Kubernetes Workshop Series

JTC02
Kubernetes
Basic
Concepts

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

Welcome to the

Kubernetes Workshop Series

Housekeeping

Meeting is being recorded to be shared on Social Media

Meeting Mute All: Unmute to speak

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

Questions:

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

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

Who am I?

Niklaus Hirt

Passionate about tech for over 35 years

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

✉ nikh@ch.ibm.com
@nhirt

Agenda - Kubernetes Basic Concepts

Module 0: Prepare the Labs
Module 1: Kubernetes
Module 2: Use Kubernetes in the cloud
Module 3: Kubernetes Applied
Module 4: Kubernetes Hands-On

Sources and documentation will be available here:

Recording from JTC01
https://www.youtube.com/channel/UCIS0jmGOQrG2AKKPkTJYj9w/about

https://github.com/niklaushirt/k8s_training_public

https://github.com/niklaushirt/training

Session Quiz & Feedback

We will collect some feedback and run a quiz or two.

Please make sure you can access https://kahoot.it/
either on your PC of Phone.

You will get the Game PIN
later in the training.

QUIZZ!!!

https://kahoot.it/

Kubernetes Workshop Series

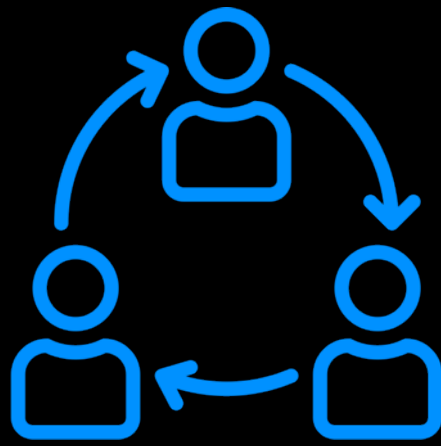
Prepare the Labs

00

Session Objectives

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

Following some lectures will be ***hands-on*** work that each participant can to complete.



JTC90 Lab Setup

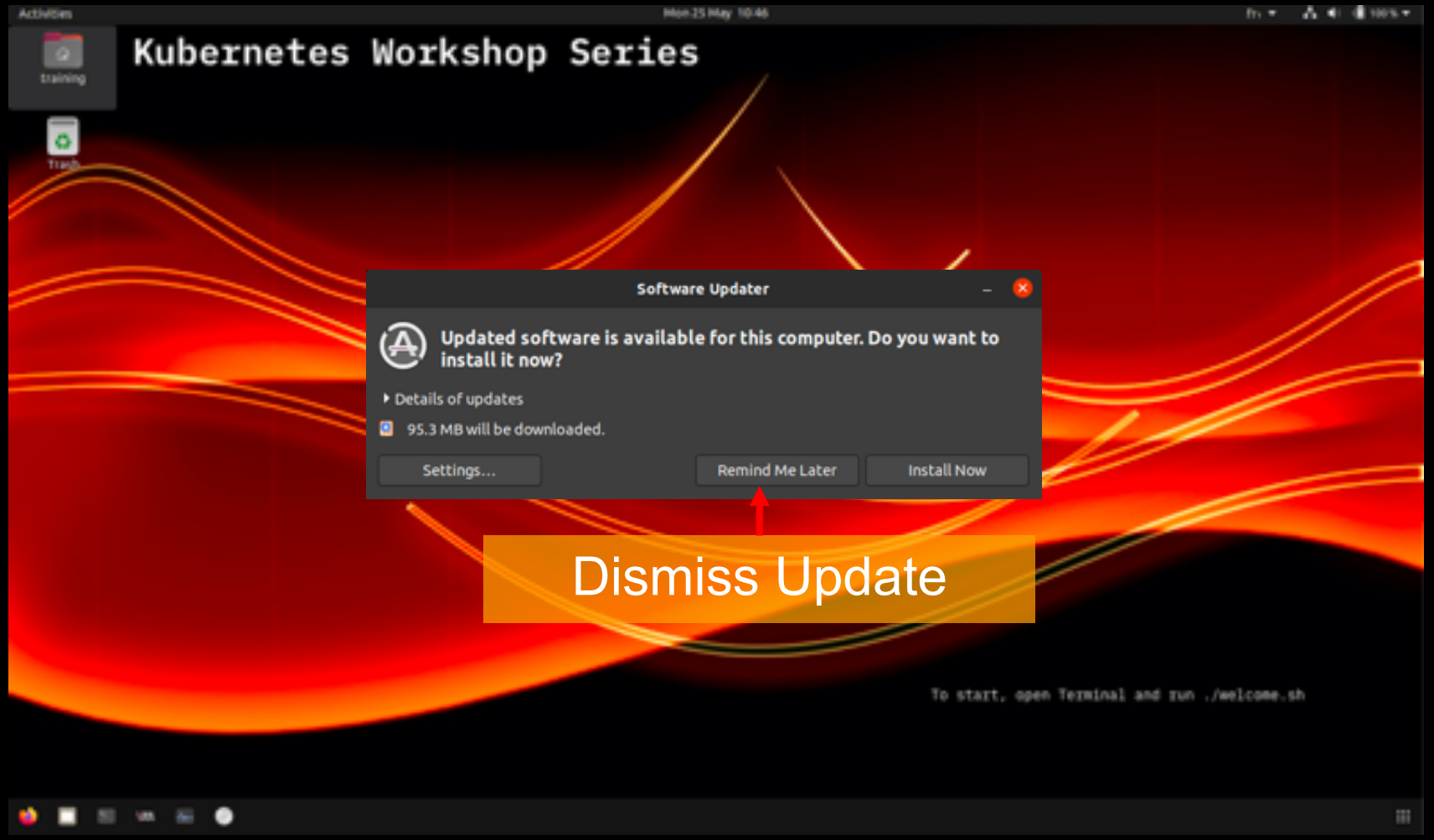
Task 1: Download Training VM

Task 2: Setup VMWare / VirtualBox

Task 3: Start Training VM

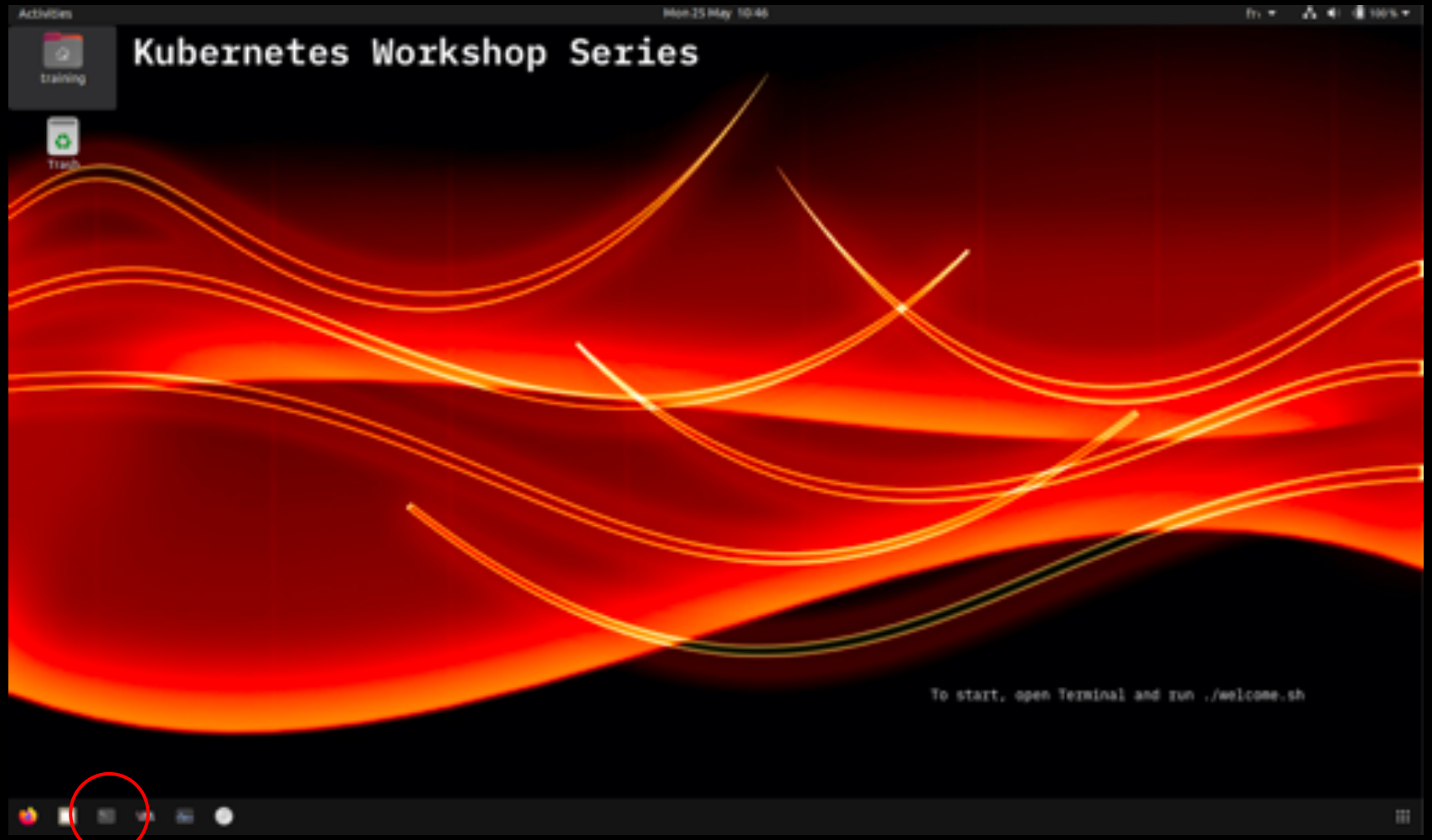
Task 4: Login / Check

Accessing your Personal Training Environment



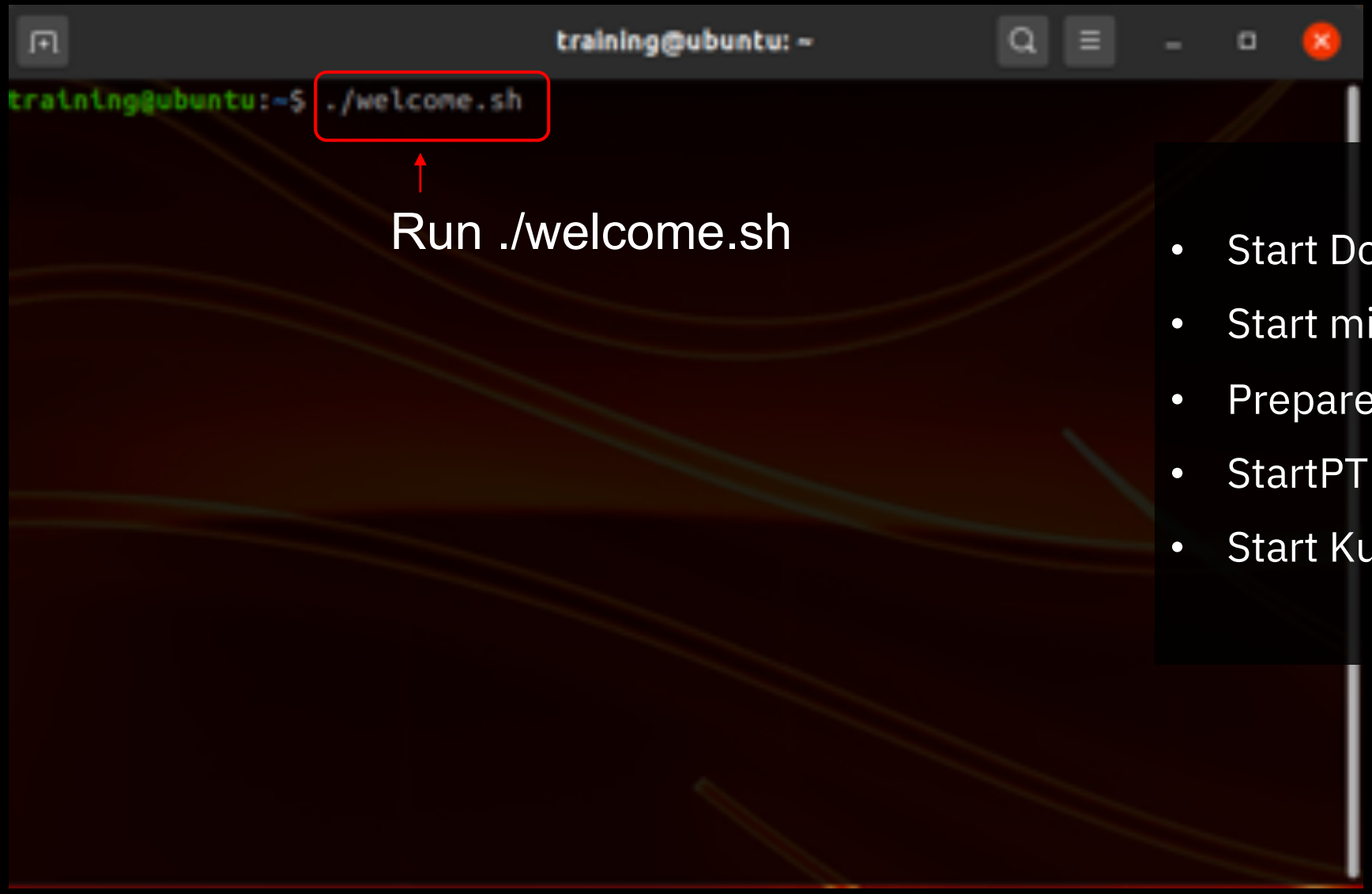
Accessing your Personal Training Environment

Accessing your Personal Training Environment



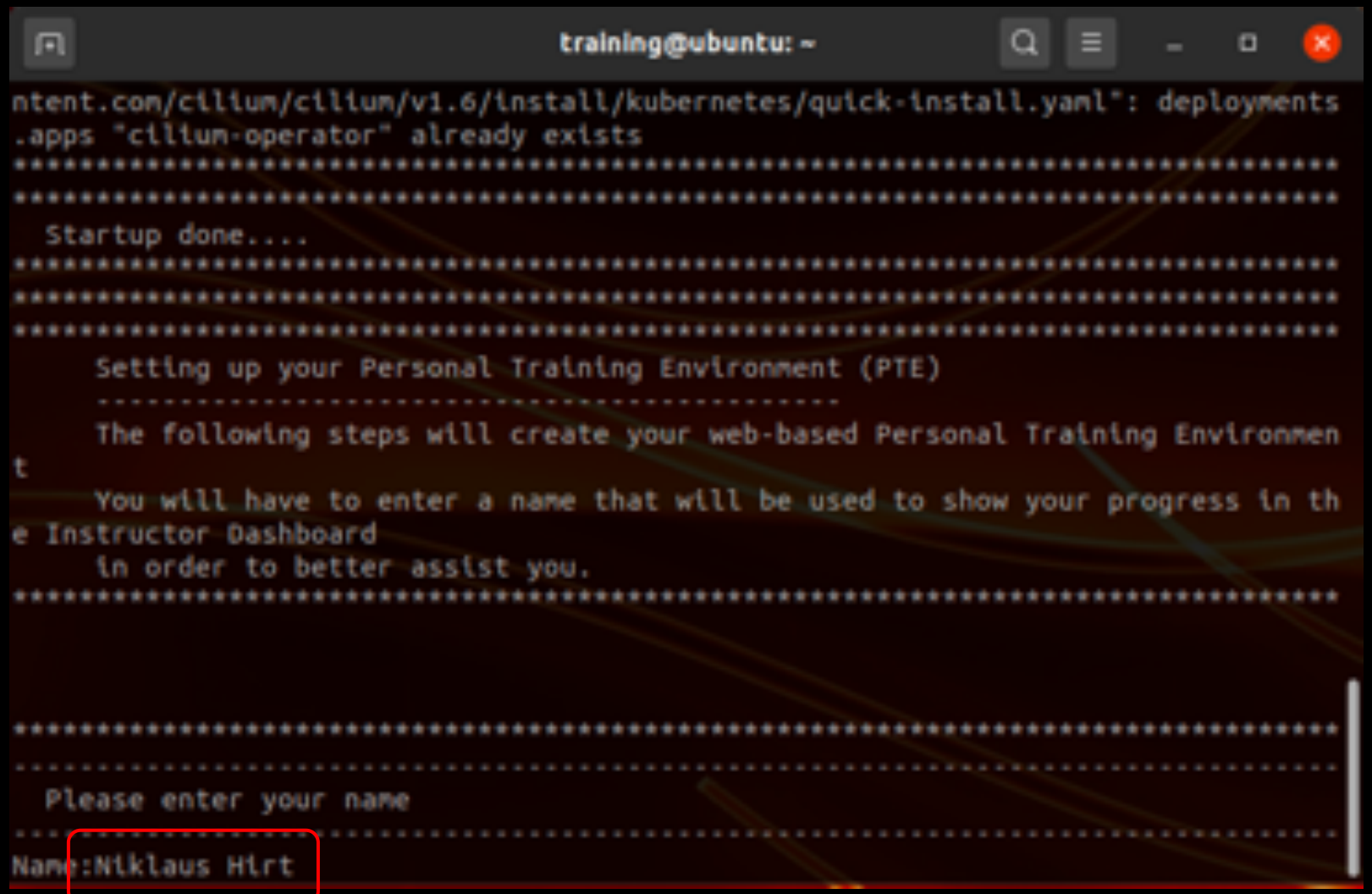
Start Terminal

Accessing your Personal Training Environment



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

Accessing your Personal Training Environment



Enter your name

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

Accessing your Personal Training Environment

Troubleshooting

- If the startup script doesn't work you can run ./resetEnvironment.sh
  (this can take up to 30 minutes as it has to redownload all Docker images)

- If you lose your PTE Webpage just run `minikube service student-ui`

- Windows 10 problems can mostly be fixed by turning off Hyper-V by running (as admin)
  `bcdedit /set hypervisorlaunchtype off`
  and rebooting.
  This disables Hyper-V and allows Virtualbox to support nested virtualisation.

- You can turn it back on again with
  `bcdedit /set hypervisorlaunchtype auto`

Accessing your Personal Training Environment

Troubleshooting

I have added a standalone version to the Git repository for participants wishing to run the Labs directly on their PC.

This is **untested** and I cannot guarantee that all the Labs will be working 100%.

You must have the following setup on your PC:

- Minikube
- Docker
- Git

1. Clone the repository to your home directory
   ```
   git clone https://github.com/niklaushirt/training.git
   ```
2. Go to the installation directory
   ```
   cd ~/training/standalone
   ```
3. Run the preparation script
   ```
   ./welcome.sh
   ```

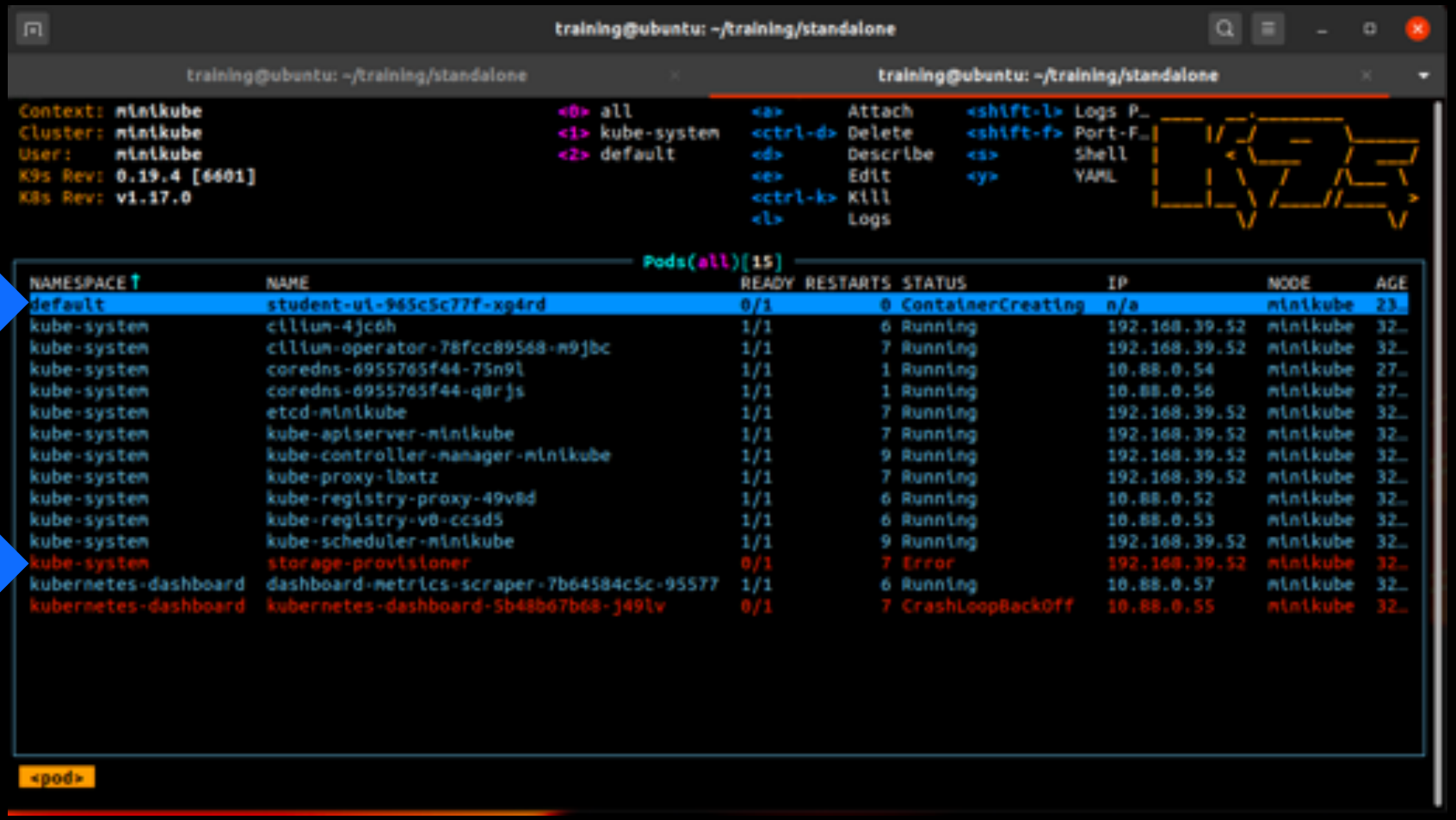
Accessing your Personal Training Environment

Troubleshooting

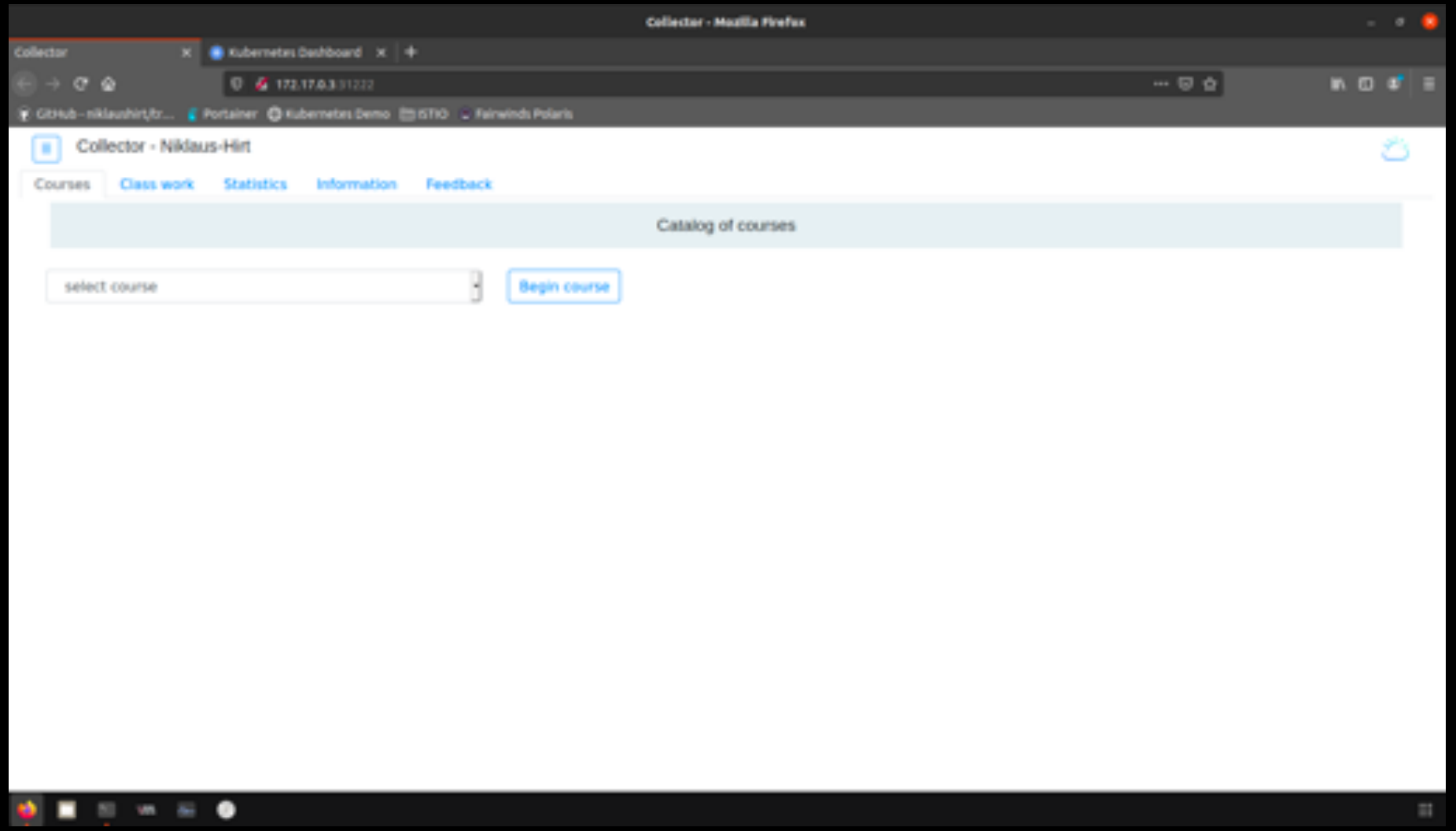
- Run ***k9s*** in the Terminal – wait for all the pods to be Running (blue – 1/1)

Image pulling - wait

Dependencies - wait



Accessing your Personal Training Environment



When completed, your PTE and Kubernetes Dashboard will open automatically

JTC90 Lab Setup

EVERYBODY OK ?

Task 1: Download Training VM

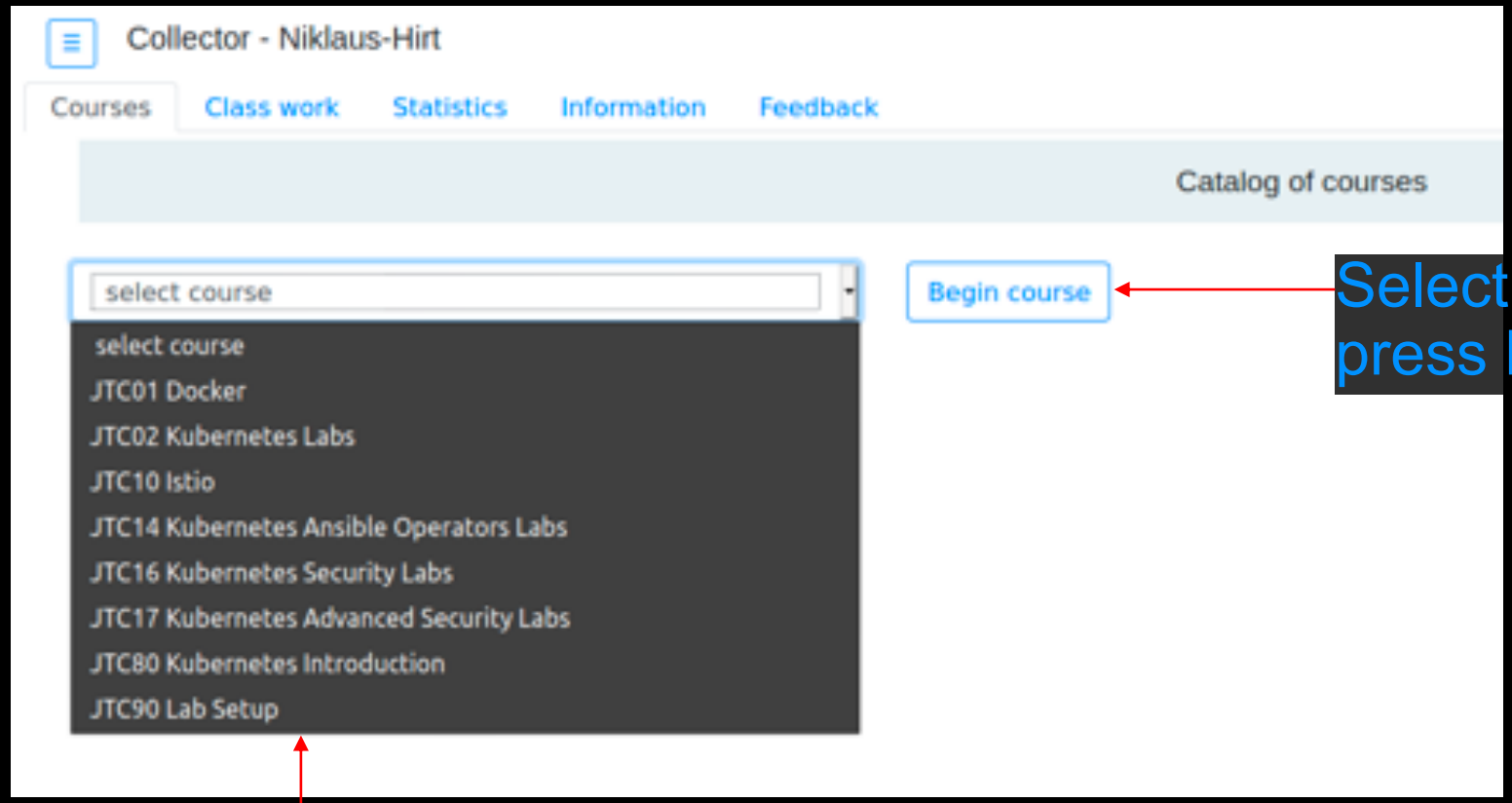
Task 2: Setup VMWare / VirtualBox

Task 3: Start Training VM

Task 4: Login / Check

Accessing your Personal Training Environment

Name will be shown

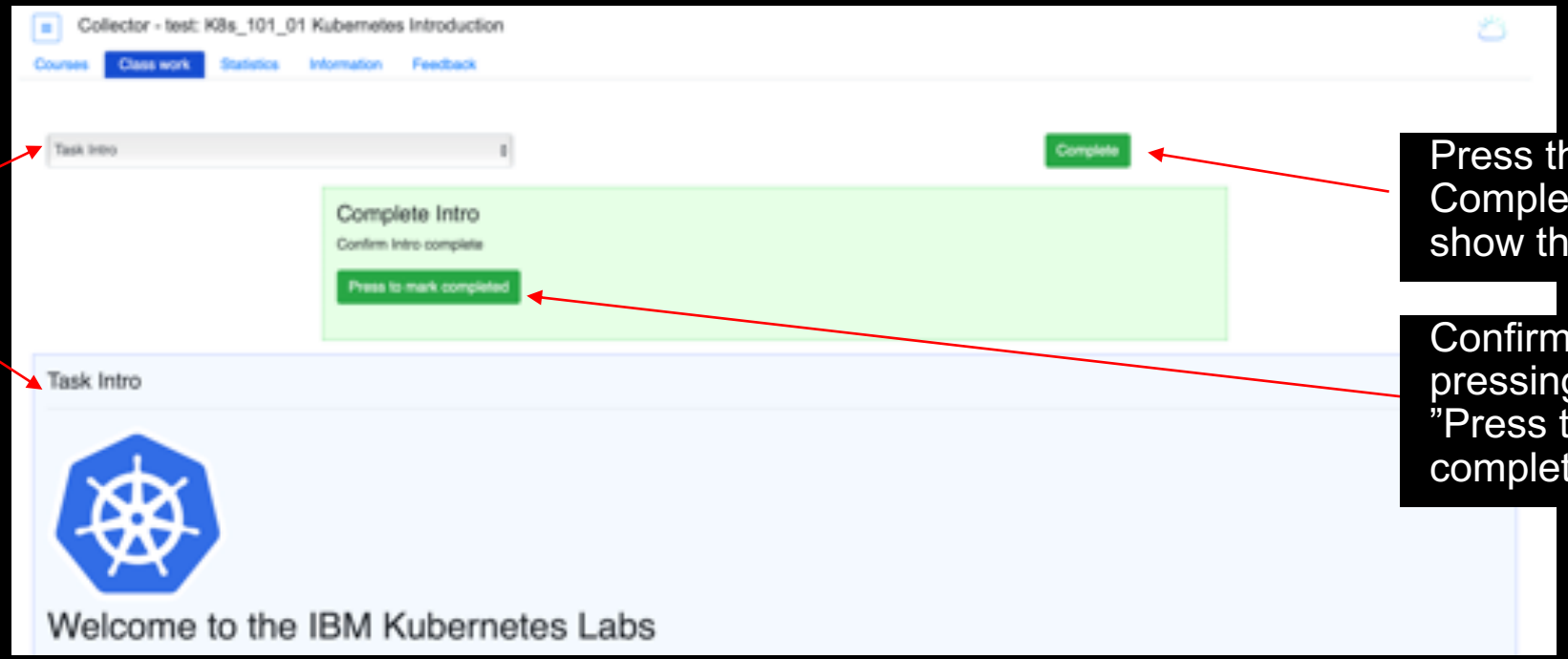


Select course and press button to begin

Current course catalog

Class Work

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



Press the green Complete button to show the green portion.

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

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

Following your progress

Course title

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

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

Kubernetes Workshop Series

Kubernetes

01

What happened so far...
Everybody Loves Containers

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

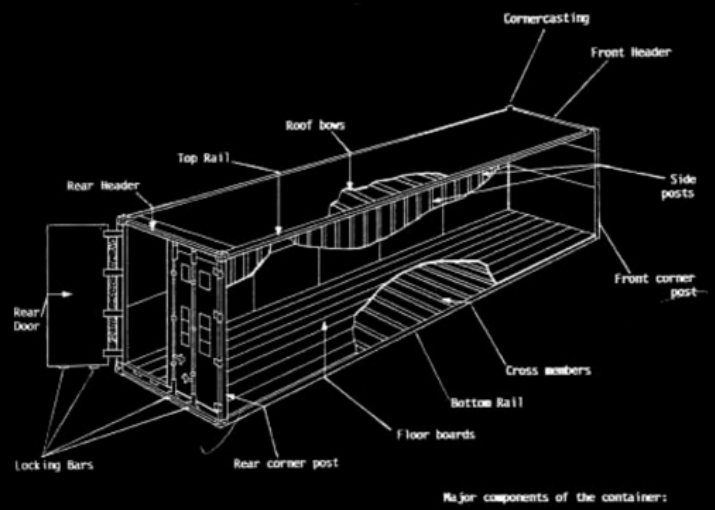
Containers work by **isolating the differences between applications** inside the container so that everything outside the container can be standardized

What happened so far…
Microservices implementation with Containers

Why it works – separation of concerns

Development

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



Operations

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

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

Everybody Loves Containers

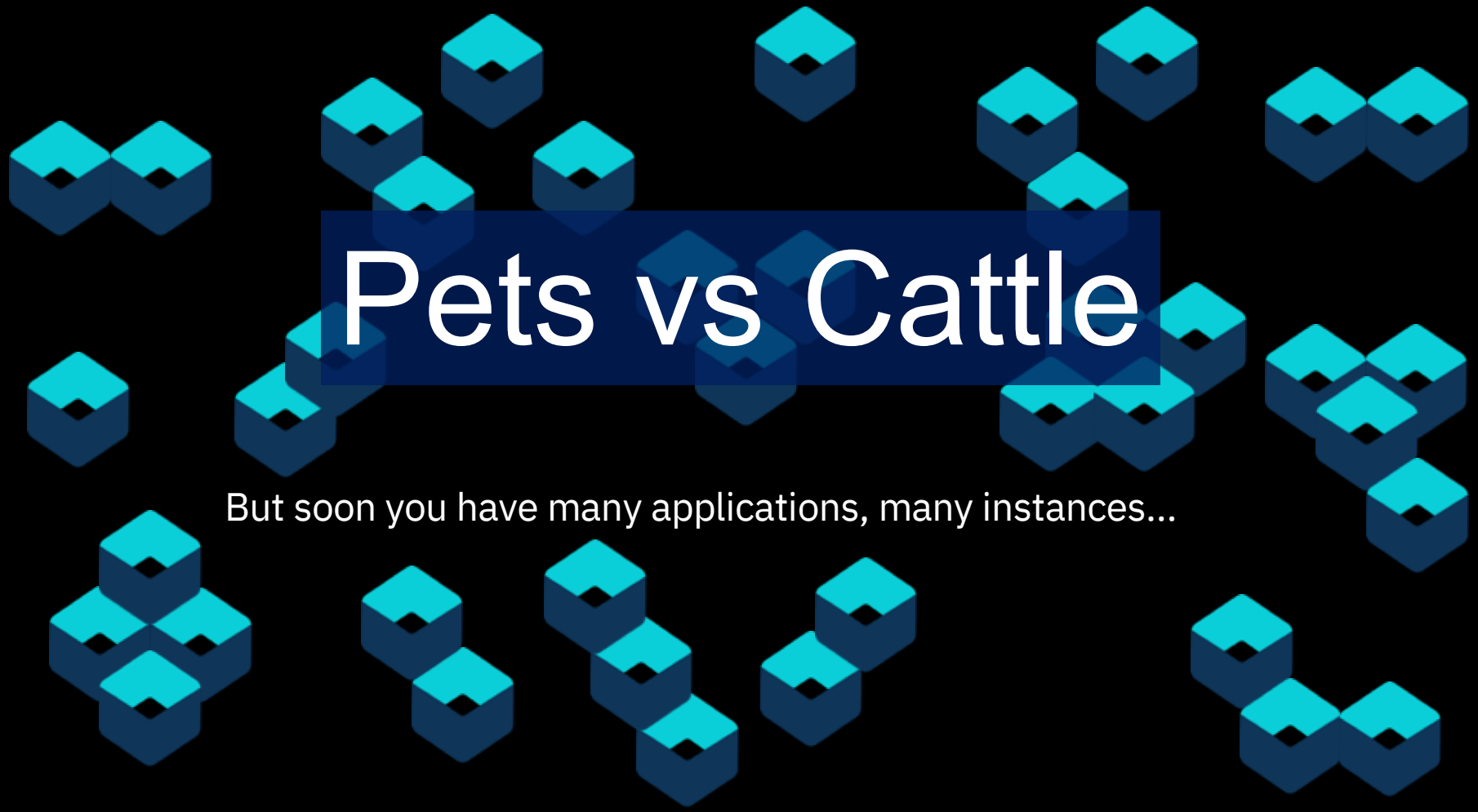
But when you go

From this….

To this….

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

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



Pets vs Cattle

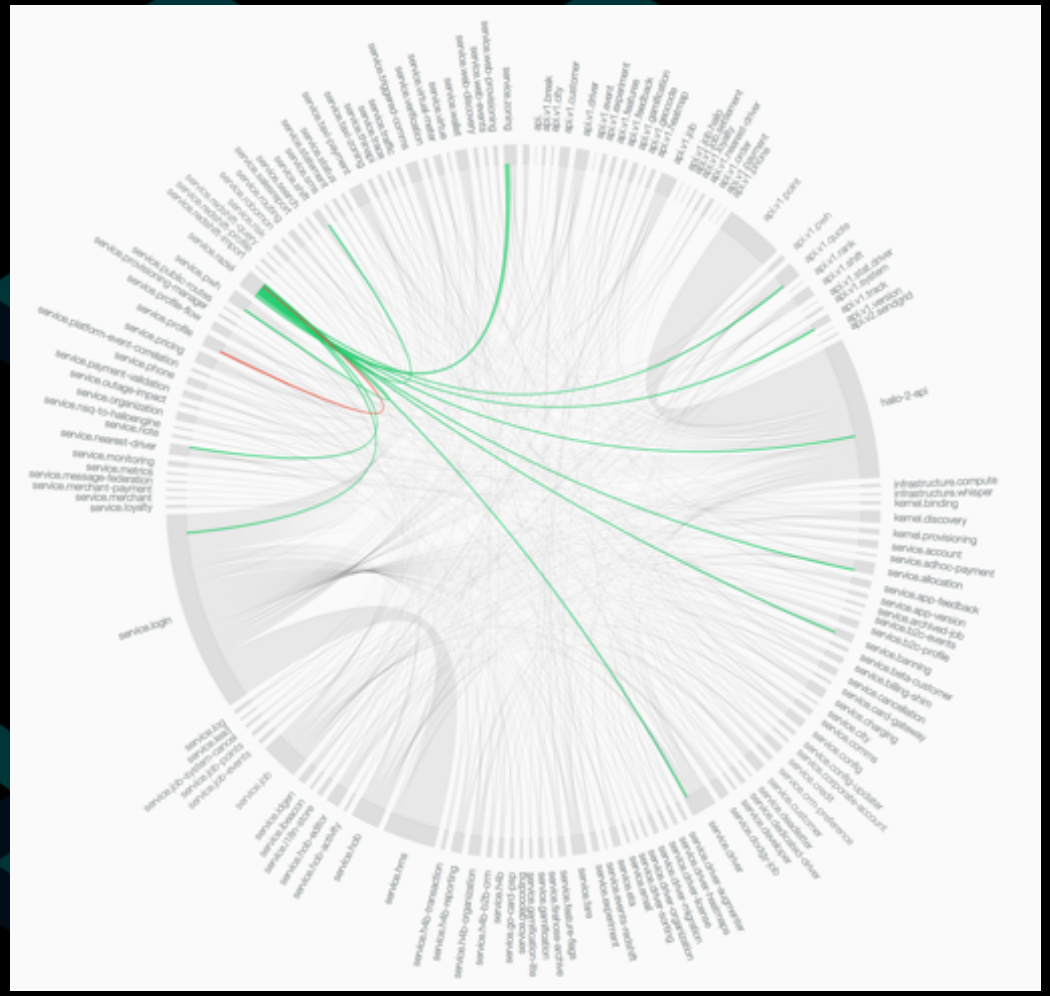
But soon you have many applications, many instances...

The trade off

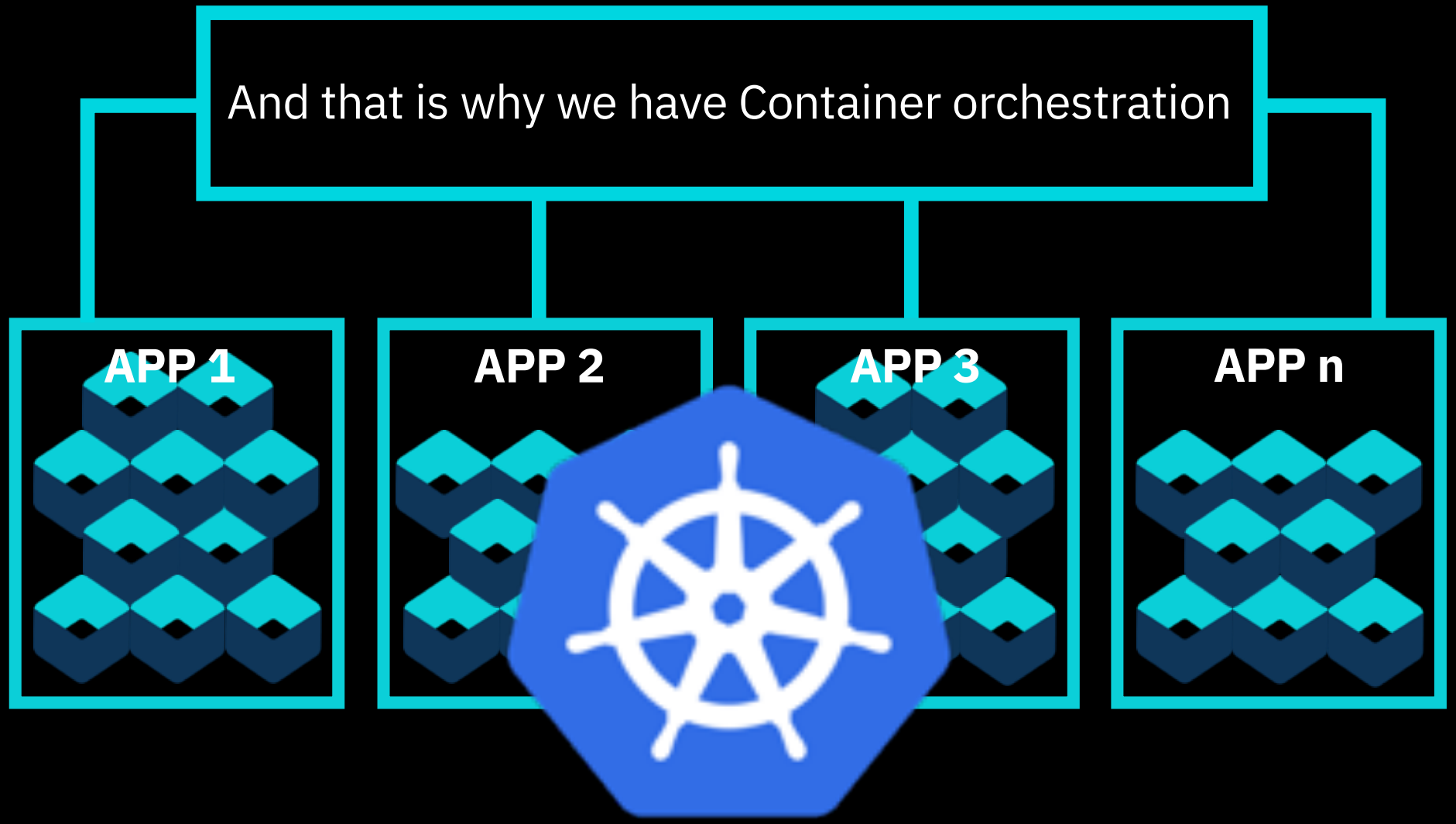
Improved delivery velocity

in exchange for

increased operational complexity



Enter Kubernetes (K8s)



Kubernetes – Declarative System

Imperative Systems

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

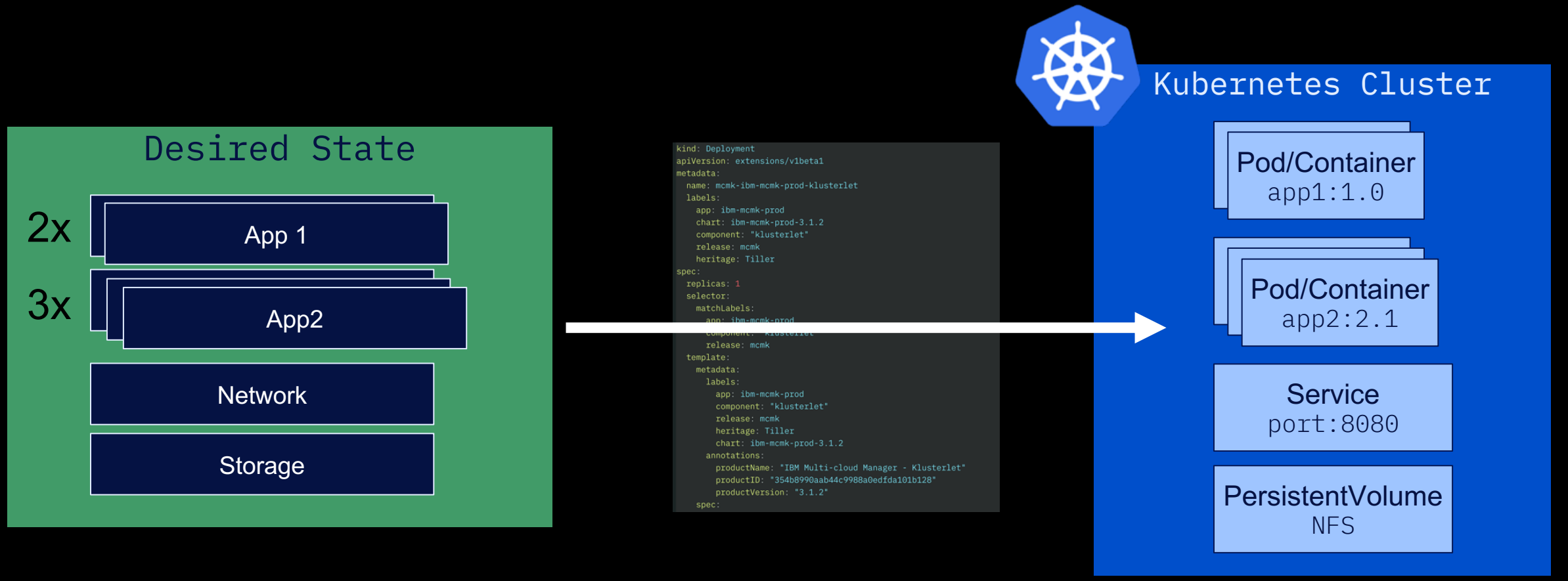
Declarative Systems

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

Kubernetes – Declarative System

The Desired State

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



So what is Kubernetes?

Container orchestrator

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

Manage applications, not machines

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

Designed for extensibility

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

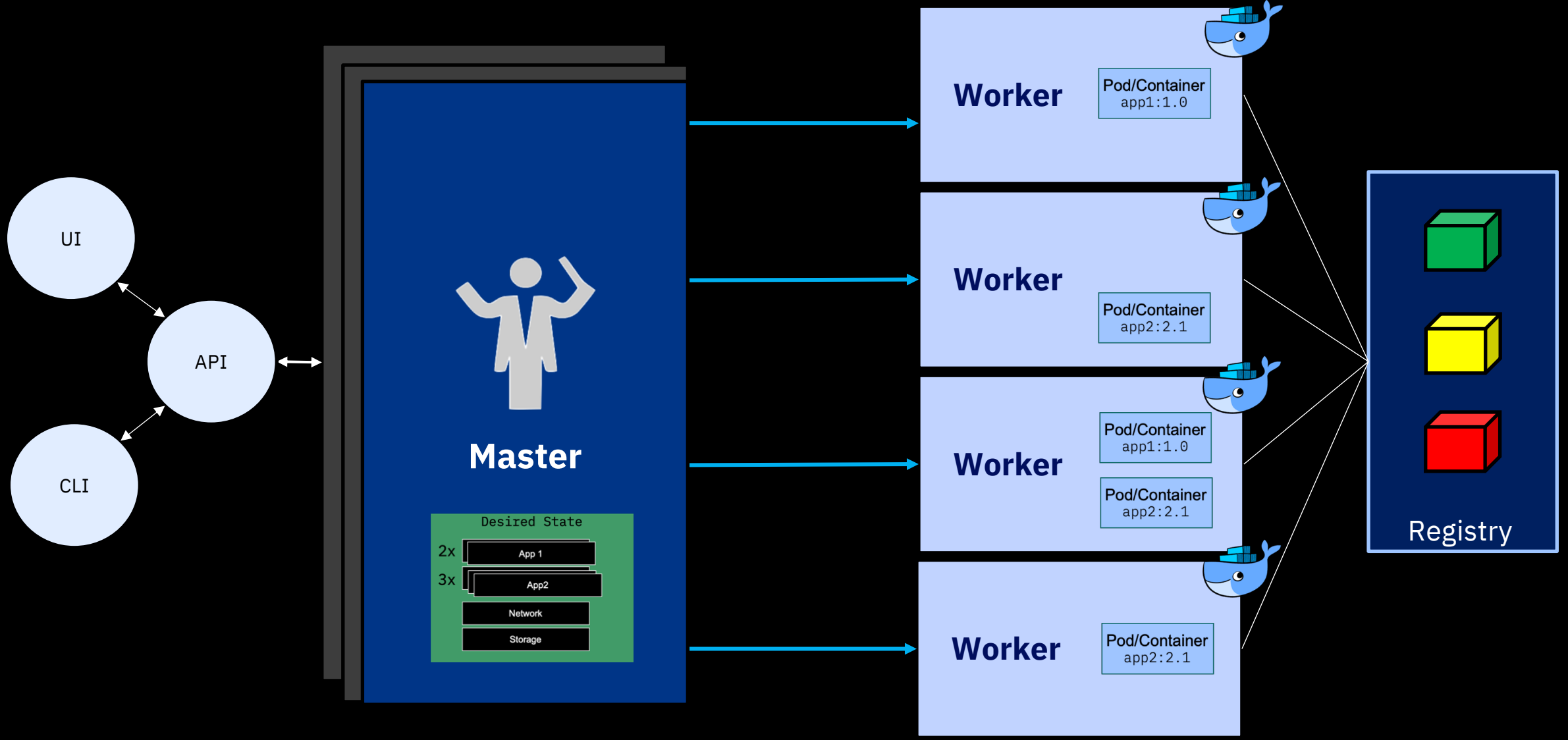
Open source project managed by the Linux Foundation

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

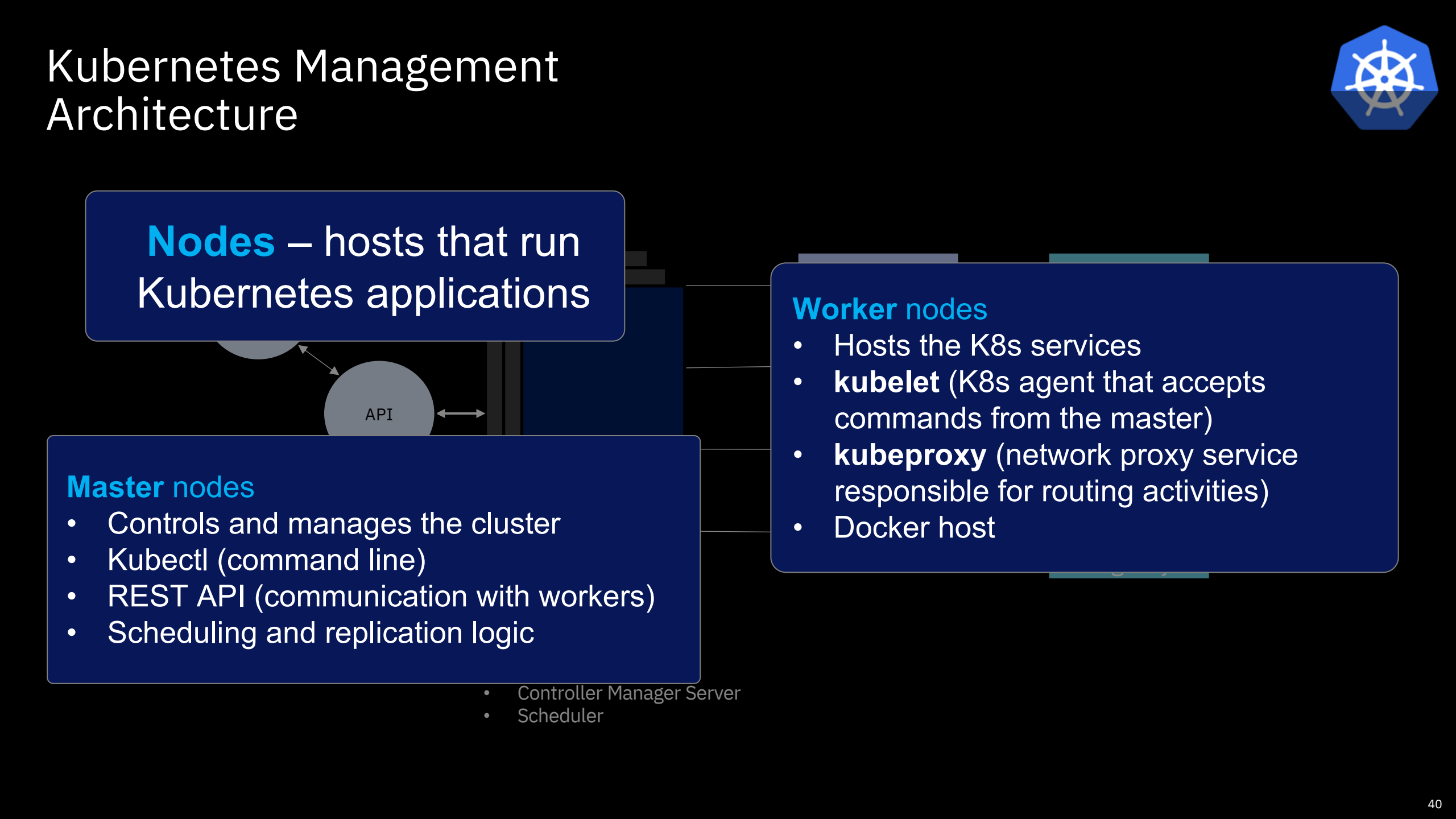
Kubernetes Strengths

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

Kubernetes Management Architecture



Kubernetes Management Architecture



Nodes – hosts that run Kubernetes applications

API

Master nodes

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

- Controller Manager Server
- Scheduler

Worker nodes

- Hosts the K8s services
- **kubelet** (K8s agent that accepts commands from the master)
- **kubeproxy** (network proxy service responsible for routing activities)
- Docker host

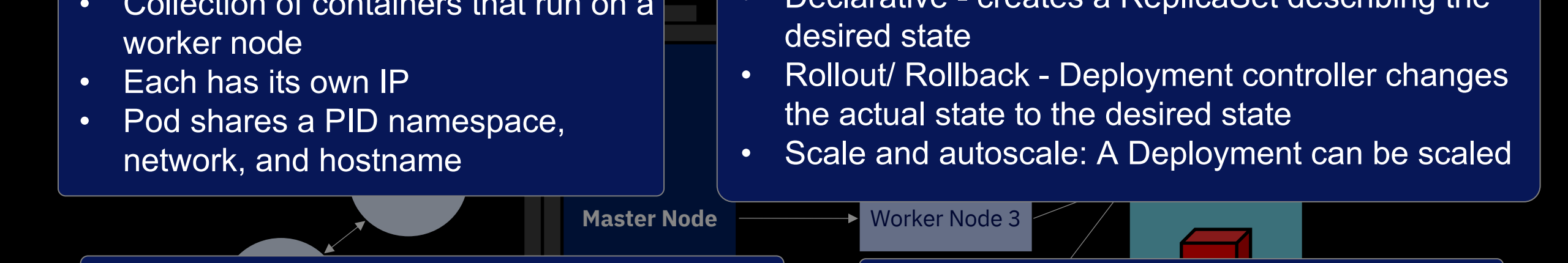
Kubernetes Management Architecture

Pods

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

Deployment

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



ReplicationSet

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

Service

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

Kubernetes Management Architecture

ConfigMap

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

Labels

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

Secrets

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

API

Worker Node 3

Worker Node n

Registry

- Etcd
- API Server
- Controller Manager Server
- Scheduler

What does Kubernetes really offer ?

Intelligent Scheduling

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

Self Healing

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

Horizontal Scaling

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

Service Discovery and Load Balancing

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

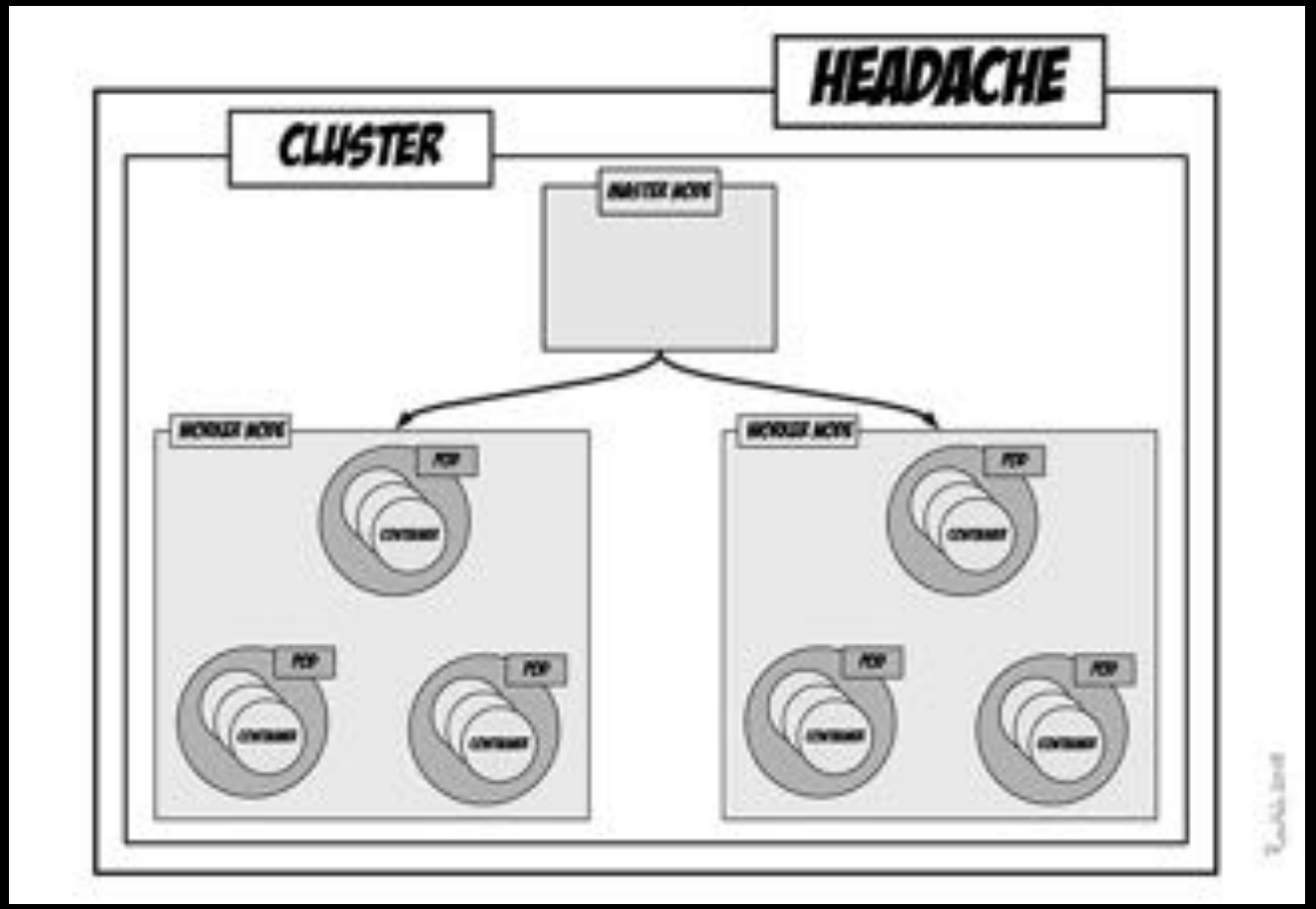
Automated rollout and rollback

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

Secret and configuration management

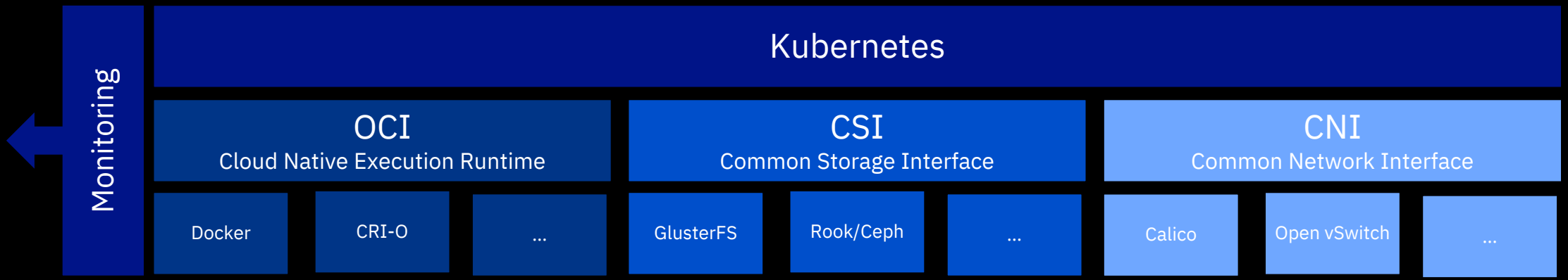
Deploy and update secrets and application configuration without rebuilding your image and without exposing secrets in your stack configuration.

Kubernetes
Assemble vs. Operate



Source: ROELBOB - https://devops.com/kubernetes-in-the-real-world/

Kubernetes Cluster Layers



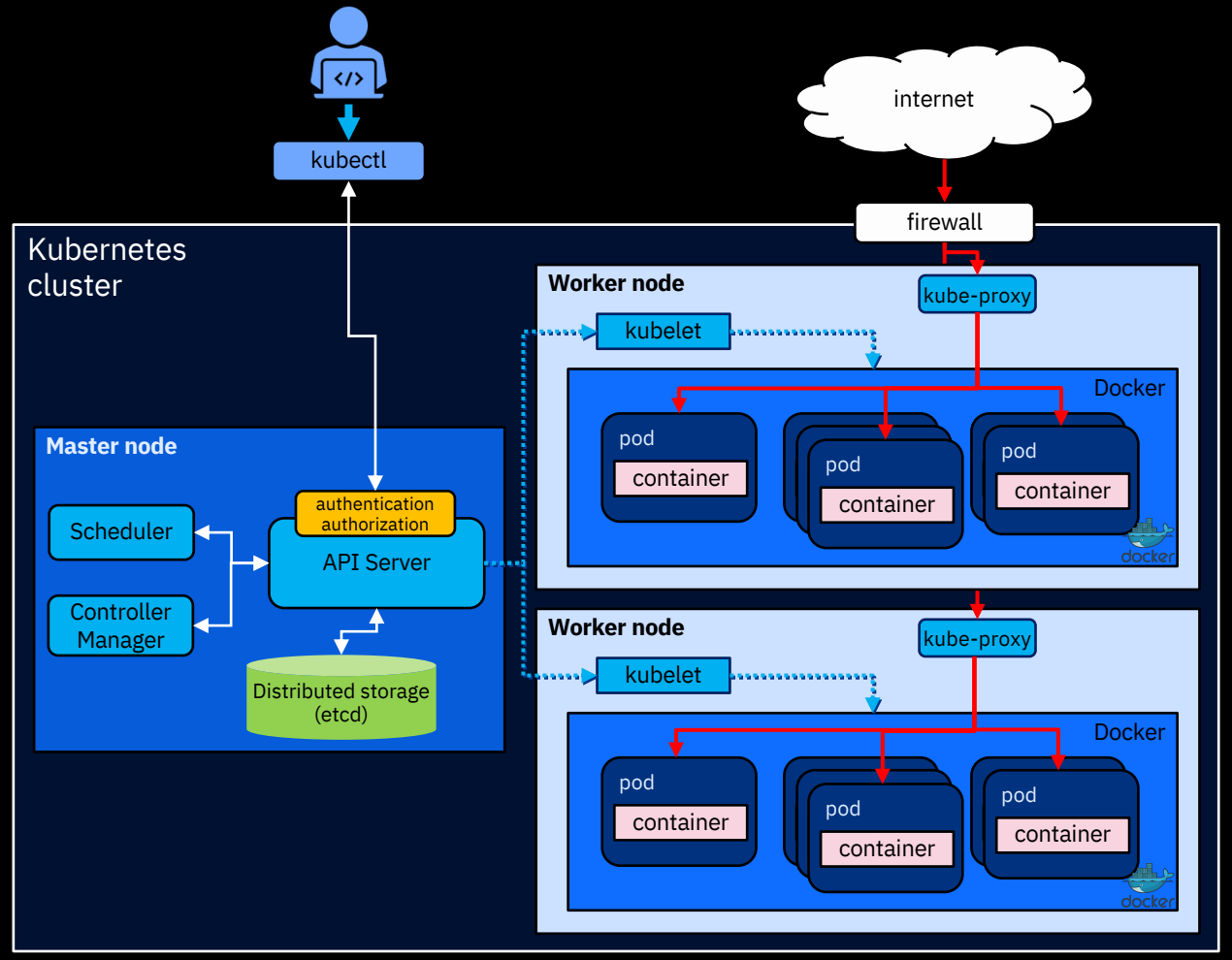
Kubernetes Cluster Architecture

Master node

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

Worker nodes

- Node where pods are run
- Docker engine
- kubelet agent accepts & executes commands from the master to manage pods
- kube-proxy – routes inbound or ingress traffic



Master Node Components

Etcd

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

API Server

- Exposes API for managing Kubernetes
- Used by kubectrl CLI

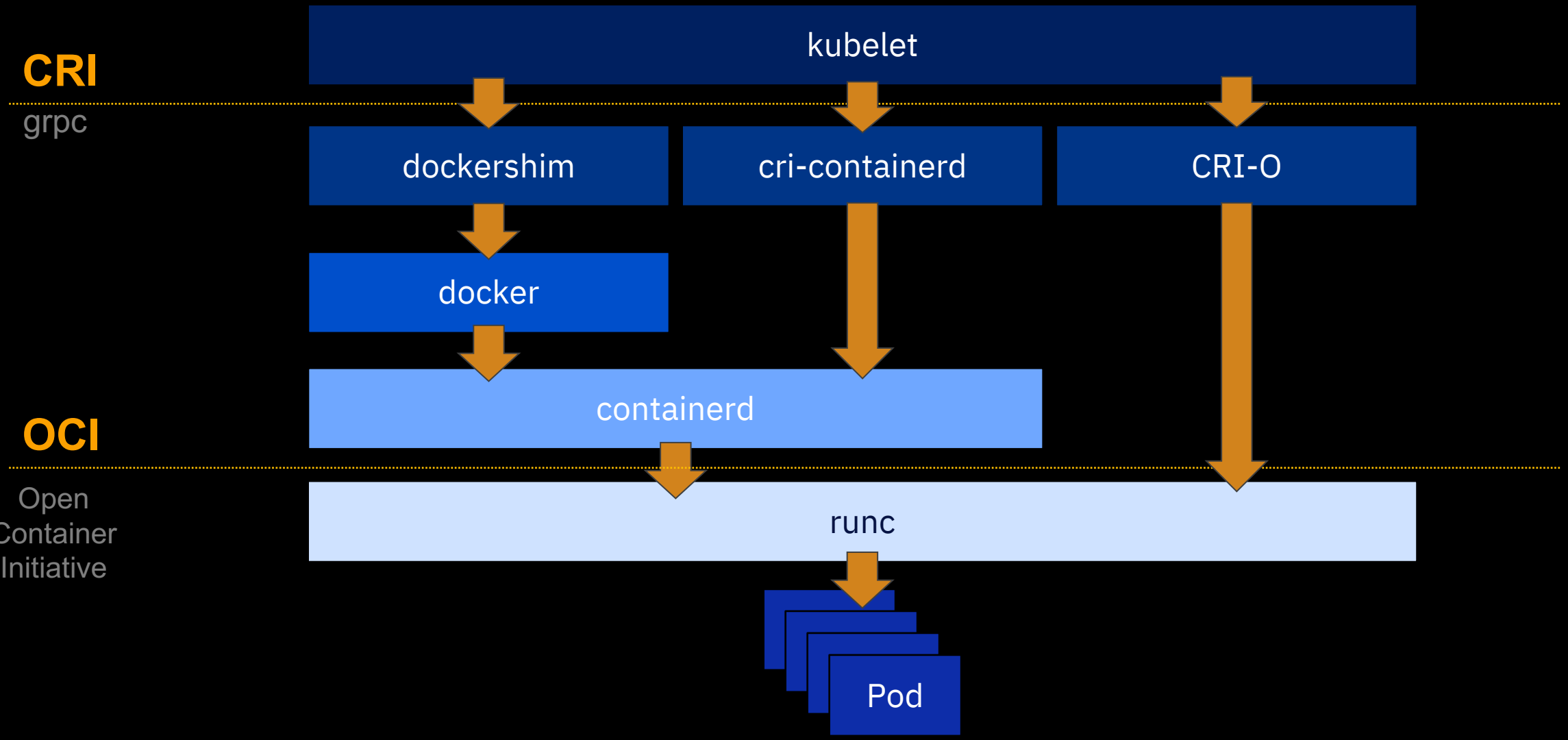
Controller manager

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

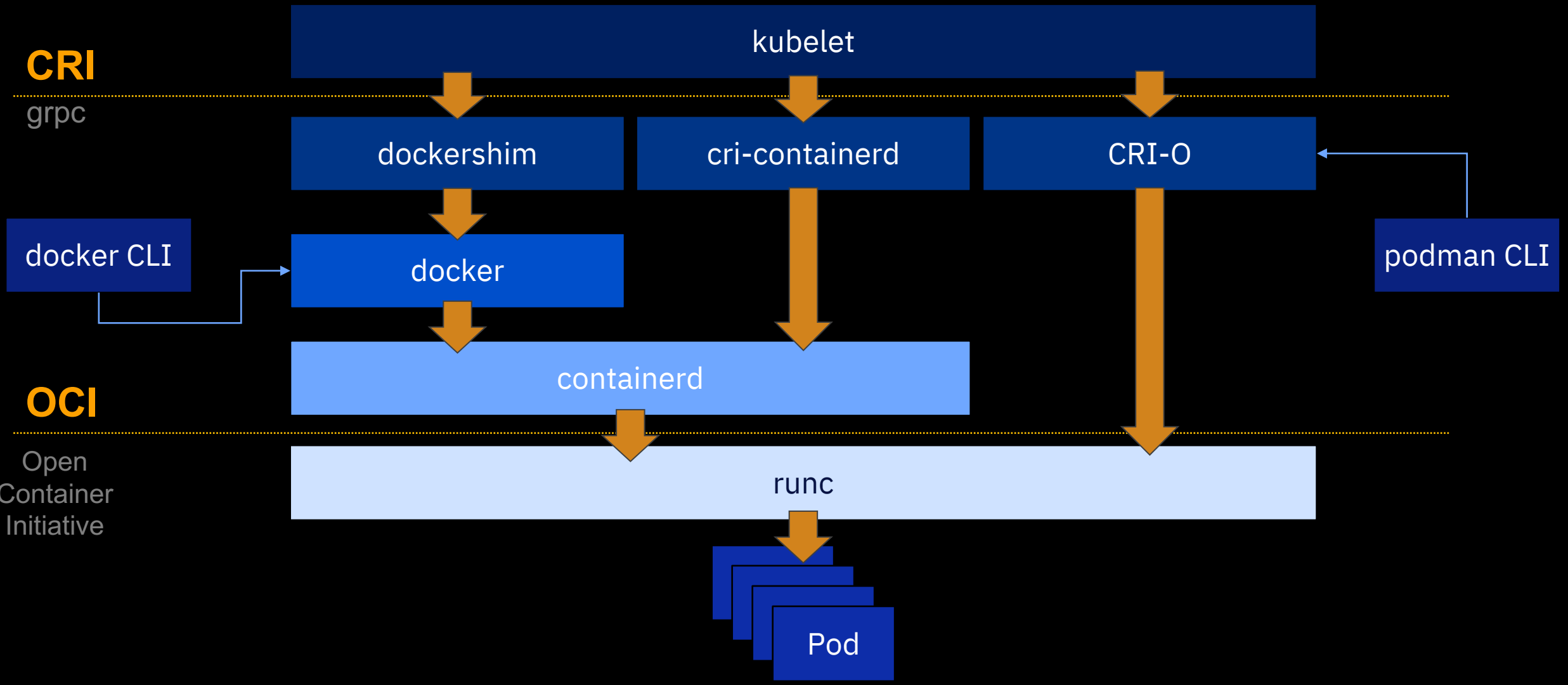
Scheduler

- Selects the worker node each pods runs in

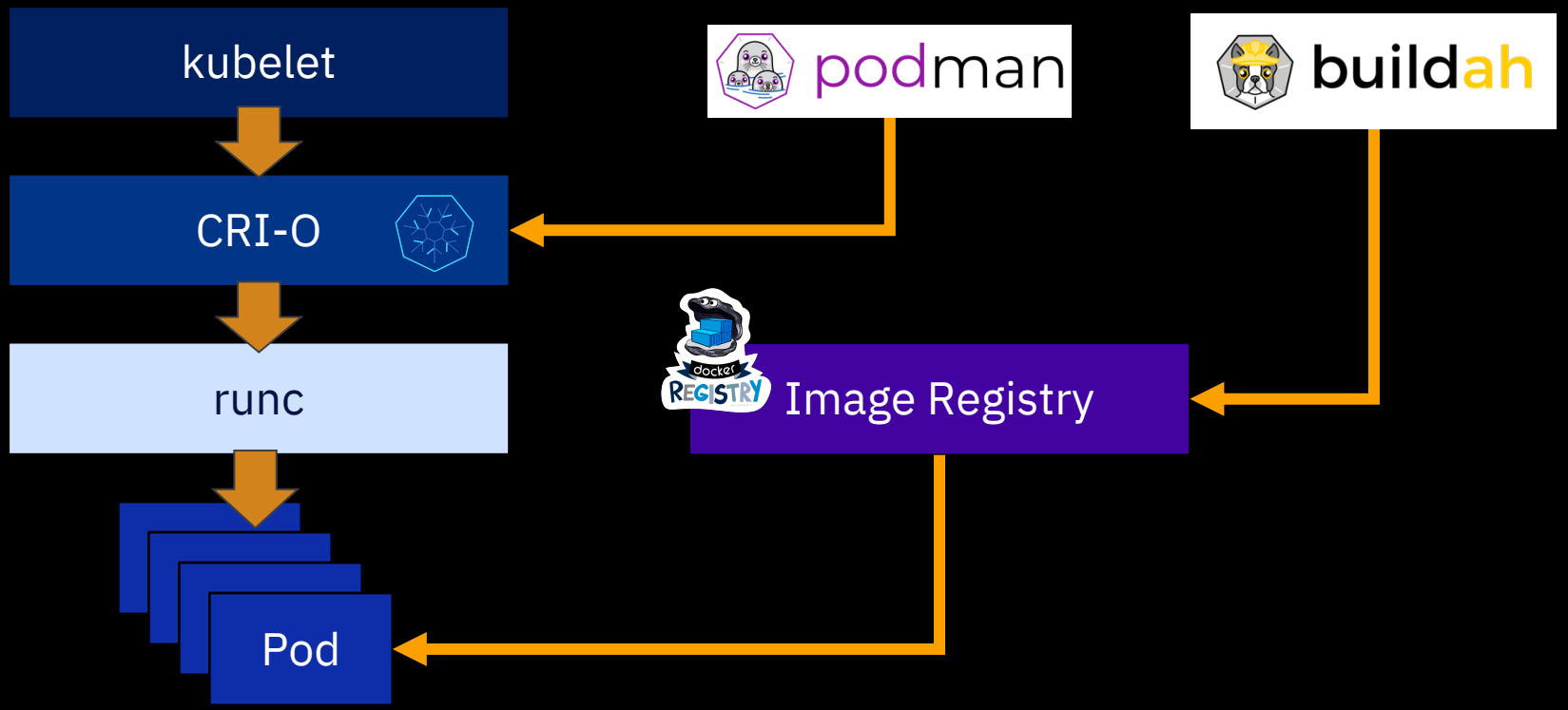
Kubernetes – Common Runtime Interface



Kubernetes – Common Runtime Interface



Kubernetes – CRI-O



Kubernetes – Why CRI-O?

cri-o

- **A Truly Open Project:** Operated as part of the broader Kubernetes community. Contributors from companies including Red Hat, SUSE, Intel, Google, Alibaba, IBM and more.
- **Lightweight:** CRI-O is made of lots of small components. In comparison, the Docker Engine is a heavyweight daemon which is communicated to using the docker CLI tool in a client/server fashion.
- **More Securable:** As CRI-O containers are children of the process that spawned it (not the daemon) they're fully compatible with tooling like cgroups & security constraints to provide an extra layer of protection to your containers. Every container run using the Docker CLI is a 'child' of that large Docker Daemon. This complicates or outright prevents the use of this tooling.
- **Aligned with Kubernetes:** As an official Kubernetes project, CRI-O releases in lock step with Kubernetes, with similar version numbers. ie. CRI-O 1.11.x works with Kubernetes 1.11.x.

Kubernetes 1.16 APIs

As the Kubernetes API evolves, APIs are periodically reorganized or upgraded. When APIs evolve, the old API is deprecated and eventually removed..

Deprecated APIs in 1.16

- **NetworkPolicy**
  Migrate from extensions/v1beta1 to networking.k8s.io/v1 AP
- **PodSecurityPolicy**
  Migrate from extensions/v1beta1to policy/v1beta1 API
- **DaemonSet, Deployment, StatefulSet, and ReplicaSet**
  Migrate from extensions/v1beta1 or apps/v1beta2 to apps/v1 API

Remediation

- Rewrite or
- `kubectl convert -f ./my-deployment.yaml --output-version apps/v1`

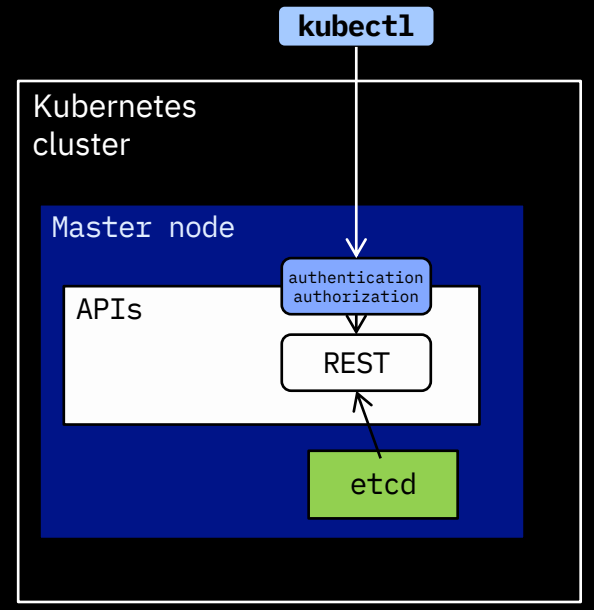
Source: https://kubernetes.io/blog/2019/07/18/api-deprecations-in-1-16/

kubectl – talking to the Cluster

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

`kubectl` looks for a file named config in the $HOME/.kube directory.
It contains all the information needed to communicate with your cluster.

```
training@ubuntu:~/training/demo-app/k8sdemo$ kubectl config view
apiVersion: v1
clusters:
- cluster:
    certificate-authority: /home/training/.minikube/ca.crt
    server: https://172.17.0.4:8443
  name: minikube
contexts:
- context:
    cluster: minikube
    user: minikube
  name: minikube
current-context: minikube
kind: Config
preferences: {}
users:
- name: minikube
  user:
    client-certificate: /home/training/.minikube/profiles/minikube/client.crt
    client-key: /home/training/.minikube/profiles/minikube/client.key
```

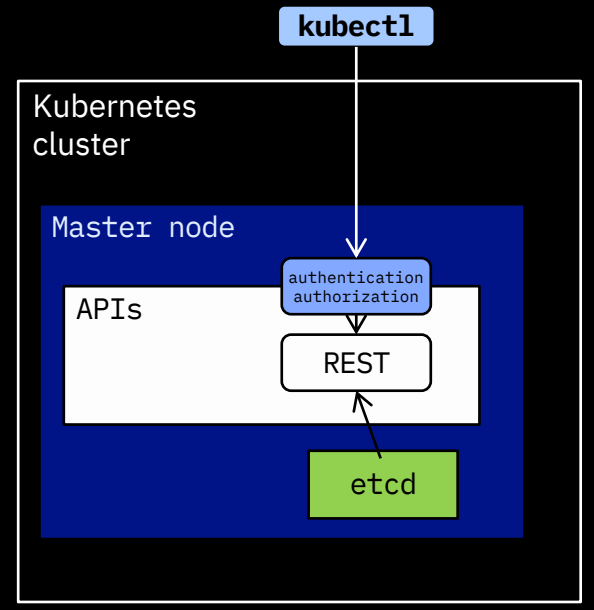



kubectl – talking to the Cluster

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

kubectl [command] [TYPE] [NAME]

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

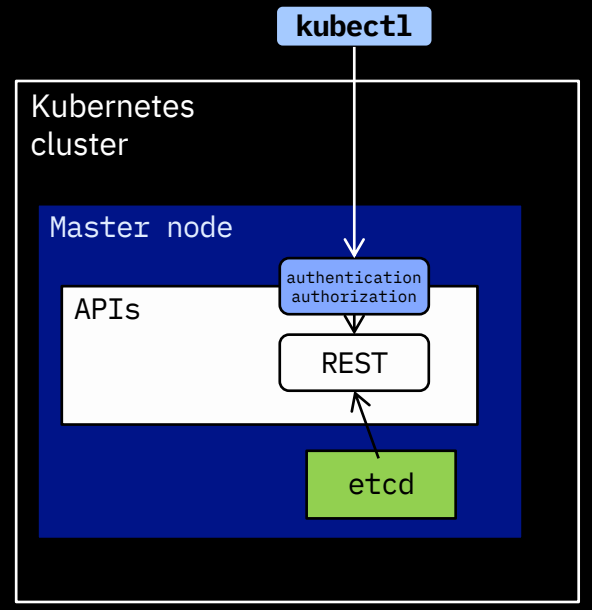


kubectl – talking to the Cluster

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

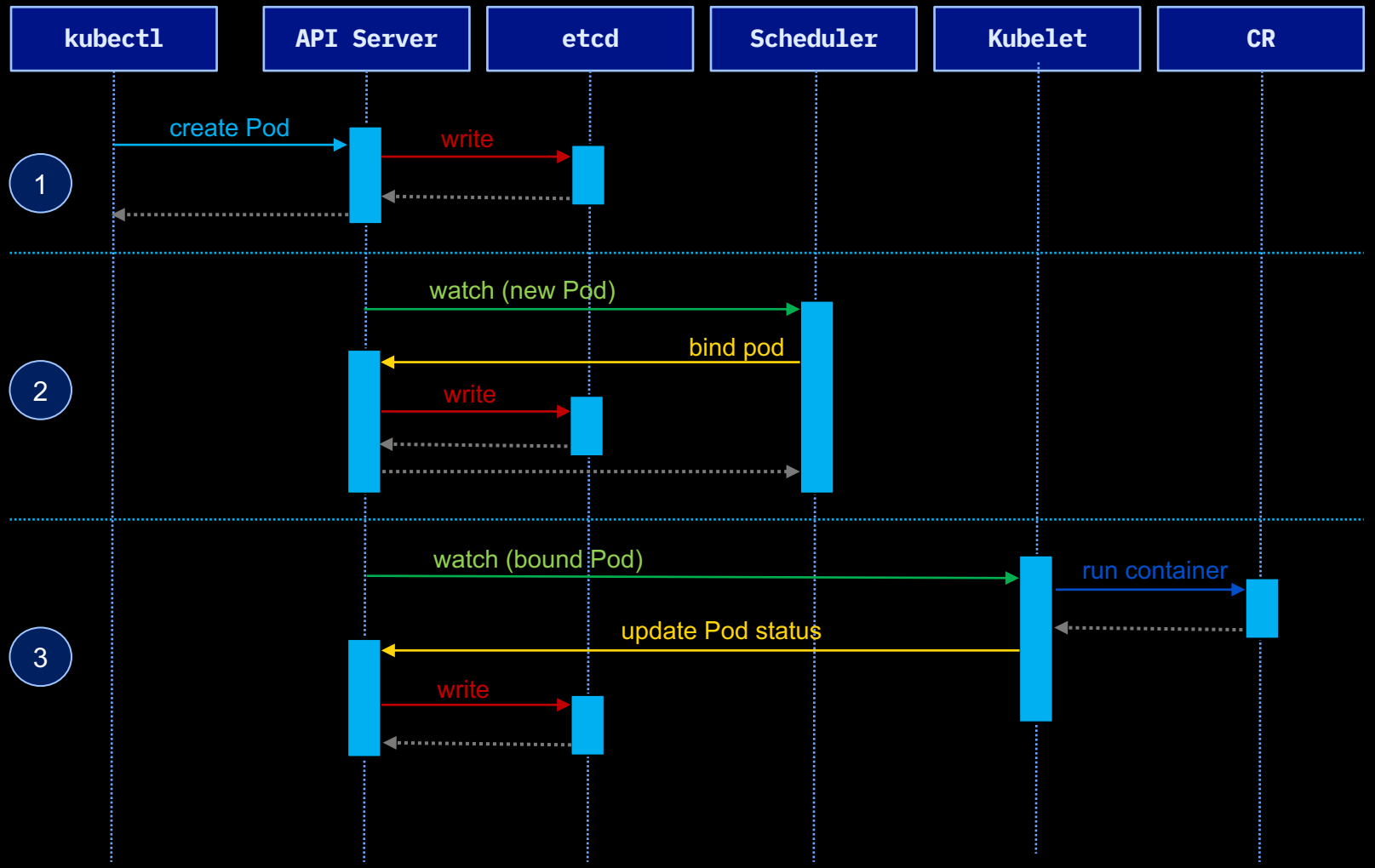
- Kubernetes uses etcd as a key-value database store.
- It stores the configuration of the Kubernetes cluster in etcd.
- It also stores the *actual* state of the system and the *desired* state of the system in etcd.

- Anything you might read from a `kubectl get xyz` command is coming from etcd.
- Any change you make via `kubectl create` will cause an entry in etcd to be updated.



kubectl – talking to the Cluster

What does this actually do: `kubectl run nginx --image=nginx:1.7.9`



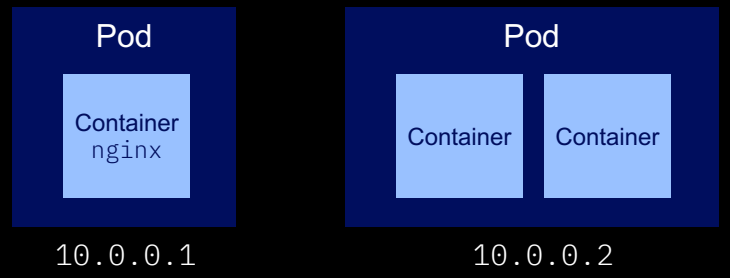
1. Create a Pod via the API Server and the API server writes it to etcd
2. The scheduler notices an “unbound” Pod and decides which node to run that Pod on. It writes that binding back to the API Server.
3. The Kubelet notices a change in the set of Pods that are bound to its node. It, in turn, runs the container via the container runtime (i.e. Docker).

The Kubelet monitors the status of the Pod via the container runtime. As things change, the Kubelet will reflect the current status back to the API Server

Pod

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

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



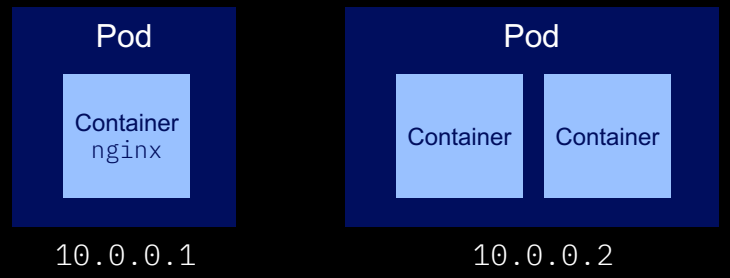
Creating a Pod

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

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

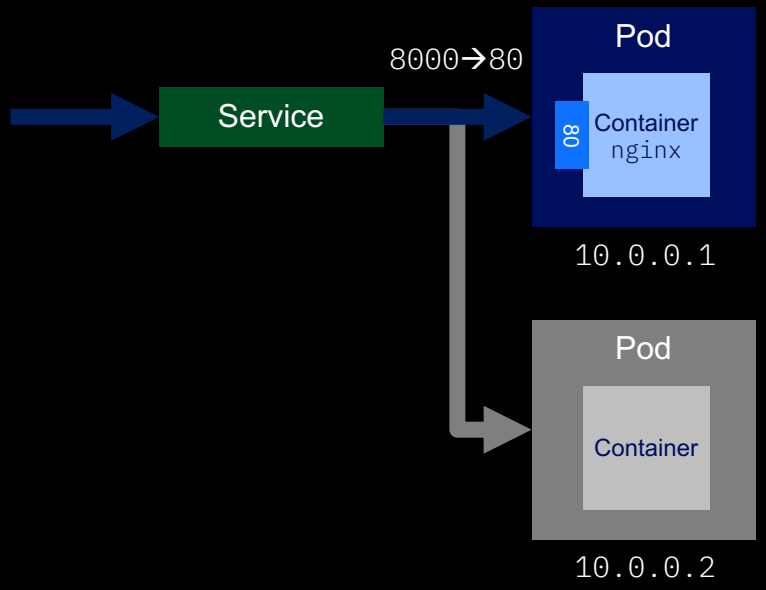
example.yaml

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



Service

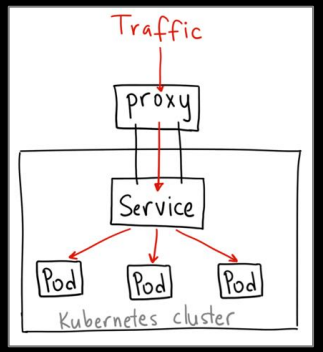
- A service provides a way to refer to a set of Pods (selected by labels) with a single static IP address.
- Creates an entry in the Kubernetes DNS



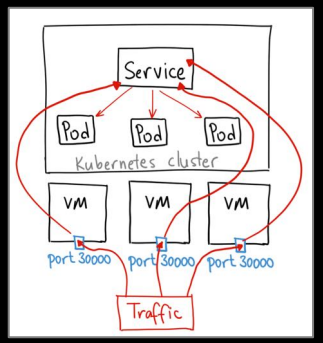

```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  ports:
  - port: 8000
    targetPort: 80
    protocol: TCP
  selector:
    app: nginx
```

Service

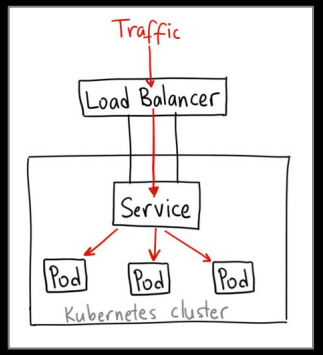
ClusterIP: This type exposes the service on the cluster internal IP. This means that the service is only reachable from within the cluster.



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



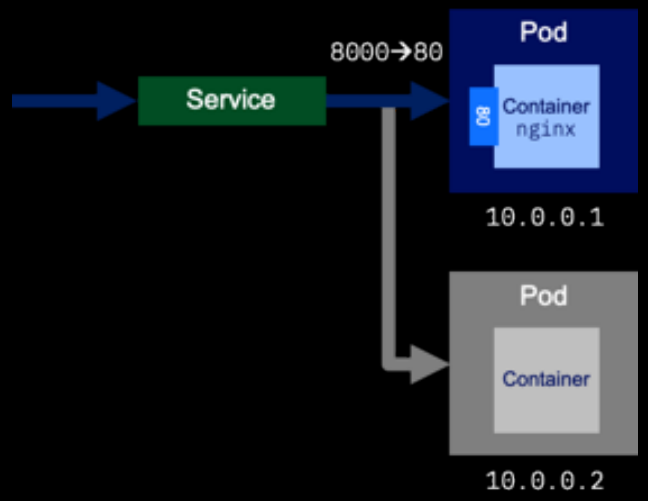
LoadBalancer: This service type exposes a service using the cloud provider load balancer.



Creating a Service

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

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



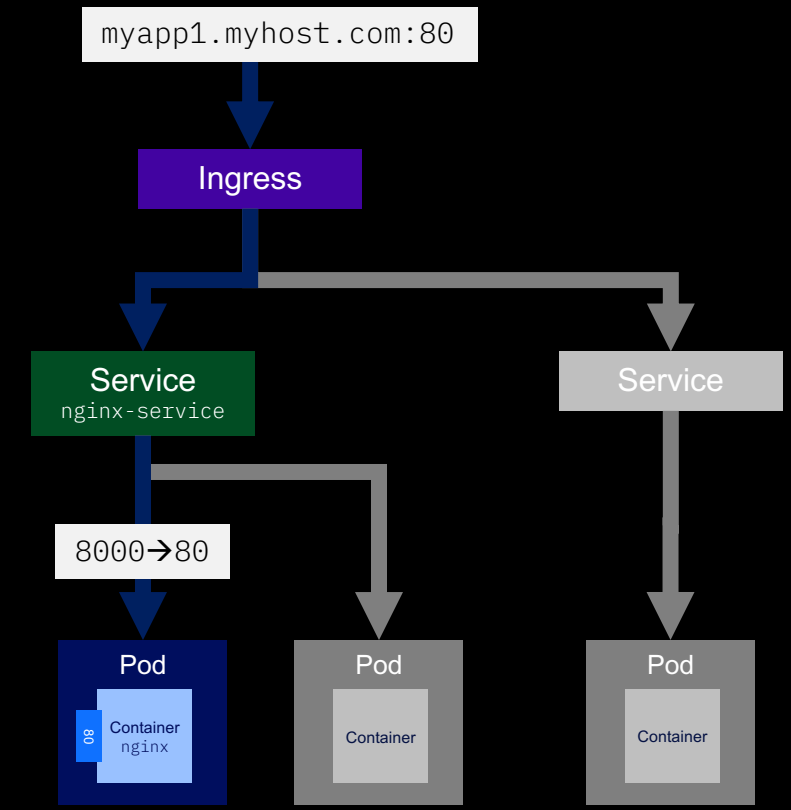
example.yaml

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

Ingress

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

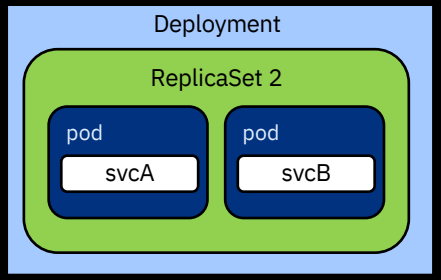
```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: nginx-ingress
  annotations:
    ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - host: myapp1.myhost.com
    http:
      paths:
      - backend:
          serviceName: nginx-service
          servicePort: 80
```



Deployments & ReplicaSets

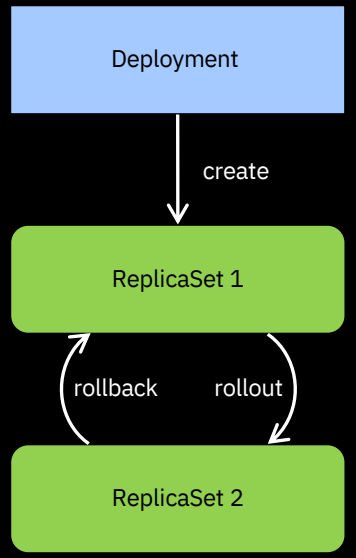
Deployment

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



ReplicaSet

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



Deployment

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

```
kubectl create –f example.yaml
```

example.yaml

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

StatefulSet, DaemonSet, Job...

StatefulSet

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

DaemonSet

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

Job

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

Cron Job

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

Naming

Name

- Each resource object by type has a unique name

Namespace

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

Resource Quota

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

Naming

So a Kubernetes service with the name ***k8s-demo-service*** in namespace ***default*** could be addressed:

When referred from the same namespace

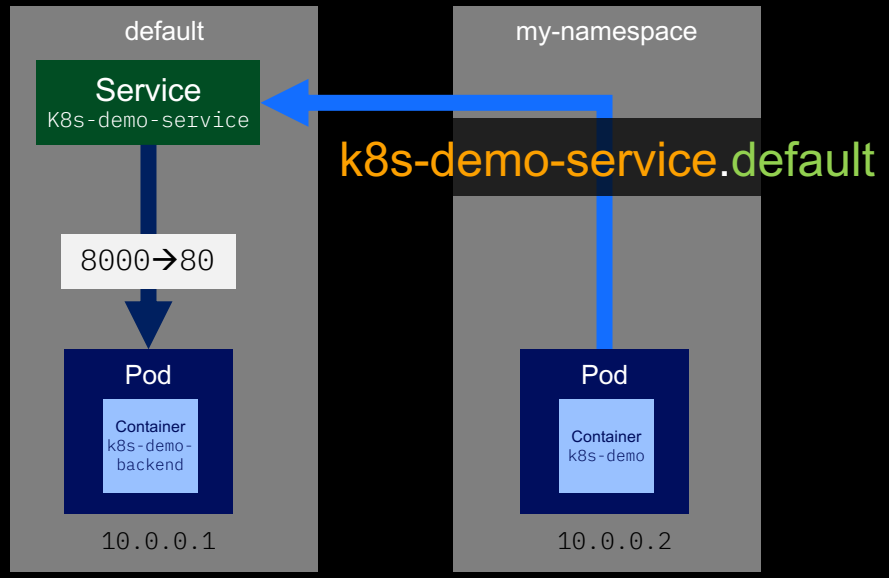
k8s-demo-service

When referred from another namespace

k8s-demo-service.default

Fully qualified name

k8s-demo-service.default.svc.cluster.local



Kubernetes User Security

- **Authentication**
  - OIDC Tokens
  - ServiceAccount Tokens
- **Authorization**
  - RBAC
  - Role
  - RoleBinding
  - ClusterRole
  - ClusterRoleBinding
- **Namespaces**
  - Help to scope access control to a

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

More about this in two weeks

JTC17 – Kubernetes Advanced Security

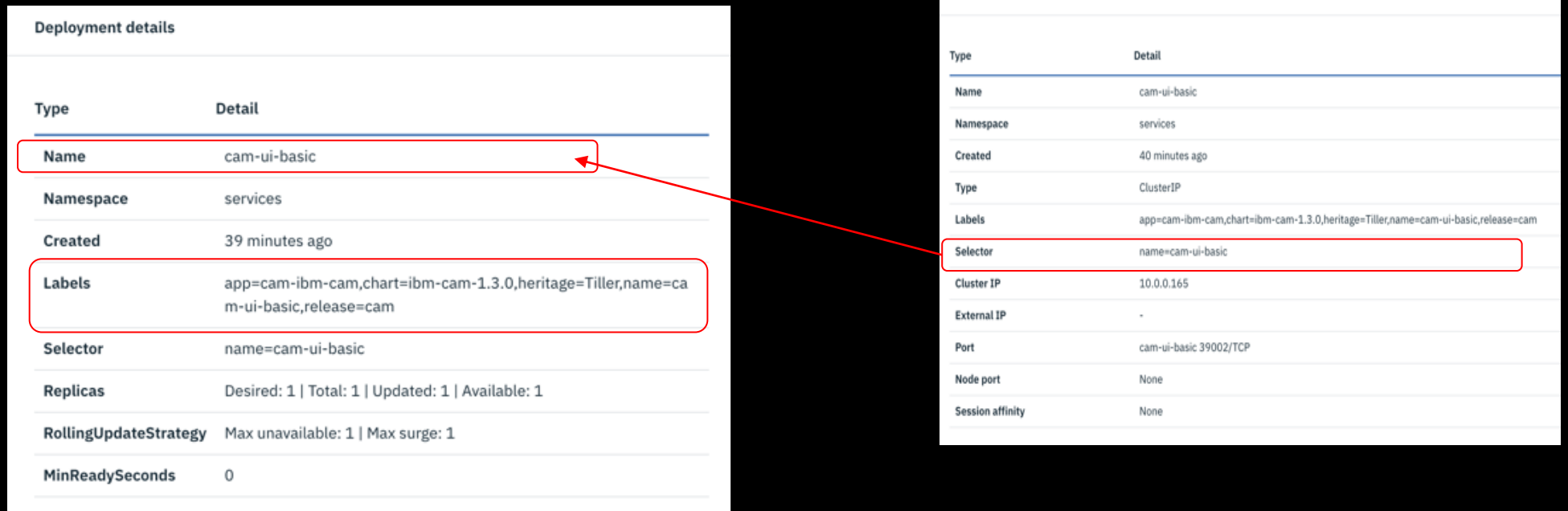
Configuring Resources and Containers

Label

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

Selector

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



Configuring Resources and Containers

ConfigMap

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

Secret

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

Configuring Resources and Containers

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

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

Can be used as normal environment variable (here SPECIAL_LEVEL_KEY=aliens)

Kubernetes Autoscaling

Horizontal Pod Autoscaling (HPA)

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

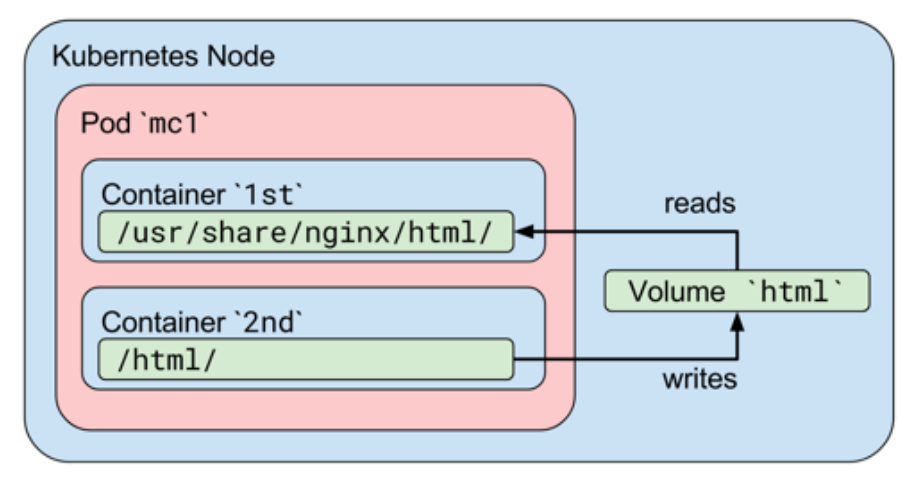
```
$ kubectl autoscale deployment <deployment-name> --cpu-percent=50
  --min=1 --max=10 deployment "<hpa-name>" autoscaled
```

Creates a horizontal pod autoscaler

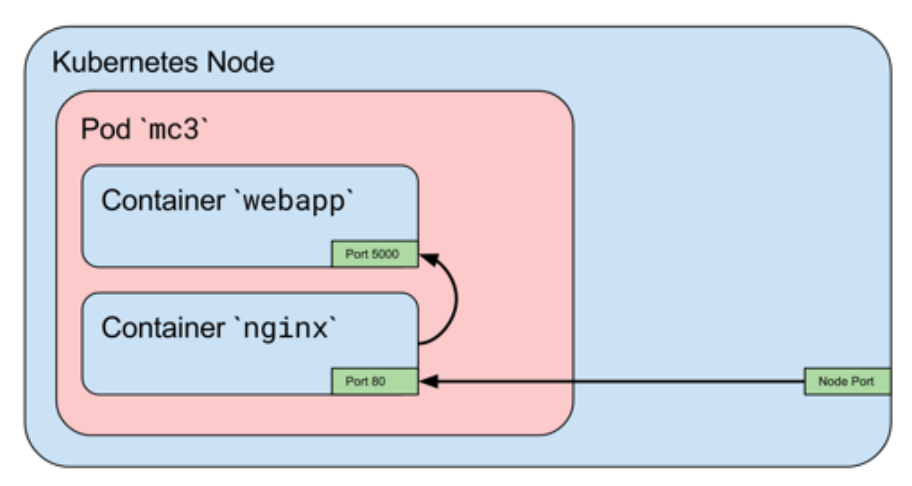
- An HPA instance
- Maintains between 1 and 10 replicas of the pods controlled by the deployment
- Maintains an average CPU utilization across all pods of 50%

Multi-Container Pod Design Patterns

Shared Volumes

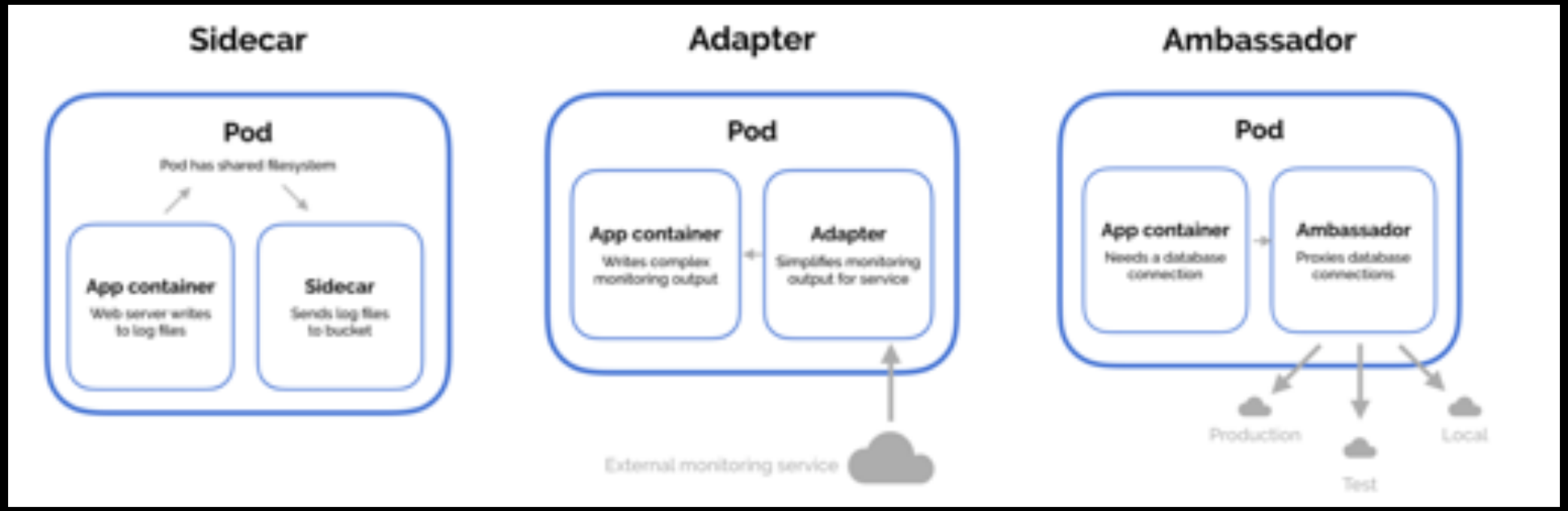


No Network isolation



In what order containers are being started in a Pod?

Multi-Container Pod Design Patterns



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

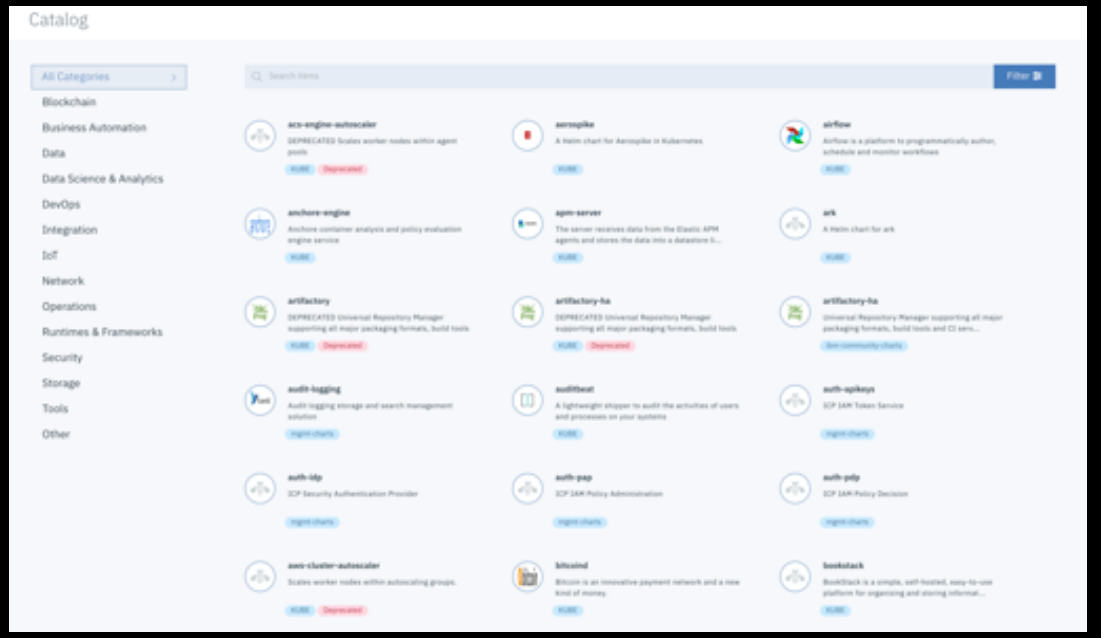
Adapter pattern
The adapter pattern is used to standardize and normalize application output or **monitoring data** for aggregation.

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

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

HELM Charts

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

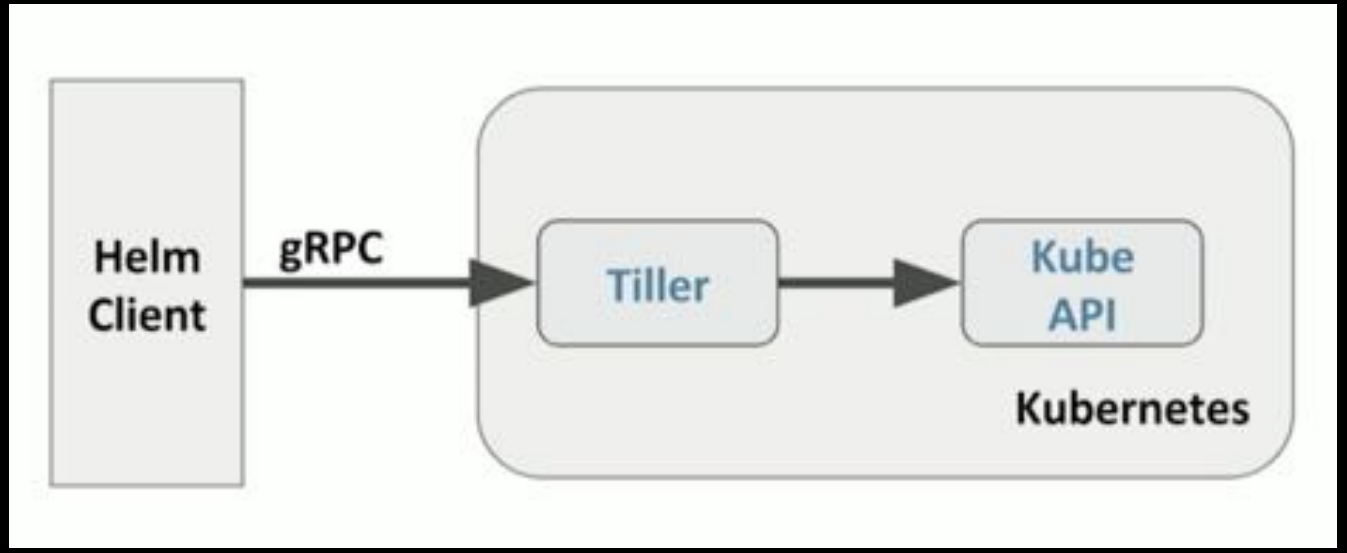


HELM Example

Helm 2 has 2 Components :

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

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

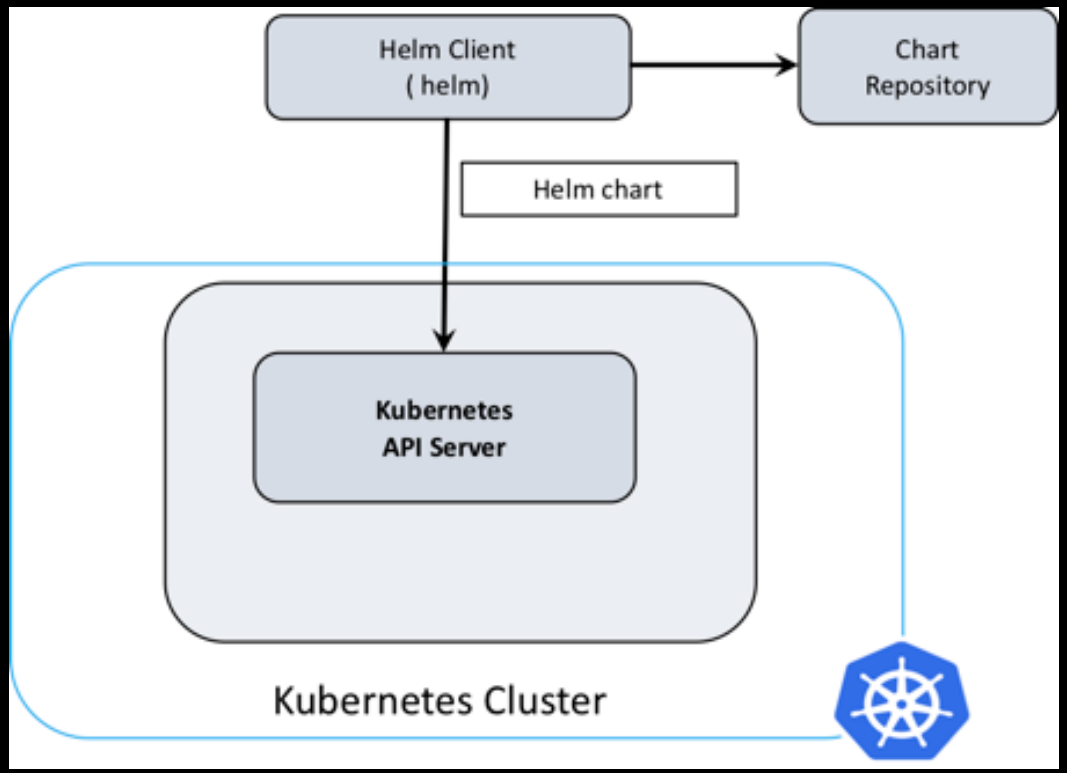


HELM Example

Helm 3 uses Custom Resource Definitions (CRD):

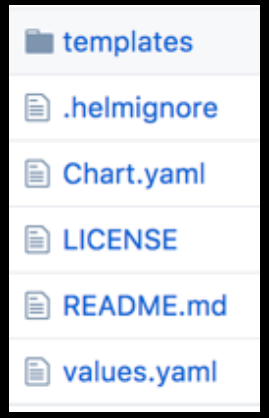
Tillerless : No tiller needed anymore

Client : The Helm client interacts with the cluster and stores information in in-Cluster CRDs.



HELM Example

- Chart :
  A Helm package that contains information sufficient for installing a set of Kubernetes resources into a Kubernetes cluster.
  - Chart.yaml
    Basic information about the chart (name, versions, links, icon, ...)
  - Templates (yaml)
    Kubernetes deployment manifests (yaml) with placeholders for external parameters.
  - values.yaml
    A set of variables that will show up in the UI (or customized at the helm command line) and be passed on at deploy time to the deployment manifests.



Deployment Chart

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: {{ template "fullname" . }}-pv-claim
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: {{ .Values.storage.size }}
```

Values.yaml file

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

Resources

Linux Foundation – Introduction to Kubernetes

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

Kubernetes tutorial

– https://kubernetes.io/docs/tutorials/kubernetes-basics/

Introduction to container orchestration

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

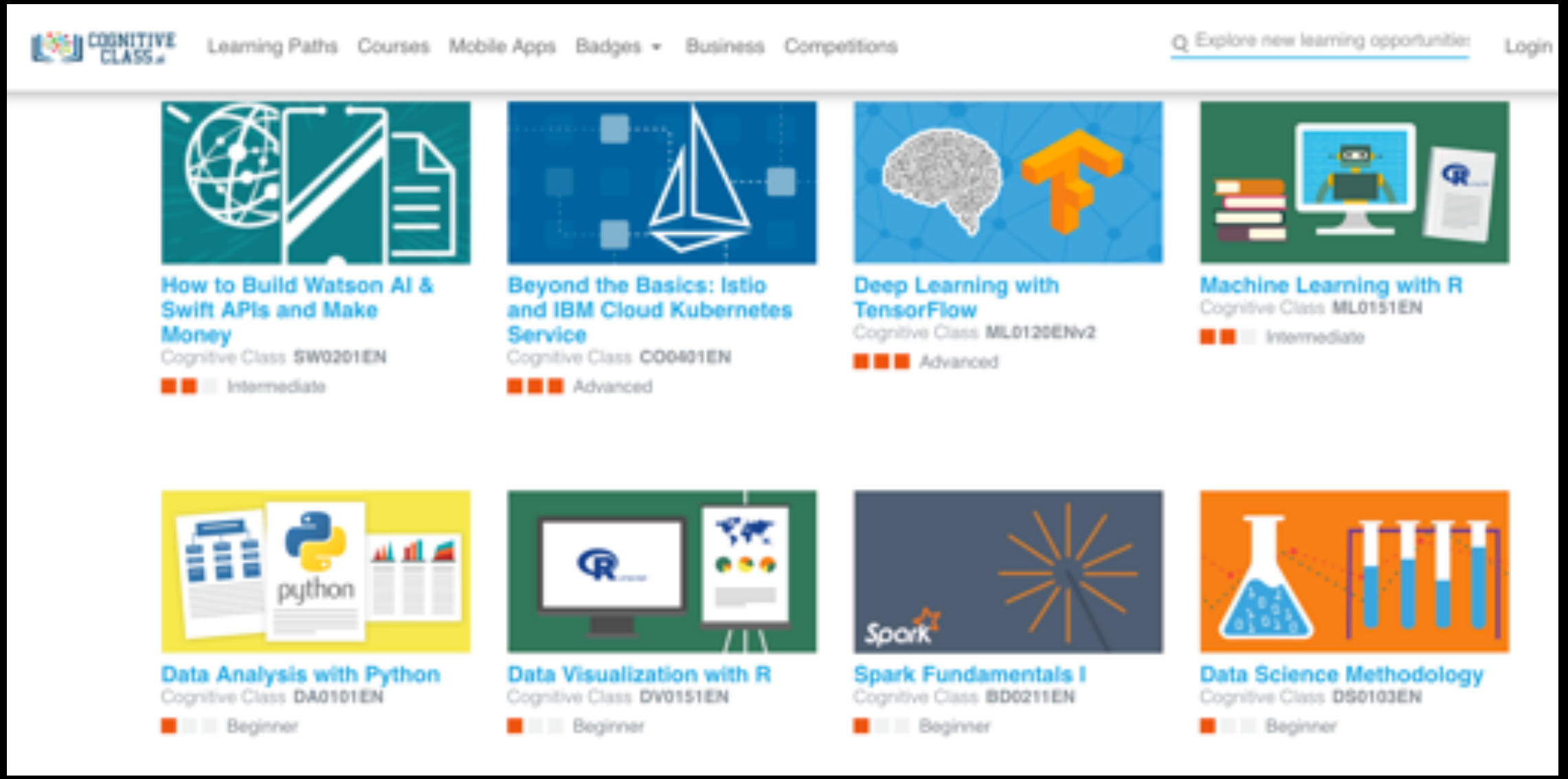
TNS Research: The Present State of Container Orchestration

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

Large-scale cluster management at Google with Borg

– https://research.google.com/pubs/pub43438.html

Cognitiveclass.ai – Free basic to advanced classes on AI, Cloud... etc



Benefits of using a Kubernetes Based Platform

Speed

- Fast boot-time
- Easy scalability
- Rapid deployment

13x

More software releases

Portability

- Between different environments
- Between private and public cloud

Eliminate

“works on my machine” syndrome

Efficiency

- Better usage of computer resources

~47%

Reduction of VMs, OS licensing and server cost

Automation

- Next level standardization and automation

Reduce

operation cost

QUESTIONS?

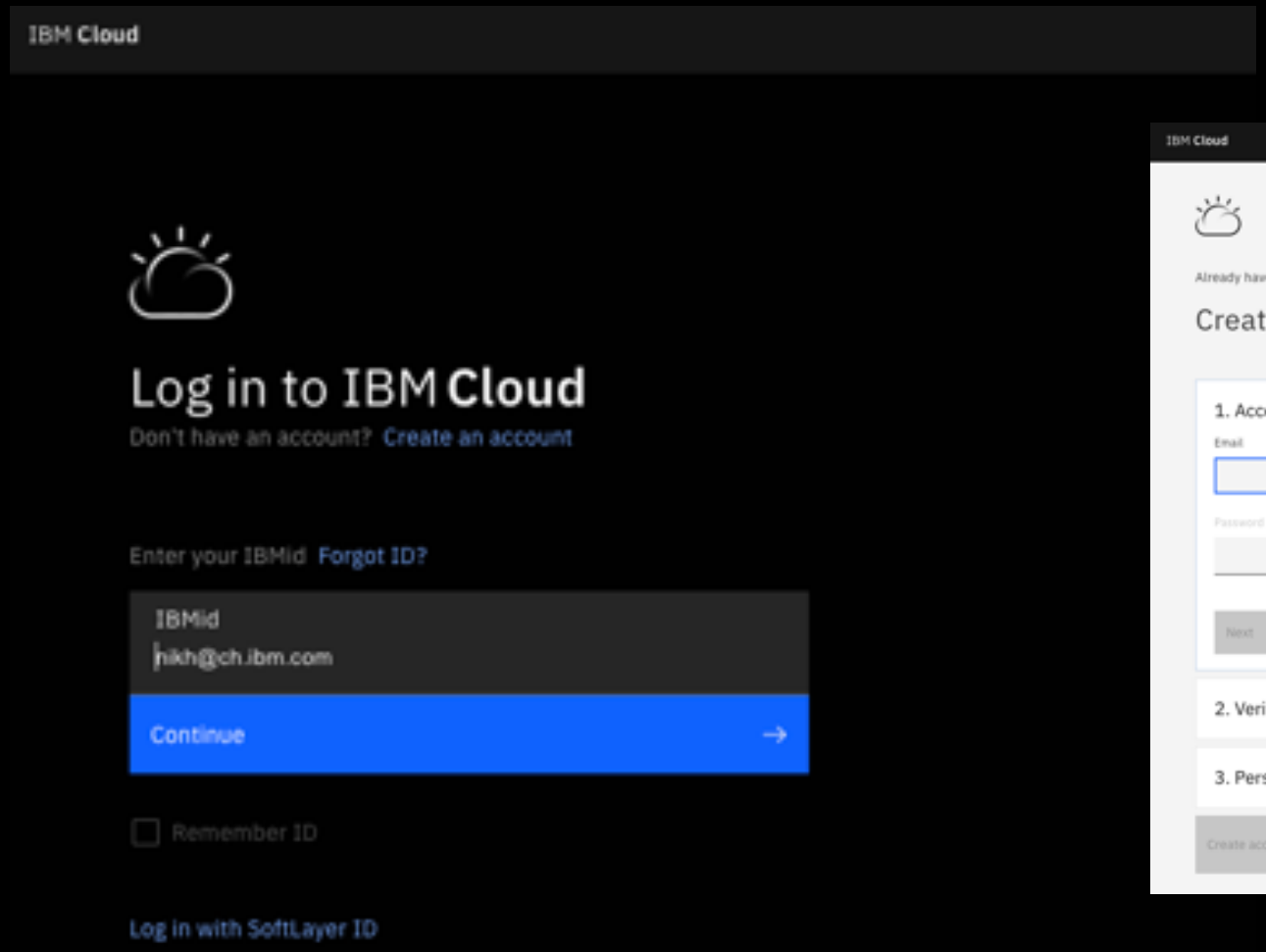
Kubernetes Workshop Series

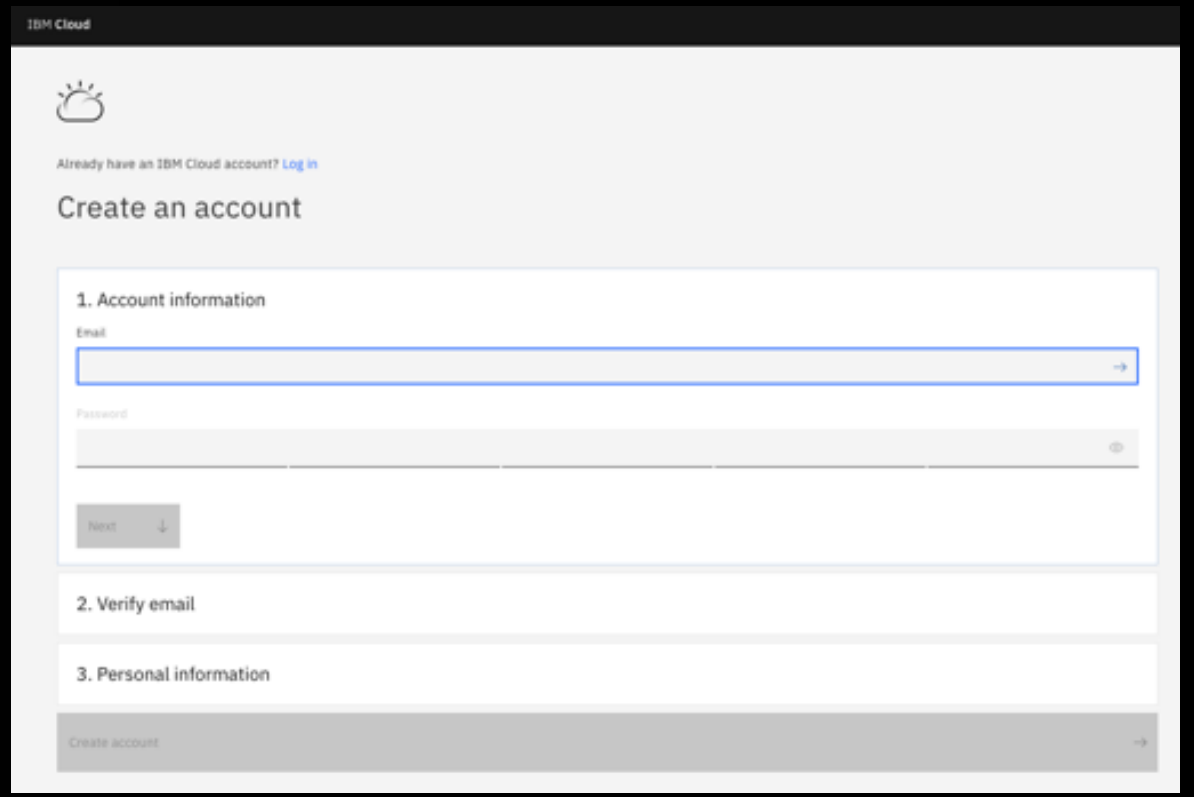
Kubernetes – *Use Kubernetes in the cloud*

02

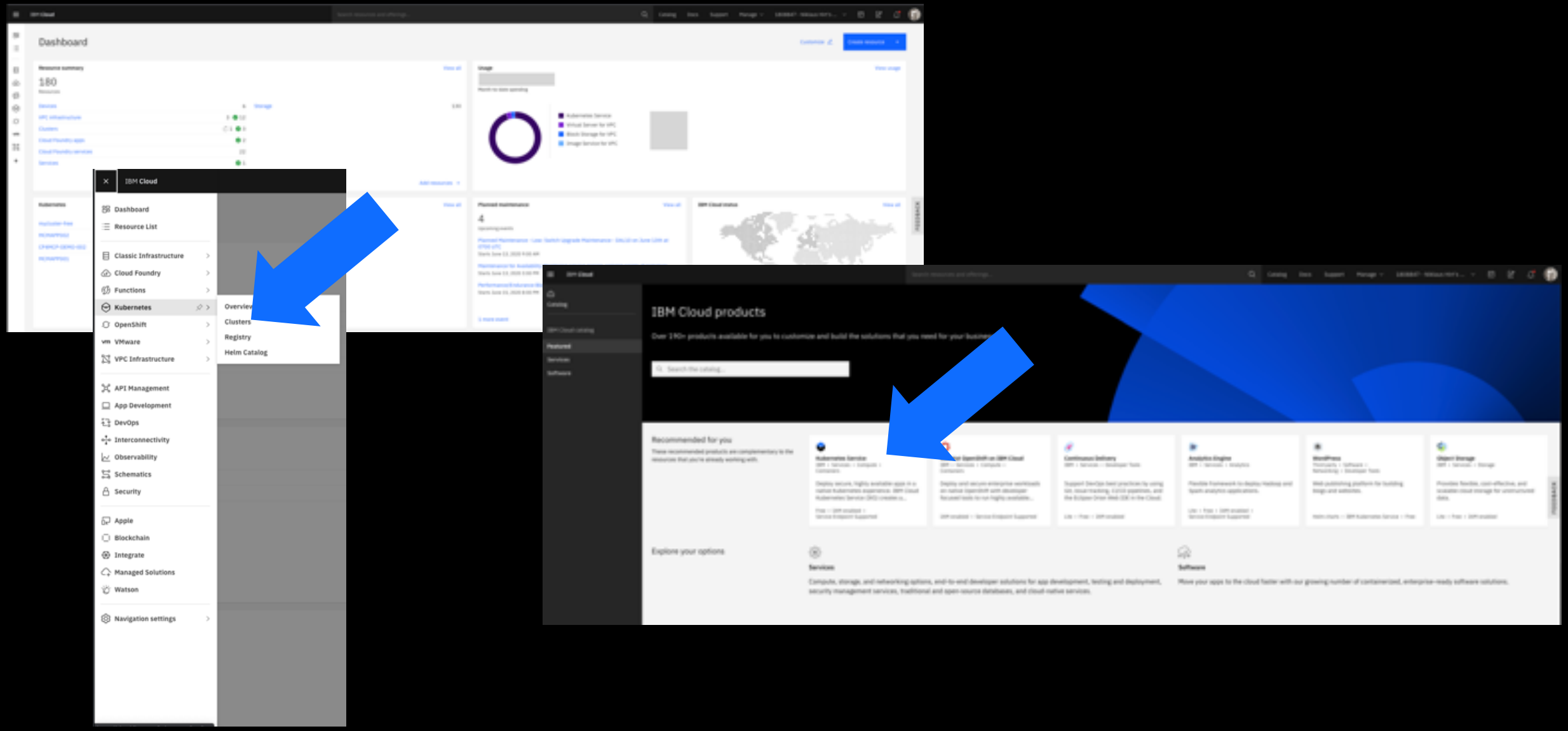
Kubernetes in the Cloud

Go to cloud.ibm.com

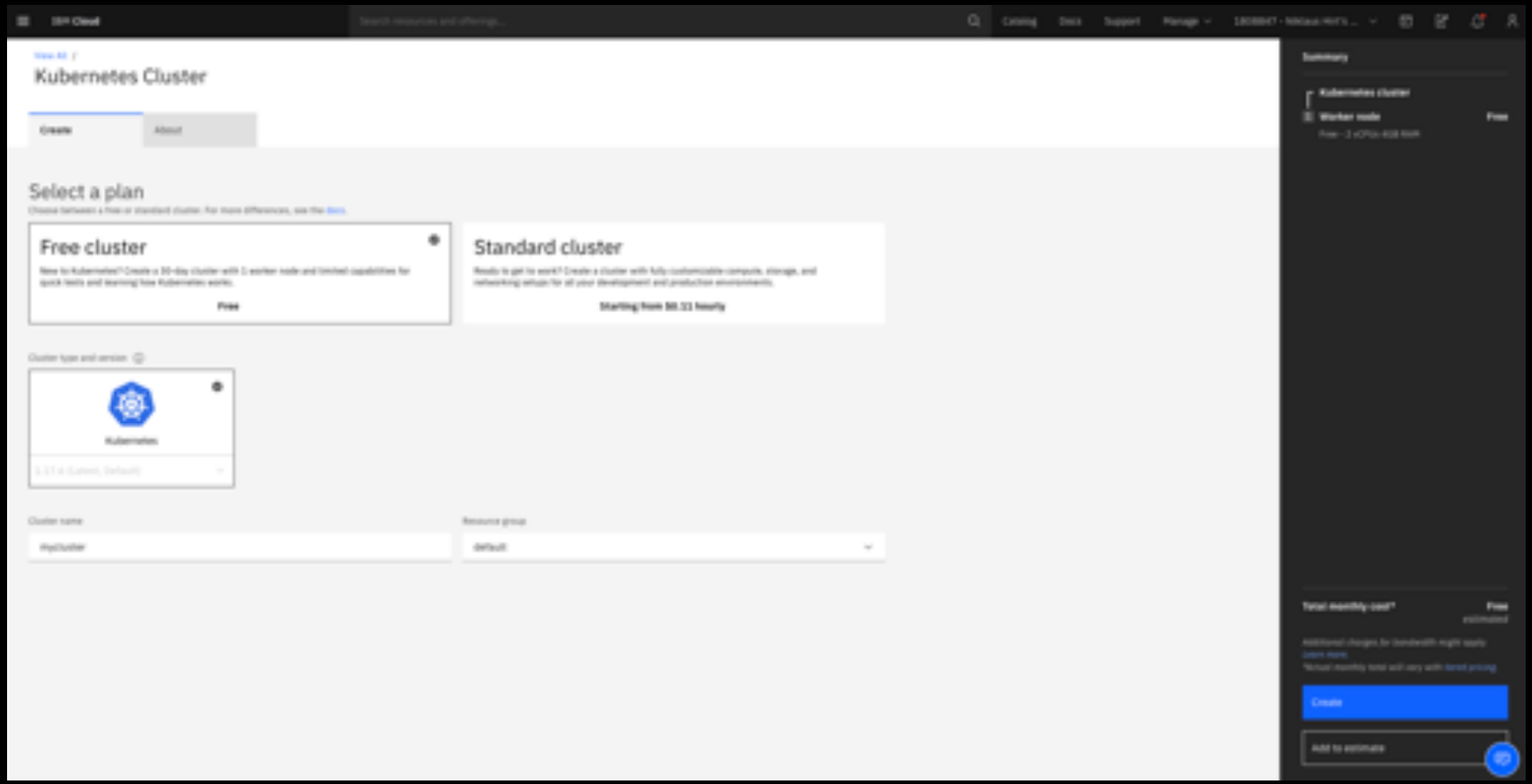




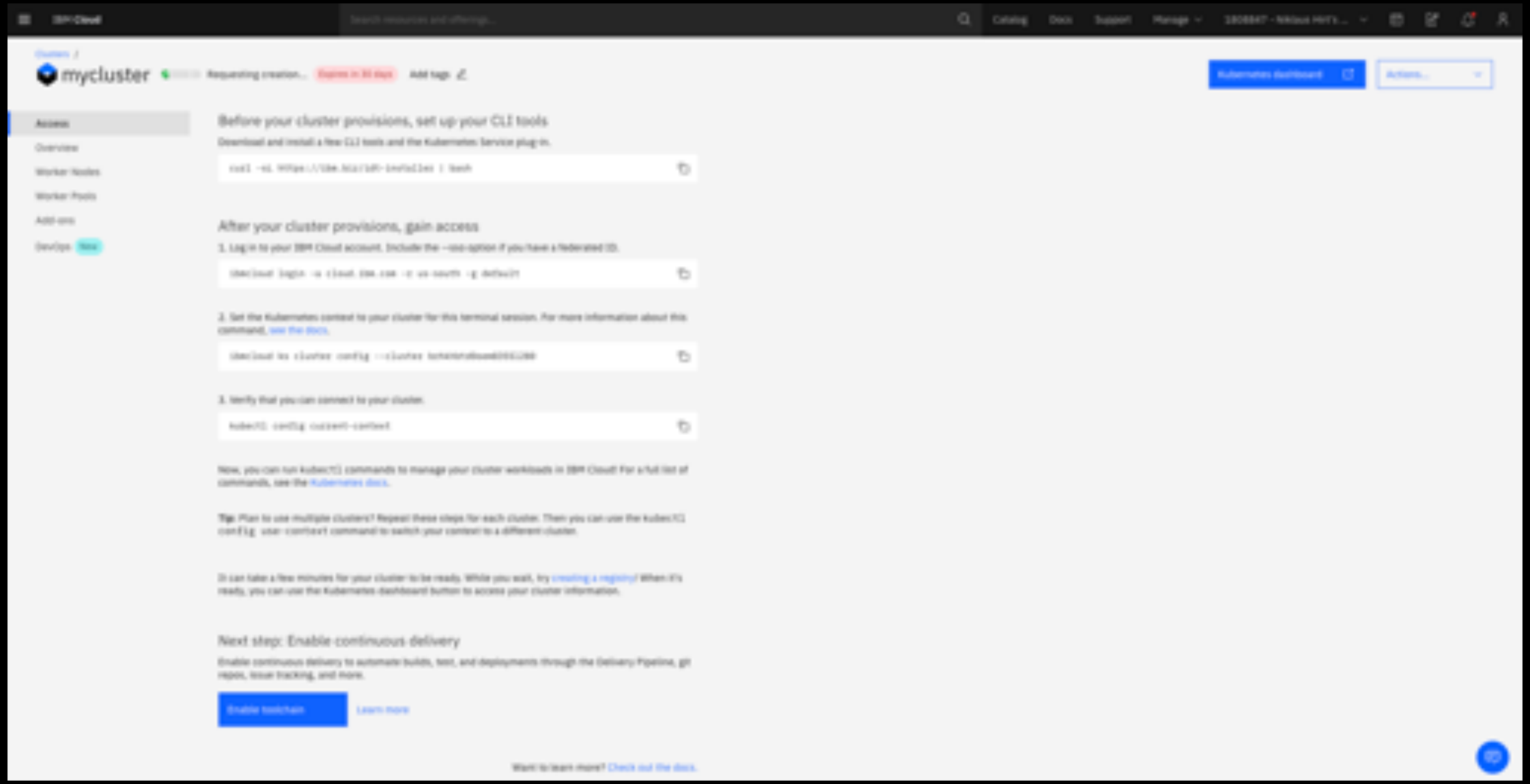
Kubernetes in the Cloud



Kubernetes in the Cloud



Kubernetes in the Cloud



Kubernetes in the Cloud

Let me show you...

QUESTIONS?

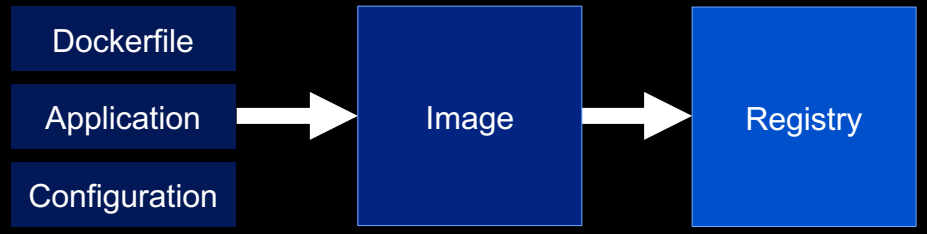
Kubernetes Workshop Series

Kubernetes - *Applied*

03

..but how do I use this for real?

Minimal Application – Container Image




```
FROM node:8-stretch

# Change working directory
WORKDIR "/app"

# Update packages and install dependency packages for services
RUN apt-get update \
&& apt-get dist-upgrade -y \
&& apt-get clean \
&& echo 'Finished installing dependencies'

# Install npm production packages
COPY package.json /app/
RUN cd /app; npm install --production

COPY . /app

ENV NODE_ENV production
ENV BACKEND_URL
https://api.nasa.gov/planetary/apod\?api_key\=DEMO_KEY
ENV PORT 3000

EXPOSE 3000

CMD ["npm", "start"]
```

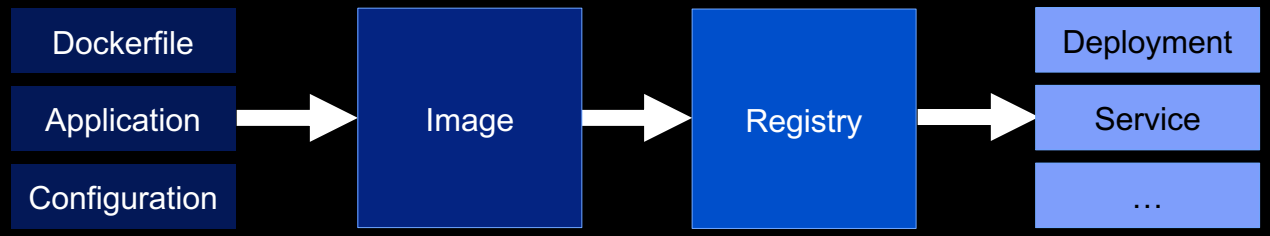
Minimum components you need:

- Built Application or source
  (NodeJS, Java, Golang, ...)
- Dockerfile

  → Use the right base image for your app

```
docker build –t niklaushirt/k8s-demo:1.0 .
docker push niklaushirt/k8s-demo:1.0
```

Minimal Application – Kubernetes Manifests

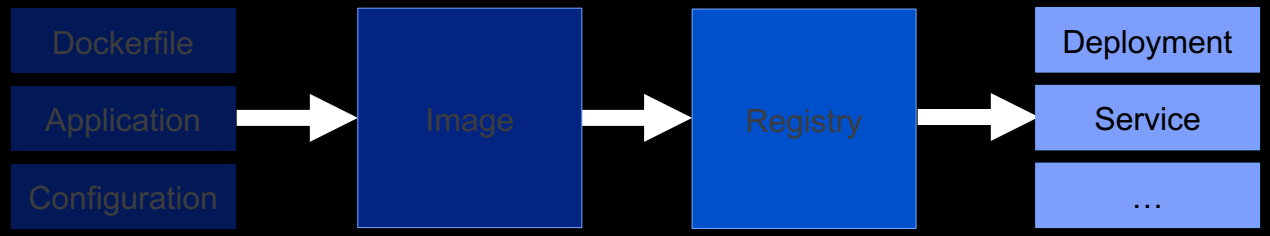



```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: k8s-demo-deployment
  labels:
    app: k8s-demo
spec:
  replicas: 3
  selector:
    matchLabels:
      app: k8s-demo
  template:
    metadata:
      labels:
        app: k8s-demo
    spec:
      containers:
      - name: k8s-demo
        imagePullPolicy: Always
        image: niklaushirt/k8s-demo:1.0
        ports:
        - containerPort: 80
```

Minimum components you need:

- Deployment
- Service

Minimal Application – Kubernetes Manifests

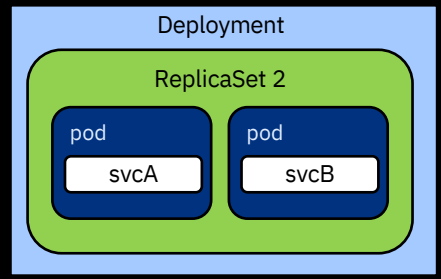


Deployment

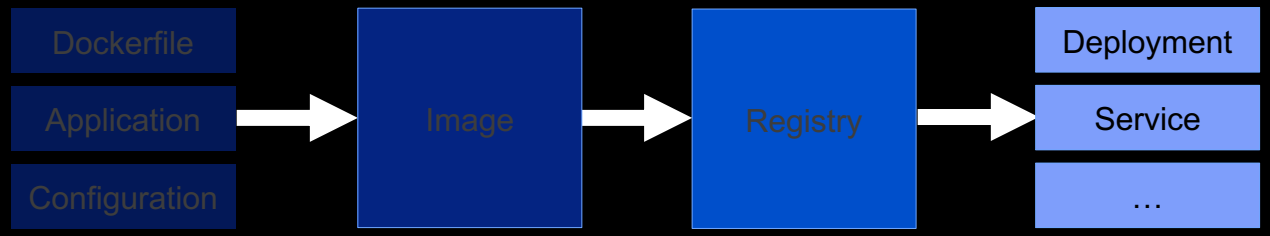
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: k8s-demo-deployment
  labels:
    app: k8s-demo
spec:
  replicas: 3
  selector:
    matchLabels:
      app: k8s-demo
  template:
    metadata:
      labels:
        app: k8s-demo
    spec:
      containers:
      - name: k8s-demo
        imagePullPolicy: Always
        image: niklaushirt/k8s-demo:1.0
        ports:
        - containerPort: 80
```

Minimum components you need:

- Deployment
- Service



Minimal Application – Kubernetes Manifests



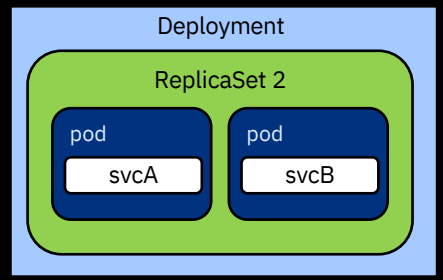
Deployment

ReplicaSet

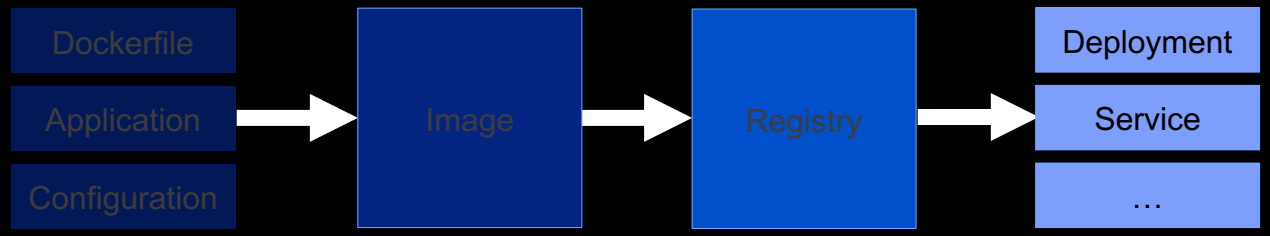
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: k8s-demo-deployment
  labels:
    app: k8s-demo
spec:
  replicas: 3
  selector:
    matchLabels:
      app: k8s-demo
  template:
    metadata:
      labels:
        app: k8s-demo
    spec:
      containers:
      - name: k8s-demo
        imagePullPolicy: Always
        image: niklaushirt/k8s-demo:1.0
        ports:
        - containerPort: 80
```

Minimum components you need:

- Deployment
- Service



Minimal Application – Kubernetes Manifests



Deployment

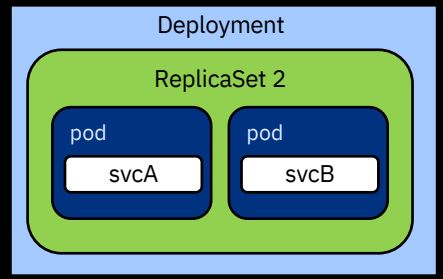
ReplicaSet

Pod

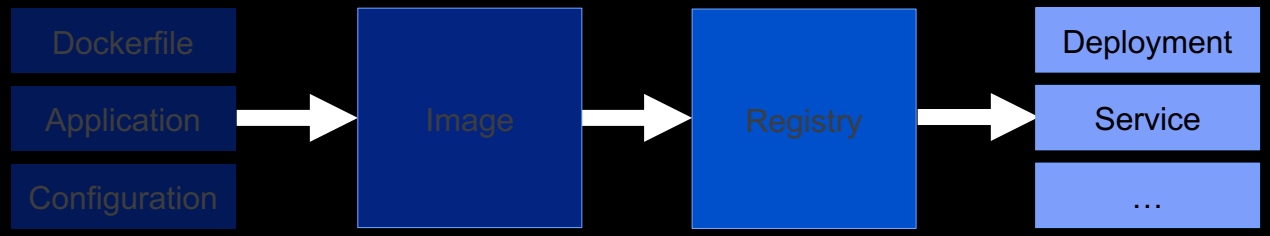
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: k8s-demo-deployment
  labels:
    app: k8s-demo
spec:
  replicas: 3
  selector:
    matchLabels:
      app: k8s-demo
  template:
    metadata:
      labels:
        app: k8s-demo
    spec:
      containers:
      - name: k8s-demo
        imagePullPolicy: Always
        image: niklaushirt/k8s-demo:1.0
        ports:
        - containerPort: 80
```

Minimum components you need:

- Deployment
- Service



Minimal Application – Kubernetes Manifests



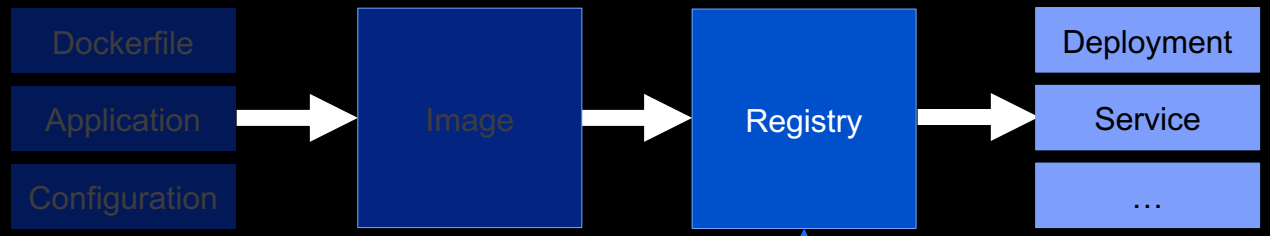

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: k8s-demo-deployment
  labels:
    app: k8s-demo
spec:
  replicas: 3
  selector:
    matchLabels:
      app: k8s-demo
  template:
    metadata:
      labels:
        app: k8s-demo
    spec:
      containers:
      - name: k8s-demo
        imagePullPolicy: Always
        image: niklaushirt/k8s-demo:1.0
        ports:
        - containerPort: 80
```

Minimum components you need:

- Deployment
- Service

Selectors must use the label of the deployment!!

Minimal Application – Kubernetes Manifests




```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: k8s-demo-deployment
  labels:
    app: k8s-demo
spec:
  replicas: 3
  selector:
    matchLabels:
      app: k8s-demo
  template:
    metadata:
      labels:
        app: k8s-demo
    spec:
      containers:
      - name: k8s-demo
        imagePullPolicy: Always
        image: niklaushirt/k8s-demo:1.0
        ports:
        - containerPort: 80
```

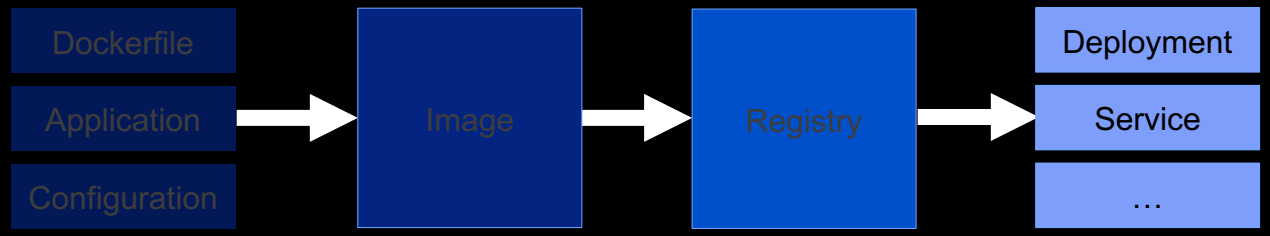
Minimum components you need:

- Deployment
- Service

imagePullPolicy:

IfNotPresent: only pulled if not already present locally
Always: every time the kubelet launches a container

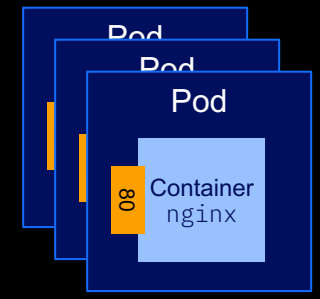
Minimal Application – Kubernetes Manifests



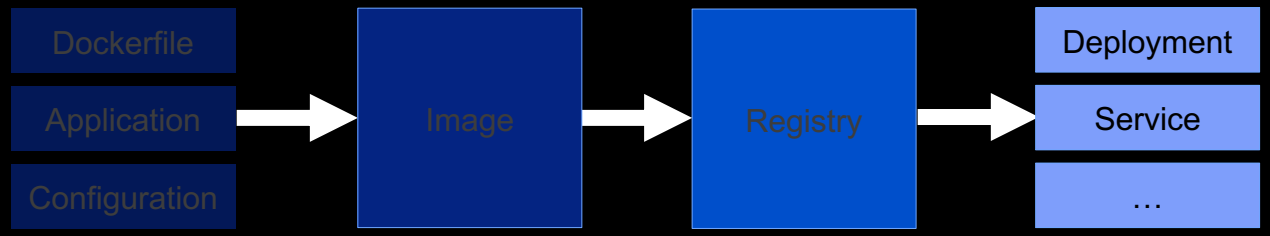

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: k8s-demo-deployment
  labels:
    app: k8s-demo
spec:
  replicas: 3
  selector:
    matchLabels:
      app: k8s-demo
  template:
    metadata:
      labels:
        app: k8s-demo
    spec:
      containers:
      - name: k8s-demo
        imagePullPolicy: Always
        image: niklaushirt/k8s-demo:1.0
        ports:
        - containerPort: 80
```

Minimum components you need:

- Deployment
- Service



Minimal Application – Kubernetes Manifests

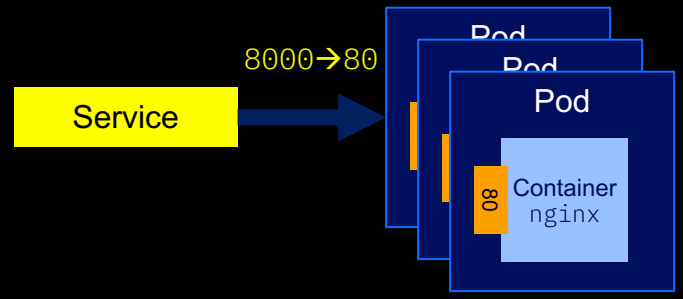



```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: k8s-demo-deployment
  labels:
    app: k8s-demo
spec:
  replicas: 3
  selector:
    matchLabels:
      app: k8s-demo
  template:
    metadata:
      labels:
        app: k8s-demo
    spec:
      containers:
      - name: k8s-demo
        imagePullPolicy: Always
        image: niklaushirt/k8s-demo:1.0
        ports:
        - containerPort: 80
```

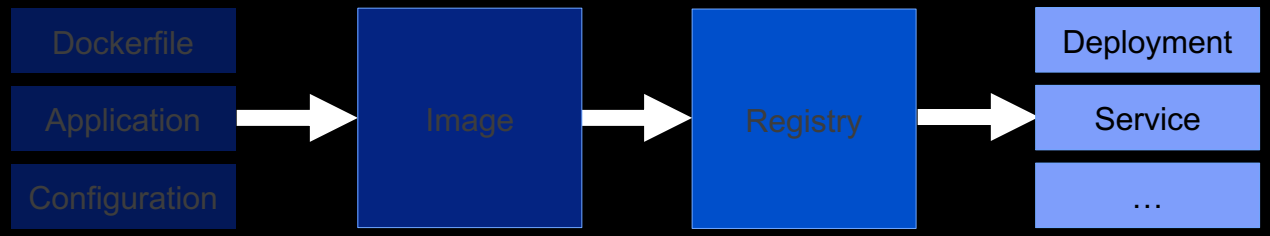
```yaml
apiVersion: v1
kind: Service
metadata:
  name: k8s-demo-service
spec:
  ports:
  - port: 8000
    targetPort: 80
    protocol: TCP
  selector:
    app: k8s-demo
```

Minimum components you need:

- Deployment
- Service



Minimal Application – Kubernetes Manifests

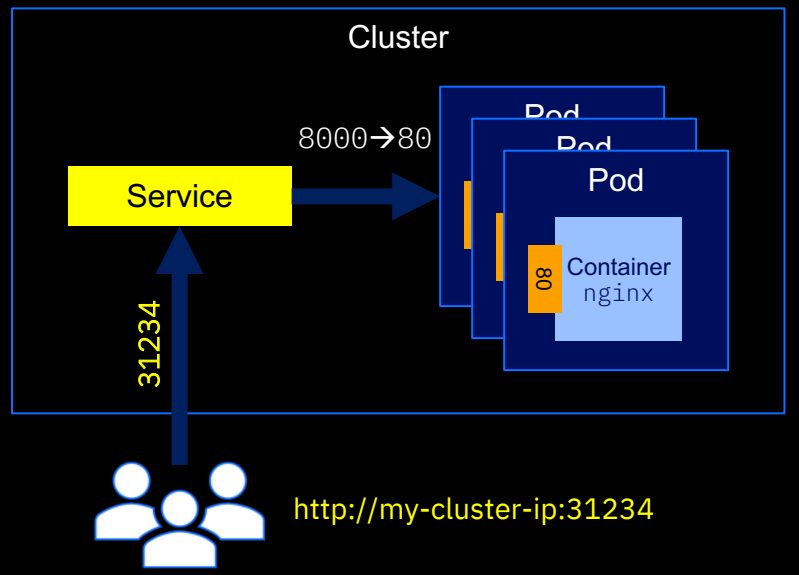



```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: k8s-demo-deployment
  labels:
    app: k8s-demo
spec:
  replicas: 3
  selector:
    matchLabels:
      app: k8s-demo
  template:
    metadata:
      labels:
        app: k8s-demo
    spec:
      containers:
      - name: k8s-demo
        imagePullPolicy: Always
        image: niklaushirt/k8s-demo:1.0
        ports:
        - containerPort: 80
```

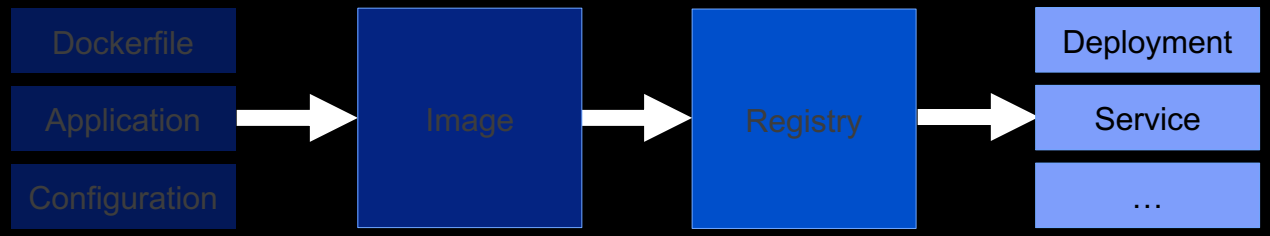
```yaml
apiVersion: v1
kind: Service
metadata:
  name: k8s-demo-service
spec:
  type: NodePort
  ports:
  - port: 8000
    targetPort: 80
    nodePort: 31234
    protocol: TCP
  selector:
    app: k8s-demo
```

Minimum components you need:

- Deployment
- Service



Minimal Application – Kubernetes Manifests

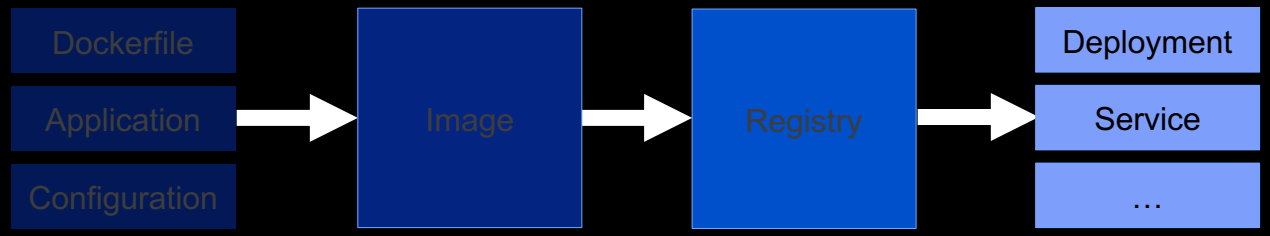



```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
…
spec:
…
template:
…
  resources:
    requests:
      memory: 60Mi
      cpu: 10m
    limits:
      memory: 500Mi
      cpu: 100m
```

Minimum components you need:

- Deployment
- Service

Minimal Application – Kubernetes Manifests

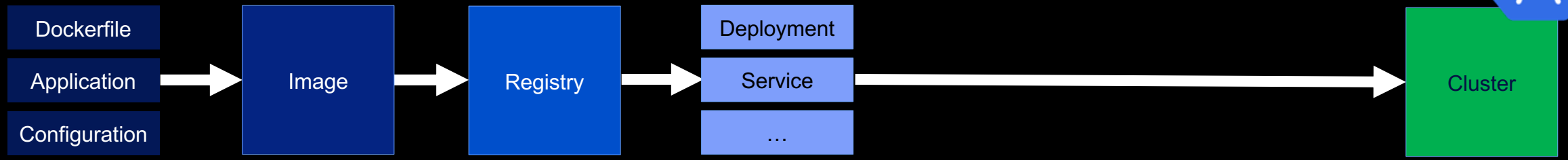



```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
…
spec:
…
template:
…
  env:
    - name: PORT
      value : "3000"
    - name: APPLICATION_NAME
      value: k8sdemo
    - name: BACKEND_URL
      value: http
```

Minimum components you need:

- Deployment
- Service

Minimal Application – Kubernetes Manifests




```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: k8s-demo-deployment
  labels:
    app: k8s-demo
spec:
  replicas: 3
  selector:
    matchLabels:
      app: k8s-demo
  template:
    metadata:
      labels:
        app: k8s-demo
    spec:
      containers:
      - name: k8s-demo
        image: niklaushirt/k8s-demo:1.0
        ports:
        - containerPort: 80
```

```yaml
apiVersion: v1
kind: Service
metadata:
  name: k8s-demo-service
spec:
  ports:
  - port: 8000
    targetPort: 80
    protocol: TCP
  selector:
    app: k8s-demo
```

Minimum components you need:

- Deployment
- Service

```
kubectl apply -f deployment.yaml
kubectl apply -f service.yaml
```

Minimal Application – Helm


```yaml
apiVersion: v1
name: k8s-demo
description: Demo App for the training.
keywords:
  - Training
maintainers:
  - name: Niklaus Hirt

version: 1.10.0
appVersion: 19.0.0.6

home: https://github.com/niklaushirt
icon: https://github.com/niklaushirt/logo-01.png
sources:
  - https://github.com/niklaushirt/charts/tree/master/stable
```

Minimum components we need:

Chart.yaml

Basic information about the chart

Templates (yaml)

Kubernetes deployment manifests (yaml) with placeholders for external parameters

values.yaml

A set of variables that will be passed on at deploy time to the deployment manifest templates

Minimal Application – Helm


```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: k8s-demo-deployment
  labels:
    app: k8s-demo
spec:
  replicas: {{ .Values.replicas }}
  selector:
    matchLabels:
      app: k8s-demo
  template:
    metadata:
      labels:
        app: k8s-demo
    spec:
      containers:
      - name: k8s-demo
        image: {{ .Values.image.name }}:{{ .Values.image.tag }}
        ports:
        - containerPort: 80
```

Minimum components we need:

Chart.yaml

Basic information about the chart

Templates (yaml)

Kubernetes deployment manifests (yaml) with placeholders for external parameters

values.yaml

A set of variables that will be passed on at deploy time to the deployment manifest templates

Minimal Application – Helm


```yaml
apiVersion: v1
image:
  name:
    - niklaushirt/k8s-demo
  tag: 1.0
  pullPolicy: IfNotPresent
replicas: 1
```

Minimum components we need:

Chart.yaml

Basic information about the chart

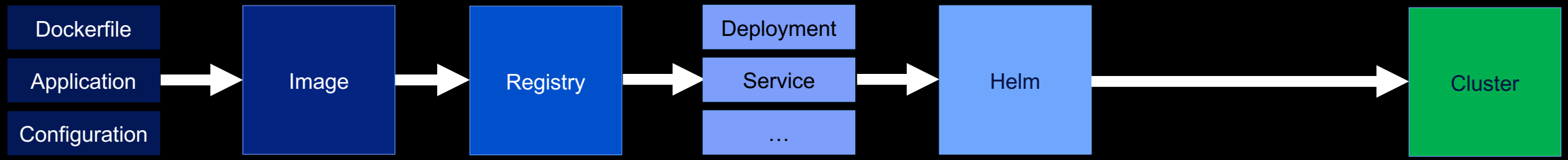
Templates (yaml)

Kubernetes deployment manifests (yaml) with placeholders for external parameters

values.yaml

A set of variables that will be passed on at deploy time to the deployment manifest templates

Minimal Application – Helm



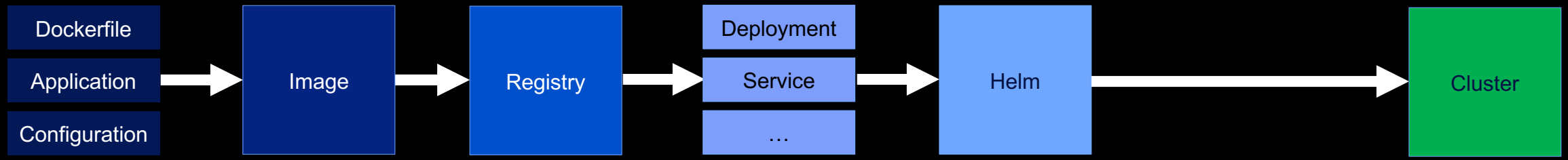

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: k8s-demo-deployment
  labels:
    app: k8s-demo
spec:
  replicas: {{ .Values.replicas }}
  selector:
    matchLabels:
      app: k8s-demo
  template:
    metadata:
      labels:
        app: k8s-demo
    spec:
      containers:
      - name: k8s-demo
        image: {{ .Values.image.name }}:{{ .Values.image.tag }}
        ports:
        - containerPort: 80
```

```yaml
apiVersion: v1
image:
  name:
    - niklaushirt/k8s-demo
  tag: 1.0
  pullPolicy: IfNotPresent
replicas: 1
```

```
helm install k8s-demo
```

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: k8s-demo-deployment
  labels:
    app: k8s-demo
spec:
  replicas: 1
  selector:
    matchLabels:
      app: k8s-demo
  template:
    metadata:
      labels:
        app: k8s-demo
    spec:
      containers:
      - name: k8s-demo
        image: niklaushirt/k8s-demo:1.0
        ports:
        - containerPort: 80
```

Minimal Application – Helm



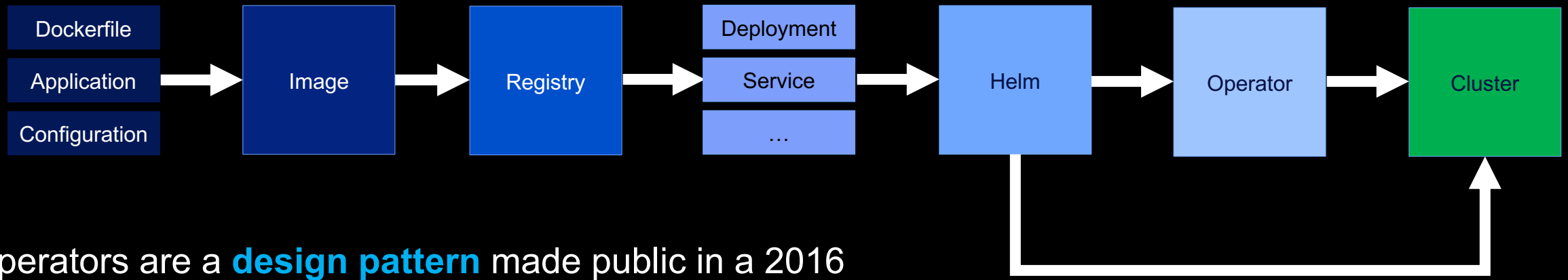

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: k8s-demo-deployment
  labels:
    app: k8s-demo
spec:
  replicas: {{ .Values.replicas }}
  selector:
    matchLabels:
      app: k8s-demo
  template:
    metadata:
      labels:
        app: k8s-demo
    spec:
      containers:
      - name: k8s-demo
        image: {{ .Values.image.name }}:{{ .Values.image.tag }}
        ports:
        - containerPort: 80
```

```yaml
apiVersion: v1
image:
  name:
    - niklaushirt/k8s-demo
  tag: 1.0
  pullPolicy: IfNotPresent
~~replicas: 1~~
```

```
helm install --set replicas=3 k8s-demo
```

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: k8s-demo-deployment
  labels:
    app: k8s-demo
spec:
  replicas: 3
  selector:
    matchLabels:
      app: k8s-demo
  template:
    metadata:
      labels:
        app: k8s-demo
    spec:
      containers:
      - name: k8s-demo
        image: niklaushirt/k8s-demo:1.0
        ports:
        - containerPort: 80
```

Minimal Application – Operators



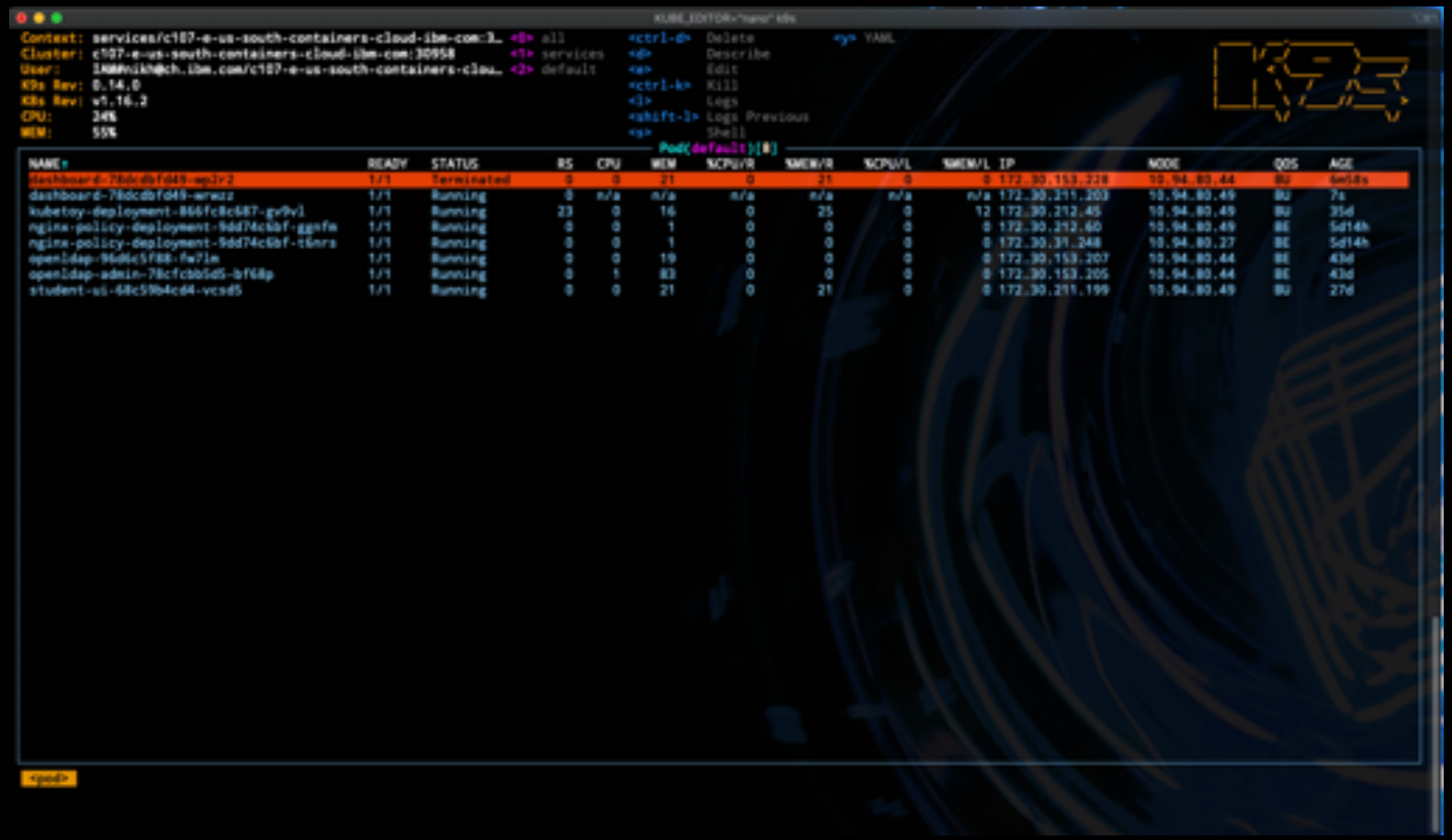
Operators are a **design pattern** made public in a 2016 CoreOS blog post.

The goal of an Operator is to put **operational knowledge into software**.

Operators implement and automate common
Day-1 (installation, configuration, etc) and
Day-2 (re-configuration, update, backup, failover, restore, etc.) **activities**
in a piece of software running inside your Kubernetes cluster, by integrating natively with Kubernetes concepts and APIs.

More about this in three weeks JTC14 – Kubernetes Operators

K9s – the Norton Commander of Kubernetes



Preinstalled in the training VM

For dumb people like me
(who just don't get vi)

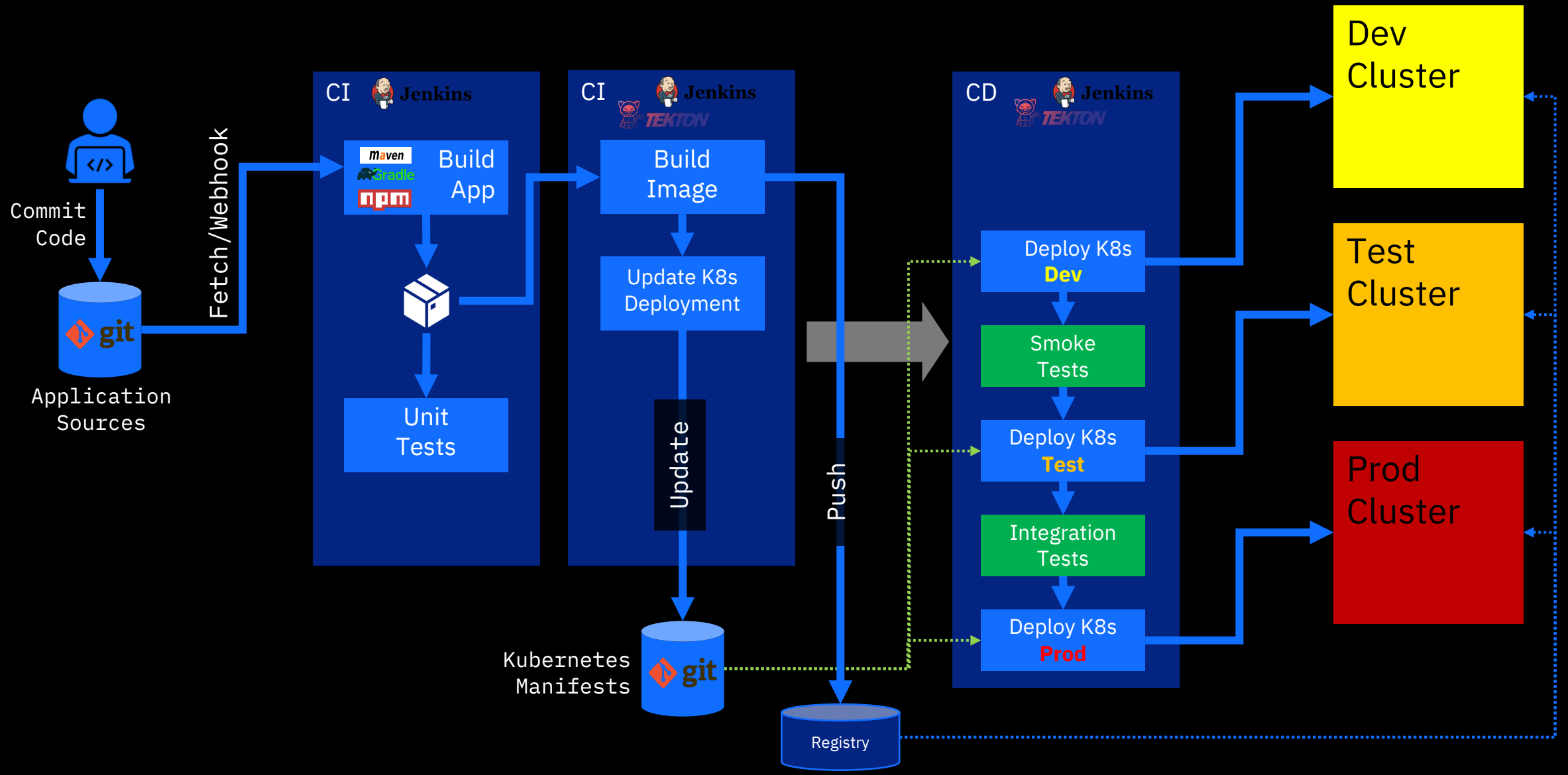
```
KUBE_EDITOR="nano" k9s
```

I did create an alias in .bashrc/.zshrc

```
alias k9s='KUBE_EDITOR="nano" k9s'
```

https://github.com/derailed/k9s

Kubernetes – CI/CD Pipeline - GitOps



QUESTIONS?

QUIZZ!!!

https://kahoot.it/

How do we secure this?

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

More on
this next week.....

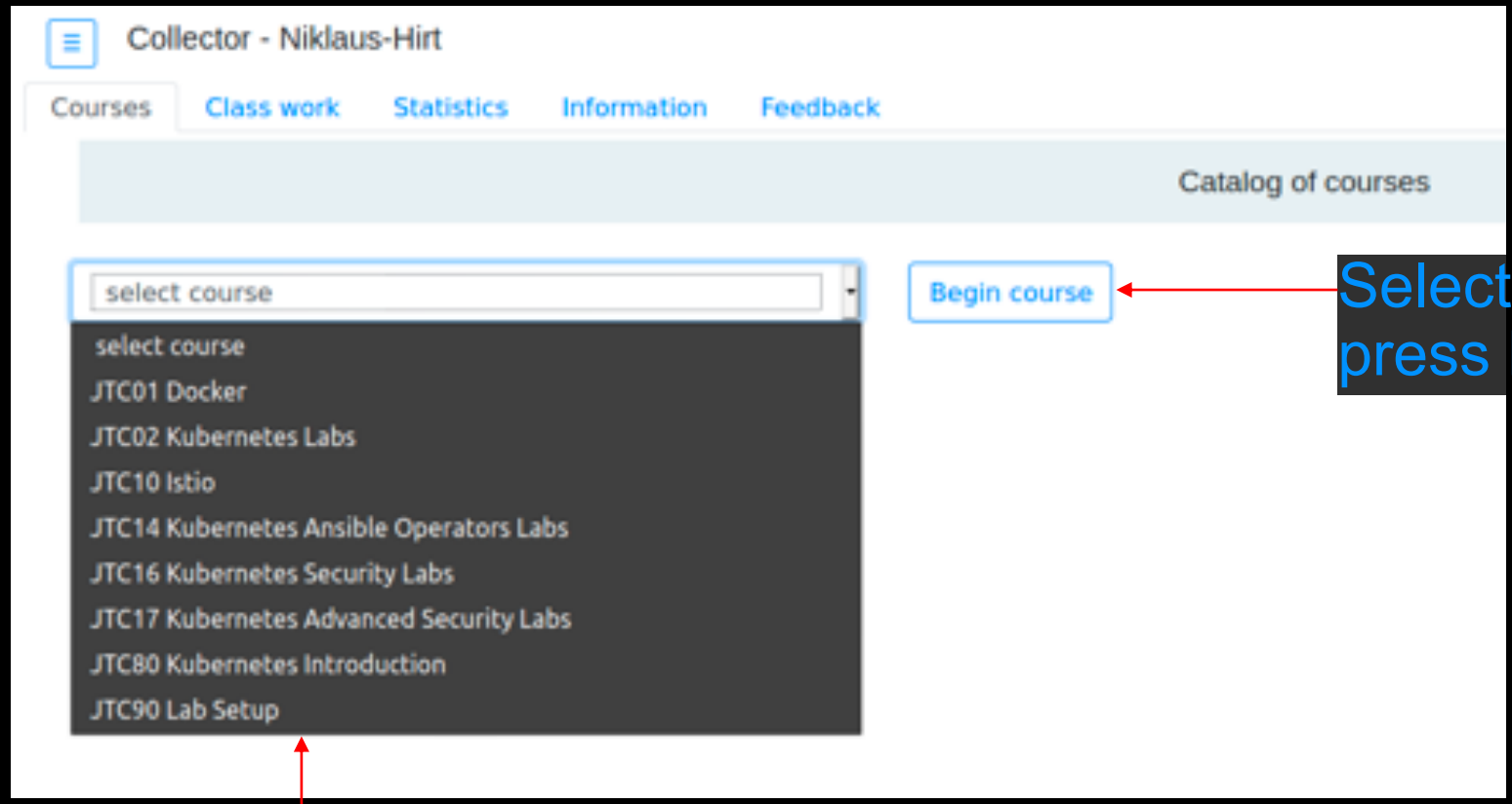
Kubernetes Workshop Series

Kubernetes - *Hands-On*

04

Starting Course JTC02 Kubernetes

Name will be shown



Select course and press button to begin

Current course catalog

JTC02 Labs

Lab 1: Refresher/overview over Kubernetes.

Lab 2: Running your first Pod on Kubernetes.

Lab 3: Deploy a Web Application

Lab 4: Scale an application on Kubernetes

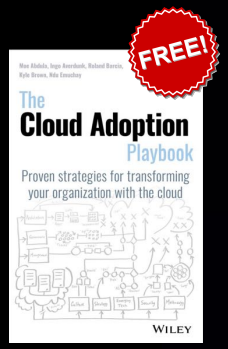
Lab 5: Basics of persisting data in Kubernetes

READY SET GO!!!!

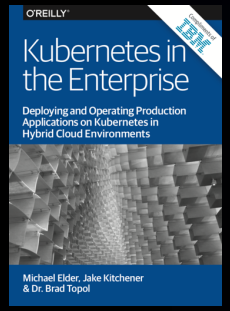
Duration: 60 mins

QUESTIONS?

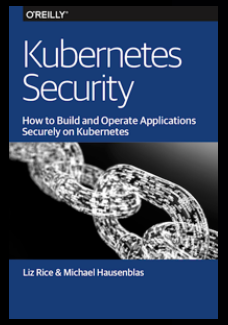
Kubernetes – Some Reading Tips



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



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



https://kubernetes-security.info/

Sources and documentation will be available here:

https://github.com/niklaushirt/k8s_training_public

https://github.com/niklaushirt/training

See you next week!

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

Kubernetes Workshop Series

Kubernetes Security Basics

Niklaus Hirt


nikh@ch.ibm.com


@nhirt

THANK YOU!!!!