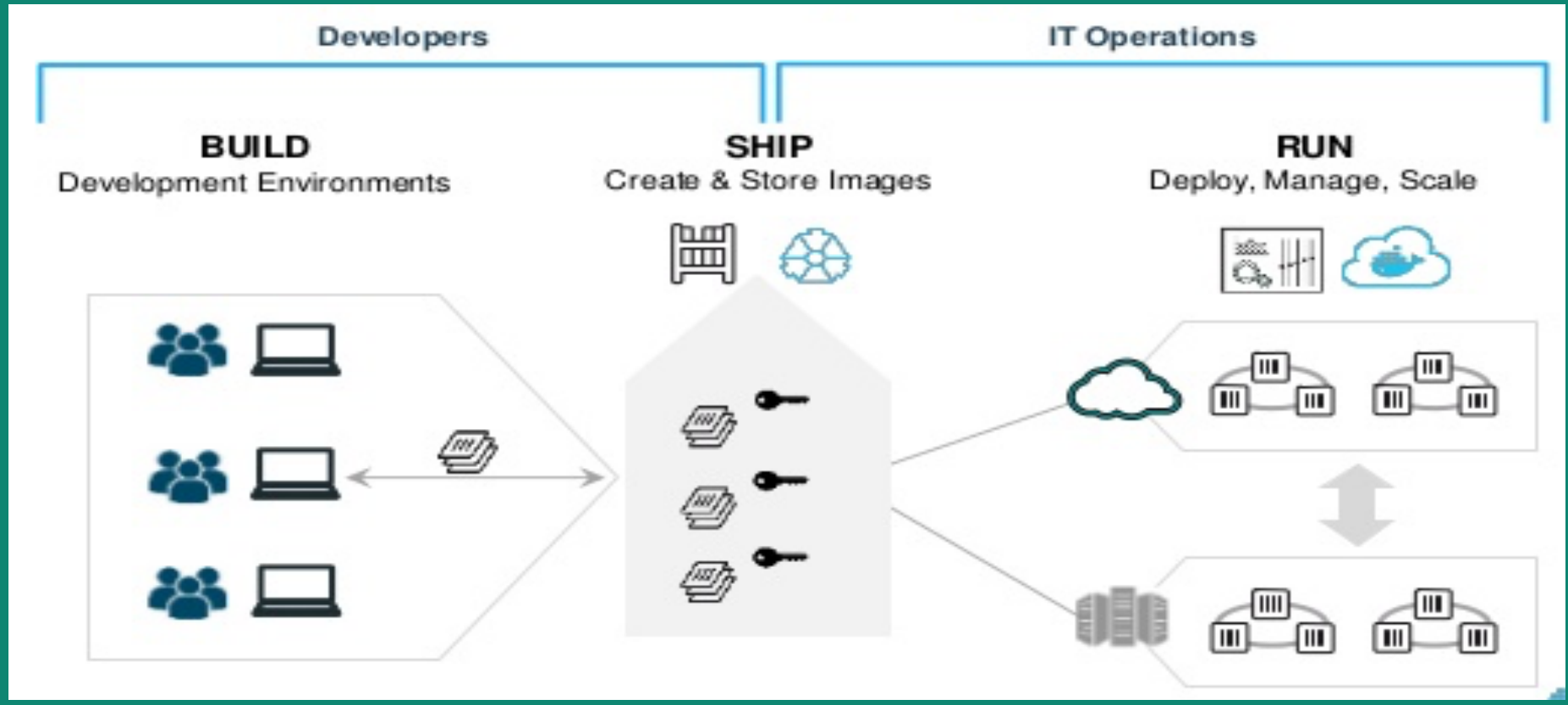
- No installation of software
- Docker Image using Docker Platform
- (so one step ahead)

Docker Images

- Tar file containing a container's filesystem + metadata
- For sharing and redistribution
  - Global/public registry for sharing: DockerHub
- Similar, in concept, to a VM image



Phases : Build , Ship, Run

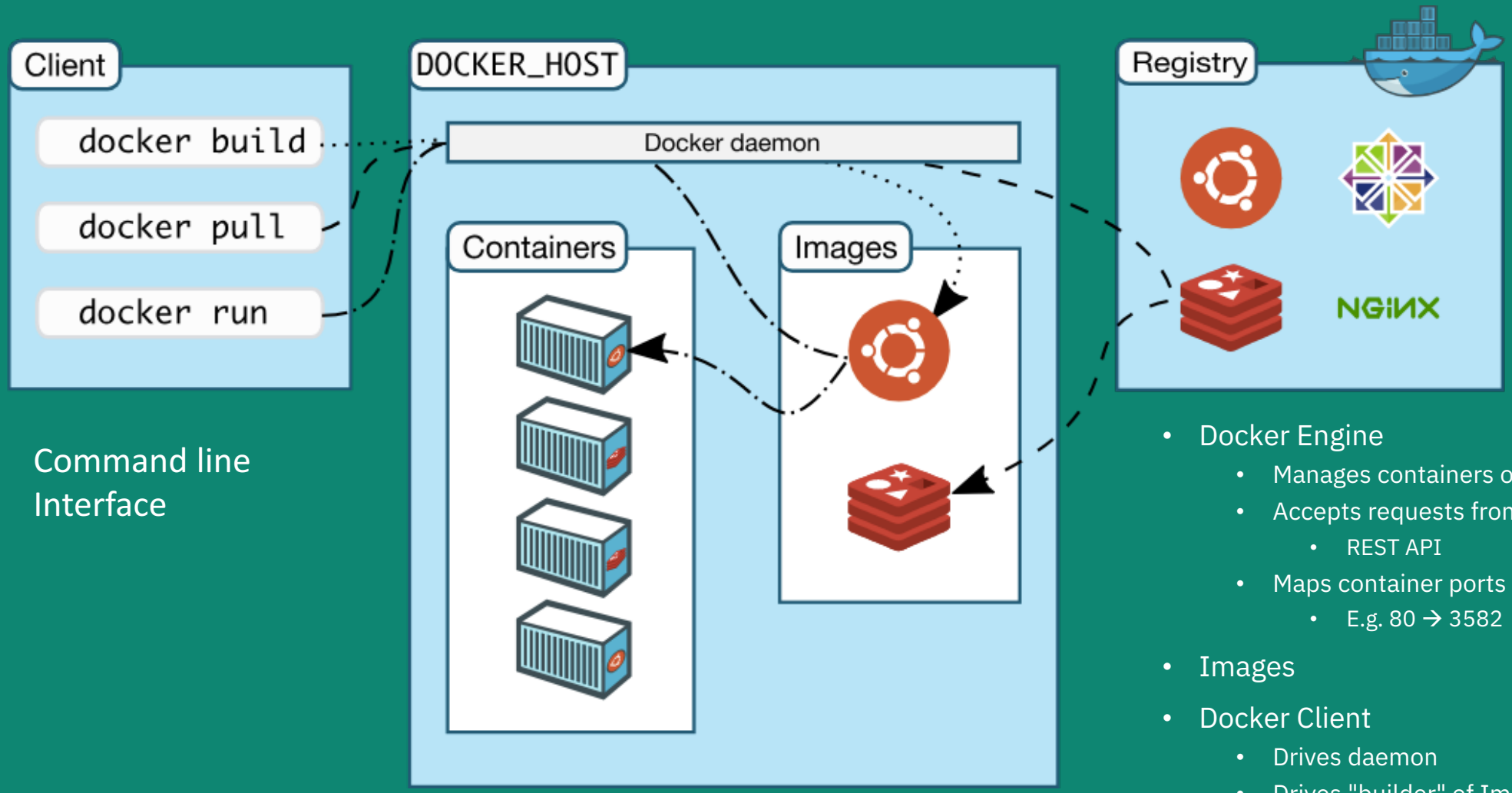


- Tools to create containerized application
- Package App – dependencies, infra as template – This is called Image

- Share Apps in Secure and collaboration manner
- Docker images stored, shared & managed in **Docker Registry**
- **Docker Hub(Public)** default registry for all images

- Deploy, manage and scale apps
- Container is runtime representation of image
- Lifecycle of container – run, start, scale, stop, move, delete

Docker Architecture



Command line Interface

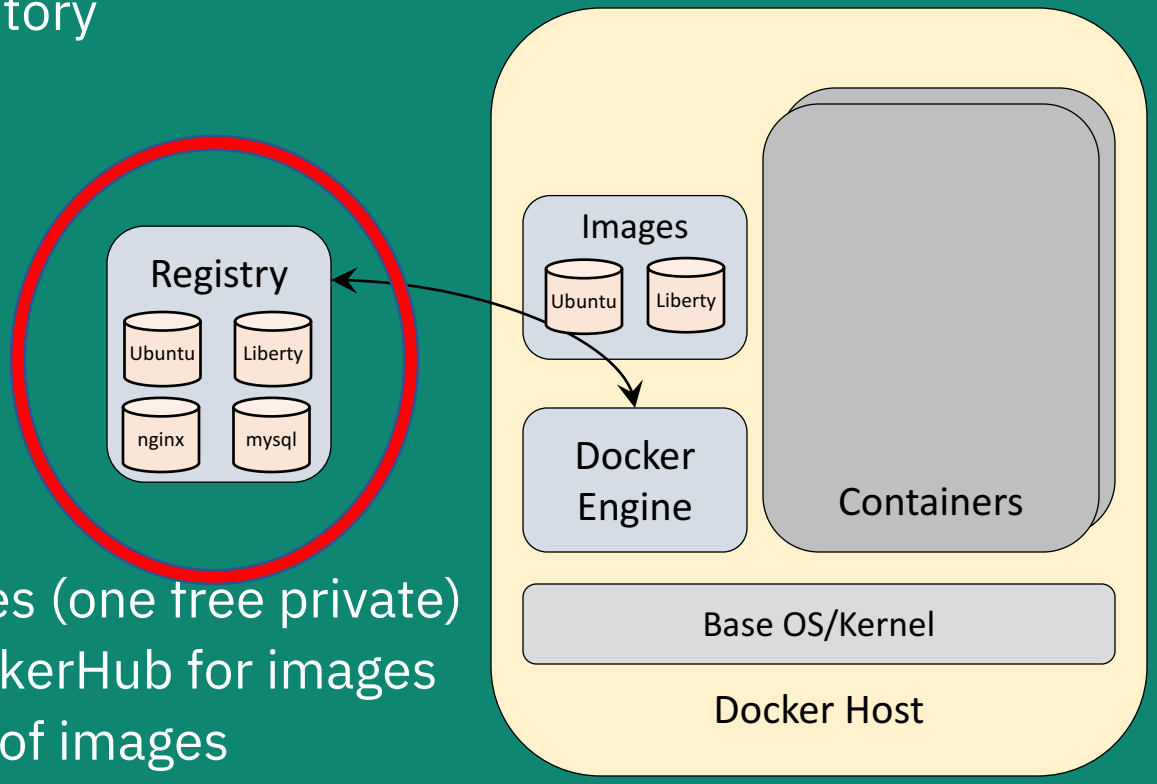
Docker Engine

- Docker Engine
  - Manages containers on a host
  - Accepts requests from clients
    - REST API
  - Maps container ports to host ports
    - E.g. 80 → 3582
- Images
- Docker Client
  - Drives daemon
  - Drives "builder" of Images
- Docker Registry
  - Image DB

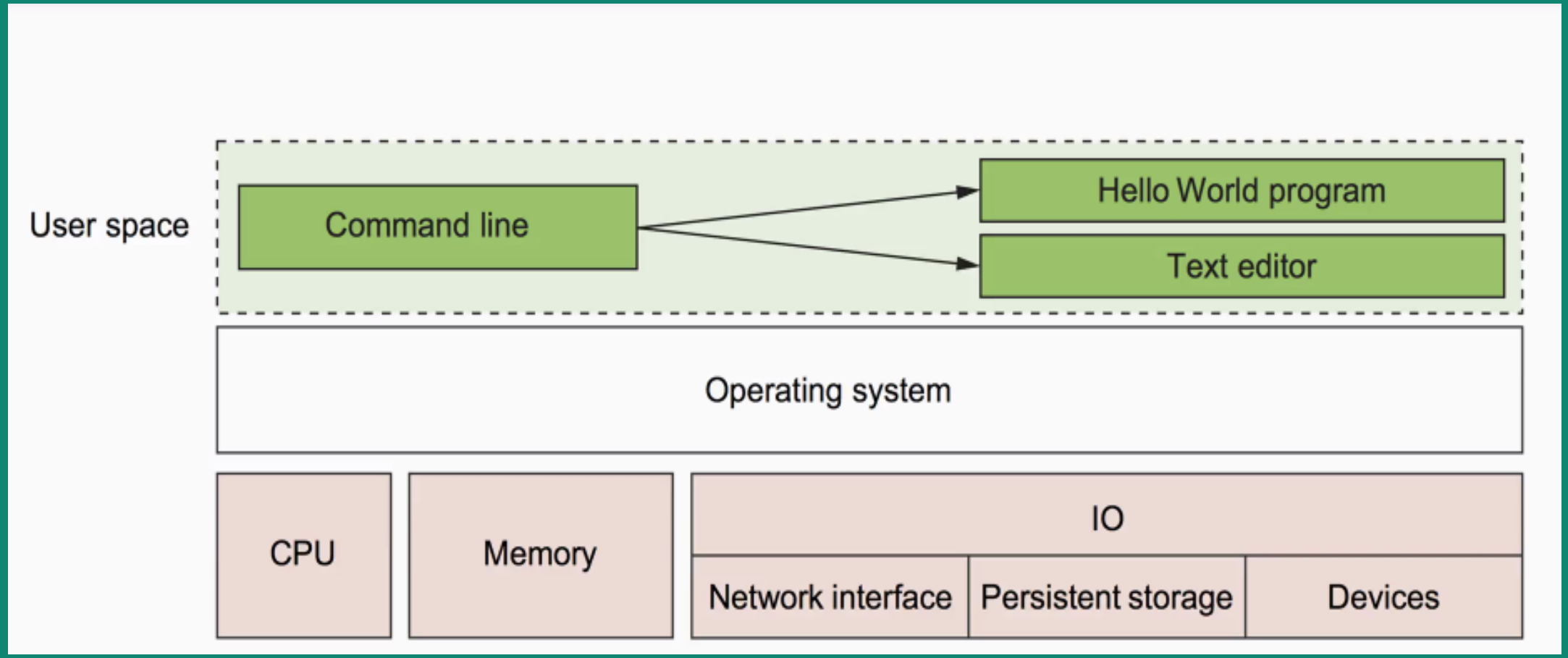
Docker Registry

- Creating and using images is only part of the story
- Sharing them is the other

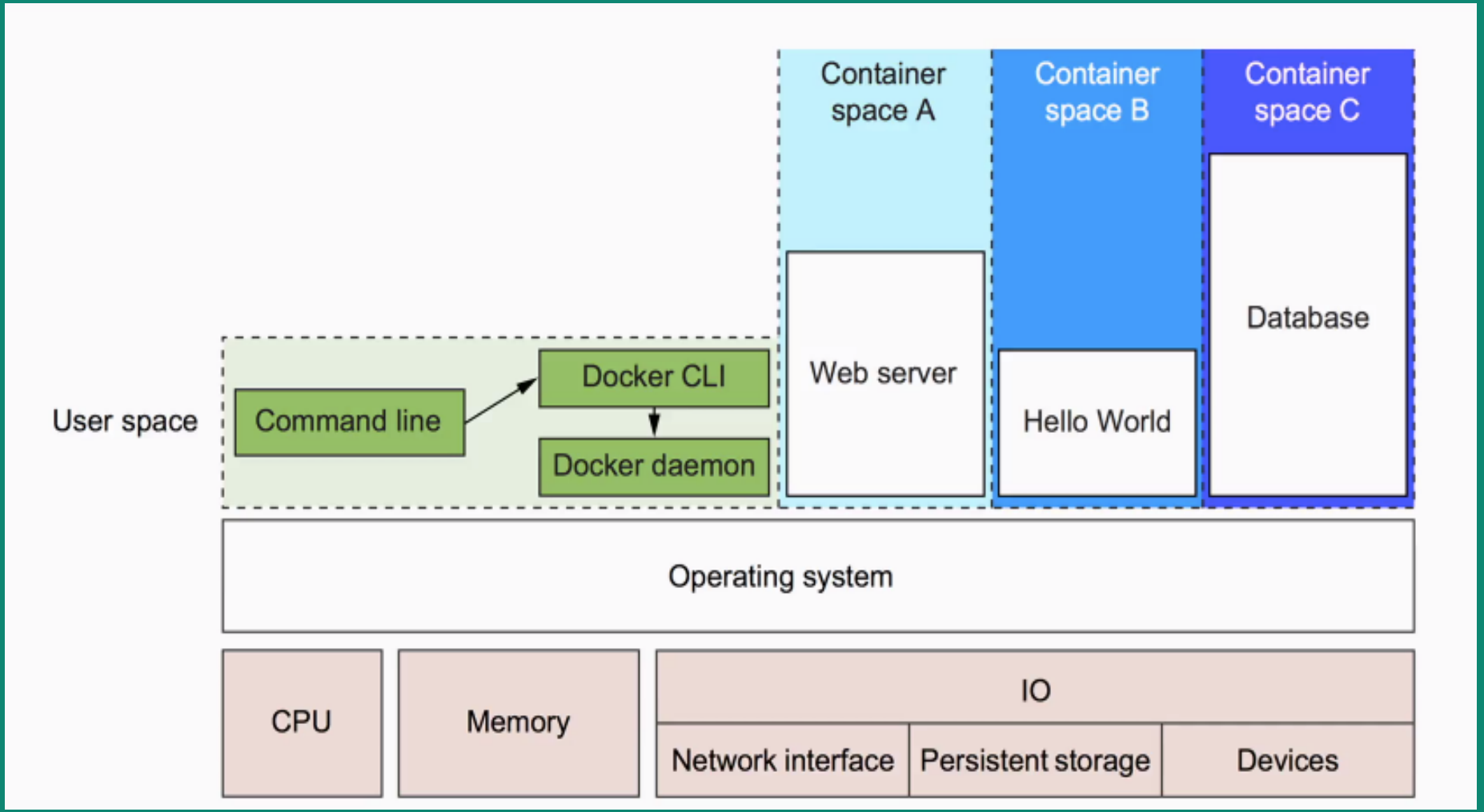
- DockerHub - http://hub.docker.com
  - Public registry of Docker Images
  - Hosted by Docker Inc.
  - Free for public images, pay for private ones (one free private)
  - By default docker engines will look in DockerHub for images
  - Web interface for searching, descriptions of images



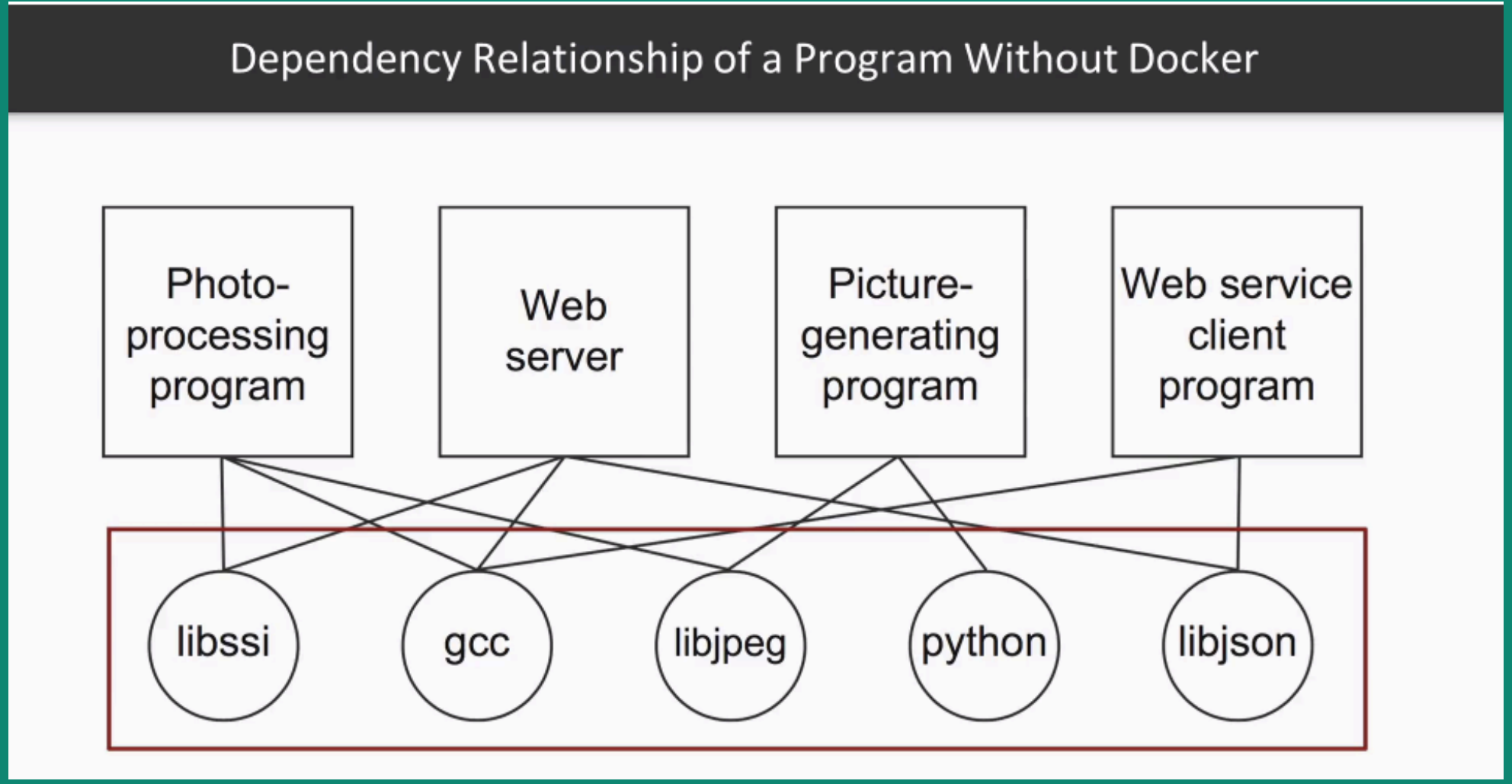
Isolation : Linux



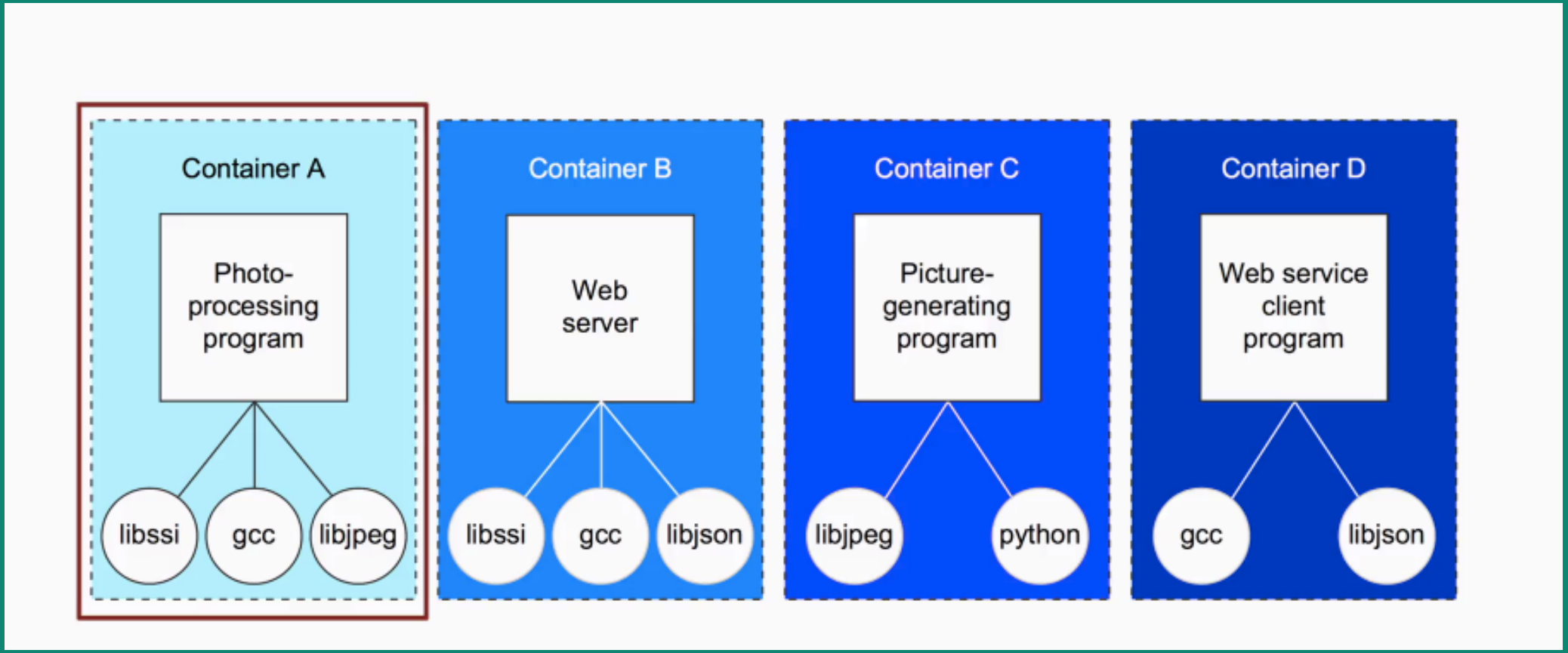
Docker Isolation : Process Level



Running programs without docker



Each App : With needed Dependencies



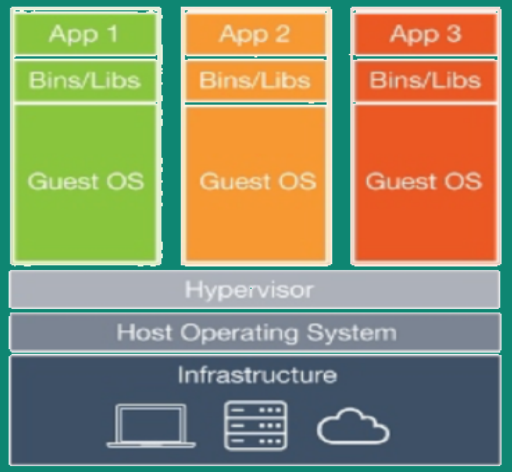
So : What is Docker?

- At its core, Docker is tooling to manage containers
  - Docker is not a technology, it’s a tool or platform
  - Simplified existing technology to enable it for the masses

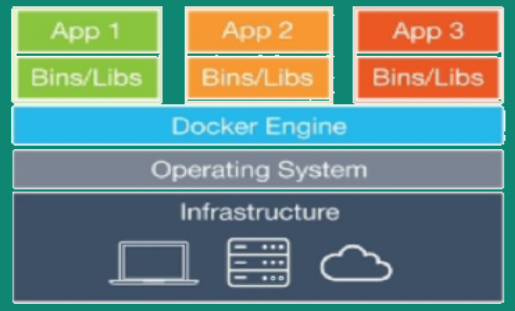
- **Tooling** to manage containers
  - Containers are not new
  - Docker just made them easy to use

- Docker creates and manages the lifecycle of containers
  - Setup filesystem
  - CRUD container
    - Setup networks
    - Setup volumes / mounts
    - Create: start new process telling OS to run it in isolation

Docker Container Vs VMWare

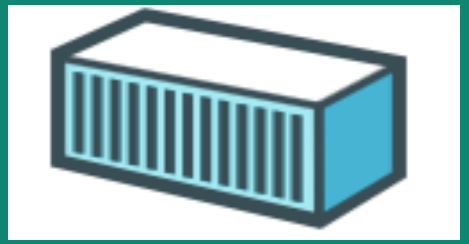


vmware



docker

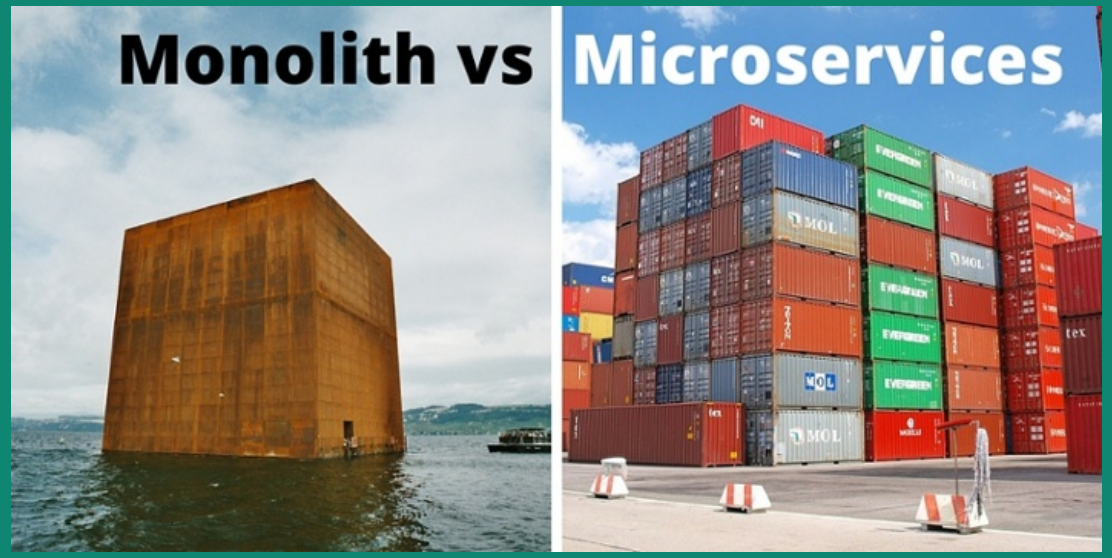
Build

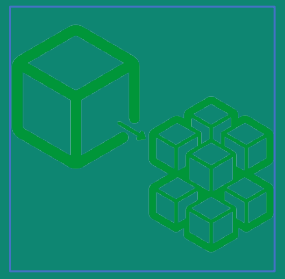


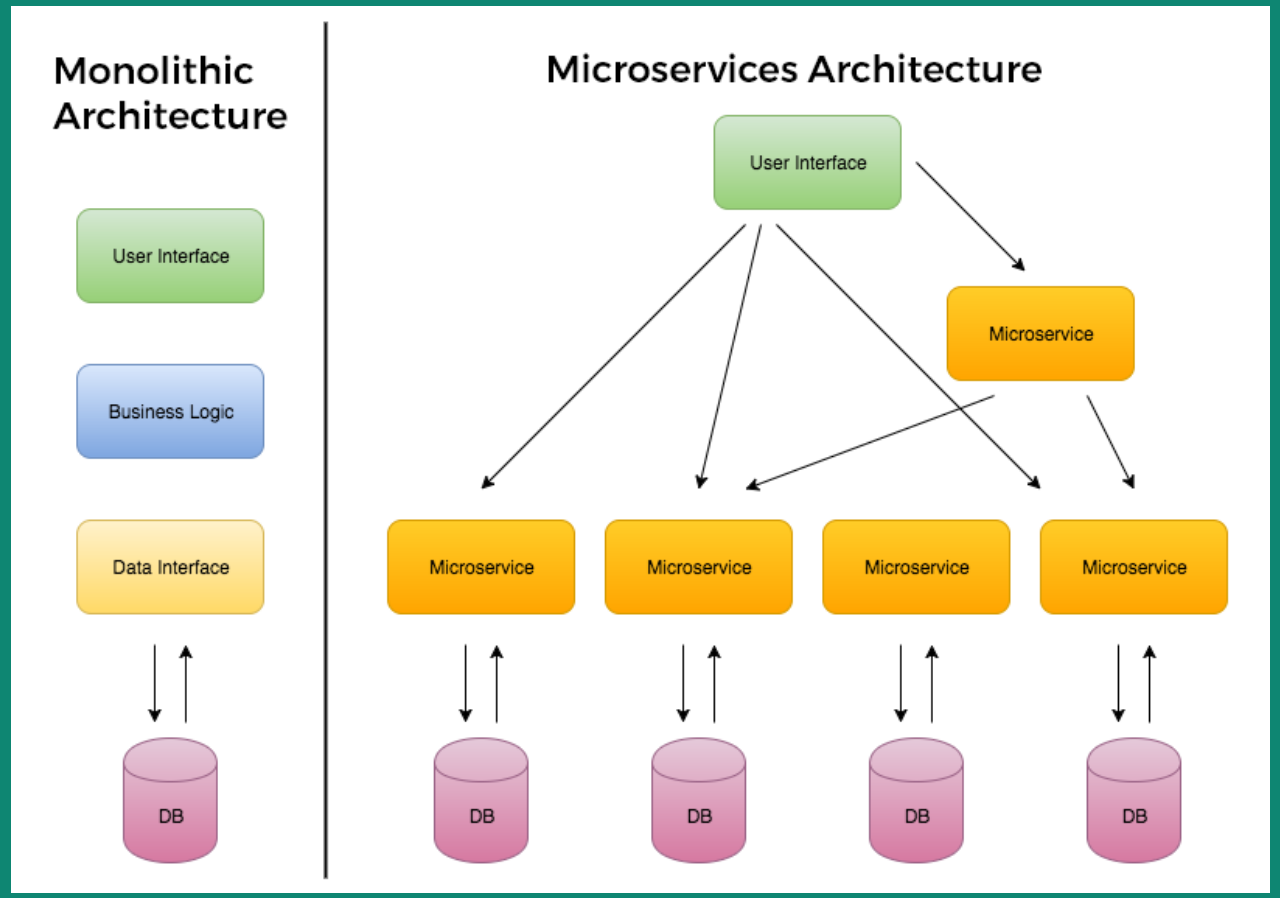
Ship

Run

Microservices & Cloud Native Apps





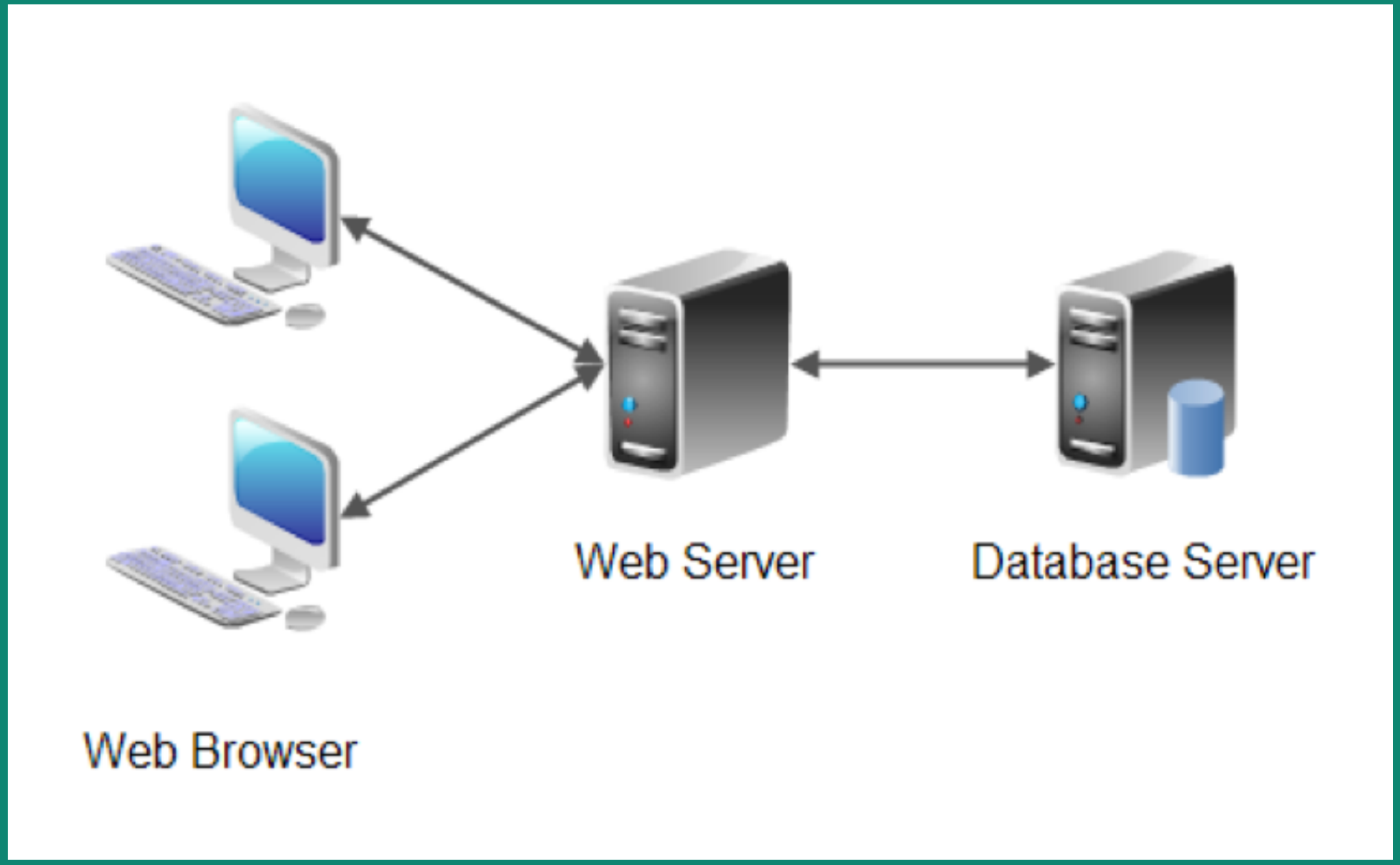


Docker – Use cases

- Server Apps like web servers, mail servers, databases, and proxies
- Desktop software's, tools , email clients etc

Running these in a container and as a user with reduced previlgies will help protect your system from attack

Docker – Use Case – Lab Scenario



Docker : Lab

- Install Docker (Pre-req)
- Get sample code from Github
- Docker build to Create Web app Container : Python based
- Use Redis as a database, and create database container
- Use Docker Compose to connect containers
- Run Docker based application Locally

Docker – in a nutshell

- Docker is just a tool to manage containers
  - Key concepts: Containers, Engine, Images, Registry

- Docker value-add:
  - An excellent User Experience
  - Image Layers
  - Easily shared images - DockerHub

- Why? When compared to VMs:
  - Better resource utilization - CPU, Memory, Disk
  - Faster start-up times
  - Easier tooling/scripting

More on Dockers..

We’re now done with this overview of Docker

There are still many other Docker features that we didn’t touch on

For example

- networking, volumes
- and security,
- Docker Machine ,
- Docker Swarm, etc

http://play-with-docker.com

Why Containers are Appealing to Users

Lightweight & Fast
Faster startup/showdown. Gives services near instant scaling capabilities.

Faster Time to Market
Apps & dependencies are bundled into a single image. Host, OS, distro and deployment are independent allowing for workload portability.

Version Tracking
User easily rolls between versions

Simplified Isolation
Each container has its own network stack with controls over ports and permissions.

Enhanced Security
Containers allow for finer-grained control over data and software installed. Reduces the attack surface area/vulnerabilities of the apps.

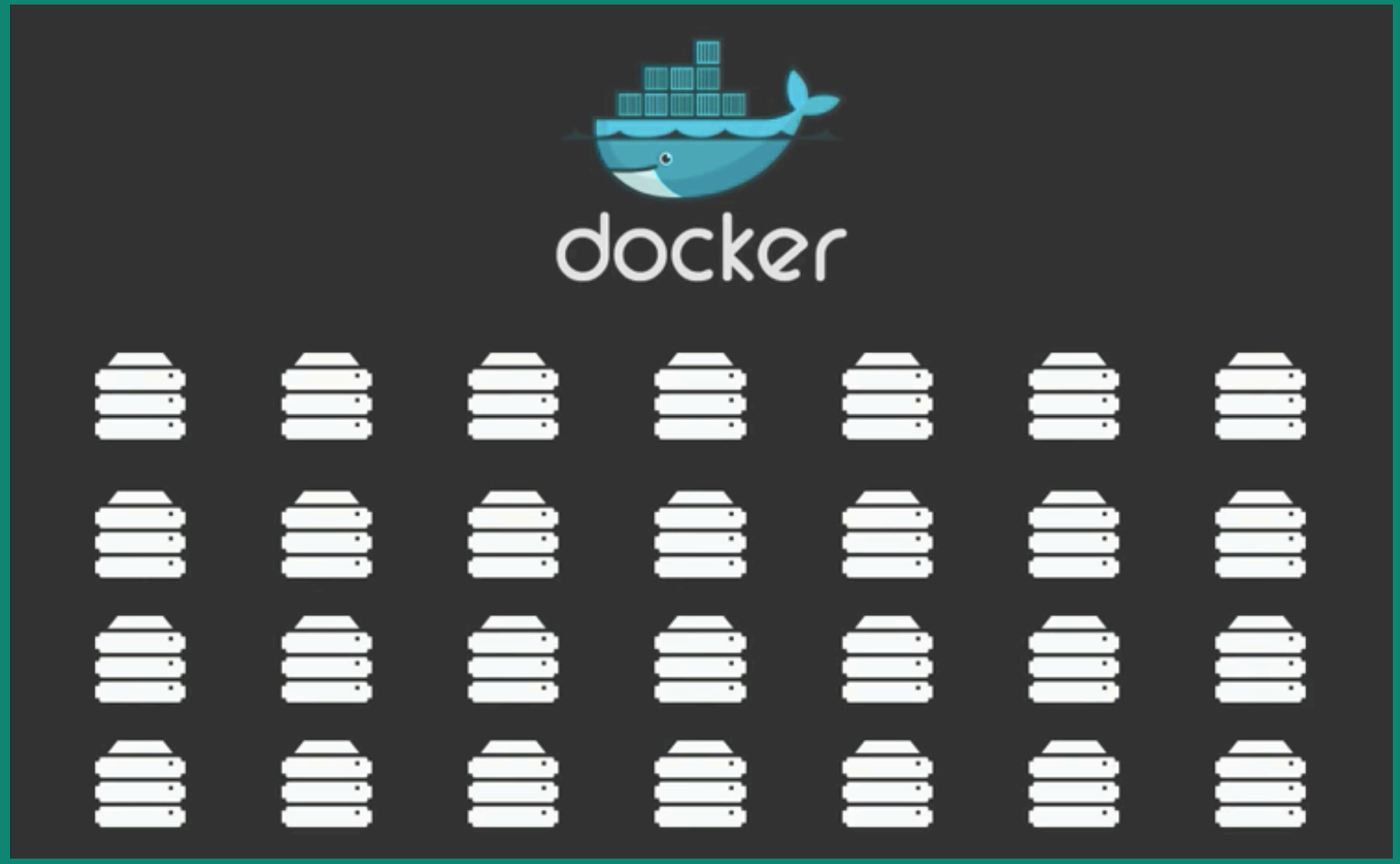
Easier to Manage
Enables frequent patch of applications while reducing the effort of validating compatibility between apps/environment.

Simpler to Maintain
Install, run, maintain and upgrade applications and their envs quickly, consistently and more efficiently than VMs.

Resource Friendly
Can host more containers then corresponding VMs.

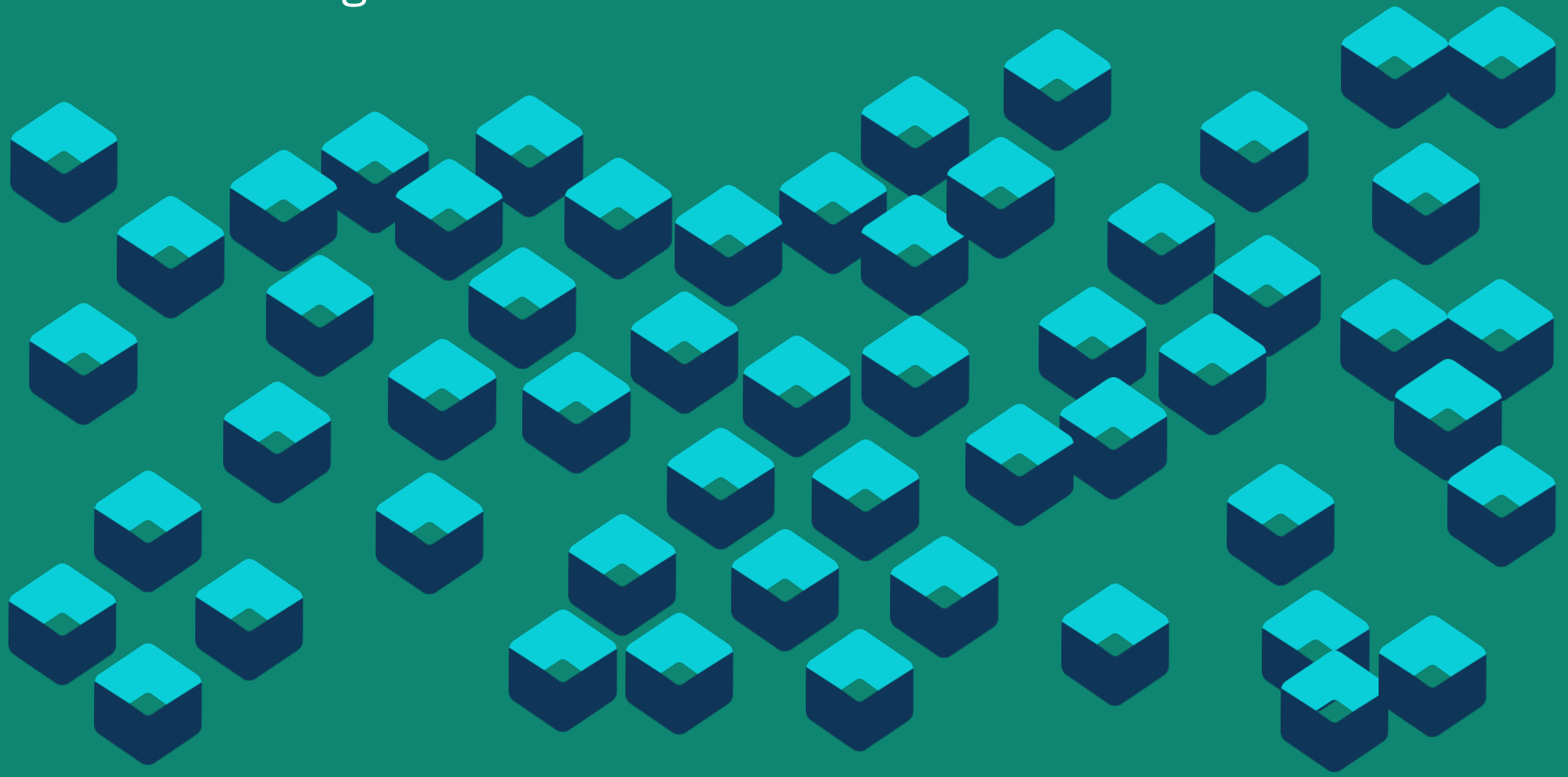
Deployed on 3 servers – Simple

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

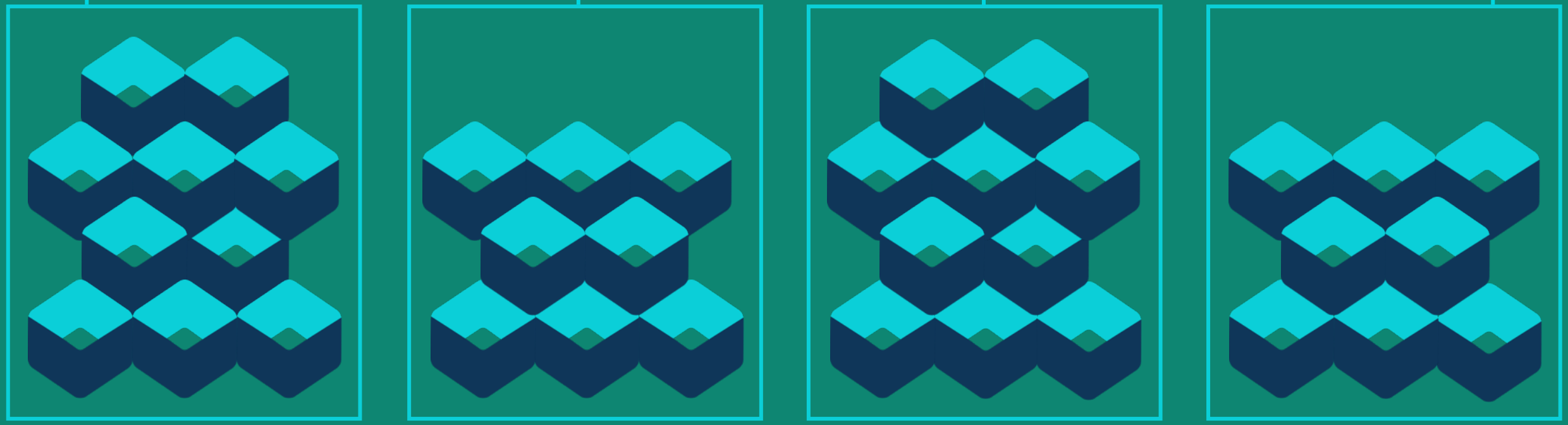
Containers are great but … can lead into lack of control & chaos



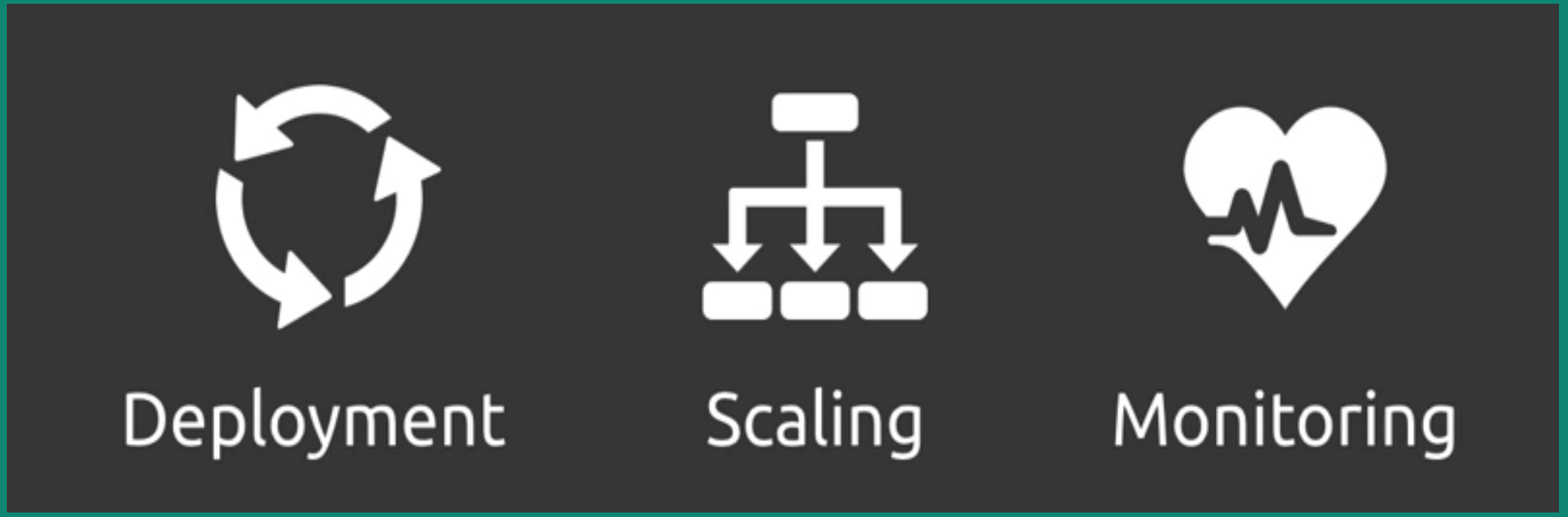
Kubernetes – (Κυβερνήτης - Captain in Greek)

Regain control with Containers and Kubernetes

- Organize and Govern the Container Chaos



Kubernetes – Platform to work with containers



The need for container orchestration

Through docker, Containers became more popular, are becoming standard unit of deployment

- But What you do in a situation when
  - Application grows - multiple, independent microservice
  - Experience higher load
  - Applications running with a huge number of components, can be tricky
  - Critical Services
  - Releasing new features – Agile Application Development

- As a developer you have building docker images, pushing images and creating containers locally
  - Way to take those docker containers and make them into distributed systems.
  - Run large scale production based containers – on small and large systems, cloud provider .

So - What is Kubernetes?

- Open Source - Container Orchestrator
- Help to automate DevOps – Deployment, scaling and management of containerized apps
- Helps organize container in logical units(pods, nodes)
- Will relaunch apps that have failed – (self healing)
- Expose apps as service – load balance request between multiple apps
- Usually installed on Cloud Infra (public) as well on private, hybrid
- Also a application platform that makes app deployment easier and automate (declarative)
- What's in a name? Kubernetes (K8s/Kube): "Helmsman" in ancient Greek

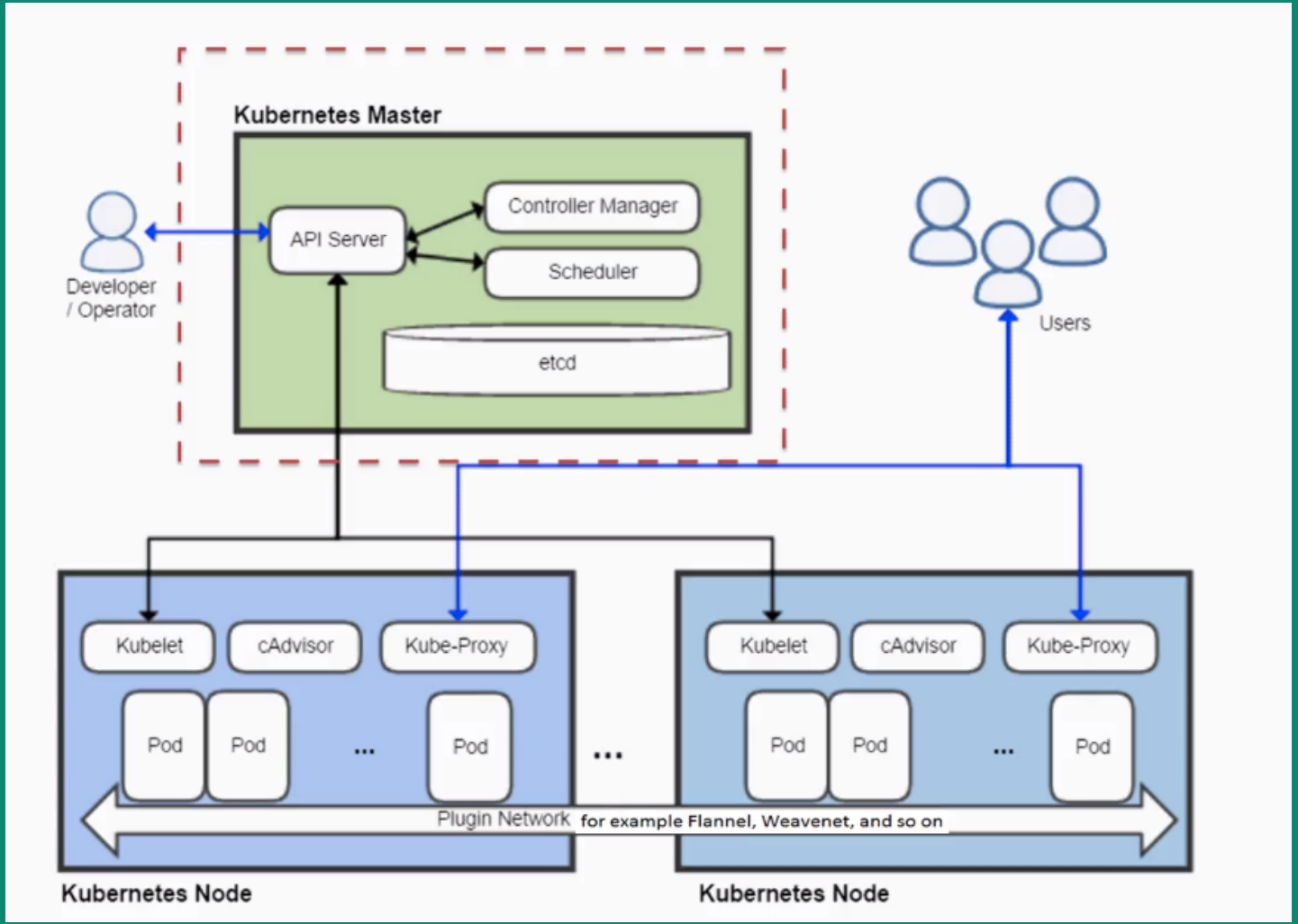
How was Kubernetes created

- Based on Google's Borg & Omega
- Open Governance
  - Cloud Native Compute Foundation
- Adoption by Enterprise
  - RedHat, Microsoft, IBM and Amazon

Where is Kubernetes

- Main Website - http://kubernetes.io
- Source Code - https://github.com/kubernetes
- Youtube Channel
- Many SIG's(Special Interest Groups), Zoom

Kubernetes Component Architecture



- Master to Expose REST API to interact With Kubernetes platform
- Master communicates with kubelet in Each node to maintain state of cluster
- Kubernetes Cluster – Master and Worker(Knodes)
- Application User requests(http) resolved by Kube-proxy
- Developer interacts with Kubectl to create K8 API objects and manage cluster

- Provides highest level of abstraction : place for containerized applications to live, be managed and orchestrated

Kubernetes Objects – What?

- Are ”record of intent” – once created K8s will constantly work to ensure that object exists
- Are persistent entities in K8s System
- Are used to represent the state of your cluster

Kubernetes Object

- Describes, What containerized applications are running and on which nodes
- Describes, The resources available to those apps
- Describes, the policies around how those applications behave, such as restart policies, upgrades and fault-tolerance

Kubernetes Objects - Define

- Can be defined as a YAML configuration with two main parts – specs and status

Two nested objects – Spec and Status

- Spec – desired state for the object – characteristics that object should have

- Status – The actual state of the object . Supplied and updated by K8s

Kubernetes Objects

Pod

- Pod/s are smallest and simplest k8s object
- Pod represents a set of running containers on cluster
- Pod typically set up to run a single primary container. It can also run optional containers that add supplementary features like logging
- Pods are commonly managed by a deployment definition/object

Deployment

- This object describe desire state
- Deployment controller changes the actual state to desired state at controlled rate
- Define deployments to create new ReplicaSets or remove existing deployments and adopt all their resources with new deployments

ReplicaSet

- Ensures that specified no of pods replica are running at any given time
- Provide mechanism for us to define the number of app instances , to be maintained
- Should be defined within a deployment object – rather than being manipulated directly
- Provides declarative updates to pods

Service

- Is an API object that describes how to access applications such as a set of Pods
- Is an abstraction which defined a logical set of Pods and a policy by which to access them , sometimes called a micro service
- Kube-Proxy in each worker node is responsible for managing virtual IPs for the service

- https://kubernetes.io/docs/concepts/services-networking/service/

Kubernetes Resource Model

A resource for every purpose

- Config Maps
- Daemon Sets
- **Deployments**
- Events
- Endpoints
- Ingress
- Jobs
- Nodes
- Namespaces
- **Pods**
- Persistent Volumes
- Replica Sets
- Secrets
- Service Accounts
- **Services**
- Stateful Sets, and more…

- Kubernetes aims to have the building blocks on which you build a cloud native platform.

- Therefore, the internal resource model **is** the same as the end user resource model.

Key Resources

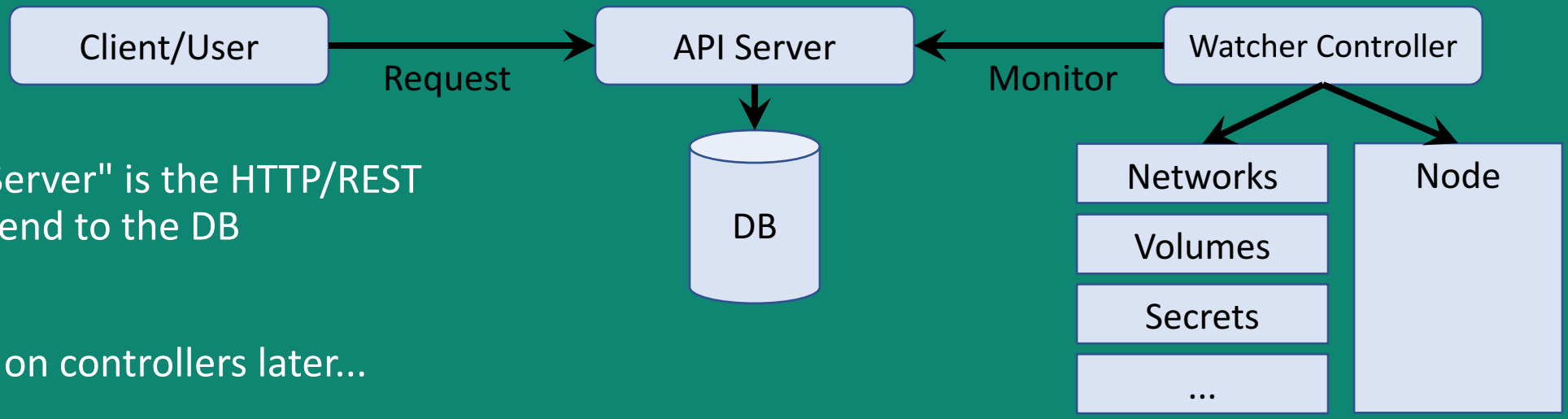
- Pod: set of co-located containers
  - Smallest unit of deployment
  - Several types of resources to help manage them
  - Replica Sets, Deployments, Stateful Sets, …

- Services
  - Define how to expose your app as a DNS entry
  - Query based selector to choose which pods apply

Kubernetes Architecture

- At its core, Kubernetes is a database (etcd).
  With "watchers" & "controllers" that react to changes in the DB.
  The controllers are what make it Kubernetes.
  This plug-ability and extensibility is part of its "secret sauce".

- DB represents the user's desired state
  - Watchers attempt to make reality match the desired state



"API Server" is the HTTP/REST front-end to the DB

More on controllers later...

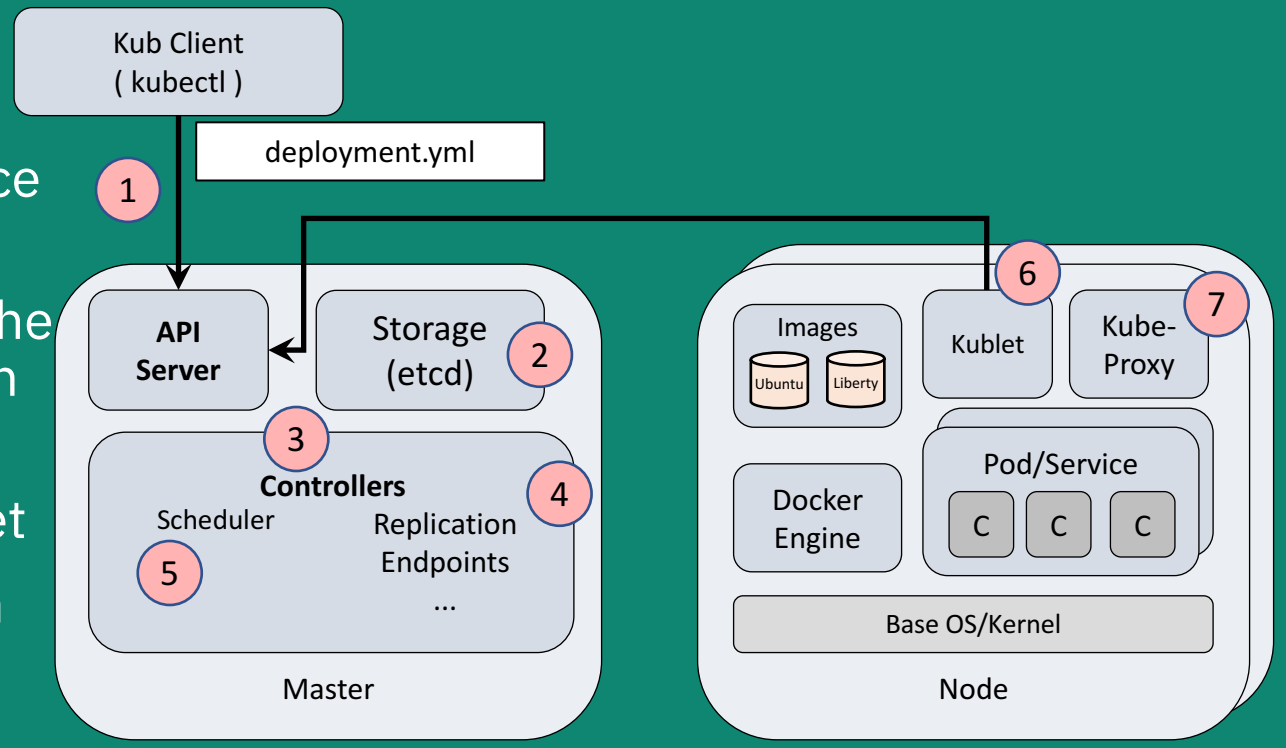
Kubernetes Client

- CLI tool to interact with Kubernetes cluster
- Platform specific binary available to download
  - https://kubernetes.io/docs/tasks/tools/install-kubectl
- The user directly manipulates resources via json/yaml

```
$ kubectl (create|get|apply|delete) -f myResource.yaml
```

Kubernetes in Action! – Application deployment flow

1. User via "kubectl" deploys a new application
2. API server receives the request and stores it in the DB (etcd)
3. Watchers/controllers detect the resource changes and act upon it
4. ReplicaSet watcher/controller detects the new app and creates new pods to match the desired # of instances
5. Scheduler assigns new pods to a kubelet
6. Kubelet detects pods and deploys them via the container runing (e.g. Docker)
7. Kubeproxy manages network traffic for the pods – including service discovery and load-balancing



K8s and Docker Swarm

Kubernetes	**Docker Swarm**
Developed by Google	Developed by Docker Swarm
Has a vast Open source community	Has a smaller community compared to Kubernetes
More extensive and customizable	Less extensive and less customizable
Requires heavy setup	Easy to set up and fits well into Docker ecosystem
Has high fault tolerance	Has low fault tolerance
Provides strong guarantees to cluster states, at the expense of speed	Facilitates for quick container deployment and scaling even in very large clusters
Enables load balancing when container pods are defined as services.	Features automated internal load balancing through any node in the cluster.

- Source https://dzone.com/articles/docker-swarm-vs-kubernetes-what-you-really-need-to

Lab : Kubernetes

- To push an image of an application to IBM Cloud Container Registry

- Pre-requisite
  - Install the CLIs (IBM Cloud, IBM Container Service Plugins and Docker
  - Create a Cluster on IBM Cloud

Lunch Break

Introduction to Containers, Kubernetes and ICP Part 2

Mangesh Patankar – Developer Advocate
mapatank@in.ibm.com
@MangeshPatank

Raghavendra Deshpande – Developer Advocate
ragdeshp@in.ibm.com
@ragdeshp

Karan Chaturvedi – Developer Advocate
karanchat@in.ibm.com
@Ec08Karan

Agenda

Part 2

- YAML
- Lab : Kubernetes Application Deployment
- Kubernetes @ IBM
- ICP – IBM Cloud Private – Overview, Demo
- Helm - Overview, Demo
- Summary

YAML

YAML

- Definition of YAML
- Why YAML?
- Relation to JSON and XML
- Preview of YAML
- Properties vs YAML

YAML Applied

- Kubernetes
- POD creation
- Service creation
- Deployment creation
- Basic and Advanced yaml features
- Code Pattern

Definition of YAML

"YAML is a human friendly data serialization standard for all programming languages."

- www.yaml.org

Why YAML?

- Easily readable by humans, expressive and extensible.
- Easy to implement and use.
- Easily portable between programming languages.
- Matches the native data structures of agile languages.
- Has a consistent model to support generic tools.
- Supports one-pass processing.
- Convenience: You'll no longer have to add all of your parameters to the command line
- Maintenance: YAML files can be added to source control, so you can track changes
- Flexibility: You'll be able to create much more complex structures using YAML than you can on the command line

XML vs JSON vs YAML

XML

```
<Servers>
    <Server>
        <name>Server1</name>
        <owner>John</owner>
        <created>12232012</created>
        <status>active</status>
    <Server>
</Servers>
```

JSON

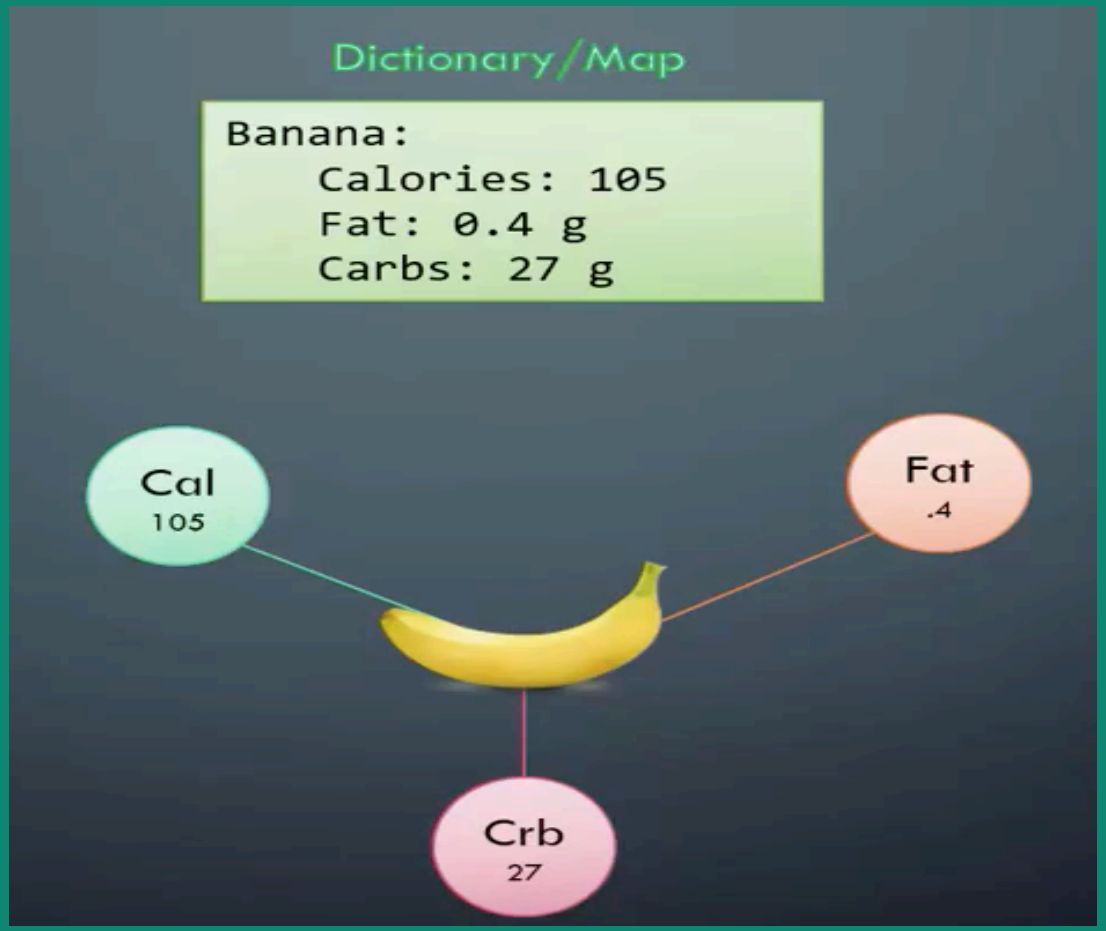
```
{
    Servers: [
        {
        name: Server1,
        owner: John,
        created: 12232012,
        status: active,
        }
    ]
}
```

YAML

```
Servers:
    -   name: Server1
        owner: John
        created: 12232012
        status: active
```

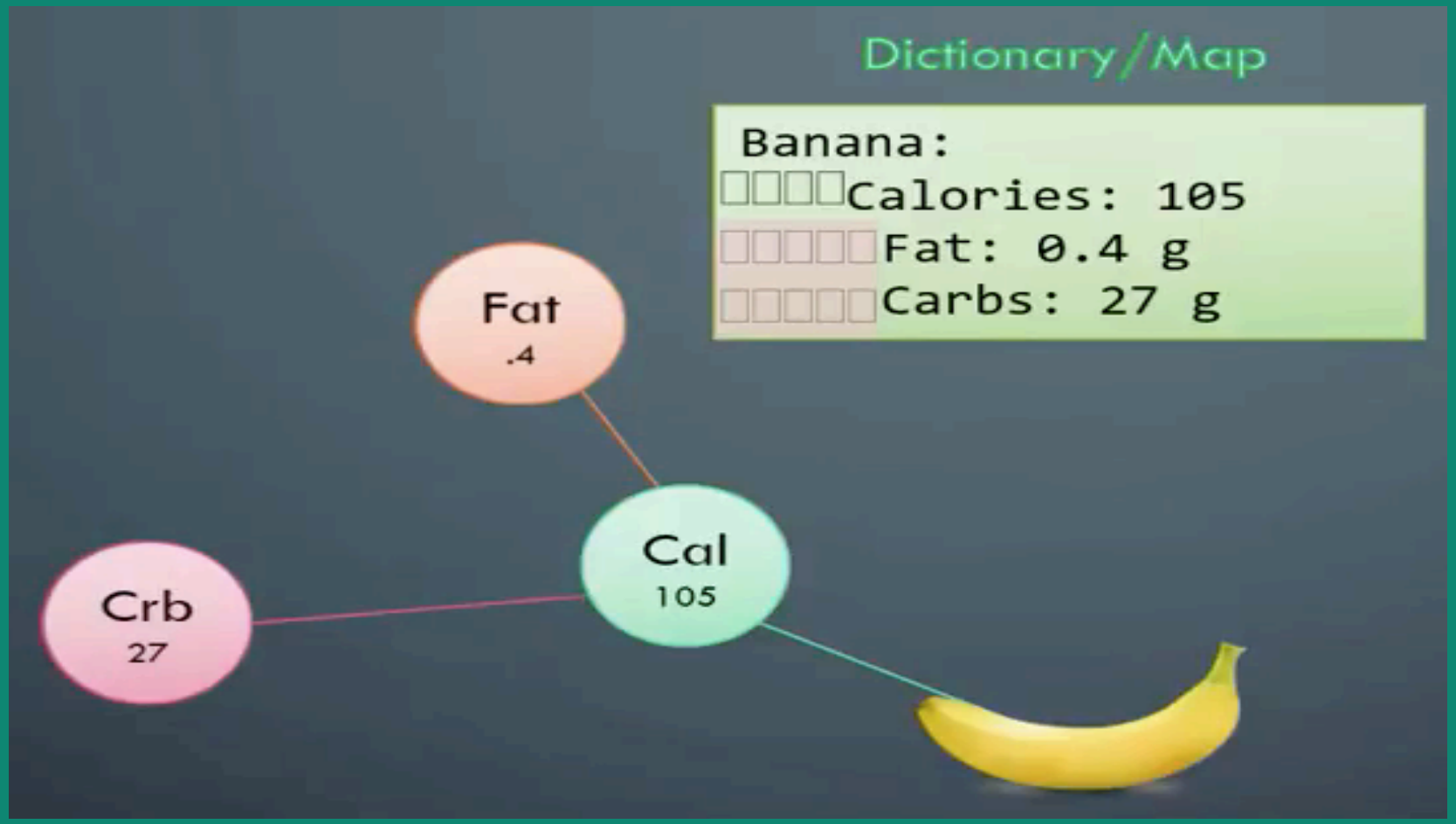
Key Value Pair, Array/Lists, Dictionary/Map

Importance of SPACES in YAML

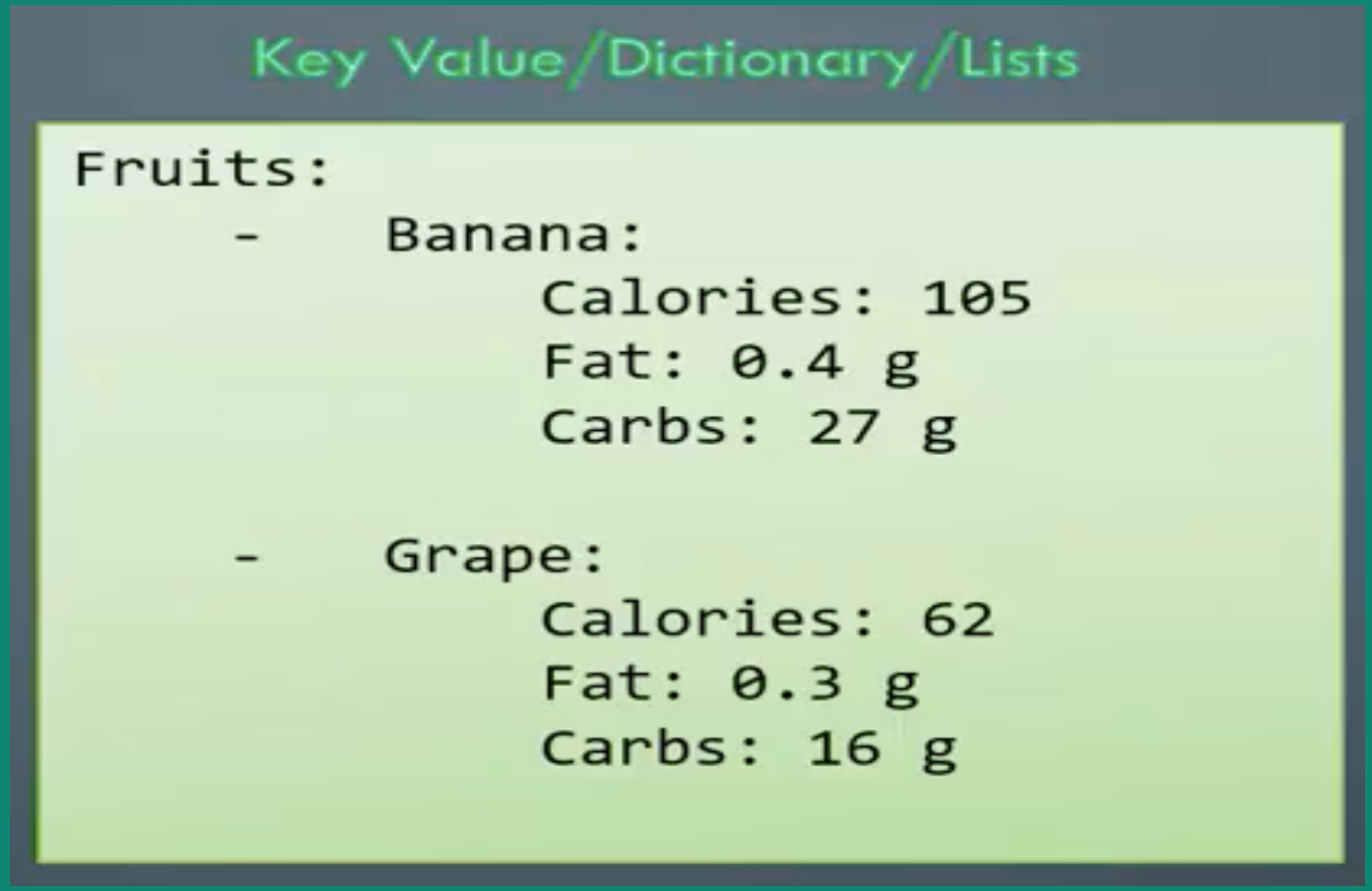


Importance of SPACES in YAML

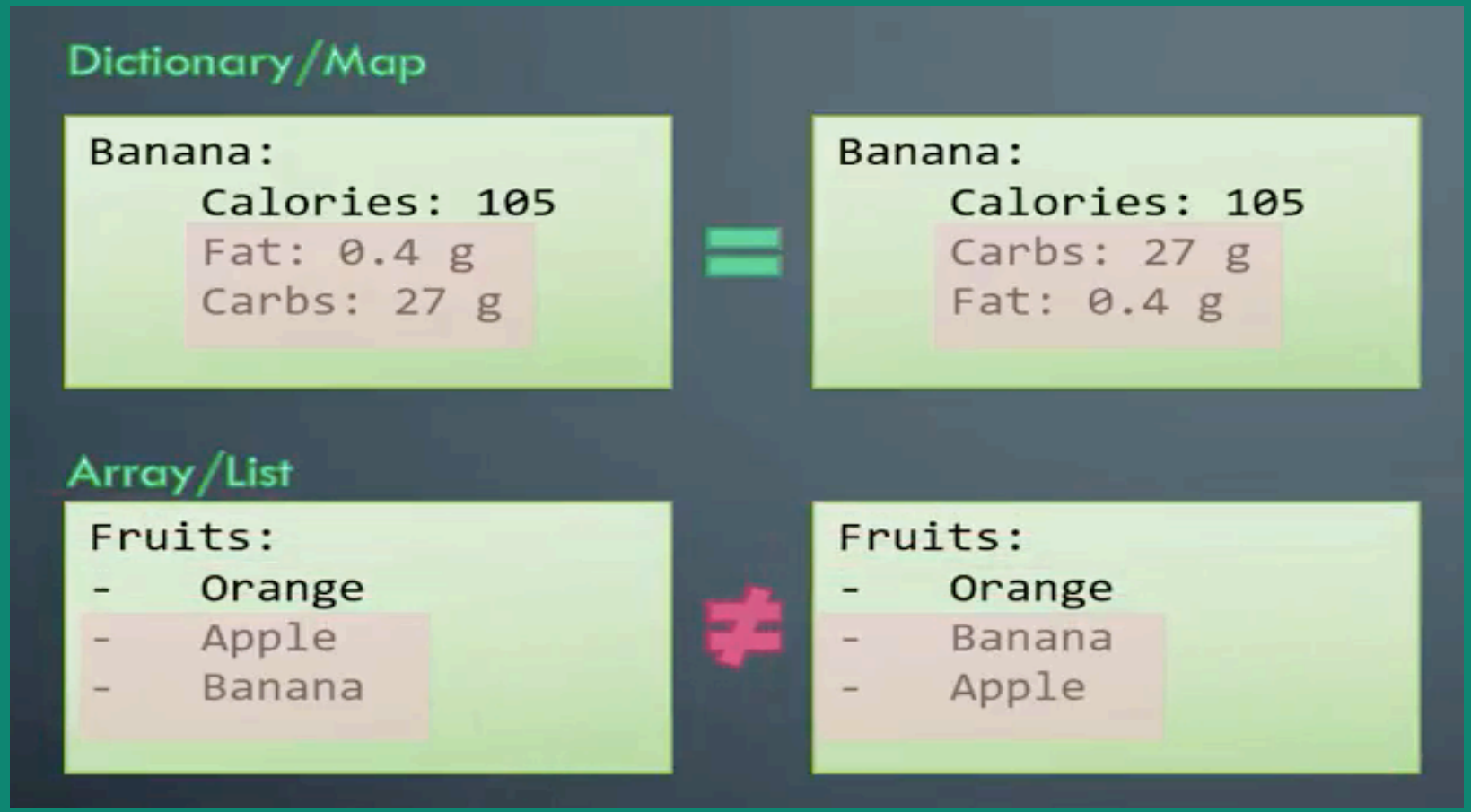
NEVER use tabs in a YAML file



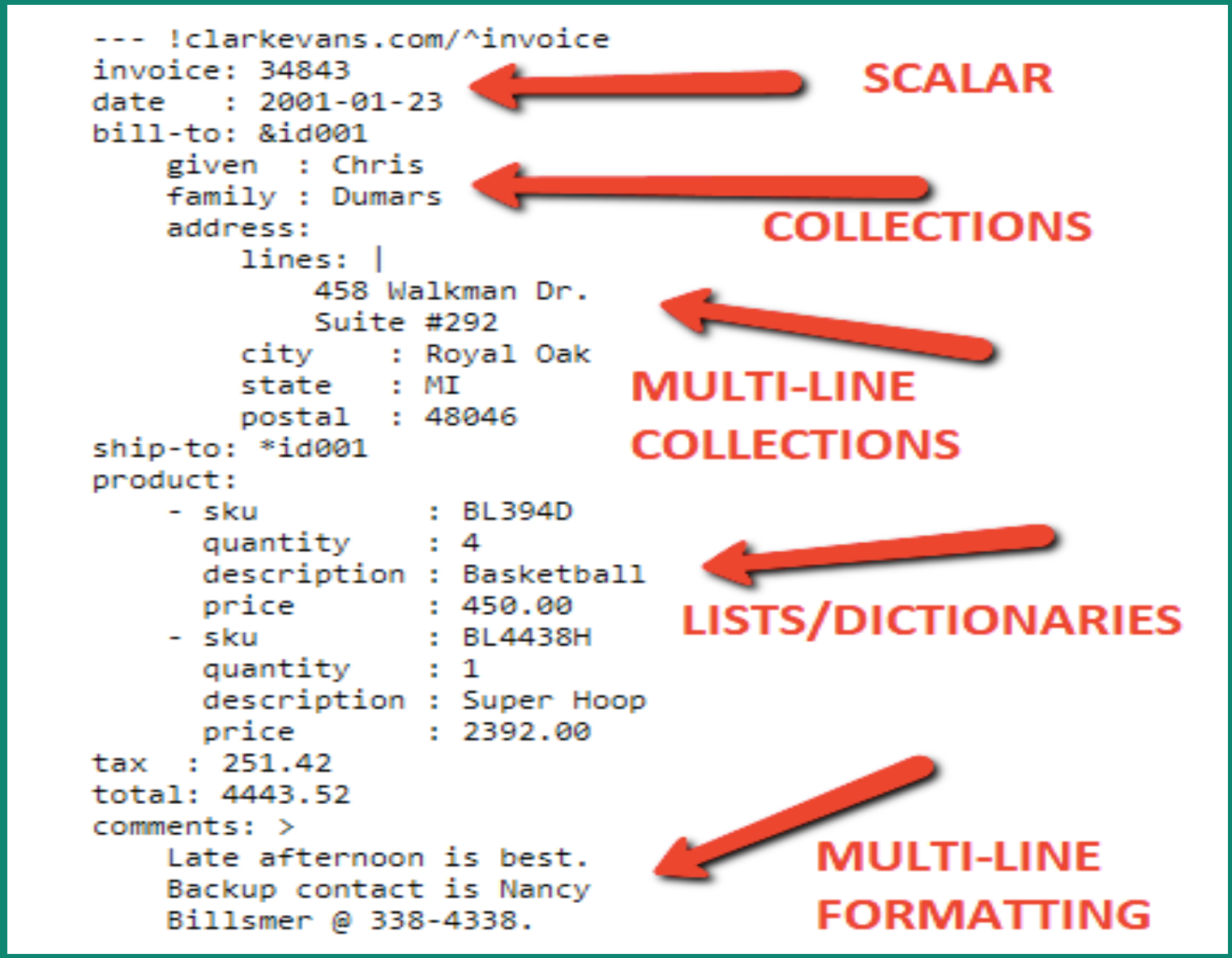
Complex YAML structures



Complex YAML structures



YAML Basics in one slide



YAML Validator/Formatter

https://jsonformatter.org/yaml-formatter

YAML Formatter

Sample

```yaml
# Default values for lets-chat.
# This is a YAML-formatted file.
# Declare variables to be passed into your
    templates.
replicaCount: 1
image:
  repository: sdelements/lets-chat
  tag: latest
  pullPolicy: IfNotPresent
service:
  name: lets-chat
  tier: frontend
  type: NodePort
  protocol: TCP
  internalPort: 8080
  externalPort: 8080
  nodeport: 30090
mongo:
  host: mongo
resources:
  limits:
    memory: 512Mi
  requests:
    memory: 128Mi
```

Load Data

Format / Beautify

Download

Formatted YAML

```yaml
replicaCount: 1
image:
    repository: sdelements/lets-chat
    tag: latest
    pullPolicy: IfNotPresent
service:
    name: lets-chat
    tier: frontend
    type: NodePort
    protocol: TCP
    internalPort: 8080
    externalPort: 8080
    nodeport: 30090
mongo:
    host: mongo
resources:
    limits:
        memory: 512Mi
    requests:
        memory: 128Mi
```

YAML Conversion to JSON

https://www.json2yaml.com/convert-yaml-to-json

https://yaml-online-parser.appspot.com/

YAML ✓ Copy YAML

```yaml
---
# <- yaml supports comments, json does not
# did you know you can embed json in yaml?
# try uncommenting the next line
# { foo: 'bar' }

json:
  - rigid
  - better for data interchange
yaml:
  - slim and flexible
  - better for configuration
object:
  key: value
  array:
    - null_value:
    - boolean: true
    - integer: 1
paragraph: >
    Blank lines denote

    paragraph breaks
content: |-
    Or we
    can auto
    convert line breaks
    to save space
```

JSON Copy JS

```json
{
   "json": [
      "rigid",
      "better for data interchange"
   ],
   "yaml": [
      "slim and flexible",
      "better for configuration"
   ],
   "object": {
      "key": "value",
      "array": [
         {
            "null_value": null
         },
         {
            "boolean": true
         },
         {
            "integer": 1
         }
      ]
   },
   "paragraph": "Blank lines denote\nparagraph breaks\n",
   "content": "Or we\ncan auto\nconvert line breaks\nto save space"
}
```

Properties file vs YAML

application.properties

```
spring.datasource.url = jdbc:mysql://mysql-st
spring.datasource.username = sa
spring.datasource.password = password

# Keep the connection alive if idle for a lon
spring.datasource.testWhileIdle = true
spring.datasource.validationQuery = SELECT 1

# Show or not log for each sql query
spring.jpa.show-sql = true

# Hibernate ddl auto (create, create-drop, up
spring.jpa.hibernate.ddl-auto = update

# Naming strategy
spring.jpa.hibernate.naming-strategy = org.hi

# Use spring.jpa.properties.* for Hibernate n
# stripped before adding them to the entity m

# The SQL dialect makes Hibernate generate be
spring.jpa.properties.hibernate.dialect = org

server.port=8086
```

application.yml

```
spring:
  datasource:
    url: jdbc:mysql://mysql-standalone:3306/t
    username: sa
    password: password
    testWhileIdle: true
    validationQuery: select 1
  jpa:
    show-sql: true
    hibernate:
      ddl-auto: update
      naming-strategy: org.hibernate.cfg.Impr
    properties:
      hibernate:
        dialect: org.hibernate.dialect.MySQL5


server:
  port: 8086
```

YAML Applied

Usage in Kubernetes: Web POD creation

```yaml
apiVersion: "v1"
kind: Pod
metadata:
  name: web
  labels:
    app: web
spec:
  containers:
    - name: web
      image: mhbauer/webapp
      ports:
        - containerPort: 5000
          protocol: TCP
```

Usage in Kubernetes: DB POD creation

```yaml
apiVersion: "v1"
kind: Pod
metadata:
  name: redis
  labels:
    app: db
spec:
  containers:
    - name: redis
      image: redis
      ports:
        - containerPort: 6379
```

Usage in Kubernetes: Service creation

```yaml
apiVersion: v1
kind: Service
metadata:
  name: web
  labels:
    app: web
spec:
  selector:
    app: web
  type: NodePort
  ports:
   - port: 5000
     nodePort: 31000
```

Usage in Kubernetes: Service creation

```yaml
apiVersion: v1
kind: Service
metadata:
  name: redis
  labels:
    app: db
spec:
  selector:
    app: db
  ports:
  - port: 6379
```

Usage in Kubernetes: Deployment creation

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: web
spec:
  replicas: 2
  template:
    metadata:
      labels:
        app: web
        version: v1
    spec:
      containers:
      - name: web
        image:          /webapp
        imagePullPolicy: Always
        ports:
        - containerPort: 5000
```

Kubernetes deployment: Basic specification

Tag	Description
apiVersion	Describes the versioned schema of the deployment object defined by the yaml specification file
Kind	Describes the resource defined by the yaml specification file. Kubernetes can infer kind based on the endpoint to which the specification is submitted
metadata.name	Deployments are uniquely identified within a namespace by their name.
metadata.namespace	Namespaces allow you to create separations within your cluster. Typically, objects can only communicate with other objects if they share a namespace. If the namespace field is omitted from your specification, the namespace 'default' is used
spec	The spec object is the real meat of the deployment specification yaml. It is where you define the actual desired behavior of the Kubernetes Deployment.
spec.template	Describes the Pods Kubernetes should create as part of the deployment.
spec.template.spec	Provides the explicit instructions for the desired behavior of a Pod
spec.template.spec.containers	An array of objects that must include at least 1 container. Each of the containers in this array describes the behavior of a containerized application within a Pod
spec.template.spec.containers[i].name	Each container within a Pod must have a unique name
spec.template.spec.containers[i].image	Describes which Docker image should run in the deployed container.

Kubernetes deployment: Advanced specification

How to control how many Pods are created in a deployment?

```yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  template:
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
```

Kubernetes deployment: Advanced specification

How to control the strategy with which Kubernetes replaces existing Pods?

```yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 1
      maxSurge: 1
  replicas: 3
  template:
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
```

```yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  strategy:
    type: Recreate
  replicas: 3
  template:
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
```

Kubernetes deployment: Advanced specification

How to dictate when a Pod is available after an update?

```yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  minReadySeconds: 10
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 1
      maxSurge: 1
  replicas: 3
  template:
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
```

Kubernetes deployment: Advanced specification

How to control how many revisions of a given deployment are kept by Kubernetes?

```yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  revisionHistoryLimit: 5
  minReadySeconds: 10
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 1
      maxSurge: 1
  replicas: 3
  template:
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
```

Kubernetes deployment: Advanced specification

How to add labels to my Kubernetes deployment specification?

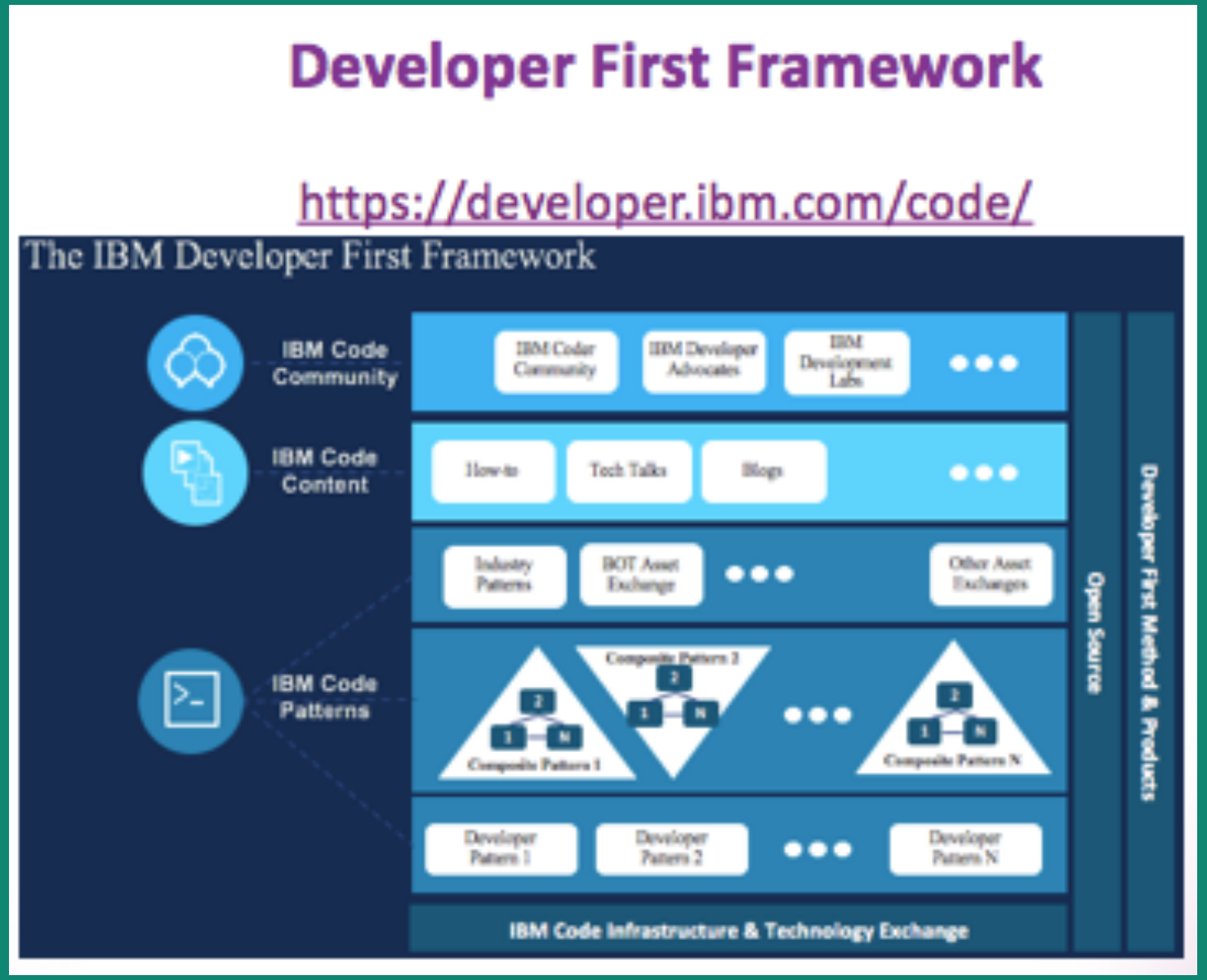
```yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  revisionHistoryLimit: 5
  minReadySeconds: 10
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 1
      maxSurge: 1
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
        deployer: distelli
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
```

Kubernetes deployment: Advanced specification

Use label selectors in a deployment specification to tell my deployment to only target pods with certain labels?

```yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  revisionHistoryLimit: 5
  minReadySeconds: 10
  selector:
    matchLabels:
      app: nginx
      deployer: distelli
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 1
      maxSurge: 1
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
        deployer: distelli
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
```

IBM Code


Developer First Framework
https://developer.ibm.com/code/
The IBM Developer First Framework
IBM Code Community
IBM Code Content
IBM Code Patterns
IBM Coder Community
IBM Developer Advocates
IBM Development Labs
How-to
Tech Talks
Blogs
Industry Patterns
BOT Asset Exchange
Other Asset Exchanges
Composite Pattern 1
Composite Pattern 2
Composite Pattern N
Developer Pattern 1
Developer Pattern 2
Developer Pattern N
IBM Code Infrastructure & Technology Exchange
Open Source
Developer First Method & Products

YAML : Code Pattern

Usage of YAML: Code Pattern

- https://developer.ibm.com/code/patterns/deploy-spring-boot-microservices-on-kubernetes/

- Spring Boot is an opinionated framework for quickly building production-ready Spring applications.

- This journey shows you how to create and deploy Spring Boot microservices within a polyglot application and then deploy the app to a Kubernetes cluster. It has heavy usage of YAML

Lab: YAML : Deploying to IBM Cloud Kubernetes Cluster

Kubernetes @ IBM

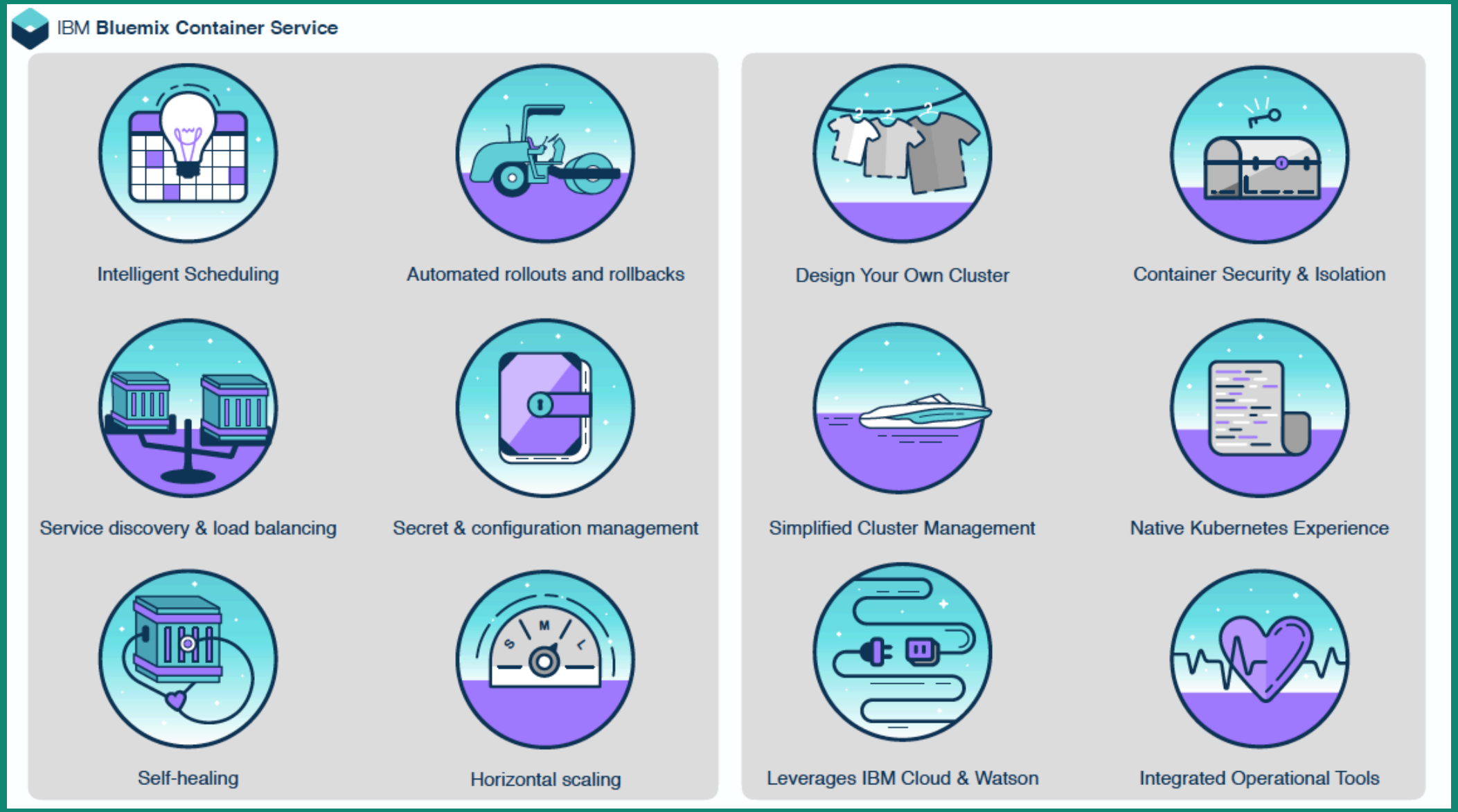
- Offerings / Plans
  - IBM Cloud Container Service – Docker containers orchestrated by K8s
  - ICP – IBM Cloud Private

IBM Bluemix Container Service

- Combining Docker and Kubernetes to deliver powerful tools,
  - An intuitive user experience, and built-in security and
  - Isolation to enable rapid delivery of applications - all while
  - Leveraging IBM Cloud Services including cognitive
  - Capabilities from Watson.

www.ibm.com/cloud-computing/bluemix/containers

IBM Bluemix Container Service



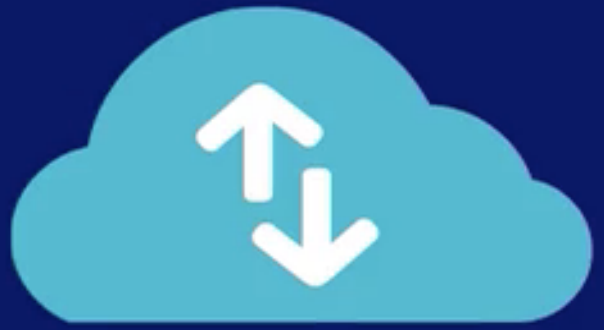
ICP – IBM Cloud Private

Agenda

1. Introduction to ICP
2. ICP – Architectural Components
3. Use cases
4. Catalogue - Helm : Package Manger
5. Demo : Deploy an Application

Private Cloud

Benefits of Public Cloud



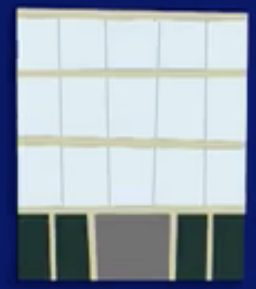
rapid deployment

scalability

ease of use

elasticity

Benefits of On – Premise



control

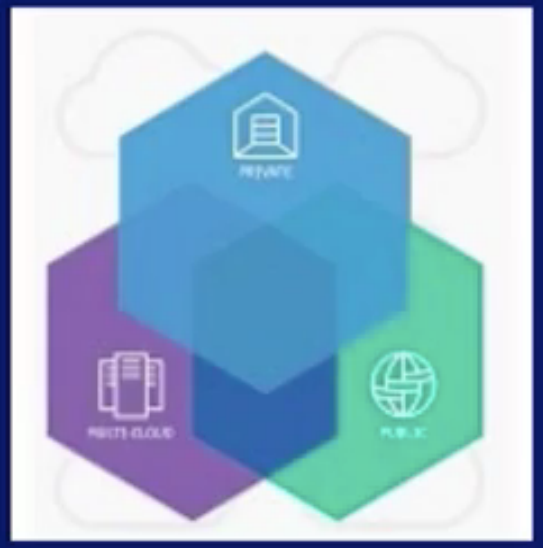
performance

predictable cost

security

flexible management

IBM Private Cloud



Kubernetes

Private image repository

Management console

Monitoring frameworks

Kubernetes + Docker + Helm

Based on open source platform

Key Aspects of IBM Private Cloud

1. Elastic runtime for enterprise developers
2. Cloud Ready IBM Middleware
3. Production ready operations facility
4. Built in support for Continuous Delivery

IBM Cloud Private - Editions - Solutions for enterprise use cases

Community

Use Case
Create cloud-native applications in a non-production environment

Platform
- Kubernetes
- Core services (Security, logging, monitoring, etc)
- Catalog of containerized content

Freely available in Docker Store

The Community Edition is limited to 1 Master Node, and is for non-production use.

Cloud Native

Use Case
Create cloud-native applications in a production environment

Platform
- Kubernetes
- Core services (Security, logging, monitoring, etc)
- Catalog of containerized content

Cloud Foundry (Optional add-on)

IBM Enterprise Software
- Microservice Builder
- WebSphere Liberty
- IBM SDK for node.js
- Cloud Automation Manager

Enterprise

Use Case
- Modernize and optimize existing applications
- Open enterprise data centers

Platform
- Kubernetes
- Core services (Security, logging, monitoring, etc)
- Catalog of containerized content

Cloud Foundry (Optional add-on)

IBM Enterprise Software
- Everything in Cloud Native, plus:

\+ WAS ND
\+ MQ Advanced
\+ API Connect Professional
\+ Db2 Direct Advanced (separate PN)
\+ UrbanCode Deploy (separate PN)

ICP – Editions – More...

Community

Open Source*

Toolchains & Runtimes
Jenkins
Apache Tomcat
Open Liberty

Messaging
RabbitMQ

Data Services
MongoDB
PostgreSQL
Redis

Clustering
Galera

Http Servers
Nginx

Terminal Access
Web Terminal

IBM Software

Toolchain & Runtimes
IBM Microservice Builder
IBM WebSphere Liberty for Developers
IBM SDK for Node.js

Messaging
MQ Advanced for Developers

Data Services
IBM Db2 Dev-C
IBM Db2 Warehouse Dev-C
IBM Data Server Manager (for Db2 Dev-C)
IBM Cloudant Developer Edition

Data Science
IBM Data Science Experience Developer Ed.

Integration
IBM Integration Bus for Developers
IBM DataPower Gateway for Developers

App Modernization Tooling
IBM Transformation Advisor

Monitoring
IBM Cloud APM for DevOps (Beta)

HPC
IBM Spectrum LSF Community Edition

**Open source is not warranted by IBM unless otherwise specified*

Cloud Native

Access to Community content, plus...

Toolchain & Runtimes
IBM Microservice Builder
IBM WebSphere Liberty
IBM SDK for Node.js

Multi-Cloud Management
IBM Cloud Automation Manager

Cloud Foundry Buildpacks (add-on)
IBM WebSphere Liberty,
Node.js, Swift, .Net

Enterprise

All Cloud Native content, plus...

Toolchain & Runtimes
IBM WebSphere Application Server ND [1]
IBM UrbanCode Deploy (add-on charge) [1]

Integration and Messaging
IBM MQ Advanced
IBM API Connect [1]

Data Services
IBM Db2 Direct Advanced Edition with
Data Server Manager (add-on charge)

Available a la carte

Add to Cloud Native or Enterprise, or bring your existing license

Messaging
IBM MQ Advanced

Integration
IBM Integration Bus
IBM DataPower Gateway Virtual Edition

Data Services
IBM Db2 Direct Advanced Edition
/ AESE, with Data Server Manager
IBM Data Science Experience Local*
IBM Db2 Warehouse Enterprise*

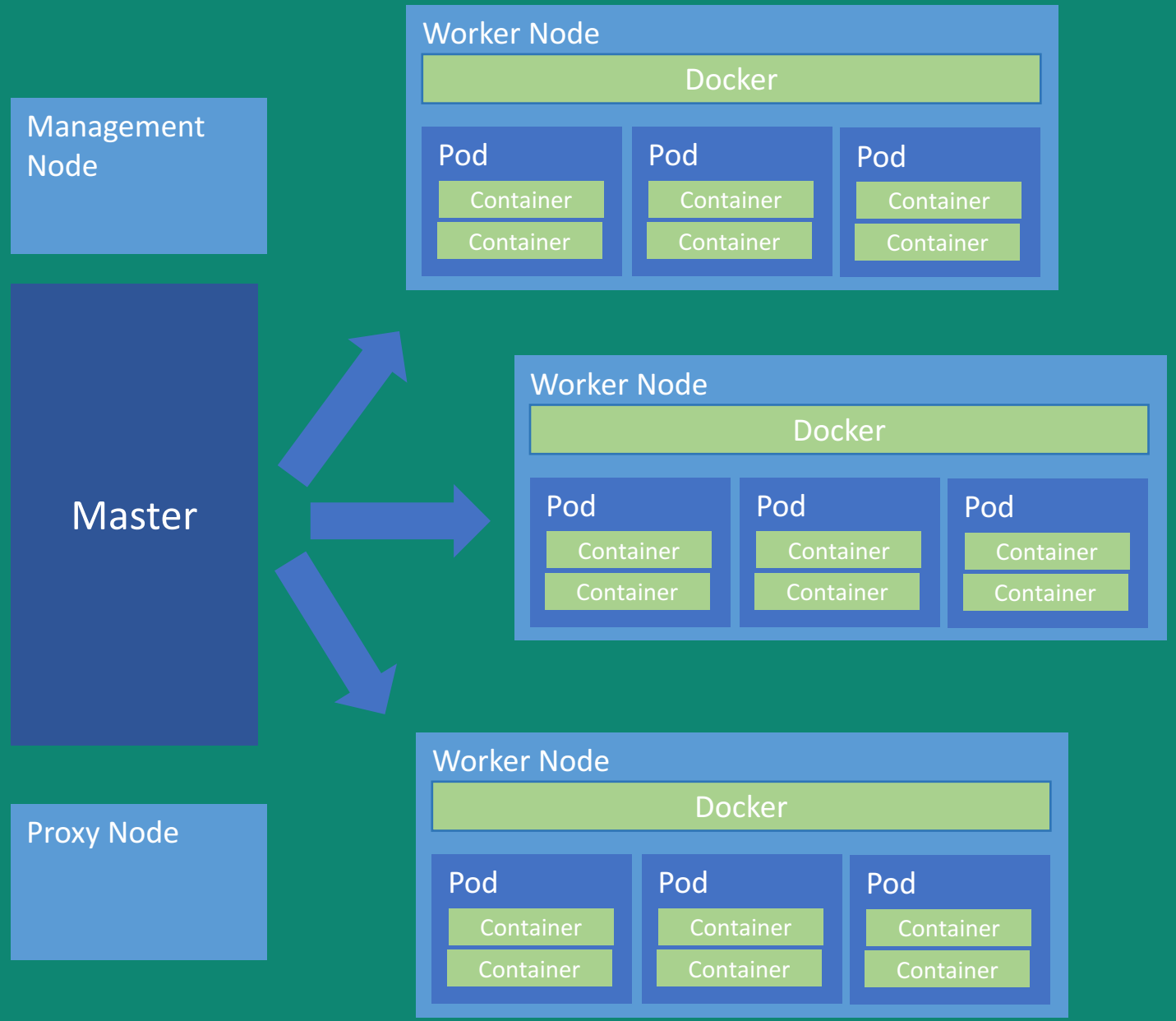
*[1] Runs on VMs | *coming soon*

Benefits of ICP

- Choice of Infrastructure
  - existing infrastructure, OpenStack, VMware or a combination of these.
- Choice of Compute Model
  - including infrastructure as code or a programmable infrastructure, Kubernetes, or Cloud Foundry
- Multitenant Container Orchestration
  - With K8s, you get multi-tenant container orchestration for deploying and managing microservices applications
- Application Isolation and Security
  - Network and storage policy-based controls for application isolation and security
- Auto-scaling and Self Healing
  - automatic health checking and failover
- Integrated DevOps

Configuration topologies

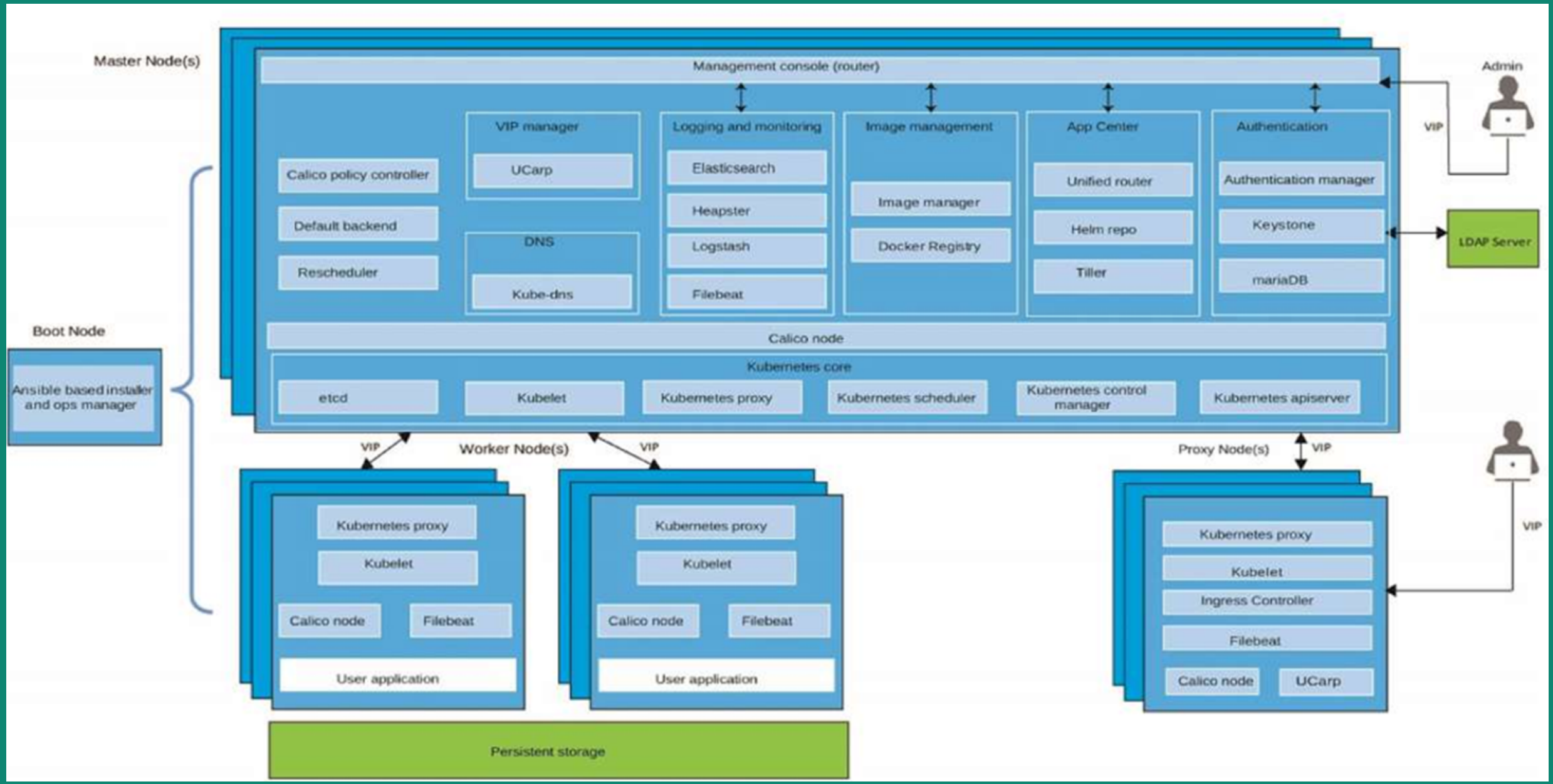
- Simple
  - Single machine install (master is a worker)
  - Great for testing and learning about the platform
- Standard
  - Single master (single master, 3 workers, 1 proxy)
  - Great for non-production testing environment
- High Availability
  - Multiple masters (3 masters, 3+ workers, 3 proxy)
  - Production installation



ICP – types of nodes

- **Boot node:** A boot or bootstrap node is used for running installation, configuration, node scaling, and cluster updates. Only one boot node is required for any cluster. You can use a single node for both master and boot.
- **Master node:** A master node provides management services and controls the worker nodes in a cluster. Master nodes host processes that are responsible for resource allocation, state maintenance, scheduling, and monitoring. Multiple master nodes are in a high availability (HA) environment to allow for failover if the leading master host fails. Host that can act as the master are called master candidates.
- **Worker node:** A worker node is a node that provides a containerized environment for running tasks. As demands increase, more worker nodes can easily be added to your cluster to improve performance and efficiency. A cluster can contain any number of worker nodes, but a minimum of one worker node is required.
- **Proxy Node:** A proxy node is a node that transmits external request to the services created inside your cluster. Multiple proxy nodes are deployed in a high availability (HA) environment to allow for failover if the leading proxy host fails. While you can use a single node as both master and proxy, it is best to use dedicated proxy nodes to reduce the load on the master node. A cluster must contain at least one proxy node if load balancing is required inside the cluster.

IBM Cloud Private – architecture components

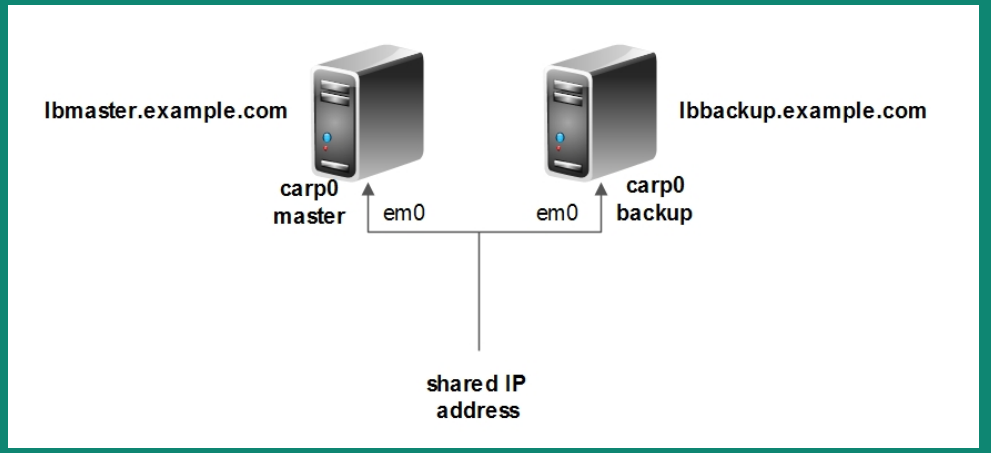


K8s master components

- *Master components provide the cluster's control plane and global decisions about the cluster and detecting and responding to cluster events*
- **Etcd:** A strong, consistent, and highly-available key value store which Kubernetes uses for persistent storage of all of its API objects.
- **Kubelet:** The primary "node agent" that runs on each node.
- **K8s Proxy:** The Kubernetes network proxy runs on each node.
- **K8s Scheduler:** A policy-rich, topology-aware, workload-specific function that significantly impacts availability, performance, and capacity.
- **K8s Control Manager:** A daemon that embeds the core control loops shipped with K8s. In K8s, a controller is a control loop that watches the shared state of the cluster through the apiserver and makes changes attempting to move the current state towards the desired state.
- **K8s apiserver:** Validates and configures data for the api objects which include pods, services, replication controllers, and others. The API Server services REST operations and provides the frontend to the cluster's shared state through which all other components interact.

Network services components

- **DNS:** (kube-dns, Cluster DNS) K8s DNS schedules a DNS pod and service on the cluster, and configures the kubelets to tell individual containers to use the DNS service's IP to resolve DNS names. Every service defined in the cluster (including the DNS server itself) is assigned a DNS name. By default, a client pod's DNS search list will include the pod's own namespace and the cluster's default domain.
- **VIP and UCarp:** UCarp allows a couple of hosts to share common virtual IP (or floating IP) addresses in order to provide automatic failover.



Calico

A new approach to virtual networking and network security for containers, VMs, and bare metal services, that provides a rich set of security enforcement capabilities running on top of a highly scalable and efficient virtual network

- The calico/node Docker container runs on the Kubernetes master and each Kubernetes node in the cluster
- The calico-cni plug-in integrates directly with the Kubernetes kubelet process on each node to discover which pods have been created, and adds them to Calico networking
- The calico/kube-policy-controller container runs as a pod on top of Kubernetes and implements the NetworkPolicy API

PROJECT CALICO

Ingress resources

- **Ingress:** Typically, services and pods have IPs only routable by the cluster network. All traffic that ends up at an edge router is either dropped or forwarded elsewhere.
  - An Ingress is a collection of rules that allow inbound connections to reach the cluster services.
  - It can be configured to give services externally-reachable URLs, load balance traffic, terminate SSL, offer name based virtual hosting etc.
  - Users request ingress by POSTing the Ingress resource to the API server

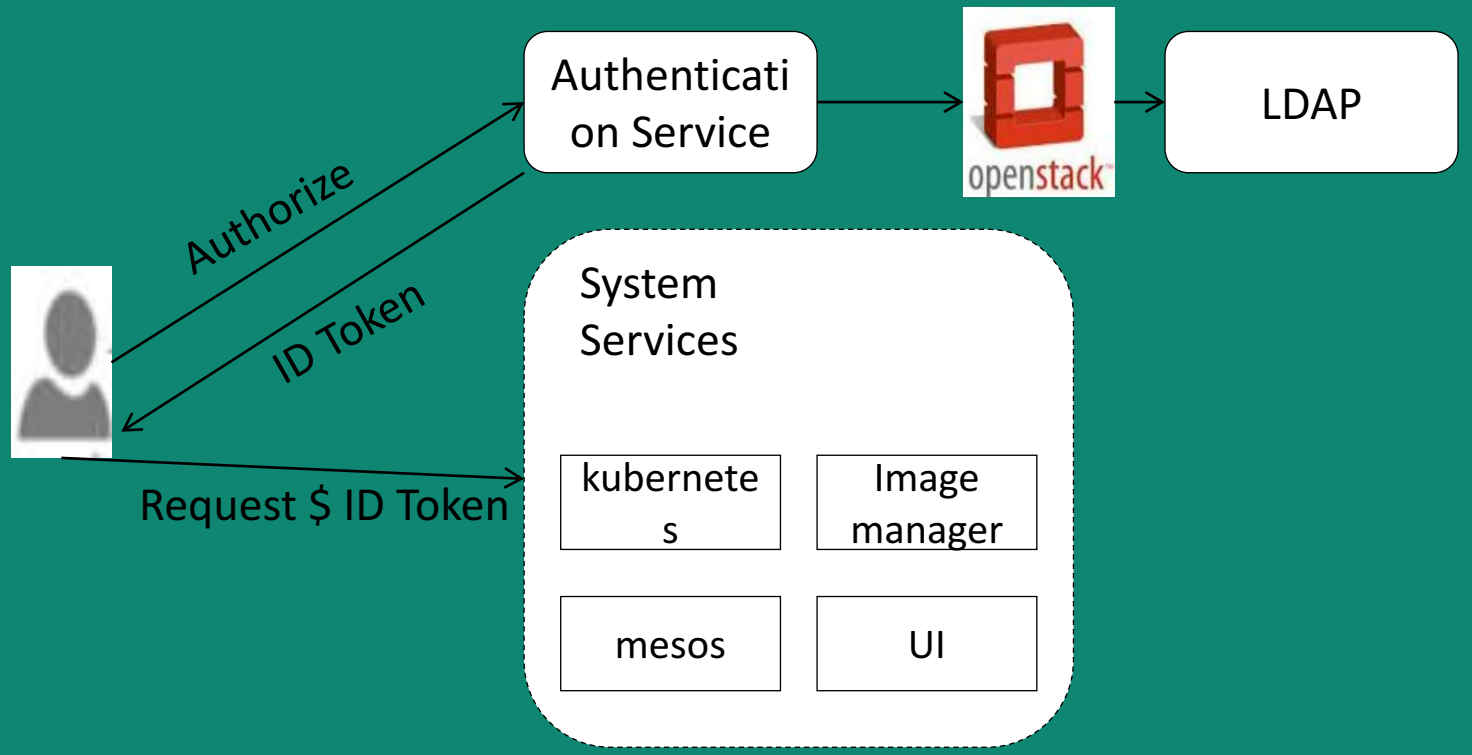
- **Ingress Controller:** Responsible for fulfilling the Ingress, usually with a load balancer, though it may also configure your edge router or additional frontends to help handle the traffic in an HA manner.

Authentication components

Authentication Manager: Provides an HTTP API for managing users. Protocols are implemented in a RESTful manner. Keystone is used for authentication. Pass-through is used for external LDAP integration.

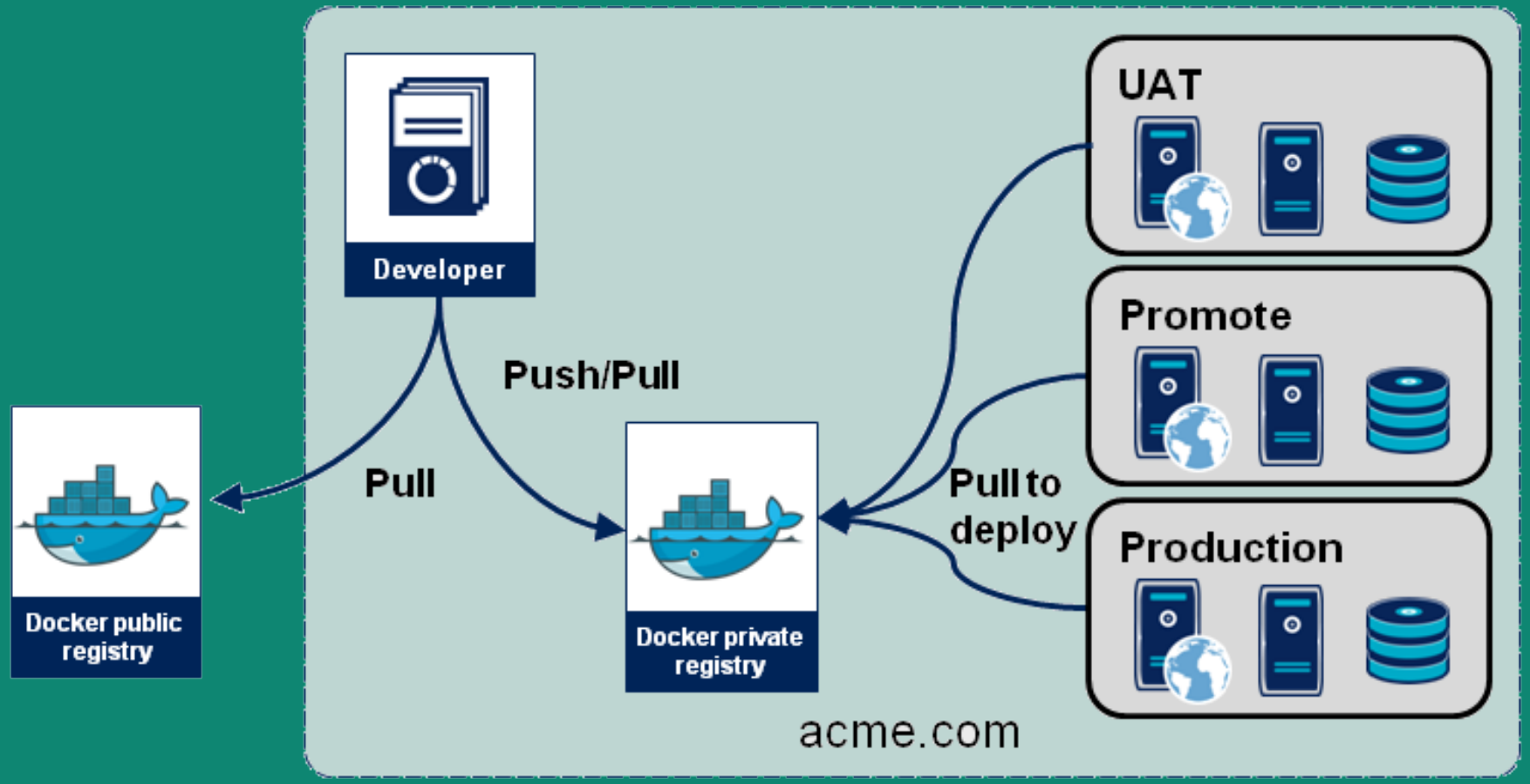
Keystone: The OpenStack provided identity service currently supporting token-based authN and user-service authorization.

MariaDB: An open source relational database made by the original developers of MySQL. In this case it is used to back-end Keystone.



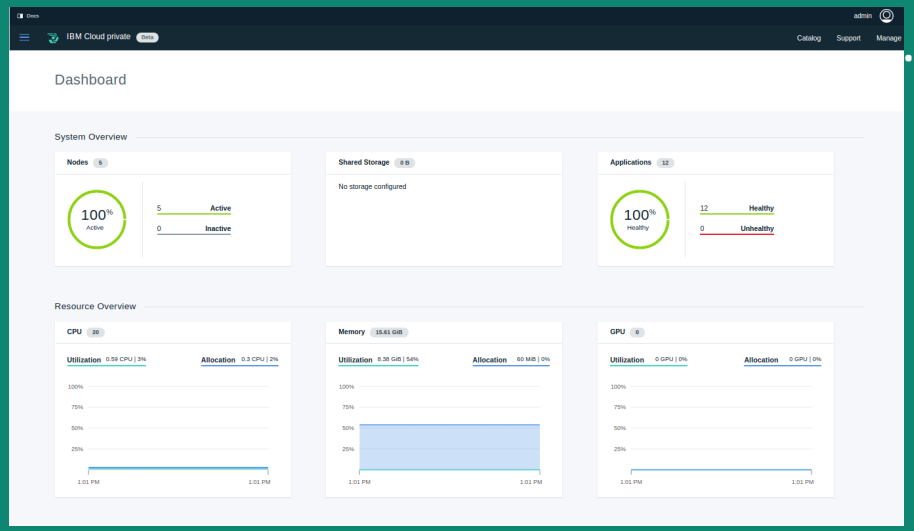
Images and Registries

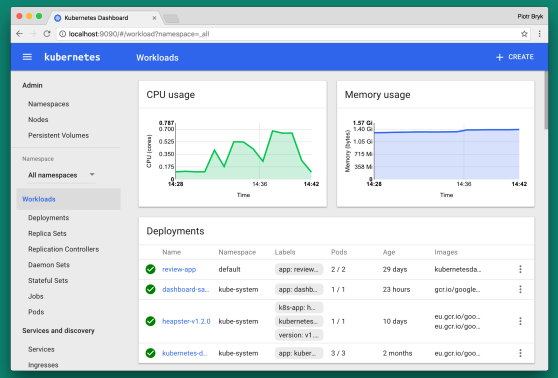
- You create a Docker image and push it to a registry before referring to it in a Kubernetes pod
- There will likely be many registries used in your deployment



User Interfaces

- **Cluster Management Console:** (ICP component) Use to manage, monitor, and troubleshoot your applications and cluster from a single, centralized, and secure management console.
- **K8s Web UI:** Can use to deploy containerized applications to a Kubernetes cluster, troubleshoot your containerized application, and manage the cluster itself along with its attendant resources.
- **kubectl:** A command-line interface for running commands against Kubernetes clusters.



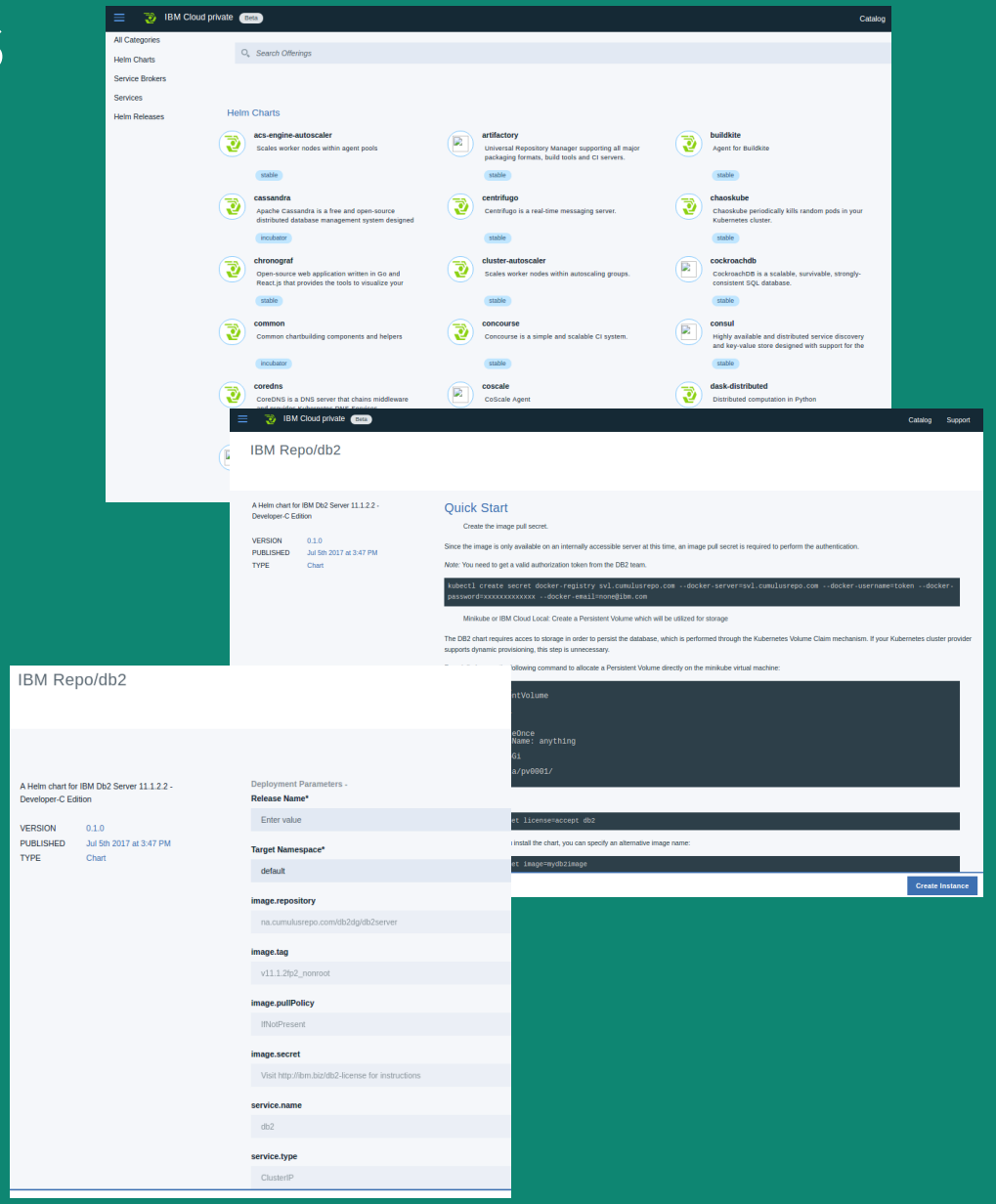



```
[root@ip-172-31-56-194 kubernetes]# export PATH=/home/ec2-user/kubernetes/platforms/linux/amd64:$PATH
[root@ip-172-31-56-194 kubernetes]# kubectl get nodes
NAME                          LABELS                                                    STATUS    AGE
ip-172-20-0-138.ec2.internal   kubernetes.io/hostname=ip-172-20-0-138.ec2.internal   Ready     3m
ip-172-20-0-25.ec2.internal    kubernetes.io/hostname=ip-172-20-0-25.ec2.internal    Ready     20m
[root@ip-172-31-56-194 kubernetes]# kubectl run ttnd-nginx --image=nginx
replicationcontroller "ttnd-nginx" created
[root@ip-172-31-56-194 kubernetes]# kubectl get pods
NAME              READY     STATUS    RESTARTS   AGE
ttnd-nginx-wp2y9   0/1       Pending   0          8s
[root@ip-172-31-56-194 kubernetes]# kubectl get pods --namespace=kube-system
NAME                                                  READY     STATUS    RESTARTS   AGE
elasticsearch-logging-v1-mcd2q                        1/1       Running   0          23m
elasticsearch-logging-v1-mmqqm                        1/1       Running   0          23m
fluentd-elasticsearch-ip-172-20-0-138.ec2.internal    1/1       Running   0          3m
fluentd-elasticsearch-ip-172-20-0-25.ec2.internal     1/1       Running   0          20m
heapster-v10-eco91                                    1/1       Running   0          23m
kibana-logging-v1-slamn                               1/1       Running   0          23m
kube-dns-v9-7pe5g                                     4/4       Running   0          23m
kube-ui-v2-0573n                                      1/1       Running   0          23m
monitoring-influxdb-grafana-v2-mgzvo                  2/2       Running   0          23m
[root@ip-172-31-56-194 kubernetes]#
[root@ip-172-31-56-194 kubernetes]#
[root@ip-172-31-56-194 kubernetes]# kubectl get pods
NAME              READY     STATUS    RESTARTS   AGE
ttnd-nginx-wp2y9   1/1       Running   0          36s
```

Application Center components

Application center provides a centralized location from which you can browse, and install packages in your cluster.

- **Helm:** A tool for managing Kubernetes charts. Charts are packages of pre-configured Kubernetes resources.
- **Helm Repository:** A Helm chart repository is a location where packaged charts can be stored and shared.
- **Tiller:** Runs inside of the cluster, and manages releases (installations) of your charts.

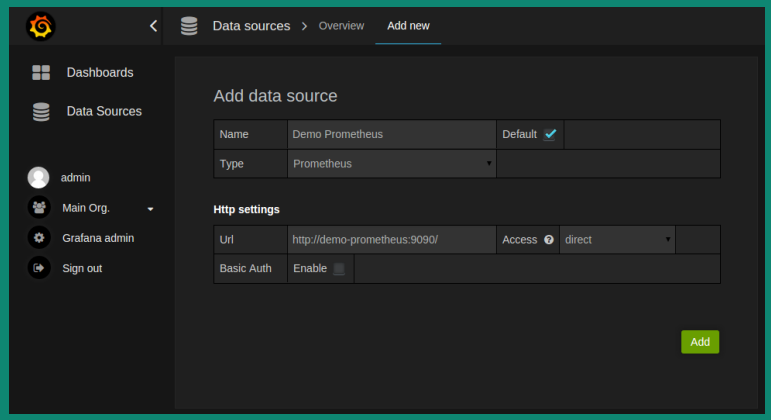


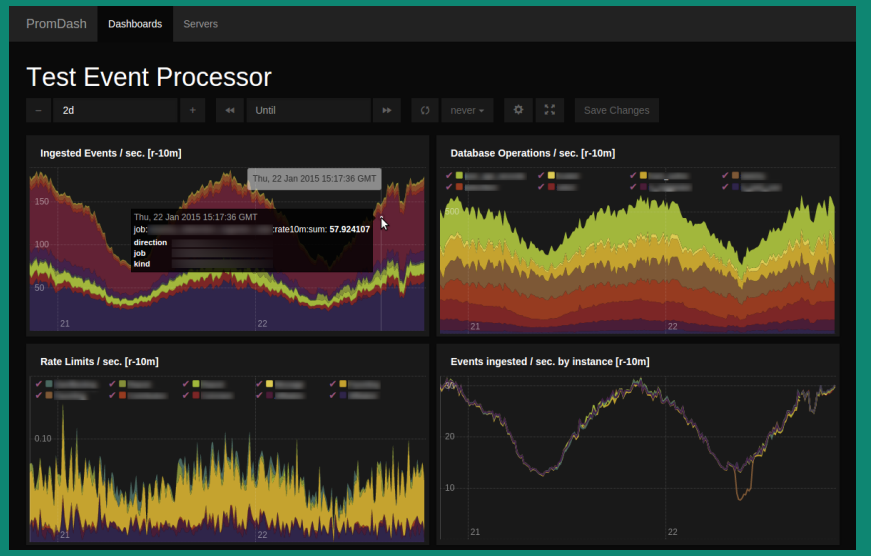
Logging components

- *The easiest and most embraced logging method for containerized applications is to write to standard out and standard error*
- **Filebeat:** A log data shipper for local files. Filebeat monitors the log directories or specific log files, tails the files, and forwards them either to Elasticsearch and/or Logstash for indexing.
- **Elasticsearch:** An open source full-text search engine based on Lucene. It provides HTTP web interface and schema-free JSON documents.
- **Logstash:** A open source tool for collecting, parsing, and storing logs for future use.
- **Heapster:** The Kubernetes network proxy runs on each node.
- **Kibana:** An open source data visualization plugin for Elasticsearch. Users can create bar, line and scatter plots, or pie charts and maps on top of large volumes of data.

Monitoring: Prometheus and Grafana

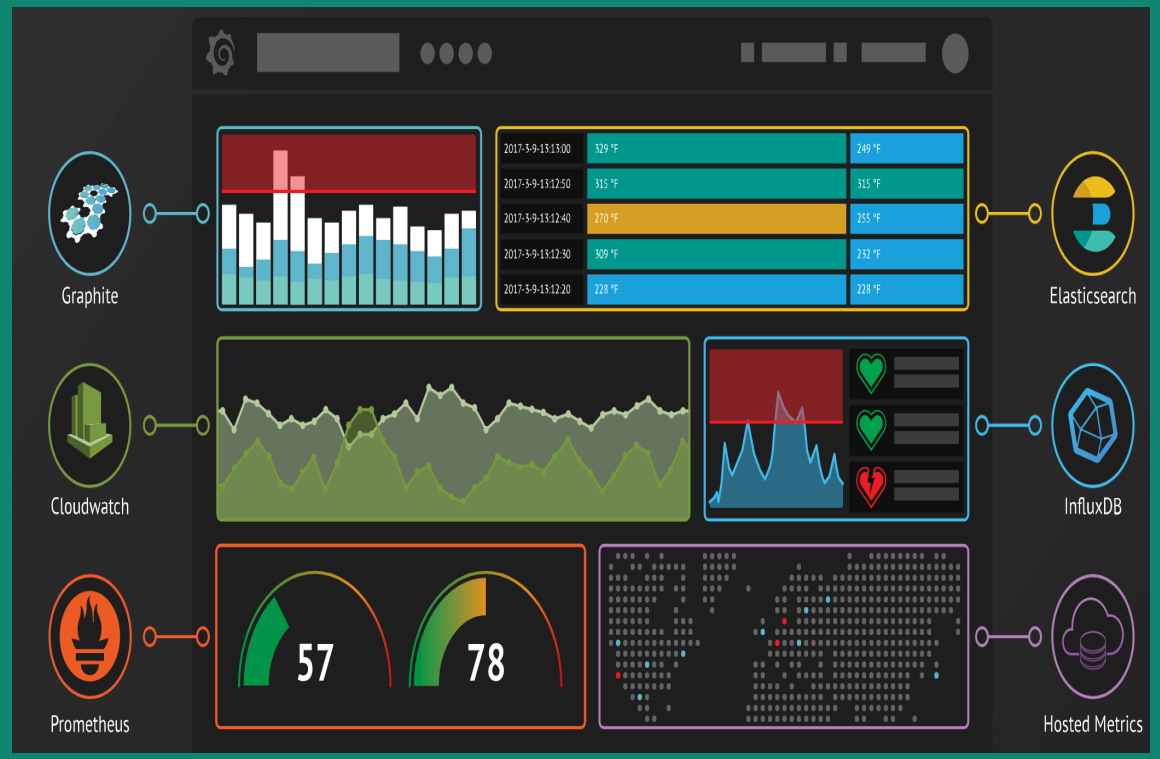
- **Prometheus:** An open-source systems monitoring and alerting toolkit originally built at SoundCloud. Since its inception in 2012, many companies and organizations have adopted Prometheus, and the project has a very active developer and user community. It is now a standalone open source project and maintained independently of any company.
- **Grafana:** An open-source, general purpose dashboard and graph composer, which runs as a web application.





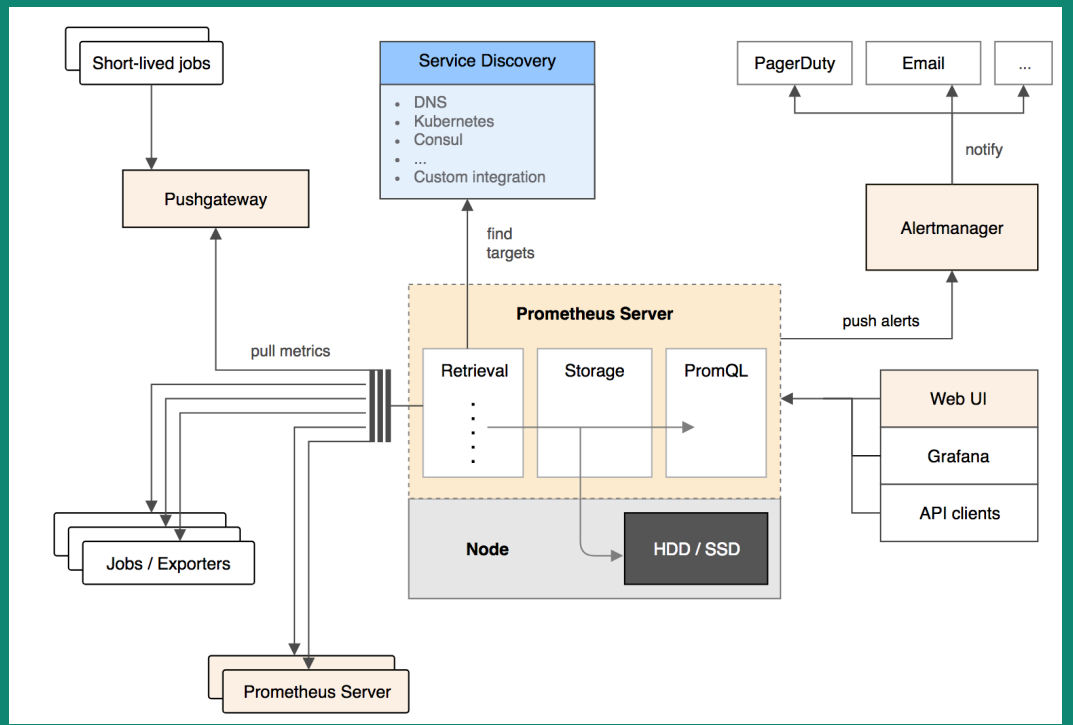
Grafana & Prometheus

Grafana : The open platform for beautiful **analytics and monitoring**



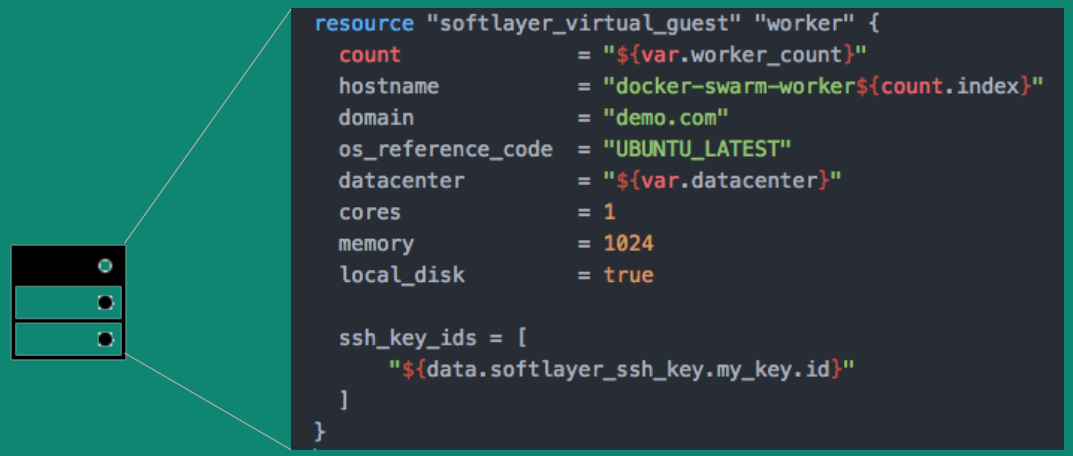
Grafana

Prometheus : is an open-source systems **monitoring and alerting** toolkit



Prometheus

Terraform



Terraform is an infrastructure as code software. It allows users to define a datacenter infrastructure in a high-level configuration language, from which it can create an execution plan to build the infrastructure in a service provider such as IBM SL , AWS, Google, MSFT as well as Open Stack & VMWare

Persistent storage components (1 of 2)

- Traditionally Containers: stateless, ephemeral in nature
  - Storage exists within the container
  - The container goes away and so goes the storage
- Some applications desire state and thus persistent storage:
  - Specific aspects of configuration
  - Database (structured and unstructured)
  - Application data (website definitions, etc.)
- Storage must be universally accessible across the K8s environment

Persistent storage components (2 of 2)

ICP Persistent Storage Support: HostPath, NFS, GlusterFS, vSphereVolume

Note: Reclaim policy and access modes and behaviors can vary

Access Modes:

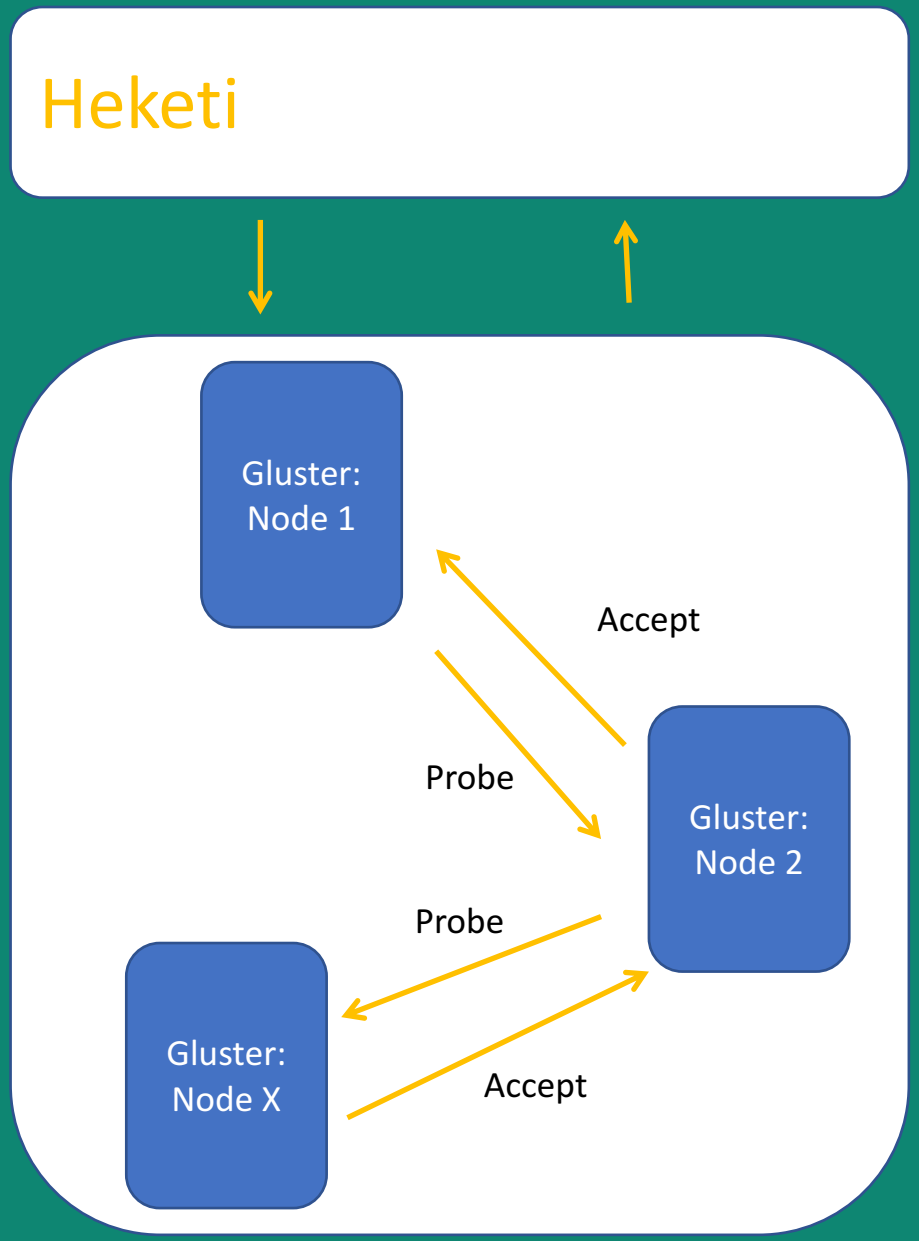
- ReadWriteOnce – the volume can be mounted as read-write by a single node
- ReadOnlyMany – the volume can be mounted read-only by many nodes
- ReadWriteMany – the volume can be mounted as read-write by many nodes

GlusterFS

- Network Attached Storage
- POSIX-Compliant distributed file-system
- Differentiator -> No central metadata server
- Aggregator of storage and metadata
- Flexible scaling for capacity and performance
- NOTE: Red Hat Gluster Storage (formerly Red Hat Storage Server) is built upon the open source GlusterFS at gluster.org

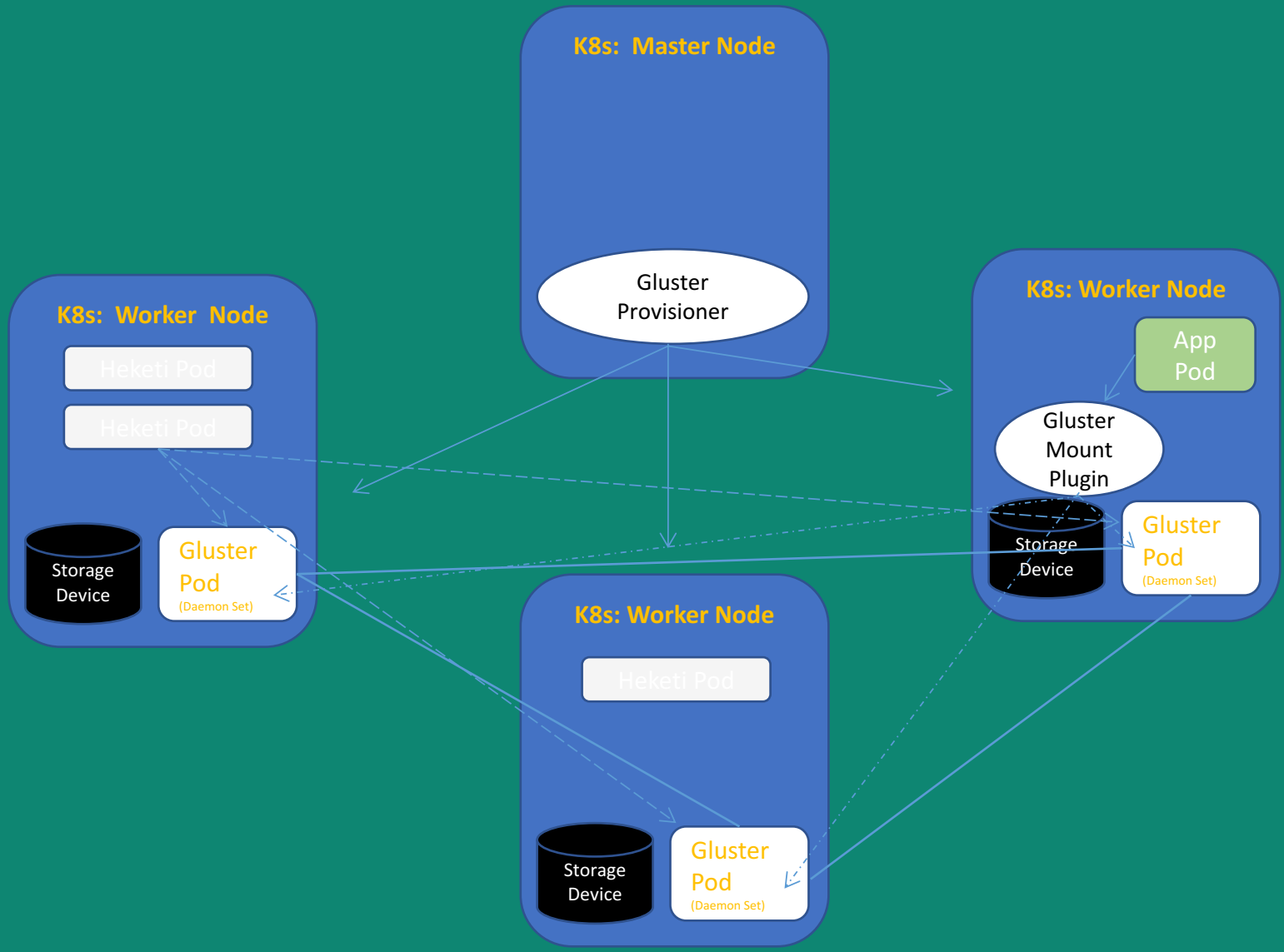
GlusterFS and Heketi

- Restful management interface for the management of GlusterFS volumes
- For Kubernetes, it provides dynamic provisioning of volumes
  - You provide nodes with storage devices
  - Heketi assembles the topology
  - Manages life-cycle of bricks, volumes, etc
  - ICP + K8s Talk to Heketi
- https://github.com/heketi/heketi



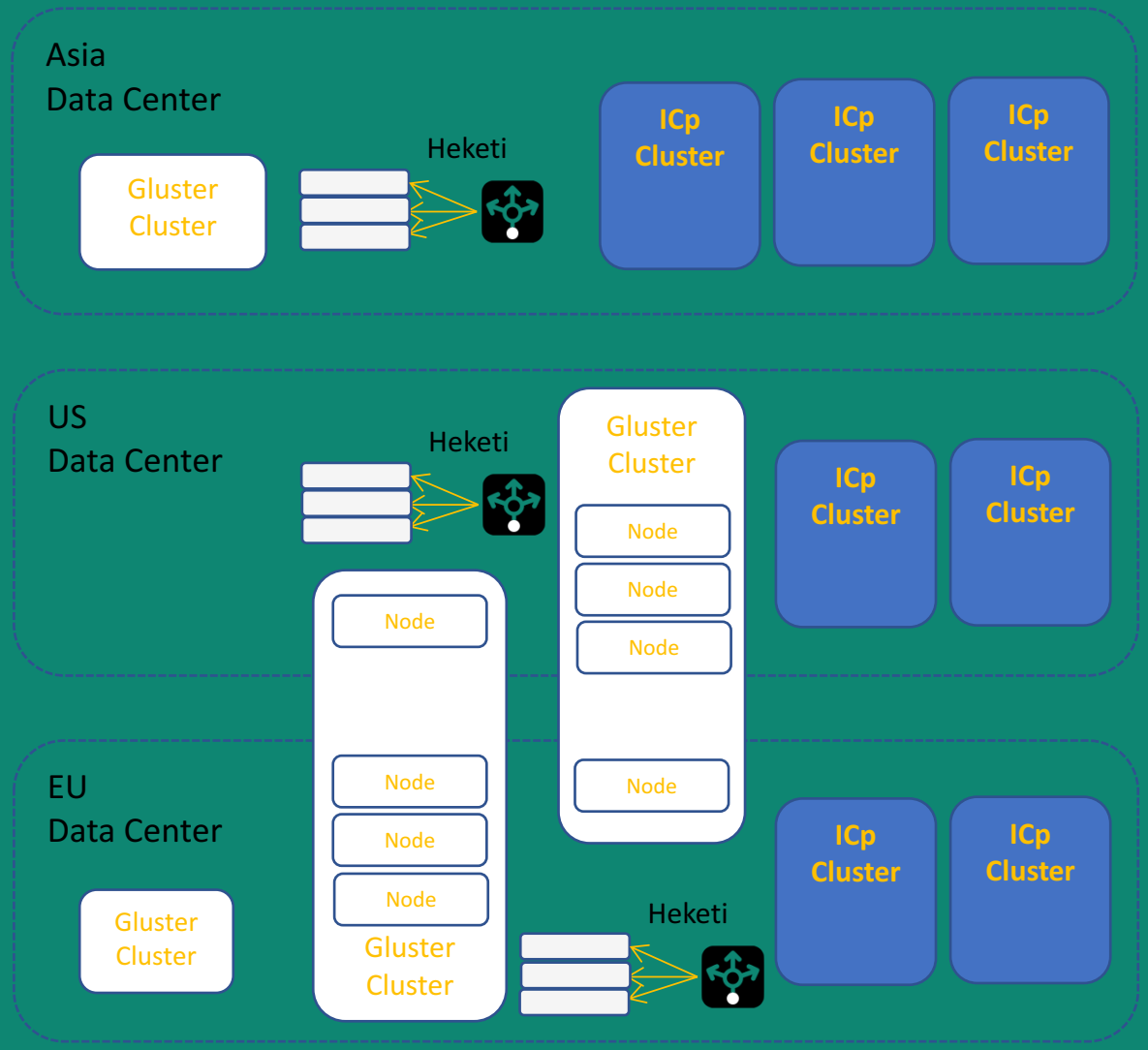
GlusterFS - common deployment architecture

- Pod-ased Components
- Gluster pods Deployed to each K8s node
- Heketi pod or pods deployed to K8s

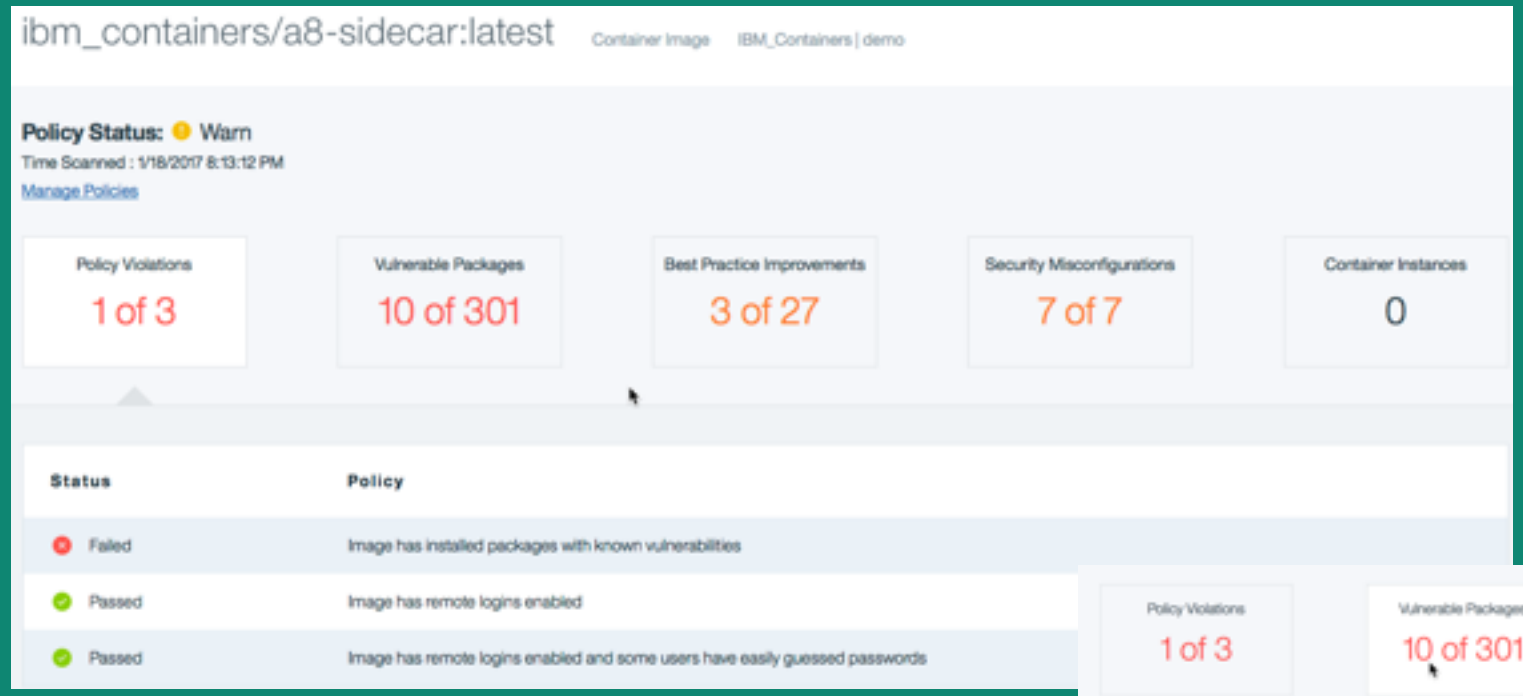


GluserFS architecture extended (example)

- Gluster cluster (trusted storage pool) serving one or more ICP cluster in the same data center
- Geo replication replicates data to remote GlusterFS node (one way)
- One ICP may access one or more GlusterFS trusted storage pools



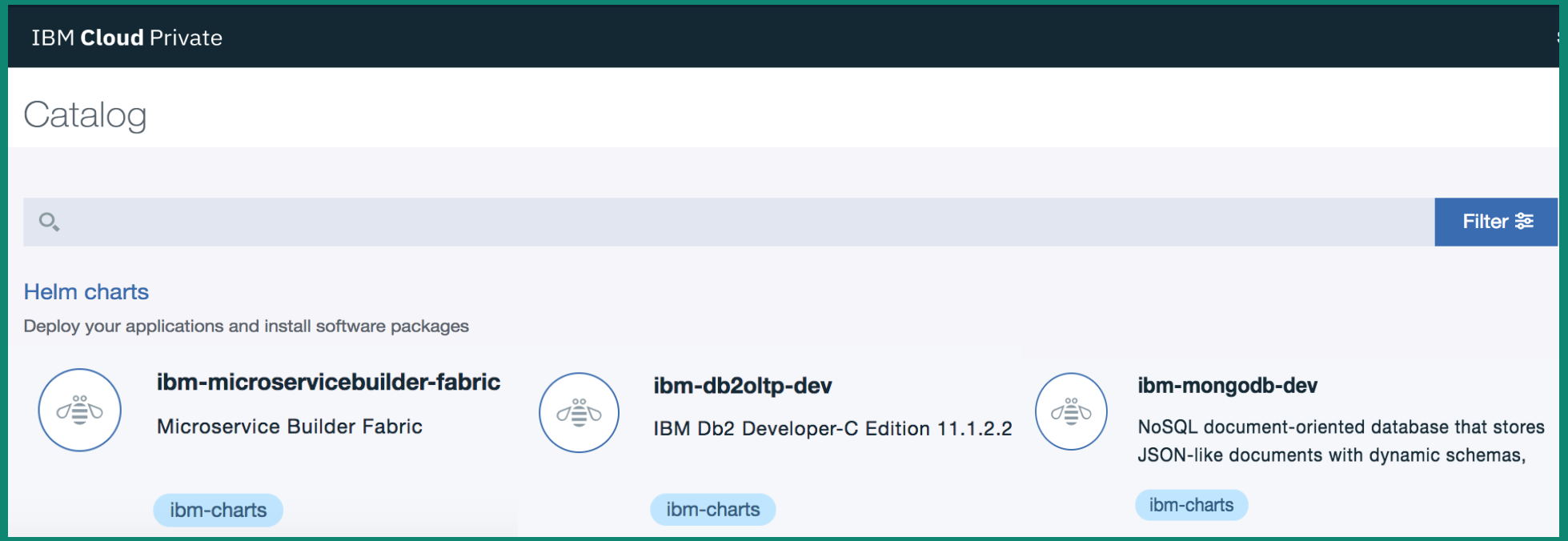
Container Platform Security – Vulnerability Advisor



- Policy Violations
- Vulnerable Packages
- Security Misconfigurations
- Best Practices
- Administrative control over deployments
- Live Container Scanning
- Integration between Vulnerability Advisor and IBM X-Force

What is the Catalog today?

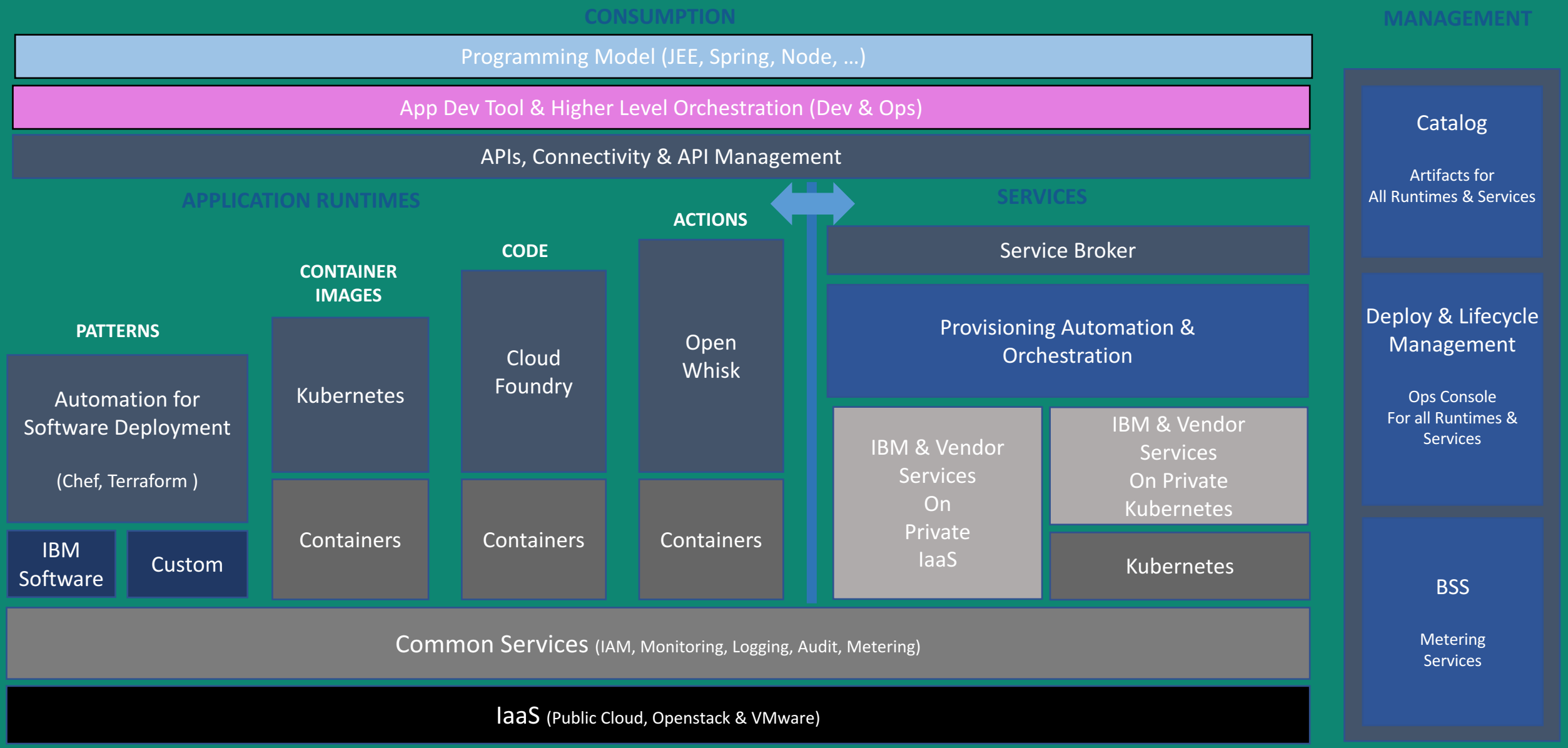
- A collection of Deployment Packages (Helm) displayed as tiles which provide access to software under separate license terms



*Helm is the **open standard** for Application **Packaging and Deployment** for **Kubernetes***
Helm charts automate the deployment of resources and prereqs including locations of Docker images

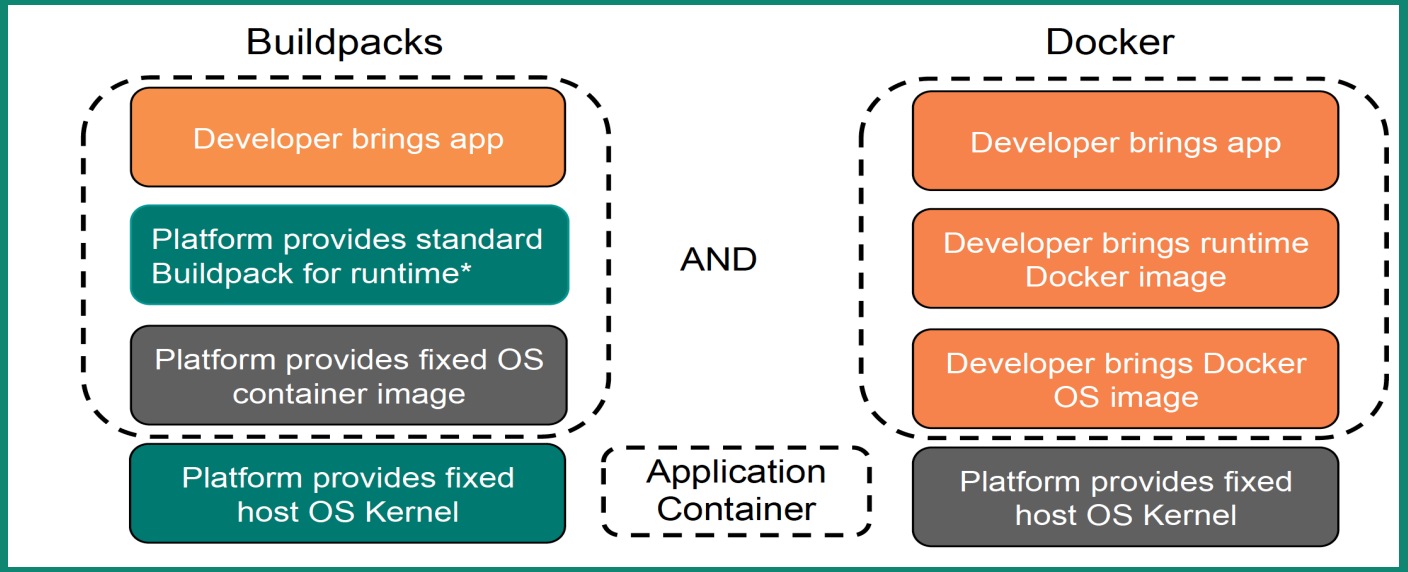
IBM Private Cloud – A more detailed view

developer.ibm.com/**code**



Cloud Foundry vs Kubernetes

CLOUDFOUNDRY



Kubernetes

*Node .Js, Tomcat, Pearl, Ruby on Rails …

- CF and Docker Containers with Kubernetes (K8) are operating on different levels of abstraction

- Choices to make
  - Select K8 when you want the flexibility to control all of the underlying technology, and deploy whatever you'd like wherever you'd like at large scale – Optimizing for Performance & Scalability.
  - Chose CF if you prefer to write code, and have the platform take care of all of the "plumbing" to get the code into a running application, monitor it's health, and scale at the expense of control and flexibility – Optimizing for Speed & Simplicity

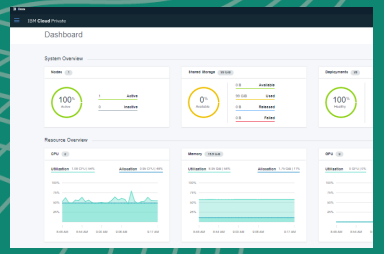
- IBM's recommendation – Use Both platforms at their best
  - Do rapid prototyping on Cloud Foundry PaaS
  - Harden prototypes & deploy on an Enterprise Grade K8 platform

IBM

developer.ibm.com/**code**

IBM Cloud Private

ICP and Kubernetes

kubernetes +  + + +

Kubernetes based container platform

Industry leading container orchestration platform across private, dedicated & public clouds

ICP Console

Integrated console providing graphical admin capability, security, logging, monitoring, etc.

Operational Services

To simplify hybrid automation, integration, management & developer experience

IBM Middleware, Data & Analytics Services

Cloud enabled middleware, databases & analytics to leverage and optimize current investments

ICP : What's the Difference - Extended Kubernetes Components

Core Operational Services

Monitoring Service
Prometheus or BYO

Logging Service
Elk or BYO

Metering Service
Product Insights

Security
Identify and Access Management
- LDAP integration and RBAC

Vulnerability Advisor (beta)

Open Source

Toolchains & Runtimes
Jenkins
Apache Tomcat
Open Liberty

Messaging
RabbitMQ

Data Services
MongoDB
PostgreSQL
Redis

Clustering
Galera

Http Servers
Nginx

Terminal Access
Web Terminal

IBM Software

Toolchain & Runtimes
IBM Microservice Builder
IBM WebSphere Liberty
IBM SDK for Node.js

Messaging
IBM MQ Advanced for Developers
IBM MQ Advanced

Data Services
IBM Db2 Dev-C
IBM Data Server Manager (for Db2 Dev-C)
IBM Db2 Direct Advanced Edition / AESE with Data Server Manager
IBM Db2 Warehouse Dev-C
IBM Db2 Warehouse Enterprise*
IBM Cloudant Developer Edition

Multi-cloud Management
IBM Cloud Automation Manager

Data Science
IBM Data Science Experience Developer Edition
IBM Data Science Experience Local*

Integration
IBM Integration Bus for Developers
IBM Integration Bus
IBM DataPower Gateway for Developers
IBM DataPower Gateway Virtual Edition

App Modernization Tooling
IBM Transformation Advisor

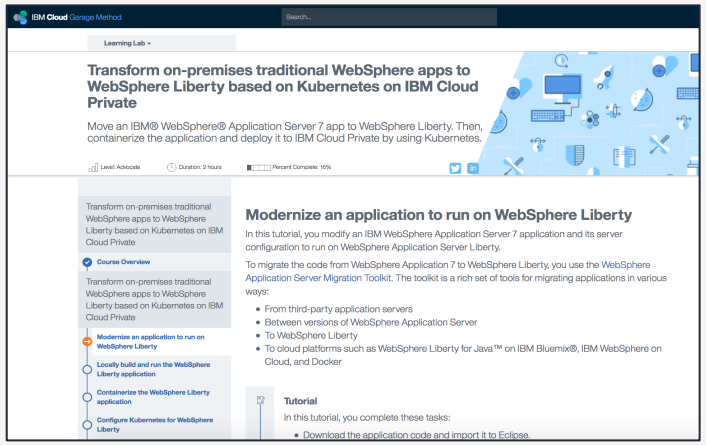
Monitoring
IBM Cloud Application Performance Management for DevOps (beta)

HPC
IBM Spectrum LSF Community Edition

**coming soon*

\+ Develop or bring your own...
Self written, community and open source compatible with Kubernetes 1.7

What's the Difference – Operational Services

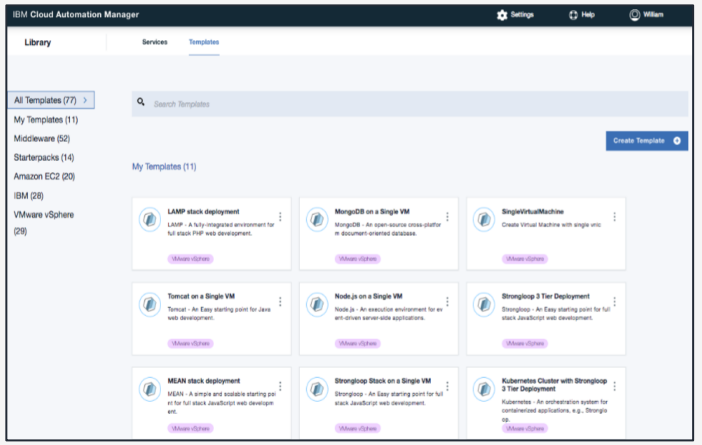


IBM Cloud Architecture Center

Best practice guidance
Tutorials and sample code

IBM Transformation Advisor

Assess & Manage traditional apps
Expose, Refactor, Shift, Extend

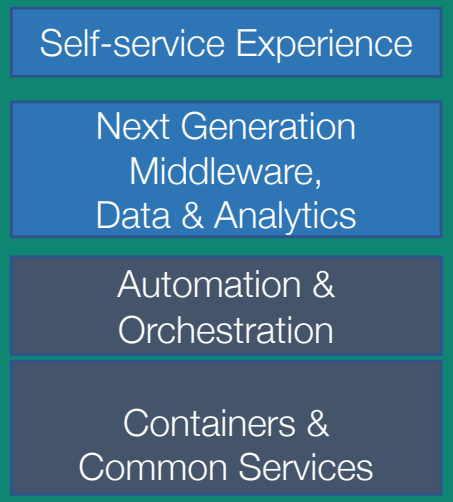


IBM Cloud Automation Manager

Multi-Cloud Provisioning
Pre-Built Automation Content

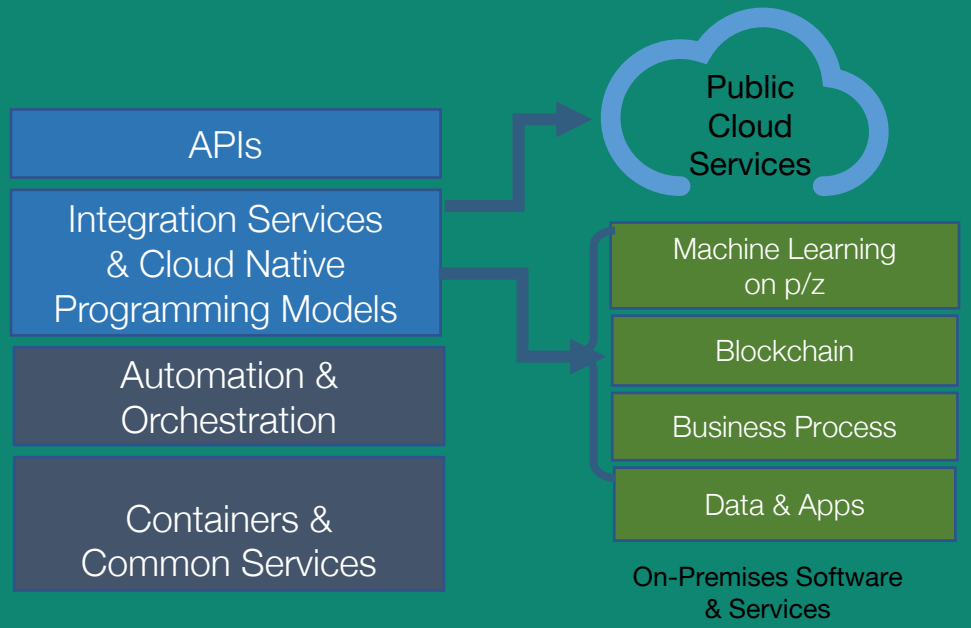
ICP Use Cases

1. Optimize legacy apps with cloud



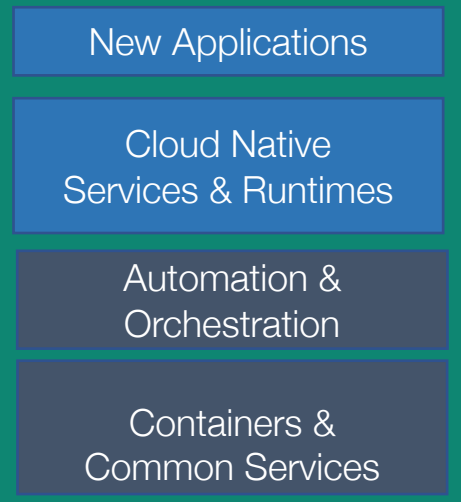
Cloud-enabled middleware

2. Open your datacenter to work with cloud services



Integration & Hybrid Cloud

3. Create new cloud native applications



New Applications

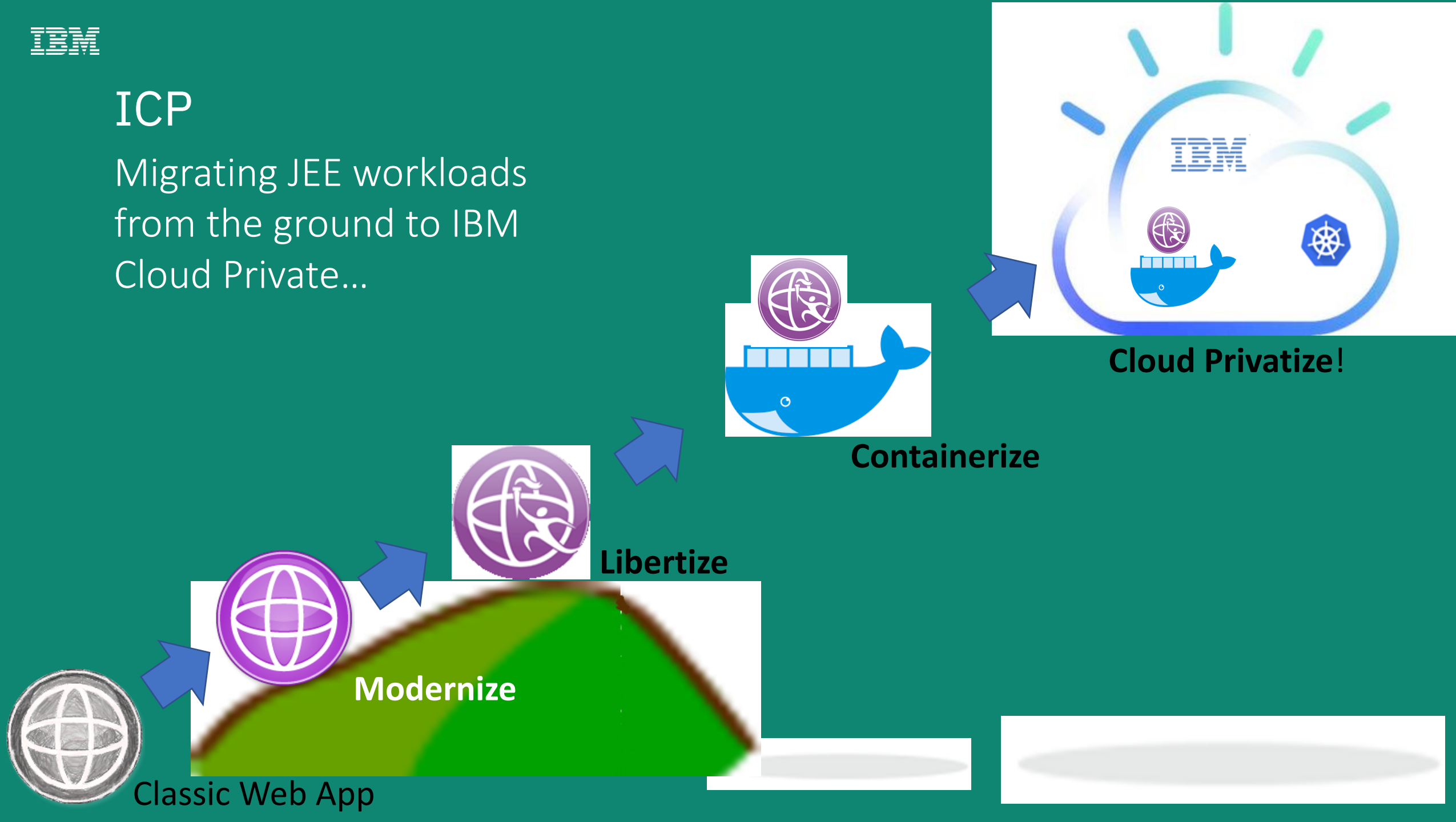
Why IBM Cloud Private

- **Rapid development and deployment:**
  - Minutes or hours vs. days or weeks, huge catalogue of OS and IBM Services , advanced Analytics and Machine learning options …
  - Non-disruptive upgrade of platform integrated with enterprise network, storage, security, performance and production needs
  - Enablement of new and existing developers & integration with existing Dev/Sec/Ops tools
- **Investment leverage:**
  - Infrastructure choice and complete portability
  - Leverage existing applications and skills while reducing TCO
  - Open community-based platforms for choice and flexibility, on- and off-premises
  - No Vendor lock in !
- **Application modernization:**
  - Modernization and optimization across multi-cloud environments – Develop Once, deploy anywhere
  - Reduced risk by running applications on enterprise-grade software & data platforms optimized for cloud
- **Differentiated enterprise integration:**
  - Set of new services available on-premises, complemented with public cloud services (Watson)
  - Integration of applications with services for operational simplicity and reduced cost
  - Integrated cloud management solutions to automatically provision and govern multi-cloud environments with speed and control , Coupled with IBM's expertise & Services

124

ICP

Migrating JEE workloads from the ground to IBM Cloud Private…



Helm :Package Manager

Topics

- Life without Helm
- What is Helm
- How to use helm
- Hands-on with helm

Life without Helm

- Write Kubernetes manifests by hand
- Do this every time you need to release anything
- Figure out your own sharing
- Tweak resources by hand
- Use kubectl to manage these

Helm

- Package manager for Kubernetes
  - RedHat/Fedora – rpm
  - Microsoft – msi
  - Ubuntu – apt
  - Kubernetes – helm

The client is responsible for **managing charts**, and the server is responsible for **managing releases**

- Two components – a client (helm) and a server (tiller)

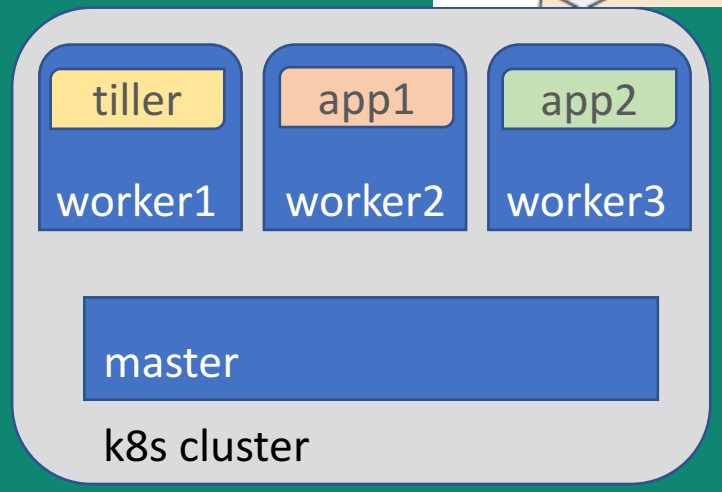
Helm Client

or...

helm

1. Manage charts
2. Communicate with server

1. Render charts
2. Deploy
3. Interface for state tracking

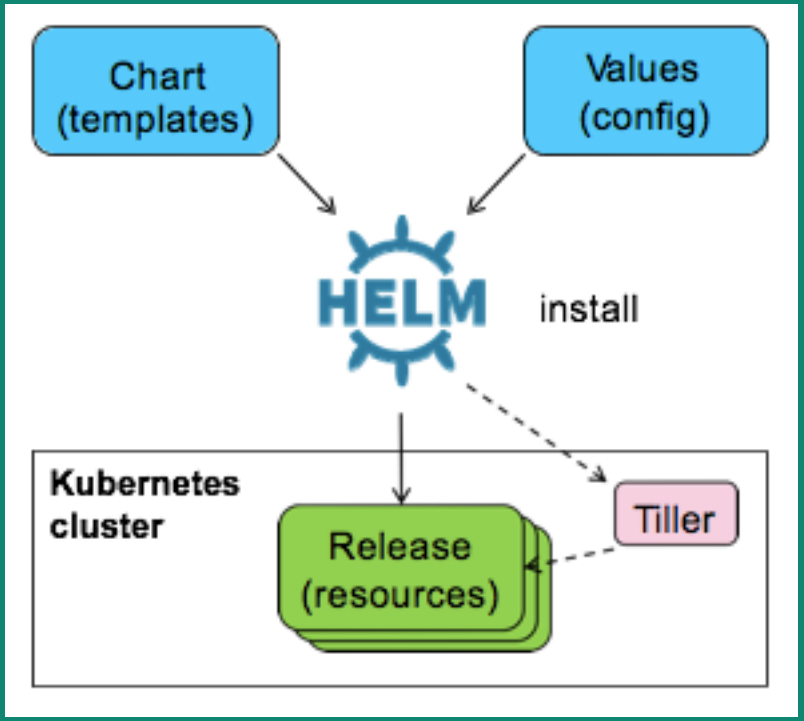


What is Helm

- Helm enables multiple Kubernetes resources to be created with a single command
  - Deploying an application often involves creating and configuring multiple resources
  - A Helm chart defines multiple resources as a set
- An application in Kubernetes typically consists of (at least) two resource types
  - Deployment – Describes a set of pods to be deployed together
  - Services – Endpoints for accessing the APIs in those pods
  - Could also include ConfigMaps, Secrets, Ingress, etc.
- A default chart for an application consists of a deployment template and a service template
  - The chart creates all of these resources in a Kubernetes cluster as a set
  - Rather than manually having to create each one separately via `kubectl`

Helm Terminology : Helm, Charts, Repositories, Releases, Tiller

- Helm
  - Helm installs charts into Kubernetes, creating a new release for each installation
  - To find new charts, search Helm chart repositories
- Chart
  - Templates for a set of K8s resources necessary to run an application
  - The chart includes a values file that configures the resources
- Repository
  - Storage for Helm charts
  - stable– The namespace of the hub for official charts
- Release
  - An instance of a chart running in a Kubernetes cluster
  - The same chart installed multiple times creates many releases
- Tiller
  - Helm templating engine, runs in a pod in a Kubernetes cluster
  - Tiller processes the chart to generate the resource manifests, then installs the release into the cluster
  - Tiller stores each release as a Kubernetes config map



Helm : Advantages

- Deploy all of the resources for an application with a single command
  - Makes deployment easy and repeatable

```
$ helm install <chart>
```

- Separates configuration settings from manifest formats
  - Edit the values without changing the rest of the manifest
  - `values.yaml` – Update to deploy the application differently

- Upgrade a running release to a new chart version

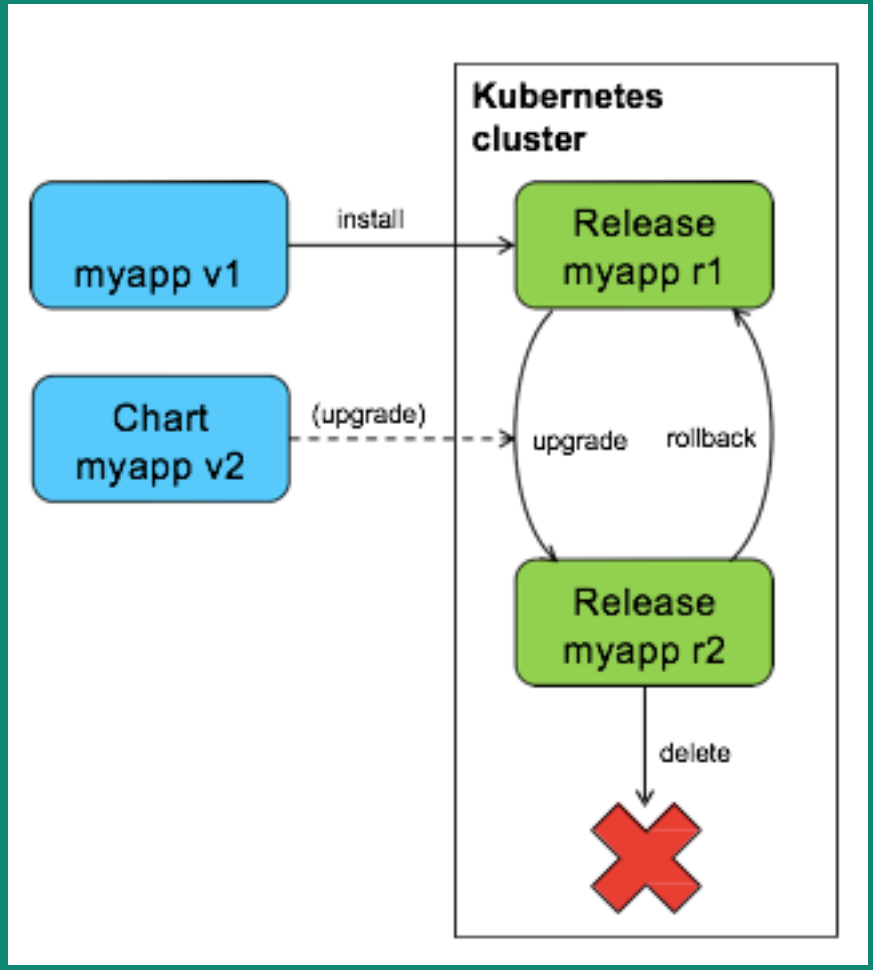
```
$ helm upgrade <release> <chart>
```

- Rollback a running release to a previous revision

```
$ helm rollback <release> <revision>
```

- Delete a running release

```
$ helm delete <release>
```

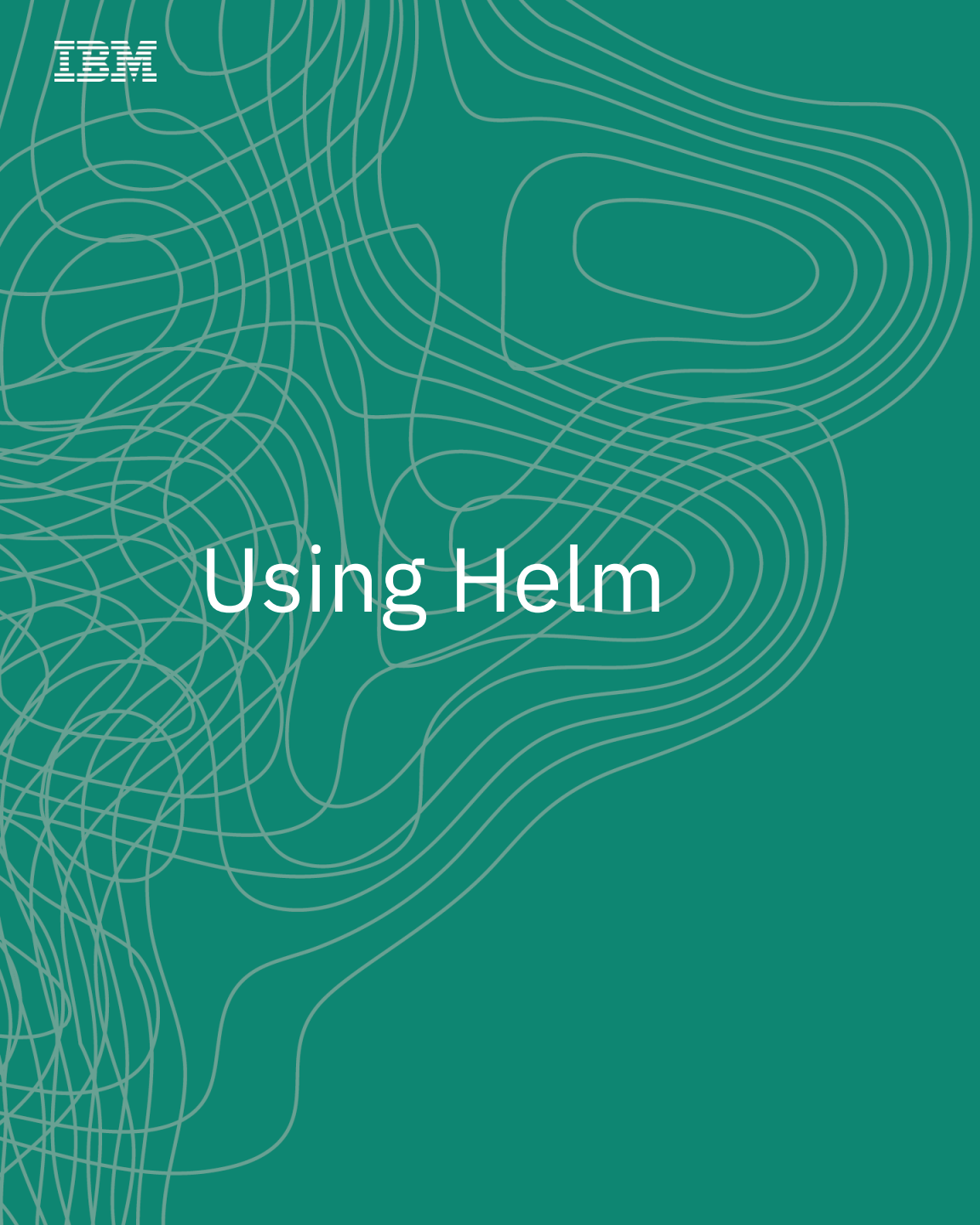


Helm packages are called "charts"

- Each chart is a folder named for the chart
- A minimum chart contains:
  - description - Chart.yaml
  - templates – folder with kubernetes resource definitions

- Most charts include values.yaml to specify installation defaults used in templates

```
wordpress/
  Chart.yaml          # A YAML file containing information about the chart
  LICENSE             # OPTIONAL: A plain text file containing the license for the chart
  README.md           # OPTIONAL: A human-readable README file
  requirements.yaml # OPTIONAL: A YAML file listing dependencies for the chart
  values.yaml         # The default configuration values for this chart
  charts/             # OPTIONAL: A directory containing any charts upon which this chart depends.
  templates/          # A directory of templates that, when combined with values,
                      # will generate valid Kubernetes manifest files.
  templates/NOTES.txt # OPTIONAL: A plain text file containing short usage notes
```



developer.ibm.com/**code**

Using Helm

Helm: Commands

- Install Tiller
- `$ helm init`
- Create a chart
- `$ helm create <chart>`
- List the repositories
- `$ helm repo list`
- Search for a chart
- `$ helm search <keyword>`
- Info about a chart
- `$ helm inspect <chart>`
- Deploy a chart (creates a release)
- `$ helm install <chart>`
- List all releases
- `$ helm list --all`
- Get Status of release
- `$ helm status <release>`
- Get details about the release
- `$ helm get <release>`
- Upgrade a release
- `$ helm upgrade release <chart>`
- Rollback a release
- `$ helm rollback release <revision>`
- Delete a release
- `$ helm delete <release>`

Working with repositories

```
$ helm repo list
    NAME URL
    stable https://kubernetes-charts.storage.googleapis.com
```

```
$ helm search jenkin
    NAME            VERSION         DESCRIPTION
    stable/jenkins          0.14.1  Open source continuous integration server. It s...
```

Add chart to repo

```
$ helm repo add my-charts https://my-charts.storage.googleapis/com/
```

Installing an application

To deploy an application into Kubernetes, install that application's Helm chart

- $ helm search mysql

NAME VERSION DESCRIPTION

stable/mysql 0.3.5 Fast, reliable, scalable, and easy to use open-...

stable/mariadb 2.1.10 Fast, reliable, scalable, and easy to use open-...

$ helm install stable/mysql

```
$ helm install stable/mysql

Fetched stable/mysql to mysql-0.1.1.tgz
NAME: loping-toad
LAST DEPLOYED: Thu Oct 20 14:54:24 2016
NAMESPACE: default
STATUS: DEPLOYED

RESOURCES:
==> v1/Secret
NAME                TYPE      DATA      AGE
loping-toad-mysql   Opaque    2         3s

==> v1/Service
NAME                CLUSTER-IP          EXTERNAL-IP          PORT(S)    AGE
loping-toad-mysql   192.168.1.5         <none>               3306/TCP   3s

==> extensions/Deployment
NAME                DESIRED   CURRENT   UP-TO-DATE           AVAILABLE AGE
loping-toad-mysql   1         0         0                    0         3s

==> v1/PersistentVolumeClaim
NAME                STATUS    VOLUME    CAPACITY  ACCESSMODES         AGE
loping-toad-mysql   Pending
```

- Install output
  - Details about the release
  - Details about its resources
- Chart
  - `stable/mysql`
- Release name
  - `loping-toad` (auto generated)
- Resources
  - Four total, one of each type
  - All named `loping-toad-mysql`
  - `Secret`
  - `Service`
  - `Deployment`
  - `PersistentVolumeClaim`

Developing Charts

- Creating a new chart generates a directory with sample files

```
$ helm create my-chart
$ tree my-chart
```

```
my-chart/
    |- Chart.yaml                    # Information about the chart
    |- values.yaml                   # The default configuration values for this chart
    |- charts/                       # Charts that this chart depends on
    |- templates/                    # The template files
        |- NOTES.txt                                  # OPTIONAL: A plain text file
containing short usage notes
        |- _helpers.tpl              # OPTIONAL: The default location for template partials
        |- deployment.yaml
        |- service.yaml
```

- By default, a chart starts with sample templates for a Kubernetes deployment and service
  - In the simplest case, just edit the `values.yaml` file

Chart LifeCycle Hooks

Hooks

- **pre-install**
  - Executes after templates are rendered
  - Before any resources are created in Kubernetes
- **post-install**
  - Executes after all resources are loaded into Kubernetes
- **pre-delete**
  - Executes before any resources are deleted from Kubernetes
- **post-delete**
  - Executes after all of the release's resources have been deleted
- **pre-upgrade**
  - Executes after templates are rendered
  - Before any resources are loaded into Kubernetes
- **post-upgrade**
  - Executes after all resources have been upgraded
- **pre-rollback**
  - Executes after templates are rendered
  - Before any resources have been rolled back
- **post-rollback**
  - Executes after all resources have been modified

Hooks in the Helm Install Lifecycle

1. User runs an install in the Helm CLI
2. Helm CLI loads the chart into Tiller
3. Tiller renders the templates
4. Tiller executes the pre-install hooks
5. Tiller loads the resulting resources into Kubernetes
6. Tiller executes the post-install hook
7. Tiller returns the release data to the client
8. The client exits

- A hook can be any Kubernetes resource
  - it's a k8s job
  - Goes into template directory

Sharing Charts

- A chart is a directory
  - Easy for a Helm client to use the chart directories on the same computer
  - Difficult to share with other users on other computers

- Packaging a Chart
  - Build charts.yaml file and other files into tar file

```
$ helm package <chart-path>
```

- Chart repository
  - HTTP server that houses a index.yaml file and optionally some packaged charts
  - Server can be any HTTP server that can serve YAML and tar files and can answer GET requests
    - Ex: Google Cloud Storage (GCS) bucket, Amazon S3 bucket, Github Pages, or even create your own web server
  - To add a chart to the repository, copy it to the directory and regenerate the index

Lab: Helm Charts: Prepare and install

Summary

- Seen Containers and How Docker tool simplify managing containers
- How Containers can be orchestrated using Kubernetes
- Kubectl CLI to (Client) to manage the orchestration
- At last we have seen
  - IBM leverages Kubernetes for Container Services and IBM Cloud Private
  - Helm : Package Manager

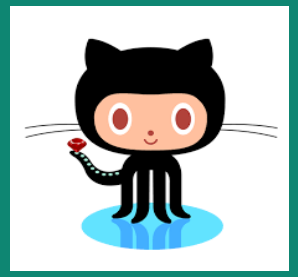
https://developer.ibm.com/code/

Signup for IBM Cloud
https://bluemix.net

https://www.ibm.com/watson/products-services/

Stay Connected and continue coding !

developer.ibm.com/**code**



Code & instructions

https://github.com/IBMDevConnect
https://github.com/IBM
https://github.com/IBM-Cloud
https://ibm-cloud.github.io/#!/
http://ibm.github.io
https://github.com/watson-developer-cloud
https://github.com/ibm-bluemix-mobile-services

developerWorks

https://developer.ibm.com/in/
https://developer.ibm.com/tv/

Recipes

https://developer.ibm.com/recipes/

Join our Slack team and stay in touch with the experts

https://ibmdevconnect.slack.com

Send in your request

http://ibm.biz/slackrequest

Data Science Experience

https://datascience.ibm.com

Apply for IBM Global Entrepreneur Program

https://developer.ibm.com/startups

Join our Meetup groups

Bangalore : https://www.meetup.com/IBMDevConnect-Bangalore

Delhi / Gurugram / Noida : https://www.meetup.com/ibmcloudecosystem/

Mumbai / Pune : https://www.meetup.com/Cloud-Mumbai-Meetup/

Hyderabad / Vishakapatnam: https://www.meetup.com/Hyderabad-Cognitive-with-Cloud