Developing containerized applications on Kubernetes

Unit objectives

- Explain containers and the difference between containers and virtual machines (VMs).
- Describe container orchestration (Kubernetes).
- List the key capabilities of Kubernetes.
- Articulate the importance of using Kubernetes to prevent vendor lock-in.
- Describe the Kubernetes building blocks: Pod, Deployment, and Service.
- Scale and auto-scale your Deployment for high availability.

Containers

Topics

- Containers
- Container orchestration
- Introducing Kubernetes
- Kubernetes architecture
- Kubernetes objects
- Next steps

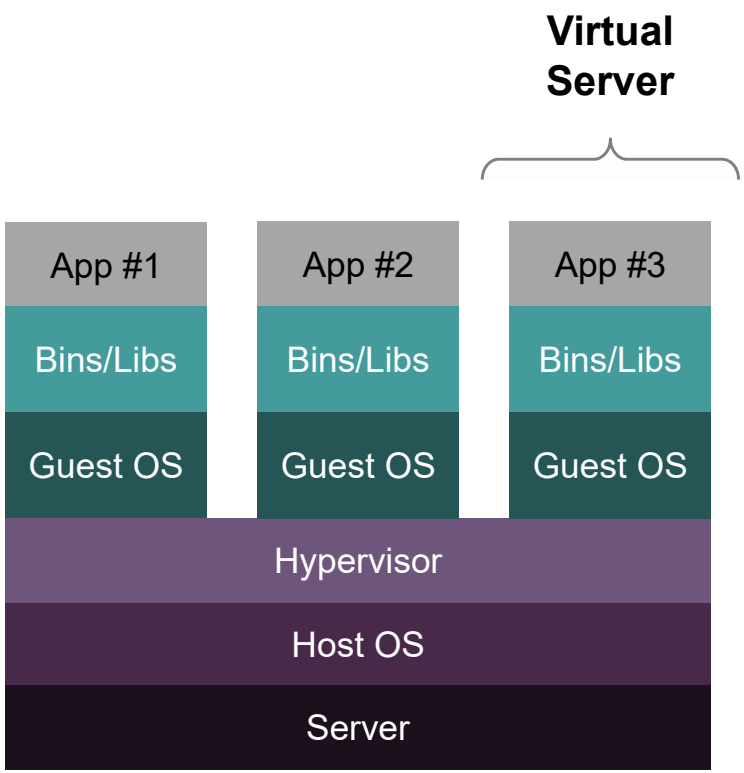
Containers

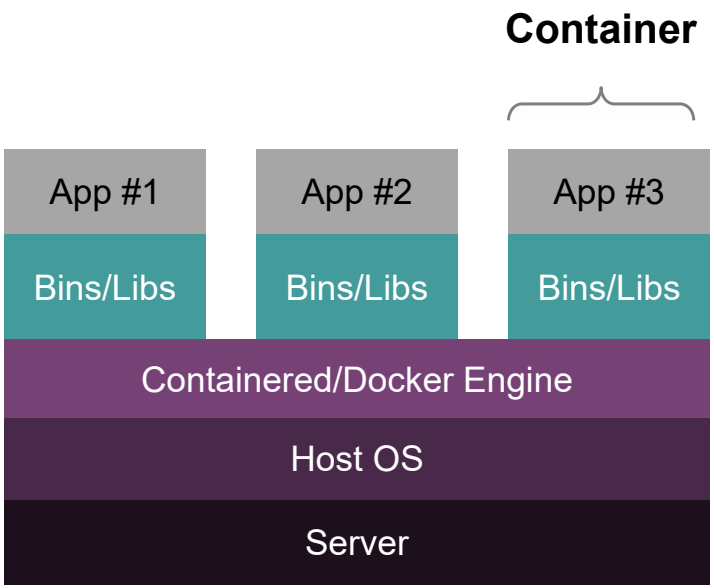
- Containers provide operating system (OS)-level virtualization.
- Containers are isolated from each other and package the application, code, tools, and libraries together.
- Containers run from a single OS kernel.

Linux + Container Engine

Containers versus virtual machines and the benefits of containers

- Containers are isolated, but share the kernel.
- Containers are isolated by hiding information (such as namespaces).
- Containers offer speed, agility, and portability.





Dockerfile: Building a container?

A Dockerfile is a text document that contains collections of commands and instructions that are automatically run in sequence in the Docker environment for building a new Docker image.

```
# Simple nginx web server image
FROM nginx:alpine

# Metadata
LABEL maintainer "Mihai Criveti"

# Serving static HTML - copy com
COPY html /usr/share/nginx/html
```

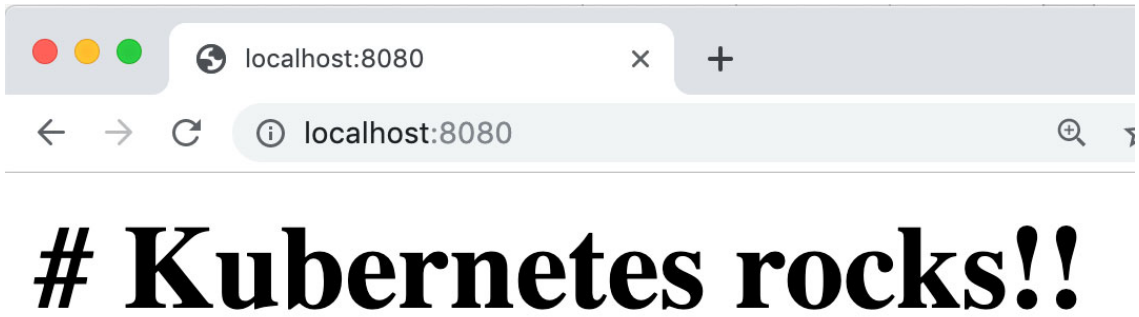
The **`docker build`** command builds an image from a Dockerfile.

Building a container image from a Dockerfile

The following code shows how to build a simple webserver image and test it locally:

```
$ vi Dockerfile
FROM nginx:alpine
COPY index.html /usr/share/nginx/html
$ vi index.html
<html> <body>
<h1> # Kubernetes rocks!! </h1>
</body> </html>
$ docker build -t cloud-course/helloweb:v1 .
$ docker run -p 8080:80 -d cloud-course/helloweb:v1
```

This graphic shows the result of the code.



Container orchestration

Topics

- Containers
- Container orchestration
- Introducing Kubernetes
- Kubernetes architecture
- Kubernetes objects
- Next steps

Container orchestration

You can manage the deployment, placement, and lifecycle of containers at scale.

Container orchestration performs the following functions:

- Cluster management
- Self-healing
- Replication
- Service discovery
- Scheduling
- Scaling and workload auto-scaling
- Persistent storage
- Blue-green deployments
- Same API
- Managing stateful and stateless applications.

Common container orchestration platforms include *Kubernetes*, *Docker Swarm*, and *Apache Mesos*. In this course, the focus is on Kubernetes.

Introducing Kubernetes

Topics

- Containers
- Container orchestration
- Introducing Kubernetes
- Kubernetes architecture
- Kubernetes objects
- Next steps

What is Kubernetes

- According to kubernetes.io, Kubernetes has the following features:
  - It is a portable and extensible open source platform for managing and scaling containerized workloads and services.
  - It facilitates both declarative configurations and automation.
  - It has a large and rapidly growing infrastructure.
  - Kubernetes services, support, and tools are widely available.
- Kubernetes is an open source project that is hosted by the *Cloud Native Computing Foundation* (CNCF).
- Kubernetes prevents vendor lock-in because it is offered by major cloud providers, such as IBM Cloud Kubernetes Service, IBM Cloud Private, Red Hat OpenShift, Azure Kubernetes Service, and Google Kubernetes Engine.

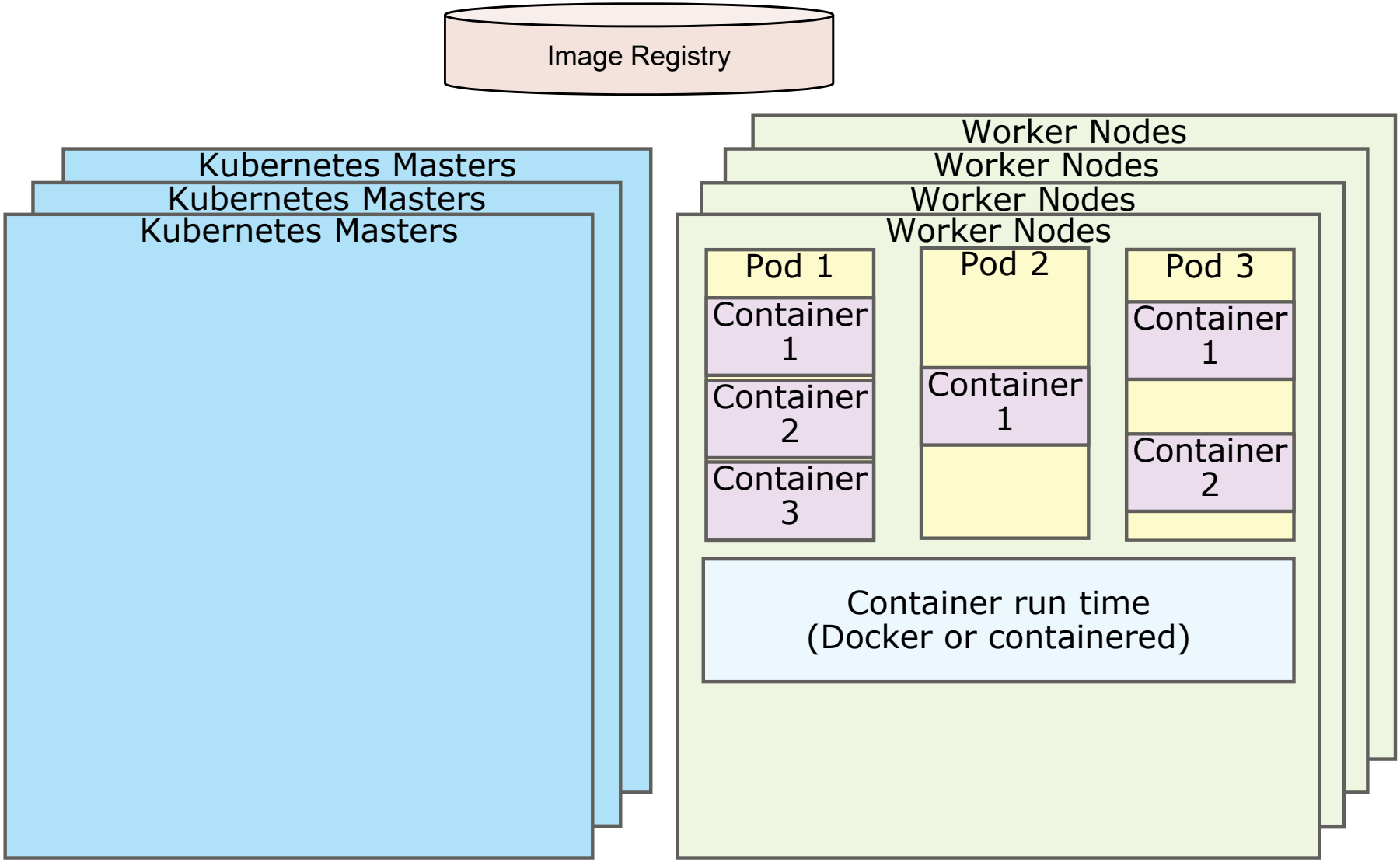
Kubernetes architecture

Topics

- Containers
- Container orchestration
- Introducing Kubernetes
- Kubernetes architecture
- Kubernetes objects
- Next steps

Kubernetes architecture

Multiple master and worker nodes provide scaling and high availability.

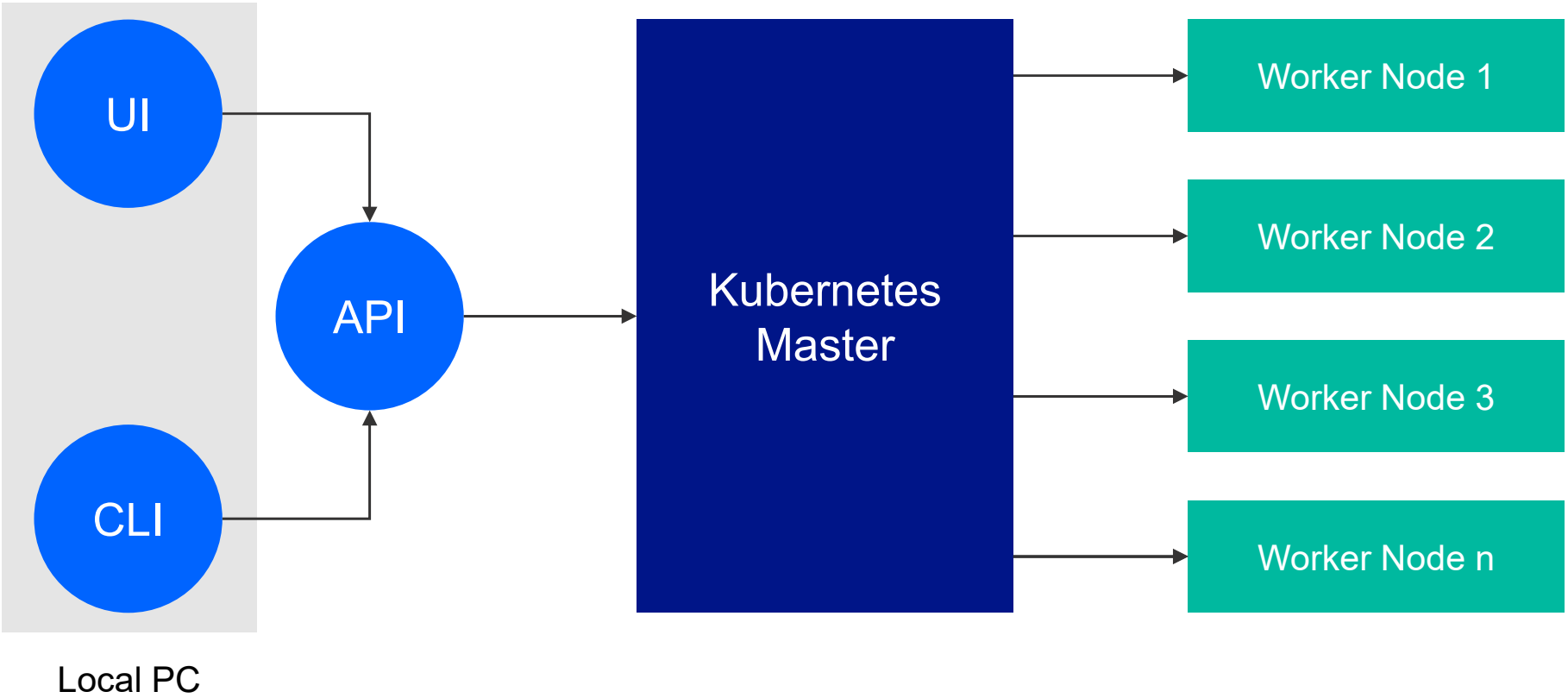


Kubernetes objects

Topics

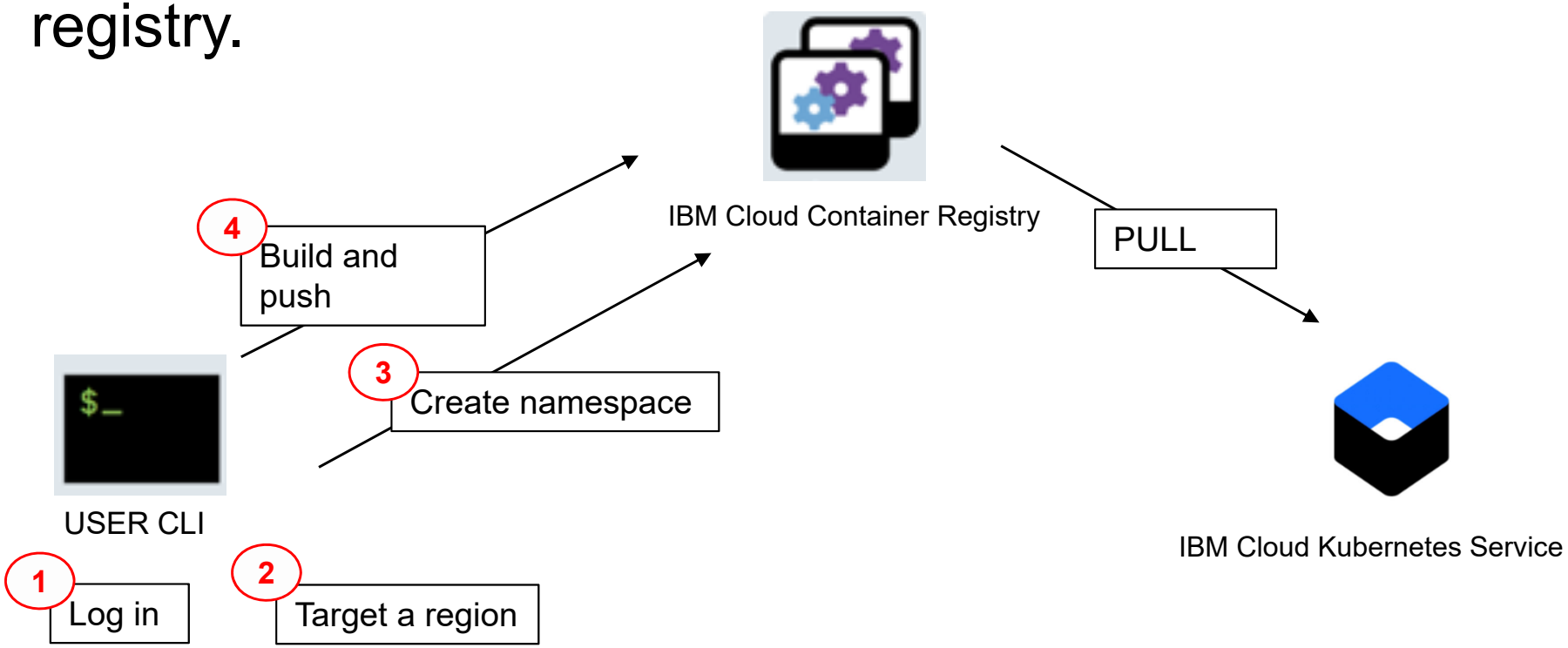
- Containers
- Container orchestration
- Introducing Kubernetes
- Kubernetes architecture
- Kubernetes objects
- Next steps

Interaction with a Kubernetes cluster

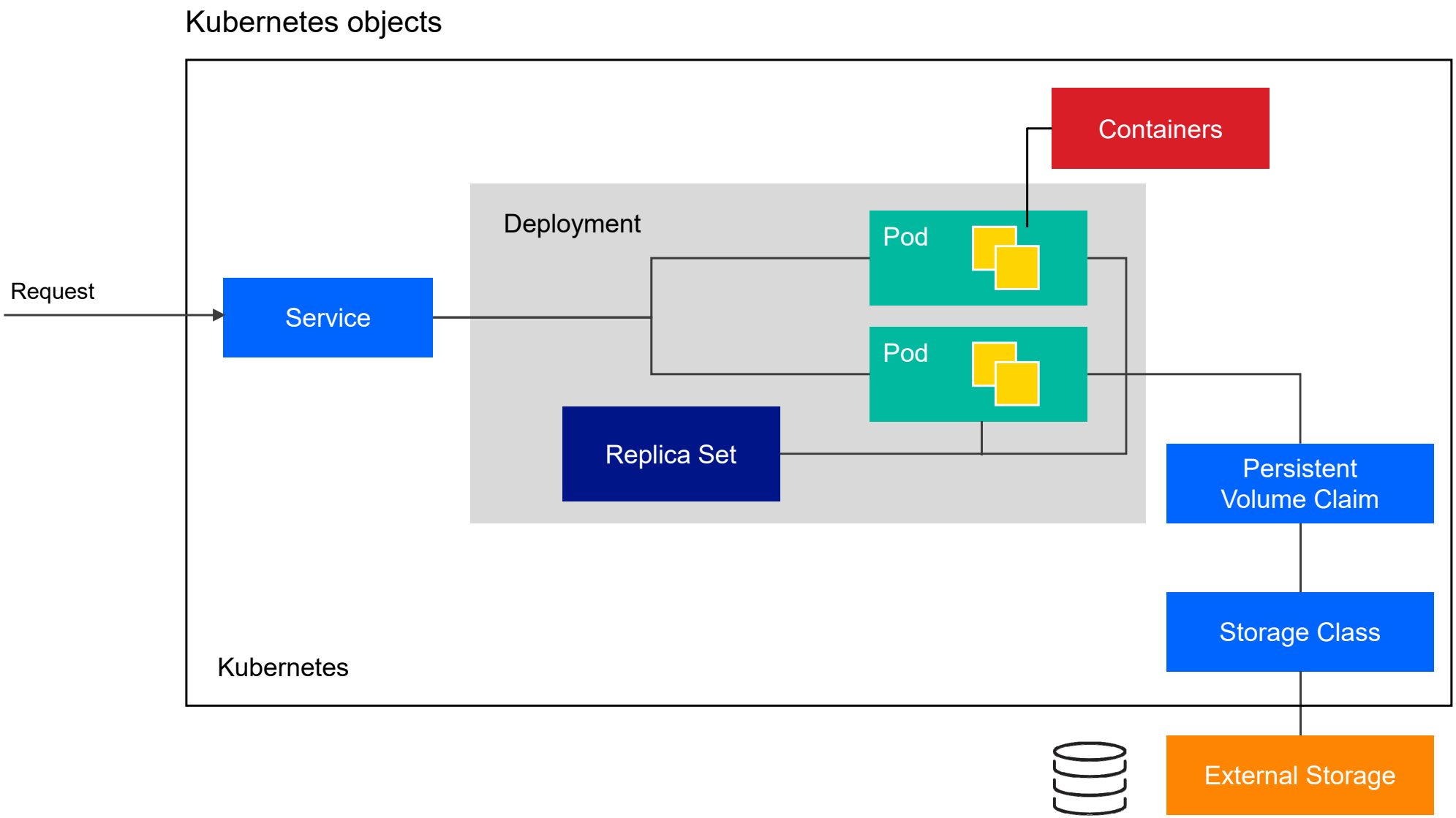


Building a container and storing it in the container registry

- A container registry stores and distributes container images.
- Examples for Container Registry are IBM Cloud Container Registry, and Docker Hub.
- The flow shows the steps of building an image directly into the IBM Cloud Container Registry as an example container registry.

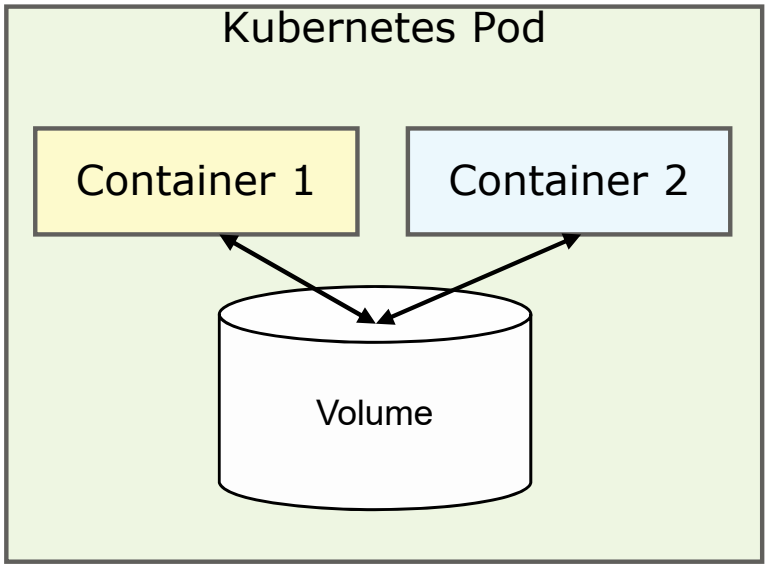


Understanding Kubernetes objects



Pods

- A Pod is the smallest unit of a Deployment that can be managed by Kubernetes.
- It consists of a group of one or more containers with a shared network, storage, and a specification for how to run containers.
- It contains one or more application containers that are tightly coupled.



Deployment

The difference between kind: Pod, and Deployment is that Deployment maintains the replicas, so when you delete a Pod, it is rescheduled automatically by the master so that the current replicas match the wanted one.

Here is an example of a Deployment manifest file in YAML format:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-student-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: hello-student
  template:
    metadata:
      labels:
        app: hello-student
    spec:
      containers:
      - name: hello-student
        image: us.icr.io/cr-student/hello-student:v1
        ports:
        - containerPort: 3000
```

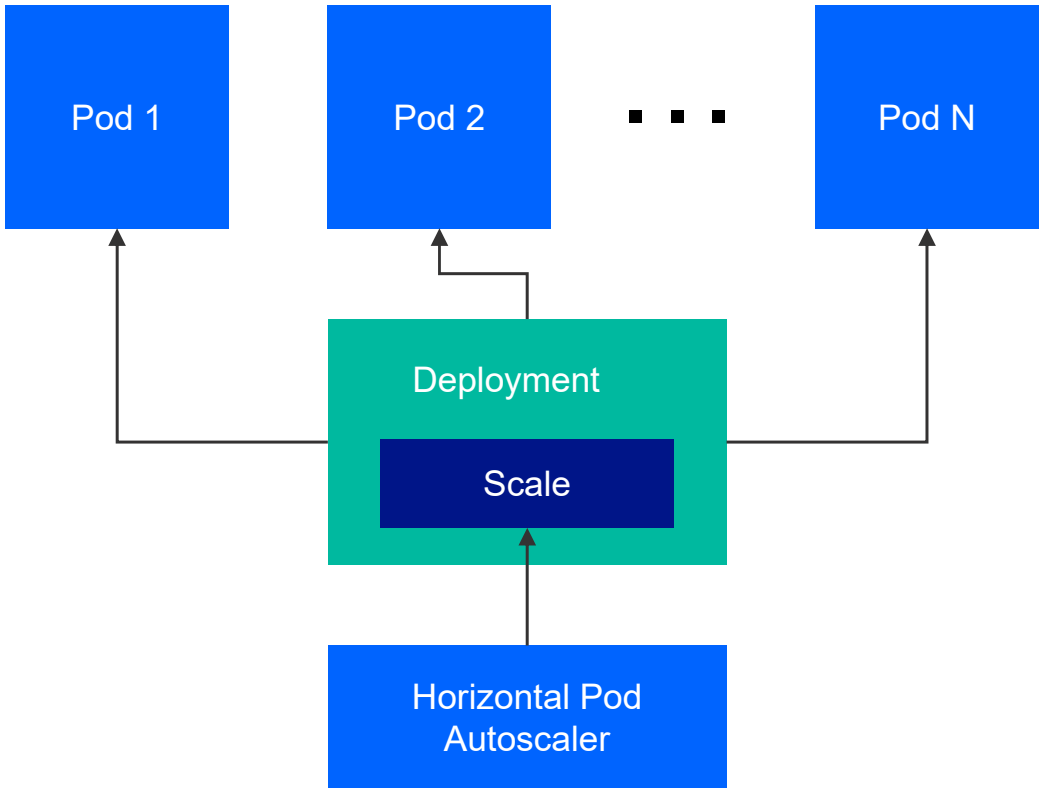
Scaling the Deployment

- Scale in and out the Deployment by changing the number of replicas by running the following command:

```
kubectl scale deployment <deployment name> --replicas=4
```

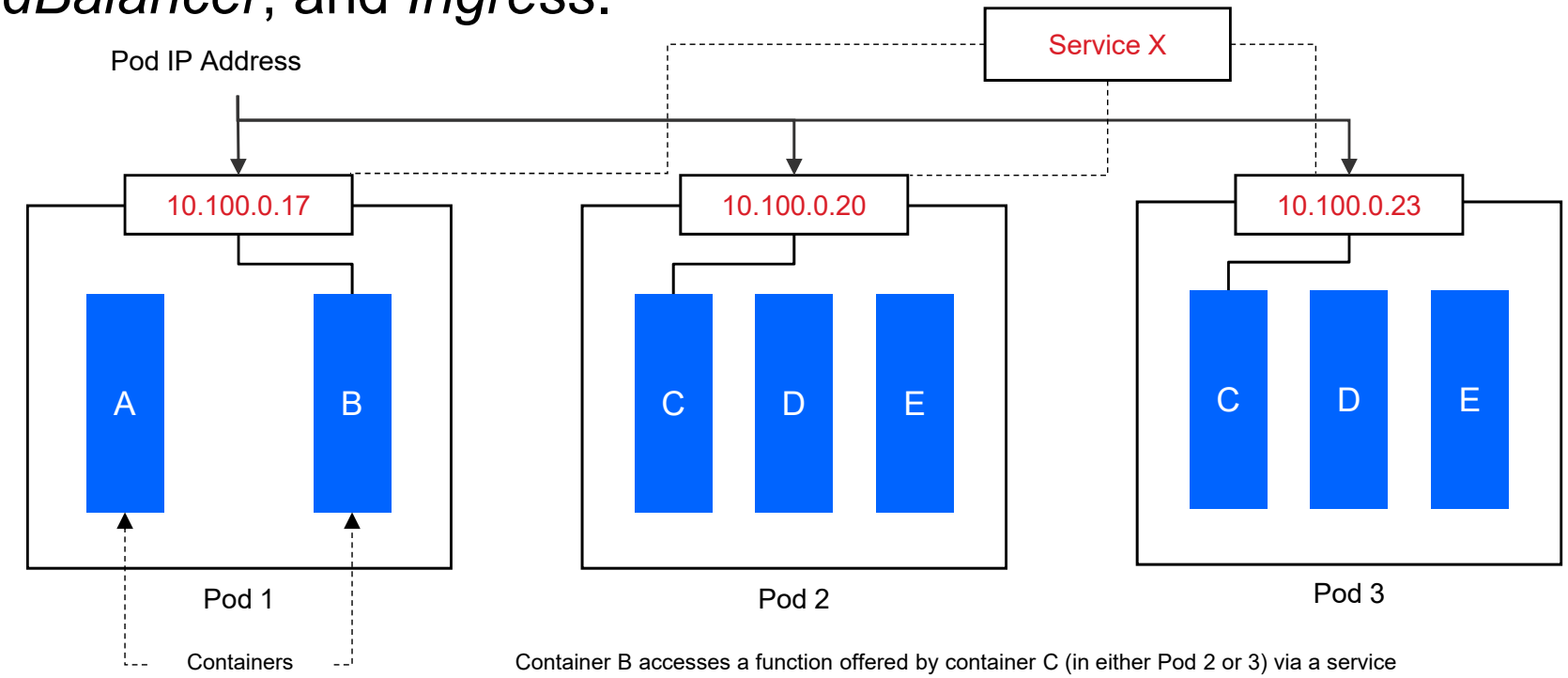
- For example, you can use Horizontal Pod Autoscaler to increase or decrease automatically the number of instances of your apps based on CPU by running the following command:

```
kubectl autoscale deployment <deployment name> --cpu-percent=80 --
min=4 --max=10
```



Services

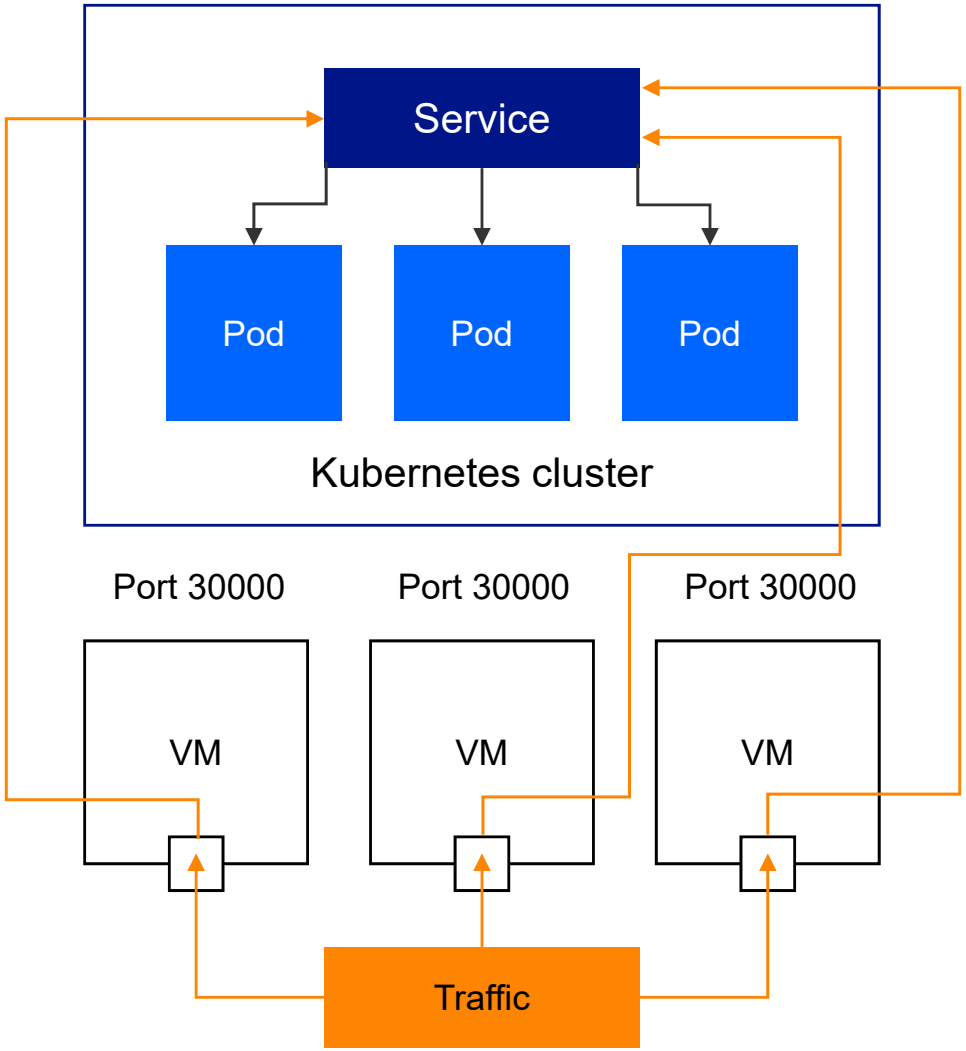
- *Services* describe a logical set of Pods and a policy to access them.
- The set of Pods that is targeted by a Service is usually determined by a *Label Selector*.
- The Service propagates state and networking information to all the worker nodes.
- Various Service exposure types exist, such as *ClusterIP*, *NodePort*, *LoadBalancer*, and *Ingress*.



Services example: NodePort

- NodePort exposes a Service on all the worker nodes on a specific port.

```
apiVersion: v1
kind: Service
metadata:
  name: hello-student-service
spec:
  selector:
    app: hello-student
  ports:
  - name: http
    protocol: TCP
    port: 80
    nodePort: 30000
  type: NodePort
```



Storage: Persistence Volumes

- By default, the file systems in Kubernetes offer ephemeral storage. As such, data does not survive a container restart.
- Kubernetes volumes provide persistent storage.
- This storage can also be used as a shared disk space across containers within a Pod.

Example types of volumes:

- CephFS
- GlusterFS
- NFS
- vSphere Volumes

Next steps

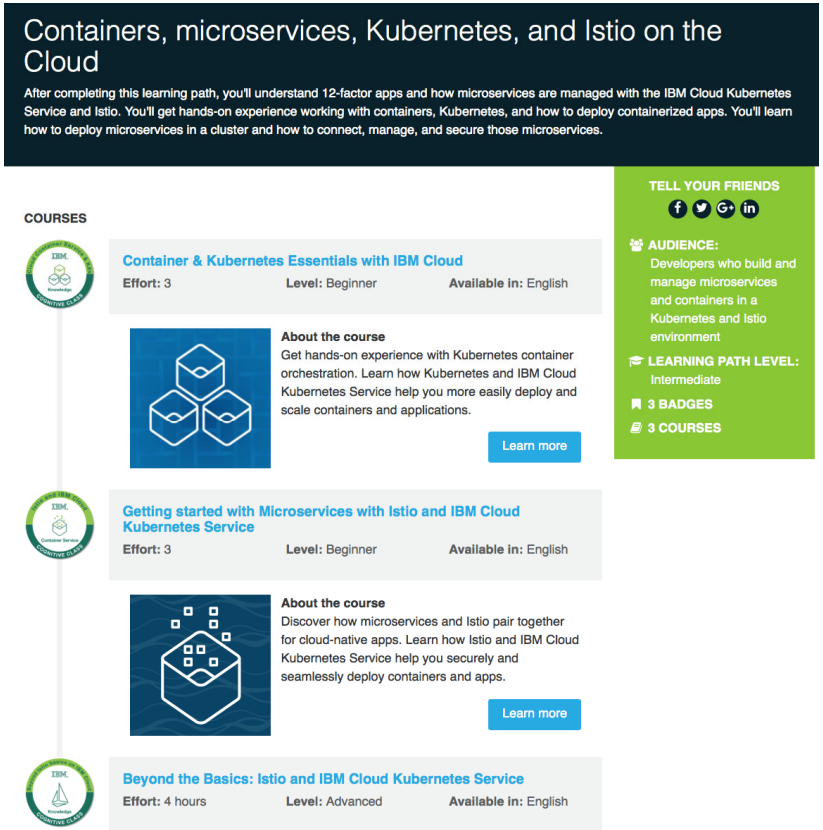
Topics

- Containers
- Container orchestration
- Introducing Kubernetes
- Kubernetes architecture
- Kubernetes objects
- Next steps

Further reading

For more information, see the following resources:

- https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/
- https://www.ibm.com/cloud/garage/content/course/kubernetes-101
- https://cognitiveclass.ai (for free badges and training)



Unit summary

- Explain containers and the difference between containers and virtual machines (VMs).
- Describe container orchestration (Kubernetes).
- List the key capabilities of Kubernetes.
- Articulate the importance of using Kubernetes to prevent vendor lock-in.
- Describe the Kubernetes building blocks: Pod, Deployment, and Service.
- Scale and auto-scale your Deployment for high availability.